Group Project: Movie and TV Review Platform

The group project is a full-stack web application built over the entire quarter. It accounts for 50% of your course grade. All sprints are due Sunday at the end of the sprint week.

Stack: PostgreSQL → Express/TypeScript API → Next.js Front-End

Data Source: TMDB (The Movie Database) — all groups proxy both movie and TV data.

Auth: Shared instructor-hosted authentication service (OAuth2).

Start Here

Before Sprint 0, before technical specifications, before you write code: **understand what you're building and why**.

A client came to us with a problem. A technical lead explained the solution. Read their conversation. It's the foundation for all ten weeks of sprints.

What This Project Is

You're building a **full-stack web application** for discovering and reviewing movies and TV shows. Your group does this in two phases across the quarter:

- **Sprints 1–4 (Weeks 3–6): Back-End phase.** Your group is the back-end team at a fictional company. You build a REST API (Express, TypeScript, PostgreSQL, Prisma) serving movie/TV data with user ratings and reviews.
- **Sprints 5–8 (Weeks 7–10): Front-End phase.** Your group becomes the front-end team at a *different* fictional company. You build a Next.js web app on top of **another group's** API — the one they built during their back-end phase, handed off to you as a live deployment plus OpenAPI docs (not source code).

With 9 groups in the course, the back-end APIs form a ring. Your API will be consumed by one downstream partner group's front-end. You in turn consume one upstream partner group's API for your own front-end work. Every group is in two companies across the quarter — one as the back-end team, one as the front-end team.

This mirrors how real software companies work. Your API design has to be good: someone else is reading your docs and building against them. Bug reports from your downstream

partner arrive through a real bug-report API, not Slack. You experience both sides of the contract in a single quarter.

The Core Architecture

During Your Back-End Phase (Sprints 1–4)

- A web API serving movie and TV data
- Your own PostgreSQL database for user-generated content (ratings, reviews)
- A proxy to The Movie Database (TMDB) API
- Authentication wired to the shared login service (provided by instructor)
- Bug-report API endpoints (`/issues`) that your downstream partner's front-end team will consume during their Sprint 5

During Your Front-End Phase (Sprints 5–8)

- A **Bug Tracker front-end** (Sprint 5) on top of **your own** API's `POST /issues` endpoint — a public report form, **no authentication**. Your *downstream* partner group uses it to file bugs against your API while they build their consumer app. Triage happens BE-side (Postman against your admin-protected `/issues` routes, or Prisma Studio on the `issues` table). Intentionally scoped small so Sprint 5 can focus on Next.js fundamentals.
- A **movies/TV consumer app** (Sprints 6–8) on top of your **upstream partner's** API — this is where you first meet NextAuth + OAuth2, and where you experience consuming an API you didn't design.

What You Don't Build

- Your own movie/TV metadata database (TMDB handles that)
- Your own authentication system (shared service handles that)

The Key Constraint

Your database only stores **user-generated content**: who rated what, what reviews people wrote, who has accounts. TMDB is the source of truth for media metadata.

Key Principles

API Design is a Conversation — The backend team writes the API spec *before* shipping code. The frontend team reads it and gives feedback. The spec is a contract.

Divergence is Intentional — Your team owns all API design choices. How you structure search, pagination, filtering, ratings, reviews, user profiles. Route names, response shapes, endpoint design. Different is not wrong. When frontend teams integrate with APIs from other

groups, they learn the real-world skill of reading documentation, understanding design trade-offs, and adapting to divergent systems.

Bug Tracking is Structured — Not Slack, not email. When frontend teams hit bugs, they file issues with title, description, reproduction steps. You see them on a dashboard. Real workflow.

Deployment is Early — Your API ships to production by Week 5. Your Bug Tracker front-end goes live in Week 7. Your consumer app ships in Week 10. Users can actually use your code.

Sprint Schedule

Sprint	Week	Due	Phase	Focus
Sprint 0	2	Sun Apr 12	Setup	Form groups, clone repos, deploy
Sprint 1	3	Sun Apr 19	Back-End	Express server, TMDB proxy routes, OpenAPI docs
Sprint 2	4	Sun Apr 26	Back-End	Prisma, PostgreSQL, user provisioning, ratings CRUD
Sprint 3	5	Sun May 3	Back-End	Full route coverage, RBAC, combined endpoints
Sprint 4	6	Sun May 10	Back-End	Deployed, documented, tested
Sprint 5	7	Sun May 17	Front-End	Bug Tracker FE on **own** API's `/issues`
Sprint 6	8	Sun May 24	Front-End	Consumer app: NextAuth OAuth2 + initial pages on **partner's** API
Sprint 7	9	Sun May 31	Front-End	Consumer app: authenticated routes, partner API integration

Sprint	Week	Due	Phase	Focus
Sprint 8	10	Sun Jun 7	Front-End	Deployed, responsive, complete

Cross-Group API Swap

In the front-end phase (Sprints 5–8, Weeks 7–10), every group builds **two** separate front-end deployments:

- **Sprint 5 (Week 7) — Bug Tracker FE on your own API.** Small, single-sprint FE. Public report form only, no auth. You already know this API inside-out because you built it. Good ramp into Next.js fundamentals without auth complexity.
- **Sprints 6–8 (Weeks 8–10) — Consumer app on your upstream partner's API.** The main front-end project. NextAuth + OAuth2 arrive here. You don't control the API — you read their OpenAPI docs, build against the contract, and file issues through *their* Bug Tracker FE when something doesn't work.

With 9 groups arranged in a ring, every back-end has exactly one consumer-app consumer and every consumer app has exactly one back-end provider. 1:1, not two APIs per consumer app.

Next Steps

1. **Read the full client conversation** — This is the problem statement and the high-level solution architecture.
2. **Form your group** — 4–5 people who stay together the whole quarter. Your group builds the back-end (Sprints 1–4), then pivots to building a front-end on a partner group's API (Sprints 5–8).
3. **Get your starter repos** — Your instructor will provide the back-end template at the start of Sprint 1 and the front-end template at the start of Sprint 5.
4. **Sprint 0 begins** — You'll get specific sprint requirements each week.

Documents

- **Full Client Conversation** — Read this first. It's the requirements conversation between the client and the technical lead.
- Sprint 0
- Sprint 1

- Sprint 2
- Sprint 3
- Sprint 4
- Sprint 5
- Sprint 6
- Sprint 7
- Sprint 8