

AP

Anime Watch Planner

Le guide complet du projet : comment il fonctionne, pourquoi il est construit ainsi, et tout son code source.

PROJET PERSONNEL RÉALISÉ AVEC CLAUDE OPUS (IA). CE N'EST PAS UN PROJET OFFICIEL, JUSTE UN HOBBY. MES VRAIS PROJETS, RÉALISÉS SANS IA, SERONT PARTAGÉS À PART.

Application en ligne anime-planner.vercel.app

Version du **8 octobre 2026**

Mitchel Prince

Sommaire

1. Ce que fait l'application

- À propos de ce guide
- Le projet en chiffres
- Ce qu'on peut faire avec
- Ce qui le rend différent

2. Les bases, expliquées simplement

- Ce qu'est un site web
- Frontend et backend
- React : construire une page avec des briques
- TypeScript : JavaScript avec des garde-fous
- Les API, GraphQL et JSON
- Les outils autour du code
- Git, GitHub et Vercel

3. Utiliser l'application, étape par étape

- Étape 1 : trouver une série
- Étape 2 : la page de la série
- Étape 3 : régler son plan
- Étape 4 : choisir quand regarder
- Étape 5 : lire son planning
- Étape 6 : enregistrer et suivre sa progression
- Étape 7 : exporter et partager
- Les autres panneaux
- Sur un téléphone

4. L'organisation du projet

- L'arborescence
- À quoi sert chaque dossier
- La règle d'or : garder la logique pure
- Les dépendances

5. Comment tout s'assemble

- Vue d'ensemble
- Les données descendent, les événements remontent
- L'état que garde App
- Suivre une action : vous cliquez sur une affiche
- Une autre action : vous changez votre temps quotidien
- Ouvrir un lien directement

6. Le moteur de planning

- Étape 1 : combien dure un épisode ?
(getNetEpisodeRuntime)
- Étape 2 : quels épisodes ? (getWatchableEpisodes)
- Étape 3 : répartir en journées (generateSchedule)
- Le code, ligne par ligne
- Pourquoi « glouton » et pas optimal
- « Par épisodes » n'est pas un second moteur

7. Les dates, le calendrier et l'export .ics

- Deux couches séparées
- Des numéros de jour aux dates (scheduleDates)
- Pourquoi les dates sont du texte, calculées en UTC
- La grille du mois (monthGrid)
- Les séries encore en diffusion
- Le fichier .ics
- Sur iPhone et iPad

8. Enregistrer ses plannings et sa progression

- Où sont stockés les plans
- Un plan par série
- La progression se compte par épisode, pas par jour
- Lire avec prudence
- Mettre à niveau les anciennes sauvegardes
- Les chiffres de progression (progress.ts)

9. Parler à AniList

- Ce que fournit AniList
- Les quatre questions que pose l'application
- Des listes légères, des détails complets
- Envoyer une question (request)
- Mettre de l'ordre dans les réponses
- Le piège des titres d'épisodes
- Le piège des descriptions
- « Popular right now », pas « cette saison »

10. L'interface : thème, composants, animations et accessibilité

- L'arbre des composants
- Deux couleurs de marque, et celles de chaque série
- La mise en page et Tailwind
- Les animations
- L'accessibilité
- Quand quelque chose tourne mal

11. Les adresses, les moteurs de recherche et les aperçus de liens

- Chaque série a sa propre adresse
- Changer de page sans charger une nouvelle page
- Le problème des liens directs, et sa solution
- Ce que voient les moteurs de recherche et les aperçus de liens
- Le prérendu : des pages prêtes à l'avance
- Les anciens liens de partage

12. La sécurité

- Qu'est-ce qui pourrait mal tourner ?
- Les règles

Les en-têtes de sécurité

Les dépendances

13. Les tests

Ce qu'est un test

Ce qui est testé

Comment les tests sont organisés

Les lancer

Ce qui n'est pas testé automatiquement

14. Du code au site : construction et mise en ligne

Construire le site (npm run build)

La mise en ligne avec Vercel

Comment ce guide est fabriqué

15. Limites connues et choix assumés

Volontairement pas fait

Limites connues

Ce qui a changé dans cette version (octobre 2026)

16. Installer et lancer le projet soi-même

Ce qu'il faut

Récupérer le code et le lancer

Les autres commandes

Publier sa propre copie

Par où commencer à lire le code

Des idées pour aller plus loin

Annexe A. Le code source complet

A.1 Configuration du projet

package.json
tsconfig.json
tsconfig.app.json
tsconfig.node.json
vite.config.ts
vercel.json
.gitignore

A.2 La page et les fichiers publics

index.html
public/favicon.svg
public/robots.txt
public/_headers
public/_redirects

A.3 Les types partagés

src/types.ts

A.4 La logique pure (src/lib)

src/lib/schedule.ts
src/lib/format.ts
src/lib/theme.ts

src/lib/calendar.ts
src/lib/progress.ts
src/lib/storage.ts
src/lib/ics.ts
src/lib/route.ts
src/lib/seo.ts
src/lib/share.ts
src/lib/platform.ts
src/lib/download.ts

A.5 Parler à AniList

src/api/anilist.ts

A.6 Les animations

src/motion/presets.ts
src/motion/hooks.ts

A.7 Les composants de l'interface

src/components/icons.tsx
src/components/SectionHeading.tsx
src/components/Motion.tsx
src/components/SearchBar.tsx
src/components/AnimeCard.tsx
src/components/AnimeListRow.tsx
src/components/SavedSchedules.tsx
src/components/SearchStep.tsx
src/components/ConfigForm.tsx
src/components/WatchPrefsForm.tsx
src/components/StreamingIcon.tsx
src/components/CalendarView.tsx
src/components/ScheduleView.tsx
src/components/ConfigureStep.tsx
src/components/ErrorBoundary.tsx

A.8 L'application

src/index.css
src/App.tsx
src/main.tsx

A.9 Le script de build

scripts/prerender.ts

A.10 Les tests

src/lib/schedule.test.ts
src/lib/format.test.ts
src/lib/theme.test.ts
src/lib/calendar.test.ts
src/lib/progress.test.ts
src/lib/storage.test.ts
src/lib/ics.test.ts
src/lib/route.test.ts
src/lib/seo.test.ts
src/lib/share.test.ts
src/lib/platform.test.ts

A.11 La documentation

README.md
CLAUDE.md

Annexe B. Glossaire

1. Ce que fait l'application

Anime Watch Planner est un site web qui répond à une question simple :

« Si je regarde cet anime tant de minutes par jour, quand vais-je le finir, et qu'est-ce que je regarde chaque jour ? »

Vous choisissez une série, vous indiquez le temps dont vous disposez, et le site vous donne un plan jour par jour : quels épisodes regarder à quelle date, avec leurs vrais titres, la durée de chaque journée, et le jour où vous aurez fini. Vous pouvez enregistrer ce plan, cocher ce que vous avez regardé, et ajouter tout le plan à l'agenda de votre téléphone ou de votre ordinateur.

Le site est en ligne à l'adresse **anime-planner.vercel.app**. Il est gratuit, ne demande aucun compte, et tout ce que vous enregistrez reste dans votre propre navigateur. L'interface de l'application est en anglais ; ce guide cite donc les boutons avec leur nom anglais.

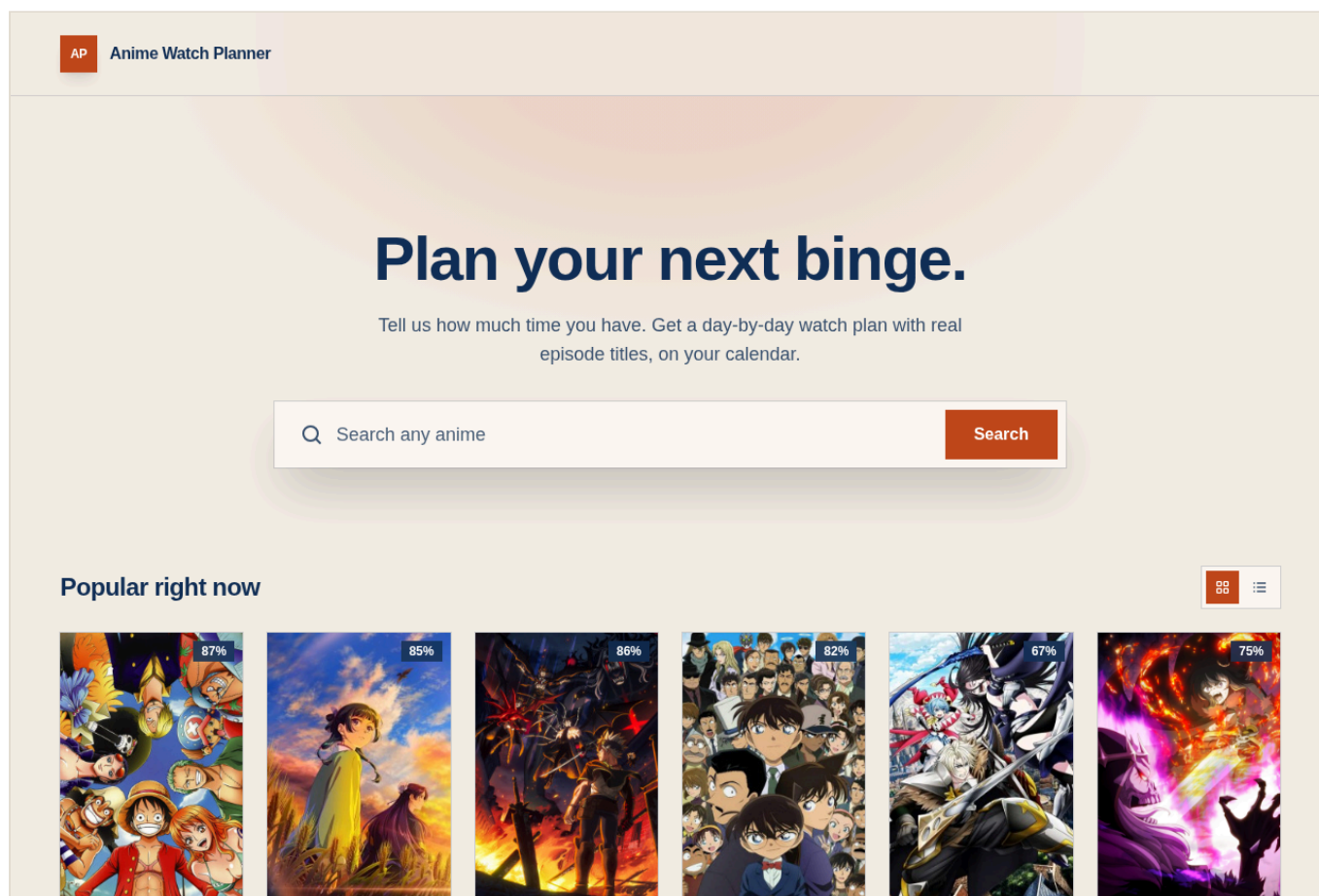


Figure 1. La page d'accueil : un champ de recherche et les séries populaires du moment.

À propos de ce guide

Ce guide explique tout le projet, de ce que l'on voit à l'écran jusqu'à chaque ligne de code. Il est écrit pour que **tout le monde puisse le suivre**, y compris les personnes qui n'ont jamais programmé.

- **Les chapitres 1 à 5** ne demandent aucune connaissance en programmation. Ils expliquent ce que fait l'application, les idées de base derrière n'importe quel site web, comment utiliser l'application et comment le projet est organisé.

- **Les chapitres 6 à 14** entrent dans la machine : les calculs, les dates, la sauvegarde, la connexion à la base de données d'anime, l'interface, les moteurs de recherche, la sécurité, les tests et la mise en ligne. Chaque mot technique est expliqué la première fois qu'il apparaît, et le **glossaire** (annexe B) les rassemble tous.
- **Les chapitres 15 et 16** présentent les limites et les choix assumés du projet, puis comment l'installer et le lancer soi-même.
- **L'annexe A** contient **le code source complet** : chaque fichier du projet, présenté par un court paragraphe et imprimé avec ses numéros de ligne.

Important : il s'agit d'un projet personnel, réalisé comme loisir avec l'aide de **Claude Opus**, une intelligence artificielle (IA) d'Anthropic, utilisée via l'outil Claude Code. Ce n'est pas un projet officiel ni commercial. Mes vrais projets, réalisés sans IA, seront partagés à part.

Le projet en chiffres

Adresse	anime-planner.vercel.app
Source des données	AniList, une base de données d'anime publique et gratuite
Où sont vos données	Uniquement dans votre navigateur (rien n'est envoyé à un serveur à nous : il n'y en a pas)
Prix et compte	Gratuit, sans compte, sans connexion
Code	Environ 6 800 lignes réparties dans 60 fichiers, écrites en TypeScript
Tests automatiques	131 tests qui vérifient les calculs, les dates, la sauvegarde, les règles de sécurité et la lisibilité des couleurs
Hébergement	Vercel, mis à jour automatiquement depuis GitHub

Ce qu'on peut faire avec

- **Trouver n'importe quel anime**, en le cherchant ou en choisissant parmi les séries populaires du moment.
- **Régler son temps quotidien**, en minutes (« 90 minutes par jour ») ou en épisodes (« 3 épisodes par jour »).
- **Sauter les génériques** : l'application peut retirer environ 3 minutes par épisode pour les chansons de début et de fin, si vous les passez.
- **Choisir quand regarder** : tous les jours, en semaine, le week-end, ou les jours de votre choix ; à une heure précise ou « n'importe quand » ; avec un rappel si vous voulez ; dans votre fuseau horaire.
- **Lire son plan** sous forme de liste, jour par jour, avec les vrais titres des épisodes, ou sous forme de calendrier mensuel.
- **L'enregistrer et suivre sa progression** : cochez chaque jour ; la page d'accueil affiche alors une rangée « Continue watching » avec votre avancement et votre série de jours consécutifs.
- **L'ajouter à son agenda** (Google Agenda, Calendrier d'Apple, Outlook...) avec un seul fichier.
- **Partager un lien** vers la page de la série.
- **Voir où regarder** la série légalement (plateformes de streaming officielles), **l'ordre de visionnage** d'une franchise (saison 1, saison 2, le film...) et **la bande-annonce**.

AP

Anime Watch Planner

Q

Search anime

< Back



Frieren: Beyond Journey's End

Sousou no Frieren

TV · 2023 · MADHOUSE · 28 ep · 24 min each

Adventure

Drama

Fantasy

The adventure is over but life goes on for an elf mage just beginning to learn what living is all about. Elf mage Frieren and her courageous fellow adventurers have defeated the Demon King and brought peace to the land. But Frieren will long outlive the rest of her former party. How will she come to understand what lif...

Read more

Save

▶ Watch trailer

Your plan

By time

By episodes

Daily watch time

90

min

You'll finish on

Saturday, October 17

Watching every day · 10 days from start to finish

Start date

10/08/2026

📅

Figure 2. La page d'une série. Toute la page prend les couleurs de l'affiche (ici, le vert de Frieren).

Ce qui le rend différent

Il est construit autour de votre temps, pas autour d'une liste. Beaucoup de sites d'anime aident à tenir une liste de ce qu'on veut regarder. Celui-ci transforme une série en vrai plan, avec des dates, selon le temps dont vous disposez réellement.

Il n'affiche que des informations réelles. Les titres des épisodes viennent de la base AniList. L'application n'invente jamais de résumé de ce qui se passe dans un épisode : aucune source fiable ne possède de tels résumés pour la plupart des séries, et un résumé inventé pourrait affirmer des choses fausses sur une vraie série comme si c'étaient des faits. L'application montre donc les vrais titres, ou rien.

Il respecte votre vie privée. Pas de compte, pas de serveur à nous qui stockerait quoi que ce soit. Vos plans enregistrés vivent dans votre navigateur. Le seul service extérieur que l'application contacte est AniList, pour obtenir les informations sur les séries.

2. Les bases, expliquées simplement

Ce chapitre explique les idées nécessaires pour comprendre la suite du guide. Si vous créez déjà des sites web, vous pouvez le passer.

Ce qu'est un site web

Quand vous ouvrez un site, votre navigateur (Chrome, Firefox, Safari...) télécharge quelques fichiers depuis un ordinateur quelque part sur Internet, appelé **serveur**, et les transforme en la page que vous voyez. Trois langages travaillent ensemble :

- **HTML** décrit la *structure* : « ici un titre, ici un bouton, ici une image ». C'est le squelette.
- **CSS** décrit l'*apparence* : couleurs, tailles, espacements, la mise en page sur téléphone ou sur ordinateur. Ce sont les vêtements.
- **JavaScript** décrit le *comportement* : ce qui se passe quand on clique, tape ou fait défiler. Ce sont les muscles.

Frontend et backend

Beaucoup de sites ont deux moitiés. Le **frontend** est ce qui fonctionne dans votre navigateur. Le **backend** est un programme qui tourne sur un serveur : il stocke des données dans une base, gère les comptes, etc.

Anime Watch Planner n'a qu'un frontend. C'est un ensemble de fichiers que n'importe quel hébergeur simple peut servir. Tout le travail (recherche, calculs, sauvegarde) se fait dans votre navigateur. Les informations sur les anime viennent d'**AniList**, un service extérieur qui ouvre sa base de données à tous. Le site est donc peu coûteux à héberger, rapide et respectueux de la vie privée : il n'existe aucun serveur à nous qui pourrait laisser fuiter vos données.

React : construire une page avec des briques

Écrire toute une interface directement en HTML et JavaScript devient vite confus. **React** est une bibliothèque (une boîte à outils de code prêt à l'emploi) créée par Meta, qui permet de construire une interface à partir de **composants** : de petites briques réutilisables, comme des Lego.

Chaque composant est une fonction qui reçoit des informations, appelées **props**, et renvoie ce qu'il faut afficher. Par exemple, le composant `SectionHeading` du projet reçoit un titre et le dessine dans le style commun à toutes les sections. Un composant peut aussi garder sa propre mémoire, appelée **état** (*state*) : par exemple, si un menu est ouvert. Quand l'état ou les props changent, React ne redessine que ce qui a changé.

Le projet compte une vingtaine de composants : la barre de recherche, la carte d'un anime, le calendrier, les formulaires, etc. Un composant particulier, `App`, garde l'état le plus important et décide quelle page afficher (chapitre 5).

TypeScript : JavaScript avec des garde-fous

Le code est écrit en **TypeScript**, c'est-à-dire du JavaScript avec des **types** : des étiquettes qui disent quel genre de valeur contient chaque chose. Par exemple :

```
function formatMinutes(minutes: number): string
```

Cette ligne dit que `formatMinutes` prend un nombre et rend du texte. Si un morceau de code essayait de lui donner du texte au lieu d'un nombre, TypeScript signalerait l'erreur **avant même que le site ne tourne**. Les navigateurs ne comprennent que JavaScript : TypeScript est donc converti en JavaScript lors de la construction du site ; les types ne servent qu'à attraper les erreurs.

Les API, GraphQL et JSON

Une **API** (interface de programmation) permet à un programme de demander quelque chose à un autre programme, un peu comme on commande sur un menu. AniList propose une API : notre application lui envoie une question (« donne-moi l'anime qui s'appelle Frieren ») et reçoit une réponse.

L'API d'AniList utilise **GraphQL**, un langage dans lequel on décrit *exactement* les informations voulues. Par exemple, cette question demande le titre et le nombre d'épisodes de l'anime numéro 154587 :

```
query {  
  Media(id: 154587) {  
    title { english }  
    episodes  
  }  
}
```

La réponse revient en **JSON**, un format de texte simple pour les données, lisible par les humains comme par les programmes :

```
{ "data": { "Media": { "title": { "english": "Frieren: Beyond Journey's End" }, "episodes": 28 } } }
```

Les outils autour du code

- **npm** télécharge et gère les **paquets** : du code écrit par d'autres personnes, que le projet réutilise (React, par exemple). La liste se trouve dans `package.json`.
- **Vite** est l'outil de construction du projet. Pendant le développement, il fait tourner le site sur votre propre ordinateur et le rafraîchit instantanément quand vous modifiez le code. Pour la production, il **regroupe** les nombreux fichiers sources en quelques petits fichiers optimisés.
- **Tailwind CSS** permet d'écrire le CSS avec de petits noms de classes tout prêts, directement dans les composants, comme `font-semibold` (texte en demi-gras) ou `px-4` (marge intérieure horizontale).
- **anime.js** est une petite bibliothèque d'animation (le nom est une coïncidence : elle n'a rien à voir avec les anime). Elle fait apparaître, glisser et rebondir les éléments.
- **Vitest** lance les **tests** : de petits programmes qui vérifient que le vrai code donne les bonnes réponses (chapitre 13).
- **localStorage** est un petit espace de stockage que chaque navigateur accorde à chaque site. L'application y garde vos plans enregistrés (chapitre 8).

Git, GitHub et Vercel

- **Git** enregistre l'historique de chaque modification du code : rien n'est jamais perdu et chaque changement peut être expliqué.
- **GitHub** conserve cet historique en ligne. Le dépôt de ce projet est privé.
- **Vercel** héberge le site. Chaque fois qu'une nouvelle version du code est envoyée sur GitHub, Vercel reconstruit automatiquement le site et met la nouvelle version en ligne, généralement en moins d'une minute (chapitre 14).

3. Utiliser l'application, étape par étape

Ce chapitre parcourt l'application telle qu'un utilisateur la voit. Les captures d'écran ont été prises sur le site en ligne avec la série *Frieren: Beyond Journey's End* (28 épisodes de 24 minutes).

Étape 1 : trouver une série

La page d'accueil présente un grand champ de recherche et, en dessous, une grille des séries **populaires du moment** (les séries en cours de diffusion, classées par popularité). Chaque carte montre l'affiche, la note moyenne donnée par les utilisateurs d>AniList (par exemple 87 %), le titre, le format (TV, film...) et l'année.

Tapez un nom et appuyez sur **Search**. Les résultats s'affichent dans le même genre de grille. Deux petits boutons permettent de passer d'une **grille** d'affiches à une **liste** plus détaillée (studio, genres). **Clear** ramène à la page d'accueil.

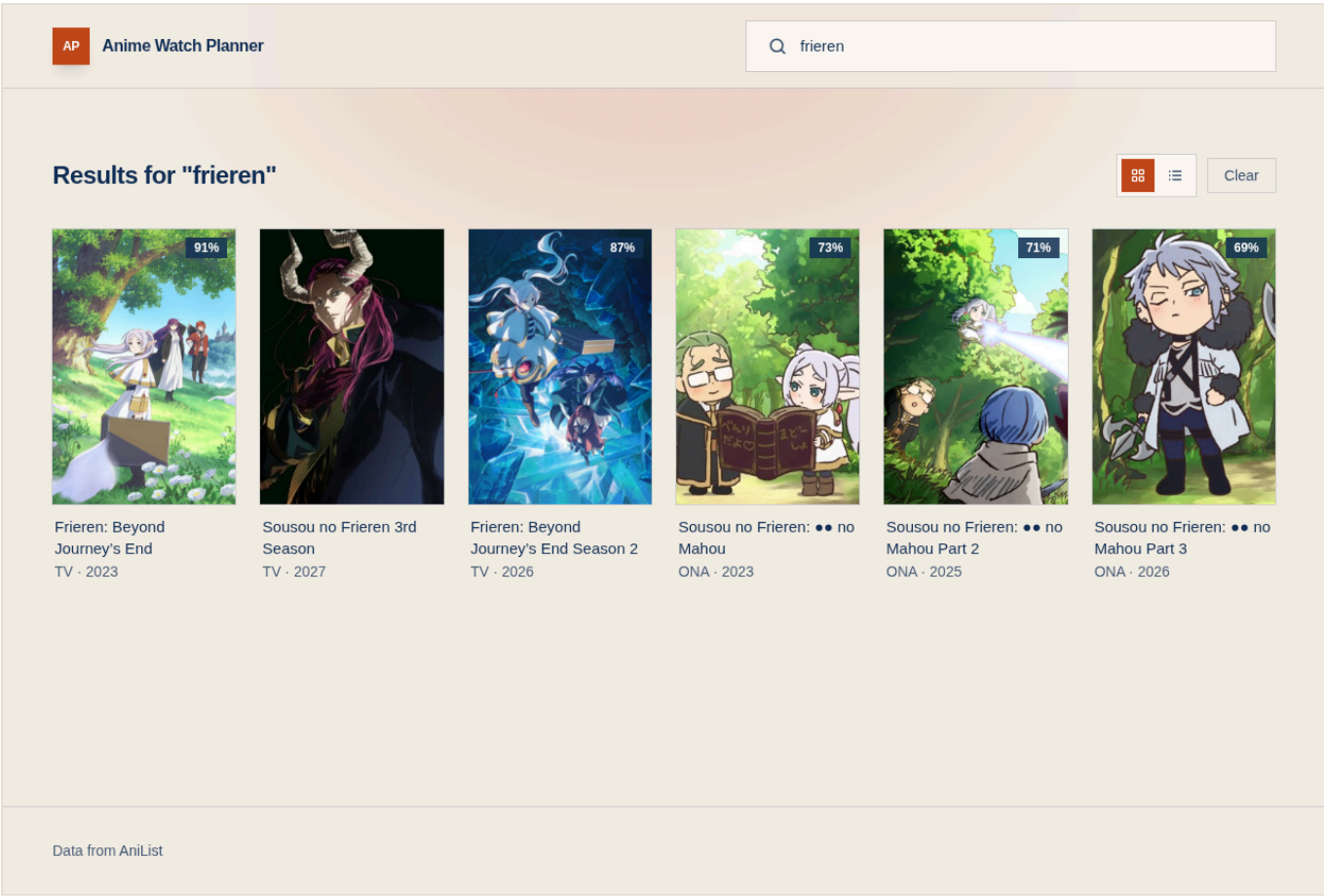


Figure 3. Les résultats de la recherche « frieren » : la série, ses saisons et ses épisodes spéciaux.

Étape 2 : la page de la série

Cliquez sur une série pour ouvrir sa page. En haut :

- l'**affiche**, le **titre** (et le titre japonais d'origine en lettres latines, appelé *romaji*, quand il est différent) ;
- une ligne d'informations : format, année, studio, nombre d'épisodes, minutes par épisode ;
- les **genres**, et si de nouveaux épisodes sortent encore, la date du prochain ;
- le **synopsis**, réduit à trois lignes avec **Read more** pour le lire en entier ;
- le bouton **Save** (enregistrer), et **Watch trailer** quand la série a une bande-annonce sur YouTube.

Toute la page change de couleur pour s'accorder à l'affiche : le fond en prend une teinte légère, et les boutons et les mises en valeur une version plus profonde, assez foncée pour que leur texte blanc reste lisible. Elle est calculée à partir d'une seule couleur qu'AniList fournit pour chaque série (chapitre 10). La page d'accueil garde le bleu marine et l'orange du site.

Étape 3 : régler son plan

À gauche, le panneau **Your plan** décide combien d'épisodes vont dans chaque journée :

- **By time** (par temps) : tapez votre temps de visionnage quotidien en minutes, ou touchez une des valeurs rapides (30, 60, 90, 120, 180).
- **By episodes** (par épisodes) : indiquez plutôt un nombre d'épisodes par jour (1, 2, 3, 4, 6). L'application le convertit en minutes en coulisses.
- **Skip openings and endings** (sauter les génériques) : retire 3 minutes à chaque épisode, pour les chansons de début et de fin. Ce n'est jamais appliqué aux films, ni aux épisodes courts de moins de 15 minutes.

Le plan se met à jour instantanément à chaque changement.

Étape 4 : choisir quand regarder

Le panneau **When you watch** transforme le plan en vraies dates :

- **Days you watch** (jours de visionnage) : touchez les jours de la semaine (M, T, W... pour Monday, Tuesday, Wednesday...), ou utilisez **Every day** (tous les jours), **Weekdays** (en semaine) ou **Weekends** (le week-end). Au moins un jour doit rester sélectionné.
- **Watch time** (heure) : une heure facultative. Sans heure, les événements de l'agenda durent « toute la journée ». Avec une heure, ils commencent à ce moment-là et durent le temps des épisodes du jour.
- **Reminder** (rappel) : disponible dès qu'une heure est choisie : à l'heure de début, ou 10, 30 ou 60 minutes avant.
- **Time zone** (fuseau horaire) : quel fuseau détermine ce qu'est « aujourd'hui » et l'heure de sortie des nouveaux épisodes. Par défaut, celui de votre appareil.

Étape 5 : lire son planning

À droite, l'encadré du haut indique **quand vous aurez fini**, par exemple « Saturday, October 17 » (samedi 17 octobre), avec une **Start date** (date de début) modifiable. En dessous, quatre chiffres : le nombre de jours de visionnage, d'épisodes par jour, de minutes par épisode, et le temps total. Si une partie de votre temps quotidien ne peut pas servir (90 minutes contiennent trois épisodes de 24 minutes, soit 72 minutes), une ligne indique combien de minutes restent inutilisées.

Le planning lui-même a deux vues :

- **List** (liste) : une ligne par jour de visionnage, avec sa date, son numéro de jour, ses épisodes (« Ep 1–3 »), sa durée (« 1h 12m ») et les **vrais titres** de ces épisodes. La case à gauche marque le jour comme regardé.
- **Calendar** (calendrier) : une grille mensuelle qui montre quels épisodes tombent à quelle date. Cliquez sur une date pour voir ses détails sous la grille et la cocher.

AP

Anime Watch Planner

Search anime

Your plan

By time

By episodes

Daily watch time

90

min

30

60

90

120

180

☐

Skip openings and endings

When you watch

Days you watch

M

T

W

T

F

S

S

Every day

Weekdays

Weekends

Watch time

--:--

⌚

No time set: calendar events are all-day. Set one for timed events and reminders.

Reminder

No reminder

▼

Time zone

This device (America / Guadeloupe)

▼

List

Calendar

☐

Today

Day 1

Ep 1-3

The Journey's End · It Didn't Have to Be Magic... · Killing Magic

1h 12m

☐

Fri, Oct 9

Day 2

Ep 4-6

The Land Where Souls Rest · Phantoms of the Dead · The Hero of the Village

1h 12m

☐

Sat, Oct 10

Day 3

Ep 7-9

Like a Fairy Tale · Frieren the Slayer · Aura the Guillotine

1h 12m

☐

Sun, Oct 11

Day 4

Ep 10-12

A Powerful Mage · Winter in the Northern Lands · A Real Hero

1h 12m

☐

Mon, Oct 12

Day 5

Ep 13-15

Aversion to One's Own Kind · Privilege of the Young · Smells Like Trouble

1h 12m

☐

Tue, Oct 13

Day 6

Ep 16-18

Long-Lived Friends · Take Care · First-Class Mage Exam

1h 12m

☐

Wed, Oct 14

Day 7

Ep 19-21

Well-Laid Plans · Necessary Killing · The World of Magic

1h 12m

☐

Thu, Oct 15

Day 8

Ep 22-24

Future Enemies · Conquering the Labyrinth · Perfect Replicas

1h 12m

Figure 4. La vue liste : chaque jour avec sa date, ses épisodes, sa durée et les vrais titres des épisodes.

AP

Anime Watch Planner

Search anime

Your plan

By time

By episodes

Daily watch time

90

min

30

60

90

120

180

☐

Skip openings and endings

When you watch

Days you watch

M

T

W

T

F

S

S

Every day

Weekdays

Weekends

Watch time

--:--

⌚

No time set: calendar events are all-day. Set one for timed events and reminders.

Reminder

No reminder

▼

Time zone

This device (America / Guadeloupe)

▼

List

Calendar

October 2026

Today

<

>

Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8 Ep 1-3	9 Ep 4-6	10 Ep 7-9	11 Ep 10-12
12 Ep 13-15	13 Ep 16-18	14 Ep 19-21	15 Ep 22-24	16 Ep 25-27	17 Ep 28	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

■ Planned

■ Watched

Figure 5. La vue calendrier : 28 épisodes à raison de 3 par jour, du jeudi 8 au samedi 17 octobre.

11 / 149

Pour une série **encore en diffusion** qui n'a pas encore de nombre d'épisodes définitif, l'application planifie les épisodes déjà sortis et le signale dans un encadré teinté de la couleur de la page, avec la date du prochain épisode. Les épisodes prévus à une date antérieure à leur sortie sont marqués **Not out yet** (pas encore sorti).

Étape 6 : enregistrer et suivre sa progression

Save enregistre le plan dans votre navigateur. Le bouton affiche alors **Saved** ; si vous modifiez un réglage, il devient **Update** ; si le navigateur refuse d'enregistrer, il affiche **Retry save**. Cocher un jour enregistre automatiquement le plan, car la progression doit bien être stockée quelque part.

De retour sur la page d'accueil, une rangée **Continue watching** (reprendre) montre chaque plan enregistré avec le jour où reprendre (« Day 1 of 10 · today »), une barre de progression et la date du prochain nouvel épisode. Si vous avez regardé quelque chose plusieurs jours de suite, une **série de jours** apparaît (« 3-day streak »). Dans un plan, un bouton mène directement à **Today's episodes** (les épisodes du jour) ou à **Resume day N** (reprendre au jour N).

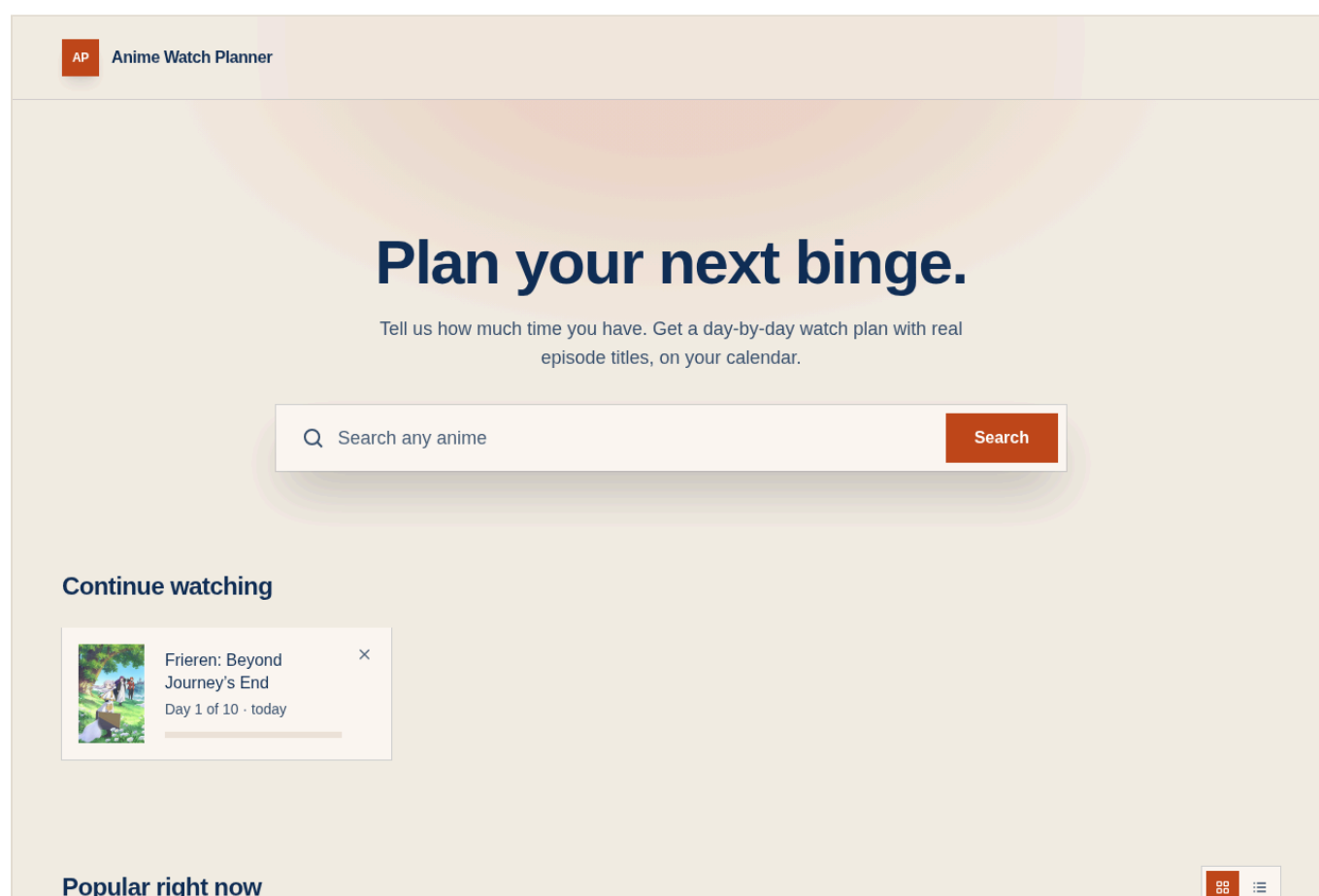


Figure 6. Un plan enregistré dans la rangée « Continue watching ».

Étape 7 : exporter et partager

Le panneau **Share & export** propose deux boutons :

- **Add to calendar** télécharge un fichier **.ics** . En l'ouvrant, chaque jour de visionnage s'ajoute à votre agenda, avec les titres des épisodes dans chaque événement, et les dates de sortie des prochains épisodes pour les séries encore en diffusion. Sur iPhone ou iPad, le fichier s'ouvre directement pour que Safari propose « Ajouter au calendrier ».

- **Copy link** copie l'adresse de la série, comme `anime-planner.vercel.app/anime/154587-frieren-beyond-journey-s-end`. La personne qui l'ouvre obtient la série avec des informations à jour et les réglages par défaut.

Les autres panneaux

- **Where to watch** (où regarder) liste les plateformes de streaming officielles connues d'AniList pour la série, avec leur logo. La disponibilité dépend de votre pays.
- **Watch order** (ordre de visionnage) apparaît pour les franchises à plusieurs entrées (saison 1, saison 2, un film...), dans l'ordre de sortie. Cliquez sur une entrée pour l'ouvrir.

Sur un téléphone

La mise en page s'adapte aux petits écrans : les panneaux s'empilent verticalement, le calendrier affiche des points au lieu d'étiquettes, et les champs de saisie sont assez grands pour que le téléphone ne zoome pas quand on les touche.

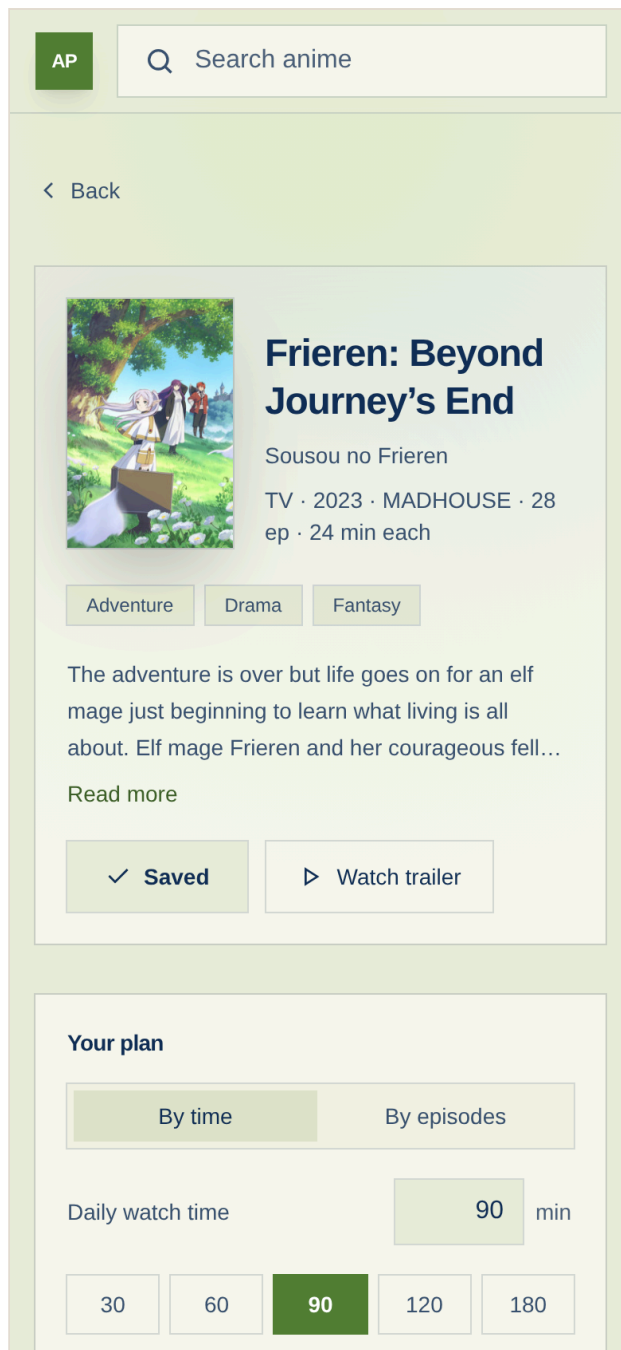


Figure 7. La même page sur un téléphone.

4. L'organisation du projet

L'arborescence

Voici tous les fichiers du projet, générés à partir du vrai dépôt au moment de la fabrication de ce guide :

```
anime-planner/  
├── guide/  
├── public/  
│   ├── _headers  
│   ├── _redirects  
│   ├── favicon.svg  
│   └── robots.txt  
├── scripts/  
│   └── prerender.ts  
├── src/  
│   ├── api/  
│   │   └── anilist.ts  
│   ├── components/  
│   │   ├── AnimeCard.tsx  
│   │   ├── AnimeListRow.tsx  
│   │   ├── CalendarView.tsx  
│   │   ├── ConfigForm.tsx  
│   │   ├── ConfigureStep.tsx  
│   │   ├── ErrorBoundary.tsx  
│   │   ├── icons.tsx  
│   │   ├── Motion.tsx  
│   │   ├── SavedSchedules.tsx  
│   │   ├── ScheduleView.tsx  
│   │   ├── SearchBar.tsx  
│   │   ├── SearchStep.tsx  
│   │   ├── SectionHeading.tsx  
│   │   ├── StreamingIcon.tsx  
│   │   └── WatchPrefsForm.tsx  
│   ├── lib/  
│   │   ├── calendar.test.ts  
│   │   ├── calendar.ts  
│   │   ├── download.ts  
│   │   ├── format.test.ts  
│   │   ├── format.ts  
│   │   ├── ics.test.ts  
│   │   ├── ics.ts  
│   │   ├── platform.test.ts  
│   │   ├── platform.ts  
│   │   ├── progress.test.ts  
│   │   ├── progress.ts  
│   │   ├── route.test.ts  
│   │   ├── route.ts  
│   │   ├── schedule.test.ts  
│   │   ├── schedule.ts  
│   │   ├── seo.test.ts  
│   │   ├── seo.ts  
│   │   ├── share.test.ts  
│   │   ├── share.ts  
│   │   ├── storage.test.ts  
│   │   ├── storage.ts  
│   │   ├── theme.test.ts  
│   │   └── theme.ts  
│   ├── motion/  
│   │   ├── hooks.ts  
│   │   └── presets.ts  
│   ├── App.tsx  
│   ├── index.css  
│   ├── main.tsx  
│   └── types.ts  
├── .gitignore  
├── CLAUDE.md  
└── index.html
```

```

├── package-lock.json
├── package.json
├── README.md
├── tsconfig.app.json
├── tsconfig.json
├── tsconfig.node.json
├── vercel.json
└── vite.config.ts

```

À quoi sert chaque dossier

Dossier	Rôle	Exemples
(racine)	Les réglages du projet : quels paquets utiliser, comment vérifier et construire le code, comment l'héberger	package.json , vite.config.ts , vercel.json
public/	Des fichiers servis tels quels	l'icône de l'onglet, robots.txt
src/	L'application elle-même	App.tsx , main.tsx , index.css , types.ts
src/lib/	La logique pure : des calculs sans écran ni réseau	le moteur de planning, les dates, la progression, le fichier .ics
src/api/	La connexion à AniList	anilist.ts
src/motion/	Les réglages et outils d'animation	presets.ts , hooks.ts
src/components/	Les morceaux de l'interface	la barre de recherche, les cartes, les formulaires, le calendrier
scripts/	Des programmes lancés pendant la construction du site	prerender.ts
guide/	Les sources de ce guide	les chapitres en français et en anglais, le script de fabrication

Les fichiers terminés par `.test.ts` , à côté du code dans `src/lib/` , sont les **tests** (chapitre 13).

La règle d'or : garder la logique pure

L'idée la plus importante dans l'organisation du projet est la différence entre **logique pure** et **effets de bord**.

Une **fonction pure** ne fait que calculer : avec les mêmes entrées, elle donne toujours la même réponse, et elle ne modifie rien en dehors d'elle-même. « 28 épisodes de 24 minutes avec 90 minutes par jour donnent 10 jours » est pur. Un **effet de bord** touche le monde extérieur : dessiner à l'écran, télécharger un fichier, interroger AniList, lire l'horloge ou le stockage du navigateur.

Le projet les sépare :

- `src/lib/` contient la logique pure. Le moteur (`schedule.ts`), les dates (`calendar.ts`), la progression (`progress.ts`), le fichier calendrier (`ics.ts`), les adresses (`route.ts`), les balises de référencement (`seo.ts`), les outils de mise en forme (`format.ts`) et les couleurs de la page d'une série (`theme.ts`) ne touchent jamais à l'écran ni au réseau.
- Les quelques effets de bord sont isolés dans de petits endroits clairement nommés : `storage.ts` (stockage du navigateur), `download.ts` (téléchargement de fichiers), `platform.ts` (détection de l'appareil), `api/anilist.ts` (réseau), et les composants et `App.tsx` (écran et état).

Pourquoi est-ce important ? Le code pur est facile à comprendre et **facile à tester** : un test lui donne simplement des entrées et vérifie la réponse, sans navigateur ni Internet. C'est pourquoi les 131 tests visent tous `src/lib/`.

Les dépendances

Les paquets sont du code écrit par d'autres que le projet réutilise. Ils sont listés dans `package.json`, en deux groupes : les `dependencies` font partie du site que les visiteurs téléchargent ; les `devDependencies` ne servent que pendant le développement et la construction. Ce tableau est généré à partir des vrais fichiers du projet :

Paquet	Version exacte	Utilisé
<code>animejs</code>	4.5.0	dans le site
<code>react</code>	18.3.1	dans le site
<code>react-dom</code>	18.3.1	dans le site
<code>@tailwindcss/vite</code>	4.3.3	pour développer
<code>@types/react</code>	18.3.31	pour développer
<code>@types/react-dom</code>	18.3.7	pour développer
<code>@vitejs/plugin-react</code>	6.1.2	pour développer
<code>highlight.js</code>	11.12.0	pour développer
<code>marked</code>	18.1.0	pour développer
<code>tailwindcss</code>	4.3.3	pour développer
<code>tsx</code>	4.23.15	pour développer
<code>typescript</code>	7.0.2	pour développer
<code>vite</code>	8.3.3	pour développer
<code>vitest</code>	5.0.3	pour développer

En tout, `package-lock.json` fixe la version exacte de **158** paquets : ceux du tableau et tous ceux dont ils dépendent à leur tour.

Leur rôle :

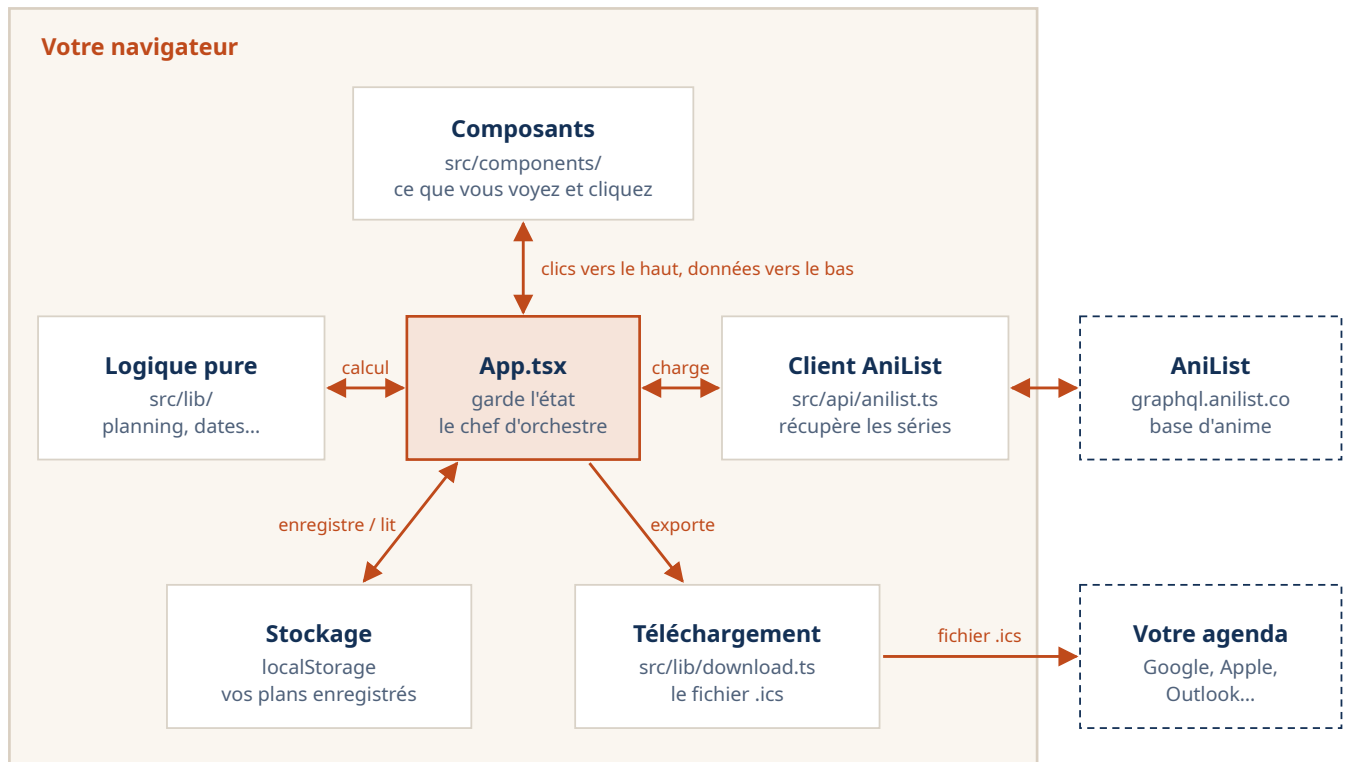
- **react** et **react-dom** : construire l'interface à partir de composants et la dessiner dans la page.
- **animejs** : les animations.
- **tailwindcss** et **@tailwindcss/vite** : le système de styles et son branchement sur Vite.
- **vite** et **@vitejs/plugin-react** : le serveur de développement et la construction pour la production, avec la prise en charge de React.
- **typescript** : la vérification des types.
- **@types/react** et **@types/react-dom** : la description des types de React, pour que TypeScript connaisse ses fonctions.
- **vitest** : le lancement des tests.
- **tsx** : l'exécution directe de scripts TypeScript avec Node.js (le script de prérendu et la fabrication de ce guide).
- **marked** et **highlight.js** : transformer les chapitres de ce guide en HTML et colorer son code. Ils ne servent qu'à fabriquer le guide.

À propos de package-lock.json

Quand npm installe un paquet, ce paquet a souvent besoin d'autres paquets, qui en demandent d'autres, et ainsi de suite. `package-lock.json` enregistre **la version exacte de chacun d'eux**, pour que le projet s'installe à l'identique sur n'importe quel ordinateur, aujourd'hui comme dans un an. Il est écrit et mis à jour automatiquement par npm, jamais à la main, et contient plusieurs milliers de lignes de noms, de numéros de version et d'empreintes de sécurité (des *hashes*). C'est pourquoi c'est **le seul fichier non imprimé** dans l'annexe A : il ajouterait une cinquantaine de pages illisibles, sans rien expliquer de plus que le tableau ci-dessus.

5. Comment tout s'assemble

Vue d'ensemble



Tout ce qui se trouve dans le grand cadre fonctionne dans votre navigateur. Seules deux choses sont à l'extérieur : **AniList**, que l'application interroge pour obtenir les informations sur les séries, et **votre agenda**, qui reçoit le fichier `.ics` si vous exportez votre plan.

Au centre se trouve **App.tsx**, le chef d'orchestre. Il garde les informations importantes (quelle série est ouverte, vos réglages, vos plans enregistrés) et décide quelle page afficher. Autour de lui :

- **Les composants** dessinent ce que vous voyez. Ils reçoivent d' **App** les données à afficher et, quand vous cliquez ou tapez, ils préviennent **App** de ce qui s'est passé.
- **La logique pure** (`src/lib/`) fait les calculs : le planning, les dates, la progression, le fichier calendrier.
- **Le client AniList** (`src/api/anilist.ts`) envoie les questions à AniList et met de l'ordre dans les réponses.
- **Le stockage du navigateur** garde vos plans enregistrés, via `src/lib/storage.ts`.
- **Le téléchargement** transforme le texte du calendrier en un fichier que votre navigateur enregistre.

Les données descendent, les événements remontent

Les projets React suivent une règle simple, et celui-ci la suit strictement : **les données descendent, les événements remontent**.

App transmet les données à ses composants sous forme de props : « voici la série, voici les réglages, voici le résultat du calcul ». Les composants ne modifient jamais ces données eux-mêmes. À la place, **App** leur fournit aussi des fonctions à appeler, comme `onSave` ou `onConfigChange`. Quand vous cliquez sur Save, le bouton appelle `onSave` ; **App** enregistre le plan, met à jour son état, et React redessine tout ce qui en dépend.

Comme tout l'état important est rassemblé au même endroit, on ne se demande jamais d'où vient une information ni qui a le droit de la changer.

L'état que garde App

État	Ce qu'il contient
<code>step</code>	La page affichée : <code>'search'</code> (accueil et résultats) ou <code>'configure'</code> (la page d'une série)
<code>query</code> , <code>searchedFor</code>	Ce qui est tapé dans la recherche, et ce que concernent les résultats affichés
<code>results</code> , <code>loading</code> , <code>error</code>	Les résultats, si une recherche est en cours, et un éventuel message d'erreur
<code>popular</code>	La liste « Popular right now » (<code>null</code> pendant le chargement)
<code>selected</code>	La série ouverte
<code>config</code>	Votre plan : minutes par jour et saut des génériques ou non
<code>calendar</code>	Quand vous regardez : date de début, jours, heure, rappel, fuseau horaire
<code>savedSchedules</code>	Tous les plans enregistrés dans ce navigateur
<code>saveFailed</code>	Si le dernier enregistrement a échoué
<code>opening</code>	Si une série se charge depuis un lien direct

Suivre une action : vous cliquez sur une affiche

Voici exactement ce qui se passe, étape par étape, quand vous cliquez sur une série de la page d'accueil :

1. L'affiche est un composant `AnimeCard` . Son bouton appelle `onSelect(anime)` .
2. Cette fonction est `handleSelect` , dans `App.tsx` . Si vous avez déjà enregistré cette série, elle reprend vos réglages enregistrés ; sinon elle garde les réglages actuels. Elle appelle `openAnime` .
3. `openAnime` met l'état à jour : `selected` devient cette série, `step` devient `'configure'` , et l'adresse dans la barre du navigateur devient celle de la série, par exemple `/anime/154587-frieren-beyond-journey-s-end` .
4. React redessine : comme `step` vaut maintenant `'configure'` , App affiche la page `ConfigureStep` au lieu de l'accueil, et donne à la page les couleurs que `posterTheme` a tirées de l'affiche de la série (chapitre 10).
5. Le planning est calculé par `generateSchedule(selected, config)` . App entoure ce calcul de `useMemo` , un outil de React qui mémorise le résultat et ne recalcule que si la série ou les réglages changent.
6. La carte ne contenait que les informations légères de la liste (ni synopsis, ni titres d'épisodes). En arrière-plan, `openAnime` demande donc tous les détails à AniList avec `getAnimeById` . Quand ils arrivent, `selected` est mis à jour et la page se complète : synopsis, vrais titres des épisodes, liens de streaming et ordre de visionnage.

