

The logo consists of the letters 'AP' in white, bold, sans-serif font, centered within a solid orange square.

Anime Watch Planner

The complete project guide: how it works, why it is built this way, and all of its source code.

PERSONAL PROJECT MADE WITH CLAUDE OPUS (AI). THIS IS NOT AN OFFICIAL PROJECT, JUST A HOBBY. MY REAL PROJECTS, MADE WITHOUT AI, WILL BE SHARED SEPARATELY.

Live app anime-planner.vercel.app

Version of **October 8, 2026**

Mitchel Prince

Contents

1. What the app does

- About this guide
- The project in numbers
- What you can do with it
- What makes it different

2. The basics, explained simply

- What a website is
- Frontend and backend
- React: building a page from bricks
- TypeScript: JavaScript with safety checks
- APIs, GraphQL and JSON
- The tools around the code
- Git, GitHub and Vercel

3. Using the app, step by step

- Step 1: find a show
- Step 2: the show's page
- Step 3: set your plan
- Step 4: choose when you watch
- Step 5: read your schedule
- Step 6: save it and track your progress
- Step 7: export and share
- The other panels
- On a phone

4. How the project is organized

- The folder tree
- What each folder is for
- The golden rule: keep the logic pure
- The dependencies

5. How it all fits together

- The big picture
- Data flows down, events flow up
- The state that App keeps
- Following one action: you click a poster
- Another action: you change your daily time
- Opening a link directly

6. The scheduling engine

- Step 1: how long is one episode?
(`getNetEpisodeRuntime`)
- Step 2: which episodes? (`getWatchableEpisodes`)
- Step 3: split into days (`generateSchedule`)
- The code, line by line
- Why "greedy" and not optimal
- "By episodes" is not a second engine

7. Dates, the calendar and the .ics export

- Two separate layers
- From day numbers to dates (`scheduleDates`)
- Why dates are text, calculated in UTC
- The month grid (`monthGrid`)
- Shows still airing
- The .ics file
- On iPhone and iPad

8. Saving your schedules and progress

- Where plans are stored
- One plan per show
- Progress is per episode, not per day
- Reading defensively
- Upgrading old saves
- The progress numbers (`progress.ts`)

9. Talking to AniList

- What AniList provides
- The four questions the app asks
- Light lists, rich details
- Sending a question (`request`)
- Tidying up the answers
- The episode titles trap
- The description trap
- "Popular right now", not "this season"

10. The interface: theme, components, animation and accessibility

- The component tree
- Two brand colors, and each show's own
- Layout and Tailwind
- Animation
- Accessibility
- When something goes wrong

11. Addresses, search engines and link previews

- Every show has its own address
- Changing pages without loading a new page
- The problem with direct links, and the fix
- What search engines and link previews see
- Prerendering: pages ready in advance
- Old share links

12. Security

- What could go wrong?
- The rules
- Security headers
- Dependencies

13. Tests

- What a test is
- What is tested
- How the tests are set up
- Running them
- What isn't tested automatically

14. From code to website: build and deployment

- Building the site (npm run build)
- Putting it online with Vercel
- How this guide is built

15. Known limits and deliberate choices

- Deliberately not done
- Known limits
- What changed in this version (October 2026)

16. Install and run it yourself

- What you need
- Get the code and start it
- The other commands
- Publishing your own copy
- Where to start reading the code
- Ideas for going further

Appendix A. The complete source code

A.1 Project configuration

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

A.2 The page and public files

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

A.3 Shared types

```
src/types.ts
```

A.4 Pure logic (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 Talking to AniList

```
src/api/anilist.ts
```

A.6 Animation

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

A.7 Interface components

```
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 The application

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

A.9 The build script

```
scripts/prerender.ts
```

A.10 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 Documentation

