group-project sprint

Sprint 6 — Consumer App: Sign-In & Browse

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

Due Date

Sunday, May 24, 2026, 11:59 PM

Sprint Narrative

The swap happens this week. You're no longer the back-end team — you're the front-end team on top of an upstream partner's deployed API, the one they hardened for partner-ready handoff at the end of Week 6. Two firsts land together: your first contact with the partner whose contract you'll build against for three sprints, and the first real OAuth2 sign-in via NextAuth against the deployed Auth² instance. The product side stays narrow: the first read-only pages — search, browse, and a movie/show detail page — rendering against your partner's API. No ratings, no reviews, no writes yet; those arrive in Sprint 7. The bar this week is *I can sign in, find something on the partner's API, and read about it.*

MVP

By the end of this sprint, your team has made contact with your upstream partner group, holds its consumer-client credentials from the instructor, and can sign a visitor in through NextAuth against the deployed Auth² instance with an access token audience-scoped to your upstream partner's API. The deployed (or local) consumer app renders search results, a browse or popular page, and a movie/show detail page against your partner's deployed API. The session's access token decodes with the correct `aud` claim for your partner's audience. No writes ship this sprint — rating and reviewing arrive in Sprint 7.

Course Learning Objectives

This sprint contributes to the following course learning objectives:

- **LO 3:** Implement authentication and authorization using token-based and federated identity patterns (OAuth2 + NextAuth against Auth²)
- **LO 4:** Build interactive front-end applications using a component-based framework (Next.js)
- **LO 7:** Collaborate in teams using version control workflows, sprint milestones, and code review (cross-group API handoff)

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — reading a partner's OpenAPI spec to learn route shapes you didn't design, debugging the OAuth2 round-trip when the `aud` claim or the redirect URI is wrong, and deciding which of your partner's routes to consume first
- **Communication/Self-Expression** — making first contact with your upstream partner group, requesting credentials cleanly from the instructor, and filing structured bug reports through your partner's Bug Tracker FE instead of pinging on Slack

---

The Swap

Sprint 6 is when the ring rotates. Your group has been the back-end team for the last six weeks; starting Tuesday of Week 8, you're the front-end team in a *different* company.

- **Your upstream partner** is **Group N−1** (the ring closes — Group 1's upstream is Group 9). Their deployed API is the one your consumer app talks to for the next three sprints.
- **Your downstream partner** (Group N+1) is the team that's building a consumer app on *your* API. They've been collecting bug reports through your Bug Tracker FE since last Sunday. Stay responsive on the triage side even though your team's focus this week is on the FE work.

Your group keeps your existing BE deployment running through the rest of the quarter — your downstream partner needs it live.

> **GitHub Classroom — Sprint 6 Group Repo**
>
> The Sprint 6 GitHub Classroom assignment is a **group** assignment, like every other sprint — one repository per team, not one per student. The starter repo is **empty** — your team drops your consumer app in fresh. This is a different repo from your BE repo (Sprints 1–4) and from your Sprint 5 Bug Tracker FE; the consumer app lives on its own through Sprints 6–8.
>
> **Accept the assignment:** Sprint 6 — Consumer App on GitHub Classroom
>
> One team member accepts first and creates the team; remaining members accept the same link and join the existing team.

User Stories

As a team, we want to make first contact with our upstream partner group so that we know which API our consumer app is built on.

How to find them: Canvas → People → Groups gives you the roster of your upstream group (Group N−1). For real-time contact, the course Discord's `#group-meet` channel under the **Project Groups** category is where cross-group introductions happen — drop a message tagging the partner group and set up a call or async thread.

Reach out to Group N−1 early this week. Collect, at minimum:

- Their **deployed API base URL** (their Render/Vercel/Railway URL)
- Their **OpenAPI spec URL** (typically `<base-url>/api-docs`)
- Their **Bug Tracker FE URL** (the public report form they deployed in Sprint 5)
- The **audience string** their `requireAuth` expects (typically `group-N-api`, where N is *their* group number)
- A pointer to their **partner-facing README**

Record all of it on your team repo — in the FE repo's `README.md`, as defaults in your `.env.example`, or both. This is the same handoff your downstream partner is doing with you right now for the API you shipped at the end of Week 6; you're on the other side this time.

> **Set Up Your Own Group Discord Server**
>
> For day-to-day intra-group coordination over the rest of the quarter, spin up your team's **own** Discord server and invite the instructor in. Keeping ongoing sprint chatter off the main course server gives your team a private space for stand-ups, working discussions, and screen-shares — and keeps `#group-meet` usable for actual cross-group business.