Une autre action : vous changez votre temps quotidien

1. Vous touchez « 60 » dans le panneau du plan. `ConfigForm` appelle `onConfigChange({ dailyBudget: 60, skipOpEd: false })` .
2. App enregistre le nouveau `config` .

3. Comme `config` a changé, `useMemo` relance `generateSchedule` : 60 minutes contiennent maintenant deux épisodes de 24 minutes par jour, et le plan passe de 10 à 14 jours.
4. `ScheduleView` se redessine avec le nouveau résultat. Les lignes réapparaissent en glissant, pour signaler visuellement que le plan a changé, et les chiffres défilent jusqu'à leurs nouvelles valeurs.

Ouvrir un lien directement

Quand quelqu'un ouvre directement `anime-planner.vercel.app/anime/154587-...` (depuis un lien partagé, par exemple) :

1. Le navigateur charge `index.html`, qui charge le code. `main.tsx` dessine `App` dans la page.
2. Au démarrage, `App` lance `resolveLocation`. Cette fonction lit l'adresse, et `parseAnimePath` en extrait le numéro de la série : 154587.
3. Si cette série est enregistrée dans le navigateur, elle s'ouvre instantanément à partir de la copie enregistrée. Sinon, un message « Loading show... » s'affiche pendant que `getAnimeById` la récupère sur AniList.
4. Les boutons Précédent et Suivant fonctionnent aussi : le navigateur les signale par un événement appelé `popstate`, et `App` relance `resolveLocation`.

6. Le moteur de planning

Le moteur est le cœur du projet. Il se trouve dans `src/lib/schedule.ts`, une centaine de lignes, et ne fait qu'une chose : **à partir d'une série et de votre plan, répartir les épisodes en journées**. Il est pur (chapitre 4) : il ne connaît ni les dates, ni l'écran, ni Internet. Il travaille en trois étapes, chacune avec sa fonction.

Étape 1 : combien dure un épisode ? (`getNetEpisodeRuntime`)

AniList donne à chaque série une durée (`duration`) : les minutes par épisode, ou la durée totale pour un film. Le moteur part de ce nombre :

- Si **sauter les génériques** est désactivé, un épisode dure sa `duration` complète.
- S'il est activé, le moteur retire **3 minutes** (environ une minute et demie pour le générique de début et autant pour celui de fin), mais **seulement** si la série n'est pas un film **et** que l'épisode dure au moins **15 minutes**. Les épisodes très courts n'ont souvent pas de vrai générique, et un film n'a pas de génériques entre ses épisodes.
- Si AniList n'indique aucune durée, la fonction renvoie `null`, qui veut dire « inconnu ».

Série	Durée	Saut activé ?	Durée nette
Un épisode TV normal	24 min	non	24 min
Un épisode TV normal	24 min	oui	21 min
Un format court (TV short)	12 min	oui	12 min (trop court pour être réduit)
Un film	120 min	oui	120 min (les films ne sont jamais réduits)

Étape 2 : quels épisodes ? (`getWatchableEpisodes`)

Ensuite, le moteur dresse la liste des épisodes à planifier :

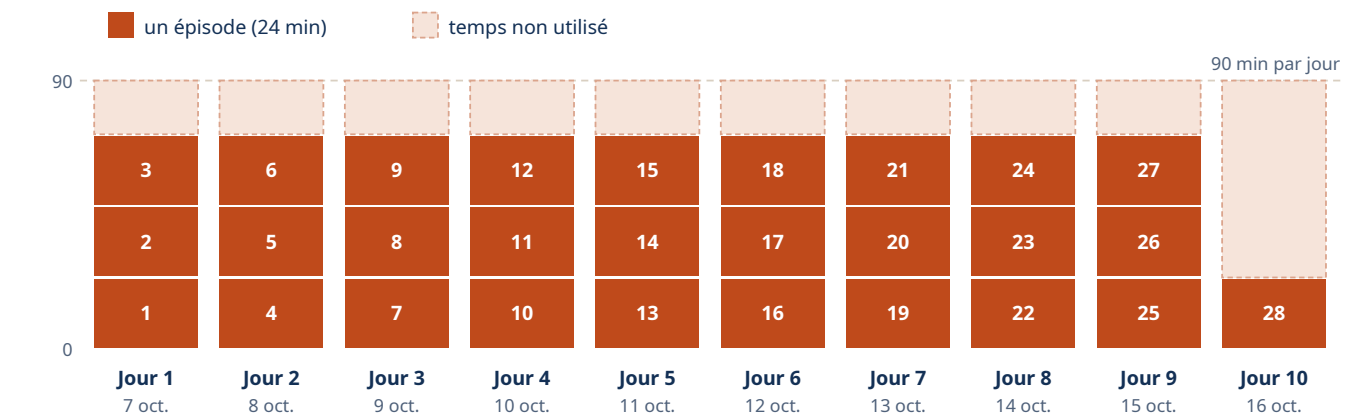
- Normalement, tous les épisodes de 1 au total : une série de 28 épisodes donne `[1, 2, ..., 28]`.
- Un **film** est toujours un seul « épisode » : `[1]`.
- Une série **encore en diffusion** n'a souvent pas encore de total sur AniList (`episodes` est vide). Plutôt que d'abandonner, le moteur utilise les épisodes **déjà sortis**. AniList indique quel épisode sort ensuite (`nextAiringEpisode`) : si l'épisode 1 150 sort ensuite, alors 1 149 sont sortis, et le plan couvre les épisodes 1 à 1 149. C'est ainsi qu'une série comme One Piece, avec plus de mille épisodes et sans total définitif, obtient quand même un vrai plan. Le résultat est marqué `isFallback`, et la page affiche un message « still airing » (encore en diffusion).
- S'il n'y a vraiment rien à planifier (pas de total, pas en diffusion, ou en diffusion mais rien de sorti), elle renvoie `null`.

Étape 3 : répartir en journées (`generateSchedule`)

La règle est simple : **autant d'épisodes entiers que votre temps quotidien en contient, chaque jour**, dans l'ordre, jusqu'à épuisement des épisodes. En maths : épisodes par jour = `floor(temps quotidien ÷ durée d'un épisode)`, où `floor` veut dire « arrondir à l'entier inférieur ».

Prenons *Frieren* : 28 épisodes de 24 minutes, avec 90 minutes par jour. `90 ÷ 24 = 3,75`, arrondi à **3 épisodes par jour**. Trois épisodes durent 72 minutes : 18 minutes restent inutilisées chaque jour. 28 épisodes à 3 par jour font 9 journées complètes (27 épisodes) plus une 10e journée avec le dernier épisode

seul : **10 jours**, pour un total de $28 \times 24 = 672$ minutes, soit 11 heures 12 minutes. C'est exactement ce que montrent les captures du chapitre 3.



La fonction renvoie soit un plan, soit une erreur. Un plan contient :

Champ	Signification	Frieren à 90 min
days	La liste des jours, chacun avec son numéro, ses épisodes et ses minutes	10 jours
netRuntime	Minutes par épisode après la réduction éventuelle	24
episodesPerDay	Épisodes par journée complète	3
episodeCount	Nombre d'épisodes planifiés	28
totalMinutes	Temps de visionnage total	672
unusedMinutesPerDay	Minutes restantes chaque journée complète	18
overBudget	Un épisode dure plus que le temps quotidien	non
isPartial	Le nombre d'épisodes est « sortis jusqu'ici », pas un total définitif	non

Quand un épisode dure plus que votre temps quotidien

Si vous choisissez 45 minutes par jour pour une série aux épisodes de 50 minutes, aucun épisode n'y tient. Au lieu de renvoyer un plan vide, le moteur met **un épisode par jour** et active `overBudget`, et la page explique que chaque journée dépasse.

Quand un plan est impossible

Le moteur renvoie une erreur, que la page transforme en phrase claire :

Erreur	Quand	Message affiché
invalid-budget	Le temps quotidien vaut 0, est négatif ou vide	« Enter a daily watch time above 0 minutes. »
unknown-duration	AniList n'indique pas la durée des épisodes	« AniList doesn't list an episode length for this show. »
unknown-episode-count	Pas de total et rien de sorti	« AniList doesn't list an episode count for this show yet... »

Le code, ligne par ligne

Voici le cœur de `generateSchedule` (le fichier complet est dans l'annexe A) :

```
const fitsPerDay = Math.floor(config.dailyBudget / netRuntime);
const overBudget = fitsPerDay === 0;
const episodesPerDay = Math.max(1, fitsPerDay);

const days: ScheduleDay[] = [];
for (let i = 0; i < episodes.length; i += episodesPerDay) {
  const chunk = episodes.slice(i, i + episodesPerDay);
  days.push({ day: days.length + 1, episodes: chunk, minutes: chunk.length * netRuntime });
}
```

- `Math.floor(...)` arrondit à l'entier inférieur : 3,75 devient 3.
- Si cela donne 0, `overBudget` est vrai, et `Math.max(1, ...)` garantit qu'il y a quand même au moins un épisode par jour.
- La boucle parcourt la liste des épisodes par pas de `episodesPerDay`. À chaque tour, `slice` découpe le groupe (« chunk ») d'épisodes suivant, et une nouvelle journée est ajoutée avec son numéro, ses épisodes et sa durée.

Pourquoi « glouton » et pas optimal

Cette méthode s'appelle **gloutonne** : elle remplit chaque journée autant que possible, dans l'ordre, sans regarder plus loin. Elle perd les minutes restantes (18 par jour dans l'exemple de Frieren). Une méthode plus astucieuse pourrait, par exemple, alterner des journées plus courtes et plus longues pour utiliser chaque minute. Le projet garde volontairement la règle simple : elle est prévisible (chaque journée complète se ressemble), facile à expliquer et facile à tester. De toute façon, les épisodes ont tous la même durée, donc une méthode plus astucieuse servirait rarement.

« Par épisodes » n'est pas un second moteur

Le panneau du plan permet de choisir « 3 épisodes par jour » plutôt que des minutes. Ce n'est qu'une facilité du formulaire (`ConfigForm.tsx`) : le formulaire convertit ce choix en minutes exactes ($3 \times 24 = 72$ minutes) et enregistre ce nombre. Comme 72 divisé par 24 fait exactement 3, le moteur retombe sur 3 épisodes par jour. Le moteur et les données enregistrées ne voient jamais que des minutes : il n'y a qu'une seule règle à tester et à qui faire confiance.

7. Les dates, le calendrier et l'export .ics

Deux couches séparées

Le moteur (chapitre 6) ne connaît que « Jour 1, Jour 2, ... Jour 10 ». Il ignore quelle est la date d'aujourd'hui. Transformer ces numéros en vraies dates est le rôle d'une seconde couche, `src/lib/calendar.ts`, qui utilise vos **préférences de calendrier** :

Préférence	Signification	Par défaut
<code>startDate</code>	La date de début, au format <code>aaaa-mm-jj</code>	aujourd'hui
<code>watchDays</code>	Les jours de la semaine où vous regardez (0 = dimanche ... 6 = samedi)	tous les jours
<code>watchTime</code>	Une heure de la journée, au format <code>HH:MM</code> , ou rien	rien (« n'importe quand »)
<code>reminderMinutes</code>	Combien de temps avant <code>watchTime</code> vous prévenir, ou rien	rien
<code>timezone</code>	Un nom de fuseau horaire comme <code>America/Guadeloupe</code> , ou rien pour celui de l'appareil	rien

Garder les deux couches séparées signifie que changer vos jours de visionnage ne change jamais quels épisodes sont regroupés, et que le moteur reste simple.

Des numéros de jour aux dates (`scheduleDates`)

La règle : **le jour 1 est le premier jour de visionnage à partir de la date de début ; chaque jour suivant est le prochain jour de visionnage après le précédent.**

Exemple : vous commencez le mercredi 7 octobre et regardez **en semaine seulement**. Le Jour 1 est le mercredi 7, le Jour 2 le jeudi 8, le Jour 3 le vendredi 9. Le samedi et le dimanche sont sautés : le Jour 4 est le lundi 12, et ainsi de suite. Le plan de Frieren en 10 jours se termine alors le mardi 20 octobre au lieu du vendredi 16.

Dans le code, la boucle avance d'un jour à la fois et saute tout jour dont le jour de la semaine n'est pas dans votre liste :

```
for (const day of days) {
  while (!allowed.has(weekday(date))) date = addDays(date, 1);
  dates.set(day.day, date);
  date = addDays(date, 1);
}
```

Si la liste des jours de visionnage était vide, cette boucle ne s'arrêterait jamais. La fonction traite donc une liste vide comme « tous les jours », et le formulaire ne laisse jamais désélectionner le dernier jour restant.

Pourquoi les dates sont du texte, calculées en UTC

Les dates sont stockées sous forme de texte simple, comme `2026-10-07`, et l'arithmétique des jours (« ajouter un jour ») se fait en **UTC**, l'heure de référence mondiale, qui ne change jamais d'heure en été ou en hiver.

Pourquoi ? Dans la plupart des pays qui changent d'heure, un jour par an ne dure que 23 heures et un autre en dure 25. Un code qui ajoute « 24 heures » pour passer au jour suivant peut alors tomber deux fois sur le même jour, ou en sauter un. Travailler avec des dates simples en UTC évite complètement ce piège ; un test vérifie qu'un plan qui traverse un changement d'heure garde bien une date par jour.

Seules deux fonctions manipulent de vrais instants, et donc des fuseaux horaires : `localIso`, qui répond à « quelle date sommes-nous maintenant, dans ce fuseau ? », et `formatAiring`, qui affiche l'heure de sortie d'un épisode. Elles utilisent **Intl**, la boîte à outils de dates et d'heures intégrée à tous les navigateurs, qui connaît tous les fuseaux horaires du monde.

La grille du mois (`monthGrid`)

La vue calendrier montre des semaines entières commençant le lundi. `monthGrid` trouve le lundi précédant (ou égal à) le 1er du mois et construit des rangées de 7 jours jusqu'à couvrir le mois. Les jours qui appartiennent au mois précédent ou suivant sont marqués pour être affichés en plus pâle. Passer d'un mois à l'autre est un simple calcul sur « année × 12 + mois », qui gère le passage de décembre à janvier sans cas particulier.

Les séries encore en diffusion

Pour une série encore en diffusion, AniList peut donner les prochaines dates de sortie (`airingSchedule`, seulement dans la requête détaillée) ou au moins la prochaine (`nextAiringEpisode`). `upcomingAirings` prend la meilleure des deux, ne garde que les dates futures, les trie, et calcule la date de chacune dans votre fuseau horaire. L'application s'en sert à plusieurs endroits :

- l'en-tête de la série : « Episode 13 airs Sat, Oct 10 at 5:30 PM (in 3 days) » ;
- le calendrier : un point ou une étiquette bleus aux dates de sortie ;
- la liste : **Not out yet** sur un jour planifié qui arrive avant la sortie de ses épisodes ;
- la rangée des plans enregistrés : « New ep 13 · Sat, Oct 10 » ;
- le fichier `.ics` : un événement par sortie à venir.

Le fichier .ics

`.ics` (iCalendar) est un format texte standard que toutes les applications d'agenda comprennent. `src/lib/ics.ts` le construit comme du texte simple ; `src/lib/download.ts` le confie ensuite au navigateur sous forme de fichier. Voici un événement, tel que l'application l'écrit pour le Jour 1 de Frieren sans heure choisie :

```
BEGIN:VEVENT
UID:154587-1@anime-planner
DTSTAMP:20261007T120000Z
DTSTART;VALUE=DATE:20261007
DTEND;VALUE=DATE:20261008
SUMMARY:Frieren: Beyond Journey's End — Ep 1-3
DESCRIPTION:The Journey's End\nIt Didn't Have to Be Magic...\nKilling Magic
END:VEVENT
```

- `UID` est le nom permanent de l'événement. Il est construit à partir du numéro de la série et du numéro du jour : si vous importez à nouveau le fichier après avoir changé votre plan, votre agenda **met à jour** les événements au lieu de créer des doublons.
- Sans heure, l'événement dure **toute la journée** (`VALUE=DATE`, se terminant le lendemain).

- Avec une heure, par exemple 20:00, l'événement commence à 20 h et dure le temps des épisodes du jour. Ces heures sont écrites sans fuseau horaire (« flottantes ») : elles veulent dire 20 h où que vous soyez, ce qui correspond bien à un créneau personnel.
- Avec un rappel, un bloc `VALARM` demande à l'agenda de vous prévenir, par exemple 30 minutes avant (`TRIGGER: -PT30M`).
- Les sorties des prochains épisodes sont de vrais instants : elles sont écrites en UTC (terminées par `Z`) et marquées `TRANSP:TRANSPARENT`, ce qui veut dire qu'elles ne vous rendent pas « occupé ».

Le format a des règles strictes, et le code les suit : les lignes se terminent par les deux caractères « retour chariot + saut de ligne » ; les virgules, points-virgules, barres obliques inverses et sauts de ligne à l'intérieur du texte sont **échappés** avec une barre oblique inverse ; et les lignes de plus de 75 caractères sont **repliées** sur la ligne suivante, qui commence par une espace. L'échappement compte aussi pour la sécurité : sans lui, un titre de série contenant un saut de ligne pourrait ajouter de fausses lignes à votre agenda (chapitre 12).

Sur iPhone et iPad

Sur iOS, télécharger un fichier `.ics` est peu pratique. L'application l'ouvre donc directement dans l'onglet, ce qui pousse Safari à proposer son propre écran « Ajouter au calendrier ». `src/lib/platform.ts` reconnaît les iPhone et les iPad, y compris les iPad récents, qui se font passer pour des Mac : un « Mac » avec écran tactile est traité comme un iPad.

8. Enregistrer ses plannings et sa progression

Où sont stockés les plans

Il n'y a ni compte ni serveur : vos plans sont donc stockés dans votre propre navigateur, dans le **localStorage**, un petit espace de stockage (quelques mégaoctets) que chaque navigateur accorde à chaque site. Tout se trouve sous un seul nom (une « clé ») : `anime-planner:schedules:v1`. Le `v1` final est un numéro de version : un futur format incompatible pourrait utiliser une nouvelle clé sans embrouiller les anciennes données.

La valeur est une liste de plans enregistrés, écrite en texte JSON. Un plan enregistré ressemble à ceci (abrégé) :

```
{
  "id": "6f1c2a7e-...",
  "savedAt": 1791345600000,
  "anime": { "id": 154587, "title": { "romaji": "Sousou no Frieren", "english": "Frieren: Beyond Journey's End" }, "episodes": 28, "duration": 24 },
  "config": { "dailyBudget": 90, "skipOpEd": false },
  "calendar": { "startDate": "2026-10-07", "watchDays": [0, 1, 2, 3, 4, 5, 6], "watchTime": null,
  "reminderMinutes": null, "timezone": null },
  "watched": [{ "episode": 1, "completedAt": 1791349200000 }]
}
```

- `id` est un code unique aléatoire.
- `savedAt` et `completedAt` sont des instants, comptés en millisecondes depuis le 1er janvier 1970 (une convention courante en programmation).
- `anime` est un **instantané** des informations de la série au moment de l'enregistrement, pour que le plan s'affiche immédiatement, même hors ligne.
- `config` et `calendar` sont vos réglages.
- `watched` liste les épisodes que vous avez cochés, et quand.

Un plan par série

Enregistrer une série déjà enregistrée **remplace** ses réglages au lieu d'en ajouter une seconde copie, et garde votre progression. Quand vous ouvrez un plan enregistré, l'instantané s'affiche aussitôt, puis l'application récupère discrètement des informations à jour sur AniList (une série en cours peut avoir de nouveaux épisodes).

La progression se compte par épisode, pas par jour

C'est l'un des choix les plus subtils du projet. Quand vous cochez « Jour 1 », l'application enregistre que les **épisodes 1, 2 et 3** sont vus, et non « le jour 1 ».

Pourquoi ? Parce que les journées sont recalculées dès que votre plan change. Avec 90 minutes par jour, le Jour 2 contient les épisodes 4 à 6. Si vous passez à 60 minutes par jour, le Jour 2 devient les épisodes 3 et 4. Si la progression était enregistrée par numéro de jour, changer votre plan marquerait soudain d'autres épisodes comme vus. Enregistrée par épisode, votre progression reste juste quoi que vous changiez : une journée s'affiche simplement comme vue quand **tous** ses épisodes le sont.

Cocher une journée fonctionne ainsi (`toggleEpisodesWatched`) : si tous les épisodes de cette journée sont déjà vus, ils sont tous décochés ; sinon, ceux qui manquent sont cochés, avec l'heure actuelle. Et comme la progression a besoin d'un plan enregistré pour exister, **cocher une journée enregistre automatiquement le plan** s'il ne l'était pas encore.

Lire avec prudence

Le stockage du navigateur n'est pas toujours fiable. Il peut être désactivé (dans certains modes de navigation privée), plein, ou contenir des données abîmées. La règle de `storage.ts` est : **aucun problème de stockage ne doit jamais faire planter l'application**.

- Chaque lecture et chaque écriture est entourée de `try ... catch`, une construction qui intercepte les erreurs. Si la lecture échoue, l'application se comporte comme si rien n'était enregistré ; si l'écriture échoue, le bouton affiche **Retry save**.
- Chaque plan enregistré est **vérifié** à la lecture (`hasBaseShape`) : il doit avoir un identifiant, une date d'enregistrement, une série avec un numéro et un titre, et des réglages valides. Tout le reste est ignoré plutôt que cru sur parole.
- Les préférences de calendrier sont vérifiées champ par champ (`normalizeCalendar`). Un champ absent ou invalide (une date comme le 30 février, un fuseau horaire inconnu) reprend sa valeur par défaut, tandis que les champs valides sont conservés.

Mettre à niveau les anciennes sauvegardes

Le format des plans enregistrés a évolué au cours du projet, et les anciennes sauvegardes doivent continuer à fonctionner :

- **Les anciennes sauvegardes enregistraient la progression par jour** (`{ day: 2, completedAt: ... }`). À la lecture, `normalizeWatched` recalcule ce plan avec sa propre série et ses propres réglages enregistrés, retrouve quels épisodes contenait le jour 2, et convertit l'entrée en ces épisodes, en gardant l'heure d'origine.
- **Les sauvegardes antérieures aux préférences de calendrier** reçoivent des préférences par défaut, commençant le jour de leur enregistrement, pour que leurs dates correspondent à la progression déjà faite.

Cette approche s'appelle **migration à la lecture** : les anciennes données sont mises à niveau à chaque lecture, au lieu de créer une nouvelle clé de stockage et d'abandonner les anciennes données.

Les chiffres de progression (`progress.ts`)

- **Jour de reprise** : la première journée qui n'est pas entièrement vue. Le bouton « Resume day N » et la carte « Continue watching » l'utilisent.
- **Pourcentage** : les journées entièrement vues divisées par le nombre total de journées, arrondi.
- **Série de jours** (*streak*) : combien de jours consécutifs, en remontant à partir d'aujourd'hui, vous avez coché quelque chose, tous plans confondus. Si vous n'avez encore rien regardé aujourd'hui, la série compte toujours tant que la veille compte, pour qu'elle ne retombe pas à zéro à minuit avant que vous ayez eu le temps de regarder.

9. Parler à AniList

Ce que fournit AniList

AniList est une grande base de données d'anime, tenue par sa communauté. Elle propose une **API GraphQL** publique et gratuite à une seule adresse, `https://graphql.anilist.co`. Les questions en lecture seule, comme celles que pose cette application, ne demandent ni clé ni compte. Tout ce que l'application sait d'une série vient de là : titres, épisodes, durées, affiches, genres, studios, notes, synopsis, titres d'épisodes, dates de sortie, liens de streaming et séries liées.

Les quatre questions que pose l'application

`src/api/anilist.ts` contient quatre requêtes GraphQL :

Fonction	Question	Utilisée pour
<code>searchAnime</code>	Les 12 meilleurs résultats d'une recherche	les résultats de recherche
<code>getPopularAiring</code>	Les 30 séries en diffusion les plus populaires	« Popular right now »
<code>getTrendingAnime</code>	Les 25 séries tendance du moment	les pages prérendues (chapitre 11)
<code>getAnimeById</code>	Tout sur une série	la page d'une série

Toutes excluent le contenu pour adultes (`isAdult: false`). Voici la requête de recherche :

```
query SearchAnime($search: String!) {
  Page(page: 1, perPage: 12) {
    media(search: $search, type: ANIME, isAdult: false, sort: SEARCH_MATCH) {
      ...MediaFields
    }
  }
}
```

`$search` est une **variable** : le texte que vous avez tapé est envoyé à côté de la requête au lieu d'être collé dedans. `...MediaFields` désigne un **fragment**, une liste de champs partagée par les quatre requêtes, pour que leurs réponses aient toujours la même forme.

Des listes légères, des détails complets

Le fragment partagé ne contient que des champs légers : titres, format, statut, nombre d'épisodes, durée, année, note, affiche, genres, studio principal, prochain épisode et bande-annonce. Une liste de 30 séries reste ainsi rapide à télécharger.

Les champs lourds ne sont demandés que pour une série à la fois, dans `getAnimeById` : le synopsis, les titres des épisodes (`streamingEpisodes`), les 8 prochaines sorties (`airingSchedule`), les liens externes, et les séries liées sur deux niveaux. C'est pourquoi ouvrir une série affiche d'abord ce que la carte savait déjà, puis complète les détails un instant plus tard (chapitre 5).

Envoyer une question (`request`)

Chaque question part sous forme de requête HTTP **POST** contenant du JSON : la requête et ses variables. Ensuite :

- Si AniList répond **429**, l'application a posé trop de questions. AniList accepte environ 90 requêtes par minute et indique dans un en-tête `Retry-After` combien de secondes attendre. L'application affiche « AniList rate limit hit, try again in 30s ».
- Si la réponse contient des erreurs ou aucune donnée, l'application affiche le message d'erreur d'AniList.
- Sinon, elle renvoie les données.

Pour rester loin de cette limite, une recherche ne part que quand on appuie sur Search (pas à chaque touche tapée). Et quand une nouvelle requête commence, la précédente est annulée avec un `AbortController` : une vieille réponse lente ne peut jamais remplacer une réponse plus récente.

Mettre de l'ordre dans les réponses

Les réponses d'AniList sont remises en forme avant qu'un composant ne les voie :

- La liste des studios `studios { nodes { name } }` est aplatie en une simple liste de noms.
- La couleur de l'affiche est vérifiée par `safeHexColor` (elle finit dans les styles de la page, voir le chapitre 12).
- **Où regarder** (`toWatchLinks`) : AniList liste beaucoup de liens externes. L'application ne garde que ceux de type `STREAMING` (services officiels sous licence), un par site, et uniquement des adresses `https` sûres. Elle ne propose volontairement aucune source non officielle.
- **Prochains épisodes** (`toUpcomingEpisodes`) : les entrées sans numéro d'épisode et heure entiers sont écartées, les autres sont triées.
- **Ordre de visionnage** (`buildWatchOrder`) : en partant de la série ouverte, l'application suit les liens « préquelle » et « suite » jusqu'à deux pas de distance, dans la même requête, garde les séries TV, films, spéciaux, OVA et ONA, enlève les doublons et les trie par année. Cela correspond à l'ordre de l'histoire pour presque toutes les franchises ; une série sortie dans le désordre pourrait être mal classée, c'est une limite connue. Une franchise enregistrée comme une seule longue entrée, comme One Piece, n'a simplement pas de liste.

Le piège des titres d'épisodes

Les titres des épisodes viennent des plateformes de streaming, sous forme de liste comme `"Episode 130 - Scent of Danger"`. On s'attendrait à ce que le premier titre soit l'épisode 1, le deuxième l'épisode 2, et ainsi de suite. **Ce n'est pas le cas.** Pour One Piece, la liste compte 69 entrées, dans l'ordre inverse, à partir de l'épisode 130.

L'application ne se fie donc jamais à la position dans la liste. `episodeTitleMap` lit le numéro écrit au début de chaque titre (« Episode 130 ») et range le titre sous ce numéro. Les titres sans numéro lisible sont écartés. La règle : **un titre absent vaut mieux qu'un titre faux.**

Le piège des descriptions

Le synopsis est demandé avec `description(asHtml: false)`, ce qui devrait vouloir dire « texte brut ». En pratique, AniList y laisse des balises HTML : `<br>` pour les sauts de ligne, et pour certaines séries `<i>` (italique) ou `<b>` (gras), ainsi que des codes comme `&` pour « & ». Jusqu'à cette version, l'application ne retirait que `<br>` : des séries comme Death Note, One Piece ou Jujutsu Kaisen affichaient du texte brut comme `<i>Note: ...</i>`.

La nouvelle fonction `synopsisText` corrige cela : `<br>` devient un saut de ligne, toutes les autres balises sont retirées, et les codes redeviennent des caractères, `&` en dernier pour qu'un texte qui dit vraiment `<` reste tel quel. Quatre nouveaux tests la couvrent, dont le vrai cas de Death Note.

« Popular right now », pas « cette saison »

La liste de l'accueil montre les séries **en cours de diffusion**, classées par popularité, plutôt que « les séries de la saison ». En début de saison, presque toutes les nouvelles séries n'ont aucun épisode sorti ni durée connue : une liste « de la saison » serait pleine de séries impossibles à planifier. L'application filtre aussi la liste avec le moteur lui-même (`isPlannable`) et garde 18 séries réellement planifiables : 18 remplit des rangées complètes, que la grille ait 2, 3 ou 6 colonnes. Si la requête échoue, la section disparaît simplement.

10. L'interface : thème, composants, animations et accessibilité

L'arbre des composants

La page est construite à partir de composants imbriqués. Voici l'arbre, de l'extérieur vers l'intérieur :

- **ErrorBoundary** : le filet de sécurité autour de tout.
 - **App** : l'état et le cadre de la page (en-tête, pied de page).
 - **SearchBar** : le champ de recherche de l'en-tête (sur toutes les pages sauf l'accueil).
 - **SearchStep** : la page d'accueil, ou les résultats de recherche.
 - **SearchBar** (la grande version), **SavedSchedules** (« Continue watching »), **SectionHeading**, le bouton grille/liste, et un **AnimeCard** ou un **AnimeListRow** par série ; des cases **Shimmer** pendant le chargement.
 - **ConfigureStep** : la page d'une série.
 - L'en-tête (affiche, titre, synopsis, boutons Save et bande-annonce) et la bande-annonce.
 - Le panneau latéral : **ConfigForm** (Your plan), **WatchPrefsForm** (When you watch), les liens de streaming avec **StreamingIcon**, l'ordre de visionnage, et Share & export.
 - **ScheduleView** : la date de fin, les chiffres (**CountUp**), les messages et la progression, puis la liste, ou **CalendarView** pour la grille du mois.

Petites pièces partagées : **icons.tsx** (icônes au trait), **SectionHeading** (titres de section) et **Motion.tsx** (**CountUp**, **Spinner**, **Shimmer**, **GlitchText**).

Deux couleurs de marque, et celles de chaque série

L'allure du site vient d'une planche d'inspiration construite sur deux couleurs Pantone : un bleu marine (Classic Blue, #133055) et un orange (Adobe, #BF4A1B), sur un fond crème chaleureux (#F3EDE4). Elles sont définies une seule fois, en haut de **src/index.css**, et la page d'accueil utilise exactement celles-ci.

Quand vous ouvrez une série, la page prend les couleurs de son affiche. **AniList** fournit une couleur par série (pour Frieren, #bbf1a1, un vert clair). **App.tsx** la confie à **posterTheme**, dans **src/lib/theme.ts**, qui renvoie deux couleurs. **App.tsx** les donne à la page sous forme de deux variables CSS :

- **--poster**, la **teinte** : le fond et les cartes en reçoivent un peu, pour que la page prenne la couleur de la série (un vert pâle pour Frieren, un rose pâle pour Jujutsu Kaisen).
- **--poster-accent**, l'**accent** : la couleur des boutons, des interrupteurs, des barres de progression, des cases à cocher et des mises en valeur.

Tout le reste est calculé à partir de ces deux valeurs dans **src/index.css**, avec la fonction CSS **color-mix**, qui mélange deux couleurs dans une proportion donnée :

```
--tint: var(--poster, var(--cream));
--accent: var(--poster-accent, var(--orange));
--bg: color-mix(in srgb, var(--tint) 16%, var(--cream));
--surface: color-mix(in srgb, var(--tint) 8%, #fbf8f3);
```

var(--poster, var(--cream)) veut dire « la teinte, ou le crème s'il n'y en a pas ». Sur la page d'accueil, aucune des deux variables n'est définie : la page reste donc bleu marine et orange.

Pourquoi les couleurs sont calculées : un texte lisible

Le texte des boutons est blanc, et un texte blanc sur une couleur pâle, comme le vert clair de Frieren, serait très difficile à lire. La lisibilité se mesure avec un **rapport de contraste**, de 1:1 (deux couleurs identiques) à 21:1 (du noir sur du blanc). Les règles d'accessibilité (WCAG) demandent au moins **4,5:1** pour un texte normal. Le CSS sait mélanger des couleurs mais pas mesurer un contraste : c'est donc `posterTheme` qui s'en charge :

1. Une couleur grise n'a pas de teinte à prendre : la fonction ne renvoie rien et la page garde le bleu marine et l'orange.
2. Si un texte blanc sur la couleur de l'affiche atteint déjà 4,5:1, cette couleur devient l'accent, telle quelle. Le rouge foncé de Chainsaw Man (`#6b1a1a`) est utilisé sans changement.
3. Sinon, la fonction assombrit la couleur jusqu'à ce qu'un texte blanc atteigne 4,5:1. Elle l'assombrit en **OKLCH**, une façon de décrire une couleur par sa clarté, son intensité et sa teinte : seule la clarté baisse, la couleur garde donc sa teinte. Le vert clair de Frieren devient un vert plus profond (`#518136`) au lieu d'un gris terne.

La teinte reçoit le traitement inverse : une couleur d'affiche foncée est éclaircie, pour qu'une couverture sombre donne à la page une version claire de sa couleur au lieu de la griser.

`src/lib/theme.test.ts` vérifie tout le système avec 4 096 couleurs réparties sur toutes les teintes et toutes les clartés : chaque sorte de texte doit rester lisible sur chaque fond où elle apparaît (chapitre 13).

Les jetons et les couleurs du texte

Les composants n'utilisent que ces **jetons** (`bg(--surface)` , `ring(--line)` , `text(--accent-text)` ...), jamais de couleurs fixes : toute la page suit donc automatiquement. Le texte utilise trois nuances de bleu marine : `--ink` pour les titres et le texte courant, `--ink-2` pour le texte secondaire et `--ink-3` pour les indications. Elles sont mélangées au crème fixe et restent donc les mêmes sur toutes les séries. Seule exception : le texte posé sur une image d'affiche (la note), qui reste blanc sur une étiquette bleu marine.

La mise en page et Tailwind

Le style est écrit avec des classes Tailwind directement dans les composants. Par exemple, `bg(--accent) px-5 py-2.5 text-sm font-semibold text(--on-accent)` veut dire : fond couleur d'accent, marges intérieures horizontales et verticales, petit texte en demi-gras, dans la couleur prévue pour le texte sur l'accent (le blanc). Les préfixes comme `sm:` ou `lg:` n'appliquent un style qu'à partir d'une certaine largeur d'écran ; c'est ainsi que la mise en page s'adapte :

- La grille d'affiches a **2 colonnes** sur téléphone, **3** sur tablette et **6** sur ordinateur. L'accueil affiche 18 séries, un nombre qui remplit des rangées complètes dans les trois cas.
- Sur la page d'une série, le panneau latéral et le planning s'empilent sur les petits écrans. Sur les grands écrans, ils sont côte à côte et le panneau latéral reste visible pendant qu'on fait défiler la page (il est « collant », *sticky*).

Deux détails pratiques montrent le soin apporté. Les champs de saisie utilisent au moins une police de 16 pixels sur téléphone, car l'iPhone zoome sur les champs plus petits quand on les touche. Et le cadre extérieur utilise `overflow-x-clip` plutôt que le plus courant `overflow-hidden` : ce dernier casserait sans prévenir l'en-tête et le panneau latéral « collants ».

Le projet a aussi deux règles de style :

- **Pas d'emoji ni de symboles** dans l'interface. Ils s'affichent différemment sur chaque appareil. À la place, `icons.tsx` fournit de petites icônes dessinées qui prennent la couleur du texte qui les entoure.

- **Des coins carrés partout.** Aucun composant n'utilise les classes `rounded-...` de Tailwind. Les navigateurs dessinent les cases à cocher avec des coins arrondis, quels que soient les styles : `src/index.css` les redessine donc en simples carrés, avec une coche découpée par `clip-path`. Même les icônes ont des extrémités de trait carrées.

Les animations

Les animations utilisent la bibliothèque `anime.js` et suivent des règles strictes (`src/motion/`) :

- **On n'anime que le mouvement et la transparence** : position, taille, rotation, opacité, découpage. **Jamais les couleurs**, car elles viennent des jetons du thème et doivent suivre l'affiche automatiquement.
- **Court** : moins de 600 millisecondes environ. Le but : « soigné, jamais gênant ».
- **Avant le premier affichage** : les animations démarrent dans `useLayoutEffect`, un moment de React qui a lieu avant que le navigateur n'affiche la page ; rien n'apparaît donc brièvement à sa position finale avant de s'animer.
- **Propre ensuite** : une fois finie, une animation retire les styles qu'elle avait ajoutés et ne laisse aucune trace.
- **Respectueux** : si votre appareil est réglé sur « réduire les animations » (un réglage d'accessibilité), toutes les animations sont sautées et les éléments apparaissent directement dans leur état final.

Ce qui bouge : les pages et les listes apparaissent en fondu, avec un léger décalage entre les éléments ; les chiffres défilent jusqu'à leur nouvelle valeur ; la barre de progression s'allonge ; le repère Liste/Calendrier glisse sous le bouton actif avec un effet de ressort ; les mois du calendrier glissent sur le côté ; les titres d'épisodes s'ouvrent en accordéon ; chaque bouton s'écrase légèrement quand on appuie (un seul écouteur pour toute la page, `installPressFeedback`) ; et le titre de l'en-tête joue un court effet « glitch » au survol.

L'accessibilité

L'accessibilité, c'est rendre le site utilisable par tous, y compris les personnes qui utilisent un clavier, un lecteur d'écran (un logiciel qui lit la page à voix haute) ou un texte agrandi.

- Tout ce qui est cliquable est un vrai `<button>` ou un vrai lien : cela fonctionne au clavier et est annoncé correctement.
- Les boutons à bascule indiquent s'ils sont activés (`aria-pressed`), et les boutons qui ne sont qu'une icône ont une étiquette textuelle (`aria-label`), comme « Remove Frieren from saved schedules ».
- Le calendrier est une vraie grille (`role="grid"`), et chaque case se décrit elle-même : « Thursday, October 8, Ep 1-3, watched ».
- Le champ de recherche a une étiquette invisible à l'écran mais lue par les lecteurs d'écran.
- Le focus du clavier est signalé par un anneau de couleur visible.
- Le texte garde un contraste d'au moins 4,5:1 avec son fond, quelles que soient les couleurs de la série (voir plus haut), et les tests le vérifient.
- Les images décoratives ont une description vide, pour que les lecteurs d'écran les sautent.
- La réduction des animations est respectée (voir plus haut).

Quand quelque chose tourne mal

Si une erreur inattendue survient pendant l'affichage, React laisserait normalement une page blanche. Cette version ajoute `ErrorBoundary`, un composant qui intercepte ces erreurs et affiche « Something went wrong », avec un bouton **Reload** (recharger) et un lien vers l'accueil. Vos plans enregistrés ne sont pas touchés : ils sont dans le stockage du navigateur, pas dans la page.

11. Les adresses, les moteurs de recherche et les aperçus de liens

Chaque série a sa propre adresse

La page de chaque série a une adresse lisible, par exemple :

```
anime-planner.vercel.app/anime/154587-frieren-beyond-journey-s-end
```

- `154587` est le numéro de la série dans AniList. **Seul ce numéro sert** à retrouver la série.
- `frieren-beyond-journey-s-end` est le **slug** : le titre, rendu utilisable dans une adresse par `slugify` (minuscules, accents retirés, tout ce qui n'est ni lettre ni chiffre remplacé par un tiret). Il n'est là que pour les humains et les moteurs de recherche. Si le titre change sur AniList, les anciens liens fonctionnent toujours, puisque seul le numéro compte. Un titre sans aucune lettre latine n'a simplement pas de slug : `/anime/123`.

`src/lib/route.ts` construit ces adresses (`animePath`) et les relit (`parseAnimePath`), en n'acceptant que cette forme exacte.

Changer de page sans charger une nouvelle page

L'application n'utilise pas de bibliothèque de « routage ». Ses deux vues (accueil et page d'une série) sont un simple élément d'état dans `App.tsx`. Quand vous ouvrez une série, l'application change l'adresse dans la barre du navigateur avec `history.pushState`, une fonction du navigateur qui modifie l'adresse **sans** recharger la page. Quand vous appuyez sur Précédent ou Suivant, le navigateur envoie un événement `popstate`, et l'application relit l'adresse (`resolveLocation`) pour afficher la bonne vue.

Le problème des liens directs, et sa solution

Le site est une **application monopage** : il n'existe qu'un seul vrai fichier HTML, `index.html`. Si quelqu'un ouvre directement `/anime/154587-...`, l'hébergeur cherche un fichier portant ce nom, ne trouve rien, et répondrait « 404 Not Found » (page introuvable).

La solution est une règle de **réécriture** (*rewrite*) dans `vercel.json` : toute adresse commençant par `/anime/` reçoit la page de l'application, et l'application lit ensuite l'adresse elle-même. Une réécriture n'est pas une redirection : l'adresse dans la barre du navigateur ne change pas. Un détail a coûté une correction de bug dans l'histoire du projet : avec le réglage d'adresses propres de Vercel (`cleanUrls`), la réécriture doit pointer vers `/`, et non vers `/index.html`, sinon toutes les pages de séries non prérendues renvoyaient une 404. La même règle existe dans `public/_redirects` pour les autres hébergeurs.

Ce que voient les moteurs de recherche et les aperçus de liens

Chaque page se décrit aux moteurs de recherche et aux réseaux sociaux avec des balises placées dans la partie invisible de la page, le `<head>` :

- `<title>` : le texte de l'onglet et des résultats de recherche, comme « Frieren: Beyond Journey's End watch schedule | Anime Watch Planner ».
- `description` : un court résumé (160 caractères au plus), construit à partir des informations (« 28 épisodes, 24 min each ») et du début du synopsis.

- `canonical` : l'adresse officielle de la page, pour que les moteurs de recherche ne comptent pas ses variantes comme des doublons.
- Les balises **Open Graph** (`og:...`) et **Twitter** (`twitter:...`) : le titre, la description et l'image affichés quand on colle un lien dans Discord, WhatsApp, X, Slack... L'image est la grande version de l'affiche ; le format de carte est `summary` car les affiches sont plus hautes que larges.
- **JSON-LD** : un bloc de données structurées dans le vocabulaire schema.org qui dit aux moteurs de recherche « cette page parle d'une série TV appelée..., de 28 épisodes, de ces genres ». Il ne contient volontairement aucune note : les données de liste d'AniList n'ont pas de nombre de votes, et en inventer un serait trompeur.

Tout cela est calculé par `src/lib/seo.ts`. Les mêmes fonctions servent à deux endroits : dans `App.tsx`, qui met à jour les balises chaque fois que vous ouvrez une autre série, et dans le script de construction ci-dessous. Les deux ne peuvent donc jamais se contredire.

Le prérendu : des pages prêtes à l'avance

Il y a un piège. Mettre à jour les balises avec JavaScript fonctionne pour Google, qui exécute JavaScript. Mais les robots qui fabriquent les aperçus de liens (Discord, Slack, WhatsApp...) n'exécutent pas JavaScript : ils lisent le fichier HTML brut, qui n'a que les balises de l'accueil.

La réponse est le **prérendu**, fait par `scripts/prerender.ts` juste après la construction du site :

1. Il demande à AniList les séries tendance et les séries populaires : **deux requêtes** en tout.
2. Pour chaque série planifiable, il écrit une copie d' `index.html` au nom de la série (par exemple `dist/anime/154587-frieren-beyond-journey-s-end.html`), dans laquelle seul le bloc entre les marqueurs `<!--seo-start-->` et `<!--seo-end-->` est remplacé par les balises de cette série.
3. Il écrit `sitemap.xml`, la liste de toutes ces pages pour les moteurs de recherche (`robots.txt` leur indique où la trouver).

Quand un visiteur ouvre une de ces adresses, l'hébergeur trouve le vrai fichier et le sert ; l'application se charge normalement et prend la main. Toutes les autres séries fonctionnent toujours grâce à la réécriture, avec leurs balises posées par l'application une fois JavaScript lancé. Au moment de la rédaction de ce guide, une vingtaine de séries étaient prérendues ; la liste se renouvelle à chaque déploiement. Si AniList est en panne pendant une construction, le script affiche un avertissement et le site est quand même publié, simplement sans ces pages supplémentaires.

Les anciens liens de partage

Les anciennes versions partageaient les plans avec des liens comme `/?share=eyJpZCI6...`, qui encodent le numéro de la série et vos réglages. Ils fonctionnent toujours : l'application les décode (très prudemment, voir le chapitre 12), ouvre la série avec ces réglages, et remplace l'adresse par la version propre.

12. La sécurité

Qu'est-ce qui pourrait mal tourner ?

Le site n'a ni comptes, ni serveur à lui, ni clés secrètes : beaucoup de dangers classiques n'existent donc tout simplement pas ici. Il n'y a ni base de données à voler, ni mot de passe qui pourrait fuiter. Ce qui reste, ce sont **les données venues de l'extérieur** qui arrivent dans votre navigateur :

- **Les données d'AniList** : écrites par sa communauté et modérées, mais tout de même du texte venant d'inconnus.
- **Les liens** : n'importe qui peut fabriquer un lien vers le site, y compris un ancien lien `?share=` au contenu quelconque.
- **Le stockage du navigateur** : il pourrait contenir des données abîmées ou modifiées.
- **Le fichier `.ics`** : il contient des titres de séries.

Le principal danger avec les données extérieures s'appelle **XSS** (*cross-site scripting*, injection de script) : si un texte venu de l'extérieur était traité comme du code, quelqu'un pourrait glisser un programme caché dans, par exemple, la description d'une série, et il s'exécuterait dans votre navigateur. Chaque règle ci-dessous ferme une porte.

Les règles

1. Le texte extérieur est toujours affiché comme du texte, jamais comme du HTML. Le code n'utilise jamais `dangerouslySetInnerHTML`, `innerHTML` ni `eval`, les trois moyens habituels de transformer du texte en code actif. React affiche automatiquement le texte comme du texte : même si un synopsis contenait `<script>`, vous verriez seulement ces caractères. La nouvelle fonction `synopsisText` nettoie les descriptions avant l'affichage, et son résultat reste affiché comme du simple texte.

2. Uniquement des liens `https`. Un lien peut commencer par `javascript:`, et dans ce cas, cliquer dessus exécute du code au lieu d'ouvrir une page ; React 18 ne l'empêche pas. Chaque adresse venant des données (liens de streaming, images) passe donc par `safeExternalUrl`, qui n'accepte que `https:` (et transforme `http:` en `https:`). Tout le reste est refusé.

3. Uniquement de vraies couleurs. La couleur d'affiche fournie par AniList se retrouve dans les styles de la page (`--poster` et `--poster-accent`). `safeHexColor` n'accepte qu'une vraie couleur hexadécimale comme `#bbf1a1`. Ensuite, `posterTheme` la lit sous forme de nombres et écrit des couleurs qu'elle a calculées elle-même : aucun autre texte ne peut se glisser dans les styles.

4. Les bandes-annonces sont encadrées. L'identifiant YouTube d'une bande-annonce doit faire de 6 à 20 caractères parmi les lettres, chiffres, tirets et tirets bas, pour que les données ne puissent pas orienter le cadre vidéo vers une autre adresse. La vidéo utilise `youtube-nocookie.com` (la version de YouTube plus respectueuse de la vie privée), elle tourne dans un cadre **isolé** (*sandbox*, limité dans ce qu'il peut faire), et rien n'est chargé depuis YouTube avant que vous cliquiez sur « Watch trailer ».

5. Les liens de partage sont reconstruits, jamais crus sur parole. `decodeShareParam` lit un ancien lien de partage, puis construit un objet entièrement neuf qui ne contient que les champs connus, après avoir vérifié chacun : le numéro de série doit être un entier positif, le temps quotidien compris entre 1 et 1 440 minutes (une journée entière), et le réglage « sauter » un vrai ou faux. Tout champ supplémentaire d'un lien fabriqué est ignoré, et tout ce qui est mal formé est refusé.

6. Le texte du calendrier est échappé. Dans un fichier `.ics`, un saut de ligne commence une nouvelle instruction. Si un titre contenait un saut de ligne, il pourrait ajouter de fausses instructions à votre agenda. Le code échappe les sauts de ligne, y compris le caractère « retour chariot » isolé que certaines applications d'agenda traitent comme un saut de ligne, ainsi que les virgules, points-virgules et barres obliques inverses.

7. Les données enregistrées sont vérifiées. Les plans enregistrés sont validés champ par champ à la lecture (chapitre 8), et tout ce qui est inattendu est écarté.

8. Les données structurées ne peuvent pas sortir de leur boîte. Dans les pages prérédées, le bloc JSON-LD se trouve dans une balise `<script>`. Si un titre contenait `</script>`, il pourrait fermer cette balise trop tôt. Le code écrit chaque `<` de ce bloc sous la forme `<`, qui veut dire le même caractère en JSON mais ne peut pas fermer une balise. Le texte des autres balises est échappé avec `escapeHtml`.

Les en-têtes de sécurité

L'hébergeur envoie avec chaque page des instructions supplémentaires, appelées **en-têtes** (*headers*). Ils sont définis dans `vercel.json` et, à l'identique, dans `public/_headers` pour les autres hébergeurs :

En-tête	Ce qu'il fait
Content-Security-Policy (CSP)	Le plus important : la liste de ce que la page a le droit de charger. Des scripts uniquement depuis le site lui-même ; des images uniquement depuis le site et le serveur d'images d'AniList ; des requêtes de données uniquement vers l'API d'AniList ; des cadres uniquement depuis youtube-nocookie.com. Même si quelqu'un parvenait à injecter du code, le navigateur refuserait de l'exécuter ou de le laisser envoyer des données ailleurs.
X-Frame-Options: DENY	D'autres sites ne peuvent pas afficher ce site dans un cadre. Cela empêche le clickjacking : vous piéger pour cliquer sur une copie invisible du site.
X-Content-Type-Options: nosniff	Le navigateur doit faire confiance au type de fichier déclaré et ne jamais le deviner, ce qui ferme une famille de ruses.
Referrer-Policy	Quand vous suivez un lien vers un autre site, celui-ci apprend seulement de quel site vous venez, pas l'adresse complète.
Permissions-Policy	Le site déclare qu'il n'utilise jamais la caméra, le micro, la localisation ni les paiements : rien sur la page ne peut les demander.

Une conséquence pratique : tout nouveau service extérieur doit être ajouté à la CSP, sinon il est bloqué sans bruit sur le vrai site tout en fonctionnant sur l'ordinateur du développeur (le serveur de développement n'envoie pas ces en-têtes).

Les dépendances

Le code écrit par d'autres peut aussi avoir des failles. `npm audit` compare les paquets du projet à une liste publique de problèmes connus. Dans cette version, il a signalé un problème, dans un paquet utilisé seulement pendant la construction du site (jamais envoyé aux visiteurs). Il a été corrigé avec `npm audit fix`, et l'audit signale maintenant **0 vulnérabilité**.

13. Les tests

Ce qu'est un test

Un **test** est un petit programme qui fait tourner un morceau du vrai code avec des entrées connues et vérifie la réponse. Si quelqu'un modifie plus tard le code et casse quelque chose, le test échoue et dit exactement ce qui ne va pas, avant que la modification n'atteigne qui que ce soit.

Voici un vrai test tiré de `src/lib/schedule.test.ts`. Il vérifie la règle gloutonne du chapitre 6 avec une série de 10 épisodes de 24 minutes et 90 minutes par jour :

```
it('buckets floor(budget / runtime) episodes per day, greedily', () => {
  const result = generateSchedule(anime({ episodes: 10 }), config());
  if (!result.ok) throw new Error(result.error);
  const s = result.schedule;
  expect(s.episodesPerDay).toBe(3);
  expect(s.unusedMinutesPerDay).toBe(18);
  expect(s.days.map((d) => d.episodes)).toEqual([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]);
  expect(s.days.map((d) => d.minutes)).toEqual([72, 72, 72, 24]);
  expect(s.totalMinutes).toBe(240);
  expect(s.overBudget).toBe(false);
});
```

Il se lit comme une phrase : « il répartit les épisodes par jour, de façon gloutonne : générer un planning pour 10 épisodes avec les réglages par défaut, et **s'attendre à** (*expect*) 3 épisodes par jour, 18 minutes inutilisées, les journées `[1,2,3]`, `[4,5,6]`, `[7,8,9]`, `[10]`, etc. » `anime(...)` et `config(...)` sont de petits outils définis en haut du fichier, qui fabriquent une fausse série et des réglages par défaut.