```
README.md
CLAUDE.md
```

Appendix B. Glossary

1. What the app does

Anime Watch Planner is a website that answers one simple question:

"If I watch this anime for so many minutes a day, when will I finish, and what do I watch each day?"

You pick a show, say how much time you have, and the site gives you a day-by-day plan: which episodes to watch on which date, with their real titles, how long each day takes, and the day you will finish. You can save the plan, tick off what you have watched, and add the whole plan to your phone's or computer's calendar.

The site is online at **anime-planner.vercel.app**. It is free, it needs no account, and everything you save stays in your own browser.

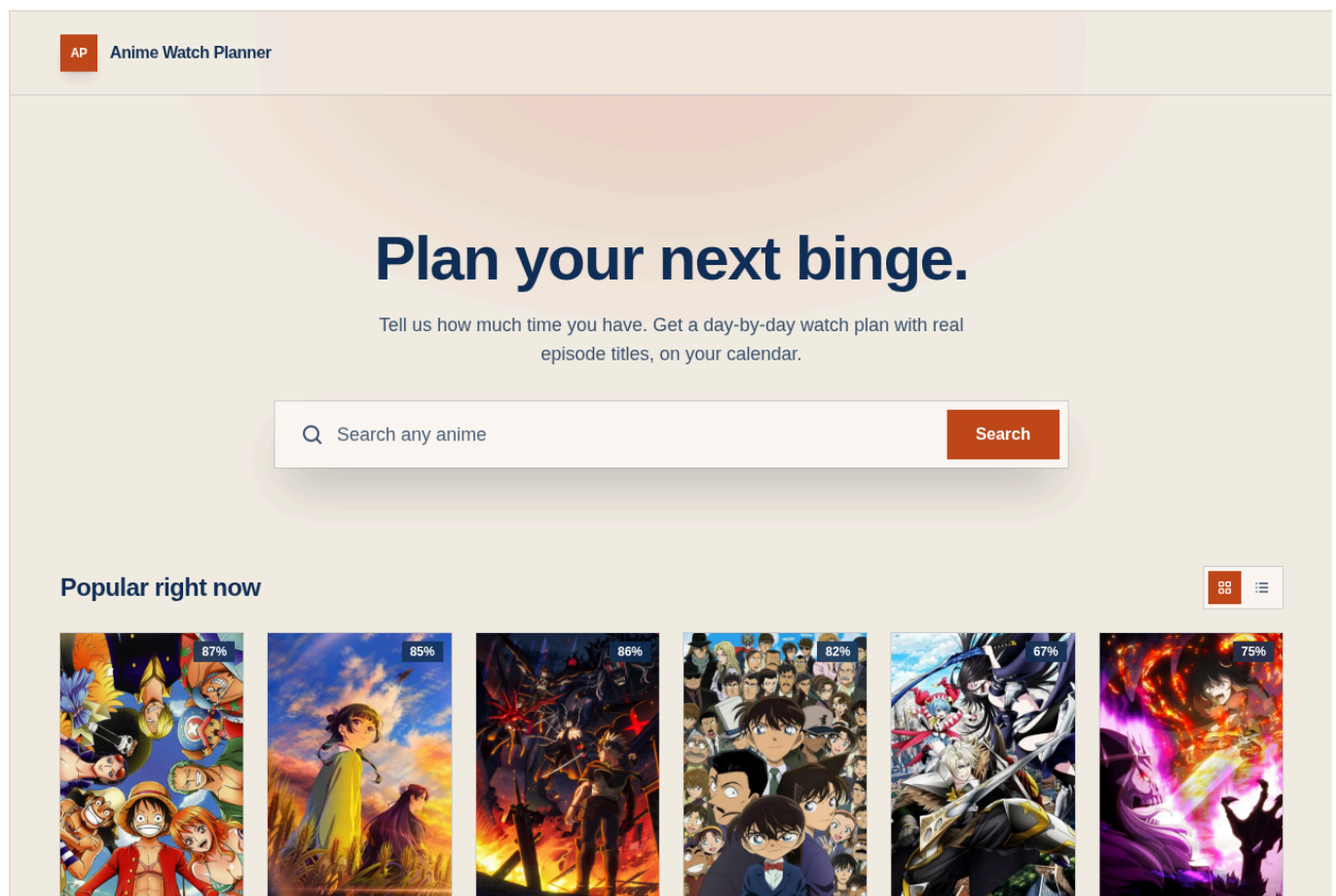


Figure 1. The homepage: a search field and the shows that are popular right now.

About this guide

This guide explains the whole project, from what you see on screen down to every line of code. It is written so that **anyone can follow it**, including people who have never programmed.

- **Chapters 1 to 5** need no programming knowledge. They explain what the app does, the basic ideas behind any website, how to use the app, and how the project is organized.
- **Chapters 6 to 14** go inside the machine: the calculations, dates, saving, the connection to the anime database, the interface, search engines, security, tests and putting the site online. Every technical word is explained the first time it appears, and the **glossary** (Appendix B) collects them all.

- **Chapters 15 and 16** cover the project's limits and deliberate choices, and how to install and run it yourself.
- **Appendix A** contains **the complete source code**: every file of the project, each introduced by a short paragraph and printed with line numbers.

Important: this is a personal hobby project, made with the help of **Claude Opus**, an artificial intelligence (AI) from Anthropic, used through the Claude Code tool. It is not an official or commercial project. My real projects, made without AI, will be shared separately.

The project in numbers

Address	anime-planner.vercel.app
Data source	AniList, a free public anime database
Where your data lives	Only in your browser (nothing is sent to a server of ours, there isn't one)
Cost and account	Free, no account, no sign-in
Code	About 6,800 lines in 60 files, written in TypeScript
Automated tests	131 tests that check the calculations, dates, saving, security rules and the readability of colors
Hosting	Vercel, updated automatically from GitHub

What you can do with it

- **Find any anime**, by searching or by picking one of the shows that are popular right now.
- **Set your daily time**, either in minutes ("90 minutes a day") or in episodes ("3 episodes a day").
- **Skip openings and endings**: the app can remove about 3 minutes per episode for the opening and closing songs, if you skip them.
- **Choose when you watch**: every day, weekdays only, weekends only, or any days you pick; at a set time or "any time"; with an optional reminder; in your time zone.
- **Read your plan** as a list, day by day, with the real episode titles, or as a month calendar.
- **Save it and track your progress**: tick off each day; the homepage then shows a "Continue watching" row with your progress and a streak of consecutive days.
- **Add it to your calendar** (Google Calendar, Apple Calendar, Outlook...) with one file.
- **Share a link** to the show's page.
- **See where to watch** the show legally (official streaming services), the **watch order** of a franchise (season 1, season 2, the movie...), and the **trailer**.



Figure 2. A show's page. The whole page takes on the colors of the show's poster (here, green for Frieren).

What makes it different

It is built around your time, not around a list. Many anime sites help you keep a list of what you want to watch. This one turns a show into a real plan with dates, based on the time you actually have.

It only shows real information. The episode titles come from the AniList database. The app never invents a summary of what happens in an episode: no reliable source has such summaries for most shows, and an invented one could state wrong things about a real show as if they were facts. So the app shows real titles, or nothing.

It respects your privacy. There is no account and no server of ours storing anything. Your saved plans live in your browser. The only outside service the app talks to is AniList, to get information about shows.

2. The basics, explained simply

This chapter explains the ideas you need to understand the rest of the guide. If you already build websites, you can skip it.

What a website is

When you open a website, your browser (Chrome, Firefox, Safari...) downloads a few files from a computer somewhere on the internet, called a **server**, and turns them into the page you see. Three languages work together:

- **HTML** describes the *structure*: "here is a title, here is a button, here is an image". Think of it as the skeleton.
- **CSS** describes the *look*: colors, sizes, spacing, the layout on a phone or a computer. Think of it as the clothes.
- **JavaScript** describes the *behavior*: what happens when you click, type or scroll. Think of it as the muscles.

Frontend and backend

Many websites have two halves. The **frontend** is what runs in your browser. The **backend** is a program running on a server: it stores data in a database, handles accounts, and so on.

Anime Watch Planner has only a frontend. It is a set of files that any simple host can serve. All the work (searching, calculating, saving) happens in your browser. The information about anime comes from **AniList**, an outside service that offers its database to anyone. This makes the site cheap to host, fast, and private: there is no server of ours that could leak your data.

React: building a page from bricks

Writing a whole interface directly in HTML and JavaScript gets messy fast. **React** is a library (a toolbox of ready-made code) made by Meta that lets you build an interface from **components**: small, reusable bricks, like Lego.

Each component is a function that receives some information, called **props**, and returns what to display. For example, the project's `SectionHeading` component receives a title and draws it in the style used by every section. A component can also keep its own memory, called **state**: for example, whether a menu is open. When the state or the props change, React redraws only what changed on the page.

The project has about 20 components: the search bar, the anime card, the calendar, the forms, and so on. One special component, `App`, holds the most important state and decides which page to show (chapter 5).

TypeScript: JavaScript with safety checks

The code is written in **TypeScript**, which is JavaScript plus **types**: labels that say what kind of value each thing holds. For example:

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

This line says that `formatMinutes` takes a number and gives back text. If some code tried to pass it text instead of a number, TypeScript would report the mistake **before the site even runs**. Browsers only understand JavaScript, so TypeScript is converted to JavaScript when the site is built; the types are only there

to catch mistakes.

APIs, GraphQL and JSON

An **API** (Application Programming Interface) is a way for one program to ask another program for something, a bit like ordering from a menu. AniList offers an API: our app sends it a question ("give me the anime called Frieren") and receives an answer.

AniList's API uses **GraphQL**, a language where you describe *exactly* the information you want. For example, this asks for the title and episode count of the anime with number 154587:

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

The answer comes back as **JSON**, a simple text format for data that both humans and programs can read:

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

The tools around the code

- **npm** downloads and manages **packages**: code written by other people that the project reuses (React, for example). The list lives in `package.json`.
- **Vite** is the project's build tool. While developing, it runs the site on your own computer and refreshes it instantly when you change the code. For production, it **bundles** the many source files into a few small, optimized files.
- **Tailwind CSS** is a way of writing CSS with small ready-made class names directly in the components, such as `font-semibold` (semi-bold text) or `px-4` (horizontal padding).
- **anime.js** is a small animation library (the name is a coincidence; it has nothing to do with anime). It makes elements fade in, slide and bounce.
- **Vitest** runs the **tests**: little programs that check that the real code gives the right answers (chapter 13).
- **localStorage** is a small storage space that every browser gives each website. The app keeps your saved plans there (chapter 8).

Git, GitHub and Vercel

- **Git** records the history of every change made to the code, so nothing is ever lost and every change can be explained.
- **GitHub** stores that history online. This project's repository is private.
- **Vercel** hosts the site. Each time new code is sent to GitHub, Vercel automatically rebuilds the site and puts the new version online, usually in under a minute (chapter 14).

3. Using the app, step by step

This chapter walks through the app as a user sees it. The screenshots were taken from the live site with the show *Frieren: Beyond Journey's End* (28 episodes of 24 minutes).

Step 1: find a show

The homepage has a large search field and, below it, a grid of the shows that are **popular right now** (currently airing shows, sorted by popularity). Each card shows the poster, the average score given by AniList users (for example 87%), the title, the format (TV, Movie...) and the year.

Type a name and press **Search**. The results appear in the same kind of grid. Two small buttons switch between a **grid** of posters and a **list** with more details (studio, genres). **Clear** brings back the homepage.

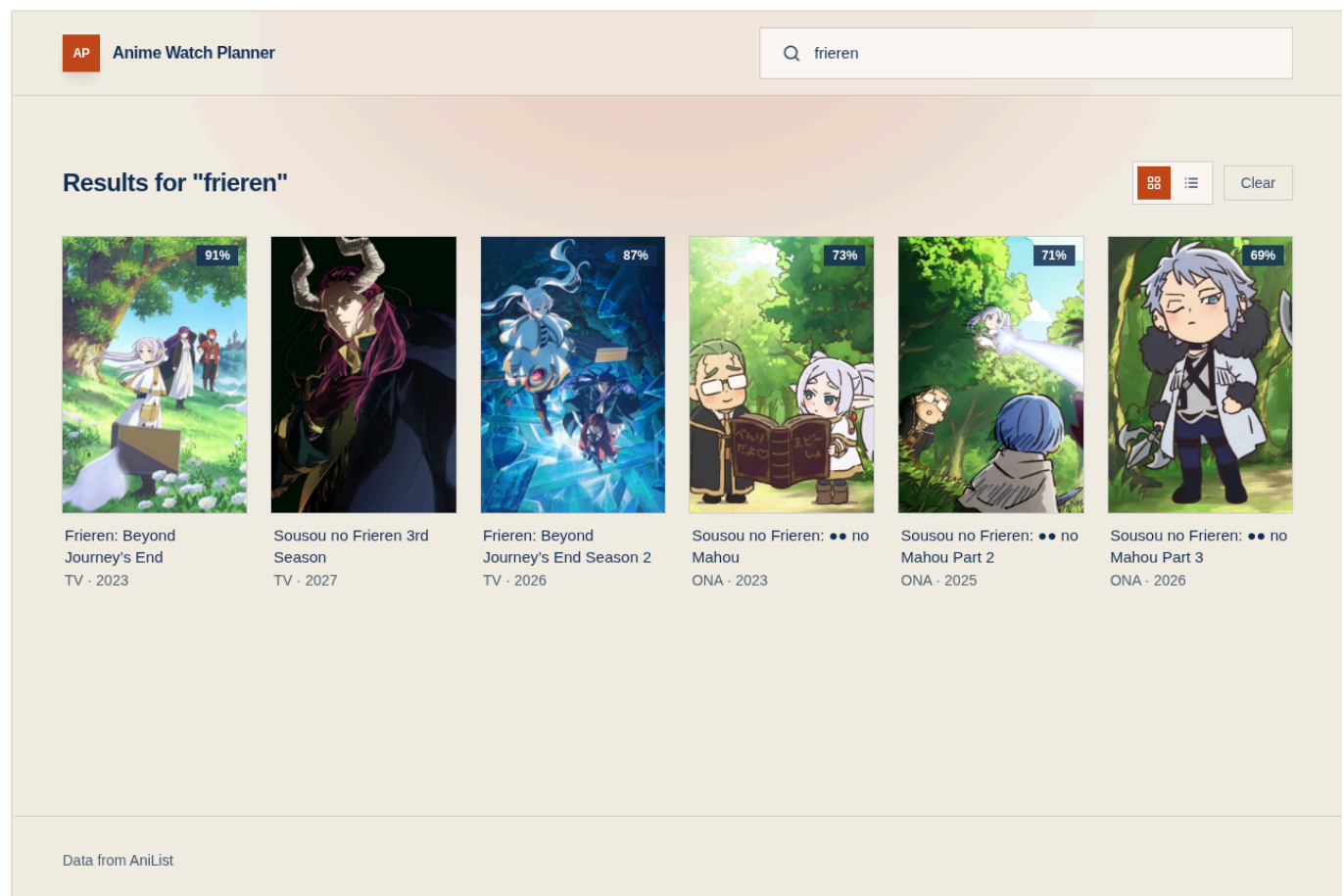


Figure 3. Search results for "frieren": the series, its seasons and its specials.

Step 2: the show's page

Click a show to open its page. At the top:

- the **poster**, the **title** (and the original Japanese title in Latin letters, called *romaji*, when it differs);
- a line of facts: format, year, studio, number of episodes, minutes per episode;
- the **genres**, and if new episodes are still coming out, when the next one airs;
- the **synopsis**, shortened to three lines with **Read more**;
- the **Save** button, and **Watch trailer** when the show has one on YouTube.

The whole page changes color to match the poster: the background takes a light tint of it, and the buttons and highlights a deeper version, dark enough for their white text to stay readable. This is calculated from a single color that AniList provides for each show (chapter 10). The homepage keeps the site's own navy and orange.

Step 3: set your plan

On the left, the **Your plan** panel decides how many episodes go into each day:

- **By time:** type your daily watch time in minutes, or tap one of the quick values (30, 60, 90, 120, 180).
- **By episodes:** say how many episodes per day instead (1, 2, 3, 4, 6). The app converts this into minutes behind the scenes.
- **Skip openings and endings:** removes 3 minutes from each episode, for the opening and closing songs. This is never applied to movies, or to short episodes under 15 minutes.

The plan updates instantly as you change anything.

Step 4: choose when you watch

The **When you watch** panel turns the plan into real dates:

- **Days you watch:** tap the days of the week (M, T, W...), or use **Every day**, **Weekdays** or **Weekends**. At least one day must stay selected.
- **Watch time:** an optional time of day. Without one, calendar events are "all day". With one, they start at that time and last as long as that day's episodes.
- **Reminder:** available once a time is set: at start time, or 10, 30 or 60 minutes before.
- **Time zone:** which time zone decides what "today" is and when new episodes air. By default, your device's own.

Step 5: read your schedule

On the right, the top box says **when you will finish**, for example "Saturday, October 17", with a **Start date** you can change. Below it, four numbers: the number of watch days, episodes per day, minutes per episode, and the total time. If part of your daily time can't be used (90 minutes holds three 24-minute episodes, which take 72 minutes), a line says how many minutes go unused.

The schedule itself has two views:

- **List:** one row per watch day, with its date, its day number, its episodes ("Ep 1–3"), its length ("1h 12m") and the **real titles** of those episodes. The checkbox on the left marks the day as watched.
- **Calendar:** a month grid showing which episodes fall on which date. Click a date to see its details below the grid and tick it off there.

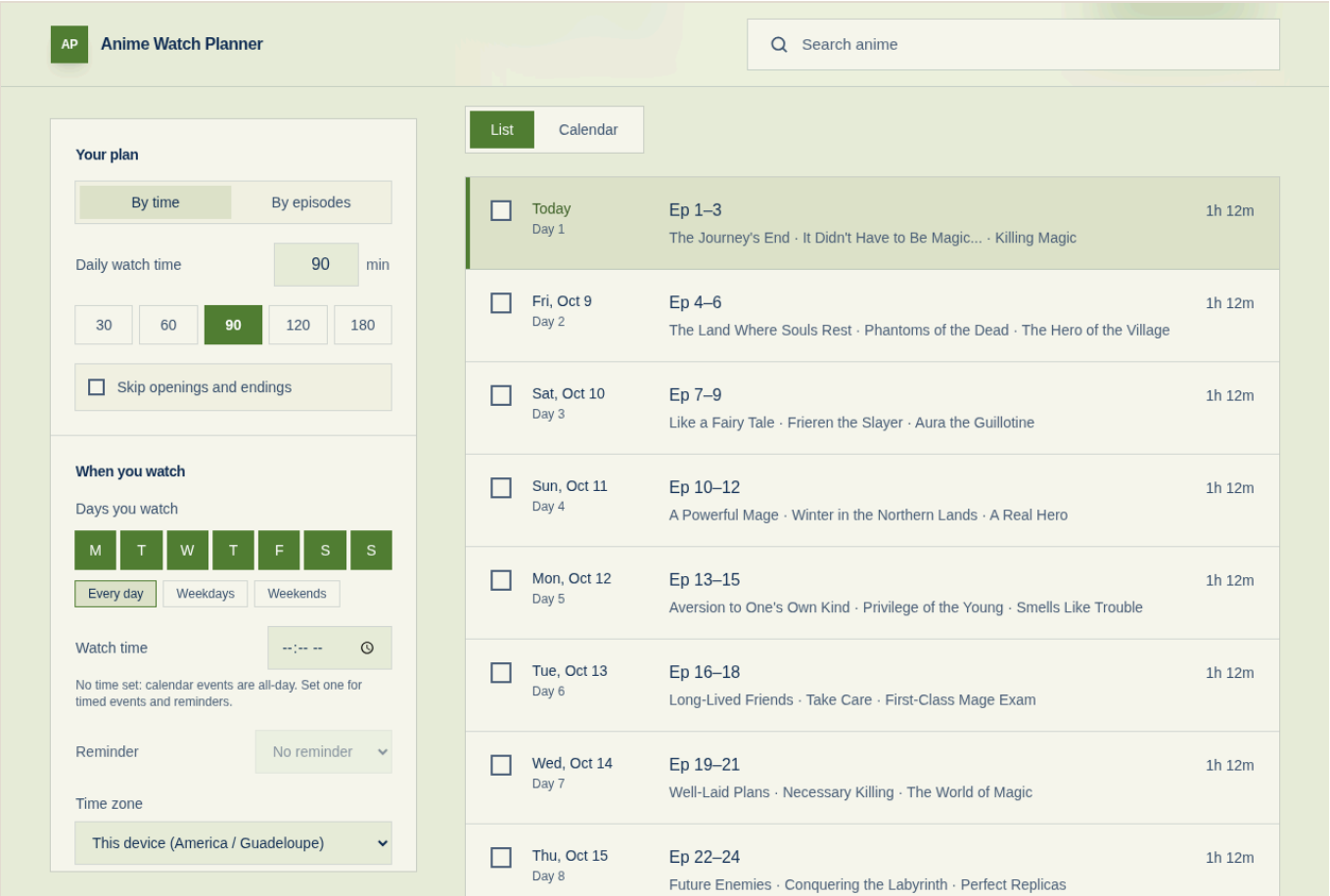


Figure 4. The list view: each day with its date, episodes, length and real episode titles.

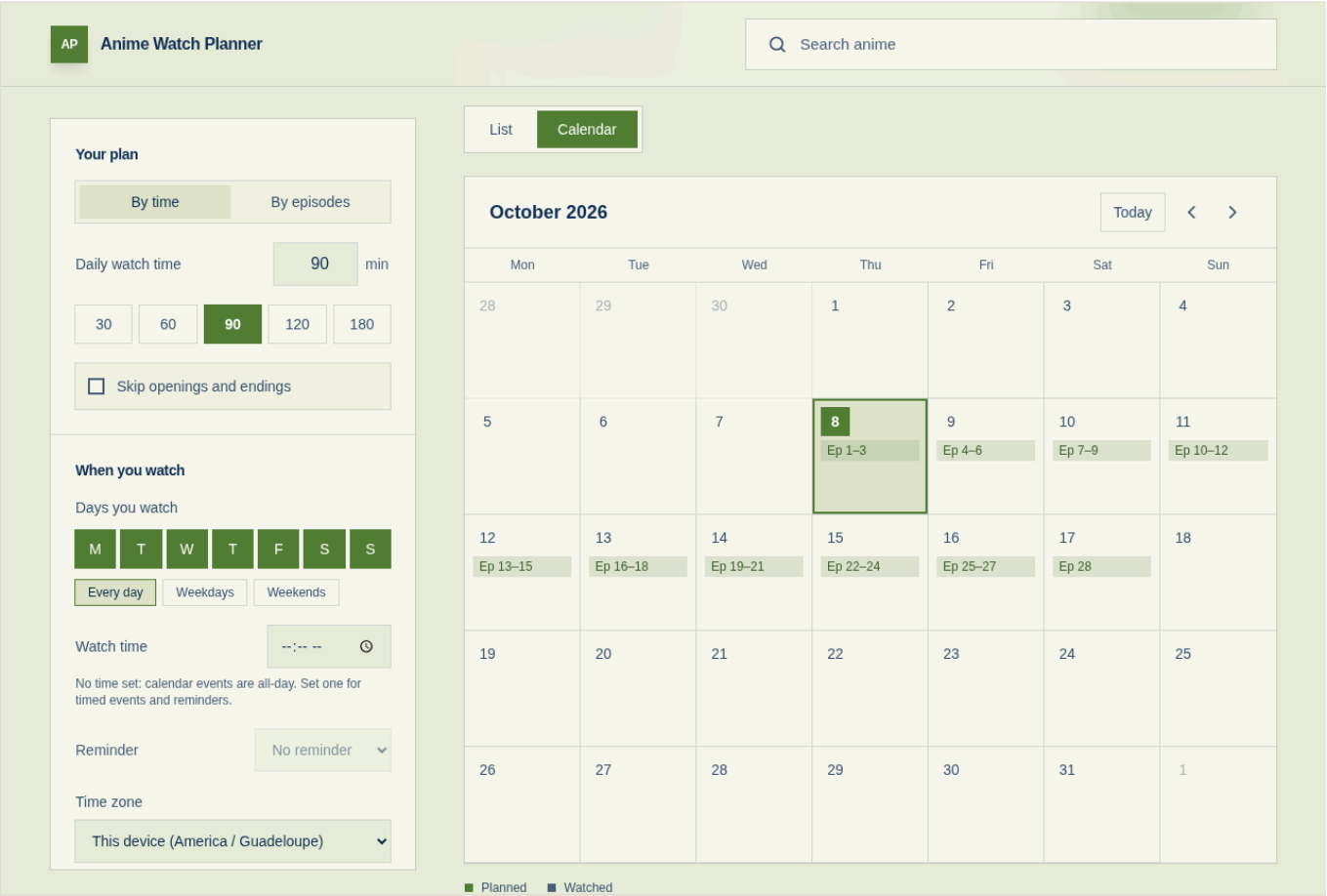


Figure 5. The calendar view: 28 episodes at 3 a day, from Thursday October 8 to Saturday October 17.

For a show that is **still airing** and has no final episode count yet, the app plans the episodes already out and says so in a box tinted with the page's color, with the date of the next episode. Episodes planned for a date before they air are marked **Not out yet**.

Step 6: save it and track your progress

Save stores the plan in your browser. The button then shows **Saved**; if you change a setting, it becomes **Update**; if the browser refuses to store anything, it shows **Retry save**. Ticking a day off saves the plan automatically, since progress needs somewhere to live.

Back on the homepage, a **Continue watching** row shows each saved plan with the day to resume ("Day 1 of 10 · today"), a progress bar, and when the next new episode airs. If you watched something on consecutive days, a **streak** appears ("3-day streak"). Inside a plan, a button jumps straight to **Today's episodes** or **Resume day N**.

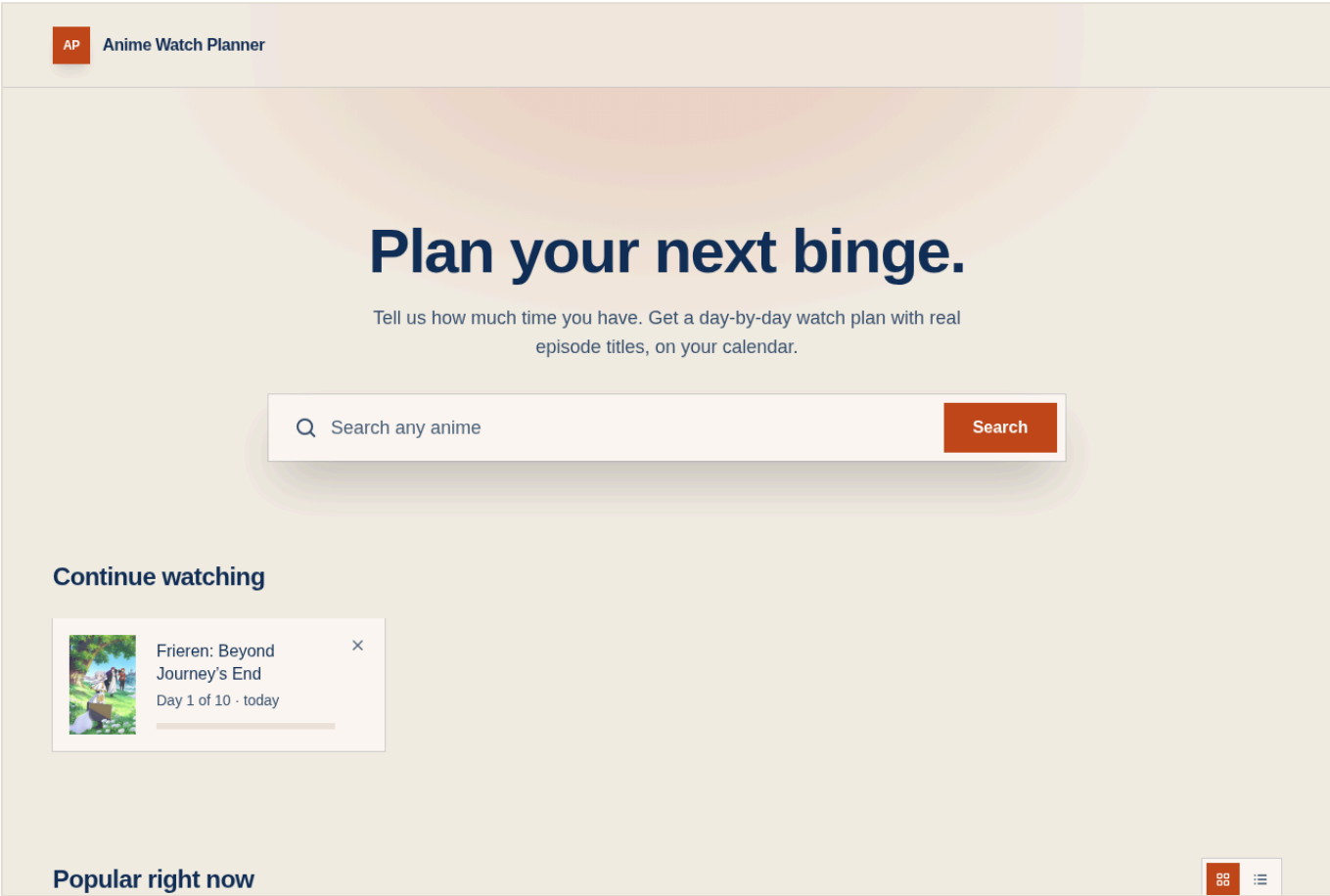


Figure 6. A saved plan in the Continue watching row.

Step 7: export and share

The **Share & export** panel has two buttons:

- **Add to calendar** downloads an `.ics` file. Opening it adds every watch day to your calendar app, with the episode titles in each event, and the air dates of upcoming episodes for shows still airing. On an iPhone or iPad, the file opens directly so Safari can offer "Add to Calendar".
- **Copy link** copies the show's address, such as `anime-planner.vercel.app/anime/154587-frieren-beyond-journey-s-end`. The person who opens it gets the show with fresh information and default settings.

The other panels

- **Where to watch** lists the official streaming services AniList knows for the show, with their logos.
Availability depends on your country.
- **Watch order** appears for franchises with several entries (season 1, season 2, a movie...), in release order.
Click an entry to open it.

On a phone

The layout adapts to small screens: the panels stack vertically, the calendar shows dots instead of labels, and text fields are large enough that phones don't zoom in when you tap them.

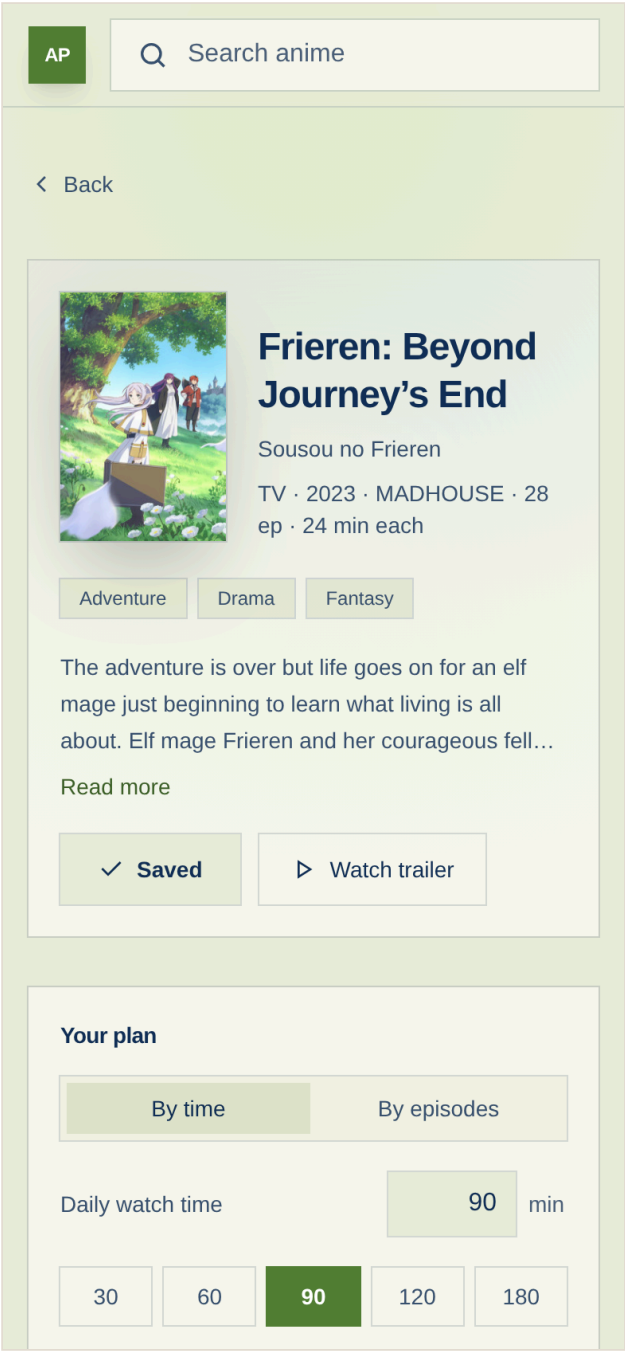


Figure 7. The same page on a phone.

4. How the project is organized

The folder tree

Here is every file in the project, generated from the real repository when this guide was built:

