group-project # sprint

Sprint 8 — Consumer App: Ship It

School of Engineering and Technology, University of Washington Tacoma
TCSS 460 — Client/Server Programming, Spring 2026

Due Date

Sunday, June 7, 2026, 11:59 PM

Sprint Narrative

This is it. Three sprints of consumer-app work — sign in, browse, write — and now you ship. Sprint 8 is when the app stops being a thing in progress and becomes a thing your users can find, open on their phone, navigate without getting lost, and credit you for. The center of this week's work is **style and theme consistency**: one typography scale, one spacing system, one color palette, one component vocabulary — applied uniformly across sign-in, browse, search, detail, profile, and the new About page. Sprint 7 set the bar at *intentional*; Sprint 8 raises it to *finished*. Around that core sit the rough edges you've been deferring: views that look fine on a laptop and break at 375px, keyboard paths nobody has tested, contrast that fails Lighthouse, posters that load slowly because nothing's sized them, browse pages that block on a full payload when they could stream the first batch. You also close the loop on the API you've been running for ten weeks — your downstream partner has been filing bugs in your `/issues` queue across three sprints; this is the week you close them out so finals week starts clean. End the quarter with an app you'd show someone.

MVP

By the end of this sprint, your consumer app presents a **coherent visual identity across every view** — a single typography scale, spacing system, color palette, and component vocabulary applied uniformly from sign-in through the About page, with no view stylistically out of step with the others. The app is deployed, responsive on a phone-sized viewport, and ships with an About page crediting your team and your upstream partner group, plus **one front-end feature your team invented** — something not in any spec, deployed and discoverable in the live app. A Lighthouse audit has been run against the deployed FE, the report is committed to the repo, and the top **accessibility and performance** findings have been addressed — the app is measurably better on both axes than it was at the start of the sprint. Your back-end's `/issues` queue is closed out — no open bugs from your downstream partner carried into finals week. A short end-of-quarter retrospective is in the team's meeting minutes.

Course Learning Objectives

This sprint contributes to the following course learning objectives:

- **LO 4:** Build interactive front-end applications using a component-based framework (Next.js — responsive layout, theme-driven components, accessibility, and the final design pass)
- **LO 5:** Deploy full-stack applications to cloud infrastructure (the consumer app shipping to a public URL alongside the back-end your team has run since Week 5)
- **LO 7:** Collaborate in teams using version control workflows, sprint milestones, and code review (closing out your `/issues` queue for your downstream partner, end-of-quarter retrospective)

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — reading a Lighthouse report and deciding which findings are real bugs vs. tool noise, identifying root causes when a view "feels wrong" on a phone, distinguishing polish work from perf bugs that hide under polish
- **Communication/Self-Expression** — writing the About page, naming your team's design system out loud rather than letting it emerge accidentally, running a retrospective that's honest enough to be useful

---

User Stories

As a team, we want our consumer app to present a single, coherent visual identity across every view so that a user moving between sign-in, browse,

search, detail, profile, and About never feels like they jumped to a different product.

This is *the* sprint story. Walk the app end-to-end and name your design system out loud: the typography scale (how many sizes, where each one is used), the spacing system (a 4px or 8px grid? MUI's `spacing` tokens?), the color palette (primary, secondary, surface, text, error — and what each one means in your app), the button hierarchy (primary / secondary / tertiary / text), the card and list patterns, the empty / loading / error states. Pick a set of components and reuse them at every call site; if a view is rolling its own, that's the work.

Sprint 7's bar was *intentional, not perfect* — this sprint's bar is *finished*. By the end of the week, nothing on any view should look like it was thrown together for that view only. If two pages use cards, they should look like the same card. If two routes have an empty state, the empty state should look and read like it came from the same hand. If two buttons mean the same thing, they should *be* the same button.

If you're on MUI, lean on the theme — extend it, don't fight it. Tokens for color, typography, spacing, and shape propagate everywhere they're referenced; changing one value should change the whole app. If you're hand-rolling, factor out the shared components and import them everywhere instead of restyling at every call site.

Guide

Styling & Component Libraries — Theme tokens, design systems, shared components, the MUI patterns from FE-2 that let you change one value and have it propagate

---

As a visitor, I want an About page so that I can see who built this app and what data it's built on.