Ce qui est testé

Le projet compte **131 tests** répartis dans 11 fichiers, tous consacrés à la logique pure de `src/lib/`. Un test écrit une fois avec un tableau de cas (`it.each`) compte une fois par cas, comme Vitest les compte :

Fichier	Tests	Ce qu'il vérifie
<code>schedule.test.ts</code>	22	La durée d'un épisode avec et sans réduction (exactement 15 minutes est réduit, les formats courts et les films jamais) ; la liste des épisodes pour les séries normales, les films et les séries en cours ; la répartition en journées ; les journées qui dépassent ; les erreurs
<code>calendar.test.ts</code>	18	Uniquement de vraies dates (pas de 30 février) ; l'ajout de jours d'un mois ou d'une année à l'autre ; les jours de la semaine ; les jours sautés ; aucun décalage lors d'un changement d'heure ; une liste de jours vide ; les grilles de mois ; les dates de sortie ; les fuseaux horaires
<code>format.test.ts</code>	22	Les titres d'épisodes associés par numéro et non par position (sur le modèle des vraies données de One Piece) ; les liens et couleurs sûrs ; les identifiants YouTube ; les plages « Ep 1-3, 5 » ; les slugs ; le nettoyage des descriptions, dont le cas de Death Note
<code>storage.test.ts</code>	15	Enregistrer, remplacer, lister du plus récent au plus ancien, supprimer ; données corrompues et stockage en panne ; cocher et décocher des journées ; la mise à niveau des anciennes sauvegardes ; les préférences de calendrier par défaut et invalides
<code>progress.test.ts</code>	12	Le jour de reprise ; le pourcentage ; la progression conservée quand les journées sont redécoupées ; tous les cas de la série de jours

Fichier	Tests	Ce qu'il vérifie
<code>ics.test.ts</code>	12	Un événement par jour aux bonnes dates ; les fins de ligne ; les titres d'épisodes ; l'échappement, dont une tentative d'injecter des lignes par un titre ; les événements à heure fixe ; les rappels ; les dates de sortie, sans aucune date passée
<code>share.test.ts</code>	11	L'aller-retour des anciens liens de partage ; un texte utilisable dans une adresse ; l'unicode ; le refus des données incohérentes, des formes inattendues et des champs en trop
<code>theme.test.ts</code>	7	Les couleurs de la page d'une série : sans couleur, avec une couleur invalide ou grise, la page garde le bleu marine et l'orange ; une couleur foncée est utilisée telle quelle ; une couleur pâle est assombrie jusqu'à ce qu'un texte blanc atteigne 4,5:1, en gardant sa teinte ; les teintes foncées sont éclaircies ; et un balayage de 4 096 couleurs qui vérifie chaque texte sur chaque fond où il apparaît
<code>seo.test.ts</code>	5	Le titre, l'adresse canonique et l'image ; une description courte sans HTML ; l'échappement qui empêche les données de casser la page ; les films marqués comme films
<code>route.test.ts</code>	4	La construction et la lecture des adresses de séries ; le refus de tout le reste
<code>platform.test.ts</code>	3	La reconnaissance des iPhone et des iPad, y compris les iPad qui se font passer pour des Mac

Comment les tests sont organisés

- Les tests utilisent **Vitest** et sont placés à côté du code qu'ils testent (`schedule.ts` et `schedule.test.ts` côte à côte).
- Ils tournent dans **Node.js**, pas dans un navigateur, ce qui les rend très rapides : les 131 s'exécutent en un tiers de seconde environ. Cela ne fonctionne que parce que le code testé est pur (chapitre 4).
- Quand le code a besoin d'un élément du navigateur, le test fournit une imitation. Par exemple, `storage.test.ts` remplace `localStorage` par une simple liste en mémoire, et peut même la faire échouer exprès pour vérifier que l'application tient le coup.
- Les dates des tests sont fixes (par exemple le 15 janvier 2026) plutôt que « aujourd'hui », pour que les tests donnent le même résultat tous les jours de l'année.

Les lancer

```
$ npm test
```

```
Test Files  11 passed (11)
   Tests    131 passed (131)
  Duration  334ms
```

Ce qui n'est pas testé automatiquement

Les composants (ce que l'on voit à l'écran) n'ont pas de tests automatiques : ils ont été vérifiés à la main et avec des captures d'écran, comme celles de ce guide. C'est une lacune connue (chapitre 15). Les parties les plus sujettes aux erreurs, les calculs, les dates et la gestion des données, sont celles que les tests couvrent.

14. Du code au site : construction et mise en ligne

Construire le site (`npm run build`)

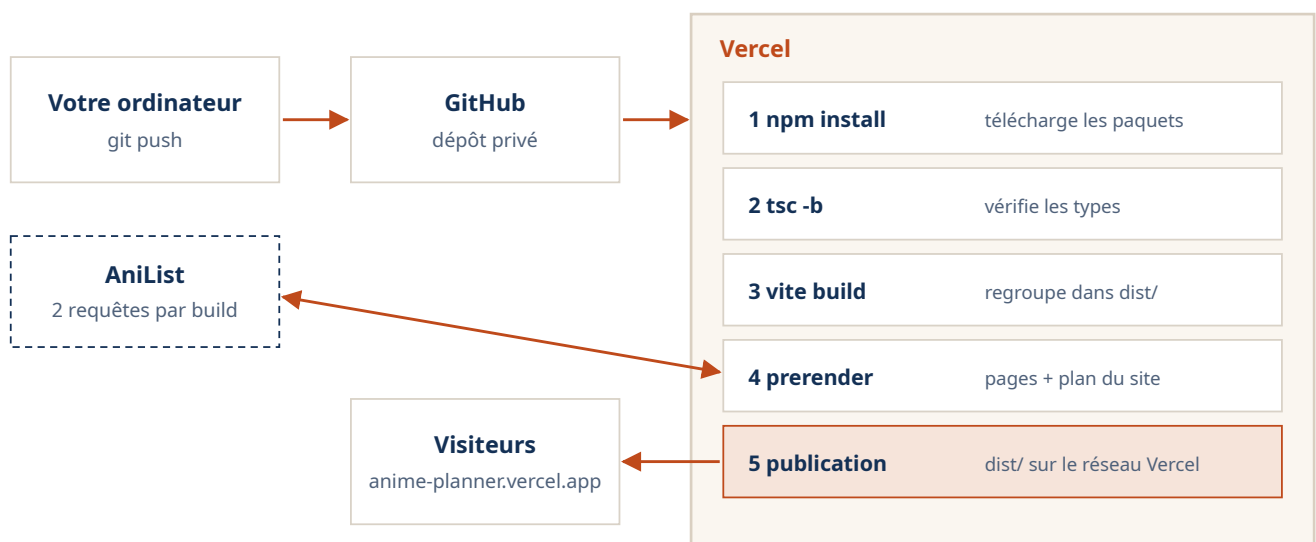
Le code que vous lisez dans l'annexe A n'est pas ce que les visiteurs téléchargent. La construction le transforme en quelques fichiers optimisés. La commande `build` de `package.json` lance trois étapes, l'une après l'autre ; si l'une échoue, la construction s'arrête :

```
tsc -b && vite build && tsx scripts/prerender.ts
```

1. `tsc -b` : TypeScript vérifie que tout le projet ne contient pas d'erreurs de types. Il ne produit aucun fichier ; il arrête seulement la construction si quelque chose ne va pas.
2. `vite build` : Vite regroupe une trentaine de fichiers sources (et les bibliothèques qu'ils utilisent) en trois : une page HTML, un fichier CSS et un fichier JavaScript, avec des noms comme `index-B2R0Yh5X.js`. La partie qui semble aléatoire change dès que le contenu change, pour que les navigateurs puissent garder les anciennes versions en cache sans risque. Le JavaScript pèse environ 260 Ko, soit environ 87 Ko une fois compressé pour le réseau.
3. `tsx scripts/prerender.ts` : écrit une page par série populaire et le plan du site (chapitre 11).

Le résultat arrive dans le dossier `dist/` : `index.html`, le dossier `assets/`, le dossier `anime/` des pages prérendues, `sitemap.xml`, et les fichiers copiés depuis `public/`.

La mise en ligne avec Vercel



Le site est hébergé par **Vercel**, et la mise en ligne est automatique :

1. Une modification est enregistrée avec git et envoyée sur GitHub (`git push`).
2. Vercel est relié au dépôt GitHub. Il remarque la nouvelle version et lance une **construction** sur ses propres ordinateurs : il installe les paquets, puis lance `npm run build` (les trois étapes ci-dessus). L'étape de prérendu demande à AniList les séries populaires.
3. Si tout réussit, Vercel publie le dossier `dist/` sur son réseau mondial de serveurs (un **CDN**), pour que le site se charge vite de partout. L'ensemble prend nettement moins d'une minute. Si la construction

échoue, la version précédente reste simplement en ligne.

`vercel.json` indique trois choses à Vercel : servir les pages sans `.html` dans l'adresse (`cleanUrls`), répondre à toute adresse `/anime/...` avec l'application (la réécriture du chapitre 11), et envoyer les en-têtes de sécurité avec chaque page (chapitre 12).

Le projet fonctionne aussi sur Netlify ou Cloudflare Pages : `public/_headers` et `public/_redirects` contiennent les mêmes règles dans leur format. Ces deux fichiers et `vercel.json` doivent toujours dire la même chose.

Comment ce guide est fabriqué

Ce guide est lui-même généré à partir du dépôt, par `npm run guide` (le dossier `guide/`) :

1. Les chapitres sont écrits en Markdown, un format de texte simple, dans `guide/fr/` et `guide/en/`. `marked` les transforme en HTML.
2. L'arborescence et le tableau des dépendances du chapitre 4 sont générés à partir des vrais fichiers.
3. L'annexe A est assemblée à partir de **chaque fichier connu de git**, dans l'ordre et avec les descriptions de `guide/files.json`. `highlight.js` colore le code, et les numéros de ligne sont ajoutés. **La fabrication refuse de se lancer si un fichier du projet manque à la liste** : l'annexe ne peut donc pas devenir obsolète sans qu'on le remarque.
4. Les captures d'écran ont été prises sur le site en ligne par `guide/screenshots.ts`, qui pilote un navigateur Chromium sans fenêtre (*headless*).
5. Chromium « imprime » ensuite la page de chaque langue en PDF, avec les numéros de page, grâce à son protocole de pilotage à distance (`guide/cdp.ts`).

15. Limites connues et choix assumés

Aucun projet ne fait tout. Ce chapitre liste ce que l'application ne fait pas volontairement, et ses limites connues, avec leurs raisons. Être clair sur les limites fait partie d'une façon honnête de construire un logiciel.

Volontairement pas fait

- **Des résumés d'épisodes écrits par une IA.** Il serait facile de demander à une IA d'écrire « ce qui se passe dans l'épisode 5 ». Mais aucune source publique (ni AniList ni MyAnimeList) ne possède de vrais résumés épisode par épisode pour la plupart des séries, et un résumé inventé d'un vrai épisode déjà sorti pourrait affirmer des choses fausses comme des faits. L'application affiche les vrais titres des épisodes, ou rien.
- **Des listes de vidéos YouTube et un lecteur de musique.** Il faudrait une clé d'API YouTube. Une telle clé est limitée à environ 100 recherches par jour, partagées entre tous les visiteurs, et serait visible par n'importe qui dans le code du site. Les bandes-annonces ne posent pas ce problème : AniList fournit directement leurs identifiants, sans clé.
- **Les conventions et événements anime.** Aucune source publique gratuite n'existe. Les seules options étaient une liste faite à la main qui deviendrait vite obsolète, ou copier les annonces d'un autre site, ce que leurs conditions d'utilisation interdisent généralement.
- **Les sites de streaming non officiels et les données d'épisodes « filler ».** « Where to watch » ne liste que des services officiels. Les données sur les épisodes de remplissage (épisodes qui ne viennent pas de l'histoire originale) devraient être copiées depuis un site de fans, avec le même problème de conditions d'utilisation.

Limites connues

- **Les plans vivent dans un seul navigateur.** Sans compte, les plans ne vous suivent pas de votre téléphone à votre ordinateur.
- **Le planning est glouton.** Les minutes restantes sont perdues chaque jour (chapitre 6). C'est une simplification volontaire.
- **Seules les séries populaires sont prérendues.** Entre 20 et 50 séries environ ont leur propre page toute prête, avec un aperçu de lien complet. Les autres séries fonctionnent normalement, mais les services qui n'exécutent pas JavaScript (Discord, Slack, WhatsApp) affichent pour elles l'aperçu générique.
- **La recherche affiche 12 résultats**, sans « page suivante ».
- **L'ordre de visionnage est trié par année**, ce qui peut être faux pour une franchise sortie dans le désordre de l'histoire.
- **Les plans des séries en cours ne font que grandir.** Si AniList corrige plus tard le nombre d'épisodes d'une série à la baisse, un ancien plan enregistré ne se met à jour qu'une fois rouvert.
- **Pas de tests automatiques de l'interface.** Les composants ont été vérifiés à la main et avec des captures d'écran.
- **L'export calendrier sur iPhone** ne peut être vraiment testé que sur un vrai iPhone.
- **Les très longs plannings** (One Piece dépasse 390 jours à 90 minutes par jour) affichent 30 jours à la fois, avec « Show more » et « Show all », pour que la page reste rapide. Tous les jours sont quand même calculés.
- **Un seul filet de sécurité** (*error boundary*) protège toute l'application. Une version plus fine pourrait garder le reste de la page fonctionnel si une partie échouait.

- **Certaines pages de séries ont des couleurs douces.** AniList fournit une seule couleur moyenne par affiche. Quand elle est pâle ou terne, comme le pêche clair de Demon Slayer, la page prend un ton discret ; quand elle est grise, la page garde le bleu marine et l'orange.

Ce qui a changé dans cette version (octobre 2026)

Ce guide décrit le projet après une relecture et quelques améliorations :

- **Corrigé** : les synopsis affichaient des balises HTML brutes comme `<i>` et `<b>` pour 6 des 40 séries les plus populaires. La nouvelle fonction `synopsisText` les nettoie (chapitre 9), avec 4 nouveaux tests.
- **Ajouté** : un filet de sécurité, pour qu'une erreur inattendue affiche une page utile au lieu d'une page blanche (chapitre 10).
- **Redessiné** : une nouvelle allure claire en deux couleurs tirées d'une planche d'inspiration, le bleu marine (Pantone Classic Blue) et l'orange (Pantone Adobe) sur fond crème, avec des coins carrés partout (chapitre 10).
- **Amélioré** : la page de chaque série prend les couleurs de son affiche grâce à `posterTheme`, qui assombrit les couleurs pâles pour que le texte blanc des boutons reste lisible (chapitre 10), avec 7 nouveaux tests.
- **Mis à jour** : un correctif de sécurité dans un paquet utilisé seulement pour la construction, et des mises à jour mineures de Vite, Vitest et de l'extension React. `npm audit` signale 0 vulnérabilité.
- **Publié** : le site à l'adresse anime-planner.vercel.app, mis en ligne automatiquement depuis GitHub.
- **Documenté** : un README pour GitHub, et ce guide.

16. Installer et lancer le projet soi-même

Ce qu'il faut

- **Node.js** en version 22 (22.12 ou plus récente), 24 ou 26. Il inclut **npm**.
- **git**, pour récupérer le code.
- Pour ce guide uniquement : le navigateur **Chromium** (ou Chrome, en donnant son chemin dans la variable `CHROME`).

Récupérer le code et le lancer

Le dépôt est privé. Si vous y avez accès :

```
git clone https://github.com/Princemitchel/anime-planner.git
cd anime-planner
npm install
npm run dev
```

`npm run dev` affiche une adresse locale, généralement `http://localhost:5173` . Ouvrez-la dans votre navigateur : c'est l'application complète, qui tourne sur votre ordinateur. Modifiez un fichier, enregistrez-le, et la page se met à jour instantanément.

Si vous n'avez que ce guide, vous avez quand même tout : l'annexe A contient chaque fichier. Recréez les mêmes dossiers et fichiers, plus un `package-lock.json` créé par `npm install` , et les commandes ci-dessus fonctionnent de la même façon.

Les autres commandes

Commande	Ce qu'elle fait
<code>npm test</code>	lance les 131 tests (chapitre 13)
<code>npm run build</code>	vérifie les types, construit le site dans <code>dist/</code> et prérend les pages des séries (chapitre 14)
<code>npm run preview</code>	sert le site construit en local, pour le vérifier avant de le publier
<code>npm run guide</code>	fabrique les deux PDF de ce guide dans <code>guide/dist/</code>
<code>npm run guide:screenshots</code>	reprend les captures d'écran de ce guide sur le site en ligne

Publier sa propre copie

1. Créez un compte Vercel gratuit et importez le dépôt GitHub, ou lancez `npx vercel` dans le dossier du projet. Vercel reconnaît tout seul un projet Vite.
2. Si votre copie obtient une autre adresse, remplacez `https://anime-planner.vercel.app` à quatre endroits : `SITE_URL` dans `src/lib/seo.ts` , les balises par défaut dans `index.html` , la ligne du plan du site dans `public/robots.txt` , et l'adresse attendue dans `src/lib/seo.test.ts` .
3. Chaque `git push` publie ensuite une nouvelle version.

Par où commencer à lire le code

Un bon ordre pour explorer l'annexe A :

1. `src/types.ts` : la forme de toutes les données.
2. `src/lib/schedule.ts` et son test : le cœur du projet, court et pur.
3. `src/lib/calendar.ts` , puis `progress.ts` et `storage.ts` .
4. `src/api/anilist.ts` : comment arrivent les données.
5. `src/App.tsx` : comment tout est relié.
6. Les composants, en commençant par `ConfigureStep.tsx` et `ScheduleView.tsx` .

Des idées pour aller plus loin

Pour s'exercer, voici des améliorations qui respectent la conception du projet :

- Ajouter un bouton « page suivante » aux résultats de recherche (AniList gère les pages).
- Ajouter des tests pour un composant, par exemple `ConfigForm` , avec une bibliothèque de test pour React.
- Proposer un planning « équilibré » qui répartit les épisodes uniformément entre les journées, en option à côté de la règle gloutonne.
- Permettre d'exporter et d'importer ses plans enregistrés sous forme de fichier, pour les transférer vers un autre navigateur.

Annexe A. Le code source complet

Cette annexe contient **tous les fichiers du projet** (60 fichiers, 6 827 lignes), dans un ordre de lecture logique : la configuration, la page et les fichiers publics, les types, la logique pure, l'accès à AniList, les animations, les composants, l'application, le script de build, les tests, puis la documentation. Chaque fichier commence par un court paragraphe qui explique son rôle ; les numéros de ligne permettent de s'y retrouver. Les commentaires dans le code sont en anglais, comme le code lui-même. Un seul fichier n'est pas imprimé : `package-lock.json`, une liste de plusieurs milliers de lignes générée automatiquement par npm (voir le chapitre 4).

A.1 Configuration du projet

`package.json`

32 lignes

La carte d'identité du projet pour npm : son nom, les commandes disponibles (`dev` , `build` , `test` , `guide` ...) et la liste des bibliothèques dont il a besoin, séparées entre celles du site (`dependencies`) et celles qui servent seulement à développer (`devDependencies`).

```
1 {
2   "name": "anime-planner",
3   "private": true,
4   "version": "0.1.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite",
8     "build": "tsc -b && vite build && tsx scripts/prerender.ts",
9     "preview": "vite preview",
10    "test": "vitest run",
11    "guide": "tsx guide/build.ts",
12    "guide:screenshots": "tsx guide/screenshots.ts"
13  },
14  "dependencies": {
15    "animejs": "^4.5.0",
16    "react": "^18.3.1",
17    "react-dom": "^18.3.1"
18  },
19  "devDependencies": {
20    "@tailwindcss/vite": "^4.3.3",
21    "@types/react": "^18.3.31",
22    "@types/react-dom": "^18.3.7",
23    "@vitejs/plugin-react": "^6.1.1",
24    "highlight.js": "^11.12.0",
25    "marked": "^18.1.0",
26    "tailwindcss": "^4.3.3",
27    "tsx": "^4.23.15",
28    "typescript": "^7.0.2",
29    "vite": "^8.3.1",
30    "vitest": "^5.0.2"
31  }
32 }
```

`tsconfig.json`

7 lignes

Le point d'entrée de la configuration TypeScript. Il ne contient que deux références : une configuration pour le code de l'application et une pour le fichier de configuration de Vite.

```
1 {
2   "files": [],
3   "references": [
4     { "path": "./tsconfig.app.json" },
5     { "path": "./tsconfig.node.json" }
6   ]
7 }
```

tsconfig.app.json

19 lignes

Les règles de TypeScript pour tout le dossier `src/` : mode strict, interdiction des variables inutilisées, JSX pour React. `noEmit` signifie que TypeScript vérifie seulement le code ; c'est Vite qui produit les fichiers finaux.

```

1  {
2    "compilerOptions": {
3      "tsBuildInfoFile": "./node_modules/.tmp/tsconfig.app.tsbuildinfo",
4      "target": "ES2022",
5      "lib": ["ES2022", "DOM", "DOM.Iterable"],
6      "module": "ESNext",
7      "moduleResolution": "bundler",
8      "jsx": "react-jsx",
9      "types": ["vite/client"],
10     "noEmit": true,
11     "isolatedModules": true,
12     "skipLibCheck": true,
13     "strict": true,
14     "noUnusedLocals": true,
15     "noUnusedParameters": true,
16     "noFallthroughCasesInSwitch": true
17   },
18   "include": ["src"]
19 }
```

tsconfig.node.json

14 lignes

Les règles de TypeScript pour `vite.config.ts`, qui tourne dans Node.js et non dans le navigateur.

```

1  {
2    "compilerOptions": {
3      "tsBuildInfoFile": "./node_modules/.tmp/tsconfig.node.tsbuildinfo",
4      "target": "ES2023",
5      "lib": ["ES2023"],
6      "module": "ESNext",
7      "moduleResolution": "bundler",
8      "noEmit": true,
9      "isolatedModules": true,
10     "skipLibCheck": true,
11     "strict": true
12   },
13   "include": ["vite.config.ts"]
14 }
```

vite.config.ts

7 lignes

La configuration de Vite, l'outil qui sert le site pendant le développement et le compile pour la production. Elle active deux extensions : React et Tailwind CSS.

```

1  import { defineConfig } from 'vite';
2  import react from '@vitejs/plugin-react';
3  import tailwindcss from '@tailwindcss/vite';
4
5  export default defineConfig({
6    plugins: [react(), tailwindcss()],
7  });
```

vercel.json

36 lignes

Les réglages de l'hébergeur Vercel : des adresses propres sans `.html`, la redirection interne des pages `/anime/...` vers l'application, et les en-têtes de sécurité envoyés avec chaque page (dont la politique de sécurité du contenu, CSP).

```

1  {
2    "cleanUrls": true,
3    "rewrites": [
```

```

4      {
5        "source": "/anime/:path*",
6        "destination": "/"
7      }
8    ],
9    "headers": [
10     {
11       "source": "/(.*)",
12       "headers": [
13         {
14           "key": "Content-Security-Policy",
15           "value": "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:
https://s4.anilist.co; connect-src https://graphql.anilist.co; frame-src https://www.youtube-nocookie.com; frame-
ancestors 'none'; base-uri 'self'; form-action 'self'; object-src 'none'"
16         },
17         {
18           "key": "X-Content-Type-Options",
19           "value": "nosniff"
20         },
21         {
22           "key": "X-Frame-Options",
23           "value": "DENY"
24         },
25         {
26           "key": "Referrer-Policy",
27           "value": "strict-origin-when-cross-origin"
28         },
29         {
30           "key": "Permissions-Policy",
31           "value": "camera=(), microphone=(), geolocation=(), payment=()"
32         }
33       ]
34     }
35   ]
36 }

```

.gitignore

6 lignes

La liste des fichiers que git doit ignorer : les bibliothèques téléchargées (`node_modules`), les fichiers générés (`dist` , les PDF du guide) et les fichiers locaux de Vercel.

```

1 node_modules
2 dist
3 *.tsbuildinfo
4 .vercel
5 .env*
6 guide/dist

```

A.2 La page et les fichiers publics

index.html

29 lignes

La seule page HTML du site. Elle est presque vide : un conteneur `<div id="root">` dans lequel React dessine toute l'application, et le bloc des balises SEO entre les marqueurs `seo-start` et `seo-end` , que le script de prérendu réécrit pour chaque série.

```

1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <meta name="theme-color" content="#f3ede4" />
7     <link rel="icon" href="/favicon.svg" type="image/svg+xml" />
8     <!-- Between these markers: the page's SEO tags. scripts/prerender.ts
9      rewrites this block per show at build time, and App.tsx keeps it
10     current while browsing. Defaults come from HOME_SEO in src/lib/seo.ts. -->
11     <!--seo-start-->
12     <title>Anime Watch Planner: plan any anime day by day</title>
13     <link rel="canonical" href="https://anime-planner.vercel.app/" />

```

```

14   <meta name="description" content="Plan any anime around the time you actually have: a day-by-day watch schedule
with real episode titles, your watch days, reminders and a calendar export." />
15   <meta property="og:site_name" content="Anime Watch Planner" />
16   <meta property="og:type" content="website" />
17   <meta property="og:title" content="Anime Watch Planner: plan any anime day by day" />
18   <meta property="og:description" content="Plan any anime around the time you actually have: a day-by-day watch
schedule with real episode titles, your watch days, reminders and a calendar export." />
19   <meta property="og:url" content="https://anime-planner.vercel.app/" />
20   <meta name="twitter:card" content="summary" />
21   <meta name="twitter:title" content="Anime Watch Planner: plan any anime day by day" />
22   <meta name="twitter:description" content="Plan any anime around the time you actually have: a day-by-day watch
schedule with real episode titles, your watch days, reminders and a calendar export." />
23   <!--seo-end-->
24 </head>
25 <body class="bg-(--cream) text-(--navy)">
26   <div id="root"></div>
27   <script type="module" src="/src/main.tsx"></script>
28 </body>
29 </html>

```

public/favicon.svg

1 lignes

La petite icône affichée dans l'onglet du navigateur : un carré orange avec « AP » en blanc, dessiné en SVG (un format d'image vectoriel, donc net à toutes les tailles).

```

1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 64 64"><rect width="64" height="64" fill="#bf4a1b"/><text
x="32" y="41" font-family="system-ui,-apple-system,Segoe UI,Roboto,sans-serif" font-size="26" font-weight="900"
text-anchor="middle" fill="ffffff">AP</text></svg>

```

public/robots.txt

4 lignes

Un fichier lu par les moteurs de recherche : il les autorise à tout explorer et leur indique où trouver le plan du site (sitemap.xml).

```

1 User-agent: *
2 Allow: /
3
4 Sitemap: https://anime-planner.vercel.app/sitemap.xml

```

public/_headers

8 lignes

Les mêmes en-têtes de sécurité que `vercel.json`, au format de Netlify et Cloudflare Pages, pour pouvoir héberger le site ailleurs que sur Vercel.

```

1 # Security headers for Netlify / Cloudflare Pages (Vercel reads vercel.json).
2 # Keep in sync with vercel.json. See CLAUDE.md "Security".
3 /*
4   Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self'
data: https://s4.anilist.co; connect-src https://graphql.anilist.co; frame-src https://www.youtube-nocookie.com;
frame-ancestors 'none'; base-uri 'self'; form-action 'self'; object-src 'none'
5   X-Content-Type-Options: nosniff
6   X-Frame-Options: DENY
7   Referrer-Policy: strict-origin-when-cross-origin
8   Permissions-Policy: camera=(), microphone=(), geolocation=(), payment=()

```

public/_redirects

5 lignes

La même redirection interne des pages `/anime/...` que `vercel.json`, au format de Netlify et Cloudflare Pages.

```

1 # Netlify / Cloudflare Pages (Vercel reads vercel.json). Keep in sync with
2 # vercel.json's rewrites. A show page that was prerendered at build time is a
3 # real file and is served first; any other /anime/<id> URL falls back to the
4 # app shell, which loads the show by id. 200 = rewrite, not redirect.
5 /anime/* /index.html 200

```

A.3 Les types partagés

src/types.ts

156 lignes

Le dictionnaire des données du projet. Il décrit la forme de chaque objet : un anime (`Anime`), un réglage de planning (`ScheduleConfig`), les préférences de calendrier (`CalendarPrefs`), un jour du planning (`ScheduleDay`), un planning sauvegardé (`SavedSchedule`)... TypeScript s'en sert pour vérifier que tout le code manipule ces objets correctement.

```
1 // Shared domain types. Change shapes here rather than inlining ad hoc types
2 // in components.
3
4 export type AnimeFormat =
5   | 'TV'
6   | 'TV_SHORT'
7   | 'MOVIE'
8   | 'SPECIAL'
9   | 'OVA'
10  | 'ONA'
11  | 'MUSIC';
12
13 export type AnimeStatus =
14   | 'FINISHED'
15   | 'RELEASING'
16   | 'NOT_YET_RELEASED'
17   | 'CANCELLED'
18   | 'HIATUS';
19
20 /** Mirrors the `MediaFields` fragment in `api/anilist.ts`. */
21 export interface Anime {
22   id: number;
23   title: {
24     romaji: string;
25     english: string | null;
26   };
27   format: AnimeFormat | null;
28   status: AnimeStatus | null;
29   /** Total episode count. `null` while a show is still airing or unannounced. */
30   episodes: number | null;
31   /** Minutes per episode (or total runtime, for movies). */
32   duration: number | null;
33   seasonYear: number | null;
34   averageScore: number | null;
35   coverImage: {
36     large: string | null;
37     /** Higher-resolution cover, used for link previews. Absent on old saved snapshots. */
38     extraLarge?: string | null;
39     color: string | null;
40   };
41   genres: string[];
42   studios: { name: string }[];
43   nextAiringEpisode: { episode: number; airingAt: number } | null;
44   trailer: { id: string; site: string } | null;
45
46   // The following are only populated by `getAnimeById`, not search/trending
47   // results — undefined there, never fetched for list views (too heavy).
48   description?: string | null;
49   streamingEpisodes?: { title: string }[];
50   /** Chronological watch order (see `buildWatchOrder` in `api/anilist.ts`). */
51   relations?: { id: number; title: string; seasonYear: number | null; isCurrent: boolean }[];
52   /** Official streaming services from AniList (`externalLinks`, STREAMING only). */
53   watchLinks?: WatchLink[];
54   /** Next few not-yet-aired episodes, soonest first (`airingSchedule`). */
55   upcomingEpisodes?: AiringEpisode[];
56 }
57
58 export interface AiringEpisode {
59   episode: number;
60   /** Unix seconds, as Anilist returns it. */
61   airingAt: number;
62 }
63
64 export interface WatchLink {
65   site: string;
```

```

66 url: string;
67 /** Brand color from AniList, e.g. "#F88B24". */
68 color: string | null;
69 }
70
71 export interface ScheduleConfig {
72 /** Minutes the user is willing to watch per day. */
73 dailyBudget: number;
74 skipOpEd: boolean;
75 }
76
77 /**
78 * When the user watches: turns the engine's "Day N" into real dates, and
79 * drives the calendar view and `.ics` export. Separate from
80 * `ScheduleConfig`, which only decides how episodes are split into days.
81 */
82 export interface CalendarPrefs {
83 /** Local date, `yyyy-mm-dd`. Day 1 is the first watch day on or after it. */
84 startDate: string;
85 /** Weekdays the user watches, 0 = Sunday ... 6 = Saturday. Never empty. */
86 watchDays: number[];
87 /** Local `HH:MM`, or `null` for "any time" (all-day calendar events). */
88 watchTime: string | null;
89 /** Minutes before `watchTime` to remind, or `null` for none. */
90 reminderMinutes: number | null;
91 /**
92 * IANA zone that "today" and air times are shown in, or `null` for the
93 * device's own. Display/planning only: the `.ics` export stays floating.
94 */
95 timezone: string | null;
96 }
97
98 export interface ScheduleDay {
99 /** 1-based day index. */
100 day: number;
101 episodes: number[];
102 minutes: number;
103 }
104
105 export interface Schedule {
106 days: ScheduleDay[];
107 /** Per-episode runtime after the optional OP/ED trim. */
108 netRuntime: number;
109 episodesPerDay: number;
110 episodeCount: number;
111 totalMinutes: number;
112 /** Minutes left over on each full day (see greedy note in schedule.ts). */
113 unusedMinutesPerDay: number;
114 /** True when one episode alone exceeds the daily budget. */
115 overBudget: boolean;
116 /** True when `episodeCount` came from "aired so far", not a real total. */
117 isPartial: boolean;
118 }
119
120 export type ScheduleError =
121 | 'invalid-budget'
122 | 'unknown-episode-count'
123 | 'unknown-duration';
124
125 export type ScheduleResult =
126 | { ok: true; schedule: Schedule }
127 | { ok: false; error: ScheduleError };
128
129 export interface WatchedEpisode {
130 episode: number;
131 /** Epoch ms. */
132 completedAt: number;
133 }
134
135 /** A schedule the user saved; persisted by `lib/storage.ts`. */
136 export interface SavedSchedule {
137 id: string;
138 /** Epoch ms. */
139 savedAt: number;
140 /** Snapshot at save time, so it renders without refetching. */
141 anime: Anime;
142 config: ScheduleConfig;

```

```

143  /** Filled with defaults on read for records saved before it existed. */
144  calendar: CalendarPrefs;
145  /**
146   * Episodes marked watched, each with when. Per-episode (not per-day) so
147   * changing the daily budget doesn't move progress onto other episodes.
148   * Older records stored `{ day }` entries; `getSavedSchedules` migrates them.
149   */
150  watched: WatchedEpisode[];
151  }
152
153  /** Whether the open anime + config matches what's saved for it. */
154  export type SaveStatus = 'unsaved' | 'saved' | 'changed' | 'failed';
155
156  export type Step = 'search' | 'configure';

```

A.4 La logique pure (src/lib)

src/lib/schedule.ts

101 lignes

Le moteur de calcul, le cœur du projet. À partir d'un anime et d'un temps quotidien, il calcule la durée nette d'un épisode, la liste des épisodes à regarder et leur répartition jour par jour. Il est « pur » : il ne fait que des calculs, sans réseau ni affichage, ce qui le rend facile à tester.

```

1  // Pure calculation engine: inputs in, plain data out. No fetch, no window,
2  // no React. Keep it that way so it stays unit-testable.
3
4  import type { Anime, ScheduleConfig, ScheduleDay, ScheduleResult } from '../types';
5
6  /** Minutes trimmed per episode when "skip OP/ED" is on (~1.5 min each). */
7  export const OP_ED_TRIM_MINUTES = 3;
8
9  /** Episodes shorter than this are treated as shorts and never trimmed. */
10 export const MIN_DURATION_FOR_TRIM = 15;
11
12 /**
13  * Per-episode runtime in minutes: `duration - (skipOpEd ? 3 : 0)`.
14  * The 3-minute trim applies only when `format !== 'MOVIE'` AND
15  * `duration >= 15`; shorts and movies are never trimmed.
16  * Returns `null` when Anilist has no duration for the show.
17  */
18 export function getNetEpisodeRuntime(anime: Anime, config: ScheduleConfig): number | null {
19   const { duration, format } = anime;
20   if (duration == null || duration <= 0) return null;
21
22   const trimmable = format !== 'MOVIE' && duration >= MIN_DURATION_FOR_TRIM;
23   return config.skipOpEd && trimmable ? duration - OP_ED_TRIM_MINUTES : duration;
24 }
25
26 /**
27  * Every episode number, 1..N. Movies always return `[1]`.
28  *
29  * When Anilist has no published total (`episodes: null` – typically a
30  * still-airing show), falls back to "episodes aired so far"
31  * (`nextAiringEpisode.episode - 1`) rather than giving up: an ongoing show
32  * like One Piece shouldn't be a dead end just because its final episode
33  * count isn't known yet. `isFallback` on the result flags this case so the
34  * caller can label the schedule as partial. Returns `null` only when
35  * there's truly nothing to schedule (no total, not airing, or airing but
36  * nothing has aired yet).
37  */
38 export function getWatchableEpisodes(
39   anime: Anime,
40 ): { episodes: number[]; isFallback: boolean } | null {
41   if (anime.format === 'MOVIE') return { episodes: [1], isFallback: false };
42
43   if (anime.episodes != null && anime.episodes > 0) {
44     return { episodes: Array.from({ length: anime.episodes }, (_, i) => i + 1), isFallback: false };
45   }
46
47   const airedSoFar = anime.nextAiringEpisode ? anime.nextAiringEpisode.episode - 1 : 0;
48   if (airedSoFar > 0) {
49     return { episodes: Array.from({ length: airedSoFar }, (_, i) => i + 1), isFallback: true };

```

```

50     }
51
52     return null;
53 }
54
55 /**
56  * Buckets episodes into days, greedily, at
57  * `floor(dailyBudget / netRuntime)` episodes per day.
58  *
59  * This deliberately wastes leftover minutes each day (90-min budget with
60  * 24-min episodes -> 3/day, 18 min unused). It's a known simplification,
61  * not a bug.
62  *
63  * If one episode is longer than the whole budget, it schedules 1 episode
64  * per day and sets `overBudget` instead of returning an empty schedule.
65  */
66 export function generateSchedule(anime: Anime, config: ScheduleConfig): ScheduleResult {
67     if (!Number.isFinite(config.dailyBudget) || config.dailyBudget <= 0) {
68         return { ok: false, error: 'invalid-budget' };
69     }
70
71     const netRuntime = getNetEpisodeRuntime(anime, config);
72     if (netRuntime == null) return { ok: false, error: 'unknown-duration' };
73
74     const watchable = getWatchableEpisodes(anime);
75     if (watchable == null) return { ok: false, error: 'unknown-episode-count' };
76     const { episodes, isFallback } = watchable;
77
78     const fitsPerDay = Math.floor(config.dailyBudget / netRuntime);
79     const overBudget = fitsPerDay === 0;
80     const episodesPerDay = Math.max(1, fitsPerDay);
81
82     const days: ScheduleDay[] = [];
83     for (let i = 0; i < episodes.length; i += episodesPerDay) {
84         const chunk = episodes.slice(i, i + episodesPerDay);
85         days.push({ day: days.length + 1, episodes: chunk, minutes: chunk.length * netRuntime });
86     }
87
88     return {
89         ok: true,
90         schedule: {
91             days,
92             netRuntime,
93             episodesPerDay,
94             episodeCount: episodes.length,
95             totalMinutes: episodes.length * netRuntime,
96             unusedMinutesPerDay: overBudget ? 0 : config.dailyBudget - episodesPerDay * netRuntime,
97             overBudget,
98             isPartial: isFallback,
99         },
100     };
101 }

```

src/lib/format.ts

140 lignes

De petites fonctions d'affichage et de sécurité : choisir le titre à montrer, écrire « 2h 15m », nettoyer la description d'AniList (`synopsisText`), associer les titres d'épisodes à leur numéro, n'accepter que des liens `https:` et des couleurs valides, fabriquer l'adresse d'une bande-annonce YouTube.

```

1  import type { Anime, AnimeFormat } from '../types';
2
3  export function displayTitle(anime: Anime): string {
4      return anime.title.english ?? anime.title.romaji;
5  }
6
7  /** 135 -> "2h 15m", 45 -> "45m". */
8  export function formatMinutes(minutes: number): string {
9      const h = Math.floor(minutes / 60);
10     const m = minutes % 60;
11     if (h === 0) return `${m}m`;
12     return m === 0 ? `${h}h` : `${h}h ${m}m`;
13 }
14
15 const FORMAT_LABELS: Record<AnimeFormat, string> = {

```

```

16   TV: 'TV',
17   TV_SHORT: 'TV Short',
18   MOVIE: 'Movie',
19   SPECIAL: 'Special',
20   OVA: 'OVA',
21   ONA: 'ONA',
22   MUSIC: 'Music',
23 };
24
25 export function formatLabel(format: AnimeFormat | null): string {
26   return format ? FORMAT_LABELS[format] : 'Unknown';
27 }
28
29 /**
30  * Anilist description as display text, keeping its paragraph breaks. Even
31  * with `asHtml: false` the text still has `  
` tags, and some shows also
32  * carry ``/`` (Death Note, One Piece, ...). `  
` becomes a newline,
33  * every other tag is dropped, and the few entities Anilist uses are decoded,
34  * `&` last so `&lt;` stays `<`. The result is rendered as text,
35  * never as HTML.
36  */
37 export function synopsisText(html: string): string {
38   return html
39     .replace(/<br\s*\/?>/gi, '\n')
40     .replace(/<[^\>]*>/g, '')
41     .replace(/&lt;/g, '<')
42     .replace(/&gt;/g, '>')
43     .replace(/&quot;/g, '"')
44     .replace(/&#0?39;/g, "'")
45     .replace(/&/g, '&')
46     .replace(/\t\s+/g, '\n')
47     .replace(/<n{3,}/g, '\n\n')
48     .trim();
49 }
50
51 const EPISODE_PREFIX = /^episode\s*(\d+)\s*[-:]?\s*/i;
52
53 /**
54  * Episode number -> title, keyed by the number parsed from each title's
55  * "Episode N" prefix. Anilist's `streamingEpisodes` can't be read by
56  * index: for One Piece it's 69 entries in reverse order starting at 130.
57  * Titles without a parseable number are dropped rather than guessed.
58  */
59 export function episodeTitleMap(list: { title: string }[] | undefined): Map<number, string> {
60   const map = new Map<number, string>();
61   for (const { title } of list ?? []) {
62     const match = EPISODE_PREFIX.exec(title);
63     if (!match) continue;
64     const n = Number(match[1]);
65     const rest = title.slice(match[0].length).trim();
66     if (!map.has(n) && rest) map.set(n, rest);
67   }
68   return map;
69 }
70
71 /** Days until an Anilist `airingAt` unix-seconds timestamp, rounded up. */
72 export function daysUntil(airingAtSeconds: number): number {
73   return Math.max(0, Math.ceil((airingAtSeconds * 1000 - Date.now()) / 86_400_000));
74 }
75
76
77 /**
78  * Only lets `https:` URLs through (upgrading `http:`). Anilist link data is
79  * user-submitted, and React 18 doesn't block `javascript:` hrefs, so every
80  * external link from API data goes through here before it reaches an href.
81  */
82 export function safeExternalUrl(url: string): string | null {
83   try {
84     const parsed = new URL(url);
85     if (parsed.protocol === 'http:') parsed.protocol = 'https:';
86     return parsed.protocol === 'https:' ? parsed.href : null;
87   } catch {
88     return null;
89   }
90 }
91
92 const HEX_COLOR = /^(?:[0-9a-f]{3}|[0-9a-f]{6})$/i;

```

```

93
94 /** A CSS hex color, or null. Keeps API strings out of style values as-is. */
95 export function safeHexColor(color: string | null | undefined): string | null {
96   return color && HEX_COLOR.test(color) ? color : null;
97 }
98
99 const YOUTUBE_ID = /^[A-Za-z0-9_-]{6,20}$/;
100
101 /**
102  * Privacy-friendly embed URL (youtube-nocookie.com) for an AniList trailer,
103  * or null when it isn't a YouTube trailer or the id looks malformed. The id
104  * is checked so API data can't steer the iframe to another path.
105  */
106 export function youtubeEmbedUrl(trailer: Anime['trailer']): string | null {
107   if (!trailer || trailer.site !== 'youtube' || !YOUTUBE_ID.test(trailer.id)) return null;
108   return `https://www.youtube-nocookie.com/embed/${trailer.id}?rel=0`;
109 }
110
111 /** [1,2,3,5] -> "Ep 1-3, 5" */
112 export function describeEpisodes(episodes: number[]): string {
113   const ranges: string[] = [];
114   let start = episodes[0];
115   let prev = start;
116   for (const ep of [...episodes.slice(1), NaN]) {
117     if (ep === prev + 1) {
118       prev = ep;
119       continue;
120     }
121     ranges.push(start === prev ? `${start}` : `${start}-${prev}`);
122     start = prev = ep;
123   }
124   return `Ep ${ranges.join(', ')}`;
125 }
126
127 /** "Re:ZERO -Starting Life-" -> "re-zero-starting-life". Accents are folded; may return "". */
128 export function slugify(title: string): string {
129   return title
130     .normalize('NFKD')
131     .replace(/[\u0300-\u036f]/g, '')
132     .toLowerCase()
133     .replace(/^[a-z0-9]+/g, '-')
134     .replace(/^-+|-+$/g, '');
135 }
136
137 /** AniList description as one line of plain text (tags dropped, whitespace collapsed). */
138 export function plainText(html: string): string {
139   return synopsisText(html).replace(/\s+/g, ' ').trim();
140 }

```

src/lib/theme.ts

136 lignes

Les couleurs de la page d'une série. À partir de la couleur d'affiche fournie par AniList, `posterTheme` choisit une couleur de boutons sur laquelle un texte blanc reste lisible (assombrie en OKLCH si besoin, sans changer de teinte) et une teinte claire pour le fond. Une affiche grise garde le bleu marine et l'orange du site.

```

1 // Pure color math for the per-show theme. No fetch, no window, no React -
2 // same rules as schedule.ts.
3 //
4 // A show page takes its colors from the poster (AniList `coverImage.color`).
5 // CSS can mix colors but can't check contrast, so the accent is picked here:
6 // the poster color itself, or, when white text on it would fall under
7 // 4.5:1, the same hue darkened until it passes. Darkening happens in OKLCH
8 // (lightness down, hue and chroma kept where sRGB allows), so a pale cover
9 // gives a deeper version of its own color instead of a muddy one.
10 // index.css derives every other token from these two values.
11
12 export interface PosterTheme {
13   /**
14    * The poster color, lightened when it's dark: index.css tints the
15    * background and surfaces with it, and a dark tint would gray the page.
16    */
17   tint: string;
18   /** Poster-colored accent that white text can sit on (4.5:1 or more). */
19   accent: string;

```

```

20 }
21
22 /** Covers below this OKLCH chroma read as gray, so they keep the brand colors. */
23 const MIN_CHROMA = 0.04;
24
25 /**
26  * The tint keeps at least this OKLCH lightness, so a dark cover tints the
27  * page with its hue, not its darkness.
28  */
29 const TINT_MIN_LIGHTNESS = 0.75;
30
31 /** A little over 4.5:1, so rounding to 8-bit hex can't drop below it. */
32 const TARGET_CONTRAST = 4.6;
33
34 /** Gamma-encoded sRGB channels, 0-1. */
35 type Rgb = [number, number, number];
36
37 function parseHex(hex: string): Rgb | null {
38   const match = /^#([0-9a-f]{3}|[0-9a-f]{6})$/i.exec(hex);
39   if (!match) return null;
40   const full = match[1].length === 3 ? [...match[1]].map((c) => c + c).join('') : match[1];
41   return [0, 2, 4].map((i) => parseInt(full.slice(i, i + 2), 16) / 255) as Rgb;
42 }
43
44 function toHex(rgb: Rgb): string {
45   return `#${rgb.map((c) => Math.round(c * 255).toString(16).padStart(2, '0')).join('')}`;
46 }
47
48 const toLinear = (c: number) => (c <= 0.04045 ? c / 12.92 : ((c + 0.055) / 1.055) ** 2.4);
49 const fromLinear = (c: number) => (c <= 0.0031308 ? 12.92 * c : 1.055 * c ** (1 / 2.4) - 0.055);
50
51 /** WCAG relative luminance. */
52 function luminance(rgb: Rgb): number {
53   const [r, g, b] = rgb.map(toLinear);
54   return 0.2126 * r + 0.7152 * g + 0.0722 * b;
55 }
56
57 /** WCAG contrast ratio (1-21) between two hex colors; 1 if either is invalid. */
58 export function contrastRatio(a: string, b: string): number {
59   const ca = parseHex(a);
60   const cb = parseHex(b);
61   if (!ca || !cb) return 1;
62   const [hi, lo] = [luminance(ca), luminance(cb)].sort((x, y) => y - x);
63   return (hi + 0.05) / (lo + 0.05);
64 }
65
66 // OKLab matrices from Björn Ottosson, https://bottosson.github.io/posts/oklab/
67
68 /** sRGB -> OKLCH: [lightness 0-1, chroma, hue in radians]. */
69 function toOkLch(rgb: Rgb): [number, number, number] {
70   const [r, g, b] = rgb.map(toLinear);
71   const l = Math.cbrt(0.4122214708 * r + 0.5363325363 * g + 0.0514459929 * b);
72   const m = Math.cbrt(0.2119034982 * r + 0.6806995451 * g + 0.1073969566 * b);
73   const s = Math.cbrt(0.0883024619 * r + 0.2817188376 * g + 0.6299787005 * b);
74   const L = 0.2104542553 * l + 0.793617785 * m - 0.0040720468 * s;
75   const A = 1.9779984951 * l - 2.428592205 * m + 0.4505937099 * s;
76   const B = 0.0259040371 * l + 0.7827717662 * m - 0.0808675766 * s;
77   return [L, Math.hypot(A, B), Math.atan2(B, A)];
78 }
79
80 /** OKLCH -> linear sRGB, unclamped: a channel outside 0-1 means out of gamut. */
81 function okLchToLinear(L: number, C: number, h: number): Rgb {
82   const A = C * Math.cos(h);
83   const B = C * Math.sin(h);
84   const l = (L + 0.3963377774 * A + 0.2158037573 * B) ** 3;
85   const m = (L - 0.1055613458 * A - 0.0638541728 * B) ** 3;
86   const s = (L - 0.0894841775 * A - 1.291485548 * B) ** 3;
87   return [
88     4.0767416621 * l - 3.3077115913 * m + 0.2309699292 * s,
89     -1.2684380046 * l + 2.6097574011 * m - 0.3413193965 * s,
90     -0.0041960863 * l - 0.7034186147 * m + 1.707614701 * s,
91   ];
92 }
93
94 const inGamut = (rgb: Rgb) => rgb.every((c) => c >= -1e-7 && c <= 1 + 1e-7);
95
96 /** OKLCH -> sRGB, lowering chroma (lightness and hue kept) until it fits. */

```

```

97 function fromOklch(L: number, C: number, h: number): Rgb {
98   let chroma = C;
99   if (!inGamut(oklchToLinear(L, C, h))) {
100     let lo = 0;
101     let hi = C;
102     for (let i = 0; i < 20; i++) {
103       const mid = (lo + hi) / 2;
104       if (inGamut(oklchToLinear(L, mid, h))) lo = mid;
105       else hi = mid;
106     }
107     chroma = lo;
108   }
109   return oklchToLinear(L, chroma, h).map((c) => fromLinear(Math.min(1, Math.max(0, c)))) as Rgb;
110 }
111
112 /**
113  * The show page's colors for a poster color, or null to keep the brand
114  * navy and orange: no color, an invalid one, or a gray cover (no hue to
115  * take on).
116  */
117 export function posterTheme(color: string | null | undefined): PosterTheme | null {
118   const rgb = color ? parseHex(color) : null;
119   if (!rgb) return null;
120   const [L, C, h] = toOklch(rgb);
121   if (C < MIN_CHROMA) return null;
122
123   const tint = toHex(L >= TINT_MIN_LIGHTNESS ? rgb : fromOklch(TINT_MIN_LIGHTNESS, C, h));
124   const passes = (c: Rgb) => 1.05 / (luminance(c) + 0.05) >= TARGET_CONTRAST;
125   if (passes(rgb)) return { tint, accent: toHex(rgb) };
126
127   // Highest lightness that passes. Black (L = 0) always does.
128   let lo = 0;
129   let hi = L;
130   for (let i = 0; i < 20; i++) {
131     const mid = (lo + hi) / 2;
132     if (passes(fromOklch(mid, C, h))) lo = mid;
133     else hi = mid;
134   }
135   return { tint, accent: toHex(fromOklch(lo, C, h)) };
136 }

```

src/lib/calendar.ts

239 lignes

La couche des dates : elle transforme le « Jour 3 » du moteur en une vraie date selon la date de début et les jours de visionnage choisis, construit les grilles de mois du calendrier, gère les fuseaux horaires et les dates de diffusion des prochains épisodes.

```

1 // Pure date layer: maps the engine's "Day N" onto real calendar dates using
2 // the user's CalendarPrefs, and builds month grids for the calendar view.
3 // No fetch, no window, no React.
4 //
5 // Calendar dates are `yyyy-mm-dd` strings, and day arithmetic runs in UTC
6 // so a daylight-saving change can never shift or skip a day. Only
7 // `localIso` (turning a real instant into the user's local date) and
8 // `formatAiring` use a time zone, on purpose: the device's own, or the one
9 // picked in `CalendarPrefs.timezone` (`null` = device).
10
11 import type { AiringEpisode, Anime, CalendarPrefs, ScheduleDay } from '../types';
12
13 export const ALL_DAYS = [0, 1, 2, 3, 4, 5, 6];
14 export const WEEKDAYS = [1, 2, 3, 4, 5];
15 export const WEEKENDS = [0, 6];
16 /** Allowed reminder offsets in minutes; `null` means no reminder. */
17 export const REMINDER_OPTIONS: (number | null)[] = [null, 0, 10, 30, 60];
18
19 const ISO_DATE = /^(d{4})-(d{2})-(d{2})$/;
20 const HH_MM = /^([01]d|2[0-3]):([0-5]d)$/;
21 const DAY_MS = 86_400_000;
22
23 function pad(n: number): string {
24   return String(n).padStart(2, '0');
25 }
26
27 function toUtcMs(iso: string): number {

```

```

28     const [y, m, d] = iso.split('-').map(Number);
29     return Date.UTC(y, m - 1, d);
30 }
31
32 function fromUtcMs(ms: number): string {
33     const date = new Date(ms);
34     return `${date.getUTCFullYear()}-${pad(date.getUTCMonth() + 1)}-${pad(date.getUTCDate())}`;
35 }
36
37 /** A real calendar date in `yyyy-mm-dd` form (rejects 2026-02-30). */
38 export function isIsoDate(value: unknown): value is string {
39     if (typeof value !== 'string' || !ISO_DATE.test(value)) return false;
40     return fromUtcMs(toUtcMs(value)) === value;
41 }
42
43 export function isWatchTime(value: unknown): value is string {
44     return typeof value === 'string' && HH_MM.test(value);
45 }
46
47 export function addDays(iso: string, n: number): string {
48     return fromUtcMs(toUtcMs(iso) + n * DAY_MS);
49 }
50
51 /** 0 = Sunday ... 6 = Saturday. */
52 export function weekday(iso: string): number {
53     return new Date(toUtcMs(iso)).getUTCDay();
54 }
55
56 /** Whether `zone` is an IANA time zone this runtime understands. */
57 export function isTimeZone(zone: unknown): zone is string {
58     if (typeof zone !== 'string' || zone === '') return false;
59     try {
60         new Intl.DateTimeFormat('en-US', { timeZone: zone });
61         return true;
62     } catch {
63         return false;
64     }
65 }
66
67 /**
68  * The calendar date of an instant (e.g. an air time or "now") in `timezone`,
69  * or on the device's clock when it's null.
70  */
71 export function localIso(date: Date, timezone?: string | null): string {
72     if (!timezone) return `${date.getFullYear()}-${pad(date.getMonth() + 1)}-${pad(date.getDate())}`;
73     // en-CA formats dates as yyyy-mm-dd.
74     return new Intl.DateTimeFormat('en-CA', { timeZone: timezone, year: 'numeric', month: '2-digit', day: '2-digit'
75 }).format(date);
76 }
77
78 export function todayIso(now: Date = new Date(), timezone?: string | null): string {
79     return localIso(now, timezone);
80 }
81
82 /** The zone the device is set to, e.g. "Europe/Paris" (or null if the runtime won't say). */
83 export function deviceTimeZone(): string | null {
84     const zone = new Intl.DateTimeFormat().resolvedOptions().timeZone;
85     return isTimeZone(zone) ? zone : null;
86 }
87
88 const FALLBACK_ZONES = [
89     'UTC', 'America/Los_Angeles', 'America/Denver', 'America/Chicago', 'America/New_York',
90     'America/Toronto', 'America/Mexico_City', 'America/Bogota', 'America/Lima', 'America/Sao_Paulo',
91     'America/Argentina/Buenos_Aires', 'Europe/London', 'Europe/Lisbon', 'Europe/Paris', 'Europe/Berlin',
92     'Europe/Madrid', 'Europe/Rome', 'Europe/Warsaw', 'Europe/Istanbul', 'Europe/Moscow', 'Africa/Lagos',
93     'Africa/Cairo', 'Africa/Johannesburg', 'Asia/Dubai', 'Asia/Kolkata', 'Asia/Bangkok', 'Asia/Jakarta',
94     'Asia/Manila', 'Asia/Singapore', 'Asia/Shanghai', 'Asia/Seoul', 'Asia/Tokyo', 'Australia/Perth',
95     'Australia/Sydney', 'Pacific/Auckland',
96 ];
97
98 let zones: string[] | null = null;
99
100 /** Every IANA zone the runtime knows, falling back to a list of major ones. */
101 export function listTimeZones(): string[] {
102     if (zones) return zones;
103     try {
104         const all = (Intl as { supportedValuesOf?: (key: string) => string[] }).supportedValuesOf?.('timeZone');
105         zones = all && all.length > 0 ? all : FALLBACK_ZONES;
106     }