```
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

What each folder is for

Folder	Role	Examples
(root)	Project settings: which packages to use, how to check and build the code, how to host it	package.json , vite.config.ts , vercel.json
public/	Files served exactly as they are	the tab icon, robots.txt
src/	The application itself	App.tsx , main.tsx , index.css , types.ts
src/lib/	Pure logic: calculations with no screen and no network	the schedule engine, dates, progress, the .ics builder
src/api/	The connection to AniList	anilist.ts
src/motion/	Animation settings and helpers	presets.ts , hooks.ts
src/components/	The pieces of the interface	the search bar, cards, forms, the calendar
scripts/	Programs run while building the site	prerender.ts
guide/	The sources of this guide	chapters in French and English, the build script

The files ending in `.test.ts` next to the code in `src/lib/` are the **tests** (chapter 13).

The golden rule: keep the logic pure

The most important organizing idea in the project is the difference between **pure logic** and **side effects**.

A **pure function** only calculates: give it the same inputs and it always gives the same answer, and it changes nothing outside itself. "28 episodes of 24 minutes with 90 minutes a day gives 10 days" is pure. A **side effect** touches the outside world: drawing on the screen, downloading a file, asking AniList, reading the clock or the browser's storage.

The project keeps them apart:

- `src/lib/` holds pure logic. The engine (`schedule.ts`), the dates (`calendar.ts`), the progress math (`progress.ts`), the calendar file (`ics.ts`), the addresses (`route.ts`), the search tags (`seo.ts`), the formatting helpers (`format.ts`) and the colors of a show's page (`theme.ts`) never touch the screen or the network.
- The few side effects are isolated in small, clearly named places: `storage.ts` (browser storage), `download.ts` (file downloads), `platform.ts` (device check), `api/anilist.ts` (network), and the components and `App.tsx` (screen and state).

Why does this matter? Pure code is easy to understand and **easy to test**: a test just gives it inputs and checks the answer, with no browser and no internet. That is why all 131 tests target `src/lib/`.

The dependencies

Packages are code written by others that the project reuses. They are listed in `package.json`, in two groups: `dependencies` end up in the site visitors download; `devDependencies` are only used while developing and building. This table is generated from the project's real files:

Package	Exact version	Used
<code>animejs</code>	4.5.0	in the site
<code>react</code>	18.3.1	in the site
<code>react-dom</code>	18.3.1	in the site
<code>@tailwindcss/vite</code>	4.3.3	for development
<code>@types/react</code>	18.3.31	for development
<code>@types/react-dom</code>	18.3.7	for development
<code>@vitejs/plugin-react</code>	6.1.2	for development
<code>highlight.js</code>	11.12.0	for development
<code>marked</code>	18.1.0	for development
<code>tailwindcss</code>	4.3.3	for development
<code>tsx</code>	4.23.15	for development
<code>typescript</code>	7.0.2	for development
<code>vite</code>	8.3.3	for development
<code>vitest</code>	5.0.3	for development

In total, `package-lock.json` pins the exact version of **158** packages: those in the table and everything they depend on in turn.

What they do:

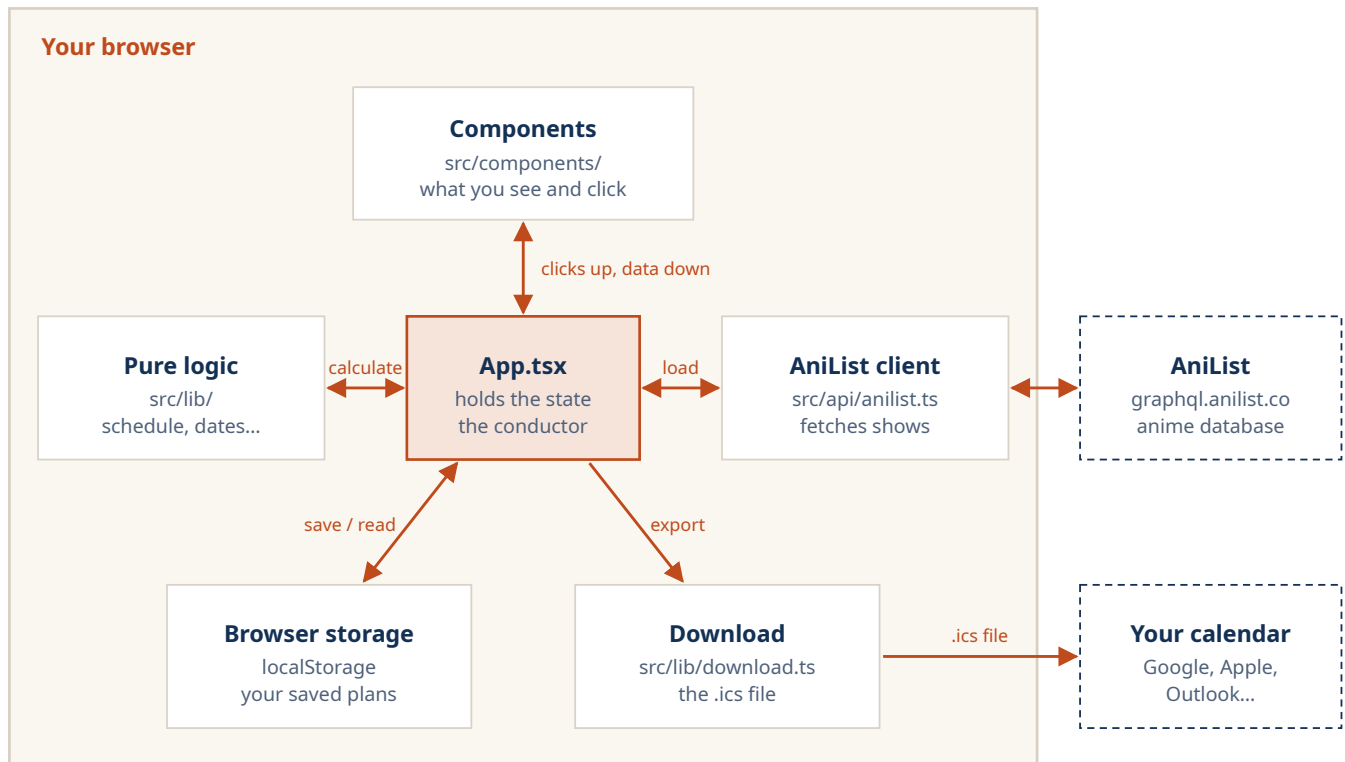
- **react** and **react-dom**: build the interface from components and draw it in the page.
- **animejs**: the animations.
- **tailwindcss** and **@tailwindcss/vite**: the styling system and its connection to Vite.
- **vite** and **@vitejs/plugin-react**: the development server and the production build, with React support.
- **typescript**: checks the types.
- **@types/react** and **@types/react-dom**: type descriptions for React, so TypeScript knows its functions.
- **vitest**: runs the tests.
- **tsx**: runs TypeScript scripts directly with Node.js (the prerender script and this guide's build).
- **marked** and **highlight.js**: turn this guide's chapters into HTML and color its code. They are only used to make the guide.

About package-lock.json

When npm installs a package, that package often needs other packages, which need others, and so on. `package-lock.json` records the **exact version of every one of them**, so that the project installs identically on any computer, today or in a year. It is written and updated automatically by npm, never by hand, and it is several thousand lines of names, version numbers and security fingerprints (hashes). That is why it is the **only file not printed** in Appendix A: it would add around fifty pages that nobody could read, without explaining anything that the table above doesn't.

5. How it all fits together

The big picture



Everything inside the large frame runs in your browser. Only two things are outside it: **AniList**, which the app asks for information about shows, and **your calendar app**, which receives the `.ics` file if you export your plan.

At the center is **App.tsx**, the conductor. It holds the important information (which show is open, your settings, your saved plans) and decides which page to display. Around it:

- **Components** draw what you see. They receive the data to display from **App**, and when you click or type, they tell **App** what happened.
- **Pure logic** (`src/lib/`) does the calculations: the schedule, the dates, the progress, the calendar file.
- **The AniList client** (`src/api/anilist.ts`) sends questions to AniList and tidies up the answers.
- **Browser storage** keeps your saved plans, through `src/lib/storage.ts`.
- **Download** turns the calendar text into a file your browser saves.

Data flows down, events flow up

React projects follow one simple rule, and this one follows it strictly: **data flows down, events flow up**.

App passes data down to its components as props: "here is the show, here are the settings, here is the result of the calculation". Components never change that data themselves. Instead, **App** also passes them functions to call, like `onSave` or `onConfigChange`. When you click Save, the button calls `onSave`; **App** saves the plan, updates its state, and React redraws whatever depends on it.

Because all the important state lives in one place, there is never any doubt about where a piece of information comes from or who is allowed to change it.

The state that App keeps

State	What it holds
<code>step</code>	Which page is shown: <code>'search'</code> (homepage and results) or <code>'configure'</code> (a show's page)
<code>query</code> , <code>searchedFor</code>	What is typed in the search field, and what the current results are for
<code>results</code> , <code>loading</code> , <code>error</code>	The search results, whether a search is running, and any error message
<code>popular</code>	The "Popular right now" list (<code>null</code> while it loads)
<code>selected</code>	The show that is open
<code>config</code>	Your plan: daily minutes and whether to skip openings and endings
<code>calendar</code>	When you watch: start date, days, time, reminder, time zone
<code>savedSchedules</code>	Every plan saved in this browser
<code>saveFailed</code>	Whether the last save failed
<code>opening</code>	Whether a show is loading from a direct link

Following one action: you click a poster

Here is exactly what happens, step by step, when you click a show on the homepage:

1. The poster is an `AnimeCard` component. Its button calls `onSelect(anime)` .
2. That function is `handleSelect` in `App.tsx` . If you saved this show before, it reuses your saved settings; otherwise it keeps the current ones. It calls `openAnime` .
3. `openAnime` updates the state: `selected` becomes this show, `step` becomes `'configure'` , and the address in the browser bar changes to the show's own address, such as `/anime/154587-frieren-beyond-journey-s-end` .
4. React redraws: since `step` is now `'configure'` , App shows the `ConfigureStep` page instead of the homepage, and gives the page the colors that `posterTheme` picked from the show's poster (chapter 10).
5. The schedule is calculated by `generateSchedule(selected, config)` . App wraps this in `useMemo` , a React tool that remembers the result and only recalculates when the show or the settings change.
6. The card only had the light list information (no synopsis, no episode titles). So in the background, `openAnime` asks AniList for the full details with `getAnimeById` . When they arrive, `selected` is updated and the page fills in the synopsis, the real episode titles, the streaming links and the watch order.

Another action: you change your daily time

1. You tap "60" in the plan panel. `ConfigForm` calls `onConfigChange({ dailyBudget: 60, skipOpEd: false })` .
2. App stores the new `config` .
3. Because `config` changed, `useMemo` runs `generateSchedule` again: 60 minutes now holds two 24-minute episodes a day, so the plan grows from 10 to 14 days.
4. `ScheduleView` redraws with the new result. The rows slide in again as a visual sign that the plan changed, and the numbers count to their new values.

Opening a link directly

When someone opens `anime-planner.vercel.app/anime/154587-...` directly (from a shared link, for example):

1. The browser loads `index.html`, which loads the code. `main.tsx` draws `App` in the page.
2. When `App` starts, it runs `resolveLocation`. This reads the address, and `parseAnimePath` extracts the show's number: 154587.
3. If this show is saved in the browser, it opens instantly from the saved copy. Otherwise, a "Loading show..." message appears while `getAnimeById` fetches it from AniList.
4. The back and forward buttons work too: the browser announces them with an event called `popstate`, and `App` runs `resolveLocation` again.

6. The scheduling engine

The engine is the heart of the project. It lives in `src/lib/schedule.ts`, about 100 lines, and does one job: **given a show and your plan, split the episodes into days**. It is pure (chapter 4): it doesn't know about dates, screens or the internet. It works in three steps, each with its own function.

Step 1: how long is one episode? (`getNetEpisodeRuntime`)

AniList gives each show a `duration`: the minutes per episode, or the total length for a movie. The engine starts from that number:

- If **skip openings and endings** is off, an episode lasts its full `duration`.
- If it is on, the engine removes **3 minutes** (about a minute and a half for the opening song and as much for the ending), but **only** if the show is not a movie **and** the episode lasts at least **15 minutes**. Very short episodes often have no real opening, and a movie has no openings between episodes.
- If AniList has no duration at all, the function returns `null`, meaning "unknown".

Show	Duration	Skip on?	Net length
A normal TV episode	24 min	no	24 min
A normal TV episode	24 min	yes	21 min
A short (TV short)	12 min	yes	12 min (too short to trim)
A movie	120 min	yes	120 min (movies are never trimmed)

Step 2: which episodes? (`getWatchableEpisodes`)

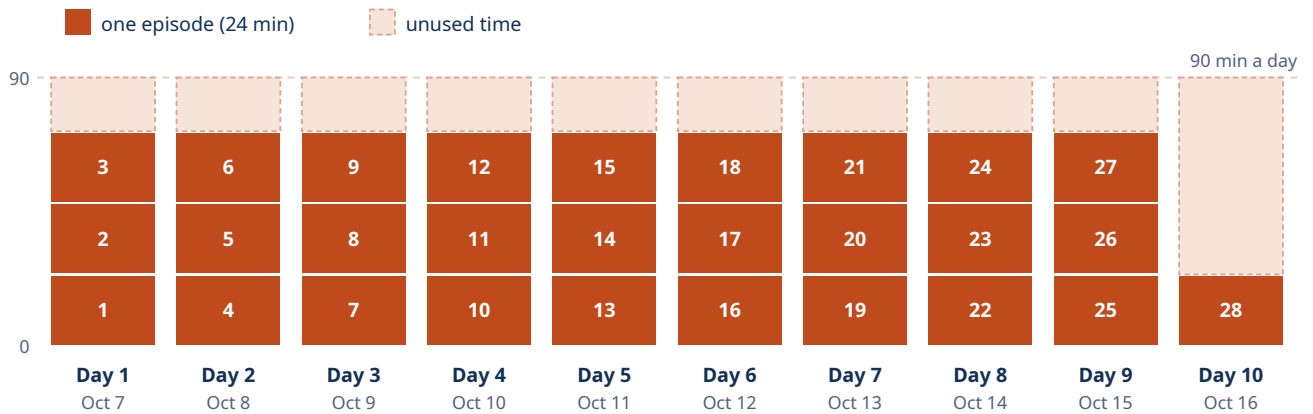
Next, the engine lists the episodes to plan:

- Normally, every episode from 1 to the total: a 28-episode show gives `[1, 2, ..., 28]`.
- A **movie** is always a single "episode": `[1]`.
- A show **still airing** often has no total on AniList yet (`episodes` is empty). Rather than give up, the engine uses the episodes **already out**. AniList says which episode airs next (`nextAiringEpisode`); if episode 1,150 airs next, then 1,149 are out, so the plan covers episodes 1 to 1,149. This is how a show like One Piece, with over a thousand episodes and no final count, still gets a real plan. The result is marked `isFallback`, and the page shows a "still airing" message.
- If there is truly nothing to plan (no total, not airing, or airing but nothing out yet), it returns `null`.

Step 3: split into days (`generateSchedule`)

The rule is simple: **as many whole episodes as fit in your daily time, every day**, in order, until the episodes run out. In maths: episodes per day = `floor(daily time ÷ episode length)`, where `floor` means "round down to a whole number".