As a team, we want to request our NextAuth consumer-client credentials from the instructor so that we can complete the OAuth2 flow against Auth².

> ⚠️ **Hard Deadline — Thursday, May 21, 11:59 PM (earlier is better)**
>
> This request must reach the instructor no later than **Thursday, May 21, at 11:59 PM**. Credentials are issued by hand; later requests may not be turned around before the Sunday due date, and without them your sign-in flow does not work and the rest of the sprint is blocked. Earlier in the week is strongly preferred — right after Tuesday's lecture is the ideal moment.

How to reach the instructor: Canvas message, Discord DM, or the course Discord's `#ask-charles` channel. Any of the three works — pick whichever you'd normally use.

Send your planned **deployed FE URL** (Vercel, Render, or whatever host you picked — if you're undecided, pick now). The instructor will add your callback path — `/api/auth/callback/tcss460` — to your group's pre-seeded consumer client in the `tcss460-sp26` tenant and return your `clientId` and `clientSecret`. Your client is already wired with an allowed audience pointing at your upstream partner's API; you do **not** need to ask for the audience to be configured.

Include `http://localhost:3000/api/auth/callback/tcss460` in the same request so local development works alongside the deployed URL.

As a visitor, I want to sign in to the consumer app so that I can access features that need an account.

Wire NextAuth (Auth.js v5) with Auth² as a custom OIDC provider. The values you need:

- **Issuer:** `https://tcss-460-iam.onrender.com` (no trailing slash — this is the single most common source of silent verification failures)
- **OAuth endpoints:** discovered automatically from the issuer's `.well-known/openid-configuration`; if you wire them manually, they live under `/v2/oauth/*`
- **Audience:** your upstream partner's audience string (from Story 1)
- **Client ID / secret:** from Story 2

A visitor clicks "Sign In", lands on Auth²'s hosted login page, signs in (or registers an account on the spot), and is redirected back into your app with a NextAuth session. That session holds the access token, the id token, and the basic profile (`sub` , `email` , `role`).

Guide

Authentication with NextAuth — Provider config, session shape, the `audience` parameter, callback URI gotchas, the `jwt` / `session` callbacks. The lecture demo for this guide is `TCSS460-frontend-2` — clone it and read alongside the guide; the same wiring drives your consumer app with two strings swapped (`AUTH_TCSS460_AUDIENCE` and `NEXT_PUBLIC_API_BASE_URL`).

As a user, I want to confirm my sign-in actually worked so that I can trust the rest of the app's auth-gated behavior.

Add a visible sign-in / sign-out surface — a header button, a `/profile` page, or both — that shows the signed-in user's email and a sign-out action. A signed-out visitor sees a "Sign In" affordance in the same spot.

As a one-time verification step, decode your session's access token at jwt.io and confirm:

- **`iss`** matches `https://tcss-460-iam.onrender.com`
- **`aud`** matches your upstream partner's audience string exactly

A wrong `aud` claim is the single most common Sprint 6 bug — your sign-in flow looks successful, NextAuth gives you a session, but every authenticated call to your partner's API returns 401. Catch it here, not in Sprint 7.

As a visitor, I want to search my partner team's API for movies and shows so that I can find something to read about.

Build a search page that calls your upstream partner's search route. Their route shape is theirs, not yours — read their OpenAPI spec to learn the query parameter names, the response field names, the result-envelope shape, and the pagination scheme they chose. Render results as a list or grid of cards using whatever fields their response actually gives you (title, year, poster, overview, etc.).

No auth is required here *unless* your partner made the route protected. If they did, the bearer token from your NextAuth session is what you attach; the guide and FE-2 both show the pattern.

Guide

Consuming a Web API — Fetch lifecycle, CORS preflight, error modes, reading a partner's spec to learn an unfamiliar contract

---

As a visitor, I want to browse what's popular or trending so that I have something to look at before I search.

Use your partner's browse / popular / trending route — whichever they chose to expose. This is your landing page or a dedicated `/browse` route. Same exercise as search: read their spec, render the shape they return. If they exposed several browse routes (popular *and* trending *and* top-rated), pick one for this sprint and light the rest up later; the goal is one working browse surface, not exhaustive coverage.

As a visitor, I want to open a movie or show detail page so that I can read about something before deciding whether to (eventually) rate it.

A detail page driven by your partner's detail route — typically keyed off a TMDB id you picked up from search or browse. Render the poster, synopsis, runtime, release date, and any enriched fields your partner shipped on their combined detail route from their Sprint 3 (community rating aggregate, recent reviews). What the page shows is bounded by what your partner's response actually contains.