```

```

104     } catch {
105         zones = FALLBACK_ZONES;
106     }
107     return zones;
108 }
109
110 export function defaultCalendarPrefs(startDate: string = todayIso()): CalendarPrefs {
111     return { startDate, watchDays: [...ALL_DAYS], watchTime: null, reminderMinutes: null, timezone: null };
112 }
113
114 /**
115  * Day number -> date. Day 1 lands on the first watch day on or after
116  * `startDate`; each later day on the next watch day after the previous one.
117  * An empty `watchDays` is treated as every day rather than looping forever.
118  */
119 export function scheduleDates(days: ScheduleDay[], prefs: CalendarPrefs): Map<number, string> {
120     const allowed = new Set(prefs.watchDays.length > 0 ? prefs.watchDays : ALL_DAYS);
121     const dates = new Map<number, string>();
122     let date = prefs.startDate;
123     for (const day of days) {
124         while (!allowed.has(weekday(date))) date = addDays(date, 1);
125         dates.set(day.day, date);
126         date = addDays(date, 1);
127     }
128     return dates;
129 }
130
131 export function finishDate(days: ScheduleDay[], prefs: CalendarPrefs): string | null {
132     if (days.length === 0) return null;
133     return scheduleDates(days, prefs).get(days[days.length - 1].day) ?? null;
134 }
135
136 export interface Month {
137     year: number;
138     /** 0-11, like `Date#getMonth`. */
139     month: number;
140 }
141
142 export function monthOf(iso: string): Month {
143     const [y, m] = iso.split('-').map(Number);
144     return { year: y, month: m - 1 };
145 }
146
147 export function shiftMonth({ year, month }: Month, delta: number): Month {
148     const total = year * 12 + month + delta;
149     return { year: Math.floor(total / 12), month: ((total % 12) + 12) % 12 };
150 }
151
152 export interface MonthCell {
153     iso: string;
154     /** False for the leading/trailing days borrowed from nearby months. */
155     inMonth: boolean;
156 }
157
158 /** Full Monday-first weeks covering the month, for a 7-column grid. */
159 export function monthGrid({ year, month }: Month): MonthCell[][] {
160     const first = `${year}-${pad(month + 1)}-01`;
161     const start = addDays(first, -((weekday(first) + 6) % 7));
162     const weeks: MonthCell[][] = [];
163     let date = start;
164     do {
165         const week: MonthCell[] = [];
166         for (let i = 0; i < 7; i++) {
167             week.push({ iso: date, inMonth: monthOf(date).month === month });
168             date = addDays(date, 1);
169         }
170         weeks.push(week);
171     } while (monthOf(date).month === month);
172     return weeks;
173 }
174
175 /** "Sat, Oct 3" by default. Formats the date itself, never shifted by time zone. */
176 export function formatDate(
177     iso: string,
178     options: Intl.DateTimeFormatOptions = { weekday: 'short', month: 'short', day: 'numeric' },
179     locale?: string,
180 ): string {

```

```

181     return new Intl.DateTimeFormat(locale, { ...options, timeZone: 'UTC' }).format(
182       new Date(toUtcMs(iso)),
183     );
184   }
185
186   /** "20:00" -> "8:00 PM" (or "20:00", depending on locale). */
187   export function formatWatchTime(hhmm: string, locale?: string): string {
188     const [h, m] = hhmm.split(':').map(Number);
189     return new Intl.DateTimeFormat(locale, { hour: 'numeric', minute: '2-digit', timeZone: 'UTC' }).format(
190       new Date(Date.UTC(2000, 0, 1, h, m)),
191     );
192   }
193
194   export interface Airing extends AiringEpisode {
195     /** Date it airs on in the chosen zone, `yyyy-mm-dd`. */
196     date: string;
197   }
198
199   /**
200    * Upcoming airings, soonest first: the detail-only `upcomingEpisodes` list
201    * when present, else the single `nextAiringEpisode` every query has.
202    */
203   export function upcomingAirings(anime: Anime, now: number = Date.now(), timezone?: string | null): Airing[] {
204     const source =
205       anime.upcomingEpisodes && anime.upcomingEpisodes.length > 0
206         ? anime.upcomingEpisodes
207         : anime.nextAiringEpisode
208           ? [anime.nextAiringEpisode]
209           : [];
210     return source
211       .filter((a) => a.airingAt * 1000 > now)
212       .sort((a, b) => a.airingAt - b.airingAt)
213       .map((a) => ({ ...a, date: localIso(new Date(a.airingAt * 1000), timezone) }));
214   }
215
216   /** Day + time labels for an air time in `timezone` (device zone when null), e.g. { day: "Sat, Oct 3", time: "5:30 PM" }. */
217   export function formatAiring(
218     airingAt: number,
219     timezone?: string | null,
220     locale?: string,
221   ): { day: string; time: string } {
222     const date = new Date(airingAt * 1000);
223     const zone = timezone ? { timeZone: timezone } : {};
224     return {
225       day: new Intl.DateTimeFormat(locale, { weekday: 'short', month: 'short', day: 'numeric', ...zone }).format(date),
226       time: new Intl.DateTimeFormat(locale, { hour: 'numeric', minute: '2-digit', ...zone }).format(date),
227     };
228   }
229
230   export function sameCalendarPrefs(a: CalendarPrefs, b: CalendarPrefs): boolean {
231     const days = (p: CalendarPrefs) => [...p.watchDays].sort((x, y) => x - y).join(',');
232     return (
233       a.startDate === b.startDate &&
234       a.watchTime === b.watchTime &&
235       a.reminderMinutes === b.reminderMinutes &&
236       (a.timezone ?? null) === (b.timezone ?? null) &&
237       days(a) === days(b)
238     );
239   }

```

src/lib/progress.ts

73 lignes

Les calculs de progression : savoir si un jour est entièrement regardé, trouver le jour où reprendre, calculer le pourcentage terminé et la série de jours consécutifs (le « streak »).

```

1  // Pure progress/streak calculations. No fetch, no window, no React – same
2  // rules as schedule.ts.
3
4  import type { SavedSchedule, ScheduleDay, WatchedEpisode } from '../types';
5
6  /** Set of watched episode numbers, for 0(1) "is this day done" checks. */
7  export function watchedSet(watched: WatchedEpisode[]): Set<number> {

```

```

8   return new Set(watched.map((w) => w.episode));
9   }
10
11  /** A day counts as watched when every one of its episodes is. */
12  export function isDayWatched(day: ScheduleDay, watched: Set<number>): boolean {
13    return day.episodes.every((ep) => watched.has(ep));
14  }
15
16  /** First day not fully watched, or null if every day is. */
17  export function getResumeDay(days: ScheduleDay[], watched: WatchedEpisode[]): number | null {
18    const set = watchedSet(watched);
19    return days.find((d) => !isDayWatched(d, set))?.day ?? null;
20  }
21
22  export interface Progress {
23    /** Fully watched days. */
24    done: number;
25    total: number;
26    /** 0-100, rounded. `0` when `total` is 0. */
27    percent: number;
28  }
29
30  export function getProgress(days: ScheduleDay[], watched: WatchedEpisode[]): Progress {
31    const set = watchedSet(watched);
32    const done = days.filter((d) => isDayWatched(d, set)).length;
33    return {
34      done,
35      total: days.length,
36      percent: days.length > 0 ? Math.round((done / days.length) * 100) : 0,
37    };
38  }
39
40  /** YYYY-MM-DD in local time, used to compare "same calendar day". */
41  function dateKey(epochMs: number): string {
42    const d = new Date(epochMs);
43    return `${d.getFullYear()}-${d.getMonth()+1}-${d.getDate()}`;
44  }
45
46  /**
47   * Consecutive-day streak across all saved schedules combined, counting
48   * backward from today. A day counts if any schedule has a `watched` entry
49   * completed on that calendar date. If today has no entry yet, the streak
50   * still counts as long as yesterday does – checking in once a day
51   * shouldn't reset at midnight before you've had a chance to watch.
52   */
53  export function getCurrentStreak(schedules: SavedSchedule[], now = Date.now()): number {
54    const watchedDates = new Set(
55      schedules.flatMap((s) => s.watched.map((w) => dateKey(w.completedAt))),
56    );
57    if (watchedDates.size === 0) return 0;
58
59    const oneDay = 86_400_000;
60    let streak = 0;
61    let cursor = now;
62
63    if (!watchedDates.has(dateKey(cursor))) {
64      cursor -= oneDay;
65      if (!watchedDates.has(dateKey(cursor))) return 0;
66    }
67
68    while (watchedDates.has(dateKey(cursor))) {
69      streak++;
70      cursor -= oneDay;
71    }
72    return streak;
73  }

```

src/lib/storage.ts

184 lignes

La sauvegarde des plannings dans le navigateur (`localStorage`). Chaque lecture et écriture est protégée : si le stockage est plein, désactivé ou corrompu, l'application continue sans planter. Elle convertit aussi les anciens formats de sauvegarde.

1 // localStorage wrapper for saved schedules. Every read and write is wrapped

```

2 // in try/catch: storage can be disabled, full, or hold corrupt data, and
3 // none of that should crash the app. Failures degrade to "not persisted".
4 //
5 // Kept separate from schedule.ts, which must stay pure.
6
7 import type { Anime, CalendarPrefs, SavedSchedule, ScheduleConfig, WatchedEpisode } from '../types';
8 import { generateSchedule } from './schedule';
9 import { REMINDER_OPTIONS, defaultCalendarPrefs, isIsoDate, isWatchTime, localIso, isTimeZone } from './calendar';
10
11 /** Bump the version suffix if the SavedSchedule shape changes incompatibly. */
12 export const STORAGE_KEY = 'anime-planner:schedules:v1';
13
14 /**
15  * Normalizes a stored `watched` list to per-episode entries. Records saved
16  * before progress went per-episode hold `{ day, completedAt }`; those are
17  * converted by re-running the record's own saved anime + config through
18  * `generateSchedule` to recover which episodes that day contained, keeping
19  * the original `completedAt`. Anything unrecognized is dropped.
20  */
21 function normalizeWatched(
22   raw: unknown,
23   anime: Anime,
24   config: ScheduleConfig,
25 ): WatchedEpisode[] {
26   if (!Array.isArray(raw)) return [];
27   const byEpisode = new Map<number, number>();
28   let dayToEpisodes: Map<number, number[]> | null = null;
29
30   for (const entry of raw) {
31     if (typeof entry !== 'object' || entry === null) continue;
32     const e = entry as { episode?: unknown; day?: unknown; completedAt?: unknown };
33     if (typeof e.completedAt !== 'number') continue;
34
35     if (typeof e.episode === 'number') {
36       if (!byEpisode.has(e.episode)) byEpisode.set(e.episode, e.completedAt);
37     } else if (typeof e.day === 'number') {
38       if (!dayToEpisodes) {
39         const result = generateSchedule(anime, config);
40         dayToEpisodes = new Map(result.ok ? result.schedule.days.map((d) => [d.day, d.episodes]) : []);
41       }
42       for (const ep of dayToEpisodes.get(e.day) ?? []) {
43         if (!byEpisode.has(ep)) byEpisode.set(ep, e.completedAt);
44       }
45     }
46   }
47   return [...byEpisode].map(([episode, completedAt]) => ({ episode, completedAt }));
48 }
49
50 /**
51  * Validates stored calendar prefs field by field; anything missing or
52  * invalid falls back to its default. Records from before calendar prefs
53  * existed start on the day they were saved, so their dates line up with
54  * the progress already made.
55  */
56 function normalizeCalendar(raw: unknown, savedAt: number): CalendarPrefs {
57   const fallback = defaultCalendarPrefs(localIso(new Date(savedAt)));
58   if (typeof raw !== 'object' || raw === null) return fallback;
59   const c = raw as Partial<Record<keyof CalendarPrefs, unknown>>;
60
61   const days = Array.isArray(c.watchDays)
62     ? [...new Set(c.watchDays.filter((d): d is number => Number.isInteger(d) && d >= 0 && d <= 6))].sort((a, b) =>
63       a - b)
64     : [];
65   const watchTime = isWatchTime(c.watchTime) ? c.watchTime : null;
66   const reminder =
67     watchTime && REMINDER_OPTIONS.includes(c.reminderMinutes as number | null)
68       ? (c.reminderMinutes as number | null)
69       : null;
70
71   return {
72     startDate: isIsoDate(c.startDate) ? c.startDate : fallback.startDate,
73     watchDays: days.length > 0 ? days : fallback.watchDays,
74     watchTime,
75     reminderMinutes: reminder,
76     timeZone: isTimeZone(c.timeZone) ? c.timeZone : null,
77   };
78 }

```

```

78
79 /**
80  * Checks the fields that existed when this record could have been written,
81  * so a record saved before `watched` was added still loads. `watched`
82  * itself is normalized separately in `getSavedSchedules`.
83  */
84 function hasBaseShape(
85   value: unknown,
86 ): value is Omit<SavedSchedule, 'watched' | 'calendar'> & { watched?: unknown; calendar?: unknown } {
87   if (typeof value !== 'object' || value === null) return false;
88   const v = value as Partial<SavedSchedule>;
89   return (
90     typeof v.id === 'string' &&
91     typeof v.savedAt === 'number' &&
92     typeof v.anime?.id === 'number' &&
93     typeof v.anime?.title?.romaji === 'string' &&
94     typeof v.config?.dailyBudget === 'number' &&
95     typeof v.config?.skipOpEd === 'boolean'
96   );
97 }
98
99 /** Newest first. Returns `[]` if storage is unavailable or unreadable. */
100 export function getSavedSchedules(): SavedSchedule[] {
101   try {
102     const raw = localStorage.getItem(STORAGE_KEY);
103     if (!raw) return [];
104     const parsed: unknown = JSON.parse(raw);
105     if (!Array.isArray(parsed)) return [];
106     return parsed
107       .filter(hasBaseShape)
108       .map((s) => ({
109         ...s,
110         calendar: normalizeCalendar(s.calendar, s.savedAt),
111         watched: normalizeWatched(s.watched, s.anime, s.config),
112       }))
113       .sort((a, b) => b.savedAt - a.savedAt);
114   } catch {
115     return [];
116   }
117 }
118
119 function write(list: SavedSchedule[]): boolean {
120   try {
121     localStorage.setItem(STORAGE_KEY, JSON.stringify(list));
122     return true;
123   } catch {
124     return false;
125   }
126 }
127
128 /**
129  * Saves (or replaces) the schedule for this anime. One entry per anime id,
130  * so re-saving updates the config instead of adding a duplicate. Preserves
131  * existing watch progress when replacing, and existing calendar prefs when
132  * `calendar` isn't given.
133  * Returns the saved record, or `null` if it couldn't be persisted.
134  */
135 export function saveSchedule(
136   anime: Anime,
137   config: ScheduleConfig,
138   calendar?: CalendarPrefs,
139 ): SavedSchedule | null {
140   const existing = getSavedSchedules().find((s) => s.anime.id === anime.id);
141   const record: SavedSchedule = {
142     id: existing?.id ?? crypto.randomUUID(),
143     savedAt: Date.now(),
144     anime,
145     config,
146     calendar: calendar ?? existing?.calendar ?? defaultCalendarPrefs(),
147     watched: existing?.watched ?? [],
148   };
149   const others = getSavedSchedules().filter((s) => s.anime.id !== anime.id);
150   return write([record, ...others]) ? record : null;
151 }
152
153 /** Returns false if storage couldn't be written. */
154 export function deleteSavedSchedule(id: string): boolean {

```

```

155     return write(getSavedSchedules().filter((s) => s.id !== id));
156 }
157
158 /**
159  * Toggles a day's worth of episodes for the saved schedule `id`: if every
160  * one of `episodes` is already watched they're all unmarked, otherwise the
161  * missing ones are marked now. Returns the updated record, or `null` if the
162  * id doesn't exist or the write failed.
163  */
164 export function toggleEpisodesWatched(id: string, episodes: number[]): SavedSchedule | null {
165     const list = getSavedSchedules();
166     const index = list.findIndex((s) => s.id === id);
167     if (index === -1) return null;
168
169     const target = list[index];
170     const watchedSet = new Set(target.watched.map((w) => w.episode));
171     const allWatched = episodes.every((ep) => watchedSet.has(ep));
172     const now = Date.now();
173     const watched = allWatched
174         ? target.watched.filter((w) => !episodes.includes(w.episode))
175         : [
176             ...target.watched,
177             ...episodes.filter((ep) => !watchedSet.has(ep)).map((episode) => ({ episode, completedAt: now })),
178         ];
179
180     const updated: SavedSchedule = { ...target, watched };
181     const next = [...list];
182     next[index] = updated;
183     return write(next) ? updated : null;
184 }

```

src/lib/ics.ts

150 lignes

La fabrication du fichier calendrier `.ics` (un format texte standard que Google Agenda, Apple Calendrier ou Outlook savent importer) : un événement par jour de visionnage, avec rappel éventuel, plus les dates de diffusion des épisodes à venir.

```

1  // Pure ICS (iCalendar) text builder. Data in, string out — no `document`,
2  // no `Blob`. The actual file download is `lib/download.ts`'s job.
3
4  import type { Anime, CalendarPrefs, Schedule } from '../types';
5  import { displayTitle } from './format';
6  import { addDays, scheduleDates, upcomingAirings } from './calendar';
7
8  function pad(n: number): string {
9      return String(n).padStart(2, '0');
10 }
11
12 /** All-day ICS date from `yyyy-mm-dd`, e.g. 20260115. */
13 function icsDate(iso: string): string {
14     return iso.replace(/-/g, '');
15 }
16
17 /**
18  * Local "floating" date-time (no Z, no TZID): calendar apps place it at that
19  * wall-clock time in whatever zone the user is in, which is what a personal
20  * "watch at 8 PM" slot means. Computed with UTC fields as plain arithmetic.
21  */
22 function icsFloating(iso: string, hhmm: string, plusMinutes = 0): string {
23     const [y, mo, d] = iso.split('-').map(Number);
24     const [h, mi] = hhmm.split(':').map(Number);
25     const t = new Date(Date.UTC(y, mo - 1, d, h, mi + plusMinutes));
26     return `${t.getUTCFullYear()}${pad(t.getUTCMonth() +
27 1)}${pad(t.getUTCDate())}T${pad(t.getUTCHours())}${pad(t.getUTCMinutes())}00`;
28 }
29
30 function icsTimestamp(date: Date): string {
31     const y = date.getUTCFullYear();
32     const m = pad(date.getUTCMonth() + 1);
33     const d = pad(date.getUTCDate());
34     return `${y}${m}${d}T${pad(date.getUTCHours())}${pad(date.getUTCMinutes())}${pad(date.getUTCSeconds())}Z`;
35 }
36
37 /** [1,2,3] -> "1-3", [5] -> "5". */

```

```

37 function episodeRange(episodes: number[]): string {
38     const first = episodes[0];
39     const last = episodes[episodes.length - 1];
40     return first === last ? `${first}` : `${first}-${last}`;
41 }
42
43 /** Escapes text per RFC 5545 (backslash, comma, semicolon, newline). */
44 function escapeText(text: string): string {
45     return text
46         // A bare CR could otherwise start a new ICS property line in some parsers.
47         .replace(/\r\n?/g, '\n')
48         .replace(/\\/g, '\\\\')
49         .replace(/;/g, '\\;')
50         .replace(/,/g, '\\,')
51         .replace(/\n/g, '\\n');
52 }
53
54 /** Folds a line at 75 octets, as RFC 5545 requires for long lines. */
55 function foldLine(line: string): string {
56     if (line.length <= 75) return line;
57     const parts: string[] = [];
58     let rest = line;
59     while (rest.length > 75) {
60         parts.push(rest.slice(0, 75));
61         rest = ' ' + rest.slice(75);
62     }
63     parts.push(rest);
64     return parts.join('\r\n');
65 }
66
67 /**
68  * One event per schedule day, on the dates `scheduleDates` gives for the
69  * user's start date and watch days. With a `watchTime` the events are timed
70  * (lasting that day's minutes) and get a reminder when `reminderMinutes` is
71  * set; without one they're all-day. Shows still airing also get one event
72  * per known upcoming air time. `episodeTitles` (episode number -> title, see
73  * `episodeTitleMap`) is optional; when present, each day's description
74  * lists the real episode titles. UIDs are stable, so re-importing updates
75  * events instead of duplicating them.
76  */
77 export function buildIcs(
78     anime: Anime,
79     schedule: Schedule,
80     prefs: CalendarPrefs,
81     episodeTitles?: Map<number, string>,
82     now: Date = new Date(),
83 ): string {
84     const title = displayTitle(anime);
85     const stamp = icsTimestamp(now);
86     const dates = scheduleDates(schedule.days, prefs);
87     const { watchTime, reminderMinutes } = prefs;
88
89     const events = schedule.days.map((day) => {
90         const date = dates.get(day.day)!;
91         const titles = episodeTitles
92             ? day.episodes
93                 .map((ep) => episodeTitles.get(ep))
94                 .filter((t): t is string => Boolean(t))
95             : [];
96
97         const summary = `${title} - Ep ${episodeRange(day.episodes)}`;
98         const lines = [
99             'BEGIN:VEVENT',
100             `UID:${anime.id}-${day.day}@anime-planner`,
101             `DTSTAMP:${stamp}`,
102             ...(watchTime
103                 ? [`DTSTART:${icsFloating(date, watchTime)}`, `DTEND:${icsFloating(date, watchTime, day.minutes)}`]
104                 : [`DTSTART;VALUE=DATE:${icsDate(date)}`, `DTEND;VALUE=DATE:${icsDate(addDays(date, 1))}`]),
105             `SUMMARY:${escapeText(summary)}`,
106         ];
107         if (titles.length > 0) {
108             lines.push(`DESCRIPTION:${escapeText(titles.join('\n'))}`);
109         }
110         if (watchTime && reminderMinutes != null) {
111             lines.push(
112                 'BEGIN:VALARM',
113                 'ACTION:DISPLAY',

```

```

114     `DESCRIPTION:${escapeText(summary)}`,
115     `TRIGGER:-PT${reminderMinutes}M`,
116     'END:VALARM',
117   );
118 }
119 lines.push('END:VEVENT');
120 return lines.map(foldLine).join('\r\n');
121 });
122
123 const airLength = anime.duration ?? 24;
124 const airings = upcomingAirings(anime, now.getTime()).map((a) => {
125   const start = new Date(a.airingAt * 1000);
126   const end = new Date(start.getTime() + airLength * 60_000);
127   return [
128     'BEGIN:VEVENT',
129     `UID:${anime.id}-air-${a.episode}@anime-planner`,
130     `DTSTAMP:${stamp}`,
131     `DTSTART:${icsTimestamp(start)}`,
132     `DTEND:${icsTimestamp(end)}`,
133     `SUMMARY:${escapeText(`${title} - Ep ${a.episode} airs`)}`,
134     'TRANSP:TRANSPARENT',
135     'END:VEVENT',
136   ]
137   .map(foldLine)
138   .join('\r\n');
139 });
140
141 return [
142   'BEGIN:VCALENDAR',
143   'VERSION:2.0',
144   'PRODID:-//anime-planner//schedule//EN',
145   'CALSCALE:GREGORIAN',
146   ...events,
147   ...airings,
148   'END:VCALENDAR',
149 ].join('\r\n');
150 }

```

src/lib/route.ts

18 lignes

Construit et lit l'adresse de la page d'une série, du type `/anime/154587-frieren-beyond-journey-s-end`. Seul le numéro compte ; le texte après est là pour les humains et les moteurs de recherche.

```

1 // Pure URL helpers for the one real route: a show's page, /anime/<id>-<slug>.
2 // The slug is only for people and search engines; lookups use the id, so an
3 // old link keeps working if a show's title changes.
4
5 import { slugify } from './format';
6
7 export function animePath(id: number, title: string): string {
8   const slug = slugify(title);
9   return `/anime/${id}${slug ? `-${slug}` : ''}`;
10 }
11
12 /** `/anime/123-some-title` (or `/anime/123`, with or without a trailing slash) -> 123. */
13 export function parseAnimePath(pathname: string): number | null {
14   const match = /^\/anime\/(\d{1,9})(?:-[a-z0-9-]*)?\/?$/i.exec(pathname);
15   if (!match) return null;
16   const id = Number(match[1]);
17   return id > 0 ? id : null;
18 }

```

src/lib/seo.ts

130 lignes

Tout ce qui concerne le référencement : le titre de chaque page, sa description, les balises d'aperçu pour les réseaux sociaux et les données structurées JSON-LD. Le même code sert dans le navigateur et pendant le build.

```

1 // Pure SEO data: page titles, descriptions, social-card tags and JSON-LD for
2 // the homepage and each show. Shared by App.tsx (updates <head> while
3 // browsing) and scripts/prerender.ts (writes static pages at build time),
4 // so both always produce the same tags.
5

```

```

6 import type { Anime } from '../types';
7 import { displayTitle, plainText, safeExternalUrl } from './format';
8 import { animePath } from './route';
9
10 export const SITE_URL = 'https://anime-planner.vercel.app';
11 export const SITE_NAME = 'Anime Watch Planner';
12
13 export interface PageSeo {
14   title: string;
15   description: string;
16   /** Absolute canonical URL. */
17   url: string;
18   image: string | null;
19   type: 'website' | 'video.tv_show' | 'video.movie';
20 }
21
22 export const HOME_SEO: PageSeo = {
23   title: 'Anime Watch Planner: plan any anime day by day',
24   description:
25     'Plan any anime around the time you actually have: a day-by-day watch schedule with real episode titles, your
watch days, reminders and a calendar export.',
26   url: `${SITE_URL}/`,
27   image: null,
28   type: 'website',
29 };
30
31 const MAX_DESCRIPTION = 160;
32
33 function truncate(text: string, max: number): string {
34   if (text.length <= max) return text;
35   const cut = text.slice(0, max - 1);
36   return `${cut.slice(0, Math.max(cut.lastIndexOf(' '), max - 20)).trimEnd()}...`;
37 }
38
39 export function animeSeo(anime: Anime): PageSeo {
40   const name = displayTitle(anime);
41   const facts = [
42     anime.format === 'MOVIE' ? null : anime.episodes ? `${anime.episodes} episodes` : null,
43     anime.duration ? `${anime.duration} min${anime.format === 'MOVIE' ? '' : ' each'}` : null,
44   ]
45     .filter(Boolean)
46     .join(', ');
47   const lead = `Plan ${name}${facts ? ` (${facts})` : ''} day by day.`;
48   const synopsis = anime.description ? plainText(anime.description) : '';
49   return {
50     title: `${name} watch schedule | ${SITE_NAME}`,
51     description: truncate(synopsis ? `${lead} ${synopsis}` : `${lead} Build a watch schedule that fits your time
and add it to your calendar.`, MAX_DESCRIPTION),
52     url: `${SITE_URL}${animePath(anime.id, name)}`,
53     image: safeExternalUrl(anime.coverImage.extraLarge ?? anime.coverImage.large ?? ''),
54     type: anime.format === 'MOVIE' ? 'video.movie' : 'video.tv_show',
55   };
56 }
57
58 export interface MetaTag {
59   attr: 'name' | 'property';
60   key: string;
61   content: string;
62 }
63
64 /** Every <meta> tag a page should have; image tags only when there's an image. */
65 export function metaTags(seo: PageSeo): MetaTag[] {
66   const tags: MetaTag[] = [
67     { attr: 'name', key: 'description', content: seo.description },
68     { attr: 'property', key: 'og:site_name', content: SITE_NAME },
69     { attr: 'property', key: 'og:type', content: seo.type },
70     { attr: 'property', key: 'og:title', content: seo.title },
71     { attr: 'property', key: 'og:description', content: seo.description },
72     { attr: 'property', key: 'og:url', content: seo.url },
73     // Covers are portrait; the wide Cards would crop them.
74     { attr: 'name', key: 'twitter:card', content: 'summary' },
75     { attr: 'name', key: 'twitter:title', content: seo.title },
76     { attr: 'name', key: 'twitter:description', content: seo.description },
77   ];
78   if (seo.image) {
79     tags.push(
80       { attr: 'property', key: 'og:image', content: seo.image },

```

```

81     { attr: 'name', key: 'twitter:image', content: seo.image },
82   );
83 }
84 return tags;
85 }
86
87 /** The meta keys that are only present with an image, so callers can remove stale ones. */
88 export const IMAGE_META_KEYS = ['og:image', 'twitter:image'];
89
90 /** schema.org data for a show page. */
91 export function animeJsonLd(anime: Anime): Record<string, unknown> {
92   const seo = animeSeo(anime);
93   return {
94     '@context': 'https://schema.org',
95     '@type': anime.format === 'MOVIE' ? 'Movie' : 'TVSeries',
96     name: displayTitle(anime),
97     ...(anime.title.english && anime.title.english !== anime.title.romaji
98       ? { alternateName: anime.title.romaji }
99       : {}),
100     url: seo.url,
101     ...(seo.image ? { image: seo.image } : {}),
102     ...(anime.description ? { description: truncate(plainText(anime.description), 500) } : {}),
103     ...(anime.genres.length ? { genre: anime.genres } : {}),
104     ...(anime.format !== 'MOVIE' && anime.episodes ? { numberOfEpisodes: anime.episodes } : {}),
105     ...(anime.seasonYear ? { startDate: String(anime.seasonYear) } : {}),
106   };
107 }
108
109 export function escapeHtml(text: string): string {
110   return text.replace(/&/g, '&amp;').replace(/</g, '&lt;').replace(/>/g, '&gt;').replace(/"/g, '&quot;');
111 }
112
113 /**
114  * The <head> block between the `<!--seo-start-->`/`<!--seo-end-->` markers in
115  * index.html, as HTML. `<` inside JSON-LD is escaped so data can't close the
116  * script tag.
117  */
118 export function renderHeadTags(seo: PageSeo, jsonLd?: Record<string, unknown>): string {
119   const lines = [
120     `<title>${escapeHtml(seo.title)}</title>`,
121     `<link rel="canonical" href="${escapeHtml(seo.url)}" />`,
122     ...metaTags(seo).map((t) => `<meta ${t.attr}="${t.key}" content="${escapeHtml(t.content)}" />`),
123   ];
124   if (jsonLd) {
125     lines.push(
126       `<script type="application/ld+json">${JSON.stringify(jsonLd).replace(/</g, '\\u003c')}</script>`,
127     );
128   }
129   return lines.map((l) => `    ${l}`).join('\n');
130 }

```

src/lib/share.ts

57 lignes

Le décodage des anciens liens de partage `?share=...`, qui contenaient l'identifiant de la série et les réglages. Ils fonctionnent toujours, et le décodage est très strict pour qu'un lien fabriqué ne puisse rien injecter.

```

1  // Pure encode/decode for the `?share=` URL param. Encodes only the anime id
2  // + config (not a full snapshot) so links stay short; the receiving app
3  // always refetches fresh Anilist data.
4
5  import type { Anime, ScheduleConfig } from '../types';
6
7  export interface SharePayload {
8    id: number;
9    config: ScheduleConfig;
10 }
11
12 function base64UrlEncode(text: string): string {
13   return btoa(unescape(encodeURIComponent(text)))
14     .replace(/\+/g, '-')
15     .replace(/\//g, '_')
16     .replace(/=+$/, '');
17 }
18

```

```

19 function base64UrlDecode(text: string): string {
20     const padded = text.replace(/-/g, '+').replace(/_/g, '/').padEnd(Math.ceil(text.length / 4) * 4, '=');
21     return decodeURIComponent(escape(atob(padded)));
22 }
23
24 export function encodeShareParam(anime: Anime, config: ScheduleConfig): string {
25     return base64UrlEncode(JSON.stringify({ id: anime.id, config }));
26 }
27
28 /** Longest daily budget a share link may carry (a full day, in minutes). */
29 const MAX_BUDGET = 1440;
30
31 /**
32  * Rebuilds a payload from known fields only, so a crafted link can't carry
33  * extra keys into app state or saved data. Rejects non-positive/non-integer
34  * ids and out-of-range budgets.
35  */
36 function toSharePayload(value: unknown): SharePayload | null {
37     if (typeof value !== 'object' || value === null) return null;
38     const v = value as { id?: unknown; config?: { dailyBudget?: unknown; skipOpEd?: unknown } };
39     const id = v.id;
40     const budget = v.config?.dailyBudget;
41     const skipOpEd = v.config?.skipOpEd;
42     if (typeof id !== 'number' || !Number.isInteger(id) || id <= 0) return null;
43     if (typeof budget !== 'number' || !Number.isFinite(budget) || budget < 1 || budget > MAX_BUDGET) {
44         return null;
45     }
46     if (typeof skipOpEd !== 'boolean') return null;
47     return { id, config: { dailyBudget: budget, skipOpEd } };
48 }
49
50 /** Returns `null` on any malformed/corrupt input rather than throwing. */
51 export function decodeShareParam(param: string): SharePayload | null {
52     try {
53         return toSharePayload(JSON.parse(base64UrlDecode(param)));
54     } catch {
55         return null;
56     }
57 }

```

src/lib/platform.ts

13 lignes

Une seule vérification : l'appareil est-il un iPhone ou un iPad ? Sur iOS, l'export calendrier s'ouvre dans l'onglet au lieu d'être téléchargé.

```

1 // Read-only device checks. Kept tiny and separate so the rest of lib/ stays
2 // free of browser globals.
3
4 /**
5  * iPhone, iPod or iPad. iPadOS 13+ reports itself as a Mac in the user
6  * agent, so a "Mac" with a touch screen counts as an iPad too.
7  */
8 export function isIOS(nav: Pick<Navigator, 'userAgent' | 'platform' | 'maxTouchPoints'> | undefined =
9     typeof navigator !== 'undefined' ? navigator): boolean {
10     if (!nav) return false;
11     if (/iPhone|iPad|iPod/.test(nav.userAgent)) return true;
12     return nav.platform === 'MacIntel' && nav.maxTouchPoints > 1;
13 }

```

src/lib/download.ts

37 lignes

Déclenche le téléchargement d'un fichier texte (le `.ics`) ou l'ouvre dans l'onglet. C'est isolé ici pour que `ics.ts` reste pur.

```

1 // Isolated side effect: triggers a browser file download. Kept out of
2 // lib/ics.ts, which stays pure (string in, string out).
3
4 /** Returns false if the download couldn't be triggered. */
5 export function downloadTextFile(filename: string, content: string, mimeType: string): boolean {
6     try {
7         const blob = new Blob([content], { type: mimeType });
8         const url = URL.createObjectURL(blob);

```

```

9      const a = document.createElement('a');
10      a.href = url;
11      a.download = filename;
12      document.body.appendChild(a);
13      a.click();
14      a.remove();
15      URL.revokeObjectURL(url);
16      return true;
17    } catch {
18      return false;
19    }
20  }
21
22  /**
23   * Opens the content in the current tab instead of downloading it. For a
24   * `text/calendar` file this lets iOS Safari show its "Add to Calendar"
25   * sheet. The object URL is revoked later, not at once, because the
26   * navigation happens asynchronously. Returns false if it couldn't start.
27   */
28  export function openTextFile(content: string, mimeType: string): boolean {
29    try {
30      const url = URL.createObjectURL(new Blob([content], { type: mimeType }));
31      window.location.assign(url);
32      setTimeout(() => URL.revokeObjectURL(url), 60_000);
33      return true;
34    } catch {
35      return false;
36    }
37  }

```

A.5 Parler à AniList

src/api/anilist.ts

333 lignes

Le client de l'API AniList. Il contient les requêtes GraphQL (recherche, séries populaires, tendances, détail d'une série), les envoie, gère les erreurs et la limite de requêtes, puis remet les réponses en forme : liens de streaming officiels, ordre de visionnage d'une franchise, épisodes à venir.

```

1  // AniList GraphQL client. Read-only queries, no API key needed.
2  //
3  // AniList rate-limits aggressively (~90 req/min, 429 + Retry-After). There
4  // is no automatic backoff here: search runs on submit, not per keystroke.
5  // Add debouncing before adding request volume.
6
7  import type { Anime } from '../types';
8  import { safeExternalUrl, safeHexColor } from '../lib/format';
9
10 const ENDPOINT = 'https://graphql.anilist.co';
11
12 // Shared by every query so the shapes can't drift. Extend this fragment
13 // (and `Anime` in types.ts) rather than duplicating fields.
14 const MEDIA_FIELDS = /* GraphQL */ `
15   fragment MediaFields on Media {
16     id
17     title {
18       romaji
19       english
20     }
21     format
22     status
23     episodes
24     duration
25     seasonYear
26     averageScore
27     coverImage {
28       large
29       extraLarge
30       color
31     }
32     genres
33     studios(isMain: true) {
34       nodes {

```

```

35         name
36     }
37 }
38 nextAiringEpisode {
39     episode
40     airingAt
41 }
42 trailer {
43     id
44     site
45 }
46 }
47 `;
48
49 const SEARCH_QUERY = /* GraphQL */ `
50     query SearchAnime($search: String!) {
51         Page(page: 1, perPage: 12) {
52             media(search: $search, type: ANIME, isAdult: false, sort: SEARCH_MATCH) {
53                 ...MediaFields
54             }
55         }
56     }
57     ${MEDIA_FIELDS}
58 `;
59
60 const TRENDING_QUERY = /* GraphQL */ `
61     query TrendingAnime {
62         Page(page: 1, perPage: 25) {
63             media(type: ANIME, isAdult: false, sort: TRENDING_DESC) {
64                 ...MediaFields
65             }
66         }
67     }
68     ${MEDIA_FIELDS}
69 `;
70
71 // "Popular right now": currently airing shows by popularity. Deliberately
72 // not "this season": at the start of a season nearly every show is still
73 // unaired (no episodes, no duration), so the list would be unplannable.
74 const POPULAR_AIRING_QUERY = /* GraphQL */ `
75     query PopularAiring {
76         Page(page: 1, perPage: 30) {
77             media(type: ANIME, isAdult: false, status: RELEASING, sort: POPULARITY_DESC) {
78                 ...MediaFields
79             }
80         }
81     }
82     ${MEDIA_FIELDS}
83 `;
84
85 const BY_ID_QUERY = /* GraphQL */ `
86     query GetAnime($id: Int!) {
87         Media(id: $id, type: ANIME) {
88             ...MediaFields
89             description(asHtml: false)
90             streamingEpisodes {
91                 title
92             }
93             airingSchedule(notYetAired: true, perPage: 8) {
94                 nodes {
95                     episode
96                     airingAt
97                 }
98             }
99             externalLinks {
100                 site
101                 url
102                 type
103                 color
104             }
105             relations {
106                 edges {
107                     relationType
108                     node {
109                         id
110                         title {
111                             romaji

```

```

112         english
113     }
114     format
115     seasonYear
116     relations {
117         edges {
118             relationType
119             node {
120                 id
121                 title {
122                     romaji
123                     english
124                 }
125                 format
126                 seasonYear
127             }
128         }
129     }
130 }
131 }
132 }
133 }
134 }
135 ${MEDIA_FIELDS}
136 `;
137
138 const RELATED_ANIME_FORMATS = new Set(['TV', 'TV_SHORT', 'MOVIE', 'SPECIAL', 'OVA', 'ONA']);
139
140 type RawAnime = Omit<Anime, 'studios'> & { studios: { nodes: { name: string }[] } };
141
142 /**
143  * Flattens AniList's `studios { nodes { name } }` to `{ name }[]`, and
144  * validates the cover color (it feeds CSS custom properties for theming).
145  */
146 function normalizeAnime(raw: RawAnime): Anime {
147     return {
148         ...raw,
149         studios: raw.studios.nodes,
150         coverImage: { ...raw.coverImage, color: safeHexColor(raw.coverImage.color) },
151     };
152 }
153
154 interface RawRelationNode {
155     id: number;
156     title: { romaji: string; english: string | null };
157     format: string | null;
158     seasonYear: number | null;
159     /** Only present at depth 1 – the query nests one level deep. */
160     relations?: { edges: RawRelationEdge[] };
161 }
162
163 interface RawRelationEdge {
164     relationType: string;
165     node: RawRelationNode;
166 }
167
168 function isWatchOrderEdge(e: RawRelationEdge): boolean {
169     return (
170         (e.relationType === 'PREQUEL' || e.relationType === 'SEQUEL') &&
171         e.node.format !== null &&
172         RELATED_ANIME_FORMATS.has(e.node.format)
173     );
174 }
175
176 /**
177  * Builds a chronological watch-order list from the current anime plus every
178  * prequel/sequel found up to 2 hops away (its direct neighbors, and their
179  * neighbors in turn) – one request, no follow-up round trips. Entries are
180  * sorted by `seasonYear`, which matches release order for the vast majority
181  * of real franchises; a show released out of story order could sort
182  * "wrong" here, same spirit as `schedule.ts`'s greedy-bin-packing caveat.
183  * A franchise with no separate PREQUEL/SEQUEL entries on AniList (e.g. One
184  * Piece, which is one continuous entry) simply produces no list – correct,
185  * not a bug.
186  */
187 function buildWatchOrder(
188     current: { id: number; title: { romaji: string; english: string | null }; seasonYear: number | null },

```

```

189 edges: RawRelationEdge[] | undefined,
190 ): Anime['relations'] {
191   if (!edges) return undefined;
192
193   const nodes = new Map<number, RawRelationNode>();
194   for (const e of edges.filter(isWatchOrderEdge)) {
195     if (e.node.id !== current.id) nodes.set(e.node.id, e.node);
196     for (const e2 of (e.node.relations?.edges ?? []).filter(isWatchOrderEdge)) {
197       if (e2.node.id !== current.id && !nodes.has(e2.node.id)) nodes.set(e2.node.id, e2.node);
198     }
199   }
200   if (nodes.size === 0) return undefined;
201
202   const all = [
203     { id: current.id, title: current.title, seasonYear: current.seasonYear, isCurrent: true },
204     ...[...nodes.values()].map((n) => ({
205       id: n.id,
206       title: n.title,
207       seasonYear: n.seasonYear,
208       isCurrent: false,
209     })),
210   ];
211
212   return all
213     .sort((a, b) => {
214       if (a.seasonYear == null) return b.seasonYear == null ? a.id - b.id : 1;
215       if (b.seasonYear == null) return -1;
216       return a.seasonYear - b.seasonYear || a.id - b.id;
217     })
218     .map((n) => ({
219       id: n.id,
220       title: n.title.english ?? n.title.romaji,
221       seasonYear: n.seasonYear,
222       isCurrent: n.isCurrent,
223     }));
224 }
225
226 export class AniListError extends Error {
227   readonly status?: number;
228   /** Seconds to wait, from the Retry-After header on a 429. */
229   readonly retryAfter?: number;
230
231   constructor(message: string, status?: number, retryAfter?: number) {
232     super(message);
233     this.name = 'AniListError';
234     this.status = status;
235     this.retryAfter = retryAfter;
236   }
237 }
238
239 async function request<T>({
240   query: string,
241   variables: Record<string, unknown>,
242   signal?: AbortSignal,
243 }): Promise<T> {
244   const res = await fetch(ENDPOINT, {
245     method: 'POST',
246     headers: { 'Content-Type': 'application/json', Accept: 'application/json' },
247     body: JSON.stringify({ query, variables }),
248     signal,
249   });
250
251   if (res.status === 429) {
252     const retryAfter = Number(res.headers.get('Retry-After')) || undefined;
253     throw new AniListError(
254       `AniList rate limit hit${retryAfter ? `, try again in ${retryAfter}s` : ''}.`,
255       429,
256       retryAfter,
257     );
258   }
259
260   const body = (await res.json()).catch(() => null) as {
261     data?: T;
262     errors?: { message: string }[];
263   } | null;
264
265   if (!res.ok || !body?.data) {

```

```

266     const message = body?.errors?.[0]?.message ?? `AniList request failed (${res.status}).`;
267     throw new AniListError(message, res.status);
268   }
269   return body.data;
270 }
271
272 export async function searchAnime(search: string, signal?: AbortSignal): Promise<Anime[]> {
273   const data = await request<{ Page: { media: RawAnime[] } }>(SEARCH_QUERY, { search }, signal);
274   return data.Page.media.map(normalizeAnime);
275 }
276
277 export async function getTrendingAnime(signal?: AbortSignal): Promise<Anime[]> {
278   const data = await request<{ Page: { media: RawAnime[] } }>(TRENDING_QUERY, {}, signal);
279   return data.Page.media.map(normalizeAnime);
280 }
281
282 export async function getPopularAiring(signal?: AbortSignal): Promise<Anime[]> {
283   const data = await request<{ Page: { media: RawAnime[] } }>(POPULAR_AIRING_QUERY, {}, signal);
284   return data.Page.media.map(normalizeAnime);
285 }
286
287 type RawAnimeDetail = Omit<RawAnime, 'relations' | 'watchLinks' | 'upcomingEpisodes'> & {
288   relations?: { edges: RawRelationEdge[] };
289   airingSchedule?: { nodes: ({ episode: unknown; airingAt: unknown } | null)[] } | null;
290   externalLinks?: { site: string; url: string; type: string | null; color: string | null }[];
291 };
292
293 /**
294  * Official streaming services only (`type` === 'STREAMING'), one per site.
295  * Anilist lists licensed platforms here (Crunchyroll, Netflix, ...); this is
296  * deliberately the only "where to watch" source.
297  */
298 function toWatchLinks(links: RawAnimeDetail['externalLinks']): Anime['watchLinks'] {
299   if (!links) return undefined;
300   const seen = new Set<string>();
301   const out: NonNullable<Anime['watchLinks']> = [];
302   for (const l of links) {
303     const url = safeExternalUrl(l.url);
304     if (l.type !== 'STREAMING' || !url || seen.has(l.site)) continue;
305     seen.add(l.site);
306     out.push({ site: l.site, url, color: safeHexColor(l.color) });
307   }
308   return out;
309 }
310
311 /** Keeps only well-formed entries, soonest first. */
312 function toUpcomingEpisodes(schedule: RawAnimeDetail['airingSchedule']): Anime['upcomingEpisodes'] {
313   const out: NonNullable<Anime['upcomingEpisodes']> = [];
314   for (const n of schedule?.nodes ?? []) {
315     if (!n) continue;
316     const { episode, airingAt } = n;
317     if (!Number.isInteger(episode) || !Number.isInteger(airingAt)) continue;
318     out.push({ episode: episode as number, airingAt: airingAt as number });
319   }
320   return out.sort((a, b) => a.airingAt - b.airingAt);
321 }
322
323 export async function getAnimeById(id: number, signal?: AbortSignal): Promise<Anime> {
324   const data = await request<{ Media: RawAnimeDetail }>(BY_ID_QUERY, { id }, signal);
325   const { relations, externalLinks, airingSchedule, ...rest } = data.Media;
326   const anime = normalizeAnime(rest);
327   return {
328     ...anime,
329     relations: buildWatchOrder(anime, relations?.edges),
330     watchLinks: toWatchLinks(externalLinks),
331     upcomingEpisodes: toUpcomingEpisodes(airingSchedule),
332   };
333 }

```

A.6 Les animations

src/motion/presets.ts

107 lignes

Les réglages communs des animations (durées, courbes, ressort) et quelques fonctions : apparition en fondu, petit « pop » de confirmation, et l'effet d'appui sur tous les boutons. Tout est désactivé si l'utilisateur a demandé moins d'animations dans son système.

```
1 // Shared anime.js settings and small imperative helpers. Everything here
2 // animates only opacity, transform and clip-path: colors come from the theme
3 // tokens in index.css and must never be tweened here.
4
5 import { animate, cleanInlineStyles, spring, stagger, type JSAnimation, type TargetsParam } from 'animejs';
6
7 export const DURATION = { fast: 180, base: 320, slow: 520 } as const;
8 export const EASE_OUT = 'outExpo';
9 export const SPRING = spring({ bounce: 0.45, duration: 380 });
10
11 export function reduceMotion(): boolean {
12   return typeof matchMedia === 'function' && matchMedia('(prefers-reduced-motion: reduce)').matches;
13 }
14
15 /** Drops the inline styles an animation left behind once it's done. */
16 const clean = (anim: JSAnimation) => {
17   cleanInlineStyles(anim);
18 };
19
20 export interface EnterOptions {
21   /** Starting vertical offset in px (0 for none). */
22   y?: number;
23   x?: number;
24   /** Starting scale (1 for none). */
25   scale?: number;
26   duration?: number;
27   delay?: number;
28   /** Delay between items; items past `maxSteps` start together. */
29   step?: number;
30   maxSteps?: number;
31   /** Stagger outward across a grid of [columns, rows] instead of by index. */
32   grid?: [number, number];
33 }
34
35 /** Fades targets in from an offset. Returns null under reduced motion. */
36 export function enter(targets: TargetsParam, opts: EnterOptions = {}): JSAnimation | null {
37   if (reduceMotion()) return null;
38   const { y = 16, x = 0, scale = 1, duration = DURATION.base, delay = 0, step = 0, maxSteps = 12, grid } = opts;
39   const staggered = grid
40     ? stagger(step, { grid, from: 'first', start: delay })
41     : step
42       ? (_, i: unknown, i = 0) => delay + Math.min(i, maxSteps) * step
43       : delay;
44   return animate(targets, {
45     opacity: { from: 0, to: 1 },
46     ...(y ? { y: { from: y, to: 0 } } : {}),
47     ...(x ? { x: { from: x, to: 0 } } : {}),
48     ...(scale !== 1 ? { scale: { from: scale, to: 1 } } : {}),
49     duration,
50     delay: staggered,
51     ease: EASE_OUT,
52     onComplete: clean,
53   });
54 }
55
56 /** A quick scale pop for confirmations (check-offs, "Copied", toggles). */
57 export function pop(target: TargetsParam | null | undefined, amount = 1.12) {
58   if (!target || reduceMotion()) return;
59   animate(target, {
60     scale: [{ to: amount, duration: 110, ease: 'outQuad' }, { to: 1, ease: SPRING }],
61     onComplete: clean,
62   });
63 }
64
65 /** Columns and rows of a CSS grid, for grid staggers. */
```

```

66 export function gridSize(el: Element, count: number): [number, number] {
67   const cols = Math.max(1, getComputedStyle(el).gridTemplateColumns.split(' ').filter(Boolean).length);
68   return [cols, Math.max(1, Math.ceil(count / cols))];
69 }
70
71 const PRESSABLE = 'button:not(:disabled), a[href], [data-press]';
72
73 /**
74  * Press feedback for every button and link on the page through one
75  * delegated listener: a small squash on press and a spring back on release.
76  * index.css keeps CSS transitions off `transform` for these elements so they
77  * don't fight the spring. Returns a cleanup function.
78  */
79 export function installPressFeedback(root: Document = document): () => void {
80   let pressed: HTMLElement | null = null;
81
82   const release = () => {
83     if (!pressed) return;
84     animate(pressed, { scale: { to: 1, ease: SPRING }, onComplete: clean });
85     pressed = null;
86   };
87   const down = (e: PointerEvent) => {
88     if (e.button !== 0 || reduceMotion()) return;
89     const el = (e.target as Element | null)?.closest<HTMLElement>(PRESSABLE);
90     if (!el || el.closest('[data-no-press]')) return;
91     pressed = el;
92     // Big targets (poster cards, calendar cells) squash less.
93     const big = el.offsetWidth * el.offsetHeight > 20_000;
94     animate(el, { scale: big ? 0.98 : 0.95, duration: 120, ease: 'outQuad' });
95   };
96
97   root.addEventListener('pointerdown', down);
98   root.addEventListener('pointerup', release);
99   root.addEventListener('pointercancel', release);
100  root.addEventListener('dragstart', release);
101  return () => {
102    root.removeEventListener('pointerdown', down);
103    root.removeEventListener('pointerup', release);
104    root.removeEventListener('pointercancel', release);
105    root.removeEventListener('dragstart', release);
106  };
107 }

```

src/motion/hooks.ts

185 lignes

Des « hooks » React qui branchent ces animations sur les composants : entrée échelonnée, glissement entre les mois, nombres qui comptent, barre de progression, repère qui suit le bouton actif, ouverture et fermeture en accordéon.

```

1  // React bindings for presets.ts. Every hook animates in useLayoutEffect, so
2  // starting styles land before the first paint (no flash), and reverts its
3  // animation on cleanup, which also keeps StrictMode's double effects safe.
4
5  import { useLayoutEffect, useRef, type DependencyList, type RefObject } from 'react';
6  import { animate, utils, type JSAnimation } from 'animejs';
7  import { DURATION, EASE_OUT, enter, gridSize, reduceMotion, SPRING, type EnterOptions } from './presets';
8
9  /**
10   * Staggered entrance for `selector` matches inside `ref` (or `ref` itself)
11   * whenever `deps` change. `grid: true` staggers outward across the element's
12   * CSS grid instead of by index.
13   */
14  export function useEnter(
15    ref: RefObject<HTMLElement | null>,
16    deps: DependencyList,
17    { selector, grid, ...opts }: Omit<EnterOptions, 'grid'> & { selector?: string; grid?: boolean } = {},
18  ) {
19    useLayoutEffect(() => {
20      const root = ref.current;
21      if (!root) return;
22      const targets = selector ? [...root.querySelectorAll<HTMLElement>(selector)] : [root];
23      if (targets.length === 0) return;
24      const anim = enter(targets, { ...opts, ...(grid ? { grid: gridSize(root, targets.length) } : {}) });
25      return () => {
26        anim?.revert();