Take *Frieren*: 28 episodes of 24 minutes, with 90 minutes a day. $90 \div 24 = 3.75$, rounded down to **3 episodes per day**. Three episodes take 72 minutes, so 18 minutes go unused each day. 28 episodes at 3 a day make 9 full days (27 episodes) plus a 10th day with the last episode alone: **10 days**, for a total of $28 \times 24 = 672$ minutes, or 11 hours 12 minutes. That is exactly what the screenshots in chapter 3 show.



The function returns either a plan or an error. A plan contains:

Field	Meaning	Frieren at 90 min
days	The list of days, each with its number, its episodes and its minutes	10 days
netRuntime	Minutes per episode after the optional trim	24
episodesPerDay	Episodes on each full day	3
episodeCount	How many episodes are planned	28
totalMinutes	Total watching time	672
unusedMinutesPerDay	Minutes left over on each full day	18
overBudget	One episode is longer than the daily time	no
isPartial	The episode count is "out so far", not a final total	no

When one episode is longer than your daily time

If you choose 45 minutes a day for a show with 50-minute episodes, zero episodes fit. Instead of returning an empty plan, the engine puts **one episode per day** and sets `overBudget`, and the page explains that each day runs over.

When a plan is impossible

The engine returns an error, which the page turns into a clear sentence:

Error	When	Message shown
invalid-budget	The daily time is 0, negative or empty	"Enter a daily watch time above 0 minutes."
unknown-duration	AniList has no episode length	"AniList doesn't list an episode length for this show."
unknown-episode-count	No total and nothing out yet	"AniList doesn't list an episode count for this show yet..."

The code, line by line

Here is the heart of `generateSchedule` (the full file is in Appendix 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(...)` rounds down: 3.75 becomes 3.
- If that gives 0, `overBudget` is true, and `Math.max(1, ...)` makes sure each day still gets at least one episode.
- The loop walks through the episode list in steps of `episodesPerDay`. Each time, `slice` cuts out the next group ("chunk") of episodes, and a new day is added with its number, its episodes and its length.

Why "greedy" and not optimal

This method is called **greedy**: it fills each day as much as it can, in order, without looking ahead. It wastes the leftover minutes (18 a day in the Frieren example). A cleverer method could, for instance, mix shorter and longer days to use every minute. The project deliberately keeps the simple rule: it is predictable (every full day looks the same), easy to explain, and easy to test. Episodes all have the same length anyway, so a cleverer method would rarely help.

"By episodes" is not a second engine

The plan panel lets you choose "3 episodes per day" instead of minutes. This is only a convenience in the form (`ConfigForm.tsx`): the form converts the choice into the exact minutes it takes ($3 \times 24 = 72$ minutes) and stores that. Since 72 divided by 24 is exactly 3, the engine lands on 3 episodes a day. The engine and the saved data only ever see minutes, so there is a single rule to test and trust.

7. Dates, the calendar and the .ics export

Two separate layers

The engine (chapter 6) only knows "Day 1, Day 2, ... Day 10". It has no idea what today's date is. Turning those numbers into real dates is the job of a second layer, `src/lib/calendar.ts`, which uses your **calendar preferences**:

Preference	Meaning	Default
<code>startDate</code>	The date you start, as <code>yyyy-mm-dd</code>	today
<code>watchDays</code>	The days of the week you watch (0 = Sunday ... 6 = Saturday)	every day
<code>watchTime</code>	A time of day, as <code>HH:MM</code> , or nothing	nothing ("any time")
<code>reminderMinutes</code>	How long before <code>watchTime</code> to remind you, or nothing	nothing
<code>timezone</code>	A time zone name such as <code>America/Guadeloupe</code> , or nothing for your device's own	nothing

Keeping the two layers apart means changing your watch days never changes which episodes go together, and the engine stays simple.

From day numbers to dates (`scheduleDates`)

The rule: **Day 1 is the first watch day on or after the start date; every later day is the next watch day after the previous one.**

Example: you start on Wednesday, October 7, and watch on **weekdays only**. Day 1 is Wednesday the 7th, Day 2 Thursday the 8th, Day 3 Friday the 9th. Saturday and Sunday are skipped, so Day 4 is Monday the 12th, and so on. The 10-day Frieren plan then ends on Tuesday, October 20 instead of Friday, October 16.

In code, the loop moves forward one day at a time and skips any day whose weekday isn't in your list:

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

If the list of watch days were somehow empty, this loop would never end. So the function treats an empty list as "every day", and the form never lets you turn off the last remaining day.

Why dates are text, calculated in UTC

Dates are stored as simple text like `2026-10-07`, and day arithmetic ("add one day") is done in **UTC**, the world's reference time, which has no daylight-saving changes.

Why? In most countries that change the clock, one day a year has only 23 hours and another has 25. Code that adds "24 hours" to move to the next day can then land on the same day twice, or skip one. Working with plain dates in UTC avoids this trap entirely; a test checks that a plan crossing a daylight-saving change keeps one date per day.

Only two functions deal with real moments in time, and so with time zones: `localIso`, which answers "what date is it right now, in this zone?", and `formatAiring`, which shows when an episode airs. They use **Intl**, the date and time toolbox built into every browser, which knows every time zone in the world.

The month grid (`monthGrid`)

The calendar view shows whole weeks starting on Monday. `monthGrid` finds the Monday on or before the 1st of the month and builds rows of 7 days until the month is covered. Days that belong to the previous or next month are marked so they can be shown faded. Moving between months is plain arithmetic on "year × 12 + month", which handles December to January without any special case.

Shows still airing

For a show that is still airing, AniList can give the next few air times (`airingSchedule`, only in the detailed request) or at least the next one (`nextAiringEpisode`). `upcomingAirings` takes the best of the two, keeps only future ones, sorts them, and works out each one's date in your time zone. The app uses them in several places:

- the show's header: "Episode 13 airs Sat, Oct 10 at 5:30 PM (in 3 days)";
- the calendar: a blue dot or label on air dates;
- the list: **Not out yet** on a planned day that comes before its episodes air;
- the saved plans row: "New ep 13 · Sat, Oct 10";
- the `.ics` file: one event per upcoming air time.

The .ics file

`.ics` (iCalendar) is a standard text format that every calendar app understands. `src/lib/ics.ts` builds it as plain text; `src/lib/download.ts` then hands it to the browser as a file. Here is one event, as the app writes it for Day 1 of Frieren with no watch time set:

```
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` is the event's permanent name. It is built from the show's number and the day number, so if you import the file again after changing your plan, your calendar **updates** the events instead of adding duplicates.
- Without a watch time, the event is **all day** (`VALUE=DATE`, ending the next day).
- With a watch time, say 20:00, the event starts at 20:00 and lasts that day's minutes. These times are written without a time zone ("floating"), so they mean 8 PM wherever you are, which is what a personal watch slot means.
- With a reminder, a `VALARM` block asks the calendar to alert you, for example 30 minutes before (`TRIGGER:-PT30M`).
- Air times of upcoming episodes are real moments, so they are written in UTC (ending in `Z`) and marked `TRANSP:TRANSPARENT`, meaning they don't make you look "busy".

The format has strict rules, and the code follows them: lines end with the two characters "carriage return + line feed"; commas, semicolons, backslashes and line breaks inside text are **escaped** with a backslash; and lines longer than 75 characters are **folded** onto the next line, which starts with a space. Escaping also matters for security: without it, a show title containing a line break could add fake lines to your calendar (chapter 12).

On iPhone and iPad

On iOS, downloading an `.ics` file is awkward. Instead, the app opens it directly in the tab, which makes Safari offer its own "Add to Calendar" screen. `src/lib/platform.ts` recognizes iPhones and iPads, including recent iPads, which pretend to be Macs: a "Mac" with a touch screen is treated as an iPad.

8. Saving your schedules and progress

Where plans are stored

There is no account and no server, so your plans are stored in your own browser, in **localStorage**: a small storage space (a few megabytes) that every browser gives each website. Everything lives under one name (a "key"): `anime-planner:schedules:v1`. The `v1` at the end is a version number, so a future, incompatible format could use a new key without confusing old data.

The value is a list of saved plans, written as JSON text. One saved plan looks like this (shortened):

```
{
  "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` is a random unique code.
- `savedAt` and `completedAt` are moments in time, counted in milliseconds since January 1, 1970 (a common convention in programming).
- `anime` is a **snapshot** of the show's information at save time, so the plan can be shown instantly, even offline.
- `config` and `calendar` are your settings.
- `watched` lists the episodes you have ticked off, and when.

One plan per show

Saving a show you already saved **replaces** its settings instead of adding a second copy, and keeps your progress. When you open a saved plan, the snapshot shows at once, then the app quietly fetches fresh information from AniList (an ongoing show may have new episodes).

Progress is per episode, not per day

This is one of the subtler choices in the project. When you tick off "Day 1", the app records **episodes 1, 2 and 3** as watched, not "day 1".

Why? Because days are recalculated whenever your plan changes. With 90 minutes a day, Day 2 is episodes 4 to 6. If you switch to 60 minutes a day, Day 2 becomes episodes 3 and 4. If progress were stored by day number, changing your plan would suddenly mark different episodes as watched. Stored by episode, your progress stays true whatever you change: a day simply shows as watched when **all** of its episodes are.

Ticking a day works like this (`toggleEpisodesWatched`): if every episode of that day is already watched, they are all unmarked; otherwise, the missing ones are marked, with the current time. And since progress needs a saved plan to live in, **ticking a day saves the plan automatically** if it wasn't saved yet.

Reading defensively

Browser storage is not always reliable. It can be turned off (in some private browsing modes), full, or contain damaged data. The rule in `storage.ts` is: **no storage problem may ever crash the app**.

- Every read and write is wrapped in `try ... catch`, a construct that catches errors. If reading fails, the app behaves as if nothing was saved; if writing fails, the Save button shows **Retry save**.
- Every saved plan is **checked** when read (`hasBaseShape`): it must have an id, a save time, a show with a number and a title, and valid settings. Anything else is ignored rather than trusted.
- Calendar preferences are checked field by field (`normalizeCalendar`). A missing or invalid field (a date like February 30, an unknown time zone) falls back to its default, while the valid fields are kept.

Upgrading old saves

The format of saved plans evolved during the project, and old saves must keep working:

- **Older saves stored progress by day** (`{ day: 2, completedAt: ... }`). When reading them, `normalizeWatched` recalculates that plan with its own saved show and settings, finds which episodes Day 2 contained, and converts the entry into those episodes, keeping the original time.
- **Saves from before calendar preferences existed** get default preferences, starting on the day they were saved, so their dates line up with the progress already made.

This approach is called **migrate on read**: old data is upgraded each time it is read, instead of creating a new storage key and leaving old data behind.

The progress numbers (`progress.ts`)

- **Resume day**: the first day that isn't fully watched. The "Resume day N" button and the "Continue watching" card use it.
- **Percentage**: fully watched days divided by total days, rounded.
- **Streak**: how many consecutive calendar days, counting back from today, you ticked something off, across all your plans. If you haven't watched anything yet today, the streak still counts as long as yesterday does, so it doesn't reset at midnight before you've had a chance to watch.

9. Talking to AniList

What AniList provides

AniList is a large, community-run anime database. It offers a free public **GraphQL API** at a single address, `https://graphql.anilist.co`. Read-only questions, like the ones this app asks, need no key and no account. Everything the app knows about a show comes from there: titles, episodes, lengths, posters, genres, studios, scores, synopses, episode titles, air dates, streaming links and related shows.

The four questions the app asks

`src/api/anilist.ts` contains four GraphQL queries:

Function	Question	Used for
<code>searchAnime</code>	The 12 best matches for a search	the search results
<code>getPopularAiring</code>	The 30 most popular shows currently airing	"Popular right now"
<code>getTrendingAnime</code>	The 25 shows trending right now	the prerendered pages (chapter 11)
<code>getAnimeById</code>	Everything about one show	a show's page

All of them exclude adult content (`isAdult: false`). Here is the search query:

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

`$search` is a **variable**: the text you typed is sent next to the query rather than pasted into it.

`...MediaFields` refers to a **fragment**, a shared list of fields used by all four queries, so their answers always have the same shape.

Light lists, rich details

The shared fragment only contains light fields: titles, format, status, episode count, length, year, score, poster, genres, main studio, next airing episode and trailer. A list of 30 shows stays quick to download.

The heavy fields are only requested for one show at a time, in `getAnimeById`: the synopsis, the episode titles (`streamingEpisodes`), the next 8 air times (`airingSchedule`), external links, and related shows two levels deep. That is why opening a show first displays what the card already knew, then fills in the details a moment later (chapter 5).

Sending a question (request)

Each question is sent as an HTTP **POST** request containing JSON: the query and its variables. Then:

- If AniList answers **429**, the app has asked too often. AniList allows roughly 90 requests per minute and says how many seconds to wait in a `Retry-After` header. The app shows "AniList rate limit hit, try again in 30s".

- If the answer contains errors or no data, the app shows AniList's own error message.
- Otherwise it returns the data.

To stay far below the limit, a search only runs when you press Search (not at every key you type). And when a new request starts, the previous one is cancelled with an `AbortController`, so a slow old answer can never replace a newer one.

Tidying up the answers

AniList's answers are reshaped before any component sees them:

- The studio list `studios { nodes { name } }` is flattened to a simple list of names.
- The poster color is checked by `safeHexColor` (it ends up in the page's styles, see chapter 12).
- **Where to watch** (`toWatchLinks`): AniList lists many external links. The app keeps only those of type `STREAMING` (official, licensed services), one per site, and only safe `https` addresses. It deliberately offers no unofficial source.
- **Upcoming episodes** (`toUpcomingEpisodes`): entries without a whole episode number and time are dropped, and the rest are sorted.
- **Watch order** (`buildWatchOrder`): starting from the open show, the app follows the "prequel" and "sequel" links two steps away, all in the same request, keeps TV series, movies, specials, OVAs and ONAs, removes duplicates, and sorts them by year. This matches the story order for almost every franchise; a show released out of story order could be listed "wrong", which is a known limit. A franchise stored as one long entry, like One Piece, simply has no list.

The episode titles trap

AniList's episode titles come from streaming services, as a list like `"Episode 130 - Scent of Danger"`. You might expect the first title to be episode 1, the second episode 2, and so on. **It isn't**. For One Piece, the list has 69 entries, in reverse order, starting at episode 130.

So the app never trusts the position in the list. `episodeTitleMap` reads the number written at the start of each title ("Episode 130"), and stores the title under that number. Titles without a readable number are dropped. The rule is: **a missing title is better than a wrong one**.

The description trap

The synopsis is requested with `description(asHtml: false)`, which should mean plain text. In practice, AniList still includes HTML tags: `<br>` for line breaks, and for some shows `<i>` (italics) or `<b>` (bold), plus codes like `&` for "&". Until this version, the app only removed `<br>`, so shows like Death Note, One Piece and Jujutsu Kaisen displayed raw text like `<i>Note: ...</i>`.

The new `synopsisText` function fixes this: `<br>` becomes a line break, every other tag is removed, and the codes are turned back into characters, with `&` handled last so that text which really says `<` stays as written. Four new tests cover it, including the real Death Note pattern.

"Popular right now", not "this season"

The homepage list shows shows **currently airing**, sorted by popularity, rather than "this season's shows". At the start of a season, almost every new show has no episodes out and no known length, so a "this season" list would be full of shows that can't be planned. The app also filters the list with the engine itself

(`isPlannable`), keeping 18 shows that can really be planned: 18 fills complete rows whether the grid has 2, 3 or 6 columns. If the request fails, the section simply disappears.

10. The interface: theme, components, animation and accessibility

The component tree

The page is built from nested components. Here is the tree, from the outside in:

- **ErrorBoundary** : the safety net around everything.
 - **App** : the state and the frame of the page (header, footer).
 - **SearchBar** : the search field in the header (on every page except the homepage).
 - **SearchStep** : the homepage, or the search results.
 - **SearchBar** (the large version), **SavedSchedules** ("Continue watching"), **SectionHeading**, the grid/list switch, and one **AnimeCard** or **AnimeListRow** per show; **Shimmer** placeholders while loading.
 - **ConfigureStep** : a show's page.
 - The header (poster, title, synopsis, Save and trailer buttons) and the trailer.
 - The side panel: **ConfigForm** (Your plan), **WatchPrefsForm** (When you watch), the streaming links with **StreamingIcon**, the watch order, and Share & export.
 - **ScheduleView** : the finish date, numbers (**CountUp**), messages and progress, then the list, or **CalendarView** for the month grid.

Small shared pieces: **icons.tsx** (stroke icons), **SectionHeading** (section titles), and **Motion.tsx** (**CountUp** , **Spinner** , **Shimmer** , **GlitchText**).

Two brand colors, and each show's own

The site's look comes from a mood board built on two Pantone colors: a navy blue (Classic Blue, **#133055**) and an orange (Adobe, **#BF4A1B**), on a warm cream background (**#F3EDE4**). They are defined once, at the top of **src/index.css** , and the homepage uses exactly these.

When you open a show, the page takes on the colors of its poster. AniList gives one color per show (for Frieren, **#bbf1a1** , a light green). **App.tsx** hands it to **posterTheme** , in **src/lib/theme.ts** , which returns two colors. **App.tsx** gives them to the page as two CSS variables:

- **--poster** , the **tint**: the background and the cards are mixed with a little of it, so the page takes on the show's hue (a pale green for Frieren, a pale pink for Jujutsu Kaisen).
- **--poster-accent** , the **accent**: the color of the buttons, toggles, progress bars, checkboxes and highlights.