A dedicated `/about` route, linked from the app's primary navigation. The page credits **your team** (names, plus the roles each member played across the quarter — back-end builders Sprints 1–4, front-end builders Sprints 5–8), names **your upstream partner group** whose API powers the browse / search / detail / rate / review experience, and acknowledges the shared services this app sits on top of: TMDB for the movie and TV metadata, Auth² for the OAuth2 sign-in, the course Token Playground if you used it.

A short paragraph on the story of the build is welcome but not required — what you learned, what surprised you, what you'd do differently. Keep the page consistent with the rest of the app's visual identity (Story 1) — it's the most personal page in the app, and it should still feel like part of the app.

As a team, we want to ship one front-end feature we invented ourselves so that the final app reflects something we actually wanted to build — not just what was asked of us.

You've spent three sprints implementing features the assignment specified. For Sprint 8, ship **one feature you came up with** — something the spec did not ask for, something you noticed missing or wanted while using the app, something that makes your team's consumer app distinct from the eight others.

The scope is your call. A few that fit comfortably in a one-week slot:

- A **watchlist** — let users mark items they want to watch later, persisted locally or against a partner route if their API exposes one
- A **search-filter UI** (genre, year, rating threshold) layered over the partner's existing search route
- **Keyboard shortcuts** for power users (`/` to focus search, `j` / `k` to navigate result lists)
- A **compare view** — pick two titles, render them side-by-side
- A **recently viewed** rail on the homepage, pulled from local storage
- A **theme toggle** (light / dark) — if you built a real theme in Story 1, this is cheap
- A **share button** that copies a deep link to the current detail page

These are examples, not a menu. The bar is *your team came up with this and shipped it*, not *you picked from this list*. Talk about it in your meeting minutes: what you considered, what you picked, why. The feature must be **deployed and discoverable** by someone opening the live app — a `TODO` in the README is not shipping.

As a user on a phone, I want the app to be usable at the screen width I actually have so that I don't have to pull out my laptop to read a review.

Walk the app at 375px wide (iPhone SE-ish) and 414px wide (larger phone) and fix what breaks: horizontal scroll where there shouldn't be any, navigation that hides the sign-in button below the fold, cards that wrap their content into illegible columns, posters that exceed the viewport, buttons sitting too close to each other to tap with a thumb. Tablet sizes are a bonus, not a requirement — the bar is *usable on the phone in the user's pocket*.

If you're on MUI, the responsive props (`xs` / `sm` / `md` / `lg`) and the Grid patterns from FE-2 are your starting point. If you're hand-rolling, this is what media queries are for. Chrome DevTools' device toolbar (Cmd+Shift+M) is the fastest way to verify without leaving your laptop.

Guide

Styling & Component Libraries — MUI responsive props, Grid breakpoints from FE-2

As a user, I want the app to meet baseline accessibility and performance standards so that it actually works for me regardless of how I navigate it or how slow my connection is.

Run **Lighthouse** against the *deployed* consumer app — Chrome DevTools → Lighthouse tab → "Analyze page load" — on the homepage and on at least one authenticated route (browse or detail is the usual choice). Save the report (HTML export or full-page screenshot) and commit it to the repo under `docs/lighthouse/` or similar.

Read the Accessibility and Performance findings. For each category, pick **the top three issues** the report flags and fix them — the high-impact stuff Lighthouse weights heavily. Document what you fixed (a couple of lines per fix is enough) in the same folder or in your README.

After fixes ship, run Lighthouse a second time against the deployed app and commit the new report alongside the first. The bar is *measurably better*, not a specific score threshold — perf scores swing with Render cold starts and a hard cutoff would punish you for hosting variance rather than your code. What we want to see is a before report, a list of what you fixed and why, and an after report showing the improvement.

Common findings you'll see and can almost always fix: missing alt text on poster images, form inputs without labels, low contrast on muted text, render-blocking resources, `next/image` not used for the poster grid, no `lang` attribute on `<html>`, missing `<main>` or other landmark elements.

Guide

Accessibility — The findings Lighthouse most often surfaces and how to fix them

---

As a team, we want our back-end's `/issues` queue closed out before finals week so that our downstream partner doesn't carry our open bugs into their wrap.

For ten weeks your group has run the back-end that your downstream partner's consumer app sits on. They've been filing bugs through your Bug Tracker FE since Sprint 5, and three

sprints of consumer-app work on their side means three sprints of bugs accumulated on yours. This week, walk the queue and triage every open report: fix what's a real bug, document what's a contract decision the partner just needs to know about, close out duplicates and won't-fixes with a comment explaining why. Use Postman or Prisma Studio against your admin-gated `/issues` routes — PATCH status, DELETE spam or resolved items.