```

```

27     };
28     // eslint-disable-next-line react-hooks/exhaustive-deps
29   }, deps);
30 }
31
32 /** Like useEnter on `ref` itself, but skips the first render: for content that changes in place. */
33 export function useSwap(ref: RefObject<HTMLElement | null>, key: unknown, opts: EnterOptions = {}) {
34   const first = useRef(true);
35   useEffect(() => {
36     if (first.current) {
37       first.current = false;
38       return;
39     }
40     if (!ref.current) return;
41     const anim = enter(ref.current, { y: 6, duration: 260, ...opts });
42     return () => {
43       anim?.revert();
44     };
45     // eslint-disable-next-line react-hooks/exhaustive-deps
46   }, [key]);
47 }
48
49 /** Slides `ref` in from the side a numeric `position` moved toward (e.g. months). */
50 export function useSlide(ref: RefObject<HTMLElement | null>, position: number) {
51   const prev = useRef(position);
52   useEffect(() => {
53     const dir = Math.sign(position - prev.current);
54     prev.current = position;
55     if (!dir || !ref.current) return;
56     const anim = enter(ref.current, { x: dir * 24, y: 0, duration: 280 });
57     return () => {
58       anim?.revert();
59     };
60   }, [position, ref]);
61 }
62
63 /**
64  * Text that counts from its previous value to `value`. The element's text is
65  * owned by this hook: render the element without children.
66  */
67 export function useCountUp(
68   ref: RefObject<HTMLElement | null>,
69   value: number,
70   format: (n: number) => string = String,
71 ) {
72   const shown = useRef<number | null>(null);
73   const formatRef = useRef(format);
74   formatRef.current = format;
75
76   useEffect(() => {
77     const el = ref.current;
78     if (!el) return;
79     const write = (n: number) => {
80       shown.current = n;
81       el.textContent = formatRef.current(n);
82     };
83     const from = shown.current;
84     if (from == null || from === value || reduceMotion()) {
85       write(value);
86       return;
87     }
88     const state = { n: from };
89     const anim = animate(state, {
90       n: value,
91       duration: DURATION.slow,
92       ease: EASE_OUT,
93       onUpdate: () => write(Math.round(state.n)),
94       onComplete: () => write(value),
95     });
96     return () => {
97       anim.pause();
98       // Always leave the exact number behind, even if interrupted.
99       write(value);
100     };
101   }, [ref, value]);
102 }
103

```

```

104  /** Animates `ref`'s width to `percent`; its width is owned by this hook, so don't set it in JSX. */
105  export function useWidth(ref: RefObject<HTMLElement | null>, percent: number) {
106    const first = useRef(true);
107    useEffect(() => {
108      const el = ref.current;
109      if (!el) return;
110      if (first.current || reduceMotion()) {
111        first.current = false;
112        el.style.width = `${percent}%`;
113        return;
114      }
115      const anim: JSAnimation = animate(el, { width: `${percent}%`, duration: DURATION.slow, ease: EASE_OUT });
116      return () => {
117        anim.pause();
118        el.style.width = `${percent}%`;
119      };
120    }, [ref, percent]);
121  }
122
123  /**
124   * Moves an absolutely positioned indicator (`pillRef`) under the active
125   * item (`activeSelector` inside `rootRef`) with a spring, and keeps it
126   * there on resize.
127   */
128  export function useIndicator(
129    rootRef: RefObject<HTMLElement | null>,
130    pillRef: RefObject<HTMLElement | null>,
131    active: string,
132    activeSelector: string,
133  ) {
134    const placed = useRef(false);
135    useEffect(() => {
136      const root = rootRef.current;
137      const pill = pillRef.current;
138      if (!root || !pill) return;
139      const place = (animated: boolean) => {
140        const target = root.querySelector<HTMLElement>(activeSelector);
141        if (!target) return;
142        const to = { x: target.offsetLeft, width: target.offsetWidth };
143        if (!animated || reduceMotion()) {
144          utils.set(pill, to);
145          return;
146        }
147        animate(pill, { x: to.x, width: to.width, ease: SPRING });
148      };
149      place(placed.current);
150      placed.current = true;
151      const observer = new ResizeObserver(() => place(false));
152      observer.observe(root);
153      return () => observer.disconnect();
154    }, [rootRef, pillRef, active, activeSelector]);
155  }
156
157  /**
158   * Expands/collapses `ref` by animating its height between 0 and its content
159   * height. The first render (and reduced motion) jumps straight to the state.
160   * Once open it's left at `height: auto`, so content can still reflow.
161   */
162  export function useCollapse(ref: RefObject<HTMLElement | null>, open: boolean) {
163    const first = useRef(true);
164    useEffect(() => {
165      const el = ref.current;
166      if (!el) return;
167      const settle = () => {
168        el.style.height = open ? '' : '0px';
169        el.style.overflow = open ? '' : 'hidden';
170      };
171      if (first.current || reduceMotion()) {
172        first.current = false;
173        settle();
174        return;
175      }
176      const from = el.getBoundingClientRect().height;
177      const to = open ? el.scrollHeight : 0;
178      el.style.overflow = 'hidden';
179      const anim = animate(el, { height: [from, to], duration: 280, ease: EASE_OUT, onComplete: settle });
180      return () => {

```

```

181     anim.pause();
182     settle();
183   };
184   }, [ref, open]);
185 }

```

A.7 Les composants de l'interface

src/components/icons.tsx

109 lignes

Les petites icônes (loupe, flèches, coche, croix...) dessinées en SVG, avec des extrémités de trait carrées comme le reste du site. Elles prennent automatiquement la couleur du texte autour d'elles.

```

1  // Tiny inline stroke icons. Used instead of emoji/Unicode glyphs so the UI
2  // renders the same everywhere and inherits `currentColor`.
3
4  import type { ReactNode } from 'react';
5
6  interface IconProps {
7    className?: string;
8  }
9
10 function Svg({ className = 'h-4 w-4', children }: IconProps & { children: ReactNode }) {
11   return (
12     <svg
13       aria-hidden="true"
14       viewBox="0 0 24 24"
15       fill="none"
16       stroke="currentColor"
17       strokeWidth={2}
18       strokeLinecap="square"
19       strokeLinejoin="miter"
20       className={className}
21     >
22       {children}
23     </svg>
24   );
25 }
26
27 export function SearchIcon(props: IconProps) {
28   return (
29     <Svg {...props}>
30       <circle cx="11" cy="11" r="7" />
31       <path d="m20 20-3.5-3.5" />
32     </Svg>
33   );
34 }
35
36 export function ChevronLeftIcon(props: IconProps) {
37   return (
38     <Svg {...props}>
39       <path d="m15 18-6-6 6-6" />
40     </Svg>
41   );
42 }
43
44 export function CheckIcon(props: IconProps) {
45   return (
46     <Svg {...props}>
47       <path d="M20 6 9 17l-5-5" />
48     </Svg>
49   );
50 }
51
52 export function XIcon(props: IconProps) {
53   return (
54     <Svg {...props}>
55       <path d="M18 6 6 18m6 6l12 12" />
56     </Svg>
57   );
58 }
59
60 export function ExternalIcon(props: IconProps) {

```

```

61   return (
62     <Svg {...props}>
63       <path d="M14 4h6v6M20 4l-9 9M18 14v5a1 1 0 0 1-1 1H5a1 1 0 0 1-1-1V7a1 1 0 0 1 1-1h5" />
64     </Svg>
65   );
66 }
67
68 export function PlayIcon(props: IconProps) {
69   return (
70     <Svg {...props}>
71       <path d="M7 5v14l11-7z" />
72     </Svg>
73   );
74 }
75
76 export function ChevronRightIcon(props: IconProps) {
77   return (
78     <Svg {...props}>
79       <path d="m9 18 6-6-6-6" />
80     </Svg>
81   );
82 }
83
84 export function GridIcon(props: IconProps) {
85   return (
86     <Svg {...props}>
87       <rect x="4" y="4" width="6" height="6" rx="1" />
88       <rect x="14" y="4" width="6" height="6" rx="1" />
89       <rect x="4" y="14" width="6" height="6" rx="1" />
90       <rect x="14" y="14" width="6" height="6" rx="1" />
91     </Svg>
92   );
93 }
94
95 export function ListIcon(props: IconProps) {
96   return (
97     <Svg {...props}>
98       <path d="M9 6h11M9 12h11M9 18h11M4 6h.01M4 12h.01M4 18h.01" />
99     </Svg>
100   );
101 }
102
103 export function ChevronDownIcon(props: IconProps) {
104   return (
105     <Svg {...props}>
106       <path d="m6 9 6 6 6-6" />
107     </Svg>
108   );
109 }

```

src/components/SectionHeading.tsx

27 lignes

Le style de titre commun à toutes les sections, avec une zone optionnelle à droite (un compteur, un bouton).

```

1  import type { ReactNode } from 'react';
2
3  interface Props {
4    children: ReactNode;
5    /** Optional right-aligned content: a count, a note, or a small action. */
6    aside?: ReactNode;
7    /** `lg` for page sections, `sm` for panels like the configure sidebar. */
8    size?: 'lg' | 'sm';
9  }
10
11 /** The one heading style shared by every section, so pages stay consistent. */
12 export default function SectionHeading({ children, aside, size = 'lg' }: Props) {
13   return (
14     <div className="flex items-baseline justify-between gap-4">
15       <h2
16         className={
17           size === 'lg'
18             ? 'text-xl font-semibold tracking-tight text-(--ink) sm:text-2xl'
19             : 'text-sm font-semibold tracking-tight text-(--ink)'
20         >
21         {children}
22       </h2>
23       {aside}
24     </div>
25   );
26 }

```

```

22     {children}
23   </h2>
24   {aside && <div className="text-sm text-(--ink-2)">{aside}</div>}
25 </div>
26 );
27 }

```

src/components/Motion.tsx

126 lignes

Des composants animés réutilisables : un nombre qui compte (`CountUp`), un indicateur de chargement (`Spinner`), des cases de chargement qui ondulent (`Shimmer`) et l'effet « glitch » du titre de l'en-tête (`GlitchText`).

```

1  import { useEffect, useLayoutEffect, useRef, type ReactNode } from 'react';
2  import { animate, createTimeline, stagger, steps, type Timeline } from 'animejs';
3  import { reduceMotion } from '../motion/presets';
4  import { useCountUp } from '../motion/hooks';
5
6  /** A number that counts to its new value when it changes. */
7  export function CountUp({
8    value,
9    format,
10   className,
11 }: {
12   value: number;
13   format?: (n: number) => string;
14   className?: string;
15 }) {
16   const ref = useRef<HTMLSpanElement>(null);
17   useCountUp(ref, value, format);
18   return <span ref={ref} className={className} />;
19 }
20
21 export function Spinner() {
22   const ref = useRef<HTMLSpanElement>(null);
23   useLayoutEffect(() => {
24     if (!ref.current || reduceMotion()) return;
25     const anim = animate(ref.current, { rotate: 360, duration: 800, ease: 'linear', loop: true });
26     return () => {
27       anim.revert();
28     };
29   }, []);
30   return (
31     <span
32       ref={ref}
33       className="h-8 w-8 border-2 border-(--line) border-t-(--accent)"
34     />
35   );
36 }
37
38 /** Loading placeholder: children marked `data-shimmer` pulse in a gentle wave. */
39 export function Shimmer({ children, className }: { children: ReactNode; className?: string }) {
40   const ref = useRef<HTMLUListElement>(null);
41   useLayoutEffect(() => {
42     if (!ref.current || reduceMotion()) return;
43     const anim = animate(ref.current.querySelectorAll('[data-shimmer]'), {
44       opacity: [1, 0.4],
45       duration: 900,
46       delay: stagger(40),
47       ease: 'inOutSine',
48       loop: true,
49       alternate: true,
50     });
51     return () => {
52       anim.revert();
53     };
54   }, []);
55   return (
56     <ul ref={ref} className={className} aria-hidden="true">
57       {children}
58     </ul>
59   );
60 }
61
62 const SLICES_A = ['inset(10% 0 60% 0)', 'inset(40% 0 30% 0)', 'inset(70% 0 5% 0)', 'inset(20% 0 50% 0)'];

```

```

63 const SLICES_B = ['inset(55% 0 15% 0)', 'inset(25% 0 45% 0)', 'inset(5% 0 75% 0)', 'inset(45% 0 25% 0)'];
64 const JITTER = [
65   [-2, 1],
66   [2, -1],
67   [-2, 0],
68   [2, 1],
69 ];
70
71 function burst(slices: string[], flip: number) {
72   return [
73     ...slices.map((clipPath, i) => ({
74       opacity: 0.8,
75       clipPath,
76       x: JITTER[i][0] * flip,
77       y: JITTER[i][1] * flip,
78       duration: 100,
79     })),
80     { opacity: 0, x: 0, y: 0, duration: 100 },
81   ];
82 }
83
84 /**
85  * Text with a color-split glitch that plays two short bursts when its
86  * closest button is hovered or keyboard-focused. The copies are real
87  * aria-hidden spans (anime.js can't reach ::before/::after).
88  */
89 export function GlitchText({ text }: { text: string }) {
90   const rootRef = useRef<HTMLSpanElement>(null);
91   const aRef = useRef<HTMLSpanElement>(null);
92   const bRef = useRef<HTMLSpanElement>(null);
93
94   useEffect(() => {
95     const trigger = rootRef.current?.closest('button') ?? rootRef.current;
96     if (!trigger) return;
97     let running: Timeline | null = null;
98     const play = (e: Event) => {
99       if (running || reduceMotion()) return;
100       if (e.type === 'focus' && !(e.currentTarget as Element).matches(':focus-visible')) return;
101       running = createTimeline({ defaults: { ease: steps(2) }, loop: 1, onComplete: () => (running = null) })
102         .add(aRef.current!, { keyframes: burst(SLICES_A, 1) }, 0)
103         .add(bRef.current!, { keyframes: burst(SLICES_B, -1) }, 0);
104     };
105     trigger.addEventListener('mouseenter', play);
106     trigger.addEventListener('focus', play);
107     return () => {
108       trigger.removeEventListener('mouseenter', play);
109       trigger.removeEventListener('focus', play);
110       running?.revert();
111     };
112   }, []);
113
114   const copy = 'pointer-events-none absolute inset-0 whitespace-nowrap';
115   return (
116     <span ref={rootRef} className="relative inline-block">
117       {text}
118       <span ref={aRef} aria-hidden="true" className={` ${copy} text(--orange)`} style={{ opacity: 0 }}>
119         {text}
120       </span>
121       <span ref={bRef} aria-hidden="true" className={` ${copy} text(--blue)`} style={{ opacity: 0 }}>
122         {text}
123       </span>
124     </span>
125   );
126 }

```

src/components/SearchBar.tsx

58 lignes

Le champ de recherche, en deux tailles : grand au centre de la page d'accueil, compact dans l'en-tête des autres pages.

```

1 import type { FormEvent } from 'react';
2 import { SearchIcon } from './icons';
3
4 interface Props {

```

```

5   query: string;
6   onQueryChange: (query: string) => void;
7   onSearch: (term: string) => void;
8   loading: boolean;
9   /** `hero` is the large homepage field; `header` is the compact one on other pages. */
10  size?: 'hero' | 'header';
11  /** Mark the form for a parent's `useEnter` stagger. */
12  enter?: boolean;
13 }
14
15 export default function SearchBar({ query, onQueryChange, onSearch, loading, size = 'header', enter }: Props) {
16   const hero = size === 'hero';
17   const handleSubmit = (e: FormEvent) => {
18     e.preventDefault();
19     onSearch(query);
20   };
21
22   return (
23     <form
24       onSubmit={handleSubmit}
25       role="search"
26       data-enter={enter || undefined}
27       className={`flex min-w-0 items-center bg(--surface) ring-1 ring(--line) transition-shadow focus-
within:ring-2 focus-within:ring(--accent)} ${
28       hero
29         ? 'h-14 w-full gap-3 pl-5 pr-1.5 shadow-2xl shadow(--navy)/10 sm:h-16 sm:pl-6 sm:pr-2'
30         : 'h-11 max-w-lg flex-1 gap-3 px-4 sm:h-12 sm:px-5'
31     }`
32   >
33     <SearchIcon className={`shrink-0 text(--ink-2)} ${hero ? 'h-5 w-5 sm:h-6 sm:w-6' : 'h-5 w-5'}` />
34     <label htmlFor={`search-${size}`} className="sr-only">
35       Search anime
36     </label>
37     /** text-base minimum on phones: iOS zooms the page into inputs under 16px. */
38     <input
39       id={`search-${size}`}
40       type="search"
41       value={query}
42       onChange={(e) => onQueryChange(e.target.value)}
43       placeholder={loading ? 'Searching...' : hero ? 'Search any anime' : 'Search anime'}
44       className={`min-w-0 flex-1 bg-transparent placeholder:text(--ink-3) focus:outline-none ${
45       hero ? 'text-base sm:text-lg' : 'text-base sm:text-[15px]'
46     }`
47     />
48     {hero && (
49       <button
50         type="submit"
51         className="h-11 shrink-0 bg(--accent) px-5 text-sm font-semibold text(--on-accent) transition
52         hover:brightness-90 sm:h-12 sm:px-7 sm:text-base"
53       >
54         Search
55       </button>
56     )}
57   </form>
58 );
59 }

```

src/components/AnimeCard.tsx

51 lignes

La carte d'une série dans la grille : l'affiche, la note en pourcentage, le titre, le format et l'année. Au survol, une lueur reprend la couleur de l'affiche.

```

1   import type { Anime } from '../types';
2   import { displayTitle, formatLabel } from '../lib/format';
3
4   interface Props {
5     anime: Anime;
6     onSelect: (anime: Anime) => void;
7     tabIndex?: number;
8   }
9
10  /**
11   * Poster card: the image carries the card, with just the score overlaid and
12   * a title + meta underneath. The hover glow uses the show's own cover color,

```

```

13  * independent of the page theme.
14  */
15  export default function AnimeCard({ anime, onSelect, tabIndex }: Props) {
16    const meta = [formatLabel(anime.format), anime.seasonYear].filter(Boolean).join(' · ');
17    const title = displayTitle(anime);
18
19    return (
20      <button
21        type="button"
22        onClick={() => onSelect(anime)}
23        tabIndex={tabIndex}
24        title={title}
25        style={{ ['--card-accent' as string]: anime.coverImage.color ?? 'var(--accent)' }}
26        className="group flex w-full flex-col gap-3 text-left focus-visible:outline-none"
27      >
28        <div className="relative aspect-[2/3] w-full overflow-hidden bg(--surface-2) ring-1 ring(--line) transition
duration-300 group-hover:-translate-y-1 group-hover:shadow-xl group-hover:shadow-[color-mix(in_srgb,var(--card-
accent)_35%,transparent)] group-hover:ring-[color-mix(in_srgb,var(--card-accent)_70%,transparent)] group-focus-
visible:ring-2 group-focus-visible:ring(--accent)">
29          {anime.coverImage.large && (
30            <img
31              src={anime.coverImage.large}
32              alt=""
33              loading="lazy"
34              className="h-full w-full object-cover transition duration-500 group-hover:scale-[1.04]"
35            />
36          )}
37          {anime.averageScore != null && (
38            <span className="absolute right-2 top-2 bg(--navy)/85 px-2 py-0.5 text-xs font-semibold tabular-nums
text-white backdrop-blur">
39              {anime.averageScore}%
40            </span>
41          )}
42        </div>
43        <div className="flex min-w-0 flex-col gap-0.5 px-0.5">
44          <p className="line-clamp-2 text-sm font-medium leading-snug text(--ink) group-hover:text(--accent-text)
sm:text-[15px]">
45            {title}
46          </p>
47          <p className="truncate text-xs text(--ink-3) sm:text-sm">{meta}</p>
48        </div>
49      </button>
50    );
51  }

```

src/components/AnimeListRow.tsx

40 lignes

La même série présentée en ligne, pour l'affichage en liste.

```

1  import type { Anime } from '../types';
2  import { displayTitle, formatLabel } from '../lib/format';
3
4  interface Props {
5    anime: Anime;
6    onSelect: (anime: Anime) => void;
7  }
8
9  /** List-view counterpart of `AnimeCard`: one scannable row per show. */
10 export default function AnimeListRow({ anime, onSelect }: Props) {
11   const title = displayTitle(anime);
12   const meta = [formatLabel(anime.format), anime.seasonYear, anime.studios[0]?.name].filter(Boolean).join(' · ');
13
14   return (
15     <button
16       type="button"
17       onClick={() => onSelect(anime)}
18       title={title}
19       className="flex w-full items-center gap-4 p-2 text-left transition hover:bg(--surface) focus-
visible:outline-none focus-visible:ring-2 focus-visible:ring(--accent) sm:gap-5 sm:p-3"
20     >
21       <div className="h-20 w-14 shrink-0 overflow-hidden bg(--surface-2) ring-1 ring(--line) sm:h-24 sm:w-16">
22         {anime.coverImage.large && (
23           <img src={anime.coverImage.large} alt="" loading="lazy" className="h-full w-full object-cover" />
24         )}
25       </div>

```

```

26     <div className="flex min-w-0 flex-1 flex-col gap-1">
27       <p className="line-clamp-2 text-base font-medium leading-snug text-(--ink)">{title}</p>
28       <p className="truncate text-sm text-(--ink-2)">{meta}</p>
29       {anime.genres.length > 0 && (
30         <p className="hidden truncate text-xs text-(--ink-3) sm:block">{anime.genres.slice(0, 4).join(' · ')}</p>
31       )}
32     </div>
33     {anime.averageScore != null && (
34       <span className="shrink-0 bg-(--surface-2) px-2.5 py-1 text-xs font-semibold tabular-nums text-(--ink)
ring-1 ring-(--line)">
35         {anime.averageScore}%
36       </span>
37     )}
38   </button>
39 );
40 }

```

src/components/SavedSchedules.tsx

122 lignes

La rangée « Continue watching » de la page d'accueil : une carte par planning sauvegardé, avec le jour où reprendre, la barre de progression (dans la couleur de la série), le prochain épisode à sortir et un bouton pour supprimer (avec une animation de fermeture).

```

1  import { useRef, type MouseEvent } from 'react';
2  import { createTimeline } from 'animejs';
3  import type { SavedSchedule } from '../types';
4  import { useEnter } from '../motion/hooks';
5  import { reduceMotion } from '../motion/presets';
6  import { displayTitle, formatMinutes } from '../lib/format';
7  import { generateSchedule } from '../lib/schedule';
8  import { posterTheme } from '../lib/theme';
9  import { getCurrentStreak, getProgress, getResumeDay } from '../lib/progress';
10 import { formatAiring, formatDate, scheduleDates, todayIso, upcomingAirings } from '../lib/calendar';
11 import SectionHeading from './SectionHeading';
12 import { XIcon } from './icons';
13
14 interface Props {
15   items: SavedSchedule[];
16   onOpen: (saved: SavedSchedule) => void;
17   onDelete: (id: string) => void;
18 }
19
20 /** "Continue watching": one horizontally scrolling, swipe-snapping row. */
21 export default function SavedSchedules({ items, onOpen, onDelete }: Props) {
22   const streak = getCurrentStreak(items);
23   const listRef = useRef<HTMLListElement>(null);
24   // On arrival only: re-staggering after a delete would look like a reload.
25   useEnter(listRef, [], { selector: ':scope > li', step: 60, x: 24, y: 0 });
26
27   // Fades and collapses the card, then removes it, so the row closes up smoothly.
28   const handleDelete = (e: MouseEvent<HTMLButtonElement>, id: string) => {
29     const card = e.currentTarget.closest('li');
30     if (!card || reduceMotion()) {
31       onDelete(id);
32       return;
33     }
34     card.style.pointerEvents = 'none';
35     createTimeline({ onComplete: () => onDelete(id) })
36       .add(card, { opacity: 0, scale: 0.92, duration: 180, ease: 'inQuad' })
37       // Negative margin eats the list's gap too, so neighbours end where they'll stay.
38       .add(card, {
39         width: 0,
40         marginRight: -parseFloat(getComputedStyle(card.parentElement!).columnGap || '0'),
41         duration: 220,
42         ease: 'inOutQuad',
43       });
44   };
45
46   return (
47     <section className="flex flex-col gap-6">
48       <SectionHeading aside={streak > 0 ? `${streak}-day streak` : undefined}>
49         Continue watching
50       </SectionHeading>
51

```

```

52     <ul ref={listRef} className="-mx-1 flex snap-x snap-mandatory gap-4 overflow-x-auto px-1 pb-2 [scrollbar-
width:thin] sm:gap-6">
53         {items.map((saved) => {
54             const { anime, config } = saved;
55             const title = displayTitle(anime);
56             const result = generateSchedule(anime, config);
57             const days = result.ok ? result.schedule.days : [];
58             const progress = getProgress(days, saved.watched);
59             const resumeDay = getResumeDay(days, saved.watched);
60             const resumeDate = resumeDay ? scheduleDates(days, saved.calendar).get(resumeDay) : undefined;
61             // The saved snapshot's nextAiringEpisode; a stale one is filtered out.
62             const timezone = saved.calendar.timezone;
63             const today = todayIso(undefined, timezone);
64             const nextAiring = upcomingAirings(anime, undefined, timezone)[0];
65
66             return (
67                 <li key={saved.id} className="group relative w-[85%] shrink-0 snap-start sm:w-80">
68                     <button
69                         type="button"
70                         onClick={() => onOpen(saved)}
71                         className="flex w-full items-center gap-5 bg(--surface) p-4 pr-12 text-left ring-1 ring(--line)
transition hover:bg(--surface-2) focus-visible:outline-none focus-visible:ring-2 focus-visible:ring(--accent)"
72                     >
73                         <div className="h-24 w-16 shrink-0 overflow-hidden bg(--surface-2)">
74                             {anime.coverImage.large && (
75                                 <img src={anime.coverImage.large} alt="" className="h-full w-full object-cover" />
76                             )}
77                         </div>
78                         <div className="flex min-w-0 flex-1 flex-col gap-1">
79                             <p className="line-clamp-2 text-base font-medium leading-snug text(--ink)">{title}</p>
80                             <p className="text-sm text(--ink-2)">
81                                 {days.length === 0
82                                     ? `${formatMinutes(config.dailyBudget)}/day`
83                                     : resumeDay
84                                     ? `Day ${resumeDay} of ${days.length} · ${
85                                         resumeDate === today ? 'today' : resumeDate ? formatDate(resumeDate) : ''
86                                     }`
87                                     : 'All watched'}
88                             </p>
89                             {nextAiring && (
90                                 <p className="text-xs text(--accent-text)">
91                                     New ep {nextAiring.episode} · {formatAiring(nextAiring.airingAt, timezone).day}
92                                 </p>
93                             )}
94                             {days.length > 0 && (
95                                 <div className="mt-2 h-1.5 overflow-hidden bg(--surface-2)">
96                                     <div
97                                         className="h-full"
98                                         style={{
99                                             width: `${progress.percent}%`,
100                                             // The same color the show's own page uses.
101                                             backgroundColor: posterTheme(anime.coverImage.color)?.accent ?? 'var(--accent)',
102                                         }}
103                                     />
104                                 </div>
105                             )}
106                         </div>
107                     </button>
108                     <button
109                         type="button"
110                         onClick={(e) => handleDelete(e, saved.id)}
111                         aria-label={`Remove ${title} from saved schedules`}
112                         className="absolute right-2 top-2 grid h-9 w-9 place-items-center text(--ink-3) transition
hover:bg-red-600/10 hover:text-red-700 pointer-fine:opacity-0 pointer-fine:group-hover:opacity-100 pointer-
fine:focus-visible:opacity-100"
113                     >
114                         <XIcon className="h-4 w-4" />
115                     </button>
116                 </li>
117             );
118         })}
119     </ul>
120 </section>
121 );
122 }

```

La page d'accueil et les résultats de recherche : le grand titre et la recherche, les plannings sauvegardés, la grille « Popular right now », les cases de chargement, et le bouton pour passer de la grille à la liste.

```

1  import { useRef, useState } from 'react';
2  import type { Anime, SavedSchedule } from '../types';
3  import { useEnter } from '../motion/hooks';
4  import { Shimmer } from './Motion';
5  import AnimeCard from './AnimeCard';
6  import AnimeListRow from './AnimeListRow';
7  import { GridIcon, ListIcon } from './icons';
8  import SavedSchedules from './SavedSchedules';
9  import SectionHeading from './SectionHeading';
10 import SearchBar from './SearchBar';
11
12 interface Props {
13   /** The term the current results are for (null when not searching). */
14   searchedFor: string | null;
15   results: Anime[] | null;
16   loading: boolean;
17   error: string | null;
18   onSelect: (anime: Anime) => void;
19   onClearSearch: () => void;
20   query: string;
21   onQueryChange: (query: string) => void;
22   onSearch: (term: string) => void;
23   popular: Anime[] | null;
24   savedSchedules: SavedSchedule[];
25   onOpenSaved: (saved: SavedSchedule) => void;
26   onDeleteSaved: (id: string) => void;
27 }
28
29 // 2 / 3 / 6 columns: every step divides the 18-item popular list evenly.
30 const GRID = 'grid grid-cols-2 gap-x-4 gap-y-8 sm:grid-cols-3 sm:gap-x-6 sm:gap-y-10 lg:grid-cols-6';
31
32 // List view: one row per show, two columns once there's room.
33 const LIST = 'grid grid-cols-[minmax(0,1fr)] gap-1 lg:grid-cols-2 lg:gap-x-8';
34
35 type View = 'grid' | 'list';
36
37 function AnimeGrid({ items, onSelect, view }: { items: Anime[]; onSelect: (a: Anime) => void; view: View }) {
38   const ref = useRef<HTMLUListElement>(null);
39   const list = view === 'list';
40   useEnter(
41     ref,
42     [items, view],
43     list ? { selector: ':scope > li', step: 30, y: 12 } : { selector: ':scope > li', grid: true, step: 45, y: 20 },
44   );
45   return (
46     <ul ref={ref} className={list ? LIST : GRID}>
47       {items.map((anime) => (
48         <li key={anime.id} className="flex">
49           {list ? <AnimeListRow anime={anime} onSelect={onSelect} /> : <AnimeCard anime={anime} onSelect={onSelect}
50         </li>
51       )}}
52     </ul>
53   );
54 }
55
56 function ViewToggle({ view, onChange }: { view: View; onChange: (view: View) => void }) {
57   return (
58     <div role="group" aria-label="Layout" className="flex gap-1 bg(--surface) p-1 ring-1 ring(--line)">
59       {(
60         [
61           ['grid', 'Grid view', GridIcon],
62           ['list', 'List view', ListIcon],
63         ] as const
64       ).map(([v, label, Icon]) => (
65         <button
66           key={v}
67           type="button"
68           aria-pressed={view === v}
69           aria-label={label}

```

```

70     title={label}
71     onClick={() => onChange(v)}
72     className={`grid h-8 w-8 place-items-center transition ${
73       view === v ? 'bg(--accent) text(--on-accent)' : 'text(--ink-2) hover:text(--ink)'
74     }`}
75   >
76     <Icon className="h-4 w-4" />
77   </button>
78   )}
79 </div>
80 );
81 }
82
83 function SkeletonGrid() {
84   return (
85     <Shimmer className={GRID}>
86       {Array.from({ length: 12 }, (_, i) => (
87         <li key={i} data-shimmer className="flex flex-col gap-3">
88           <div className="aspect-[2/3] bg(--surface-2)" />
89           <div className="h-3 w-3/4 bg(--surface-2)" />
90           <div className="h-3 w-1/2 bg(--surface-2)" />
91         </li>
92       ))}
93     </Shimmer>
94   );
95 }
96
97 /** Homepage: a search-first hero plus rows of anime, or search results in the same grid. */
98 export default function SearchStep({
99   searchedFor,
100   results,
101   loading,
102   error,
103   onSelect,
104   onClearSearch,
105   query,
106   onQueryChange,
107   onSearch,
108   popular,
109   savedSchedules,
110   onOpenSaved,
111   onDeleteSaved,
112 }: Props) {
113   const searching = loading || results !== null || error !== null;
114   // A browsing preference only, so it lives here and resets on reload.
115   const [view, setView] = useState<View>('grid');
116   const heroRef = useRef<HTMLElement>(null);
117   useEnter(heroRef, [searching], { selector: '[data-enter]', step: 70, y: 18, duration: 520 });
118
119   if (searching) {
120     return (
121       <section className="flex flex-col gap-8">
122         <SectionHeading
123           aside={
124             <div className="flex items-center gap-3">
125               <ViewToggle view={view} onChange={setView} />
126               <button
127                 type="button"
128                 onClick={onClearSearch}
129                 className="px-4 py-1.5 ring-1 ring(--line) transition hover:bg(--surface) hover:text(--ink)"
130               >
131                 Clear
132               </button>
133             </div>
134           >
135         {searchedFor ? `Results for "${searchedFor}"` : 'Results'}
136       </SectionHeading>
137
138       {error && (
139         <p role="alert" className="bg-red-50 px-5 py-4 text-sm text-red-800 ring-1 ring-red-200">
140           {error}
141         </p>
142       )}
143
144       {loading && <SkeletonGrid />}
145       {!loading && results?.length === 0 && (
146         <p className="bg(--surface) px-6 py-12 text-center text-base text(--ink-2) ring-1 ring(--line)">

```

```

147         No anime found for that search.
148     </p>
149   })
150   {!!loading && results && results.length > 0 && <AnimeGrid items={results} onSelect={onSelect} view={view}
/>}
151   </section>
152   );
153   }
154
155   return (
156     <div className="flex flex-col gap-16 sm:gap-24">
157       <section
158         ref={heroRef}
159         className="mx-auto flex w-full max-w-3xl flex-col items-center gap-6 pt-6 text-center sm:gap-8 sm:pt-12
lg:pt-16"
160       >
161         <div className="flex flex-col items-center gap-4 sm:gap-5">
162           <h1 data-enter className="text-4xl font-semibold tracking-tight text-balance text(--ink) sm:text-6xl">
163             Plan your next binge.
164           </h1>
165           <p data-enter className="max-w-xl text-base text-pretty text(--ink-2) sm:text-lg">
166             Tell us how much time you have. Get a day-by-day watch plan with real episode titles,
167             on your calendar.
168           </p>
169         </div>
170         <SearchBar
171           enter
172           size="hero"
173           query={query}
174           onChange={onChange}
175           search={search}
176           loading={loading}
177         />
178       </section>
179
180       {savedSchedules.length > 0 && (
181         <SavedSchedules items={savedSchedules} onOpen={onOpenSaved} onDelete={onDeleteSaved} />
182       )}
183
184       {/* null = still loading; [] = failed or empty, so the section is hidden. */}
185       {(popular === null || popular.length > 0) && (
186         <section className="flex flex-col gap-6">
187           <SectionHeading aside={<ViewToggle view={view} onChange={setView} />}>Popular right now</SectionHeading>
188           {popular === null ? <SkeletonGrid /> : <AnimeGrid items={popular} onSelect={onSelect} view={view} />}
189         </section>
190       )}
191     </div>
192   );
193 }

```

src/components/ConfigForm.tsx

143 lignes

Le formulaire « Your plan » : le temps par jour en minutes ou en épisodes, les boutons de valeurs rapides et la case pour sauter les génériques. Le mode « épisodes » est converti en minutes exactes, donc le moteur ne voit qu'un seul réglage.

```

1  import { useState } from 'react';
2  import type { ScheduleConfig } from '../types';
3
4  const BUDGET_PRESETS = [30, 60, 90, 120, 180];
5  const EPISODE_PRESETS = [1, 2, 3, 4, 6];
6
7  type Mode = 'time' | 'episodes';
8
9  interface Props {
10    config: ScheduleConfig;
11    onChange: (config: ScheduleConfig) => void;
12    isMovie: boolean;
13    /**
14     * Minutes per episode (`getNetEpisodeRuntime`) with and without the OP/ED
15     * trim, or null when unknown. Needed to plan by episode count.
16     */
17    runtimes: { trimmed: number | null; full: number | null };
18  }

```

```

19
20 /**
21  * Daily plan, set either as minutes or as episodes per day. Episodes are an
22  * input convenience only: they're stored as the exact minutes they take
23  * (`episodes * netRuntime`), so the engine and saved data only ever see
24  * `dailyBudget`.
25  */
26 export default function ConfigForm({ config, onChange, isMovie, runtimes }: Props) {
27   const netRuntime = config.skipOpEd ? runtimes.trimmed : runtimes.full;
28   const [mode, setMode] = useState<Mode>('time');
29   const canUseEpisodes = !isMovie && netRuntime !== null && netRuntime > 0;
30   const byEpisodes = canUseEpisodes && mode === 'episodes';
31
32   const set = <K extends keyof ScheduleConfig>(key: K, value: ScheduleConfig[K]) =>
33     onChange({ ...config, [key]: value });
34   const setEpisodes = (n: number) => set('dailyBudget', Number.isFinite(n) ? n * netRuntime! : n);
35
36   const episodes = Number.isFinite(config.dailyBudget) && netRuntime
37     ? Math.max(1, Math.floor(config.dailyBudget / netRuntime))
38     : NaN;
39   const value = byEpisodes ? episodes : config.dailyBudget;
40   const presets = byEpisodes ? EPISODE_PRESETS : BUDGET_PRESETS;
41
42   // Switching to episodes snaps the budget to a whole number of episodes,
43   // so the input and the schedule agree.
44   const switchMode = (next: Mode) => {
45     setMode(next);
46     if (next === 'episodes' && Number.isFinite(episodes)) setEpisodes(episodes);
47   };
48
49   return (
50     <div className="flex flex-col gap-5">
51       {canUseEpisodes && (
52         <div role="group" aria-label="Plan by" className="grid grid-cols-2 gap-1 bg(--bg)/50 p-1 ring-1 ring(--
53         line)">
54           {(
55             [
56               ['time', 'By time'],
57               ['episodes', 'By episodes'],
58             ] as const
59           ).map(([m, label]) => (
60             <button
61               key={m}
62               type="button"
63               aria-pressed={mode === m}
64               onClick={() => switchMode(m)}
65               className={`h-8 text-sm transition ${
66                 mode === m ? 'bg(--surface-2) font-medium text(--ink)' : 'text(--ink-2) hover:text(--ink)'
67               }`}
68             >
69               {label}
70             </button>
71           ))}
72         </div>
73       )}
74
75       <div className="flex items-center justify-between gap-3">
76         <label htmlFor="budget" className="text-sm text(--ink-2)">
77           {byEpisodes ? 'Episodes per day' : 'Daily watch time'}
78         </label>
79         <div className="flex items-center gap-2">
80           <input
81             id="budget"
82             type="number"
83             min={1}
84             step={byEpisodes ? 1 : 5}
85             value={Number.isFinite(value) ? value : ''}
86             onChange={(e) =>
87               byEpisodes ? setEpisodes(Math.round(e.target.valueAsNumber)) : set('dailyBudget',
88               e.target.valueAsNumber)
89             }
90             className="h-10 w-20 bg(--bg) px-3 text-right text-base tabular-nums ring-1 ring(--line)
91             focus:outline-none focus:ring-2 focus:ring(--accent)"
92             />
93           <span className="w-6 text-sm text(--ink-3)">{byEpisodes ? 'ep' : 'min'}</span>
94         </div>
95       </div>

```

```

93
94     <div className="grid grid-cols-5 gap-2">
95       {presets.map((n) => {
96         const active = value === n;
97         return (
98           <button
99             key={n}
100             type="button"
101             onClick={() => (byEpisodes ? setEpisodes(n) : set('dailyBudget', n))}
102             aria-pressed={active}
103             className={`h-9 text-sm tabular-nums ring-1 transition ${
104               active
105                 ? 'bg(--accent) font-semibold text(--on-accent) ring(--accent)'
106                 : 'text(--ink-2) ring(--line) hover:bg(--surface-2) hover:text(--ink)'}
107             `}
108           >
109             {n}
110           </button>
111         );
112       })}
113     </div>
114     {byEpisodes && Number.isFinite(config.dailyBudget) && (
115       <p className="-mt-2 text-xs text(--ink-3)">
116         About {config.dailyBudget} min a day at {netRuntime} min per episode.
117       </p>
118     )}
119
120     <label
121       className={`flex items-center gap-3 bg(--bg)/50 px-3 py-3 text-sm ring-1 ring(--line) ${isMovie ?
122 'opacity-50' : 'cursor-pointer text(--ink-2)}`}
123       title="Trims 3 minutes per episode. Not applied to movies or episodes under 15 minutes."
124     >
125       <input
126         type="checkbox"
127         checked={config.skipOpEd}
128         disabled={isMovie}
129         onChange={(e) => {
130           // In episode mode, keep the episode count when the runtime changes.
131           const nextRuntime = e.target.checked ? runtimes.trimmed : runtimes.full;
132           if (byEpisodes && Number.isFinite(episodes) && nextRuntime) {
133             onChange({ ...config, skipOpEd: e.target.checked, dailyBudget: episodes * nextRuntime });
134           } else {
135             set('skipOpEd', e.target.checked);
136           }
137         }}
138         className="h-4 w-4 shrink-0"
139       />
140       Skip openings and endings
141     </label>
142   </div>
143 );
144 }

```

src/components/WatchPrefsForm.tsx

189 lignes

Le formulaire « When you watch » : les jours de la semaine, l'heure de visionnage, le rappel et le fuseau horaire.

```

1  import type { CalendarPrefs } from '../types';
2  import { pop } from '../motion/presets';
3  import {
4    ALL_DAYS,
5    REMINDER_OPTIONS,
6    WEEKDAYS,
7    WEEKENDS,
8    addDays,
9    deviceTimeZone,
10   formatDate,
11   listTimeZones,
12 } from '../lib/calendar';
13
14 interface Props {
15   prefs: CalendarPrefs;
16   onChange: (prefs: CalendarPrefs) => void;
17 }
18

```

```

19  /** 2026-09-28 is a Monday: labels come from Intl so they follow the locale. */
20  const MONDAY = '2026-09-28';
21  const WEEK = Array.from({ length: 7 }, (_, i) => {
22    const iso = addDays(MONDAY, i);
23    return {
24      day: (i + 1) % 7,
25      short: formatDate(iso, { weekday: 'narrow' }),
26      long: formatDate(iso, { weekday: 'long' }),
27    };
28  });
29
30  const PRESETS = [
31    { label: 'Every day', days: ALL_DAYS },
32    { label: 'Weekdays', days: WEEKDAYS },
33    { label: 'Weekends', days: WEEKENDS },
34  ];
35
36  function reminderLabel(minutes: number | null): string {
37    if (minutes == null) return 'No reminder';
38    if (minutes === 0) return 'At start time';
39    return minutes === 60 ? '1 hour before' : `${minutes} min before`;
40  }
41
42  /** "America/New_York" -> "America / New York". */
43  function zoneLabel(zone: string): string {
44    return zone.replace(/_/g, ' ').replace(/\\/g, ' / ');
45  }
46
47  const sameDays = (a: number[], b: number[]) =>
48    a.length === b.length && b.every((d) => a.includes(d));
49
50  export default function WatchPrefsForm({ prefs, onChange }: Props) {
51    const device = deviceTimeZone();
52    const set = <K extends keyof CalendarPrefs>(key: K, value: CalendarPrefs[K]) =>
53      onChange({ ...prefs, [key]: value });
54
55    const toggleDay = (day: number, el: HTMLElement) => {
56      const on = prefs.watchDays.includes(day);
57      // The last watch day can't be turned off: a plan needs at least one.
58      if (on && prefs.watchDays.length === 1) return;
59      if (!on) pop(el);
60      set('watchDays', on ? prefs.watchDays.filter((d) => d !== day) : [...prefs.watchDays, day].sort((a, b) => a -
61        b));
62    };
63
64    return (
65      <div className="flex flex-col gap-5">
66        <fieldset className="flex flex-col gap-3">
67          <legend className="mb-3 text-sm text-(--ink-2)">Days you watch</legend>
68          <div className="grid grid-cols-7 gap-1.5">
69            {WEEK.map(({ day, short, long }) => {
70              const on = prefs.watchDays.includes(day);
71              return (
72                <button
73                  key={day}
74                  type="button"
75                  onClick={(e) => toggleDay(day, e.currentTarget)}
76                  aria-pressed={on}
77                  aria-label={long}
78                  title={long}
79                  className={`h-9 text-sm font-medium ring-1 transition ${
80                    on
81                      ? 'bg-(--accent) text-(--on-accent) ring-(--accent)'
82                      : 'text-(--ink-2) ring-(--line) hover:bg-(--surface-2) hover:text-(--ink)'}
83                >
84                  {short}
85                </button>
86              );
87            })}
88          </div>
89          <div className="flex flex-wrap gap-2">
90            {PRESETS.map((p) => {
91              const active = sameDays(prefs.watchDays, p.days);
92              return (
93                <button
94                  key={p.label}

```

```

95         type="button"
96         onClick={() => set('watchDays', [...p.days])}
97         aria-pressed={active}
98         className={`px-3 py-1 text-xs ring-1 transition ${
99             active
100             ? 'bg(--surface-2) text(--ink) ring(--accent)'
101             : 'text(--ink-2) ring(--line) hover:text(--ink)'}
102         `}
103     >
104         {p.label}
105     </button>
106 );
107 }}}
108 </div>
109 </fieldset>
110
111 <div className="flex items-center justify-between gap-3">
112     <label htmlFor="watch-time" className="text-sm text(--ink-2)">
113         Watch time
114     </label>
115     <div className="flex items-center gap-2">
116         <input
117             id="watch-time"
118             type="time"
119             value={prefs.watchTime ?? ''}
120             onChange={(e) =>
121                 onChange({
122                     ...prefs,
123                     watchTime: e.target.value || null,
124                     reminderMinutes: e.target.value ? prefs.reminderMinutes : null,
125                 })
126             }
127             className="h-10 bg(--bg) px-3 text-base text(--ink) ring-1 ring(--line) focus:outline-none
focus:ring-2 focus:ring(--accent) sm:text-sm"
128         />
129         {prefs.watchTime && (
130             <button
131                 type="button"
132                 onClick={() => onChange({ ...prefs, watchTime: null, reminderMinutes: null })}
133                 className="text-xs text(--ink-2) hover:text(--ink)"
134             >
135                 Clear
136             </button>
137         )}
138     </div>
139 </div>
140 {!prefs.watchTime && (
141     <p className="mt-3 text-xs text(--ink-3)">
142         No time set: calendar events are all-day. Set one for timed events and reminders.
143     </p>
144 )}
145
146 <div className="flex items-center justify-between gap-3">
147     <label htmlFor="reminder" className={`text-sm ${prefs.watchTime ? 'text(--ink-2)' : 'text(--ink-3)'} `}>
148         Reminder
149     </label>
150     <select
151         id="reminder"
152         value={prefs.reminderMinutes ?? ''}
153         disabled={!prefs.watchTime}
154         onChange={(e) => set('reminderMinutes', e.target.value === '' ? null : Number(e.target.value))}
155         className="h-10 bg(--bg) px-3 text-base text(--ink) ring-1 ring(--line) focus:outline-none focus:ring-
2 focus:ring(--accent) disabled:opacity-50 sm:text-sm"
156     >
157         {REMINDER_OPTIONS.map((m) => (
158             <option key={m ?? 'none'} value={m ?? ''}>
159                 {reminderLabel(m)}
160             </option>
161         ))}
162     </select>
163 </div>
164
165 <div className="flex flex-col gap-2">
166     <label htmlFor="timezone" className="text-sm text(--ink-2)">
167         Time zone
168     </label>
169     <select

```

```

170         id="timezone"
171         value={prefs.timezone ?? ''}
172         onChange={(e) => set('timezone', e.target.value || null)}
173         className="h-10 w-full min-w-0 bg-(--bg) px-3 text-base text-(--ink) ring-1 ring-(--line) focus:outline-
none focus:ring-2 focus:ring-(--accent) sm:text-sm"
174     >
175         <option value="">{device ? `This device (${zoneLabel(device)})` : 'This device'}</option>
176         {listTimeZones().map((zone) => (
177             <option key={zone} value={zone}>
178                 {zoneLabel(zone)}
179             </option>
180         ))}
181     </select>
182     <p className="text-xs text-(--ink-3)">
183         Sets which day counts as today and when new episodes air. Calendar exports use your
184         device's time.
185     </p>
186 </div>
187 </div>
188 );
189 }

```

src/components/StreamingIcon.tsx

68 lignes

Les logos des plateformes de streaming (Crunchyroll, Netflix, YouTube...) pour la section « Where to watch ». Une plateforme sans logo reçoit une tuile avec sa première lettre, dans sa couleur.

```

1 // Logos for the "Where to watch" chips. Anilist's `site` is free text, so
2 // this is a best-effort lookup: a known service gets its mark, anything else
3 // gets a monogram tile in the brand color Anilist provides.
4 //
5 // Path data comes from Simple Icons (simpleicons.org, CC0). Hulu, Prime
6 // Video, Disney+, HIDIVE and Funimation aren't there (removed at the brands'
7 // request), so they use the monogram instead of a hand-drawn logo.
8
9 interface Brand {
10     color: string;
11     /** Normalized `site` names (see `normalize`) this mark is used for. */
12     keys: string[];
13     path: string;
14 }
15
16 const BRANDS: Record<string, Brand> = {
17     // Crunchyroll
18     "crunchyroll": { color: "#FF5E00", keys: ["crunchyroll"], path: "M2.909 13.436C2.914 7.61 7.642 2.893 13.468
2.898c5.576.005 10.137 4.339 10.51 9.819q.021-.351.022-.706C24.007 5.385 18.64.006 12.012 0S.007 5.36 0 11.988 5.36
23.994 11.988 24q.412 0 .815-.027c-5.526-.338-9.9-4.928-9.894-10.538Zm16.284.155a4.1 4.1 0 0 1-4.095-4.103 4.1 4.1
0 0 1 2.712-3.855 8.95 8.95 0 0 0-4.187-1.037 9.007 9.007 0 1 0 8.997 9.016q-.001-.847-.15-1.651a4.1 4.1 0 0 1-
3.278 1.632" },
19     // Netflix
20     "netflix": { color: "#E50914", keys: ["netflix"], path: "m5.398 0 8.348 23.602c2.346.059 4.856.398
4.856.398l10.113 0H5.398zm8.489 0v9.172l4.715 13.33V0h-4.715zM5.398 1.5V24c1.873-.225 2.81-.312
4.715-.398V14.83l5.398 1.5z" },
21     // YouTube
22     "youtube": { color: "#FF0000", keys: ["youtube"], path: "M23.498 6.186a3.016 3.016 0 0 0-2.122-2.136C19.505 3.545
12 3.545 12 3.545s-7.505 0-9.377.505A3.017 3.017 0 0 0 .502 6.186C0 8.07 0 12 0 12s0 3.93.502 5.814a3.016 3.016 0 0
0 2.122 2.136c1.871.505 9.376.505 9.376.505s7.505 0 9.377-.505a3.015 3.015 0 0 0 2.122-2.136C24 15.93 24 12 24
12s0-3.93-.502-5.814zM9.545 15.568V8.432l15.818 12l-6.273 3.568z" },
23     // Bilibili
24     "bilibili": { color: "#00A1D6", keys: ["bilibili", "bilibilitv"], path: "M17.813 4.653h.854c1.51.054 2.769.578
3.773 1.574 1.004.995 1.524 2.249 1.56 3.76v7.36c-.036 1.51-.556 2.769-1.56 3.773s-2.262 1.524-3.773 1.56H5.333c-
1.51-.036-2.769-.556-3.773-1.56S.036 18.858 0 17.347v-7.36c.036-1.511.556-2.765 1.56-3.76 1.004-.996 2.262-1.52
3.773-1.574h.774l-1.174-1.12a1.234 1.234 0 0
1-.373-.906c0-.356.124-.658.373-.907l.027-.027c.267-.249.573-.373.92-.373.347 0 .653.124.92.373l9.653
4.44c.071.071.134.142.187.213h4.267a.836.836 0 0 1 .16-.213l2.853-2.747c.267-.249.573-.373.92-.373.347 0
.662.151.929.4.267.249.391.551.391.907 0 .355-.124.657-.373.906zM5.333 7.24c-.746.018-1.373.276-
1.88.773-.506.498-.769 1.13-.786 1.894v7.52c.017.764.28 1.395.786 1.893.507.498 1.134.756 1.88.773h13.334c.746-.017
1.373-.275 1.88-.773.506-.498.769-1.129.786-1.893v-7.52c-.017-.765-.28-1.396-.786-1.894-.507-.497-1.134-.755-
1.88-.773zM8 11.107c.373 0
.684.124.933.373.25.249.383.569.4.96v1.173c-.017.391-.15.711-.4.96-.249.25-.56.374-.933.374s-.684-.125-.933-.374c-.
25-.249-.383-.569-.4-.96V12.44c0-.373.129-.689.386-.947.258-.257.574-.386.947-.386zm8 0c.373 0
.684.124.933.373.25.249.383.569.4.96v1.173c-.017.391-.15.711-.4.96-.249.25-.56.374-.933.374s-.684-.125-.933-.374c-.
25-.249-.383-.569-.4-.96V12.44c.017-.391.15-.711.4-.96.249-.249.56-.373.933-.373z" },
25     // Tubi

```

```

26   "tubi": { color: "#7408FF", keys: ["tubi","tubity"], path: "M16.696 15.272v-.752c.4.548 1.107.917 1.934.917 1.475
0 2.28-.956 2.28-2.865 0-1.714-.893-2.858-2.235-2.858-.851 0-1.55.347-1.979.908v-2.06h-2.674v6.71zm1.57-2.614c0
.827-.337 1.275-.827 1.275-.486 0-.837-.452-.837-1.275s.342-1.28.837-1.28c.495 0 .828.452.828 1.28zM6.94
9.988v3.6c0 1.236.754 1.841 1.955 1.841.959 0 1.625-.396 2.028-1.064v.91h2.597V9.989h-2.675v3.14c0
.493-.346.693-.666.693-.321 0-.568-.192-.568-.655V9.989zM14.39 0H24v5.276h-2.672M6.553 11.136c0 .781-.635 1.415-
1.42 1.415-.783 0-1.419-.634-1.419-1.415 0-.782.636-1.415 1.42-1.415.784 0 1.42.633 1.42 1.415zM3.49
9.702v2.668c.005.653.327.924.976.924.225 0 .526-.053.672-.166v1.931c-.49.243-.869.378-1.535.378 0 0-.069
0-.18-.006l-.003.006c-1.614 0-2.51-1.035-2.482-2.686v-.47H0V9.99h.92V8.563h2.569Z" },
27   // Max (drawn black so the mark shows on the cream theme)
28   "max": { color: "#000000", keys: ["max","hbomax"], path: "M1.769 0A1.77 1.77 0 0 0 1.769V22.23A1.77 1.77 0 0 0
1.769 24H22.23A1.77 1.77 0 0 0 24 22.23V1.77A1.77 1.77 0 0 0 22.23 0zm12.485 3.28a4.301 4.301 0 0 1 4.3 4.302
4.301 4.301 0 0 1-1.993 3.63 6.085 6.085 0 0 1 1.054 3.422 6.085 6.085 0 0 1-6.085 6.085 6.085 6.085 0 0 1-6.085-
6.085 6.085 6.085 0 0 1 4.66-5.916 4.301 4.301 0 0 1-.152-1.136 4.301 4.301 0 0 1 4.301-4.301zm0 1.849a2.453 2.453
0 0 0-2.453 2.453 2.453 2.453 0 0 0 2.453 2.453 2.453 2.453 0 0 0 2.453-2.453 2.453 2.453 0 0 0-2.453-2.453zm-2.724
5.268a4.237 4.237 0 0 0-4.237 4.237 4.237 4.237 0 0 0 4.237 4.237 4.237 4.237 0 0 0 4.237-4.237 4.237 4.237 0 0 0-
4.237-4.237zm.032 2.54a1.781 1.781 0 1 1 0 3.562 1.781 1.781 0 0 1 0-3.562Z" },
29 };
30
31 function normalize(site: string): string {
32   return site.toLowerCase().replace(/([a-z0-9])/g, '');
33 }
34
35 const BY_KEY = new Map(Object.values(BRANDS).flatMap((b) => b.keys.map((k) => [k, b] as const)));
36
37 /**
38  * The service's logo, or a one-letter tile in `color` (AniList's brand
39  * color, already `safeHexColor`-checked) when there's no logo for it.
40  */
41 export function StreamingIcon({
42   site,
43   color,
44   className = 'h-4 w-4',
45 }: {
46   site: string;
47   color: string | null;
48   className?: string;
49 }) {
50   const brand = BY_KEY.get(normalize(site));
51   if (brand) {
52     return (
53       <svg viewBox="0 0 24 24" aria-hidden="true" className={`shrink-0 ${className}`} fill={brand.color}>
54         <path d={brand.path} />
55       </svg>
56     );
57   }
58   const letter = site.trim().charAt(0).toUpperCase() || '?';
59   return (
60     <span
61       aria-hidden="true"
62       className={`grid shrink-0 place-items-center text-[10px] font-bold leading-none text-white ${className}`}
63       style={{ backgroundColor: color ?? 'var(--accent)' }}
64     >
65       {letter}
66     </span>
67   );
68 }