Everything else is calculated from these two in **src/index.css** , with the CSS function **color-mix** , which blends two colors in a given proportion:

```
--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)) means "the tint, or cream if there is none". On the homepage, neither variable is set, so the page stays navy and orange.

Why the colors are calculated: readable text

Button text is white, and white text on a pale color, like Frieren's light green, would be very hard to read. Readability is measured with a **contrast ratio**, from 1:1 (two identical colors) to 21:1 (black on white). The accessibility guidelines (WCAG) ask for at least **4.5:1** for normal text. CSS can mix colors but can't measure contrast, so `posterTheme` does it:

1. A gray color has no hue to take on, so the function returns nothing and the page keeps navy and orange.
2. If white text on the poster color already reaches 4.5:1, that color is the accent, unchanged. Chainsaw Man's dark red (`#6b1a1a`) is used as it is.
3. Otherwise, the function darkens the color until white text reaches 4.5:1. It darkens in **OKLCH**, a way of describing a color by its lightness, its colorfulness and its hue: only the lightness goes down, so the color keeps its hue. Frieren's light green becomes a deeper green (`#518136`) instead of a muddy gray.

The tint gets the opposite treatment: a dark poster color is lightened, so a dark cover gives the page a light version of its hue instead of graying it.

`src/lib/theme.test.ts` checks the whole system with 4,096 colors spread over every hue and brightness: every kind of text must stay readable on every background it appears on (chapter 13).

Tokens and text colors

Components only use these **tokens** (`bg(--surface)` , `ring(--line)` , `text(--accent-text)` ...), never fixed colors, so the whole page follows automatically. Text uses three shades of navy: `--ink` for titles and body text, `--ink-2` for secondary text and `--ink-3` for hints. They are mixed with the fixed cream, so they stay the same on every show. The only exception is text printed over a cover image (the score badge), which stays white on a navy label.

Layout and Tailwind

The styling is written with Tailwind classes directly in the components. For example `bg(--accent) px-5 py-2.5 text-sm font-semibold text(--on-accent)` means: accent background, horizontal and vertical padding, small semi-bold text, in the color made for text on the accent (white). Prefixes like `sm:` , `lg:` apply a style only from a certain screen width, which is how the layout adapts:

- The poster grid has **2 columns** on phones, **3** on tablets and **6** on computers. The homepage shows 18 shows, a number that fills complete rows in all three cases.
- On a show's page, the side panel and the schedule stack vertically on small screens. On large screens, they sit side by side and the side panel stays visible while you scroll (it is "sticky").

Two practical details show the care taken. Text fields use at least a 16-pixel font on phones, because iPhones zoom into smaller fields when you tap them. And the outer frame uses `overflow-x-clip` rather than the more common `overflow-hidden`: the latter would silently break the "sticky" header and side panel.

The project also has two style rules:

- **No emoji or symbol characters** in the interface. They look different on every device. Instead, `icons.tsx` provides small drawn icons that take the color of the text around them.
- **Square corners everywhere**. No component uses Tailwind's `rounded-...` classes. Browsers draw checkboxes with rounded corners whatever the styles say, so `src/index.css` redraws them as plain squares, with a tick cut out by `clip-path`. Even the icons use square line ends.

Animation

Animations are done with the anime.js library, following strict rules (`src/motion/`):

- **Only movement and transparency are animated:** position, size, rotation, opacity, clipping. **Never colors**, because colors come from the theme tokens and must follow the poster automatically.
- **Short:** under about 600 milliseconds. The goal is "polished, never in the way".
- **Before the first paint:** animations start in `useLayoutEffect` , a React moment that runs before the browser shows the page, so nothing flashes in its final position before animating.
- **Clean afterwards:** once an animation ends, it removes the styles it added, so it leaves no trace.
- **Respectful:** if your device is set to "reduce motion" (an accessibility setting), every animation is skipped and elements appear directly in their final state.

What moves: pages and lists fade in, with a slight delay between items; numbers count up to their new values; the progress bar grows; the List/Calendar marker slides under the active button with a spring; calendar months slide sideways; the episode titles open like an accordion; every button squashes slightly when pressed (one listener for the whole page, `installPressFeedback`); and the header title plays a short "glitch" effect when you hover it.

Accessibility

Accessibility means making the site usable by everyone, including people who use a keyboard, a screen reader (software that reads the page aloud) or larger text.

- Everything clickable is a real `<button>` or link, so it works with the keyboard and is announced correctly.
- Toggle buttons say whether they are on (`aria-pressed`), and icon-only buttons have a text label (`aria-label`), such as "Remove Frieren from saved schedules".
- The calendar is a real grid (`role="grid"`), and each cell describes itself: "Thursday, October 8, Ep 1–3, watched".
- The search field has a label that is hidden on screen but read by screen readers.
- Keyboard focus is shown with a visible accent ring.
- Text keeps a contrast of at least 4.5:1 with its background, on every show's colors (see above), and the tests check it.
- Decorative images have an empty description, so screen readers skip them.
- Reduced motion is respected (see above).

When something goes wrong

If an unexpected error happens while drawing the page, React would normally leave a blank white page. This version adds `ErrorBoundary` , a component that catches such errors and shows "Something went wrong", with a **Reload** button and a link back home. Your saved plans are not affected: they are in browser storage, not in the page.

11. Addresses, search engines and link previews

Every show has its own address

Each show's page has a readable address, for example:

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

- `154587` is the show's number in AniList. **Only this number is used** to find the show.
- `frieren-beyond-journey-s-end` is the **slug**: the title, made safe for an address by `slugify` (lowercase, accents removed, anything that isn't a letter or digit replaced by a dash). It is only there for people and search engines. If the title changes on AniList, old links keep working, because only the number counts. A title with no Latin letters at all simply gets no slug: `/anime/123`.

`src/lib/route.ts` builds these addresses (`animePath`) and reads them back (`parseAnimePath`), accepting only exactly this shape.

Changing pages without loading a new page

The app has no "router" library. Its two views (homepage and show page) are a simple piece of state in `App.tsx`. When you open a show, the app changes the address in the browser bar with `history.pushState`, a browser function that updates the address **without** reloading the page. When you press Back or Forward, the browser sends a `popstate` event, and the app reads the address again (`resolveLocation`) to show the right view.

The problem with direct links, and the fix

The site is a **single-page application**: there is only one real HTML file, `index.html`. If someone opens `/anime/154587-...` directly, the host looks for a file with that name, finds nothing, and would answer "404 Not Found".

The fix is a **rewrite** rule in `vercel.json`: any address starting with `/anime/` is answered with the app's page, and the app then reads the address itself. A rewrite is not a redirect: the address in the browser bar stays the same. One detail cost a bug fix in the project's history: with Vercel's clean addresses setting on (`cleanUrls`), the rewrite must point to `/`, not `/index.html`, otherwise every show page that wasn't prerendered returns a 404. The same rule exists in `public/_redirects` for other hosts.

What search engines and link previews see

Each page describes itself to search engines and social networks with tags in the hidden part of the page, the `<head>`:

- `<title>`: the text in the browser tab and in search results, such as "Frieren: Beyond Journey's End watch schedule | Anime Watch Planner".
- `description`: a short summary (at most 160 characters), built from the facts ("28 episodes, 24 min each") and the start of the synopsis.
- `canonical`: the official address of the page, so search engines don't count variations of it as duplicates.
- **Open Graph** (`og:...`) and **Twitter** (`twitter:...`) tags: the title, description and image shown when you paste a link into Discord, WhatsApp, X, Slack.... The image is the large version of the poster; the card

format is `summary` because posters are tall rather than wide.

- **JSON-LD**: a block of structured data in the schema.org vocabulary that tells search engines "this page is about a TV series called..., with 28 episodes, of these genres". It deliberately contains no rating: AniList's list data has no number of votes, and a made-up one would be misleading.

All of this is computed by `src/lib/seo.ts`. The same functions are used in two places: by `App.tsx`, which updates the tags whenever you open a different show, and by the build script below. So the two can never disagree.

Prerendering: pages ready in advance

There is a catch. Updating tags with JavaScript works for Google, which runs JavaScript. But the robots that build link previews (Discord, Slack, WhatsApp...) don't run JavaScript: they read the raw HTML file, which only has the homepage's tags.

The answer is **prerendering**, done by `scripts/prerender.ts` right after the site is built:

1. It asks AniList for the trending and the popular shows: **two requests** in total.
2. For each show that can be planned, it writes a copy of `index.html` named after the show (for example `dist/anime/154587-frieren-beyond-journey-s-end.html`), with only the block between the `<!--seo-start-->` and `<!--seo-end-->` markers replaced by that show's tags.
3. It writes `sitemap.xml`, the list of all these pages for search engines (`robots.txt` tells them where it is).

When a visitor opens one of these addresses, the host finds the real file and serves it; the app loads normally and takes over. Every other show still works through the rewrite, with its tags set by the app once JavaScript runs. When this guide was written, about 20 shows were prerendered; the list refreshes on every deployment. If AniList is down during a build, the script prints a warning and the site is still published, just without these extra pages.

Old share links

Earlier versions shared plans with links like `/?share=eyJpZCI6...`, which encoded the show's number and your settings. They still work: the app decodes them (very carefully, see chapter 12), opens the show with those settings, and replaces the address with the clean one.

12. Security

What could go wrong?

The site has no accounts, no server of its own and no secret keys, so many classic dangers simply don't exist here: there is no database to steal and no password to leak. What remains is **data coming from outside** and reaching your browser:

- **AniList's data:** written by its community and moderated, but still text from strangers.
- **Links:** anyone can build a link to the site, including an old-style `?share=` link with any content.
- **Browser storage:** it could contain damaged or modified data.
- **The `.ics` file:** it contains show titles.

The main danger with outside data is called **XSS** (cross-site scripting): if text from outside were treated as code, someone could slip a hidden program into, say, a show's description, and it would run in your browser. Each rule below closes a door.

The rules

1. Outside text is always shown as text, never as HTML. The code never uses `dangerouslySetInnerHTML`, `innerHTML` or `eval`, the three usual ways of turning text into live code. React automatically shows text as text: even if a synopsis contained `<script>`, you would just see those characters. The new `synopsisText` cleans descriptions before display, and its result is still shown as plain text.

2. Only `https` links. A link can start with `javascript:`, in which case clicking it runs code instead of opening a page, and React 18 does not block this. So every address coming from data (streaming links, images) goes through `safeExternalUrl`, which accepts only `https:` (and upgrades `http:` to `https:`). Everything else is rejected.

3. Only real colors. The poster color from AniList ends up inside the page's styles (`--poster` and `--poster-accent`). `safeHexColor` only accepts a true hex color such as `#bbf1a1`. Then `posterTheme` reads it as numbers and writes out colors it calculated itself, so no other text can slip into the styles.

4. Trailers are fenced in. A trailer's YouTube id must be 6 to 20 letters, digits, dashes or underscores, so data can't steer the video frame to another address. The video uses `youtube-nocookie.com` (YouTube's privacy-friendly version), it runs in a **sandboxed** frame (limited in what it can do), and nothing loads from YouTube until you click "Watch trailer".

5. Share links are rebuilt, never trusted. `decodeShareParam` reads an old share link, then builds a brand-new object containing only the known fields, after checking each one: the show number must be a positive whole number, the daily time between 1 and 1,440 minutes (a full day), and the "skip" setting a true or false. Any extra field in a crafted link is ignored, and anything malformed is rejected.

6. Calendar text is escaped. In an `.ics` file, a line break starts a new instruction. If a title contained a line break, it could add fake instructions to your calendar. The code escapes line breaks, including the lone "carriage return" character that some calendar apps treat as a line break, as well as commas, semicolons and backslashes.

7. Saved data is checked. Saved plans are validated field by field when read (chapter 8), and anything unexpected is dropped.

8. Structured data can't escape its box. In the prerendered pages, the JSON-LD block sits inside a `<script>` tag. If a title contained `</script>`, it could close that tag early. The code writes every `<` in that block as `<`, which means the same character in JSON but can't close a tag. Other tags have their text escaped with `escapeHtml`.

Security headers

The host sends extra instructions, called **headers**, with every page. They are defined in `vercel.json` and, identically, in `public/_headers` for other hosts:

Header	What it does
<code>Content-Security-Policy (CSP)</code>	The most important one: a list of what the page is allowed to load. Scripts only from the site itself; images only from the site and AniList's image server; data requests only to AniList's API; frames only from youtube-nocookie.com. Even if someone managed to inject code, the browser would refuse to run it or let it send data elsewhere.
<code>X-Frame-Options: DENY</code>	Other sites can't show this site inside a frame. This prevents clickjacking : tricking you into clicking an invisible copy of the site.
<code>X-Content-Type-Options: nosniff</code>	The browser must trust the declared file type and never guess it, which closes a family of tricks.
<code>Referrer-Policy</code>	When you follow a link to another site, it only learns which site you came from, not the full address.
<code>Permissions-Policy</code>	The site declares that it never uses the camera, microphone, location or payments, so nothing on the page can ask for them.

A practical consequence: any new outside service must be added to the CSP, otherwise it is silently blocked on the real site while still working on a developer's computer (the development server doesn't send these headers).

Dependencies

Code written by others can have security flaws too. `npm audit` checks the project's packages against a public list of known problems. In this version, it reported one problem, in a package used only while building the site (never sent to visitors). It was fixed with `npm audit fix`, and the audit now reports **0 vulnerabilities**.

13. Tests

What a test is

A **test** is a small program that runs a piece of the real code with known inputs and checks the answer. If someone later changes the code and breaks something, the test fails and says exactly what went wrong, before the change reaches anyone.

Here is a real test from `src/lib/schedule.test.ts`. It checks the greedy rule from chapter 6 with a 10-episode show of 24 minutes and 90 minutes a day:

```
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);
});
```

Read it like a sentence: "it buckets episodes per day, greedily: generate a schedule for 10 episodes with the default settings, and **expect** 3 episodes per day, 18 unused minutes, the days `[1,2,3]`, `[4,5,6]`, `[7,8,9]`, `[10]`, and so on." `anime(...)` and `config(...)` are little helpers at the top of the file that build a fake show and default settings.

What is tested

The project has **131 tests** in 11 files, all for the pure logic in `src/lib/`. A test written once with a table of cases (`it.each`) counts once per case, as Vitest counts it:

File	Tests	What it checks
<code>schedule.test.ts</code>	22	Episode length with and without the trim (exactly 15 minutes is trimmed, shorts and movies never); the episode list for normal shows, movies and shows still airing; splitting into days; over-budget days; errors
<code>calendar.test.ts</code>	18	Real dates only (no February 30); adding days across months and years; weekdays; skipping non-watch days; no shift across a daylight-saving change; an empty watch-day list; month grids; air dates; time zones
<code>format.test.ts</code>	22	Episode titles matched by number, not position (shaped like One Piece's real data); safe links and colors; YouTube ids; "Ep 1-3, 5" ranges; slugs; cleaning descriptions, including the Death Note pattern
<code>storage.test.ts</code>	15	Save, replace, list newest first, delete; corrupt data and broken storage; ticking days on and off; upgrading old saves; default and invalid calendar preferences
<code>progress.test.ts</code>	12	Resume day; percentage; progress kept when days are re-split; every case of the streak
<code>ics.test.ts</code>	12	One event per day on the right dates; line endings; episode titles; escaping, including an attempt to inject lines through a title; timed events; reminders; air dates, and none from the past

File	Tests	What it checks
<code>share.test.ts</code>	11	Old share links round-trip; URL-safe text; unicode; garbage, wrong shapes and extra keys rejected
<code>theme.test.ts</code>	7	The colors of a show's page: no color, invalid and gray colors keep navy and orange; a dark color is used as it is; a pale one is darkened until white text reads at 4.5:1, keeping its hue; dark tints are lightened; and a sweep of 4,096 colors that checks every text against every background it appears on
<code>seo.test.ts</code>	5	Title, canonical address and image; short description without HTML; escaping so data can't break the page; movies marked as movies
<code>route.test.ts</code>	4	Building and reading show addresses; rejecting anything else
<code>platform.test.ts</code>	3	Recognizing iPhones and iPads, including iPads that pretend to be Macs

How the tests are set up

- The tests use **Vitest**, and live next to the code they test (`schedule.ts` and `schedule.test.ts` side by side).
- They run in **Node.js**, not in a browser, which makes them very fast: all 131 run in about a third of a second. This only works because the code they test is pure (chapter 4).
- Where the code needs something from the browser, the test provides a fake. For example, `storage.test.ts` replaces `localStorage` with a simple in-memory list, and can even make it fail on purpose to check that the app survives.
- Dates in tests are fixed (for example January 15, 2026) rather than "today", so the tests give the same result every day of the year.

Running them