The goal isn't zero open bugs — it's *zero bugs without a resolution*. A bug marked "won't fix, here's the contract, here's the workaround" is closed. A bug sitting open with no comment is not. If anything you find requires a real BE code change, deploy it; your downstream partner is shipping their final this week too and a fix landing on Tuesday gives them time to react.

As a team, we want to capture what we learned across the quarter so that the experience compounds for each of us individually and the team's last act is a deliberate one.

Hold a structured retrospective and write the result into your meeting minutes. Cover, at minimum:

- What went well across the ten weeks
- What you'd do differently if you were starting the quarter again
- What surprised you about working on someone else's API and having someone else work on yours
- What you learned about working with AI coding agents that you didn't know in Week 1

Two or three paragraphs per section is the right shape — long enough to be honest, short enough that you actually write it. This is for you, not for grading. Sprint 8 is the last time the team sits together with the quarter in view; spend twenty minutes on it.

Deliverables

- Consumer app presents a coherent visual identity — typography scale, spacing system, color palette, button hierarchy, and component vocabulary applied uniformly across every view
- About page at `/about` (or your equivalent), linked from primary navigation, crediting your team, your upstream partner group, and the shared services (TMDB, Auth²)
- One team-invented front-end feature — not in any spec — deployed and discoverable in the live app; rationale captured in the meeting minutes
- App is usable at 375px and 414px wide — no horizontal scroll, no broken layout, no untappable controls

- [ ] Lighthouse audit run against the deployed FE — **before** report committed to the repo
- [ ] Top three accessibility findings and top three performance findings addressed; fixes documented (a couple of lines each)
- [ ] Lighthouse **after** report committed alongside the before report, showing measurable improvement
- [ ] Consumer app is deployed and reachable at its public URL at submission time
- [ ] Back-end `/issues` queue is closed out — every open report has a resolution (fixed, won't-fix with explanation, or duplicate-closed)
- [ ] End-of-quarter retrospective documented in the team's meeting minutes
- [ ] All team members have committed to the consumer-app repository this sprint
- [ ] Meeting minutes document updated with sprint planning and any ceremonies

How to Submit

All work must be merged into `main` by the due date. Your instructor grades from the `main` branch of your GitHub Classroom repository. The consumer app continues in the **same Sprint 6 GitHub Classroom group repository** — Sprints 6, 7, and 8 all ship from the same consumer-app repo.

Guide Reference

Guide	What It Covers
Styling & Component Libraries	Theme tokens, design systems, MUI theme extension, responsive props, the FE-2 patterns that make one-value-changes-the-whole-app possible
Accessibility	The findings Lighthouse most often surfaces — labels, contrast, landmarks, focus order, alt text — and how to fix them
Next.js	App Router, server vs client components, `next/image` (a common Lighthouse perf win), environment variables for the final deploy
Deploying Next.js	Vercel + Render walkthrough, env vars, build commands, what to check when your deployed build looks different from your local dev build

Guide	What It Covers
Consuming a Web API	If you're still smoothing how the app calls the partner's API — loading states, error states, retry, the patterns that surface on the perf and a11y audits

Supporting Documents

- Sprint 7 — The rate-and-review sprint whose *intentional, not perfect* bar this sprint elevates to *finished*
- Sprint 6 — The sign-in + read foundation the whole consumer-app trilogy sits on
- Group Project Overview — Sprint schedule, ring topology, partner handoff timing
- Client Conversation — The product Buddy described in Week 1; this is the sprint where what you ship matches what was asked for
- TCSS460-frontend-2 lecture demo — Next.js 15 + Auth.js v5 + MUI v7 reference app; the theme + responsive patterns in this demo are the patterns your final polish leans on

Gen AI & Learning: AI in Group Projects

AI coding assistants are permitted and encouraged. The "every team member must be able to explain any code in your repository" bar is the standing norm and applies in full this sprint. During the Sprint 8 check-off, expect to be asked to walk through your theme configuration, your responsive breakpoints, your Lighthouse fixes, and the design vocabulary you settled on. "The agent wrote it" is not an explanation.

Polish work is also where the agent is most likely to produce code that *looks* like an improvement but isn't — a redesigned card that uses six hardcoded colors instead of theme tokens, a "responsive fix" that just hides content below a breakpoint, an a11y "fix" that adds ARIA attributes papering over a broken semantic structure. Read what the agent produced this week with the same care you'd read a teammate's pull request.

This assignment is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.