```

src/components/CalendarView.tsx

312 lignes

La vue calendrier : une grille du mois avec, pour chaque jour, les épisodes prévus, ce qui est déjà vu et les dates de diffusion. Cliquer sur un jour affiche ses détails dessous, où on peut le cocher.

```

1  import { useId, useMemo, useRef, useState } from 'react';
2  import { useCollapse, useSlide, useSwap } from '../motion/hooks';
3  import { pop } from '../motion/presets';
4  import type { ScheduleDay } from '../types';
5  import {
6    addDays,
7    formatAiring,
8    formatDate,
9    monthGrid,
10   monthOf,
11   shiftMonth,
12   type Airing,

```

```

13   type Month,
14 } from '../lib/calendar';
15 import { describeEpisodes, formatMinutes } from '../lib/format';
16 import { isDayWatched } from '../lib/progress';
17 import { CheckIcon, ChevronDownIcon, ChevronLeftIcon, ChevronRightIcon } from './icons';
18
19 interface Props {
20   days: ScheduleDay[];
21   /** Day number -> `yyyy-mm-dd`, from `scheduleDates`. */
22   dates: Map<number, string>;
23   watchedEpisodes: Set<number>;
24   airings: Airing[];
25   episodeTitles?: Map<number, string>;
26   onToggleEpisodes: (episodes: number[]) => void;
27   today: string;
28   /** Zone air times are shown in (`null` = device). */
29   timezone: string | null;
30   /** Date shown and selected first (the resume day, usually). */
31   initialDate: string;
32 }
33
34 const MONDAY = '2026-09-28';
35 const WEEKDAY_LABELS = Array.from({ length: 7 }, (_, i) => ({
36   short: formatDate(addDays(MONDAY, i), { weekday: 'short' }),
37   narrow: formatDate(addDays(MONDAY, i), { weekday: 'narrow' }),
38 }));
39
40 /** Days with more titles than this start with the list collapsed. */
41 const OPEN_UP_TO = 3;
42
43 /**
44  * A day's episode titles under a "Episode titles" header that opens and
45  * closes. Only real AniList titles are listed (no invented descriptions), so
46  * this just keeps busy days tidy. Keyed by date and count by the caller, so
47  * each day (and each re-plan) starts in its default state.
48  */
49 function EpisodeTitles({ titles }: { titles: string[] }) {
50   const [open, setOpen] = useState(titles.length <= OPEN_UP_TO);
51   const bodyRef = useRef<HTMLDivElement>(null);
52   const id = useId();
53   useCollapse(bodyRef, open);
54
55   if (titles.length === 1) return <p className="truncate text-sm text(--ink-2)">{titles[0]}</p>;
56
57   return (
58     <div className="flex flex-col">
59       <button
60         type="button"
61         aria-expanded={open}
62         aria-controls={id}
63         onClick={() => setOpen((v) => !v)}
64         className="-mx-1 flex w-fit items-center gap-1.5 px-1 py-0.5 text-sm text(--ink-2) hover:text(--ink)"
65       >
66         Episode titles
67         <span className="tabular-nums text(--ink-3)">{titles.length}</span>
68         <ChevronDownIcon className={`h-4 w-4 transition-transform duration-200 ${open ? 'rotate-180' : ''}`} />
69       </button>
70       <div ref={bodyRef} id={id} aria-hidden={!open}>
71         <ol className="flex flex-col gap-0.5 pt-1.5 text-sm text(--ink-2)">
72           {titles.map((t, i) => (
73             <li key={i} className="truncate">
74               {t}
75             </li>
76           ))}
77         </ol>
78       </div>
79     </div>
80   );
81 }
82
83 /**
84  * Month grid of the plan. Cells show what's planned (a range chip on wide
85  * screens, a dot on phones), progress, today and air dates; tapping a date
86  * opens its details below the grid, where it can be checked off.
87  */
88 export default function CalendarView({
89   days,

```

```

90   dates,
91   watchedEpisodes,
92   airings,
93   episodeTitles,
94   onToggleEpisodes,
95   today,
96   timezone,
97   initialDate,
98   }: Props) {
99   const [month, setMonth] = useState<Month>(() => monthOf(initialDate));
100  const [selected, setSelected] = useState(initialDate);
101  const gridRef = useRef<HTMLDivElement>(null);
102  const checkId = useId();
103  const labelRef = useRef<HTMLHeadingElement>(null);
104  const panelRef = useRef<HTMLElement>(null);
105  useSlide(gridRef, month.year * 12 + month.month);
106  useSlide(labelRef, month.year * 12 + month.month);
107  useSwap(panelRef, selected);
108
109  const toggleSelected = (episodes: number[]) => {
110    onToggleEpisodes(episodes);
111    pop(gridRef.current?.querySelectorAll('[aria-selected="true"] [data-pop]'), 1.2);
112  };
113
114  const dayByDate = useMemo(() => {
115    const map = new Map<string, ScheduleDay>();
116    for (const d of days) {
117      const iso = dates.get(d.day);
118      if (iso) map.set(iso, d);
119    }
120    return map;
121  }, [days, dates]);
122
123  const airingsByDate = useMemo(() => {
124    const map = new Map<string, Airing[]>();
125    for (const a of airings) map.set(a.date, [...(map.get(a.date) ?? []), a]);
126    return map;
127  }, [airings]);
128
129  const weeks = monthGrid(month);
130  const selectedDay = dayByDate.get(selected);
131  const selectedAirings = airingsByDate.get(selected) ?? [];
132  const selectedTitles = selectedDay && episodeTitles
133    ? selectedDay.episodes.map((ep) => episodeTitles.get(ep)).filter((t): t is string => Boolean(t))
134    : [];
135  const monthLabel = formatDate(`${month.year}-${String(month.month + 1).padStart(2, '0')}-01`, {
136    month: 'long',
137    year: 'numeric',
138  });
139
140  const goTo = (iso: string) => {
141    setMonth(monthOf(iso));
142    setSelected(iso);
143  };
144
145  return (
146    <div className="flex flex-col gap-4">
147      <div className="overflow-hidden bg-(--surface) ring-1 ring-(--line)">
148        <div className="flex items-center justify-between gap-3 border-b border-(--line) px-4 py-3 sm:px-6 sm:py-
1494">
150          <h3 ref={labelRef} className="text-base font-semibold text-(--ink) sm:text-lg" aria-live="polite">
151            {monthLabel}
152          </h3>
153          <div className="flex items-center gap-1">
154            <button
155              type="button"
156              onClick={() => goTo(today)}
157              className="mr-1 h-9 px-3 text-sm text-(--ink-2) ring-1 ring-(--line) transition hover:bg-(--surface-
1582) hover:text-(--ink)"
159            >
160              Today
161            </button>
162            <button
163              type="button"
164              onClick={() => setMonth((m) => shiftMonth(m, -1))}
165              aria-label="Previous month"

```

```

164         className="grid h-9 w-9 place-items-center text-(--ink-2) transition hover:bg-(--surface-2)
hover:text-(--ink)"
165     >
166         <ChevronLeftIcon className="h-5 w-5" />
167     </button>
168     <button
169         type="button"
170         onClick={() => setMonth((m) => shiftMonth(m, 1))}
171         aria-label="Next month"
172         className="grid h-9 w-9 place-items-center text-(--ink-2) transition hover:bg-(--surface-2)
hover:text-(--ink)"
173     >
174         <ChevronRightIcon className="h-5 w-5" />
175     </button>
176 </div>
177 </div>
178
179 <div className="grid grid-cols-7 border-b border-(--line) text-center text-xs font-medium text-(--ink-3)">
180     {WEEKDAY_LABELS.map((w) => (
181         <div key={w.short} className="py-2">
182             <span className="sm:hidden">{w.narrow}</span>
183             <span className="hidden sm:inline">{w.short}</span>
184         </div>
185     ))}
186 </div>
187
188 <div ref={gridRef} role="grid" className="grid grid-cols-7">
189     {weeks.flat().map(({ iso, inMonth }) => {
190         const day = dayByDate.get(iso);
191         const done = day ? isDayWatched(day, watchedEpisodes) : false;
192         const airs = airingsByDate.get(iso);
193         const isToday = iso === today;
194         const isSelected = iso === selected;
195         return (
196             <button
197                 key={iso}
198                 type="button"
199                 role="gridcell"
200                 aria-selected={isSelected}
201                 aria-label={[
202                     formatDate(iso, { weekday: 'long', month: 'long', day: 'numeric' }),
203                     day ? `${describeEpisodes(day.episodes)}${done ? ', watched' : ''}` : null,
204                     airs ? airs.map((a) => `episode ${a.episode} airs`).join(', ') : null,
205                 ]
206                     .filter(Boolean)
207                     .join(', ')}
208                 onClick={() => setSelected(iso)}
209                 className={`relative flex h-16 flex-col items-center gap-1 border-b border-r border-(--line) p-1.5
text-left transition sm:h-28 sm:items-stretch sm:p-2 [&nth-child(7n)]:border-r-0 ${
210                     isSelected ? 'bg-(--surface-2)' : 'hover:bg-(--surface-2)/50'
211                 } ${inMonth ? '' : 'opacity-40'}`}
212             >
213                 {isSelected && (
214                     <span aria-hidden="true" className="absolute inset-0 ring-2 ring-(--accent) ring-inset" />
215                 )}
216                 <span
217                     className={`grid h-7 w-7 place-items-center text-sm tabular-nums ${
218                         isToday ? 'bg-(--accent) font-semibold text-(--on-accent)' : 'text-(--ink-2)'
219                     }`}
220                 >
221                     {Number(iso.slice(8))}
222                 </span>
223
224                 {/* Phones: dots. */}
225                 <span data-pop className="flex gap-1 sm:hidden" aria-hidden="true">
226                     {day && (
227                         <span className={`h-1.5 w-1.5 ${done ? 'bg-(--ink-3)' : 'bg-(--accent)'} `} />
228                     )}
229                     {airs && <span className="h-1.5 w-1.5 bg-(--blue)" />}
230                 </span>
231
232                 {/* Wider screens: labels. */}
233                 <span className="hidden min-w-0 flex-col gap-1 sm:flex" aria-hidden="true">
234                     {day && (
235                         <span
236                             data-pop
237                             className={`flex items-center gap-1 truncate px-1.5 py-0.5 text-xs font-medium ${

```

```

238         done
239         ? 'bg(--surface-2) text(--ink-3) line-through'
240         : 'bg(--accent)/15 text(--accent-text)'
241     }` }
242   >
243   {done && <CheckIcon className="h-3 w-3 shrink-0" />}
244   <span className="truncate">{describeEpisodes(day.episodes)}</span>
245 </span>
246 }}
247 {airs?.map((a) => (
248   <span key={a.episode} className="truncate px-1.5 text-xs text(--blue)">
249     Ep {a.episode} airs
250   </span>
251 ))}
252 </span>
253 </button>
254 );
255 }}}
256 </div>
257 </div>
258
259 <div className="flex flex-wrap gap-x-5 gap-y-2 text-xs text(--ink-2)">
260   <span className="flex items-center gap-2">
261     <span className="h-2 w-2 bg(--accent)" /> Planned
262   </span>
263   <span className="flex items-center gap-2">
264     <span className="h-2 w-2 bg(--ink-3)" /> Watched
265   </span>
266   {airings.length > 0 && (
267     <span className="flex items-center gap-2">
268       <span className="h-2 w-2 bg(--blue)" /> New episode airs
269     </span>
270   )}
271 </div>
272
273 <section
274   ref={panelRef}
275   aria-label="Selected day"
276   className="flex flex-col gap-4 bg(--surface) p-5 ring-1 ring(--line) sm:p-6"
277 >
278   <h3 className="text-base font-semibold text(--ink)">
279     {formatDate(selected, { weekday: 'long', month: 'long', day: 'numeric' })}
280     {selected === today && <span className="ml-2 text-sm font-medium text(--accent-text)">Today</span>}
281   </h3>
282
283   {selectedDay ? (
284     <div className="flex items-start gap-4">
285       <input
286         id={`_${checkId}-check`}
287         type="checkbox"
288         checked={isDayWatched(selectedDay, watchedEpisodes)}
289         onChange={() => toggleSelected(selectedDay.episodes)}
290         className="mt-1 h-5 w-5 shrink-0 cursor-pointer"
291       />
292       <div className="flex min-w-0 flex-col gap-1">
293         <label htmlFor={`_${checkId}-check`} className="cursor-pointer text-base text(--ink)">
294           Day {selectedDay.day} · {describeEpisodes(selectedDay.episodes)}
295         <span className="text(--ink-2)"> · {formatMinutes(selectedDay.minutes)}</span>
296       </label>
297       {selectedTitles.length > 0 && <EpisodeTitles key={`_${selected}-${selectedTitles.length}`} titles=
298 {selectedTitles} />}
299     </div>
300   </div>
301   ) : (
302     <p className="text-sm text(--ink-2)">Nothing planned for this day.</p>
303   )}
304
305   {selectedAirings.map((a) => (
306     <p key={a.episode} className="text-sm text(--blue)">
307       Episode {a.episode} airs at {formatAiring(a.airingAt, timezone).time}
308     </p>
309   ))}
310 </section>
311 </div>
312 );
313 }

```

La partie droite de la page d'une série : la date de fin, les chiffres clés, les messages d'avertissement, la progression, le bouton liste/calendrier et la liste jour par jour avec ses cases à cocher.

```

1  import { useEffect, useLayoutEffect, useMemo, useRef, useState, type ReactNode } from 'react';
2  import type { CalendarPrefs, ScheduleError, ScheduleResult, WatchedEpisode } from '../types';
3  import { describeEpisodes, formatMinutes } from '../lib/format';
4  import { getProgress, getResumeDay, isDayWatched, watchedSet } from '../lib/progress';
5  import {
6    formatAiring,
7    formatDate,
8    formatWatchTime,
9    isIsoDate,
10   scheduleDates,
11   todayIso,
12   type Airing,
13 } from '../lib/calendar';
14 import CalendarView from './CalendarView';
15 import { CountUp } from './Motion';
16 import { useEnter, useIndicator, useSwap, useWidth } from '../motion/hooks';
17 import { enter, pop } from '../motion/presets';
18
19 /** Initial (and "show more" increment) count of days rendered for long schedules. */
20 const DAYS_PAGE_SIZE = 30;
21
22 const ERROR_MESSAGES: Record<ScheduleError, string> = {
23   'invalid-budget': 'Enter a daily watch time above 0 minutes.',
24   'unknown-episode-count':
25     "AniList doesn't list an episode count for this show yet, so a schedule can't be built.",
26   'unknown-duration': "AniList doesn't list an episode length for this show.",
27 };
28
29 interface Props {
30   result: ScheduleResult;
31   /** Real episode titles from AniList, keyed by episode number. Optional. */
32   episodeTitles?: Map<number, string>;
33   watched: WatchedEpisode[];
34   onToggleEpisodes: (episodes: number[]) => void;
35   calendar: CalendarPrefs;
36   onCalendarChange: (calendar: CalendarPrefs) => void;
37   /** Upcoming air times for a show still airing (empty otherwise). */
38   airings: Airing[];
39 }
40
41 type View = 'list' | 'calendar';
42
43 function Callout({ children }: { children: ReactNode }) {
44   return (
45     <p className="bg-(--accent)/10 px-5 py-4 text-sm leading-relaxed text-(--ink) ring-1 ring-(--accent)/30">
46       {children}
47     </p>
48   );
49 }
50
51 /** Long date label, adding the year only when it isn't this year. */
52 function longDate(iso: string, today: string): string {
53   return formatDate(iso, {
54     weekday: 'long',
55     month: 'long',
56     day: 'numeric',
57     ...(iso.slice(0, 4) !== today.slice(0, 4) ? { year: 'numeric' } : {}),
58   });
59 }
60
61 function daysBetween(from: string, to: string): number {
62   return Math.round((Date.parse(to) - Date.parse(from)) / 86_400_000);
63 }
64
65 export default function ScheduleView({
66   result,
67   episodeTitles,
68   watched,
69   onToggleEpisodes,
70   calendar,

```

```

71   onCalendarChange,
72   airings,
73 }: Props) {
74   const [shown, setShown] = useState(DAYS_PAGE_SIZE);
75   const [scrollTo, setScrollTo] = useState<number | null>(null);
76   const [view, setView] = useState<View>('list');
77   // Changing this remounts the calendar focused on that date.
78   const [calendarFocus, setCalendarFocus] = useState<string | null>(null);
79   const today = todayIso(undefined, calendar.timezone);
80
81   const finishRef = useRef<HTMLParagraphElement>(null);
82   const barRef = useRef<HTMLDivElement>(null);
83   const toggleRef = useRef<HTMLDivElement>(null);
84   const pillRef = useRef<HTMLSpanElement>(null);
85   const viewRef = useRef<HTMLDivElement>(null);
86   const rowsRef = useRef<HTMLListElement>(null);
87
88   // `result` is a fresh object only when the anime or config changes (see the
89   // `useMemo` in App.tsx), not when a day is checked off. On a new schedule,
90   // render enough rows that the resume day is always visible.
91   useEffect(() => {
92     const resume = result.ok ? getResumeDay(result.schedule.days, watched) : null;
93     setShown(Math.max(DAYS_PAGE_SIZE, (resume ?? 0) + 5));
94     // Deliberately not keyed on `watched`: checking a day off shouldn't
95     // collapse the list back down.
96     // eslint-disable-next-line react-hooks/exhaustive-deps
97   }, [result]);
98
99   useEffect(() => {
100     if (scrollTo == null) return;
101     document.getElementById(`day-${scrollTo}`)?.scrollIntoView({ behavior: 'smooth', block: 'center' });
102     setScrollTo(null);
103   }, [scrollTo, shown]);
104
105   const dates = useMemo(
106     () => scheduleDates(result.ok ? result.schedule.days : [], calendar),
107     [result, calendar],
108   );
109   const airingByEpisode = useMemo(() => new Map(airings.map((a) => [a.episode, a])), [airings]);
110
111   const finishIso = result.ok ? dates.get(result.schedule.days[result.schedule.days.length - 1].day) : undefined;
112   const percent = result.ok ? getProgress(result.schedule.days, watched).percent : 0;
113   useSwap(finishRef, finishIso);
114   useWidth(barRef, percent);
115   // Keyed on result.ok too: the toggle only exists once there's a schedule.
116   useIndicator(toggleRef, pillRef, `${view}-${result.ok}`, '[aria-pressed="true"]');
117   useSwap(viewRef, view, { y: 8 });
118   // Rows re-stagger when the plan changes (budget, show), as feedback.
119   useEnter(rowsRef, [result], { selector: ':scope > li', step: 25, maxSteps: 10, y: 10 });
120
121   // "Show more": only the newly added rows animate.
122   const prevShown = useRef(shown);
123   useEffect(() => {
124     const from = prevShown.current;
125     prevShown.current = shown;
126     if (shown <= from || !rowsRef.current) return;
127     const added = [...rowsRef.current.children].slice(from);
128     const anim = enter(added, { step: 25, maxSteps: 10, y: 10 });
129     return () => {
130       anim?.revert();
131     };
132   }, [shown]);
133
134   if (!result.ok) return <Callout>{ERROR_MESSAGES[result.error]}</Callout>;
135
136   const s = result.schedule;
137   const progress = getProgress(s.days, watched);
138   const resumeDay = getResumeDay(s.days, watched);
139   const watchedEpisodes = watchedSet(watched);
140   const finish = dates.get(s.days[s.days.length - 1].day)!;
141   const firstDate = dates.get(1)!;
142   const resumeDate = resumeDay ? dates.get(resumeDay)! : null;
143   const nextAiring = airings[0];
144
145   const jumpToResume = () => {
146     if (resumeDay == null) return;
147     if (view === 'calendar') {

```

```

148     setCalendarFocus(resumeDate);
149     return;
150 }
151 if (resumeDay > shown) setShow(resumeDay + 5);
152 setScrollTo(resumeDay);
153 };
154
155 /** Episodes planned for `date` that won't have aired by then. */
156 const notOutYet = (episodes: number[], date: string) =>
157     episodes.map((ep) => airingByEpisode.get(ep)).filter((a): a is Airing => !!a && a.date > date);
158
159 const stats: { label: string; value: number; format?: (n: number) => string }[] = [
160     { label: s.days.length === 1 ? 'Watch day' : 'Watch days', value: s.days.length },
161     { label: 'Episodes per day', value: s.episodesPerDay },
162     { label: 'Per episode', value: s.netRuntime, format: (n) => `${n} min` },
163     { label: 'Total time', value: s.totalMinutes, format: formatMinutes },
164 ];
165
166 const span = daysBetween(firstDate, finish) + 1;
167 const when = [
168     calendar.watchDays.length === 7 ? 'every day' : `${calendar.watchDays.length} days a week`,
169     calendar.watchTime ? `at ${formatWatchTime(calendar.watchTime)}` : null,
170 ]
171 .filter(Boolean)
172 .join(' ');
173
174 return (
175     <div className="flex flex-col gap-6">
176         <div className="flex flex-col gap-5 bg(--surface) p-5 ring-1 ring(--line) sm:flex-row sm:items-end
sm:justify-between sm:p-6">
177             <div className="flex flex-col gap-1">
178                 <p className="text-sm text(--ink-2)">You'll finish on</p>
179                 <p ref={finishRef} className="text-2xl font-semibold tracking-tight text(--ink) sm:text-3xl">
180                     {longDate(finish, today)}
181                 </p>
182                 <p className="text-sm text(--ink-2)">
183                     Watching {when} · {span} {span === 1 ? 'day' : 'days'} from start to finish
184                 </p>
185             </div>
186             <label className="flex items-center justify-between gap-3 text-sm text(--ink-2) sm:flex-col sm:items-start
sm:gap-1.5">
187                 Start date
188                 <input
189                     type="date"
190                     value={calendar.startDate}
191                     onChange={(e) => {
192                         if (isIsoDate(e.target.value)) onCalendarChange({ ...calendar, startDate: e.target.value });
193                     }}
194                     className="h-10 bg(--bg) px-3 text-base text(--ink) ring-1 ring(--line) focus:outline-none
focus:ring-2 focus:ring(--accent) sm:text-sm"
195                 />
196             </label>
197         </div>
198
199         <dl className="grid grid-cols-2 gap-3 sm:grid-cols-4 sm:gap-4 lg:grid-cols-2 xl:grid-cols-4">
200             {stats.map((stat) => (
201                 <div
202                     key={stat.label}
203                     className="flex flex-col gap-1 bg(--surface) p-4 ring-1 ring(--line) sm:p-5"
204                 >
205                     <dt className="text-xs text(--ink-2) sm:text-sm">{stat.label}</dt>
206                     <dd className="text-2xl font-semibold tracking-tight tabular-nums text(--ink) sm:text-3xl">
207                         <CountUp value={stat.value} format={stat.format} />
208                     </dd>
209                 </div>
210             ))}
211         </dl>
212         {s.unusedMinutesPerDay > 0 && (
213             <p className="-mt-2 text-sm text(--ink-3)">
214                 {s.unusedMinutesPerDay} min of your daily time goes unused.
215             </p>
216         )}
217
218         {s.overBudget && (
219             <Callout>
220                 One episode ({s.netRuntime} min) is longer than your daily time, so each day holds one
221                 episode and runs over.

```

```

222     </Callout>
223   })
224   {s.isPartial && (
225     <Callout>
226       Still airing with no final episode count on Anilist. You'll catch up on the {s.episodeCount}{' '}
227       episodes out so far by {longDate(finish, today)}.
228       {nextAiring && (
229         <>
230           {' '}
231           Episode {nextAiring.episode} airs {formatAiring(nextAiring.airingAt, calendar.timezone).day}; reopen
the plan
232           later to add new episodes.
233         </>
234       )}
235     </Callout>
236   })
237
238   <div className="flex flex-col gap-4 bg(--surface) p-5 ring-1 ring(--line) sm:p-6">
239     <div className="flex flex-wrap items-baseline justify-between gap-x-4 gap-y-2">
240       <p className="text-sm text(--ink-2)">
241         <CountUp value={progress.percent} format={(n) => `${n}%`} className="text-lg font-semibold tabular-nums
text(--ink)" />{' '}
242         watched · <span className="tabular-nums">{progress.done}</span> of{' '}
243         <span className="tabular-nums">{progress.total}</span> days
244       </p>
245       {resumeDay ? (
246         <button
247           type="button"
248           onClick={jumpToResume}
249           className="bg(--accent) px-4 py-1.5 text-sm font-semibold text(--on-accent) transition
hover:brightness-90"
250         >
251           {resumeDate === today ? "Today's episodes" : `Resume day ${resumeDay}`}
252         </button>
253       ) : (
254         <span className="text-sm font-medium text(--accent-text)">All watched</span>
255       )}
256     </div>
257     <div className="h-2 overflow-hidden bg(--surface-2)">
258       <div ref={barRef} className="h-full bg(--accent)" />
259     </div>
260   </div>
261
262   <div
263     ref={toggleRef}
264     role="group"
265     aria-label="Schedule view"
266     className="relative flex w-fit gap-1 bg(--surface) p-1 ring-1 ring(--line)"
267   >
268     /* One pill that springs to the active button (see useIndicator). */
269     <span ref={pillRef} aria-hidden="true" className="absolute inset-y-1 left-0 bg(--accent)" />
270     {[ 'list', 'calendar' ] as const}.map((v) => (
271       <button
272         key={v}
273         type="button"
274         aria-pressed={view === v}
275         onClick={() => setView(v)}
276         className={`relative h-9 px-5 text-sm font-medium capitalize transition ${
277           view === v ? 'text(--on-accent)' : 'text(--ink-2) hover:text(--ink)'
278         }`}
279       >
280         {v}
281       </button>
282     ))}
283   </div>
284
285   <div ref={viewRef} className="flex flex-col gap-6">
286     {view === 'calendar' ? (
287       <CalendarView
288         key={calendarFocus ?? 'initial'}
289         days={s.days}
290         dates={dates}
291         watchedEpisodes={watchedEpisodes}
292         airings={airings}
293         episodeTitles={episodeTitles}
294         onToggleEpisodes={onToggleEpisodes}
295         today={today}

```

```

296     timezone={calendar.timezone}
297     initialDate={calendarFocus ?? resumeDate ?? today}
298   />
299   ) : (
300     <>
301     <ol ref={rowsRef} className="divide-y divide-(--line) overflow-hidden bg-(--surface) ring-1 ring-(--
line)">
302       {s.days.slice(0, shown).map((day) => {
303         const done = isDayWatched(day, watchedEpisodes);
304         const isResume = day.day === resumeDay;
305         const date = dates.get(day.day)!;
306         const isToday = date === today;
307         const pending = notOutYet(day.episodes, date);
308         const titles = episodeTitles
309           ? day.episodes.map((ep) => episodeTitles.get(ep)).filter(Boolean)
310           : [];
311         return (
312           <li
313             key={day.day}
314             id={`day-${day.day}`}
315             className={`relative transition-colors ${isResume ? 'bg-(--surface-2)' : 'hover:bg-(--surface-
2)/50'}`}
316           >
317             {isResume && (
318               <span aria-hidden="true" className="absolute inset-y-0 left-0 w-1 bg-(--accent)" />
319             )}
320             <label className="flex cursor-pointer items-start gap-4 px-4 py-4 sm:gap-5 sm:px-6 sm:py-5">
321               <input
322                 type="checkbox"
323                 checked={done}
324                 onChange={(e) => {
325                   onToggleEpisodes(day.episodes);
326                   pop(e.currentTarget, 1.25);
327                 }}
328                 aria-label={`Mark day ${day.day}, ${formatDate(date)}, watched`}
329                 className="mt-0.5 h-5 w-5 shrink-0"
330               />
331               <div className="flex w-24 shrink-0 flex-col gap-0.5 sm:w-28">
332                 <span className={`text-sm font-medium ${isToday ? 'text-(--accent-text)' : 'text-(--ink)'}>
333                   {isToday ? 'Today' : formatDate(date)}
334                 </span>
335                 <span className="text-xs tabular-nums text-(--ink-3)">Day {day.day}</span>
336               </div>
337               <div className="flex min-w-0 flex-1 flex-col gap-1">
338                 <div className="flex items-baseline justify-between gap-3">
339                   <span className={`text-base ${done ? 'text-(--ink-3) line-through' : 'text-(--ink)'}>
340                     {describeEpisodes(day.episodes)}
341                   </span>
342                   <span className="shrink-0 text-sm tabular-nums text-(--ink-2)">
343                     {formatMinutes(day.minutes)}
344                   </span>
345                 </div>
346                 {titles.length > 0 && (
347                   <p className="line-clamp-2 text-sm text-(--ink-3) sm:line-clamp-1" title={titles.join('
')}}
348                     {titles.join('
')}
349                   </p>
350                 )}
351                 {pending.length > 0 && (
352                   <p className="text-sm text-(--blue)">
353                     Not out yet:{' '}
354                     {pending.map((a) => `Ep ${a.episode} airs ${formatAiring(a.airingAt,
calendar.timezone).day}`).join(', ')}
355                   </p>
356                 )}
357               </div>
358             </label>
359           </li>
360         );
361       })}
362     </ol>
363
364     {shown < s.days.length && (
365       <div className="flex flex-wrap items-center justify-center gap-3">
366         <button
367           type="button"
368           onClick={() => setShow((n) => n + DAYS_PAGE_SIZE)}

```

```

369         className="h-11 px-6 text-sm font-medium text-(--ink) ring-1 ring-(--line) transition hover:bg-(--
surface)"
370     >
371         Show {Math.min(DAYS_PAGE_SIZE, s.days.length - shown)} more ({s.days.length - shown} left)
372     </button>
373     <button
374         type="button"
375         onClick={() => setShow(s.days.length)}
376         className="h-11 px-5 text-sm text-(--ink-2) transition hover:text-(--ink)"
377     >
378         Show all
379     </button>
380 </div>
381 })
382 </>
383 })
384 </div>
385 </div>
386 );
387 }
388

```

src/components/ConfigureStep.tsx

428 lignes

La page complète d'une série : l'en-tête (affiche, titre, genres, synopsis, boutons Save et bande-annonce), la colonne de réglages (plan, jours, où regarder, ordre de visionnage, partage et export) et le planning.

```

1  import { useEffect, useLayoutEffect, useMemo, useRef, useState, type ReactNode } from 'react';
2  import { animate, svg } from 'animejs';
3  import type { Anime, CalendarPrefs, SaveStatus, ScheduleConfig, ScheduleResult, WatchedEpisode } from '../types';
4  import {
5      displayTitle,
6      episodeTitleMap,
7      formatLabel,
8      synopsisText,
9      youtubeEmbedUrl,
10     slugify,
11 } from '../lib/format';
12 import { downloadTextFile, openTextFile } from '../lib/download';
13 import { isIOS } from '../lib/platform';
14 import { formatAiring, todayIso, upcomingAirings } from '../lib/calendar';
15 import { buildIcs } from '../lib/ics';
16 import { getNetEpisodeRuntime } from '../lib/schedule';
17 import { animePath } from '../lib/route';
18 import ConfigForm from './ConfigForm';
19 import ScheduleView from './ScheduleView';
20 import { StreamingIcon } from './StreamingIcon';
21 import WatchPrefsForm from './WatchPrefsForm';
22 import SectionHeading from './SectionHeading';
23 import { useEnter } from '../motion/hooks';
24 import { pop, reduceMotion } from '../motion/presets';
25 import { CheckIcon, ChevronLeftIcon, ExternalIcon, PlayIcon } from './icons';
26
27 /** "today", "tomorrow" or "in N days" for a `yyyy-mm-dd` date in `timezone`. */
28 function relativeDay(iso: string, timezone: string | null): string {
29     const n = Math.round((Date.parse(iso) - Date.parse(todayIso(undefined, timezone))) / 86_400_000);
30     if (n <= 0) return 'today';
31     return n === 1 ? 'tomorrow' : `in ${n} days`;
32 }
33
34 interface Props {
35     anime: Anime;
36     config: ScheduleConfig;
37     onConfigChange: (config: ScheduleConfig) => void;
38     calendar: CalendarPrefs;
39     onCalendarChange: (calendar: CalendarPrefs) => void;
40     result: ScheduleResult;
41     onBack: () => void;
42     saveStatus: SaveStatus;
43     onSave: () => void;
44     onOpenRelated: (id: number) => void;
45     watched: WatchedEpisode[];
46     onToggleEpisodes: (episodes: number[]) => void;
47 }

```

```

48
49 const DURATION_POSTER = 520;
50
51 const SAVE_LABELS: Record<SaveStatus, string> = {
52   unsaved: 'Save',
53   saved: 'Saved',
54   changed: 'Update',
55   failed: 'Retry save',
56 };
57
58 function SidebarSection({ title, children }: { title: string; children: ReactNode }) {
59   return (
60     <div className="flex flex-col gap-4 p-5 sm:p-6">
61       <SectionHeading size="sm">{title}</SectionHeading>
62       {children}
63     </div>
64   );
65 }
66
67 export default function ConfigureStep({
68   anime,
69   config,
70   onConfigChange,
71   calendar,
72   onCalendarChange,
73   result,
74   onBack,
75   saveStatus,
76   onSave,
77   onOpenRelated,
78   watched,
79   onToggleEpisodes,
80 }: Props) {
81   const [synopsisExpanded, setSynopsisExpanded] = useState(false);
82   const [shareLink, setShareLink] = useState<string | null>(null);
83   const [shareCopied, setShareCopied] = useState(false);
84   const [showTrailer, setShowTrailer] = useState(false);
85
86   const headerRef = useRef<HTMLInputElement>(null);
87   const posterRef = useRef<HTMLImageElement>(null);
88   const asideRef = useRef<HTMLInputElement>(null);
89   const saveRef = useRef<HTMLButtonElement>(null);
90   const shareRef = useRef<HTMLButtonElement>(null);
91
92   // Once per show: poster settles in, then the title block and details, then the sidebar.
93   useEnter(posterRef, [anime.id], { scale: 0.94, y: 0, duration: DURATION_POSTER });
94   useEnter(headerRef, [anime.id], { selector: '[data-enter]', step: 70, delay: 80 });
95   useEnter(asideRef, [anime.id], { selector: ':scope > div', step: 50, delay: 120, y: 12 });
96
97   // Pop the save button and draw its check when a save lands (not when
98   // opening an already-saved show).
99   const lastSave = useRef({ id: anime.id, status: saveStatus });
100   useEffect(() => {
101     const prev = lastSave.current;
102     lastSave.current = { id: anime.id, status: saveStatus };
103     if (prev.id !== anime.id || prev.status === 'saved' || saveStatus !== 'saved') return;
104     pop(saveRef.current, 1.08);
105     const path = saveRef.current?.querySelector('path');
106     if (!path || reduceMotion()) return;
107     const anim = animate(svg.createDrawable(path), { draw: ['0 0', '0 1'], duration: 360, delay: 60, ease:
'outQuad' });
108     return () => {
109       anim.revert();
110     };
111   }, [anime.id, saveStatus]);
112
113   // Collapse the trailer when switching shows (e.g. via Watch order).
114   useEffect(() => setShowTrailer(false), [anime.id]);
115   const trailerUrl = youtubeEmbedUrl(anime.trailer);
116
117   const title = displayTitle(anime);
118   const isMovie = anime.format === 'MOVIE';
119   const meta = [
120     formatLabel(anime.format),
121     anime.seasonYear,
122     anime.studios[0]?.name,
123     anime.episodes !== null && !isMovie ? `${anime.episodes} ep` : null,

```

```

124     anime.duration !== null ? `${anime.duration} min${isMovie ? '' : ' each'}` : null,
125   ].filter(Boolean);
126   const synopsis = anime.description ? synopsisText(anime.description) : null;
127   const episodeTitles = useMemo(
128     () => episodeTitleMap(anime.streamingEpisodes),
129     [anime.streamingEpisodes],
130   );
131
132   const runtimes = useMemo(
133     () => ({
134       trimmed: getNetEpisodeRuntime(anime, { ...config, skipOpEd: true }),
135       full: getNetEpisodeRuntime(anime, { ...config, skipOpEd: false }),
136     }),
137     // Only the runtime inputs matter, not the budget.
138     // eslint-disable-next-line react-hooks/exhaustive-deps
139     [anime.duration, anime.format],
140   );
141
142   const airings = useMemo(
143     () => upcomingAirings(anime, undefined, calendar.timezone),
144     [anime, calendar.timezone],
145   );
146   const nextAiring = airings[0];
147
148   // iOS Safari can show its own "Add to Calendar" sheet for a calendar file
149   // opened in the tab; everywhere else a normal download is what the OS
150   // expects (tapping the file offers to import it).
151   const handleAddToCalendar = () => {
152     if (!result.ok) return;
153     const ics = buildIcs(anime, result.schedule, calendar, episodeTitles);
154     if (isIOS() && openTextFile(ics, 'text/calendar')) return;
155     downloadTextFile(`${slugify(title)} || 'schedule'.ics`, ics, 'text/calendar');
156   };
157
158   const handleShare = async () => {
159     const link = `${location.origin}${animePath(anime.id, title)}`;
160     try {
161       await navigator.clipboard.writeText(link);
162       setShareLink(null);
163       setShareCopied(true);
164       pop(shareRef.current, 1.06);
165       setTimeout(() => setShareCopied(false), 2000);
166     } catch {
167       // Clipboard API unavailable/denied: show it so the user can copy manually.
168       setShareLink(link);
169     }
170   };
171
172   const saveClass =
173     saveStatus === 'saved'
174       ? 'text(--ink) ring(--line) bg(--surface-2)/60'
175       : saveStatus === 'failed'
176       ? 'bg-red-600/10 text-red-800 ring-red-600/30 hover:bg-red-600/15'
177       : 'bg(--accent) text(--on-accent) ring(--accent) shadow-lg shadow(--accent)/25 hover:brightness-90
disabled:opacity-40';
178
179   const secondaryButton =
180     'flex h-11 items-center justify-center gap-2 px-5 text-sm font-medium text(--ink) ring-1 ring(--line)
transition hover:bg(--surface-2)';
181
182   return (
183     <div className="flex flex-col gap-8 sm:gap-10">
184       <button
185         type="button"
186         onClick={onBack}
187         className="-ml-2 flex w-fit items-center gap-1.5 py-1.5 pl-2 pr-4 text-sm text(--ink-2) transition
hover:bg(--surface) hover:text(--ink)"
188       >
189         <ChevronLeftIcon className="h-4 w-4" />
190         Back
191       </button>
192
193       <header ref={headerRef} className="relative isolate overflow-hidden bg(--surface) ring-1 ring(--line)">
194         {anime.coverImage.large && (
195           <img
196             src={anime.coverImage.large}
197             alt=""

```

```

198     aria-hidden="true"
199     className="absolute inset-0 -z-10 h-full w-full scale-125 object-cover opacity-30 blur-3xl"
200   />
201   })
202   <div
203     aria-hidden="true"
204     className="absolute inset-0 -z-10 bg-linear-to-t from(--surface) via(--surface)/85 to(--surface)/40"
205   />
206
207   {/* Phones: poster beside the title, details full width below.
208      sm and up: poster on the left spanning both rows. */}
209   <div className="grid grid-cols-[6.5rem_minmax(0,1fr)] gap-x-5 gap-y-6 p-5 sm:grid-cols-
[10rem_minmax(0,1fr)] sm:gap-x-8 sm:p-8 lg:grid-cols-[13rem_minmax(0,1fr)] lg:gap-x-10 lg:p-10">
210     {anime.coverImage.large && (
211       <img
212         ref={posterRef}
213         src={anime.coverImage.large}
214         alt=""
215         className="aspect-[2/3] w-full object-cover shadow-2xl shadow(--navy)/25 ring-1 ring(--line)
sm:row-span-2"
216       />
217     )}
218
219     <div data-enter className="flex min-w-0 flex-col justify-end gap-2 sm:justify-start">
220       <h1 className="text-2xl font-semibold leading-tight tracking-tight text-balance text(--ink) sm:text-
4xl lg:text-5xl">
221         {title}
222       </h1>
223       {anime.title.english && anime.title.english !== anime.title.romaji && (
224         <p className="text-sm text(--ink-2) sm:text-base">{anime.title.romaji}</p>
225       )}
226       <p className="text-sm text(--ink-2) sm:text-base">{meta.join(' · ')}</p>
227     </div>
228
229     <div data-enter className="col-span-2 flex min-w-0 flex-col gap-5 sm:col-span-1 sm:col-start-2">
230       {anime.genres.length > 0 && (
231         <ul className="flex flex-wrap gap-2">
232           {anime.genres.map((genre) => (
233             <li
234               key={genre}
235               className="bg(--surface-2)/70 px-3 py-1 text-xs text(--ink-2) ring-1 ring(--line) sm:text-
sm"
236             >
237               {genre}
238             </li>
239           )}
240         </ul>
241       )}
242       {nextAiring && (
243         <p className="text-sm font-medium text(--accent-text)">
244           Episode {nextAiring.episode} airs {formatAiring(nextAiring.airingAt, calendar.timezone).day} at{'
' }
245           {formatAiring(nextAiring.airingAt, calendar.timezone).time} (
246             {relativeDay(nextAiring.date, calendar.timezone)})
247         </p>
248       )}
249
250       {synopsis && (
251         <div className="flex max-w-3xl flex-col items-start gap-2 text-sm leading-relaxed text(--ink-2)
sm:text-base">
252           <p className={`whitespace-pre-line ${synopsisExpanded ? '' : 'line-clamp-3'}`}>
253             {synopsis}
254           </p>
255           <button
256             type="button"
257             onClick={() => setSynopsisExpanded((v) => !v)}
258             className="text-sm font-medium text(--accent-text) hover:underline"
259           >
260             {synopsisExpanded ? 'Show less' : 'Read more'}
261           </button>
262         </div>
263       )}
264
265       <div className="flex flex-wrap gap-3">
266         <button
267           ref={saveRef}
268           type="button"

```

```

269         onClick={onSave}
270         disabled={saveStatus === 'saved' || !result.ok}
271         title={
272             saveStatus === 'failed' ? 'Browser storage is unavailable or full.' : undefined
273         }
274         className={`flex h-11 items-center justify-center gap-2 px-6 text-sm font-semibold ring-1
transition disabled:cursor-default ${saveClass}`}
275     >
276         {saveStatus === 'saved' && <CheckIcon className="h-4 w-4" />}
277         {SAVE_LABELS[saveStatus]}
278     </button>
279     {trailerUrl && (
280         <button
281             type="button"
282             onClick={() => setShowTrailer((v) => !v)}
283             aria-expanded={showTrailer}
284             aria-controls="trailer"
285             className={secondaryButton}
286         >
287             <PlayIcon className="h-4 w-4" />
288             {showTrailer ? 'Hide trailer' : 'Watch trailer'}
289         </button>
290     )}
291 </div>
292 </div>
293 </div>
294 </header>
295
296 {trailerUrl && showTrailer && (
297     <div
298         id="trailer"
299         className="aspect-video w-full max-w-5xl overflow-hidden bg-black ring-1 ring-(--line)"
300     >
301         {/* Only mounted after a click, so nothing loads from YouTube until asked. */}
302         <iframe
303             src={trailerUrl}
304             title={` ${title} trailer `}
305             loading="lazy"
306             sandbox="allow-scripts allow-same-origin allow-presentation allow-popups"
307             referrerPolicy="strict-origin-when-cross-origin"
308             allow="encrypted-media; picture-in-picture; fullscreen"
309             allowFullScreen
310             className="h-full w-full"
311         />
312     </div>
313 )}
314
315 <div className="grid grid-cols-[minmax(0,1fr)] gap-8 lg:grid-cols-[20rem_minmax(0,1fr)] lg:gap-10 xl:grid-
cols-[22rem_minmax(0,1fr)] xl:gap-12">
316     {/* Header is h-20 on desktop; top-28 leaves a gap below it. */}
317     <aside ref={asideRef} className="flex flex-col divide-y divide-(--line) self-start bg-(--surface) ring-1
ring-(--line) lg:sticky lg:top-28 lg:max-h-[calc(100vh-8.5rem)] lg:overflow-y-auto lg:[scrollbar-width:thin]">
318         <SidebarSection title="Your plan">
319             <ConfigForm config={config} onChange={onConfigChange} isMovie={isMovie} runtimes={runtimes} />
320         </SidebarSection>
321
322         {result.ok && (
323             <SidebarSection title="When you watch">
324                 <WatchPrefsForm prefs={calendar} onChange={onCalendarChange} />
325             </SidebarSection>
326         )}
327
328         {anime.watchLinks && anime.watchLinks.length > 0 && (
329             <SidebarSection title="Where to watch">
330                 <div className="flex flex-wrap gap-2">
331                     {anime.watchLinks.map((link) => (
332                         <a
333                             key={link.site}
334                             href={link.url}
335                             target="_blank"
336                             rel="noreferrer"
337                             className="flex items-center gap-2 border-l-[3px] bg-(--surface-2) py-2 pl-3 pr-3 text-sm text-
(--ink) transition hover:text-(--accent-text)"
338                             style={{ borderLeftColor: link.color ?? 'var(--accent)' }}
339                         >
340                             <StreamingIcon site={link.site} color={link.color} />
341                             {link.site}

```

```

342         <ExternalIcon className="h-3.5 w-3.5 text(--ink-3)" />
343     </a>
344     )}}
345 </div>
346 <p className="text-xs text(--ink-3)">Availability varies by region.</p>
347 </SidebarSection>
348 }}
349
350 {anime.relations && anime.relations.length > 1 && (
351   <SidebarSection title="Watch order">
352     <ol className="-mx-2 flex flex-col gap-1">
353       {anime.relations.map((r, i) => (
354         <li key={r.id}>
355           {r.isCurrent ? (
356             <div className="flex items-center gap-3 bg(--surface-2) px-3 py-2.5 text-sm text(--ink)">
357               <span className="w-5 text-xs tabular-nums text(--ink-3)">{i + 1}</span>
358               <span className="min-w-0 flex-1 truncate">{r.title}</span>
359               <span className="text-[11px] font-medium uppercase tracking-wider text(--accent-text)">
360                 Current
361               </span>
362             </div>
363           ) : (
364             <button
365               type="button"
366               onClick={() => onOpenRelated(r.id)}
367               className="flex w-full items-center gap-3 px-3 py-2.5 text-left text-sm text(--ink-2)
368               transition hover:bg(--surface-2) hover:text(--ink)"
369             >
370               <span className="w-5 text-xs tabular-nums text(--ink-3)">{i + 1}</span>
371               <span className="min-w-0 flex-1 truncate">{r.title}</span>
372               {r.seasonYear && (
373                 <span className="text-xs text(--ink-3)">{r.seasonYear}</span>
374               )}
375             </button>
376           )}
377         </li>
378       )}}
379     </ol>
380   </SidebarSection>
381   )}
382
383 {result.ok && (
384   <SidebarSection title="Share & export">
385     <div className="grid grid-cols-2 gap-3">
386       <button
387         type="button"
388         onClick={handleAddToCalendar}
389         className="h-10 px-3 text-sm text(--ink) ring-1 ring(--line) transition hover:bg(--surface-2)"
390       >
391         Add to calendar
392       </button>
393       <button
394         ref={shareRef}
395         type="button"
396         onClick={handleShare}
397         className="h-10 px-3 text-sm text(--ink) ring-1 ring(--line) transition hover:bg(--surface-2)"
398       >
399         {shareCopied ? 'Copied' : 'Copy link'}
400       </button>
401     </div>
402     {shareLink && (
403       <input
404         readOnly
405         value={shareLink}
406         onFocus={(e) => e.currentTarget.select()}
407         aria-label="Share link"
408         className="h-10 w-full bg(--bg) px-3 text-sm text(--ink-2) ring-1 ring(--line)"
409       />
410     )}
411   </SidebarSection>
412 )}
413 </aside>
414
415 <section className="min-w-0">
416   <ScheduleView
417     result={result}
418     episodeTitles={episodeTitles}

```

```

418         watched={watched}
419         onToggleEpisodes={onToggleEpisodes}
420         calendar={calendar}
421         onCalendarChange={onCalendarChange}
422         airings={airings}
423     />
424 </section>
425 </div>
426 </div>
427 );
428 }

```

src/components/ErrorBoundary.tsx

52 lignes

Le filet de sécurité : si une erreur imprévue survient pendant l'affichage, il montre une page « Something went wrong » avec un bouton pour recharger, au lieu d'une page blanche.

```

1  import { Component, type ErrorInfo, type ReactNode } from 'react';
2
3  interface Props {
4      children: ReactNode;
5  }
6
7  interface State {
8      failed: boolean;
9  }
10
11  /**
12   * Last line of defense around the whole app: a render error shows this
13   * fallback instead of a blank page. It sits outside `.theme-root`, so it
14   * uses the :root brand colors (index.css) rather than the theme tokens.
15   * Saved schedules live in localStorage and survive the reload.
16   */
17  export default class ErrorBoundary extends Component<Props, State> {
18      state: State = { failed: false };
19
20      static getDerivedStateFromError(): State {
21          return { failed: true };
22      }
23
24      componentDidCatch(error: Error, info: ErrorInfo) {
25          console.error('Anime Watch Planner crashed:', error, info.componentStack);
26      }
27
28      render() {
29          if (!this.state.failed) return this.props.children;
30          return (
31              <main role="alert" className="mx-auto flex min-h-screen max-w-md flex-col justify-center gap-4 px-6 text(--navy)/80">
32                  <h1 className="text-2xl font-semibold tracking-tight text(--navy)">Something went wrong</h1>
33                  <p>The page hit an unexpected error. Your saved schedules are still stored in this browser.</p>
34                  <div className="flex flex-wrap gap-3 pt-2">
35                      <button
36                          type="button"
37                          onClick={() => location.reload()}
38                          className="bg(--orange) px-5 py-2.5 text-sm font-semibold text-white hover:brightness-90"
39                      >
40                          Reload
41                      </button>
42                      <a
43                          href="/"
44                          className="px-5 py-2.5 text-sm font-semibold text(--navy) ring-1 ring(--navy)/25 hover:bg(--navy)/5"
45                      >
46                          Back to home
47                      </a>
48                  </div>
49              </main>
50          );
51      }
52  }

```

A.8 L'application

src/index.css

97 lignes

La feuille de style globale : elle charge Tailwind, définit les couleurs du site (bleu marine, orange, crème) et les jetons du thème calculés à partir des couleurs de la série (`--poster` , `--poster-accent`), redessine les cases à cocher avec des coins carrés, coupe les animations CSS si l'utilisateur le demande et règle les transitions des boutons.

```
1  @import "tailwindcss";
2
3  /*
4   * Brand palette: Pantone 19-4052 Classic Blue and 17-1340 Adobe, sampled
5   * from the swatches on the project's mood board, on a warm cream. `--blue`
6   * is Pantone's own sRGB value for Classic Blue, brighter than the sampled
7   * navy; it marks air dates (dots, "Ep 5 airs") so they stand apart from
8   * text. Defined on :root so the page body and ErrorBoundary (outside
9   * `.theme-root`) can use them too.
10  */
11  :root {
12    --navy: #133055;
13    --orange: #bf4a1b;
14    --cream: #f3ede4;
15    --blue: #0f4c81;
16  }
17
18  /*
19   * Whole-page light theme. A show page takes on its poster's colors through
20   * two inputs App.tsx sets from `posterTheme` (lib/theme.ts): `--poster`
21   * tints the background and surfaces, and `--poster-accent` replaces the
22   * orange accent (buttons, toggles, progress, highlights). Both are unset on
23   * the homepage and for gray covers, which keep the brand navy and orange.
24   * `posterTheme` already darkened the accent so white text on it
25   * (`--on-accent`) holds 4.5:1, and lightened the tint so a dark cover
26   * doesn't gray the page. Text uses the `--ink*` tokens, mixed from navy
27   * against the fixed cream: `--ink` for headings and body, `--ink-2` for
28   * secondary text, `--ink-3` for hints. lib/theme.test.ts mirrors these
29   * percentages and checks text contrast over 4096 cover colors: change both
30   * together.
31  */
32  .theme-root {
33    --tint: var(--poster, var(--cream));
34    --accent: var(--poster-accent, var(--orange));
35    --accent-text: color-mix(in srgb, var(--accent) 75%, black);
36    --on-accent: #fff;
37    --bg: color-mix(in srgb, var(--tint) 16%, var(--cream));
38    --surface: color-mix(in srgb, var(--tint) 8%, #fbf8f3);
39    --surface-2: color-mix(in srgb, var(--tint) 22%, #eae2d6);
40    --line: color-mix(in srgb, var(--navy) 15%, transparent);
41    --ink: var(--navy);
42    --ink-2: color-mix(in srgb, var(--navy) 82%, var(--cream));
43    --ink-3: color-mix(in srgb, var(--navy) 74%, var(--cream));
44    color-scheme: light;
45    background-color: var(--bg);
46    transition: background-color 500ms ease;
47  }
48
49  /*
50   * Square checkboxes: native ones are drawn with rounded corners whatever
51   * `border-radius` says, so they're redrawn here. The tick is a clip-path,
52   * not an image (the CSP has no data: images). No transition: `pop()` in
53   * anime.js gives the press feedback.
54  */
55  .theme-root input[type='checkbox'] {
56    appearance: none;
57    position: relative;
58    border: 2px solid var(--ink-3);
59    background-color: var(--surface);
60  }
61  .theme-root input[type='checkbox']:checked {
62    border-color: var(--accent);
63    background-color: var(--accent);
64  }
65  .theme-root input[type='checkbox']:checked::before {
```

```

66   content: '';
67   position: absolute;
68   inset: 12%;
69   background-color: var(--on-accent);
70   clip-path: polygon(14% 44%, 0 65%, 50% 100%, 100% 16%, 80% 0%, 43% 62%);
71 }
72 .theme-root input[type='checkbox']:focus-visible {
73   outline: 2px solid var(--accent);
74   outline-offset: 2px;
75 }
76
77 @media (prefers-reduced-motion: reduce) {
78   *,
79   *::before,
80   *::after {
81     animation: none !important;
82     transition: none !important;
83   }
84 }
85
86 /*
87  * Motion lives in anime.js (src/motion/). Buttons and links get a spring
88  * squash on press (installPressFeedback), so CSS must not also transition
89  * their transform, or the two would fight. Unlayered, so it beats Tailwind's
90  * `transition` utility; colors, opacity and shadows still fade in CSS.
91  */
92 button,
93 a[href],
94 [data-press] {
95   transition-property: color, background-color, border-color, outline-color, text-decoration-color, fill,
96   stroke, opacity, box-shadow, filter, backdrop-filter;
97 }