```
$ npm test
```

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

What isn't tested automatically

The components (what you see on screen) have no automated tests: they were checked by hand and with screenshots, like the ones in this guide. This is a known gap (chapter 15). The most error-prone parts, the calculations, dates and data handling, are the ones the tests cover.

14. From code to website: build and deployment

Building the site (`npm run build`)

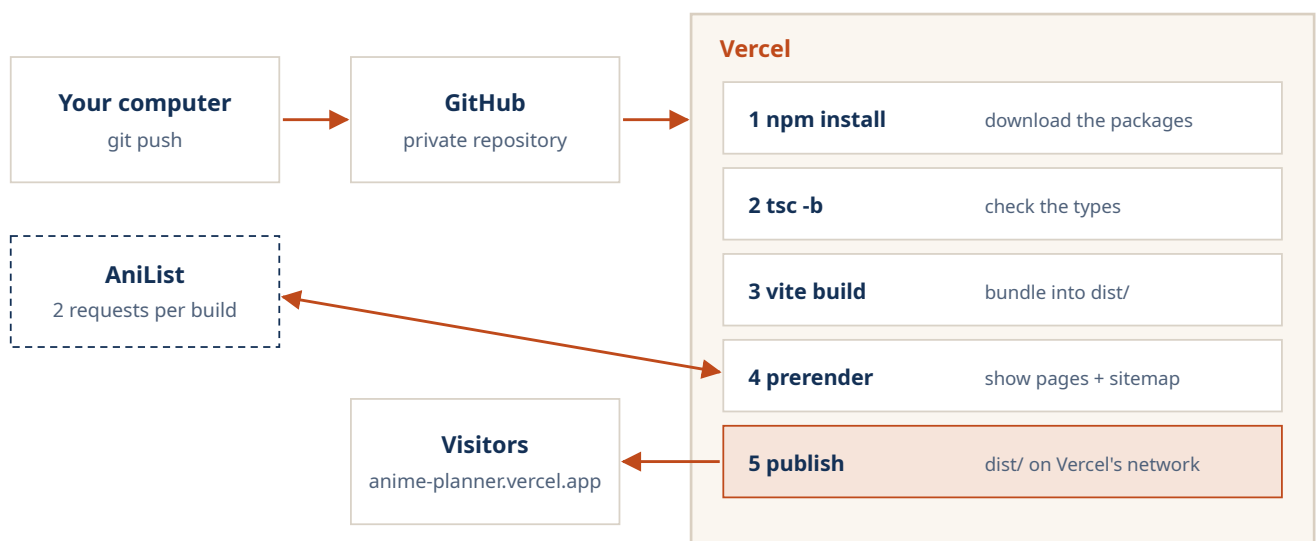
The code you read in Appendix A is not what visitors download. Building turns it into a few optimized files. The `build` command in `package.json` runs three steps, one after the other; if one fails, the build stops:

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

1. `tsc -b` : TypeScript checks the whole project for type mistakes. It produces no files; it only stops the build if something is wrong.
2. `vite build` : Vite bundles about 30 source files (and the libraries they use) into three: one HTML page, one CSS file and one JavaScript file, with names like `index-B2R0Yh5X.js` . The random-looking part changes whenever the content changes, so browsers can keep old versions in their cache safely. The JavaScript is about 260 KB, or about 87 KB once compressed for the network.
3. `tsx scripts/prerender.ts` : writes one page per popular show and the sitemap (chapter 11).

The result lands in the `dist/` folder: `index.html` , the `assets/` folder, the `anime/` folder of prerendered pages, `sitemap.xml` , and the files copied from `public/` .

Putting it online with Vercel



The site is hosted on **Vercel**, and deployment is automatic:

1. A change is recorded with git and sent to GitHub (`git push`).
2. Vercel is connected to the GitHub repository. It notices the new version and starts a **build** on its own computers: it installs the packages, then runs `npm run build` (the three steps above). The prerender step asks AniList for the popular shows.
3. If everything succeeds, Vercel publishes the `dist/` folder on its worldwide network of servers (a **CDN**), so the site loads quickly from anywhere. The whole process takes well under a minute. If the build fails, the previous version simply stays online.

`vercel.json` tells Vercel three things: serve pages without `.html` in their address (`cleanUrls`), answer every `/anime/...` address with the app (the rewrite of chapter 11), and send the security headers with every page (chapter 12).

The project also works on Netlify or Cloudflare Pages: `public/_headers` and `public/_redirects` hold the same rules in their format. These two files and `vercel.json` must always say the same thing.

How this guide is built

This guide is itself generated from the repository, by `npm run guide` (the `guide/` folder):

1. The chapters are written in Markdown, a simple text format, in `guide/en/` and `guide/fr/`. `marked` turns them into HTML.
2. The folder tree and the dependency table in chapter 4 are generated from the real files.
3. Appendix A is assembled from **every file git knows about**, in the order and with the descriptions of `guide/files.json`. `highlight.js` colors the code, and line numbers are added. **The build refuses to run if a file of the project is missing from the list**, so the appendix can't silently fall out of date.
4. The screenshots were taken from the live site by `guide/screenshots.ts`, which drives a headless Chromium browser (one without a window).
5. Chromium then "prints" each language's page to PDF, with page numbers, through its remote-control protocol (`guide/cdp.ts`).

15. Known limits and deliberate choices

No project does everything. This chapter lists what the app deliberately doesn't do, and its known limits, with the reasons. Being clear about limits is part of building software honestly.

Deliberately not done

- **AI-written episode summaries.** It would be easy to ask an AI to write "what happens in episode 5". But no public source (neither AniList nor MyAnimeList) has real per-episode summaries for most shows, and an invented summary of a real, released episode could state wrong things as facts. The app shows real episode titles or nothing.
- **Lists of YouTube videos and a music player.** These would need a YouTube API key. Such a key is limited to about 100 searches a day, shared by every visitor, and would be visible to anyone in the site's code. Trailers are fine: AniList provides their ids directly, with no key.
- **Anime conventions and events.** No free public source exists. The only options were a hand-made list that would go stale, or copying another site's listings, which their terms of use generally forbid.
- **Unofficial streaming sites and "filler" episode data.** "Where to watch" only lists official services. Data about filler episodes (episodes not based on the original story) would have to be copied from a fan site, with the same terms-of-use problem.

Known limits

- **Plans live in one browser.** There is no account, so plans don't follow you from your phone to your computer.
- **The schedule is greedy.** Leftover minutes are wasted each day (chapter 6). This is a deliberate simplification.
- **Only popular shows are prerendered.** About 20 to 50 shows get their own ready-made page with a full link preview. Other shows work normally, but services that don't run JavaScript (Discord, Slack, WhatsApp) show the generic preview for them.
- **Search shows 12 results,** with no "next page".
- **The watch order is sorted by year,** which can be wrong for a franchise released out of story order.
- **Still-airing plans only grow.** If AniList later corrects a show's episode count downward, an old saved plan updates only when reopened.
- **No automated tests for the interface.** The components were checked by hand and with screenshots.
- **The iPhone calendar path** can only be truly tested on a real iPhone.
- **Very long schedules** (One Piece has over 390 days at 90 minutes a day) show 30 days at a time, with "Show more" and "Show all", to keep the page fast. All days are still calculated.
- **One error boundary** protects the whole app. A finer version could keep the rest of the page working if one part failed.
- **Some show pages look soft.** AniList gives a single average color per cover. When it is pale or dull, like Demon Slayer's light peach, the page gets a muted tone; when it is gray, the page keeps navy and orange.

What changed in this version (October 2026)

This guide describes the project after a review and a few improvements:

- **Fixed:** synopses showed raw HTML tags such as `<i>` and `<b>` for 6 of the 40 most popular shows. The new `synopsisText` cleans them (chapter 9), with 4 new tests.
- **Added:** an error boundary, so an unexpected error shows a helpful page instead of a blank one (chapter 10).
- **Redesigned:** a new light look in two colors from a mood board, navy (Pantone Classic Blue) and orange (Pantone Adobe) on cream, with square corners everywhere (chapter 10).
- **Improved:** each show's page takes on its poster's colors through `posterTheme`, which darkens pale colors so white text on buttons stays readable (chapter 10), with 7 new tests.
- **Updated:** a security fix in a build-only package, and minor updates of Vite, Vitest and the React plugin. `npm audit` reports 0 vulnerabilities.
- **Published:** the site at anime-planner.vercel.app, deployed automatically from GitHub.
- **Documented:** a README for GitHub, and this guide.

16. Install and run it yourself

What you need

- **Node.js** version 22 (22.12 or later), 24 or 26. It includes **npm**.
- **git**, to get the code.
- For this guide only: the **Chromium** browser (or Chrome, by setting the `CHROME` variable to its path).

Get the code and start it

The repository is private. If you have access to it:

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

`npm run dev` prints a local address, usually `http://localhost:5173`. Open it in your browser: it is the full app, running on your computer. Change a file, save it, and the page updates instantly.

If you only have this guide, you have everything anyway: Appendix A contains every file. Recreate the same folders and files, plus a `package-lock.json` created by `npm install`, and the commands above work the same way.

The other commands

Command	What it does
<code>npm test</code>	runs the 131 tests (chapter 13)
<code>npm run build</code>	checks the types, builds the site into <code>dist/</code> and prerenders the show pages (chapter 14)
<code>npm run preview</code>	serves the built site locally, to check it before publishing
<code>npm run guide</code>	builds this guide's two PDFs into <code>guide/dist/</code>
<code>npm run guide:screenshots</code>	takes this guide's screenshots again from the live site

Publishing your own copy

1. Create a free Vercel account and import the GitHub repository, or run `npx vercel` in the project folder. Vercel recognizes a Vite project by itself.
2. If your copy gets a different address, replace `https://anime-planner.vercel.app` in four places: `SITE_URL` in `src/lib/seo.ts`, the default tags in `index.html`, the sitemap line in `public/robots.txt`, and the expected address in `src/lib/seo.test.ts`.
3. Every `git push` then publishes a new version.

Where to start reading the code

A good order to explore Appendix A:

1. `src/types.ts` : the shape of all the data.
2. `src/lib/schedule.ts` and its test: the heart of the project, short and pure.
3. `src/lib/calendar.ts` , then `progress.ts` and `storage.ts` .
4. `src/api/anilist.ts` : how data arrives.
5. `src/App.tsx` : how everything is wired together.
6. The components, starting with `ConfigureStep.tsx` and `ScheduleView.tsx` .

Ideas for going further

If you want to practise, here are improvements that fit the project's design:

- Add a "next page" button to the search results (AniList supports pages).
- Add tests for a component, for example `ConfigForm` , with a test library for React.
- Offer a "balanced" schedule that spreads episodes evenly across days, as an option next to the greedy rule.
- Let people export and import their saved plans as a file, to move them to another browser.

Appendix A. The complete source code

This appendix contains **every file in the project** (60 files, 6,827 lines), in a logical reading order: configuration, the page and public files, types, pure logic, talking to AniList, animation, components, the application, the build script, tests, then documentation. Each file starts with a short paragraph explaining its role; line numbers help you find your way. One file is not printed: `package-lock.json`, a list of several thousand lines generated automatically by npm (see chapter 4).

A.1 Project configuration

`package.json`

32 lines

The project's ID card for npm: its name, the available commands (`dev`, `build`, `test`, `guide` ...) and the libraries it needs, split between those used by the site (`dependencies`) and those only used while developing (`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 lines

The entry point of the TypeScript configuration. It only holds two references: one configuration for the application code and one for Vite's own config file.

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

tsconfig.app.json

19 lines

TypeScript's rules for the whole `src/` folder: strict mode, no unused variables, JSX for React. `noEmit` means TypeScript only checks the code; Vite produces the final files.

```
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 lines

TypeScript's rules for `vite.config.ts`, which runs in Node.js rather than in the browser.

```
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 lines

The configuration of Vite, the tool that serves the site during development and compiles it for production. It turns on two plugins: React and 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 lines

Settings for the Vercel host: clean addresses without `.html`, the internal rewrite of `/anime/...` pages to the app, and the security headers sent with every page (including the Content Security Policy, 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 lines

The list of files git should ignore: downloaded libraries (`node_modules`), generated files (`dist` , the guide's PDFs) and Vercel's local files.

```

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

```

A.2 The page and public files

index.html

29 lines

The site's only HTML page. It is nearly empty: a `<div id="root">` container that React draws the whole app into, and the block of SEO tags between the `seo-start` and `seo-end` markers, which the prerender script rewrites for each show.

```

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 lines

The small icon shown in the browser tab: an orange square with "AP" in white, drawn as SVG (a vector image format, so it stays sharp at any size).

```

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 lines

A file read by search engines: it allows them to crawl everything and tells them where to find the sitemap (`sitemap.xml`).

```

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

```

public/_headers

8 lines

The same security headers as `vercel.json` , in the format Netlify and Cloudflare Pages use, so the site can be hosted somewhere other than 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 lines

The same internal rewrite of `/anime/...` pages as `vercel.json` , in the Netlify and Cloudflare Pages format.

```

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 Shared types

src/types.ts

156 lines

The project's data dictionary. It describes the shape of every object: an anime (`Anime`), a plan setting (`ScheduleConfig`), calendar preferences (`CalendarPrefs`), a day of the schedule (`ScheduleDay`), a saved schedule (`SavedSchedule`)... TypeScript uses it to check that all the code handles these objects correctly.

```
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 Pure logic (src/lib)

src/lib/schedule.ts

101 lines

The calculation engine, the heart of the project. From an anime and a daily time, it works out an episode's net length, the list of episodes to watch and how they split into days. It is "pure": it only computes, with no network and no display, which makes it easy to test.

```

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 lines

Small display and safety helpers: picking the title to show, writing "2h 15m", cleaning AniList's description (synopsisText), matching episode titles to their numbers, only accepting https: links and valid colors, building a YouTube trailer address.

```

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]+/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 lines

The colors of a show's page. From the poster color AniList provides, `posterTheme` picks a button color that white text stays readable on (darkened in OKLCH when needed, keeping its hue) and a light tint for the background. A gray poster keeps the site's navy and orange.

```

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 lines

The date layer: it turns the engine's "Day 3" into a real date using the chosen start date and watch days, builds the calendar's month grids, and handles time zones and the air dates of upcoming episodes.

```

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' }).format(date);
75 }
76
77 export function todayIso(now: Date = new Date(), timezone?: string | null): string {
78     return localIso(now, timezone);
79 }
80
81 /** The zone the device is set to, e.g. "Europe/Paris" (or null if the runtime won't say). */
82 export function deviceTimeZone(): string | null {
83     const zone = new Intl.DateTimeFormat().resolvedOptions().timeZone;
84     return isTimeZone(zone) ? zone : null;
85 }
86
87 const FALLBACK_ZONES = [
88     'UTC', 'America/Los_Angeles', 'America/Denver', 'America/Chicago', 'America/New_York',
89     'America/Toronto', 'America/Mexico_City', 'America/Bogota', 'America/Lima', 'America/Sao_Paulo',
90     'America/Argentina/Buenos_Aires', 'Europe/London', 'Europe/Lisbon', 'Europe/Paris', 'Europe/Berlin',
91     'Europe/Madrid', 'Europe/Rome', 'Europe/Warsaw', 'Europe/Istanbul', 'Europe/Moscow', 'Africa/Lagos',
92     'Africa/Cairo', 'Africa/Johannesburg', 'Asia/Dubai', 'Asia/Kolkata', 'Asia/Bangkok', 'Asia/Jakarta',
93     'Asia/Manila', 'Asia/Singapore', 'Asia/Shanghai', 'Asia/Seoul', 'Asia/Tokyo', 'Australia/Perth',
94     'Australia/Sydney', 'Pacific/Auckland',
95 ];
96 let zones: string[] | null = null;
97
98 /** Every IANA zone the runtime knows, falling back to a list of major ones. */
99 export function listTimeZones(): string[] {
100     if (zones) return zones;
101     try {
102         const all = (Intl as { supportedValuesOf?: (key: string) => string[] }).supportedValuesOf?.('timeZone');
103         zones = all && all.length > 0 ? all : FALLBACK_ZONES;
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 lines

Progress calculations: whether a day is fully watched, which day to resume on, the percentage done, and the run of consecutive days (the "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()}-${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 lines

Saving schedules in the browser (`localStorage`). Every read and write is protected: if storage is full, disabled or corrupt, the app carries on without crashing. It also converts old save formats.

```

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 }
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 = getScheduledSchedules();
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 lines

Building the `.ics` calendar file (a standard text format that Google Calendar, Apple Calendar and Outlook can import): one event per watch day, with an optional reminder, plus the air dates of upcoming episodes.

```

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() + 1)}${pad(t.getUTCDate())}T${pad(t.getUTCHours())}${pad(t.getUTCMinutes())}00`;
27 }
28
29 function icsTimestamp(date: Date): string {
30   const y = date.getUTCFullYear();
31   const m = pad(date.getUTCMonth() + 1);
32   const d = pad(date.getUTCDate());
33   return `${y}${m}${d}T${pad(date.getUTCHours())}${pad(date.getUTCMinutes())}${pad(date.getUTCSeconds())}Z`;
34 }
35
36 /** [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 lines

Builds and reads the address of a show's page, like `/anime/154587-frieren-beyond-journey-s-end`. Only the number matters; the text after it is for people and search engines.

```

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 lines

Everything about search and sharing: each page's title and description, the preview tags for social networks, and JSON-LD structured data. The same code runs in the browser and during the 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 card 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 lines

Decoding the old `?share=...` links, which carried the show's id and settings. They still work, and decoding is very strict so a crafted link can't inject anything.

```

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(/>=|/|'/g, '');
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 lines

A single check: is the device an iPhone or iPad? On iOS the calendar export opens in the tab instead of downloading.

```

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' ? undefined : navigator): boolean {
10     if (!nav) return false;
11     if (/iPhone|iPod|iPad/.test(nav.userAgent)) return true;
12     return nav.platform === 'MacIntel' && nav.maxTouchPoints > 1;
13 }

```

src/lib/download.ts

37 lines

Triggers the download of a text file (the `.ics`) or opens it in the tab. Kept separate so `ics.ts` stays pure.

```

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 Talking to AniList

src/api/anilist.ts

333 lines

The AniList API client. It holds the GraphQL queries (search, popular shows, trending, one show's details), sends them, handles errors and rate limits, then reshapes the answers: official streaming links, a franchise's watch order, upcoming episodes.

```

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 Animation

src/motion/presets.ts

107 lines

Shared animation settings (durations, curves, spring) and a few helpers: fade-in entrance, a small confirmation "pop", and the press effect on every button. Everything is skipped if the user asked their system for reduced motion.