Do **not** render write affordances yet — no rate button, no review form. Those wire up in Sprint 7. A "Sign in to rate" placeholder is fine if it helps the UI feel intentionally unfinished rather than broken.

As a frontend developer, I want to file structured bugs in my partner's Bug Tracker FE so that I have a real channel when their contract surprises me.

When your partner's API returns something their spec didn't promise, 500s on a request that looks valid, or rejects a token that decodes correctly, file a bug in **their** Bug Tracker FE — the URL you collected in Story 1. Title, description, which endpoint you hit, reproduction steps.

Don't message them on Slack. The course built this system for exactly this kind of cross-team contract friction; using it is the assignment. Your partner triages on their side via Postman or Prisma Studio, the same way you'll triage the bugs your downstream partner files against you.

Deliverables

- [ ] Upstream partner contact made; their API base URL, OpenAPI URL, Bug Tracker FE URL, audience string, and README link recorded in your team repo
- [ ] Consumer-client credentials request sent to the instructor by **Thursday, May 21, 11:59 PM** with your deployed FE URL and `localhost:3000` callback
- [ ] Sprint 6 GitHub Classroom group repository accepted; consumer app committed to it
- [ ] NextAuth configured with Auth² as an OIDC provider, audience-scoped to your upstream partner's API
- [ ] Visible sign-in / sign-out surface in the app
- [ ] Session access token decodes with `iss` and `aud` claims matching the configured values (verified at least once at jwt.io)
- [ ] Search page returning results from the partner's search route
- [ ] Browse / popular page returning results from one of the partner's browse routes
- [ ] Movie/show detail page rendering one of the partner's detail routes
- [ ] No write affordances rendered (no rate, no review, no profile mutations) — those are Sprint 7
- [ ] At least one structured bug filed in your partner's Bug Tracker FE *if* their API misbehaved (none required if their API worked cleanly)
- [ ] All team members have committed to the consumer-app repository
- [ ] 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 has its **own Sprint 6 GitHub Classroom group repository**, separate from both your BE repo and your Sprint 5 Bug Tracker FE repo.

Your BE Stays Live

Your group's deployed back-end continues running through the rest of the quarter — your downstream partner is consuming it for *their* consumer app over Sprints 6–8. If their bug reports arrive in your `/issues` queue, triage them via Postman or Prisma Studio against your admin-gated routes. Their development is gated on your responsiveness the same way yours is gated on your upstream partner's.

Guide Reference

Guide	What It Covers
Authentication with NextAuth	NextAuth (Auth.js v5) wiring, custom OIDC provider, audience parameter, session shape, redirect URI registration, common debugging modes
Next.js	App Router, routing, server vs client components, fetch, environment variables
React Fundamentals	Components, props, state, controlled inputs, rendering lists
Consuming a Web API	Fetch lifecycle, CORS preflight, error modes, reading a partner's OpenAPI spec
Accessibility	Form labels, focus management, semantic markup for sign-in flows
Styling & Component Libraries	MUI v7 patterns from the FE-2 lecture demo
Deploying Next.js	Vercel + Render walkthrough, env vars, build commands

---

Supporting Documents

- Sprint 5 — Your team's Bug Tracker FE (which your downstream partner is now using to file bugs against your API)
- Group Project Overview — Sprint schedule, ring topology, partner handoff timing
- Client Conversation — The product Buddy described in Week 1; this sprint starts realizing it on the front end
- TCSS460-frontend-2 lecture demo — Next.js 15 + Auth.js v5 + MUI v7 reference app; signs in against the deployed Auth² and consumes `backend-3`'s `/v2/messages` API. The shape your consumer app is reaching for this sprint.
- TCSS 460 Token Playground — Mint a token for your partner's audience out-of-band to verify their API accepts it before debugging through NextAuth
- Sprint 6 GitHub Classroom group repo — Accept here at the start of the sprint

---

Gen AI & Learning: AI in Group Projects

AI coding assistants are permitted and encouraged. The Sprint 5 pause on "every team member must be able to explain any code in your repository" is **over** — that bar is back to normal this sprint and stays normal through the rest of the quarter. The Week 6 Evolution of Web Programming reading, the Week 7 OAuth2 reading, the upcoming React/Next.js lectures, and Check-Off 6 give you the conceptual ground to actually understand what an agent produces this week.

During sprint reviews, expect to be asked to walk through your NextAuth config, your provider params (especially `audience`), and your authenticated-fetch pattern. "The agent wrote it" is not an explanation. If you don't yet understand a piece an agent wrote, the right move is to pause, ask the agent to walk you through it, cross-check against the NextAuth guide and FE-2, and then keep going. The token you spend understanding once saves you a sprint of debugging later.

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