```

src/App.tsx

425 lignes

Le chef d'orchestre. Il garde tout l'état de l'application (la recherche, la série ouverte, les réglages, les plannings sauvegardés), appelle AniList, gère l'adresse dans la barre du navigateur, met à jour le SEO, donne à la page les couleurs de la série ouverte et choisit quelle page afficher.

```

1  // Single source of truth for app state. Components below are presentational;
2  // fetching and state live here.
3
4  import { useEffect, useMemo, useRef, useState } from 'react';
5  import type { Anime, CalendarPrefs, SavedSchedule, SaveStatus, ScheduleConfig, Step } from './types';
6  import { getAnimeById, getPopularAiring, searchAnime } from './api/anilist';
7  import { generateSchedule } from './lib/schedule';
8  import { deleteSavedSchedule, getSavedSchedules, saveSchedule, toggleEpisodesWatched } from './lib/storage';
9  import { decodeShareParam } from './lib/share';
10 import { defaultCalendarPrefs, sameCalendarPrefs } from './lib/calendar';
11 import { animePath, parseAnimePath } from './lib/route';
12 import { displayTitle } from './lib/format';
13 import { posterTheme } from './lib/theme';
14 import { HOME_SEO, IMAGE_META_KEYS, animeJsonLd, animeSeo, metaTags, type MetaTag } from './lib/seo';
15 import SearchStep from './components/SearchStep';
16 import ConfigureStep from './components/ConfigureStep';
17 import SearchBar from './components/SearchBar';
18 import { GlitchText, Spinner } from './components/Motion';
19 import { useEnter } from './motion/hooks';
20 import { installPressFeedback } from './motion/presets';
21
22 /** Shared page width + gutters, so header, main and footer line up at every size. */
23 const CONTAINER = 'mx-auto w-full max-w-7xl px-4 sm:px-8 lg:px-12 2xl:max-w-[96rem]';
24
25 const DEFAULT_CONFIG: ScheduleConfig = {
26   dailyBudget: 90,
27   skipOpEd: false,
28 };
29
30 /** Whether the engine can build a schedule for this show at all. */
31 function isPlannable(anime: Anime): boolean {
32   return generateSchedule(anime, DEFAULT_CONFIG).ok;
33 }
34
35 type UrlUpdate = 'push' | 'replace' | 'none';

```

```

36
37 /** Moves the address bar to `path` (a no-op when it's already there). */
38 function setUrl(path: string, mode: UrlUpdate) {
39   if (mode === 'none' || location.pathname + location.search === path) return;
40   if (mode === 'push') history.pushState(null, '', path);
41   else history.replaceState(null, '', path);
42 }
43
44 function upsertMeta({ attr, key, content }: MetaTag) {
45   let el = document.head.querySelector<HTMLMetaElement>(`meta[${attr}="${key}"]`);
46   if (!el) {
47     el = document.createElement('meta');
48     el.setAttribute(attr, key);
49     document.head.appendChild(el);
50   }
51   el.content = content;
52 }
53
54 /** True when the page was loaded on a show's URL (or an old share link). */
55 function startsOnShow(): boolean {
56   return parseAnimePath(location.pathname) != null || new URLSearchParams(location.search).has('share');
57 }
58
59 function errorMessage(err: unknown): string {
60   return err instanceof Error ? err.message : 'Something went wrong talking to AniList.';
61 }
62
63 export default function App() {
64   const [step, setStep] = useState<Step>('search');
65   const [query, setQuery] = useState('');
66   const [searchedFor, setSearchFor] = useState<string | null>(null);
67   const [results, setResults] = useState<Anime[] | null>(null);
68   const [loading, setLoading] = useState(false);
69   const [error, setError] = useState<string | null>(null);
70   const [selected, setSelected] = useState<Anime | null>(null);
71   const [config, setConfig] = useState<ScheduleConfig>(DEFAULT_CONFIG);
72   const [calendar, setCalendar] = useState<CalendarPrefs>(() => defaultCalendarPrefs());
73   const [popular, setPopular] = useState<Anime[] | null>(null);
74   const [savedSchedules, setSavedSchedules] = useState<SavedSchedule[]>(getSavedSchedules);
75   const [saveFailed, setSaveFailed] = useState(false);
76   // A direct link to a show is loading: show a placeholder, not the homepage.
77   const [opening, setOpening] = useState(startsOnShow);
78   const viewRef = useRef<HTMLDivElement>(null);
79
80   useEffect(() => installPressFeedback(), []);
81
82   // The homepage list is decorative: on failure (e.g. rate limit) the
83   // section just doesn't render. Shows the engine can't plan (unaired, no
84   // length) are left out; ongoing shows stay in, via the aired-so-far fallback.
85   useEffect(() => {
86     const controller = new AbortController();
87     getPopularAiring(controller.signal).then(
88       // 18 fills whole rows at every grid width (2, 3 or 6 columns).
89       (list) => setPopular(list.filter(isPlannable).slice(0, 18)),
90       // [] (not null) so the homepage hides the section instead of showing
91       // a loading skeleton forever.
92       () => setPopular([]),
93     );
94     return () => controller.abort();
95   }, []);
96
97   // Cancels whichever request is in flight when a newer one starts.
98   const inFlight = useRef<AbortController | null>(null);
99   const startRequest = () => {
100     inFlight.current?.abort();
101     const controller = new AbortController();
102     inFlight.current = controller;
103     return controller;
104   };
105
106   const handleSearch = async (term: string) => {
107     const search = term.trim();
108     if (!search) return;
109     setQuery(term);
110     setSearchFor(search);
111     setOpening(false);
112     setUrl('/', 'push');

```

```

113 // The search bar lives in the header, so this can fire from the
114 // configure page too: always land on the results view.
115 setStep('search');
116 const controller = startRequest();
117 setLoading(true);
118 setError(null);
119 try {
120   setResults(await searchAnime(search, controller.signal));
121 } catch (err) {
122   if (controller.signal.aborted) return;
123   setError(errorMessage(err));
124 } finally {
125   if (inFlight.current === controller) setLoading(false);
126 }
127 };
128
129 // Shows the anime immediately and moves the URL to its page. Unless it's
130 // already `fresh` from getAnimeById, it's refetched by id in the
131 // background so saved snapshots and list results pick up detail fields
132 // and changes (e.g. an airing show's episode count). A show without saved
133 // calendar prefs starts today, every day, any time.
134 const openAnime = async (
135   anime: Anime,
136   nextConfig: ScheduleConfig,
137   { calendar: nextCalendar, url = 'push', fresh = false }: { calendar?: CalendarPrefs; url?: UrlUpdate; fresh?:
boolean } = {},
138 ) => {
139   setOpening(false);
140   setSelected(anime);
141   setConfig(nextConfig);
142   setCalendar(nextCalendar ?? defaultCalendarPrefs());
143   setSaveFailed(false);
144   setStep('configure');
145   setUrl(animePath(anime.id, displayTitle(anime)), url);
146   if (fresh) return;
147   const controller = startRequest();
148   try {
149     const latest = await getAnimeById(anime.id, controller.signal);
150     setSelected(latest);
151     // Keep the slug canonical if the title changed since the snapshot.
152     if (parseAnimePath(location.pathname) === latest.id) {
153       setUrl(animePath(latest.id, displayTitle(latest)), 'replace');
154     }
155   } catch {
156     // What we already have is a complete MediaFields shape; keep it.
157   }
158 };
159
160 // Syncs the view to the address bar: on load and on back/forward. A show
161 // URL opens that show (instantly from a saved snapshot when there is one);
162 // an old `?share=` link opens with its config and is rewritten to the
163 // show's clean URL; anything else is the homepage.
164 const resolveLocation = () => {
165   const id = parseAnimePath(location.pathname);
166   const shared = id == null ? decodeShareParam(new URLSearchParams(location.search).get('share') ?? '') : null;
167   const targetId = id ?? shared?.id ?? null;
168
169   if (targetId == null) {
170     inFlight.current?.abort();
171     setOpening(false);
172     setStep('search');
173     return;
174   }
175   if (selected?.id === targetId && !shared) {
176     setStep('configure');
177     return;
178   }
179   const saved = savedSchedules.find((s) => s.anime.id === targetId);
180   if (saved && !shared) {
181     setOpening(false);
182     openAnime(saved.anime, saved.config, { calendar: saved.calendar, url: 'none' });
183     return;
184   }
185
186   setOpening(true);
187   const controller = startRequest();
188   getAnimeById(targetId, controller.signal)

```

```

189     .then(
190       (anime) => openAnime(anime, shared?.config ?? DEFAULT_CONFIG, { url: 'replace', fresh: true }),
191       () => {
192         if (controller.signal.aborted) return;
193         setUrl('/', 'replace');
194         setStep('search');
195       },
196     )
197     .finally(() => {
198       if (inFlight.current === controller) setOpenings(false);
199     });
200 };
201
202 // The listener is registered once; the ref always points at the latest
203 // resolver so it sees current state.
204 const resolveRef = useRef(resolveLocation);
205 resolveRef.current = resolveLocation;
206 useEffect(() => {
207   resolveRef.current();
208   const onPop = () => resolveRef.current();
209   window.addEventListener('popstate', onPop);
210   return () => window.removeEventListener('popstate', onPop);
211 }, []);
212
213 // If this show has a saved schedule, open it with that config.
214 const handleSelect = (anime: Anime) => {
215   const saved = savedSchedules.find((s) => s.anime.id === anime.id);
216   openAnime(anime, saved?.config ?? config, { calendar: saved?.calendar });
217 };
218
219 const handleOpenSaved = (saved: SavedSchedule) =>
220   openAnime(saved.anime, saved.config, { calendar: saved.calendar });
221
222 const handleOpenRelated = (id: number) => {
223   const controller = startRequest();
224   getAnimeById(id, controller.signal).then((a) => openAnime(a, config, { fresh: true }), () => {});
225 };
226
227 const handleSave = () => {
228   if (!selected) return;
229   const ok = saveSchedule(selected, config, calendar) !== null;
230   setSaveFailed(!ok);
231   if (ok) setSavedSchedules(getSavedSchedules());
232 };
233
234 const handleToggleEpisodes = (episodes: number[]) => {
235   if (!selected) return;
236   const target = savedForSelected ?? saveSchedule(selected, config, calendar) ?? undefined;
237   if (!target) return;
238   const updated = toggleEpisodesWatched(target.id, episodes);
239   if (updated) setSavedSchedules(getSavedSchedules());
240 };
241
242 const handleDeleteSaved = (id: string) => {
243   deleteSavedSchedule(id);
244   // Mirror the delete in memory even if the write failed.
245   setSavedSchedules((list) => list.filter((s) => s.id !== id));
246 };
247
248 const handleBack = () => {
249   inFlight.current?.abort();
250   setOpenings(false);
251   setStep('search');
252   setUrl('/', 'push');
253 };
254
255 const handleClearSearch = () => {
256   inFlight.current?.abort();
257   setQuery('');
258   setSearchFor(null);
259   setResults(null);
260   setError(null);
261   setLoading(false);
262 };
263
264 const handleHome = () => {
265   handleBack();

```

```

266     handleClearSearch();
267 };
268
269 const scheduleResult = useMemo(
270   () => (selected ? generateSchedule(selected, config) : null),
271   [selected, config],
272 );
273
274 const savedForSelected = selected
275   ? savedSchedules.find((s) => s.anime.id === selected.id)
276   : undefined;
277 const saveStatus: SaveStatus = saveFailed
278   ? 'failed'
279   : !savedForSelected
280     ? 'unsaved'
281     : savedForSelected.config.dailyBudget === config.dailyBudget &&
282       savedForSelected.config.skipOpEd === config.skipOpEd &&
283       sameCalendarPrefs(savedForSelected.calendar, calendar)
284     ? 'saved'
285     : 'changed';
286
287 // A show page takes on its poster's colors: App.tsx only sets `--poster`
288 // and `--poster-accent` (from `posterTheme`, lib/theme.ts); `.theme-root`
289 // in index.css derives every token from them. Unset on the homepage and
290 // for gray covers, which keep the brand navy and orange.
291 // Title, description, social tags, canonical URL and JSON-LD follow the
292 // open show (see lib/seo.ts, shared with the build-time prerender).
293 useEffect(() => {
294   const show = step === 'configure' && selected ? selected : null;
295   const seo = show ? animeSeo(show) : HOME_SEO;
296   document.title = seo.title;
297   metaTags(seo).forEach(upsertMeta);
298   if (!seo.image) {
299     for (const key of IMAGE_META_KEYS) {
300       document.head.querySelector(`meta[property="${key}"]`, meta[name="${key}"]`)?.remove();
301     }
302   }
303   document.head.querySelector<HTMLLinkElement>('link[rel="canonical"]')?.setAttribute('href', seo.url);
304   let ld = document.head.querySelector<HTMLScriptElement>('script[type="application/ld+json"]');
305   if (show) {
306     if (!ld) {
307       ld = document.createElement('script');
308       ld.type = 'application/ld+json';
309       document.head.appendChild(ld);
310     }
311     ld.textContent = JSON.stringify(animeJsonLd(show));
312   } else {
313     ld?.remove();
314   }
315 }, [step, selected]);
316
317 const showingSearchStep = step === 'search' || !selected || !scheduleResult;
318 const onHome = showingSearchStep && !opening && !loading && results === null && error === null;
319
320 const posterColors = useMemo(
321   () => (step === 'configure' ? posterTheme(selected?.coverImage.color) : null),
322   [step, selected?.coverImage.color],
323 );
324
325 // Each view (home, results, a show, loading) fades up when it replaces another.
326 const viewKey = opening && showingSearchStep
327   ? 'opening'
328   : showingSearchStep
329     ? onHome ? 'home' : 'results'
330     : `show-${selected?.id}`;
331 useEnter(viewRef, [viewKey], { y: 12 });
332
333 return (
334   <div
335     style={
336       posterColors
337       ? { ['--poster' as string]: posterColors.tint, ['--poster-accent' as string]: posterColors.accent }
338       : undefined
339     }
340     className="theme-root relative isolate flex min-h-screen flex-col overflow-x-clip text(--ink)"
341   >
342     <div aria-hidden="true" className="pointer-events-none absolute inset-0 -z-10">

```

```

343     <div
344       className="absolute -top-64 left-1/2 h-[36rem] w-[min(64rem,120vw)] -translate-x-1/2"
345       style={{
346         backgroundImage:
347           'radial-gradient(closest-side, color-mix(in srgb, var(--poster, var(--accent)) 14%, transparent) 35%,
transparent)',
348       }}
349     />
350   </div>
351
352   <header className="sticky top-0 z-30 border-b border-(--line) bg-(--bg)/70 backdrop-blur-xl transition-colors
duration-500">
353     <div className={` ${CONTAINER} flex h-16 items-center justify-between gap-4 sm:h-20 sm:gap-8`} >
354       <button
355         type="button"
356         onClick={handleHome}
357         className="flex shrink-0 items-center gap-3 text-base font-semibold tracking-tight text-(--ink)"
358       >
359         <span className="grid h-9 w-9 place-items-center bg-(--accent) text-xs font-black text-(--on-accent)
shadow-lg shadow-(--accent)/25 transition-colors duration-500">
360           AP
361         </span>
362         <span className={onHome ? 'inline' : 'hidden md:inline'}>
363           <GlitchText text="Anime Watch Planner" />
364         </span>
365       </button>
366       { /* On the home view the hero owns the search field. */
367       !onHome && (
368         <SearchBar query={query} onQueryChange={setQuery} onSearch={handleSearch} loading={loading} />
369       )
370     </div>
371   </header>
372
373   <main className={` ${CONTAINER} flex-1 py-8 sm:py-12 lg:py-16`} >
374     <div ref={viewRef}>
375       {opening && showingSearchStep ? (
376         <div role="status" className="flex flex-col items-center gap-4 py-24 text-(--ink-2)">
377           <Spinner />
378           Loading show...
379         </div>
380       ) : showingSearchStep ? (
381         <SearchStep
382           searchedFor={searchedFor}
383           results={results}
384           loading={loading}
385           error={error}
386           onSelect={handleSelect}
387           onClearSearch={handleClearSearch}
388           query={query}
389           onQueryChange={setQuery}
390           onSearch={handleSearch}
391           popular={popular}
392           savedSchedules={savedSchedules}
393           onOpenSaved={handleOpenSaved}
394           onDeleteSaved={handleDeleteSaved}
395         />
396       ) : (
397         <ConfigureStep
398           anime={selected}
399           config={config}
400           onConfigChange={setConfig}
401           calendar={calendar}
402           onCalendarChange={setCalendar}
403           result={scheduleResult}
404           onBack={handleBack}
405           saveStatus={saveStatus}
406           onSave={handleSave}
407           onOpenRelated={handleOpenRelated}
408           watched={savedForSelected?.watched ?? []}
409           onToggleEpisodes={handleToggleEpisodes}
410         />
411       )
412     </div>
413   </main>
414
415   <footer className="border-t border-(--line)">
416     <div className={` ${CONTAINER} py-8 text-sm text-(--ink-3)} >

```

```

417     Data from{ ' '}
418     <a href="https://anilist.co" className="text-(--ink-2) hover:text-(--ink) hover:underline">
419       Anilist
420     </a>
421   </div>
422 </footer>
423 </div>
424 );
425 }

```

src/main.tsx

13 lignes

Le point de départ : il trouve le `<div id="root">` de `index.html` et y affiche l'application, protégée par le filet de sécurité `ErrorBoundary`.

```

1  import { StrictMode } from 'react';
2  import { createRoot } from 'react-dom/client';
3  import App from './App';
4  import ErrorBoundary from './components/ErrorBoundary';
5  import './index.css';
6
7  createRoot(document.getElementById('root')!).render(
8    <StrictMode>
9      <ErrorBoundary>
10        <App />
11      </ErrorBoundary>
12    </StrictMode>,
13  );

```

A.9 Le script de build

scripts/prerender.ts

83 lignes

Lancé après la compilation : il demande à AniList les séries populaires et en tendance, écrit une page HTML par série avec ses propres balises SEO (pour les aperçus de liens et les moteurs de recherche), puis le plan du site `sitemap.xml`.

```

1  // Build step, run after `vite build` (see package.json). Writes a static
2  // page for each currently trending/popular show, so link previews (Discord,
3  // Slack, X...) and crawlers that don't run JavaScript see that show's title,
4  // description, cover and JSON-LD. Also writes sitemap.xml, and the default
5  // homepage tags into dist/index.html from the same source (src/lib/seo.ts).
6  //
7  // Each page is dist/index.html with only the <!--seo-start-->...<!--seo-end-->
8  // block swapped, so the app loads normally and takes over from the URL.
9  // Shows not prerendered still work through the /anime/* rewrite.
10 //
11 // Two AniList requests total, far under the rate limit. If AniList fails,
12 // this warns and still succeeds (homepage-only sitemap): the site works
13 // without these pages. Anything else (e.g. the marker block missing) is a
14 // bug and fails the build.
15
16 import { mkdir, readFile, writeFile } from 'node:fs/promises';
17 import { dirname, join } from 'node:path';
18 import { fileURLToPath } from 'node:url';
19 import { getPopularAiring, getTrendingAnime } from '../src/api/anilist';
20 import { displayTitle } from '../src/lib/format';
21 import { animePath } from '../src/lib/route';
22 import { generateSchedule } from '../src/lib/schedule';
23 import { HOME_SEO, SITE_URL, animeJsonLd, animeSeo, renderHeadTags } from '../src/lib/seo';
24 import type { Anime } from '../src/types';
25
26 const DIST = join(dirname(fileURLToPath(import.meta.url)), '..', 'dist');
27 const BLOCK = /<!--seo-start-->[\s\S]*?<!--seo-end-->/;
28
29 function withHead(shell: string, head: string): string {
30   return shell.replace(BLOCK, `<!--seo-start-->\n${head}\n    <!--seo-end-->`);
31 }
32

```

```

33 async function curatedShows(): Promise<Anime[]> {
34   const lists = await Promise.allSettled([getTrendingAnime(), getPopularAiring()]);
35   const byId = new Map<number, Anime>();
36   for (const list of lists) {
37     if (list.status === 'rejected') {
38       console.warn(`prerender: an AniList list failed: ${list.reason}`);
39       continue;
40     }
41     for (const anime of list.value) {
42       // Same rule as the homepage: skip shows a schedule can't be built for.
43       if (!byId.has(anime.id) && generateSchedule(anime, { dailyBudget: 90, skipOpEd: false })).ok) {
44         byId.set(anime.id, anime);
45       }
46     }
47   }
48   return [...byId.values()];
49 }
50
51 function sitemap(paths: string[], lastmod: string): string {
52   const urls = paths
53     .map((p) => `<url><loc>${SITE_URL}${p}</loc><lastmod>${lastmod}</lastmod></url>`)
54     .join('\n');
55   return `<?xml version="1.0" encoding="UTF-8"?>\n<urlset
56   xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">\n${urls}\n</urlset>\n`;
57 }
58
59 async function main() {
60   const shell = await readFile(join(DIST, 'index.html'), 'utf8');
61   if (!BLOCK.test(shell)) throw new Error('dist/index.html has no <!--seo-start--><!--seo-end--> block');
62
63   await writeFile(join(DIST, 'index.html'), withHead(shell, renderHeadTags(HOME_SEO)));
64
65   const shows = await curatedShows();
66   const paths = ['/'];
67   for (const anime of shows) {
68     const path = animePath(anime.id, displayTitle(anime));
69     // `<path>.html`, not `<path>/index.html`: Vercel (cleanUrls), Netlify and
70     // Cloudflare Pages all serve it at the extension-less URL.
71     const file = join(DIST, `${path}.html`);
72     await mkdir(dirname(file), { recursive: true });
73     await writeFile(file, withHead(shell, renderHeadTags(animeSeo(anime), animeJsonLd(anime))));
74     paths.push(path);
75   }
76
77   await writeFile(join(DIST, 'sitemap.xml'), sitemap(paths, new Date().toISOString().slice(0, 10)));
78   console.log(`prerender: ${shows.length} show pages + sitemap.xml (${paths.length} URLs)`);
79 }
80
81 main().catch((err) => {
82   console.error('prerender failed:', err);
83   process.exitCode = 1;
84 });

```

A.10 Les tests

src/lib/schedule.test.ts

185 lignes

Les tests du moteur de calcul : durée nette avec ou sans génériques, liste des épisodes (y compris pour les films et les séries en cours), répartition par jour, cas d'un épisode plus long que le temps quotidien.

```

1 import { describe, expect, it } from 'vitest';
2 import type { Anime, ScheduleConfig } from '../types';
3 import {
4   generateSchedule,
5   getNetEpisodeRuntime,
6   getWatchableEpisodes,
7 } from './schedule';
8
9 function anime(overrides: Partial<Anime> = {}): Anime {
10   return {
11     id: 1,
12     title: { romaji: 'Test', english: null },
13     format: 'TV',

```

```

14     status: 'FINISHED',
15     episodes: 24,
16     duration: 24,
17     seasonYear: 2020,
18     averageScore: 80,
19     coverImage: { large: null, color: null },
20     genres: [],
21     studios: [],
22     nextAiringEpisode: null,
23     trailer: null,
24     ...overrides,
25   };
26 }
27
28 const config = (overrides: Partial<ScheduleConfig> = {}): ScheduleConfig => ({
29   dailyBudget: 90,
30   skipOpEd: false,
31   ...overrides,
32 });
33
34 describe('getNetEpisodeRuntime', () => {
35   it('returns duration unchanged when skipOpEd is off', () => {
36     expect(getNetEpisodeRuntime(anime(), config())).toBe(24);
37   });
38
39   it('trims 3 minutes when skipOpEd is on', () => {
40     expect(getNetEpisodeRuntime(anime(), config({ skipOpEd: true }))).toBe(21);
41   });
42
43   it('trims at exactly 15 minutes', () => {
44     expect(getNetEpisodeRuntime(anime({ duration: 15 }), config({ skipOpEd: true }))).toBe(12);
45   });
46
47   it('does not trim shorts under 15 minutes', () => {
48     expect(getNetEpisodeRuntime(anime({ duration: 14 }), config({ skipOpEd: true }))).toBe(14);
49   });
50
51   it('does not trim movies', () => {
52     const movie = anime({ format: 'MOVIE', duration: 120, episodes: 1 });
53     expect(getNetEpisodeRuntime(movie, config({ skipOpEd: true }))).toBe(120);
54   });
55
56   it('returns null for unknown duration', () => {
57     expect(getNetEpisodeRuntime(anime({ duration: null }), config())).toBeNull();
58   });
59 });
60
61 describe('getWatchableEpisodes', () => {
62   it('lists every episode 1..N', () => {
63     expect(getWatchableEpisodes(anime({ episodes: 5 }))).toEqual({
64       episodes: [1, 2, 3, 4, 5],
65       isFallback: false,
66     });
67   });
68
69   it('always returns [1] for movies', () => {
70     expect(getWatchableEpisodes(anime({ format: 'MOVIE', episodes: 7 }))).toEqual({
71       episodes: [1],
72       isFallback: false,
73     });
74   });
75
76   it('returns null when episode count is unknown and not airing', () => {
77     expect(getWatchableEpisodes(anime({ episodes: null }))).toBeNull();
78   });
79
80   it('falls back to "aired so far" for an airing show with no published total', () => {
81     const airing = anime({
82       episodes: null,
83       status: 'RELEASING',
84       nextAiringEpisode: { episode: 1180, airingAt: 0 },
85     });
86     const result = getWatchableEpisodes(airing);
87     expect(result?.isFallback).toBe(true);
88     expect(result?.episodes).toHaveLength(1179);
89     expect(result?.episodes[0]).toBe(1);
90     expect(result?.episodes[1178]).toBe(1179);

```

```

91 });
92
93 it('returns null when the airing show has not aired anything yet', () => {
94     const notYetAired = anime({
95         episodes: null,
96         status: 'RELEASING',
97         nextAiringEpisode: { episode: 1, airingAt: 0 },
98     });
99     expect(getWatchableEpisodes(notYetAired)).toBeNull();
100 });
101 });
102
103 describe('generateSchedule', () => {
104     it('buckets floor(budget / runtime) episodes per day, greedily', () => {
105         const result = generateSchedule(anime({ episodes: 10 }), config());
106         if (!result.ok) throw new Error(result.error);
107         const s = result.schedule;
108         expect(s.episodesPerDay).toBe(3);
109         expect(s.unusedMinutesPerDay).toBe(18);
110         expect(s.days.map((d) => d.episodes)).toEqual([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]);
111         expect(s.days.map((d) => d.minutes)).toEqual([72, 72, 72, 24]);
112         expect(s.totalMinutes).toBe(240);
113         expect(s.overBudget).toBe(false);
114     });
115
116     it('uses the trimmed runtime when skipOpEd is on', () => {
117         const result = generateSchedule(anime({ episodes: 10 }), config({ skipOpEd: true }));
118         if (!result.ok) throw new Error(result.error);
119         expect(result.schedule.netRuntime).toBe(21);
120         expect(result.schedule.episodesPerDay).toBe(4);
121     });
122
123     it('schedules one per day and flags overBudget when an episode exceeds the budget', () => {
124         const movie = anime({ format: 'MOVIE', duration: 120, episodes: 1 });
125         const result = generateSchedule(movie, config({ dailyBudget: 90 }));
126         if (!result.ok) throw new Error(result.error);
127         expect(result.schedule.overBudget).toBe(true);
128         expect(result.schedule.days).toEqual([[{ day: 1, episodes: [1], minutes: 120 }]]);
129     });
130
131     it.each([0, -10, NaN])('rejects a budget of %s', (dailyBudget) => {
132         expect(generateSchedule(anime(), config({ dailyBudget }))).toEqual({
133             ok: false,
134             error: 'invalid-budget',
135         });
136     });
137
138     it('errors on unknown episode count or duration', () => {
139         expect(generateSchedule(anime({ episodes: null }), config())).toEqual({
140             ok: false,
141             error: 'unknown-episode-count',
142         });
143         expect(generateSchedule(anime({ duration: null }), config())).toEqual({
144             ok: false,
145             error: 'unknown-duration',
146         });
147     });
148
149     it('treats a zero episode count as unknown', () => {
150         expect(generateSchedule(anime({ episodes: 0 }), config())).toEqual({
151             ok: false,
152             error: 'unknown-episode-count',
153         });
154     });
155
156     it('is not partial for a normal, fully-known show', () => {
157         const result = generateSchedule(anime({ episodes: 10 }), config());
158         if (!result.ok) throw new Error(result.error);
159         expect(result.schedule.isPartial).toBe(false);
160     });
161
162     it('schedules an airing show with no published total using episodes aired so far', () => {
163         const airing = anime({
164             episodes: null,
165             status: 'RELEASING',
166             nextAiringEpisode: { episode: 4, airingAt: 0 },
167         });

```

```

168     const result = generateSchedule(airing, config());
169     if (!result.ok) throw new Error(result.error);
170     expect(result.schedule.isPartial).toBe(true);
171     expect(result.schedule.episodeCount).toBe(3);
172   });
173
174   it('still errors when an airing show has not aired anything yet', () => {
175     const notYetAired = anime({
176       episodes: null,
177       status: 'RELEASING',
178       nextAiringEpisode: { episode: 1, airingAt: 0 },
179     });
180     expect(generateSchedule(notYetAired, config())).toEqual({
181       ok: false,
182       error: 'unknown-episode-count',
183     });
184   });
185 });

```

src/lib/format.test.ts

119 lignes

Les tests des fonctions d'affichage et de sécurité : titres d'épisodes associés au bon numéro, liens et couleurs filtrés, adresses YouTube, pages d'épisodes, adresses lisibles, nettoyage des descriptions.

```

1  import { describe, expect, it } from 'vitest';
2  import { episodeTitleMap, safeExternalUrl, safeHexColor, youtubeEmbedUrl, describeEpisodes, plainText, slugify,
   synopsisText } from './format';
3
4  describe('episodeTitleMap', () => {
5    it('keys titles by their parsed episode number, not list position', () => {
6      // Shaped like One Piece's real data: reversed and starting mid-series.
7      const map = episodeTitleMap([
8        { title: 'Episode 130 - Scent of Danger' },
9        { title: 'Episode 129 - The Last Stand' },
10     ]);
11     expect(map.get(130)).toBe('Scent of Danger');
12     expect(map.get(129)).toBe('The Last Stand');
13     expect(map.get(1)).toBeUndefined();
14   });
15
16   it('handles colon and missing separators', () => {
17     const map = episodeTitleMap([
18       { title: 'Episode 3: Killing Magic' },
19       { title: 'Episode 4 Rest' }
20     ]);
21     expect(map.get(3)).toBe('Killing Magic');
22     expect(map.get(4)).toBe('Rest');
23   });
24
25   it('drops titles without a parseable number instead of guessing', () => {
26     const map = episodeTitleMap([
27       { title: 'Special Recap' },
28       { title: 'Episode 2 - Real' }
29     ]);
30     expect([...map.keys()]).toEqual([2]);
31   });
32
33   it('keeps the first title for a duplicated number', () => {
34     const map = episodeTitleMap([
35       { title: 'Episode 1 - Sub' },
36       { title: 'Episode 1 - Dub' }
37     ]);
38     expect(map.get(1)).toBe('Sub');
39   });
40
41   it('returns an empty map for undefined input', () => {
42     expect(episodeTitleMap(undefined).size).toBe(0);
43   });
44 });
45
46 describe('safeExternalUrl', () => {
47   it('keeps https and upgrades http', () => {
48     expect(safeExternalUrl('https://www.crunchyroll.com/x')).toBe('https://www.crunchyroll.com/x');
49     expect(safeExternalUrl('http://www.hulu.com/one-piece')).toBe('https://www.hulu.com/one-piece');
50   });
51
52   it.each([
53     'javascript:alert(1)',
54     'JaVaScRiPt:alert(1)',
55     'data:text/html,<script>',
56     'vbscript:x',
57     'not a url',
58     ''
59   ])('rejects %s', (url) => {
60     expect(safeExternalUrl(url)).toBeNull();
61   });
62 });

```

```

49 });
50
51 describe('safeHexColor', () => {
52   it('accepts hex colors and rejects anything else', () => {
53     expect(safeHexColor('#F88B24')).toBe('#F88B24');
54     expect(safeHexColor('#abc')).toBe('#abc');
55     expect(safeHexColor('red; background:url(https://x)')).toBeNull();
56     expect(safeHexColor('url(https://x)')).toBeNull();
57     expect(safeHexColor(null)).toBeNull();
58   });
59 });
60
61 describe('youtubeEmbedUrl', () => {
62   it('builds a nocookie embed URL for a valid YouTube id', () => {
63     expect(youtubeEmbedUrl({ id: 'tR8YH0G67Rk', site: 'youtube' })).toBe(
64       'https://www.youtube-nocookie.com/embed/tR8YH0G67Rk?rel=0',
65     );
66   });
67
68   it('rejects other sites and malformed ids', () => {
69     expect(youtubeEmbedUrl({ id: 'x1y2z3', site: 'dailymotion' })).toBeNull();
70     expect(youtubeEmbedUrl({ id: '../..//evil?x=1', site: 'youtube' })).toBeNull();
71     expect(youtubeEmbedUrl(null)).toBeNull();
72   });
73 });
74
75 describe('describeEpisodes', () => {
76   it('collapses runs into ranges', () => {
77     expect(describeEpisodes([1, 2, 3, 5])).toBe('Ep 1-3, 5');
78     expect(describeEpisodes([7])).toBe('Ep 7');
79   });
80 });
81
82 describe('slugify', () => {
83   it('makes lowercase dash-separated slugs and folds accents', () => {
84     expect(slugify('Re:ZERO -Starting Life-')).toBe('re-zero-starting-life');
85     expect(slugify('Pokémon')).toBe('pokemon');
86     expect(slugify('ワンピース')).toBe('');
87   });
88 });
89
90 describe('plainText', () => {
91   it('drops tags and collapses whitespace', () => {
92     expect(plainText('A <i>B</i><br><br>C & D')).toBe('A B C & D');
93   });
94
95   it('decodes & last, so an escaped entity stays escaped', () => {
96     expect(plainText('Tom &amp; Jerry &lt;3')).toBe('Tom & Jerry &lt;3');
97   });
98 });
99
100 describe('synopsisText', () => {
101   it('turns <br> into line breaks and drops every other tag', () => {
102     // Shaped like Death Note's real Anilist description (asHtml: false).
103     const raw =
104       'He would never turn back.\n<br><br>\n(Source: Crunchyroll)<br>\n<br>\n<i>Note: The first episode aired early.</i>';
105     expect(synopsisText(raw)).toBe(
106       'He would never turn back.\n\n(Source: Crunchyroll)\n\nNote: The first episode aired early.',
107     );
108   });
109
110   it('removes <b> and decodes entities', () => {
111     expect(synopsisText('<b>Luffy</b> & crew say "hi" &lt;3 &#039;s')).toBe(
112       'Luffy & crew say "hi" <3 \'s',
113     );
114   });
115
116   it('keeps an escaped entity escaped', () => {
117     expect(synopsisText('&lt;i&gt;')).toBe('&lt;i&gt;');
118   });
119 });

```

Les tests des couleurs d'une série : couleurs absentes, invalides ou grises, couleurs utilisées telles quelles, couleurs pâles assombries jusqu'à un contraste lisible en gardant leur teinte, et une vérification de chaque paire texte et fond sur 4 096 couleurs.

```

1  import { describe, expect, it } from 'vitest';
2  import { contrastRatio, posterTheme } from './theme';
3
4  const WHITE = '#ffffff';
5
6  const channels = (color: string) => [1, 3, 5].map((i) => parseInt(color.slice(i, i + 2), 16));
7  const hex = (rgb: number[]) => `#${rgb.map((v) => Math.round(v).toString(16).padStart(2, '0')).join('')}`;
8  /** `color-mix(in srgb, a p, b)` for opaque colors. */
9  const mix = (a: string, b: string, p: number) => {
10     const cb = channels(b);
11     return hex(channels(a).map((v, i) => v * p + cb[i] * (1 - p)));
12 };
13
14 describe('posterTheme', () => {
15     it('keeps the brand colors without a usable cover color', () => {
16         expect(posterTheme(null)).toBeNull();
17         expect(posterTheme(undefined)).toBeNull();
18         expect(posterTheme('red')).toBeNull();
19         // Gray covers have no hue to take on.
20         expect(posterTheme('#e4e4e4')).toBeNull();
21         expect(posterTheme('#5d5d5d')).toBeNull();
22         expect(posterTheme('#1a1a1a')).toBeNull();
23     });
24
25     it('uses a cover color that already carries white text as the accent', () => {
26         expect(posterTheme('#6b1a1a').accent).toBe('#6b1a1a');
27         expect(posterTheme('#6B1A1A').accent).toBe('#6b1a1a');
28     });
29
30     it('darkens a pale cover until white text on it reads at 4.5:1, keeping its hue', () => {
31         expect(contrastRatio(WHITE, '#bbf1a1')).toBeLessThan(4.5);
32         const accent = posterTheme('#bbf1a1').accent; // Frieren's pale green
33         expect(contrastRatio(WHITE, accent)).toBeGreaterThanOrEqual(4.5);
34         const [r, g, b] = channels(accent);
35         expect(g).toBeGreaterThan(r);
36         expect(g).toBeGreaterThan(b);
37     });
38
39     it('lightens the tint of a dark cover but keeps a light one as is', () => {
40         const tint = posterTheme('#6b1a1a').tint;
41         expect(contrastRatio(tint, '#000000')).toBeGreaterThanOrEqual(contrastRatio('#6b1a1a', '#000000'));
42         expect(posterTheme('#bbf1a1').tint).toBe('#bbf1a1');
43     });
44
45     it('accepts 3-digit hex', () => {
46         expect(posterTheme('#c33')).toEqual(posterTheme('#cc3333'));
47     });
48 });
49
50 // Mirrors the `.theme-root` formulas in index.css. If you change a
51 // percentage there, change it here too and rerun this.
52 describe('theme contrast across cover colors', () => {
53     const NAVY = '#133055';
54     const CREAM = '#f3ede4';
55     const ink2 = mix(NAVY, CREAM, 0.82);
56     const ink3 = mix(NAVY, CREAM, 0.74);
57
58     // The homepage, then 4096 cover colors spread over the whole sRGB cube.
59     const themes = [{ tint: CREAM, accent: '#bf4a1b' }];
60     for (let r = 0; r < 256; r += 17)
61         for (let g = 0; g < 256; g += 17)
62             for (let b = 0; b < 256; b += 17) {
63                 const theme = posterTheme(hex([r, g, b]));
64                 if (theme) themes.push(theme);
65             }
66
67     // The lowest ratio each pair reaches over all of them.
68     const lowest = new Map<string, number>();
69     const check = (pair: string, a: string, b: string) =>

```

```

70   lowest.set(pair, Math.min(lowest.get(pair) ?? Infinity, contrastRatio(a, b)));
71   for (const { tint, accent } of themes) {
72     const bg = mix(tint, CREAM, 0.16);
73     const surface = mix(tint, '#fbf8f3', 0.08);
74     const surface2 = mix(tint, '#eae2d6', 0.22);
75     const accentText = mix(accent, '#000000', 0.75);
76     const chip = mix(accent, surface, 0.15); // bg-(--accent)/15 on a card
77     check('white on accent', WHITE, accent);
78     check('accent shapes on the page', accent, bg);
79     for (const [name, back] of Object.entries({ bg, surface, surface2, chip })) {
80       check(`accent text on ${name}`, accentText, back);
81     }
82     for (const [name, back] of Object.entries({ bg, surface, surface2 })) {
83       check(`ink on ${name}`, NAVY, back);
84       check(`ink-2 on ${name}`, ink2, back);
85       check(`ink-3 on ${name}`, ink3, back);
86     }
87   }
88
89   it('themes most cover colors', () => {
90     expect(themes.length).toBeGreaterThan(3500);
91   });
92
93   it('keeps text at 4.5:1 or more where it sits', () => {
94     for (const [pair, ratio] of lowest) {
95       // Shapes (bars, rings) need 3:1. Hints on the darker panel are small
96       // print and get 4:1.
97       const min = pair === 'accent shapes on the page' ? 3 : pair === 'ink-3 on surface2' ? 4 : 4.5;
98       expect(ratio, pair).toBeGreaterThanOrEqual(min);
99     }
100   });
101 });

```

src/lib/calendar.test.ts

162 lignes

Les tests des dates : jours valides, passage d'un mois ou d'une année, jours de visionnage sautés, changement d'heure sans décalage, grilles de mois, dates de diffusion, fuseaux horaires.

```

1  import { describe, expect, it } from 'vitest';
2  import type { Anime, ScheduleDay } from '../types';
3  import {
4    addDays,
5    defaultCalendarPrefs,
6    finishDate,
7    formatAiring,
8    isIsoDate,
9    isTimeZone,
10   listTimeZones,
11   sameCalendarPrefs,
12   isWatchTime,
13   localIso,
14   monthGrid,
15   scheduleDates,
16   shiftMonth,
17   todayIso,
18   upcomingAirings,
19   weekday,
20 } from './calendar';
21
22 function days(n: number): ScheduleDay[] {
23   return Array.from({ length: n }, (_, i) => ({ day: i + 1, episodes: [i + 1], minutes: 24 }));
24 }
25
26 describe('date helpers', () => {
27   it('validates real dates and times only', () => {
28     expect(isIsoDate('2026-02-28')).toBe(true);
29     expect(isIsoDate('2026-02-30')).toBe(false);
30     expect(isIsoDate('2026-2-3')).toBe(false);
31     expect(isWatchTime('20:30')).toBe(true);
32     expect(isWatchTime('24:00')).toBe(false);
33     expect(isWatchTime('8:00')).toBe(false);
34   });
35
36   it('adds days across month and year ends', () => {

```

```

37     expect(addDays('2026-01-31', 1)).toBe('2026-02-01');
38     expect(addDays('2026-12-31', 1)).toBe('2027-01-01');
39     expect(addDays('2026-03-01', -1)).toBe('2026-02-28');
40   });
41
42   it('knows the weekday', () => {
43     expect(weekday('2026-10-03')).toBe(6); // Saturday
44     expect(weekday('2026-10-05')).toBe(1); // Monday
45   });
46
47   it('turns an instant into its local date', () => {
48     expect(localIso(new Date(2026, 9, 3, 23, 59))).toBe('2026-10-03');
49   });
50 });
51
52 describe('scheduleDates', () => {
53   it('uses consecutive days when every day is a watch day', () => {
54     const dates = scheduleDates(days(3), defaultCalendarPrefs('2026-10-03'));
55     expect([...dates.values()]).toEqual(['2026-10-03', '2026-10-04', '2026-10-05']);
56   });
57
58   it('starts on the first watch day on or after the start date', () => {
59     // Saturday start, weekdays only -> Monday, Tuesday.
60     const prefs = { ...defaultCalendarPrefs('2026-10-03'), watchDays: [1, 2, 3, 4, 5] };
61     expect([...scheduleDates(days(2), prefs).values()]).toEqual(['2026-10-05', '2026-10-06']);
62   });
63
64   it('skips non-watch days in the middle of the plan', () => {
65     const prefs = { ...defaultCalendarPrefs('2026-10-03'), watchDays: [0, 6] };
66     expect([...scheduleDates(days(3), prefs).values()]).toEqual([
67       '2026-10-03',
68       '2026-10-04',
69       '2026-10-10',
70     ]);
71   });
72
73   it('is not shifted by a daylight-saving change', () => {
74     // US and EU clocks change on these weekends.
75     const us = scheduleDates(days(3), defaultCalendarPrefs('2026-03-07'));
76     expect([...us.values()]).toEqual(['2026-03-07', '2026-03-08', '2026-03-09']);
77     const eu = scheduleDates(days(3), defaultCalendarPrefs('2026-10-24'));
78     expect([...eu.values()]).toEqual(['2026-10-24', '2026-10-25', '2026-10-26']);
79   });
80
81   it('treats an empty watch-day list as every day instead of looping forever', () => {
82     const prefs = { ...defaultCalendarPrefs('2026-10-03'), watchDays: [] };
83     expect(scheduleDates(days(2), prefs).get(2)).toBe('2026-10-04');
84   });
85
86   it('finishes on the last scheduled date', () => {
87     const prefs = { ...defaultCalendarPrefs('2026-10-03'), watchDays: [6] };
88     expect(finishDate(days(3), prefs)).toBe('2026-10-17');
89     expect(finishDate([], prefs)).toBeNull();
90   });
91 });
92
93 describe('monthGrid', () => {
94   it('covers the month in full Monday-first weeks', () => {
95     // October 2026 starts on a Thursday and ends on a Saturday.
96     const grid = monthGrid({ year: 2026, month: 9 });
97     expect(grid).toHaveLength(5);
98     expect(grid.every((week) => week.length === 7)).toBe(true);
99     expect(grid[0][0]).toEqual({ iso: '2026-09-28', inMonth: false });
100    expect(grid[0][3]).toEqual({ iso: '2026-10-01', inMonth: true });
101    expect(grid[4][6]).toEqual({ iso: '2026-11-01', inMonth: false });
102  });
103
104   it('shifts months across year boundaries', () => {
105     expect(shiftMonth({ year: 2026, month: 11 }, 1)).toEqual({ year: 2027, month: 0 });
106     expect(shiftMonth({ year: 2026, month: 0 }, -1)).toEqual({ year: 2025, month: 11 });
107   });
108 });
109
110 describe('upcomingAirings', () => {
111   const base = { nextAiringEpisode: null } as unknown as Anime;
112   const now = Date.UTC(2026, 9, 1);
113

```

```

114 it('prefers the detailed airing list, sorted, future only', () => {
115     const anime = {
116         ...base,
117         upcomingEpisodes: [
118             { episode: 11, airingAt: Date.UTC(2026, 9, 17) / 1000 },
119             { episode: 10, airingAt: Date.UTC(2026, 9, 10) / 1000 },
120             { episode: 9, airingAt: Date.UTC(2026, 8, 30) / 1000 },
121         ],
122     };
123     expect(upcomingAirings(anime, now).map((a) => a.episode)).toEqual([10, 11]);
124 });
125
126 it('falls back to nextAiringEpisode', () => {
127     const anime = { ...base, nextAiringEpisode: { episode: 10, airingAt: Date.UTC(2026, 9, 10) / 1000 } };
128     expect(upcomingAirings(anime, now).map((a) => a.episode)).toEqual([10]);
129 });
130 });
131
132 describe('time zones', () => {
133     // 2026-10-03 20:00 UTC: still Oct 3 in New York, already Oct 4 in Tokyo.
134     const instant = new Date(Date.UTC(2026, 9, 3, 20, 0));
135
136     it('gives the date of an instant in a chosen zone', () => {
137         expect(localIso(instant, 'America/New_York')).toBe('2026-10-03');
138         expect(localIso(instant, 'Asia/Tokyo')).toBe('2026-10-04');
139         expect(todayIso(instant, 'Asia/Tokyo')).toBe('2026-10-04');
140     });
141
142     it('dates and labels air times in the chosen zone', () => {
143         const anime = { nextAiringEpisode: { episode: 5, airingAt: instant.getTime() / 1000 } } as Anime;
144         expect(upcomingAirings(anime, 0, 'Asia/Tokyo')[0].date).toBe('2026-10-04');
145         expect(upcomingAirings(anime, 0, 'America/New_York')[0].date).toBe('2026-10-03');
146         const tokyo = formatAiring(instant.getTime() / 1000, 'Asia/Tokyo', 'en-US');
147         expect(tokyo).toEqual({ day: 'Sun, Oct 4', time: '5:00 AM' });
148     });
149
150     it('validates zone names and lists some', () => {
151         expect(isTimeZone('Europe/Paris')).toBe(true);
152         expect(isTimeZone('Mars/Base')).toBe(false);
153         expect(isTimeZone('')).toBe(false);
154         expect(listTimeZones()).toContain('Asia/Tokyo');
155     });
156
157     it('treats a zone change as a calendar change', () => {
158         const a = defaultCalendarPrefs('2026-10-03');
159         expect(sameCalendarPrefs(a, { ...a })).toBe(true);
160         expect(sameCalendarPrefs(a, { ...a, timezone: 'Asia/Tokyo' })).toBe(false);
161     });
162 });

```

src/lib/progress.test.ts

98 lignes

Les tests de la progression : jour de reprise, pourcentage, progression conservée quand les jours sont redécoupés, et calcul du streak.

```

1 import { describe, expect, it } from 'vitest';
2 import type { SavedSchedule, ScheduleDay, WatchedEpisode } from '../types';
3 import { getCurrentStreak, getProgress, getResumeDay } from './progress';
4 import { defaultCalendarPrefs } from './calendar';
5
6 const DAY = 86_400_000;
7
8 function schedule(watched: WatchedEpisode[]): SavedSchedule {
9     return {
10         id: '1',
11         savedAt: 0,
12         anime: {} as SavedSchedule['anime'],
13         config: { dailyBudget: 60, skipOpEd: false },
14         calendar: defaultCalendarPrefs('2026-01-01'),
15         watched,
16     };
17 }
18
19 /** 6 episodes bucketed `perDay` at a time. */

```

```

20 function days(perDay: number): ScheduleDay[] {
21   const out: ScheduleDay[] = [];
22   for (let ep = 1; ep <= 6; ep += perDay) {
23     const episodes = Array.from({ length: Math.min(perDay, 7 - ep) }, (_, i) => ep + i);
24     out.push({ day: out.length + 1, episodes, minutes: 0 });
25   }
26   return out;
27 }
28
29 const w = (...eps: number[]): WatchedEpisode[] => eps.map((episode) => ({ episode, completedAt: 0 }));
30
31 describe('getResumeDay', () => {
32   it('returns the first day not fully watched', () => {
33     expect(getResumeDay(days(2), w(1, 2))).toBe(2);
34   });
35
36   it('treats a partly watched day as not done', () => {
37     expect(getResumeDay(days(2), w(1))).toBe(1);
38   });
39
40   it('returns null when every day is watched', () => {
41     expect(getResumeDay(days(3), w(1, 2, 3, 4, 5, 6))).toBeNull();
42   });
43 });
44
45 describe('getProgress', () => {
46   it('counts fully watched days', () => {
47     expect(getProgress(days(2), w(1, 2, 3, 4))).toEqual({ done: 2, total: 3, percent: 67 });
48   });
49
50   it('handles zero days', () => {
51     expect(getProgress([], [])).toEqual({ done: 0, total: 0, percent: 0 });
52   });
53
54   it('keeps the same episodes watched when the budget re-buckets days', () => {
55     const watched = w(1, 2, 3, 4);
56     // 2 per day: days 1-2 done. 3 per day: day 1 done, day 2 half done.
57     expect(getResumeDay(days(2), watched)).toBe(3);
58     expect(getResumeDay(days(3), watched)).toBe(2);
59     expect(getProgress(days(3), watched).done).toBe(1);
60   });
61 });
62
63 describe('getCurrentStreak', () => {
64   const now = Date.UTC(2026, 0, 15, 12); // 2026-01-15 noon UTC
65
66   it('is 0 with nothing watched', () => {
67     expect(getCurrentStreak([], now)).toBe(0);
68   });
69
70   it('counts today alone as a 1-day streak', () => {
71     expect(getCurrentStreak([schedule([episode: 1, completedAt: now])], now)).toBe(1);
72   });
73
74   it('counts a streak ending yesterday even if today is empty', () => {
75     const yesterday = now - DAY;
76     expect(getCurrentStreak([schedule([episode: 1, completedAt: yesterday])], now)).toBe(1);
77   });
78
79   it('breaks on a gap older than yesterday', () => {
80     const twoDaysAgo = now - 2 * DAY;
81     expect(getCurrentStreak([schedule([episode: 1, completedAt: twoDaysAgo])], now)).toBe(0);
82   });
83
84   it('counts consecutive days across multiple schedules', () => {
85     const s1 = schedule([episode: 1, completedAt: now]);
86     const s2 = schedule([episode: 1, completedAt: now - DAY], { episode: 2, completedAt: now - 2 * DAY });
87     expect(getCurrentStreak([s1, s2], now)).toBe(3);
88   });
89
90   it('only counts one streak-day per calendar date, not per watched entry', () => {
91     const s = schedule([
92       { episode: 1, completedAt: now },
93       { episode: 2, completedAt: now },
94       { episode: 3, completedAt: now - DAY },
95     ]);
96     expect(getCurrentStreak([s], now)).toBe(2);