```

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 = 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 lines

React "hooks" that connect these animations to components: staggered entrances, sliding between months, counting numbers, the progress bar, a marker that follows the active button, accordion open and close.

```

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   useLayoutEffect(() => {
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<HTMLInputElement | null>,
130   pillRef: RefObject<HTMLInputElement | 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<HTMLInputElement>(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<HTMLInputElement | 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 Interface components

src/components/icons.tsx

109 lines

The small icons (magnifier, arrows, check, cross...) drawn in SVG, with square line ends like the rest of the site. They automatically take the color of the text around them.

```
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 lines

The heading style shared by every section, with an optional area on the right (a count, a button).

```

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

```

Reusable animated components: a counting number (`CountUp`), a loading spinner (`Spinner`), rippling loading placeholders (`Shimmer`) and the header title's glitch effect (`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 lines

The search field, in two sizes: large in the middle of the homepage, compact in the header on other 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
hover:brightness-90 sm:h-12 sm:px-7 sm:text-base"
52         >
53           Search
54         </button>
55       )}
56     </form>
57   );
58 }

```

src/components/AnimeCard.tsx

51 lines

A show's card in the grid: the poster, the percentage score, the title, format and year. On hover, a glow takes the poster's color.

```

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 lines

The same show presented as a row, for the list layout.

```

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 lines

The homepage's "Continue watching" row: one card per saved schedule, with the day to resume on, the progress bar (in the show's own color), the next episode to air and a remove button (with a closing animation).

```

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<HTMLUListElement>(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 }

```

src/components/SearchStep.tsx

193 lines

The homepage and search results: the big heading and search, saved schedules, the "Popular right now" grid, loading placeholders, and the button to switch between grid and list.

```

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     />
152     </section>
153   );
154 }
155
156 return (
157   <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         onSearch={onSearch}
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 lines

The "Your plan" form: daily time in minutes or in episodes, quick value buttons and the box to skip openings and endings. Episode mode is converted to exact minutes, so the engine only ever sees one setting.

```

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(--
line)">
53         {(
54           [
55             ['time', 'By time'],
56             ['episodes', 'By episodes'],
57           ] as const
58         ).map(([, label]) => (
59           <button
60             key={m}
61             type="button"
62             aria-pressed={mode === m}
63             onClick={() => switchMode(m)}
64             className={`h-8 text-sm transition ${
65               mode === m ? 'bg(--surface-2) font-medium text(--ink)' : 'text(--ink-2) hover:text(--ink)'
66             }}
67           >
68             {label}
69           </button>
70         ))}
71       </div>
72     )}
73
74     <div className="flex items-center justify-between gap-3">
75       <label htmlFor="budget" className="text-sm text(--ink-2)">
76         {byEpisodes ? 'Episodes per day' : 'Daily watch time'}
77       </label>
78       <div className="flex items-center gap-2">
79         <input
80           id="budget"
81           type="number"
82           min={1}
83           step={byEpisodes ? 1 : 5}
84           value={Number.isFinite(value) ? value : ''}
85           onChange={(e) =>
86             byEpisodes ? setEpisodes(Math.round(e.target.valueAsNumber)) : set('dailyBudget',
e.target.valueAsNumber)
87           }
88           className="h-10 w-20 bg(--bg) px-3 text-right text-base tabular-nums ring-1 ring(--line)
focus:outline-none focus:ring-2 focus:ring(--accent)"
89         />
90         <span className="w-6 text-sm text(--ink-3)">{byEpisodes ? 'ep' : 'min'}</span>
91       </div>
92     </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 ?
'opacity-50' : 'cursor-pointer text(--ink-2)}`}
122     title="Trims 3 minutes per episode. Not applied to movies or episodes under 15 minutes."
123   >
124     <input
125       type="checkbox"
126       checked={config.skipOpEd}
127       disabled={isMovie}
128       onChange={(e) => {
129         // In episode mode, keep the episode count when the runtime changes.
130         const nextRuntime = e.target.checked ? runtimes.trimmed : runtimes.full;
131         if (byEpisodes && Number.isFinite(episodes) && nextRuntime) {
132           onChange({ ...config, skipOpEd: e.target.checked, dailyBudget: episodes * nextRuntime });
133         } else {
134           set('skipOpEd', e.target.checked);
135         }
136       }}
137       className="h-4 w-4 shrink-0"
138     />
139     Skip openings and endings
140   </label>
141 </div>
142 );
143 }

```

src/components/WatchPrefsForm.tsx

189 lines

The "When you watch" form: days of the week, watch time, reminder and time zone.

```

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               {p.label}
104             </button>
105           );
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 lines

The streaming services' logos (Crunchyroll, Netflix, YouTube...) for the "Where to watch" section. A service with no logo gets a tile with its first letter, in its color.

```

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
19      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
20      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
21      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-
22      3.278 1.632" },
23    // Netflix
24    "netflix": { color: "#E50914", keys: ["netflix"], path: "m5.398 0 8.348 23.602c2.346.059 4.856.398
25      4.856.398L10.113 0H5.398zm8.489 0v9.172L4.715 13.33V0h-4.715zM5.398 1.5V24c1.873-.225 2.81-.312
26      4.715-.398V14.83L5.398 1.5z" },
27    // YouTube
28    "youtube": { color: "#FF0000", keys: ["youtube"], path: "M23.498 6.186a3.016 3.016 0 0 0-2.122-2.136C19.505 3.545
29      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
30      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
31      12s0-3.93-.502-5.814zM9.545 15.568V8.432L15.818 12L-6.273 3.568z" },
32    // Bilibili
33    "bilibili": { color: "#00A1D6", keys: ["bilibili", "bilibilitv"], path: "M17.813 4.653h.854c1.51.054 2.769.578
34      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-
35      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
36      3.773-1.574h.774L-1.174-1.12a1.234 1.234 0 0
37      1-.373-.906c0-.356.124-.658.373-.907L.027-.027c.267-.249.573-.373.92-.373.347 0 .653.124.92.373L9.653
38      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
39      .662.151.929.4.267.249.391.551.391.907 0 .355-.124.657-.373.906zM5.333 7.24c-.746.018-1.373.276-
40      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
41      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-
42      1.88-.773zM8 11.107c.373 0
43      .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-.
44      25-.249-.383-.569-.4-.96V12.44c0-.373.129-.689.386-.947.258-.257.574-.386.947-.386zm8 0c.373 0
45      .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-.
46      25-.249-.383-.569-.4-.96V12.44c.017-.391.15-.711.4-.96.249-.249.56-.373.933-.373z" },
47    // Tubi
48    "tubi": { color: "#7408FF", keys: ["tubi", "tubitv"], path: "M16.696 15.272v-.752c.4.548 1.107.917 1.934.917 1.475
49      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-2.674v6.712m1.57-2.614c0
50      .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
51      9.988v3.60 1.236.754 1.841 1.955 1.841.959 0 1.625-.396 2.028-1.064v.91h2.597V9.989h-2.675v3.14c0
52      .493-.346.693-.666.693-.321 0-.568-.192-.568-.655V9.989Zm14.39 0H24v5.276h-2.672M6.553 11.136c0 .781-.635 1.415-
53      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
54      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
55      0-.18-.006L-.003.006c-1.614 0-2.51-1.035-2.482-2.686v-.47H0V9.99h.92V8.563h2.569z" },
56    // 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 lines

The calendar view: a month grid showing, for each day, the planned episodes, what is already watched and air dates. Clicking a day shows its details below, where it can be checked off.

```

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-
149  4">
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-
158  2) 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"
166              className="grid h-9 w-9 place-items-center text(--ink-2) transition hover:bg(--surface-2)
167              hover:text(--ink)"
168            >
169              <ChevronLeftIcon className="h-5 w-5" />
170            </button>
171            <button
172              type="button"
173              onClick={() => setMonth((m) => shiftMonth(m, 1))}
174              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 }

```

src/components/ScheduleView.tsx

388 lines

The right side of a show's page: the finish date, the key numbers, warning messages, progress, the list/calendar switch and the day-by-day list with its checkboxes.

```

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   useLayoutEffect(() => {
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) setShown(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-
316           2)/50'}`}
317         >
318         {isResume && (
319           <span aria-hidden="true" className="absolute inset-y-0 left-0 w-1 bg(--accent)" />
320         )}
321         <label className="flex cursor-pointer items-start gap-4 px-4 py-4 sm:gap-5 sm:px-6 sm:py-5">
322           <input
323             type="checkbox"
324             checked={done}
325             onChange={(e) => {
326               onToggleEpisodes(day.episodes);
327               pop(e.currentTarget, 1.25);
328             }}
329             aria-label={`Mark day ${day.day}, ${formatDate(date)}, watched`}
330             className="mt-0.5 h-5 w-5 shrink-0"
331           />
332           <div className="flex w-24 shrink-0 flex-col gap-0.5 sm:w-28">
333             <span className={`text-sm font-medium ${isToday ? 'text(--accent-text)' : 'text(--ink)'}>
334               {isToday ? 'Today' : formatDate(date)}
335             </span>
336             <span className="text-xs tabular-nums text(--ink-3)">Day {day.day}</span>
337           </div>
338           <div className="flex min-w-0 flex-1 flex-col gap-1">
339             <div className="flex items-baseline justify-between gap-3">
340               <span className={`text-base ${done ? 'text(--ink-3) line-through' : 'text(--ink)'}>
341                 {describeEpisodes(day.episodes)}
342               </span>
343               <span className="shrink-0 text-sm tabular-nums text(--ink-2)">
344                 {formatMinutes(day.minutes)}
345               </span>
346             </div>
347             {titles.length > 0 && (
348               <p className="line-clamp-2 text-sm text(--ink-3) sm:line-clamp-1" title={titles.join(' · ')}>
349                 {titles.join(' · ')}
350               </p>
351             )}
352             {pending.length > 0 && (
353               <p className="text-sm text(--blue)">
354                 Not out yet: {
355                   pending.map((a) => `Ep ${a.episode} airs ${formatAiring(a.airingAt,
356                     calendar.timezone).day}`).join(', ')}
357                 </p>
358             )}
359           </div>
360         </label>
361       </li>
362     );
363   });
364 </ol>
365
366 {shown < s.days.length && (
367   <div className="flex flex-wrap items-center justify-center gap-3">
368     <button
369       type="button"
370       onClick={() => setShow((n) => n + DAYS_PAGE_SIZE)}
371       className="h-11 px-6 text-sm font-medium text(--ink) ring-1 ring(--line) transition hover:bg(--
372       surface)"
373     >
374       Show {Math.min(DAYS_PAGE_SIZE, s.days.length - shown)} more ({s.days.length - shown} left)
375     </button>
376     <button
377       type="button"
378       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 lines

A show's full page: the header (poster, title, genres, synopsis, Save and trailer buttons), the settings column (plan, days, where to watch, watch order, share and export) and the schedule.

```

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     onChange,
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="m-l-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     { /* Phones: poster beside the title, details full width below.
207       sm and up: poster on the left spanning both rows. */ }
208     <div className="grid grid-cols-[6.5rem_minmax(0,1fr)] gap-x-5 gap-y-6 p-5 sm:grid-cols-
209     [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-
338 (--ink) transition hover:text-(--accent-text)"
339                             style={{ borderLeftColor: link.color ?? 'var(--accent)' }}
340                         >
341                             <StreamingIcon site={link.site} color={link.color} />
342                             {link.site}
343                             <ExternalIcon className="h-3.5 w-3.5 text-(--ink-3)" />
344                         </a>
345                     ))}
346                 </div>
347                 <p className="text-xs text-(--ink-3)">Availability varies by region.</p>
348             </SidebarSection>
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)
transition hover:bg(--surface-2) hover:text(--ink)"
368             >
369               <span className="w-5 text-xs tabular-nums text(--ink-3)">{i + 1}</span>
370               <span className="min-w-0 flex-1 truncate">{r.title}</span>
371               {r.seasonYear && (
372                 <span className="text-xs text(--ink-3)">{r.seasonYear}</span>
373               )}
374             </button>
375           )}
376         </li>
377       )]}
378     </ol>
379   </SidebarSection>
380 )}
381
382 {result.ok && (
383   <SidebarSection title="Share & export">
384     <div className="grid grid-cols-2 gap-3">
385       <button
386         type="button"
387         onClick={handleAddToCalendar}
388         className="h-10 px-3 text-sm text(--ink) ring-1 ring(--line) transition hover:bg(--surface-2)"
389       >
390         Add to calendar
391       </button>
392       <button
393         ref={shareRef}
394         type="button"
395         onClick={handleShare}
396         className="h-10 px-3 text-sm text(--ink) ring-1 ring(--line) transition hover:bg(--surface-2)"
397       >
398         {shareCopied ? 'Copied' : 'Copy link'}
399       </button>
400     </div>
401     {shareLink && (
402       <input
403         readOnly
404         value={shareLink}
405         onFocus={(e) => e.currentTarget.select()}
406         aria-label="Share link"
407         className="h-10 w-full bg(--bg) px-3 text-sm text(--ink-2) ring-1 ring(--line)"
408       />
409     )}
410   </SidebarSection>
411 )}
412 </aside>
413
414 <section className="min-w-0">
415   <ScheduleView
416     result={result}
417     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 lines

The safety net: if an unexpected error happens while drawing the page, it shows a "Something went wrong" page with a Reload button instead of a blank page.

```

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(--
32         navy)/80">
33         <h1 className="text-2xl font-semibold tracking-tight text(--navy)">Something went wrong</h1>
34         <p>The page hit an unexpected error. Your saved schedules are still stored in this browser.</p>
35         <div className="flex flex-wrap gap-3 pt-2">
36           <button
37             type="button"
38             onClick={() => location.reload()}
39             className="bg(--orange) px-5 py-2.5 text-sm font-semibold text-white hover:brightness-90"
40           >
41             Reload
42           </button>
43           <a
44             href="/"
45             className="px-5 py-2.5 text-sm font-semibold text(--navy) ring-1 ring(--navy)/25 hover:bg(--navy)/5"
46           >
47             Back to home
48           </a>
49         </div>
50       </main>
51     );
52   }
53 }

```

A.8 The application

src/index.css

97 lines

The global stylesheet: it loads Tailwind, defines the site's colors (navy, orange, cream) and the theme tokens computed from the show's colors (--poster , --poster-accent), redraws checkboxes with square corners, turns off CSS

animations if the user asks, and sets up button transitions.

```
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 lines

The conductor. It holds all of the app's state (the search, the open show, the settings, saved schedules), calls AniList, manages the address in the browser bar, updates SEO, gives the page the open show's colors and picks which page to show.

```

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 }

```

The starting point: it finds the `<div id="root">` in `index.html` and draws the app into it, wrapped in the `ErrorBoundary` safety net.

```

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 The build script

Run after compiling: it asks Anilist for popular and trending shows, writes one HTML page per show with its own SEO tags (for link previews and search engines), then the `sitemap.xml` sitemap.

```

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>${lastmod}</lastmod></url>`)
54     .join('\n');
55   return `<?xml version="1.0" encoding="UTF-8"?>\n<urlset
xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">\n${urls}\n</urlset>\n`;
56 }
57
58 async function main() {
59   const shell = await readFile(join(DIST, 'index.html'), 'utf8');
60   if (!BLOCK.test(shell)) throw new Error('dist/index.html has no <!--seo-start-->/<!--seo-end--> block');
61
62   await writeFile(join(DIST, 'index.html'), withHead(shell, renderHeadTags(HOME_SEO)));
63
64   const shows = await curatedShows();
65   const paths = ['/'];
66   for (const anime of shows) {
67     const path = animePath(anime.id, displayTitle(anime));
68     // `<path>.html`, not `<path>/index.html`: Vercel (cleanUrls), Netlify and
69     // Cloudflare Pages all serve it at the extension-less URL.
70     const file = join(DIST, `${path}.html`);
71     await mkdir(dirname(file), { recursive: true });
72     await writeFile(file, withHead(shell, renderHeadTags(animeSeo(anime), animeJsonLd(anime))));
73     paths.push(path);
74   }
75
76   await writeFile(join(DIST, 'sitemap.xml'), sitemap(paths, new Date().toISOString().slice(0, 10)));
77   console.log(`prerender: ${shows.length} show pages + sitemap.xml (${paths.length} URLs)`);
78 }
79
80 main().catch((err) => {
81   console.error('prerender failed:', err);
82   process.exitCode = 1;
83 });

```

A.10 Tests

src/lib/schedule.test.ts

185 lines

Tests for the calculation engine: net length with or without openings and endings, the episode list (including movies and ongoing shows), splitting into days, and an episode longer than the daily time.

```

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 lines

Tests for the display and safety helpers: episode titles matched to the right number, filtered links and colors, YouTube addresses, episode ranges, readable addresses, cleaning 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   ])(
60     'rejects %s',
61     (url) => {
62       expect(safeExternalUrl(url)).toBeNull();
63     }
64   );
65 });
66
67 describe('safeHexColor', () => {
68   it('accepts hex colors and rejects anything else', () => {
69     expect(safeHexColor('#F88B24')).toBe('#F88B24');
70     expect(safeHexColor('#abc')).toBe('#abc');
71     expect(safeHexColor('red; background:url(https://x)')).toBeNull();
72     expect(safeHexColor('url(https://x)')).toBeNull();
73     expect(safeHexColor(null)).toBeNull();
74   });
75 });

```

```

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 });

```

src/lib/theme.test.ts

101 lines

Tests for a show's colors: missing, invalid or gray colors, colors used as they are, pale ones darkened to a readable contrast while keeping their hue, and a check of every text and background pair over 4,096 colors.

```

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')).toBeGreaterThan(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 lines

Tests for dates: valid days, crossing a month or year, skipped watch days, daylight-saving changes with no shift, month grids, air dates, time zones.

```

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 lines

Tests for progress: resume day, percentage, progress kept when days are re-split, and the streak calculation.

```

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 lines

Tests for saving, with a fake in-memory `localStorage`: save, replace, delete, survive corrupt data or broken storage, check off episodes, convert old 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 lines

Tests for the calendar file: one event per day on the right dates, episode titles, escaped special characters, timed events, reminders, air dates.

```

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 lines

Tests for page addresses: building, reading, and rejecting anything with the wrong shape.

```

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    });
26  });

```

src/lib/seo.test.ts

61 lines

Tests for SEO: a show's title, canonical address and image, a short description with no HTML, and escaping that stops

data from breaking the 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 long')).toBe(true);
36     expect(seo.description.length).toBeLessThanOrEqual(160);
37     expect(seo.description).not.toMatch(/<|>/);
38   });
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(/<\s\/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(animeJsonLd(anime())[ '@type' ]).toBe('TVSeries');
60   });
61 });
```

src/lib/share.test.ts

71 lines

Tests for the old share links: encode/decode round trips, special characters, and rejecting broken or crafted links.

```
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 lines

Tests for detecting iPhones and iPads, including the iPad that presents itself as a 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 Documentation

README.md

62 lines

The project's introduction on GitHub: what it does, the site address, features, technologies and commands.

```

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 lines

The project's architecture notes and design rules, written for a coding assistant (and readable by any developer): every important choice is explained there, along with what not to change without thinking it through.

```

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

```

those show. Pass it to `todayIso`, `upcomingAirings` and `formatAiring` at every call site (saved cards use their own record's zone). It's display/planning only on purpose: the `.ics` watch events stay floating (device-local when imported), and air-time events are UTC instants, already right in any zone.

- Air dates for shows still airing come from the detail-only `upcomingEpisodes` (AniList `airingSchedule`), falling back to `nextAiringEpisode` (`upcomingAirings`). They show in the header, the calendar, the "still airing" callout, list rows ("Not out yet") and the saved cards.
- `.ics` export: all-day events without a watch time; with one, timed "floating" local events (no time zone) lasting that day's minutes, plus a `VALARM` if a reminder is set. Upcoming air times are added as separate UTC events. UIDs are stable (`<id>-<day>`, `<id>-air-<ep>`), so re-importing updates events instead of duplicating them.
- The day panel's episode titles collapse under an "Episode titles (N)" header (`EpisodeTitles` in `CalendarView.tsx`, collapsed by default when $N > 3$). It only reveals the same real titles, never descriptions (see "Deliberately not doing" above).
- On iOS (`isIOS`), "Add to calendar" opens the file in the tab (`openTextFile`) so Safari can offer its "Add to Calendar" sheet; other platforms download it. That iOS path can't be tested without a real iPhone.

Saved schedules (`src/lib/storage.ts`)

- Key `anime-planner:schedules:v1` holds a `SavedSchedule[]` (see `types.ts`): an `Anime` snapshot, the `ScheduleConfig`, and a `watched` list of `{ episode, completedAt }`.
- Progress is **per episode, not per day**. Days are re-bucketed whenever the daily budget changes, so a day number doesn't identify the same episodes over time. A day shows as watched when all its episodes are (`isDayWatched` in `lib/progress.ts`); checking a day toggles its episodes (`toggleEpisodesWatched`).
- Old records stored `{ day, completedAt }`. `getSavedSchedules` migrates them on read by re-running the record's own saved anime + config through `generateSchedule` to recover each day's episodes. Records from before `watched` existed default to `[]`. Prefer this migrate-on-read pattern over bumping the storage key.
- One entry per anime id: re-saving replaces the config and keeps progress (and keeps calendar prefs when none are passed).
- `calendar` (`CalendarPrefs`) is validated field by field on read; bad or missing fields fall back to defaults. Records from before it existed start on the local date they were saved.
- Every read/write is in try/catch and entries are validated on read, so disabled/full storage or corrupt data degrades to "nothing saved" instead of crashing. Keep it that way.
- It's a snapshot, not a cache: opening a saved schedule renders the snapshot instantly, then `App.tsx` refetches the anime by id in the background. "Copy link" copies the show's clean URL, which carries no settings: the recipient gets fresh data and default settings (a deliberate trade for a readable, indexable URL). Old `?share=` links still open with their encoded config and are rewritten to the clean URL.
- Checking off a day implicitly saves the schedule first if it wasn't already saved, since progress has nowhere else to live.

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

```

210 ('description', 'streamingEpisodes', 'airingSchedule', 'relations',
211 'externalLinks') are
212 fetched only in `getAnimeById` and are `undefined` on list results;
213 `types.ts` marks them optional for this reason.
214 - `getAnimeById` normalizes AniList's raw shapes so components never see
215 them:
216 - `studios { nodes { name } }` becomes `{ name }[]`.
217 - `externalLinks` becomes `watchLinks`: **official STREAMING links
218   only**, one per site. This is the only "where to watch" source, on
219   purpose. Don't add unofficial/torrent sources (an extension index of
220   that kind was already declined for this project).
221 - `site` is free text, so `StreamingIcon` matches it loosely
222   (lowercased, letters/digits only) against a few embedded Simple Icons
223   marks (CC0: Crunchyroll, Netflix, YouTube, Bilibili, Tubi, Max) and
224   otherwise draws a letter tile in the link's brand color. Hulu, Prime
225   Video, Disney+, HIDIVE and Funimation aren't in Simple Icons (removed
226   at the brands' request); don't hand-draw their logos.
227 - `relations` becomes an ordered watch-order list (`buildWatchOrder`):
228   PREQUEL/SEQUEL edges fetched 2 levels deep in one request, deduped,
229   sorted by `seasonYear`. That's a release-year heuristic, not a true
230   topological sort, and can be wrong for a show released out of story
231   order. A franchise with no separate season entries (One Piece) gets no
232   list, which is correct.
233 - `description(asHtml: false)` is not really plain text: it keeps `  
`
234   tags, and some shows also carry ``/`` and entities (Death Note,
235   One Piece, Jujutsu Kaisen). The synopsis goes through `synopsisText`
236   (`lib/format.ts`: `  
` to newlines, other tags dropped, entities
237   decoded); SEO text goes through `plainText`, which builds on it.
238 - **Episode titles** (`streamingEpisodes`) must be looked up by the number
239   parsed from each title (`episodeTitleMap` in `lib/format.ts`), never by
240   array index. AniList's list can be partial and out of order: One Piece
241   has 69 entries, reversed, starting at episode 130. Titles with no
242   parseable number are dropped. A missing title beats a wrong one.
243
244 ## Theming and UI (`index.css`, `App.tsx`)
245
246 - Light theme in two brand colors from the owner's mood board: navy
247   `--navy` #133055 (Pantone Classic Blue) and orange `--orange` #BF4A1B
248   (Pantone Adobe), on cream `--cream` #F3EDE4. `--blue` #0F4C81 (Pantone's
249   own sRGB for Classic Blue) marks air dates only. These live on `:root` so
250   the body (`index.html`) and `ErrorBoundary` can use them.
251 - A show page takes on its poster's colors, like the old dark theme did.
252   `App.tsx` runs the open show's `coverImage.color` through `posterTheme`
253   (`lib/theme.ts`) and sets two inputs on the root `.theme-root` element:
254   `--poster` tints `--bg`, `--surface`, `--surface-2` and the glow at the
255   top, and `--poster-accent` replaces the orange `--accent` (buttons,
256   toggles, progress bars, highlights). Both are unset on the homepage and
257   for gray covers, which keep navy and orange.
258 - `posterTheme` exists because CSS can't check contrast. The accent is the
259   cover color, darkened in OKLCH (same hue) when white text on it
260   (`--on-accent`) would fall under 4.5:1. The tint is lightened when the
261   cover is dark, so the page takes the hue without going gray.
262   `--accent-text` is the accent darkened for text on cream. The saved-card
263   progress bars use the same accent.
264 - `lib/theme.test.ts` mirrors the `.theme-root` percentages in `index.css`
265   and checks every text/background pair over 4096 cover colors. Change the
266   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).
344 - Trailer ids are validated (`youtubeEmbedUrl`). The iframe is sandboxed,
345 uses youtube-nocookie.com, and only mounts after a click.
346 - `decodeShareParam` rebuilds the payload from known fields with range
347 checks and never passes the parsed object through.
348 - `.ics` text escaping handles `\\r` as well as `\\n`, so titles can't inject
349 calendar fields.
350 - Production security headers (CSP, `X-Frame-Options`, `nosniff`,
351 `Referrer-Policy`, `Permissions-Policy`) live in both
352 `public/_headers` (Netlify / Cloudflare Pages) and `vercel.json`. Keep the
353 two in sync. The CSP allows only `graphql.anilist.co` (fetch),
354 `s4.anilist.co` (images) and `www.youtube-nocookie.com` (frames). A new
355 external host needs a CSP entry, or it will silently fail in production
356 but work in dev (the dev server doesn't apply these headers). Don't move
357 the CSP into a `` tag: that would break Vite's dev server.
358
359 ## SEO and static pages
360
361 - `index.html` holds the default tags between `` and
362 `
```

```

364 Edit tags there, not in the HTML. The same functions drive:
365 - `App.tsx`: an effect keyed on `[step, selected]` updates the title,
366   meta tags, canonical link and JSON-LD while browsing.
367 - `scripts/prerender.ts` (after `vite build`): fetches trending + popular
368   (two requests), and writes `dist/anime/<id>-<slug>.html` for each
369   plannable show (the app shell with that show's tags) plus
370   `dist/sitemap.xml`. It also rewrites `dist/index.html`'s block from
371   `HOME_SEO`. If AniList is down it warns and still succeeds.
372 - Flat `.html` files, not `/index.html`: Vercel (`cleanUrls`), Netlify
373   and Cloudflare Pages all serve them at the extension-less URL.
374 - Show URLs that weren't prerendered fall back to the app shell through a
375   rewrite in both `vercel.json` and `public/_redirects`. A real file
376   always wins over the rewrite. Keep the two in sync, like the headers.
377   On Vercel the destination must be `/`, not `/index.html`: with
378   `cleanUrls` on, `/index.html` doesn't resolve as a rewrite target and
379   every non-prerendered show 404s.
380 - Link-preview images use `coverImage.extraLarge` (through
381   `safeExternalUrl`). `large` is AniList's medium size and looks blurry.
382   The X/Twitter card is `summary` because covers are portrait.
383 - JSON-LD has no `aggregateRating` on purpose: AniList's list data has no
384   rating count, and a made-up one is spammy structured data.
385
386 ## Declined integrations (checked, not just skipped)
387
388 - YouTube video lists and a music player: inline search results would
389   need a YouTube API key capped at ~100 searches/day, shared by every
390   visitor and exposed in the browser bundle. Trailers are fine: AniList
391   provides their ids directly, with no key needed.
392 - Anime conventions/events: no free public API exists (checked live).
393   It would mean a stale hand-curated list or scraping a listings site,
394   which is the same ToS-risk category as the AnimeFillerList scrape
395   declined for filler data. Flag the tradeoff again before adding it.
396
397 ## Known gaps / intentionally deferred
398
399 - Saved schedules live in one browser only – no sync across devices.
400 - Only the ~40-50 shows trending/popular at build time get prerendered
401   pages (real per-show tags for link previews and non-JS crawlers). Every
402   other show URL works through the rewrite and gets its tags from the app
403   once JavaScript runs, which Google handles but Discord/Slack previews
404   don't. The list refreshes on each deploy.
405 - Prerendered descriptions use list-view fields only (no synopsis), to
406   keep the build at two AniList requests.
407 - `scripts/prerender.ts` runs through `tsx` without type-checking (no
408   `@types/node` in the project). It imports the typed `src/` modules, so
409   keep its own logic thin.
410 - Tests cover only `lib/`; no component tests.
411 - One error boundary around the whole app (`ErrorBoundary.tsx`): a render
412   error shows a "Something went wrong" page with Reload, not a blank page.
413   There are no finer-grained boundaries per section.
414 - Bin-packing is greedy/naive (see above), not budget-optimal.
415 - No pagination on search results (AniList `perPage: 12`, page 1 only).
416 - `ScheduleView`'s day list paginates client-side (30 at a time, always
417   extended to include the resume day) for long schedules like One Piece's
418   390+ days. It's only a render-size guard: all days are computed, just
419   not all rendered at once.
420 - The aired-so-far fallback for ongoing shows never shrinks: if a show's
421   actual total episode count later drops from what "aired so far" implied
422   (rare, but AniList data can be corrected), an old saved schedule won't
423   self-adjust until reopened and refetched.
424
425 ## Conventions
426
427 - Components are presentational and take props; side effects (fetching,
428   state) live in `App.tsx`. Keep it that way rather than letting
429   individual components call the AniList API directly.
430 - Prefer editing `types.ts` when a shape needs to change, rather than
431   inlining a new ad hoc type in a component.
432 - This file and the code comments should stay in sync with each other –
433   if you find them disagreeing, fix both in the same change.

```

Appendix B. Glossary

API. A way for one program to ask another for information or actions, like ordering from a menu. AniList offers one.

Bundle. The few optimized files that a build tool produces from many source files.

Cache. A copy kept nearby (by your browser, for example) to avoid downloading the same thing again.

Canonical address. The official address of a page, given to search engines so that variations of it aren't counted as duplicates.

CDN (Content Delivery Network). A network of servers around the world that keeps copies of a site, so it loads quickly from anywhere.

Component. A reusable brick of a React interface: a function that receives props and returns what to display.

Contrast ratio. How different two colors are in brightness, from 1:1 (the same color) to 21:1 (black on white). Accessibility guidelines (WCAG) ask for at least 4.5:1 for normal text.

CSP (Content Security Policy). A security header that lists what a page is allowed to load and contact.

CSS. The language that describes the look of a web page.

CSS variable. A named value in CSS, such as `--accent`, that other styles reuse. Changing it in one place changes everything that uses it.

Deployment. Putting a new version of a site online.

Escaping. Writing special characters in a safe form so that they are read as text, not as instructions.

Fragment (GraphQL). A named, reusable list of fields shared by several queries.

Frontend / backend. The part of a site that runs in your browser / the part that runs on a server.

Git, GitHub. Git records the history of a project's changes; GitHub stores that history online.

GraphQL. A query language where you describe exactly the data you want back.

Greedy algorithm. A method that makes the best immediate choice at each step without looking ahead, like filling each day as much as possible.

Header (HTTP). Extra information sent with a web page, such as security instructions.

Hook (React). A function starting with `use` that gives a component an ability: memory (`useState`), remembered results (`useMemo`), actions after drawing (`useEffect`, `useLayoutEffect`).

HTML. The language that describes the structure of a web page.

HTTPS. The secure, encrypted version of the protocol used to load web pages.

IANA time zone. A standard name for a time zone, such as `Europe/Paris` or `America/Guadeloupe`.

iCalendar (.ics). A standard text format for calendar events, understood by every calendar app.

Intl. The date, time, number and language toolbox built into JavaScript.

JavaScript. The programming language of web browsers.

JSON. A simple text format for data, readable by people and programs.

JSON-LD. Structured data written in JSON that describes a page's content to search engines.

localStorage. A small storage space a browser gives each website, kept on your device.

Markdown. A simple text format for writing documents with titles, lists and links; this guide's chapters are written in it.

npm, package. npm downloads and manages packages: reusable code written by others.

OKLCH. A way of describing a color by its lightness, its colorfulness (chroma) and its hue, built so that equal steps look equal to the eye. Lowering only the lightness darkens a color without changing its hue.

Open Graph. Tags that control the title, text and image shown when a link is shared on social networks.

Prerendering. Building ready-made pages in advance, so services that don't run JavaScript still see the right content.

Props. The information a React component receives from its parent.

Pure function. A function that only calculates: same inputs, same answer, and no effect on the outside world.

Rate limit. A maximum number of requests a service accepts in a given time.

React. A library for building interfaces from components.

Regular expression (regex). A compact pattern for finding or checking text, such as "a date like 2026-10-07".

Rewrite. A server rule that answers one address with another page's content, without changing the address shown.

Sandbox. A restricted space in which something (here, the trailer frame) runs with limited powers.

SEO (Search Engine Optimization). Making pages easy for search engines to understand and show.

Single-page application (SPA). A site with one real HTML page, where JavaScript changes the content as you navigate.

Slug. The readable part of an address made from a title, such as `frieren-beyond-journey-s-end`.

State. The information a component or app remembers and that can change over time.

Tailwind CSS. A way of styling with small ready-made class names written in the HTML.

Test (unit test). A small program that checks that a piece of code gives the expected answer.

Token (design token). A named color or size, such as `--surface`, that components use instead of fixed values, so the whole look can change in one place.

TypeScript, type. JavaScript with types: labels saying what kind of value each thing holds, checked before the code runs.

UTC. The world's reference time, with no daylight-saving changes.

Vercel. The service that hosts the site and rebuilds it automatically from GitHub.

Vite. The tool that runs the site during development and builds it for production.

WCAG (Web Content Accessibility Guidelines). The international guidelines for making websites usable by everyone, including the minimum contrast between text and its background.

XSS (cross-site scripting). An attack where outside text is treated as code and runs in a visitor's browser.