```

```
97 });
98 });
```

src/lib/storage.test.ts

197 lignes

Les tests de la sauvegarde, avec un faux `localStorage` en mémoire : enregistrer, remplacer, supprimer, résister à des données corrompues ou à un stockage en panne, cocher des épisodes, convertir les anciens formats.

```
1  import { afterEach, beforeEach, describe, expect, it, vi } from 'vitest';
2  import type { Anime, ScheduleConfig } from '../types';
3  import {
4    STORAGE_KEY,
5    deleteSavedSchedule,
6    getSavedSchedules,
7    saveSchedule,
8    toggleEpisodesWatched,
9  } from './storage';
10
11  /** Minimal in-memory Storage so tests don't depend on a DOM environment. */
12  function memoryStorage(): Storage {
13    const data = new Map<string, string>();
14    return {
15      get length() {
16        return data.size;
17      },
18      clear: () => data.clear(),
19      getItem: (k) => data.get(k) ?? null,
20      key: (i) => [...data.keys()][i] ?? null,
21      removeItem: (k) => void data.delete(k),
22      setItem: (k, v) => void data.set(k, String(v)),
23    };
24  }
25
26  function anime(id: number): Anime {
27    return {
28      id,
29      title: { romaji: `Show ${id}`, english: null },
30      format: 'TV',
31      status: 'FINISHED',
32      episodes: 12,
33      duration: 24,
34      seasonYear: 2020,
35      averageScore: 80,
36      coverImage: { large: null, color: null },
37      genres: [],
38      studios: [],
39      nextAiringEpisode: null,
40      trailer: null,
41    };
42  }
43
44  const config: ScheduleConfig = { dailyBudget: 60, skipOpEd: false };
45
46  beforeEach(() => {
47    vi.stubGlobal('localStorage', memoryStorage());
48  });
49
50  afterEach(() => {
51    vi.unstubAllGlobals();
52    vi.restoreAllMocks();
53  });
54
55  describe('storage', () => {
56    it('returns [] when nothing is saved', () => {
57      expect(getSavedSchedules()).toEqual([]);
58    });
59
60    it('round-trips a saved schedule', () => {
61      const saved = saveSchedule(anime(1), config);
62      expect(saved).not.toBeNull();
63      expect(getSavedSchedules()).toEqual([saved]);
64    });
65
66    it('replaces the entry for the same anime instead of duplicating', () => {
```

```

67     saveSchedule(anime(1), config);
68     saveSchedule(anime(1), { dailyBudget: 120, skipOpEd: true });
69     const list = getSavedSchedules();
70     expect(list).toHaveLength(1);
71     expect(list[0].config).toEqual({ dailyBudget: 120, skipOpEd: true });
72   });
73
74   it('lists newest first', () => {
75     vi.spyOn(Date, 'now').mockReturnValueOnce(1000).mockReturnValueOnce(2000);
76     saveSchedule(anime(1), config);
77     saveSchedule(anime(2), config);
78     expect(getSavedSchedules().map((s) => s.anime.id)).toEqual([2, 1]);
79   });
80
81   it('deletes by id', () => {
82     const a = saveSchedule(anime(1), config)!;
83     saveSchedule(anime(2), config);
84     expect(deleteSavedSchedule(a.id)).toBe(true);
85     expect(getSavedSchedules().map((s) => s.anime.id)).toEqual([2]);
86   });
87
88   it('returns [] for corrupt JSON', () => {
89     localStorage.setItem(STORAGE_KEY, 'not json');
90     expect(getSavedSchedules()).toEqual([]);
91   });
92
93   it('drops entries that fail validation', () => {
94     const good = saveSchedule(anime(1), config)!;
95     localStorage.setItem(STORAGE_KEY, JSON.stringify([good, { id: 5 }, null]));
96     expect(getSavedSchedules()).toEqual([good]);
97   });
98
99   it('degrades gracefully when storage throws', () => {
100     vi.spyOn(localStorage, 'getItem').mockImplementation(() => {
101       throw new Error('denied');
102     });
103     vi.spyOn(localStorage, 'setItem').mockImplementation(() => {
104       throw new Error('quota');
105     });
106     expect(getSavedSchedules()).toEqual([]);
107     expect(saveSchedule(anime(1), config)).toBeNull();
108   });
109
110   it('toggles a day of episodes on, then off', () => {
111     const saved = saveSchedule(anime(1), config)!;
112     expect(toggleEpisodesWatched(saved.id, [1, 2])!.watched.map((w) => w.episode)).toEqual([1, 2]);
113     expect(toggleEpisodesWatched(saved.id, [1, 2])!.watched).toEqual([]);
114   });
115
116   it('marks only the missing episodes of a partly watched day', () => {
117     const saved = saveSchedule(anime(1), config)!;
118     toggleEpisodesWatched(saved.id, [1]);
119     expect(toggleEpisodesWatched(saved.id, [1, 2])!.watched.map((w) => w.episode)).toEqual([1, 2]);
120   });
121
122   it('migrates old per-day progress to the episodes that day held', () => {
123     // 60-min budget, 24-min episodes -> 2 per day: day 2 = episodes 3-4.
124     const saved = saveSchedule(anime(1), config)!;
125     const legacy = { ...saved, watched: [{ day: 2, completedAt: 123 }] };
126     localStorage.setItem(STORAGE_KEY, JSON.stringify([legacy]));
127     expect(getSavedSchedules()[0].watched).toEqual([
128       { episode: 3, completedAt: 123 },
129       { episode: 4, completedAt: 123 },
130     ]);
131   });
132
133   it('fills in default calendar prefs for records saved before they existed', () => {
134     const savedAt = new Date(2026, 4, 2, 12).getTime();
135     localStorage.setItem(
136       STORAGE_KEY,
137       JSON.stringify([{ id: 'x', savedAt, anime: anime(1), config, watched: [] }]),
138     );
139     expect(getSavedSchedules()[0].calendar).toEqual({
140       startDate: '2026-05-02',
141       watchDays: [0, 1, 2, 3, 4, 5, 6],
142       watchTime: null,
143       reminderMinutes: null,

```

```

144     timezone: null,
145   });
146 });
147
148 it('replaces invalid calendar fields with defaults and keeps valid ones', () => {
149   localStorage.setItem(
150     STORAGE_KEY,
151     JSON.stringify([
152       {
153         id: 'x',
154         savedAt: 0,
155         anime: anime(1),
156         config,
157         watched: [],
158         calendar: { startDate: '2026-10-03', watchDays: [6, 9, 'x', 6, 1], watchTime: '25:00', reminderMinutes:
10 },
159       },
160     ]),
161   );
162   const { calendar } = getSavedSchedules()[0];
163   expect(calendar.startDate).toBe('2026-10-03');
164   expect(calendar.watchDays).toEqual([1, 6]);
165   expect(calendar.watchTime).toBeNull();
166   // A reminder is meaningless without a time.
167   expect(calendar.reminderMinutes).toBeNull();
168 });
169
170 it('keeps a valid time zone and drops an unknown one', () => {
171   for (const [stored, expected] of [
172     ['Asia/Tokyo', 'Asia/Tokyo'],
173     ['Mars/Olympus Mons', null],
174     [42, null],
175     [undefined, null],
176   ] as const) {
177     localStorage.clear();
178     localStorage.setItem(
179       STORAGE_KEY,
180       JSON.stringify([
181         {
182           id: '1', savedAt: Date.UTC(2026, 9, 3), anime: anime(1), config, watched: [],
183           calendar: { startDate: '2026-10-03', watchDays: [6], watchTime: null, reminderMinutes: null, timezone:
stored },
184         },
185       ]),
186     );
187     expect(getSavedSchedules()[0].calendar.timezone).toBe(expected);
188   }
189 });
190
191 it('keeps calendar prefs when re-saving without new ones', () => {
192   const calendar = { startDate: '2026-10-03', watchDays: [6], watchTime: '20:00', reminderMinutes: 30, timezone:
'Asia/Tokyo' };
193   saveSchedule(anime(1), config, calendar);
194   saveSchedule(anime(1), { dailyBudget: 120, skipOpEd: true });
195   expect(getSavedSchedules()[0].calendar).toEqual(calendar);
196 });
197 });

```

src/lib/ics.test.ts

141 lignes

Les tests du fichier calendrier : un événement par jour aux bonnes dates, titres d'épisodes, caractères spéciaux échappés, événements horaires, rappels, dates de diffusion.

```

1  import { describe, expect, it } from 'vitest';
2  import type { Anime, Schedule } from '../types';
3  import { buildIcs } from './ics';
4  import { defaultCalendarPrefs } from './calendar';
5
6  const prefs = defaultCalendarPrefs('2026-01-15');
7  const NOW = new Date(Date.UTC(2026, 0, 1));
8
9  function anime(): Anime {
10     return {
11       id: 42,

```

```

12     title: { romaji: 'Test Show', english: null },
13     format: 'TV',
14     status: 'FINISHED',
15     episodes: 4,
16     duration: 24,
17     seasonYear: 2020,
18     averageScore: 80,
19     coverImage: { large: null, color: null },
20     genres: [],
21     studios: [],
22     nextAiringEpisode: null,
23     trailer: null,
24   };
25 }
26
27 const schedule: Schedule = {
28   days: [
29     { day: 1, episodes: [1, 2], minutes: 48 },
30     { day: 2, episodes: [3, 4], minutes: 48 },
31   ],
32   netRuntime: 24,
33   episodesPerDay: 2,
34   episodeCount: 4,
35   totalMinutes: 96,
36   unusedMinutesPerDay: 0,
37   overBudget: false,
38   isPartial: false,
39 };
40
41 describe('buildIcs', () => {
42   it('produces one VEVENT per day with correct dates', () => {
43     const ics = buildIcs(anime(), schedule, prefs);
44     expect(ics.match(/BEGIN:VEVENT/g)).toHaveLength(2);
45     expect(ics).toContain('DTSTART;VALUE=DATE:20260115');
46     expect(ics).toContain('DTEND;VALUE=DATE:20260116');
47     expect(ics).toContain('DTSTART;VALUE=DATE:20260116');
48     expect(ics).toContain('DTEND;VALUE=DATE:20260117');
49   });
50
51   it('uses \\r\\n line endings and wraps in VCALENDAR', () => {
52     const ics = buildIcs(anime(), schedule, prefs);
53     expect(ics.startsWith('BEGIN:VCALENDAR\r\n')).toBe(true);
54     expect(ics.endsWith('END:VCALENDAR')).toBe(true);
55   });
56
57   it('includes the episode range in the summary', () => {
58     const ics = buildIcs(anime(), schedule, prefs);
59     expect(ics).toContain('SUMMARY:Test Show - Ep 1-2');
60     expect(ics).toContain('SUMMARY:Test Show - Ep 3-4');
61   });
62
63   it('adds a description with real episode titles when given', () => {
64     const titles = new Map([
65       [1, 'A'],
66       [2, 'B'],
67       [3, 'C'],
68       [4, 'D'],
69     ]);
70     const ics = buildIcs(anime(), schedule, prefs, titles);
71     expect(ics).toContain('DESCRIPTION:A\\nB');
72     expect(ics).toContain('DESCRIPTION:C\\nD');
73   });
74
75   it('omits DESCRIPTION when no episode titles are given', () => {
76     const ics = buildIcs(anime(), schedule, prefs);
77     expect(ics).not.toContain('DESCRIPTION');
78   });
79
80   it('escapes commas and semicolons in the title', () => {
81     const withComma = { ...anime(), title: { romaji: 'A, B; C', english: null } };
82     const ics = buildIcs(withComma, schedule, prefs);
83     expect(ics).toContain('SUMMARY:A\\, B\\; C - Ep 1-2');
84   });
85
86   it('neutralizes carriage returns so a title cannot inject ICS lines', () => {
87     const evil = { ...anime(), title: { romaji: 'X\\rATTENDEE:mailto:a@b.c\\r\\nY', english: null } };
88     const ics = buildIcs(evil, schedule, prefs);

```

```

89     expect(ics.split('\r\n').some((line) => line.startsWith('ATTENDEE'))).toBe(false);
90     expect(ics).not.toMatch(/\r(?:\n)/);
91   });
92
93   it('skips days the user does not watch', () => {
94     // 2026-01-15 is a Thursday; weekends only -> Sat 17th, Sun 18th.
95     const ics = buildIcs(anime(), schedule, { ...prefs, watchDays: [0, 6] });
96     expect(ics).toContain('DTSTART;VALUE=DATE:20260117');
97     expect(ics).toContain('DTSTART;VALUE=DATE:20260118');
98     expect(ics).not.toContain('DTSTART;VALUE=DATE:20260115');
99   });
100
101   it('makes timed events lasting the day's minutes when a watch time is set', () => {
102     const ics = buildIcs(anime(), schedule, { ...prefs, watchTime: '23:30' });
103     expect(ics).toContain('DTSTART:20260115T233000');
104     // 48 minutes later crosses midnight.
105     expect(ics).toContain('DTEND:20260116T001800');
106     expect(ics).not.toContain('VALUE=DATE');
107   });
108
109   it('adds a reminder only when a watch time and reminder are both set', () => {
110     const withReminder = buildIcs(anime(), schedule, { ...prefs, watchTime: '20:00', reminderMinutes: 10 });
111     expect(withReminder.match(/BEGIN:VALARM/g)).toHaveLength(2);
112     expect(withReminder).toContain('TRIGGER:-PT10M');
113
114     const allDay = buildIcs(anime(), schedule, { ...prefs, reminderMinutes: 10 });
115     expect(allDay).not.toContain('VALARM');
116   });
117
118   it('adds air-time events for upcoming episodes of an airing show', () => {
119     const airing: Anime = {
120       ...anime(),
121       status: 'RELEASING',
122       upcomingEpisodes: [
123         { episode: 6, airingAt: Date.UTC(2026, 0, 20, 15, 0) / 1000 },
124         { episode: 5, airingAt: Date.UTC(2026, 0, 13, 15, 0) / 1000 },
125       ],
126     };
127     const ics = buildIcs(airing, schedule, prefs, undefined, NOW);
128     expect(ics).toContain('UID:42-air-5@anime-planner');
129     expect(ics).toContain('DTSTART:20260113T150000Z');
130     expect(ics).toContain('DTEND:20260113T152400Z');
131     expect(ics).toContain('SUMMARY:Test Show - Ep 6 airs');
132   });
133
134   it('leaves out air times that are already in the past', () => {
135     const airing: Anime = {
136       ...anime(),
137       upcomingEpisodes: [{ episode: 5, airingAt: Date.UTC(2025, 11, 1) / 1000 }],
138     };
139     expect(buildIcs(airing, schedule, prefs, undefined, NOW)).not.toContain('airs');
140   });
141 });

```

src/lib/route.test.ts

25 lignes

Les tests des adresses de page : construction, lecture, et rejet de tout ce qui n'a pas la bonne forme.

```

1  import { describe, expect, it } from 'vitest';
2  import { animePath, parseAnimePath } from './route';
3
4  describe('animePath / parseAnimePath', () => {
5    it('builds an id + slug path and reads the id back', () => {
6      const path = animePath(108632, 'Re:ZERO -Starting Life in Another World- Season 2');
7      expect(path).toBe('/anime/108632-re-zero-starting-life-in-another-world-season-2');
8      expect(parseAnimePath(path)).toBe(108632);
9    });
10
11    it('drops the slug when the title has no latin characters', () => {
12      expect(animePath(5, 'ワンピース')).toBe('/anime/5');
13    });
14
15    it('accepts a bare id, any slug and a trailing slash', () => {
16      expect(parseAnimePath('/anime/21')).toBe(21);
17      expect(parseAnimePath('/anime/21-an-old-title/')).toBe(21);

```

```

18   });
19
20   it('rejects anything else', () => {
21     for (const bad of ['/', '/anime', '/anime/', '/anime/0', '/anime/abc', '/anime/21/extra', '/animex/21',
22       '/anime/21-Caps']) {
23       expect(parseAnimePath(bad)).toBeNull();
24     }
25   });

```

src/lib/seo.test.ts

61 lignes

Les tests du référencement : titre, adresse canonique et image d'une série, description courte et sans HTML, et échappement qui empêche les données de casser le HTML.

```

1  import { describe, expect, it } from 'vitest';
2  import type { Anime } from '../types';
3  import { HOME_SEO, animeJsonLd, animeSeo, metaTags, renderHeadTags } from './seo';
4
5  function anime(overrides: Partial<Anime> = {}): Anime {
6    return {
7      id: 21,
8      title: { romaji: 'ONE PIECE', english: null },
9      format: 'TV',
10     status: 'RELEASING',
11     episodes: null,
12     duration: 24,
13     seasonYear: 1999,
14     averageScore: 87,
15     coverImage: { large: 'https://s4.anilist.co/file/cover.jpg', color: null },
16     genres: ['Action', 'Adventure'],
17     studios: [],
18     nextAiringEpisode: null,
19     trailer: null,
20     ...overrides,
21   };
22 }
23
24 describe('animeSeo', () => {
25   it('builds a title, canonical URL and image for the show', () => {
26     const seo = animeSeo(anime());
27     expect(seo.title).toBe('ONE PIECE watch schedule | Anime Watch Planner');
28     expect(seo.url).toBe('https://anime-planner.vercel.app/anime/21-one-piece');
29     expect(seo.image).toBe('https://s4.anilist.co/file/cover.jpg');
30     expect(seo.type).toBe('video.tv_show');
31   });
32
33   it('keeps the description short and free of HTML', () => {
34     const seo = animeSeo(anime({ episodes: 12, description: 'A <i>very</i> long story.<br>.repeat(20) }));
35     expect(seo.description.startsWith('Plan ONE PIECE (12 episodes, 24 min each) day by day. A very
36 long')).toBe(true);
37     expect(seo.description.length).toBeLessThanOrEqual(160);
38     expect(seo.description).not.toMatch(/<|>/);
39   });
40
41   describe('metaTags', () => {
42     it('only includes image tags when there is an image', () => {
43       expect(metaTags(HOME_SEO).some((t) => t.key === 'og:image')).toBe(false);
44       expect(metaTags(animeSeo(anime()))).some((t) => t.key === 'og:image')).toBe(true);
45     });
46   });
47
48   describe('renderHeadTags', () => {
49     it('escapes text so data cannot break out of attributes or the JSON-LD script', () => {
50       const evil = anime({ title: { romaji: '<<script>alert(1)</script>', english: null } });
51       const html = renderHeadTags(animeSeo(evil), animeJsonLd(evil));
52       expect(html).not.toContain('<script>alert');
53       expect(html).not.toContain('<>');
54       expect(html.match(/<\/script>g)).toHaveLength(1);
55     });
56
57     it('marks movies as Movie in JSON-LD', () => {
58       expect(animeJsonLd(anime({ format: 'MOVIE' }))[ '@type' ]).toBe('Movie');

```

```

59     expect(animeJsonId(anime())['@type']).toBe('TVSeries');
60   });
61 });

```

src/lib/share.test.ts

71 lignes

Les tests des anciens liens de partage : aller-retour encodage/décodage, caractères spéciaux, et rejet des liens abîmés ou fabriqués.

```

1  import { describe, expect, it } from 'vitest';
2  import type { Anime } from '../types';
3  import { decodeShareParam, encodeShareParam } from './share';
4
5  function anime(id: number): Anime {
6    return {
7      id,
8      title: { romaji: 'Test', english: null },
9      format: 'TV',
10     status: 'FINISHED',
11     episodes: 12,
12     duration: 24,
13     seasonYear: 2020,
14     averageScore: 80,
15     coverImage: { large: null, color: null },
16     genres: [],
17     studios: [],
18     nextAiringEpisode: null,
19     trailer: null,
20   };
21 }
22
23 describe('share', () => {
24   it('round-trips id and config', () => {
25     const encoded = encodeShareParam(anime(154587), { dailyBudget: 90, skipOpEd: true });
26     expect(decodeShareParam(encoded)).toEqual({
27       id: 154587,
28       config: { dailyBudget: 90, skipOpEd: true },
29     });
30   });
31
32   it('produces a URL-safe string (no +, /, =)', () => {
33     const encoded = encodeShareParam(anime(1), { dailyBudget: 90, skipOpEd: false });
34     expect(encoded).toMatch(/^[A-Za-z0-9_-]+$\/);
35   });
36
37   it('round-trips unicode titles safely', () => {
38     const a = { ...anime(1), title: { romaji: '進撃の巨人', english: null } };
39     const encoded = encodeShareParam(a, { dailyBudget: 30, skipOpEd: false });
40     expect(decodeShareParam(encoded)?.id).toBe(1);
41   });
42
43   it('returns null for garbage input', () => {
44     expect(decodeShareParam('not-valid-base64!!!')).toBeNull();
45     expect(decodeShareParam('')).toBeNull();
46   });
47
48   it('returns null when decoded JSON has the wrong shape', () => {
49     const badPayload = btoa(JSON.stringify({ foo: 'bar' }));
50     expect(decodeShareParam(badPayload)).toBeNull();
51   });
52
53   const raw = (obj: unknown) => btoa(JSON.stringify(obj));
54
55   it('drops unknown keys from a crafted payload', () => {
56     const decoded = decodeShareParam(
57       raw({ id: 5, config: { dailyBudget: 60, skipOpEd: false, evil: '<script>' }, extra: 1 })
58     );
59     expect(decoded).toEqual({ id: 5, config: { dailyBudget: 60, skipOpEd: false } });
60   });
61
62   it.each([
63     { id: -1, config: { dailyBudget: 60, skipOpEd: false } },
64     { id: 1.5, config: { dailyBudget: 60, skipOpEd: false } },
65     { id: 1, config: { dailyBudget: 0, skipOpEd: false } },

```

```

66     { id: 1, config: { dailyBudget: 99999, skipOpEd: false } },
67     { id: 1, config: null },
68   ])(`rejects out-of-range payload %j`, (payload) => {
69     expect(decodeShareParam(raw(payload))).toBeNull();
70   });
71 });

```

src/lib/platform.test.ts

21 lignes

Les tests de la détection d'iPhone et d'iPad, y compris l'iPad qui se présente comme un Mac.

```

1  import { describe, expect, it } from 'vitest';
2  import { isIOS } from './platform';
3
4  const nav = (userAgent: string, platform: string, maxTouchPoints = 0) => ({ userAgent, platform, maxTouchPoints });
5
6  describe('isIOS', () => {
7    it('detects iPhone and iPad user agents', () => {
8      expect(isIOS(nav('Mozilla/5.0 (iPhone; CPU iPhone OS 18_0 like Mac OS X)', 'iPhone', 5))).toBe(true);
9      expect(isIOS(nav('Mozilla/5.0 (iPad; CPU OS 16_0 like Mac OS X)', 'iPad', 5))).toBe(true);
10   });
11
12   it('detects iPadOS pretending to be a Mac', () => {
13     expect(isIOS(nav('Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)', 'MacIntel', 5))).toBe(true);
14   });
15
16   it('does not match desktop Macs, Android or missing navigator', () => {
17     expect(isIOS(nav('Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)', 'MacIntel', 0))).toBe(false);
18     expect(isIOS(nav('Mozilla/5.0 (Linux; Android 15; Pixel 9)', 'Linux armv8l', 5))).toBe(false);
19     expect(isIOS(undefined)).toBe(false);
20   });
21 });

```

A.11 La documentation

README.md

62 lignes

La présentation du projet sur GitHub : ce qu'il fait, l'adresse du site, les fonctionnalités, les technologies et les commandes.

```

1  # Anime Watch Planner
2
3  Plan any anime around the time you actually have. Pick a show, say how many
4  minutes a day you can watch, and get a day-by-day schedule with the real
5  episode titles, on the days you choose, ready to add to your calendar.
6
7  **Live:** https://anime-planner.vercel.app
8
9  ![A show's page: the plan settings on the left, the day-by-day schedule on the right](guide/images/list.png)
10
11 > Built as a hobby project with Claude Code (Claude Opus). Not an official
12 > project.
13
14 ## Features
15
16 - Search any anime (Anilist data), or start from what's popular right now
17 - Set your daily time in minutes or in episodes per day, and optionally skip
18   openings and endings (3 minutes per episode)
19 - A day-by-day list with the real episode titles, and a month calendar
20 - Pick your watch days (every day, weekdays, weekends or your own), a watch
21   time, a reminder and a time zone
22 - Export to your calendar (`.ics`), including air dates for shows still airing
23 - Save schedules in your browser, check off what you've watched, keep a
24   streak, and pick up where you left off
25 - Watch order for franchises, official streaming links and trailers
26 - A clean URL for every show, with prerendered pages for link previews and
27   search engines
28 - A light navy-and-orange design with square corners, where each show's page
29   takes on the colors of its poster (darkened when needed to keep text readable)
30

```

```

31 ## Stack
32
33 React 18, TypeScript, Vite, Tailwind CSS 4, anime.js 4 and the Anilist
34 GraphQL API, tested with Vitest. There is no backend: everything runs in the
35 browser, and saved schedules stay in the browser's `localStorage`.
36
37 ## Getting started
38
39 ...
40 npm install
41 npm run dev      # local dev server
42 npm test        # unit tests
43 npm run build    # type-check, build, prerender show pages and the sitemap
44 npm run preview  # serve the production build
45 ...
46
47 ## Project guide
48
49 A complete guide in French and English explains how everything works, with
50 the full source code. Build it with `npm run guide` (needs Chromium); the PDFs
51 land in `guide/dist/`. `CLAUDE.md` holds the architecture notes and design
52 rules.
53
54 ## Deployment
55
56 Deployed on Vercel from `main` (settings in `vercel.json`). Netlify and
57 Cloudflare Pages work too, with `public/_headers` and `public/_redirects`.
58
59 ## Data
60
61 Anime data comes from [Anilist](https://anilist.co). This project is not
62 affiliated with Anilist or any streaming service.

```

CLAUDE.md

433 lignes

Les notes d'architecture et les règles de conception du projet, écrites pour un assistant de programmation (et lisibles par n'importe quel développeur) : chaque choix important y est expliqué, avec ce qu'il ne faut pas changer sans y réfléchir.

```

1 # CLAUDE.md
2
3 Guidance for Claude Code (or any agentic coding assistant) working in this repo.
4
5 ## What this project is
6
7 Anime Watch Planner – a frontend-only web app. A user searches for an anime
8 (or picks one from the popular list), sets a daily time budget, and gets a
9 day-by-day watch schedule with real per-episode titles, optionally trimming
10 OP/ED runtime. Schedules can be saved to the browser's `localStorage` with
11 watch progress and streaks, exported to a calendar (`.ics`), or shared via the
12 show's own URL (`/anime/<id>-<slug>`). No backend, no server-side database, no
13 auth. Everything else runs client-side against the public Anilist GraphQL
14 API.
15
16 **Deliberately not doing:** AI-generated "what happens in this episode"
17 summaries. Neither Anilist nor MAL/Jikan has real per-episode plot synopses
18 at any useful coverage – only episode titles. Inventing plot text for real,
19 already-released episodes and presenting it as fact risks stating wrong
20 details about real media, so the per-day breakdown only ever shows Anilist's
21 real `streamingEpisodes` titles. Don't add a synopsis feature without
22 re-litigating this with the project owner first.
23
24 ## Stack
25
26 - React 18 + TypeScript, built with Vite
27 - Tailwind CSS for styling
28 - anime.js v4 for all motion (see "Motion" below)
29 - No state management library – `useState` in `App.tsx` is the single
30   source of truth, passed down as props. Don't introduce Redux/Zustand/etc.
31   unless the state graph actually outgrows this.
32 - No router library. The app is a two-step flow (`search` / `configure`)
33   modeled as a string literal in `App.tsx`, and each show also has a real
34   URL, `/anime/<id>-<slug>` (for SEO and sharing). That's a small
35   hand-rolled layer: `lib/route.ts` builds/parses the path (the slug is

```

```

36 cosmetic; lookups use the id), and `App.tsx` pushes it on navigation and
37 runs `resolveLocation` on load and on `popstate`. Search results aren't in
38 the URL. Don't add `react-router` unless more page types appear.
39
40 ## Commands
41
42 ...
43 npm install
44 npm run dev      # local dev server
45 npm run build    # tsc -b && vite build && tsx scripts/prerender.ts
46 npm run preview
47 npm test        # vitest run
48 ...
49
50 Tests use Vitest and live next to the code they cover
51 (`src/lib/*.test.ts`). The environment is plain Node, not jsdom -
52 `storage.test.ts` stubs `localStorage` with an in-memory Map. Keep new tests on the pure `lib/` code.
53
54 ## Architecture
55
56 ...
57 src/
58   api/anilist.ts      AniList GraphQL client (searchAnime, getTrendingAnime,
59                       getPopularAiring, getAnimeById)
60   lib/schedule.ts     Pure calculation engine - no React, no fetch
61   lib/storage.ts      localStorage wrapper for saved schedules + progress
62   lib/progress.ts     Pure progress/streak math (resume day, % done)
63   lib/calendar.ts     Pure date layer: Day N -> real date, month grids, air dates
64   lib/ics.ts          Pure .ics (calendar) text builder
65   lib/platform.ts     Read-only device check (isIOS)
66   lib/download.ts     Isolated side effect: triggers a browser file download
67   lib/share.ts        Decode for old `?share=` links (still honored)
68   lib/route.ts        Pure /anime/<id>-<slug> path build/parse
69   lib/seo.ts          Pure titles, meta/OG tags, JSON-LD (app + prerender)
70   lib/format.ts       Pure display helpers (titles, durations)
71   lib/theme.ts        Pure color math for the per-show theme (contrast-safe accent)
72   motion/             anime.js presets + React hooks (DOM side effects, not in lib/)
73   components/         Presentational components, one per UX step
74   components/ErrorBoundary.tsx Fallback page for render errors, wraps <App /> in main.tsx
75   types.ts            Shared domain types
76   App.tsx             State machine wiring the steps together
77   ...
78
79 Keep this separation. `lib/schedule.ts` in particular must stay pure
80 (inputs in, plain data out) - it's the part most worth unit testing, and
81 that only works if it doesn't reach into `fetch`, `window`, or React state.
82
83 ## The calculation engine (`src/lib/schedule.ts`)
84
85 This is the core logic and the most important file to get right:
86
87 - `getNetEpisodeRuntime(anime, config)` - `duration - (skipOpEd ? 3 : 0)`,
88   but the 3-minute trim only applies when `format !== 'MOVIE'` **and**
89   `duration >= 15`. Don't apply it to shorts or movies.
90 - `getWatchableEpisodes(anime)` - every episode, `1..N`. Movies always
91   return `[1]`. When AniList has no published total (`episodes: null`,
92   typically still-airing), falls back to "episodes aired so far"
93   (`nextAiringEpisode.episode - 1`) instead of giving up - an ongoing show
94   like One Piece (1000+ episodes, no final count on AniList) still gets a
95   real, usable schedule rather than a dead end. Returns `{ episodes,
96   isFallback }`, or `null` only when there's truly nothing to schedule (no
97   total, not airing, or airing but nothing has aired yet). `isFallback`
98   flows through to `Schedule.isPartial`, which `ScheduleView` shows as a
99   "still airing" callout.
100 - `generateSchedule(anime, config)` - buckets watchable episodes into days
101   via `floor(dailyBudget / netRuntime)` episodes per day, greedily. This
102   wastes leftover minutes each day (e.g. 90-min budget, 24-min episodes ->
103   3 episodes/day, 18 min unused every day). That's a known simplification,
104   not a bug - see "Known gaps" below before optimizing it.
105   Returns a `ScheduleResult` union: `{ ok: false, error }` for a
106   non-positive budget or a missing duration/episode count (e.g. still
107   airing). If one episode exceeds the budget it schedules 1 per day and
108   sets `overBudget` rather than returning nothing.
109 - "By episodes" (`ConfigForm.tsx`) is an input convenience, not a second
110   engine mode: N episodes/day is stored as `dailyBudget = N * netRuntime`
111   (an exact multiple, so `floor` lands on N). `ConfigureStep` passes the
112   runtime with and without the OP/ED trim, so toggling the trim keeps the

```

```

113 episode count. The mode itself isn't saved; saved data only ever has
114 minutes.
115
116 If you change the math here, update the docstring-style comments in the
117 file itself, not just this doc – they're the ones a future agent will
118 actually see while editing.
119
120 ## Calendar (`src/lib/calendar.ts`, `src/lib/ics.ts`)
121
122 - `generateSchedule` only splits episodes into numbered days. Real dates
123 come from `CalendarPrefs` (start date, watch weekdays, optional watch
124 time and reminder) through `scheduleDates`: day 1 is the first watch day
125 on or after the start date, each later day the next watch day. Keep the
126 two layers separate; don't put dates into `schedule.ts`.
127 - Dates are `yyyy-mm-dd` strings and day arithmetic runs in UTC, so a
128 daylight-saving change can't shift a day. Only `localIso` and
129 `formatAiring` use a time zone (turning real instants, like "now" or an
130 air time, into dates and labels).
131 - `CalendarPrefs.timezone` (IANA name, or `null` = the device's zone) picks
132 that zone: which day is "today" and when new episodes air, everywhere
133 those show. Pass it to `todayIso`, `upcomingAirings` and `formatAiring`
134 at every call site (saved cards use their own record's zone). It's
135 display/planning only on purpose: the `ics` watch events stay floating
136 (device-local when imported), and air-time events are UTC instants,
137 already right in any zone.
138 - Air dates for shows still airing come from the detail-only
139 `upcomingEpisodes` (AniList `airingSchedule`), falling back to
140 `nextAiringEpisode` (`upcomingAirings`). They show in the header, the
141 calendar, the "still airing" callout, list rows ("Not out yet") and the
142 saved cards.
143 - `ics` export: all-day events without a watch time; with one, timed
144 "floating" local events (no time zone) lasting that day's minutes, plus a
145 `VALARM` if a reminder is set. Upcoming air times are added as separate
146 UTC events. UIDs are stable (`<id>-<day>`, `<id>-air-<ep>`), so
147 re-importing updates events instead of duplicating them.
148 - The day panel's episode titles collapse under an "Episode titles (N)"
149 header (`EpisodeTitles` in `CalendarView.tsx`, collapsed by default when
150 N > 3). It only reveals the same real titles, never descriptions (see
151 "Deliberately not doing" above).
152 - On iOS (`isIOS`), "Add to calendar" opens the file in the tab
153 (`openTextFile`) so Safari can offer its "Add to Calendar" sheet; other
154 platforms download it. That iOS path can't be tested without a real
155 iPhone.
156
157 ## Saved schedules (`src/lib/storage.ts`)
158
159 - Key `anime-planner:schedules:v1` holds a `SavedSchedule[]` (see
160 `types.ts`): an `Anime` snapshot, the `ScheduleConfig`, and a `watched`
161 list of `{ episode, completedAt }`.
162 - Progress is per episode, not per day. Days are re-bucketed whenever
163 the daily budget changes, so a day number doesn't identify the same
164 episodes over time. A day shows as watched when all its episodes are
165 (`isDayWatched` in `lib/progress.ts`); checking a day toggles its
166 episodes (`toggleEpisodesWatched`).
167 - Old records stored `{ day, completedAt }`. `getSavedSchedules` migrates
168 them on read by re-running the record's own saved anime + config through
169 `generateSchedule` to recover each day's episodes. Records from before
170 `watched` existed default to `[]`. Prefer this migrate-on-read pattern
171 over bumping the storage key.
172 - One entry per anime id: re-saving replaces the config and keeps
173 progress (and keeps calendar prefs when none are passed).
174 - `calendar` (`CalendarPrefs`) is validated field by field on read; bad or
175 missing fields fall back to defaults. Records from before it existed
176 start on the local date they were saved.
177 - Every read/write is in try/catch and entries are validated on read, so
178 disabled/full storage or corrupt data degrades to "nothing saved" instead
179 of crashing. Keep it that way.
180 - It's a snapshot, not a cache: opening a saved schedule renders the
181 snapshot instantly, then `App.tsx` refetches the anime by id in the
182 background. "Copy link" copies the show's clean URL, which carries no
183 settings: the recipient gets fresh data and default settings (a
184 deliberate trade for a readable, indexable URL). Old `?share=` links
185 still open with their encoded config and are rewritten to the clean URL.
186 - Checking off a day implicitly saves the schedule first if it wasn't
187 already saved, since progress has nowhere else to live.
188
189 ## AniList API notes (`src/api/anilist.ts`)

```

- Endpoint: ``https://graphql.anilist.co``, single POST endpoint, standard GraphQL over HTTP. No API key needed for read-only queries like these.
- Anilist rate-limits aggressively (historically ~90 req/min, enforced via ``Retry-After`` headers on 429s). A 429 surfaces as an ``AniListError`` with ``retryAfter``, shown to the user, but there's no automatic retry or backoff. Search only fires on submit; add debouncing before adding request volume, not after.
- The homepage list ("Popular right now") is fetched once on load. A failure just hides the section. It's filtered with ``isPlannable`` in ``App.tsx`` (does ``generateSchedule`` succeed?), so ongoing shows stay in via the aired-so-far fallback and only truly unplannable ones are dropped. There's no trending section on the page any more (removed on purpose); ``getTrendingAnime`` is kept for the build-time prerender script.
- "Popular right now" (``getPopularAiring``) is currently airing shows by popularity. It is deliberately not "this season": at the start of a season nearly every show is unaired, and the list came out empty.
- ``MediaFields`` is shared by the search, trending, popular and by-id queries so the shapes can't drift. Keep it to lightweight fields since it's used by 12-30-item list queries. Detail-only fields (``description``, ``streamingEpisodes``, ``airingSchedule``, ``relations``, ``externalLinks``) are fetched only in ``getAnimeById`` and are ``undefined`` on list results; ``types.ts`` marks them optional for this reason.
- ``getAnimeById`` normalizes Anilist's raw shapes so components never see them:
 - ``studios { nodes { name } }`` becomes ``{ name }[]``.
 - ``externalLinks`` becomes ``watchLinks``: ****official STREAMING links only****, one per site. This is the only "where to watch" source, on purpose. Don't add unofficial/torrent sources (an extension index of that kind was already declined for this project).
 - ``site`` is free text, so ``StreamingIcon`` matches it loosely (lowercased, letters/digits only) against a few embedded Simple Icons marks (CC0: Crunchyroll, Netflix, YouTube, Bilibili, Tubi, Max) and otherwise draws a letter tile in the link's brand color. Hulu, Prime Video, Disney+, HIDIVE and Funimation aren't in Simple Icons (removed at the brands' request); don't hand-draw their logos.
 - ``relations`` becomes an ordered watch-order list (``buildWatchOrder``): PREQUEL/SEQUEL edges fetched 2 levels deep in one request, deduped, sorted by ``seasonYear``. That's a release-year heuristic, not a true topological sort, and can be wrong for a show released out of story order. A franchise with no separate season entries (One Piece) gets no list, which is correct.
- ``description(asHtml: false)`` is not really plain text: it keeps ``<br>`` tags, and some shows also carry ``<i>`/`<b>`` and entities (Death Note, One Piece, Jujutsu Kaisen). The synopsis goes through ``synopsisText`` (``lib/format.ts``: ``<br>`` to newlines, other tags dropped, entities decoded); SEO text goes through ``plainText``, which builds on it.
- ****Episode titles**** (``streamingEpisodes``) must be looked up by the number parsed from each title (``episodeTitleMap`` in ``lib/format.ts``), never by array index. Anilist's list can be partial and out of order: One Piece has 69 entries, reversed, starting at episode 130. Titles with no parseable number are dropped. A missing title beats a wrong one.

Theming and UI (``index.css``, ``App.tsx``)

- Light theme in two brand colors from the owner's mood board: navy ``--navy` #133055` (Pantone Classic Blue) and orange ``--orange` #BF4A1B` (Pantone Adobe), on cream ``--cream` #F3EDE4`. ``--blue` #0F4C81` (Pantone's own sRGB for Classic Blue) marks air dates only. These live on ``:root`` so the body (``index.html``) and ``ErrorBoundary`` can use them.
- A show page takes on its poster's colors, like the old dark theme did. ``App.tsx`` runs the open show's ``coverImage.color`` through ``posterTheme`` (``lib/theme.ts``) and sets two inputs on the root ``:theme-root`` element: ``--poster-tints`` ``--bg``, ``--surface``, ``--surface-2`` and the glow at the top, and ``--poster-accent`` replaces the orange ``--accent`` (buttons, toggles, progress bars, highlights). Both are unset on the homepage and for gray covers, which keep navy and orange.
- ``posterTheme`` exists because CSS can't check contrast. The accent is the cover color, darkened in OKLCH (same hue) when white text on it (``--on-accent``) would fall under 4.5:1. The tint is lightened when the cover is dark, so the page takes the hue without going gray. ``--accent-text`` is the accent darkened for text on cream. The saved-card progress bars use the same accent.
- ``lib/theme.test.ts`` mirrors the ``:theme-root`` percentages in ``index.css`` and checks every text/background pair over 4096 cover colors. Change the two together.

```

267 - Components use these tokens (`bg(--surface)`, `ring(--line)`,
268 `text(--accent-text)`, ...) and never hard-code slate surfaces. Don't
269 thread a color prop through components; the theme is set once, at the
270 top. Text uses `--ink` (headings, body), `--ink-2` (secondary) and
271 `--ink-3` (hints), mixed from navy against the fixed cream. The exception
272 is text on cover images (the score badge), which stays white on a navy
273 chip.
274 - Every corner is square: no `rounded-*` classes anywhere (Tailwind's
275 preflight already zeroes `border-radius`). Native checkboxes draw rounded
276 corners regardless, so `index.css` redraws them square (`appearance:
277 none`, clip-path tick, no image, so no CSP change). Icons in
278 `components/icons.tsx` use square caps and miter joins.
279 - Layout is responsive, mobile first. `CONTAINER` in `App.tsx` is the
280 shared width + gutters for header, main and footer. The poster grid is
281 2 / 3 / 6 columns (the 18-item popular list fills whole rows at each).
282 A Grid/List toggle (`SearchStep.tsx`, local state, not saved) swaps it
283 for `AnimeListRow` rows, one column on phones and two from `lg`.
284 The configure page stacks below `lg` and gets its sticky sidebar at `lg`
285 and up. Single-column grids need `grid-cols-[minmax(0,1fr)]`, or
286 truncated text widens them past the screen on phones. The root uses
287 `overflow-x-clip`, not `overflow-hidden`, because `overflow-hidden` makes
288 it a scroll container and breaks `sticky` (header and sidebar).
289 - Hover-only controls (the saved-schedule delete button) hide behind
290 `pointer-fine:` so they stay visible on touch screens.
291 - No emoji or Unicode symbol glyphs in the UI (project owner's call). Use
292 plain text, or the small stroke icons in `components/icons.tsx`. All
293 section headings use `SectionHeading`.
294 - The header title's glitch is `GlitchText` (`components/Motion.tsx`): two
295 real `aria-hidden` copies of the text, played by an anime.js timeline on
296 hover/keyboard focus of the header button (JS can't reach
297 `::before`/`::after`).
298
299 ## Motion (`src/motion/`, `components/Motion.tsx`)
300
301 - anime.js v4 drives every animation; there are no CSS keyframes left.
302 Hover color fades stay plain CSS `transition-*`.
303 - Animate only `opacity`, transforms (`x`/`y`/`scale`/`rotate`), `clip-path`,
304 widths and text. Never colors: those come from the theme tokens, so the
305 per-show theme keeps working with no extra code.
306 - `presets.ts`: durations, eases, `enter()` (fade up/stagger), `pop()`,
307 `reduceMotion()`, and `installPressFeedback()` (one delegated listener in
308 `App.tsx` that springs every button/link on press).
309 - `hooks.ts`: `useEnter`, `useSwap`, `useSlide`, `useCountUp`, `useWidth`,
310 `useIndicator`, `useCollapse` (height 0 <-> content, left at `auto` once
311 open). They animate in `useLayoutEffect` (starting styles land
312 before the first paint, so nothing flashes) and revert on cleanup.
313 Finished animations call `cleanInlineStyles`, so no inline
314 opacity/transform is left behind (the List/Calendar pill is the one
315 deliberate exception: its transform is its position). A leftover transform
316 would create a stacking context; keep that true for new animations.
317 - Elements whose text or width a hook owns (`CountUp`, the progress bar)
318 must not also get that text/width from JSX, or React and anime.js fight.
319 - CSS `transition` includes `transform`/`opacity`, which smears JS
320 animation. `index.css` removes `transform` from transitions on
321 buttons/links (for the press spring). Don't put `transition` or
322 `transition-all` on an element that anime.js moves; use
323 `transition-colors`/`transition-shadow`.
324 - Every animation is skipped under `prefers-reduced-motion`: hooks jump to
325 the end state, `pop`/`press`/`glitch`/`loops` don't run. Keep durations under
326 ~600 ms ("polished, never in the way", the owner's call).
327 - Deleting a saved card animates out first, then calls `onDelete`, so
328 state stays in `App.tsx`.
329
330 ## Security
331
332 No backend, accounts or secrets, so the attack surface is untrusted data
333 reaching the browser: AniList responses (user-submitted, moderated), the
334 `?share=` link (anyone can craft one), localStorage, and the `.ics` export.
335 Rules that keep it safe, so keep them:
336
337 - Never render API text as HTML: no `dangerouslySetInnerHTML`, `innerHTML`
338 or `eval`. React escapes text; the synopsis renders as text.
339 - Every external URL from data goes through `safeExternalUrl`
340 (`lib/format.ts`), which allows `https:` only. React 18 does not
341 block `javascript:` in `href`.
342 - Colors from AniList go through `safeHexColor` before reaching CSS
343 (`--poster`, brand borders).

```

- Trailer ids are validated (``youtubeEmbedUrl``). The iframe is sandboxed, uses youtube-nocookie.com, and only mounts after a click.
- ``decodeShareParam`` rebuilds the payload from known fields with range checks and never passes the parsed object through.
- ``ics`` text escaping handles ``\r`` as well as ``\n``, so titles can't inject calendar fields.
- Production security headers (CSP, ``X-Frame-Options``, ``nosniff``, ``Referrer-Policy``, ``Permissions-Policy``) live in **both** ``public/_headers`` (Netlify / Cloudflare Pages) and ``vercel.json``. Keep the two in sync. The CSP allows only ``graphql.anilist.co`` (fetch), ``s4.anilist.co`` (images) and ``www.youtube-nocookie.com`` (frames). A new external host needs a CSP entry, or it will silently fail in production but work in dev (the dev server doesn't apply these headers). Don't move the CSP into a ``<meta>`` tag: that would break Vite's dev server.

SEO and static pages

- ``index.html`` holds the default tags between ``<!--seo-start-->`` and ``<!--seo-end-->``. Everything in that block comes from ``lib/seo.ts`` (``HOME_SEO``, ``animeSeo``, ``metaTags``, ``animeJsonLd``, ``renderHeadTags``). Edit tags there, not in the HTML. The same functions drive:
 - ``App.tsx``: an effect keyed on ``[step, selected]`` updates the title, meta tags, canonical link and JSON-LD while browsing.
 - ``scripts/prerender.ts`` (after ``vite build``): fetches trending + popular (two requests), and writes ``dist/anime/<id>-<slug>.html`` for each plannable show (the app shell with that show's tags) plus ``dist/sitemap.xml``. It also rewrites ``dist/index.html``'s block from ``HOME_SEO``. If Anilist is down it warns and still succeeds.
- Flat ``*.html`` files, not ``<path>/index.html``: Vercel (``cleanUrls``), Netlify and Cloudflare Pages all serve them at the extension-less URL.
- Show URLs that weren't prerendered fall back to the app shell through a rewrite in **both** ``vercel.json`` and ``public/_redirects``. A real file always wins over the rewrite. Keep the two in sync, like the headers. On Vercel the destination must be ``/``, not ``/index.html``: with ``cleanUrls`` on, ``/index.html`` doesn't resolve as a rewrite target and every non-prerendered show 404s.
- Link-preview images use ``coverImage.extraLarge`` (through ``safeExternalUrl``). ``large`` is Anilist's medium size and looks blurry. The X/Twitter card is ``summary`` because covers are portrait.
- JSON-LD has no ``aggregateRating`` on purpose: Anilist's list data has no rating count, and a made-up one is spammy structured data.

Declined integrations (checked, not just skipped)

- **YouTube video lists and a music player**: inline search results would need a YouTube API key capped at ~100 searches/day, shared by every visitor and exposed in the browser bundle. Trailers are fine: Anilist provides their ids directly, with no key needed.
- **Anime conventions/events**: no free public API exists (checked live). It would mean a stale hand-curated list or scraping a listings site, which is the same ToS-risk category as the AnimeFillerList scrape declined for filler data. Flag the tradeoff again before adding it.

Known gaps / intentionally deferred

- Saved schedules live in one browser only – no sync across devices.
- Only the ~40-50 shows trending/popular at build time get prerendered pages (real per-show tags for link previews and non-JS crawlers). Every other show URL works through the rewrite and gets its tags from the app once JavaScript runs, which Google handles but Discord/Slack previews don't. The list refreshes on each deploy.
- Prerendered descriptions use list-view fields only (no synopsis), to keep the build at two Anilist requests.
- ``scripts/prerender.ts`` runs through ``tsx`` without type-checking (no ``@types/node`` in the project). It imports the typed ``src/`` modules, so keep its own logic thin.
- Tests cover only ``lib/``; no component tests.
- One error boundary around the whole app (``ErrorBoundary.tsx``): a render error shows a "Something went wrong" page with Reload, not a blank page. There are no finer-grained boundaries per section.
- Bin-packing is greedy/naive (see above), not budget-optimal.
- No pagination on search results (Anilist ``perPage: 12``, page 1 only).
- ``ScheduleView``'s day list paginates client-side (30 at a time, always extended to include the resume day) for long schedules like One Piece's 390+ days. It's only a render-size guard: all days are computed, just not all rendered at once.
- The aired-so-far fallback for ongoing shows never shrinks: if a show's

actual total episode count later drops from what "aired so far" implied (rare, but Anilist data can be corrected), an old saved schedule won't self-adjust until reopened and refetched.

Conventions

- Components are presentational and take props; side effects (fetching, state) live in `App.tsx`. Keep it that way rather than letting individual components call the Anilist API directly.
- Prefer editing `types.ts` when a shape needs to change, rather than inlining a new ad hoc type in a component.
- This file and the code comments should stay in sync with each other – if you find them disagreeing, fix both in the same change.

Annexe B. Glossaire

Accessibilité. Le fait de rendre un site utilisable par tous, y compris au clavier, avec un lecteur d'écran ou avec un texte agrandi.

Adresse canonique. L'adresse officielle d'une page, donnée aux moteurs de recherche pour que ses variantes ne soient pas comptées comme des doublons.

Algorithme glouton. Une méthode qui fait le meilleur choix immédiat à chaque étape sans regarder plus loin, comme remplir chaque journée autant que possible.

API. Un moyen pour un programme de demander des informations ou des actions à un autre, comme on commande sur un menu. AniList en propose une.

Application monopage (SPA). Un site avec une seule vraie page HTML, dont le contenu est changé par JavaScript pendant la navigation.

Bac à sable (sandbox). Un espace restreint dans lequel quelque chose (ici, le cadre de la bande-annonce) fonctionne avec des pouvoirs limités.

Cache. Une copie gardée à portée de main (par votre navigateur, par exemple) pour éviter de télécharger à nouveau la même chose.

CDN (réseau de diffusion de contenu). Un réseau de serveurs répartis dans le monde qui gardent des copies d'un site, pour qu'il se charge vite de partout.

Composant. Une brique réutilisable d'une interface React : une fonction qui reçoit des props et renvoie ce qu'il faut afficher.

Construction (build). La transformation du code source en quelques fichiers optimisés, prêts à être mis en ligne.

Contraste (rapport de). L'écart de luminosité entre deux couleurs, de 1:1 (la même couleur) à 21:1 (du noir sur du blanc). Les règles d'accessibilité (WCAG) demandent au moins 4,5:1 pour un texte normal.

CSP (Content Security Policy). Un en-tête de sécurité qui liste ce qu'une page a le droit de charger et de contacter.

CSS. Le langage qui décrit l'apparence d'une page web.

Échappement. Écrire des caractères spéciaux sous une forme sûre, pour qu'ils soient lus comme du texte et non comme des instructions.

Effet de bord. Tout ce qu'un morceau de code fait au monde extérieur : dessiner à l'écran, télécharger, interroger un service, lire l'horloge.

En-tête (HTTP). Une information supplémentaire envoyée avec une page web, comme des instructions de sécurité.

État (state). Les informations dont un composant ou une application se souvient et qui peuvent changer avec le temps.

Expression régulière (regex). Un motif compact pour trouver ou vérifier du texte, comme « une date du type 2026-10-07 ».

Fonction pure. Une fonction qui ne fait que calculer : mêmes entrées, même réponse, et aucun effet sur le monde extérieur.

Fragment (GraphQL). Une liste de champs nommée et réutilisable, partagée par plusieurs requêtes.

Frontend / backend. La partie d'un site qui fonctionne dans votre navigateur / la partie qui fonctionne sur un serveur.

Fuseau horaire IANA. Un nom standard de fuseau horaire, comme `Europe/Paris` ou `America/Guadeloupe`.

Git, GitHub. Git enregistre l'historique des modifications d'un projet ; GitHub conserve cet historique en ligne.

GraphQL. Un langage de requête dans lequel on décrit exactement les données que l'on veut recevoir.

Hook (React). Une fonction qui commence par `use` et donne une capacité à un composant : une mémoire (`useState`), des résultats mémorisés (`useMemo`), des actions après l'affichage (`useEffect`, `useLayoutEffect`).

HTML. Le langage qui décrit la structure d'une page web.

HTTPS. La version sécurisée et chiffrée du protocole qui sert à charger les pages web.

iCalendar (.ics). Un format texte standard pour les événements d'agenda, compris par toutes les applications d'agenda.

Intl. La boîte à outils de dates, d'heures, de nombres et de langues intégrée à JavaScript.

JavaScript. Le langage de programmation des navigateurs web.

Jeton (design token). Une couleur ou une taille nommée, comme `--surface`, que les composants utilisent à la place de valeurs fixes, pour que toute l'apparence puisse changer en un seul endroit.

JSON. Un format de texte simple pour les données, lisible par les humains et par les programmes.

JSON-LD. Des données structurées écrites en JSON qui décrivent le contenu d'une page aux moteurs de recherche.

Limite de requêtes (rate limit). Le nombre maximal de requêtes qu'un service accepte en un temps donné.

localStorage. Un petit espace de stockage qu'un navigateur accorde à chaque site, conservé sur votre appareil.

Markdown. Un format de texte simple pour écrire des documents avec titres, listes et liens ; les chapitres de ce guide sont écrits ainsi.

Mise en ligne (déploiement). La publication d'une nouvelle version d'un site.

npm, paquet. npm télécharge et gère les paquets : du code réutilisable écrit par d'autres.

OKLCH. Une façon de décrire une couleur par sa clarté, son intensité (chroma) et sa teinte, conçue pour que des écarts égaux paraissent égaux à l'œil. Ne baisser que la clarté assombrit une couleur sans changer sa teinte.

Open Graph. Des balises qui contrôlent le titre, le texte et l'image affichés quand un lien est partagé sur les réseaux sociaux.

Prérendu. La fabrication à l'avance de pages toutes prêtes, pour que les services qui n'exécutent pas JavaScript voient quand même le bon contenu.

Props. Les informations qu'un composant React reçoit de son parent.

React. Une bibliothèque pour construire des interfaces à partir de composants.

Réécriture (rewrite). Une règle du serveur qui répond à une adresse avec le contenu d'une autre page, sans changer l'adresse affichée.

Référencement (SEO). Le fait de rendre les pages faciles à comprendre et à afficher pour les moteurs de recherche.

Slug. La partie lisible d'une adresse, fabriquée à partir d'un titre, comme `frieren-beyond-journey-s-end`.

Tailwind CSS. Une façon de mettre en forme avec de petits noms de classes tout prêts, écrits dans le HTML.

Test (test unitaire). Un petit programme qui vérifie qu'un morceau de code donne la réponse attendue.

TypeScript, type. JavaScript avec des types : des étiquettes qui disent quel genre de valeur contient chaque chose, vérifiées avant que le code ne tourne.

UTC. L'heure de référence mondiale, sans changement d'heure été/hiver.

Variable CSS. Une valeur nommée en CSS, comme `--accent`, que d'autres styles réutilisent. La changer à un seul endroit change tout ce qui l'utilise.

Vercel. Le service qui héberge le site et le reconstruit automatiquement depuis GitHub.

Vite. L'outil qui fait tourner le site pendant le développement et le construit pour la production.

WCAG (Web Content Accessibility Guidelines). Les règles internationales qui rendent les sites utilisables par tous, dont le contraste minimal entre un texte et son fond.

XSS (injection de script). Une attaque dans laquelle un texte venu de l'extérieur est traité comme du code et s'exécute dans le navigateur d'un visiteur.