group-project # sprint

Sprint 1 — TMDB Proxy

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

Due Date

Sunday, April 19, 2026, 11:59 PM

Sprint Narrative

Your API is deployed and breathing — the heartbeat proves it. But right now it's an empty shell. The client's first priority is content: users need to search for movies and TV shows, browse what's popular, and pull up details on anything that catches their eye. All of that data lives in TMDB — your job is to proxy it through your own API so the frontend never touches a third-party key.

And here's the critical part: **your API is not a passthrough.** Every response must be transformed into your own schema and documented in your OpenAPI spec. A frontend developer reading your `/api-docs` should have no idea TMDB exists behind the scenes. You decide what fields to include, what to rename, what to leave out. Raw TMDB JSON coming out of your API is not acceptable — you are building *your* API, not a mirror of someone else's.

MVP

By the end of this sprint, your API proxies TMDB data for both movies and TV shows — search, details, and popular browsing — all through endpoints your team designed. Every response is transformed into your own documented schema (not raw TMDB JSON). Your OpenAPI spec covers all new endpoints, your test suite verifies them, and your TMDB API key is nowhere in your source code.

Course Learning Objectives

This sprint contributes to the following course learning objectives:

- **LO 1:** Design and implement RESTful web APIs using a server-side framework
- **LO 5:** Deploy full-stack applications to cloud infrastructure
- **LO 7:** Collaborate in teams using version control workflows, sprint milestones, and code review

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — designing API response schemas and making data transformation decisions

---

🔧 Project Setup

Update Your Code Quality Tools

Your project's ESLint configuration has been updated with stricter rules that catch real bugs. One team member should run through these steps, then open a PR so the rest of the team gets the changes.

1. Open a terminal in the root of your project folder

This is the folder that contains your `package.json`.

2. Install the new package

```
npm install -D eslint-config-prettier@^10.1.0
```

3. Download the updated config files

```
curl -O https://raw.githubusercontent.com/UWT-SET-TCSS460-LECTURE-MATERIALS/TCSS460-group-project-backend/main/eslint.config.mjs
curl -O https://raw.githubusercontent.com/UWT-SET-TCSS460-LECTURE-MATERIALS/TCSS460-group-project-backend/main/tsconfig.eslint.json
```

4. Add the `lint:fix` script to `package.json`

```
npm pkg set scripts.lint:fix="eslint --fix ."
```

5. Verify everything works

```
npm run lint
```

What changed:

Rule	What It Does
`eqeqeq`	Requires `===` instead of `==`
`prefer-const`	Flags `let` when the variable is never reassigned
`no-console`	Warns on `console.log` — suppress intentional ones with `// eslint-disable-next-line no-console`
`no-floating-promises`	Catches `async` calls missing `await`

Rule	What It Does
`eslint-config-prettier`	Prevents ESLint/Prettier rule conflicts

Guide

Code Quality Tools — Full walkthrough of ESLint and Prettier configuration

---

User Stories

As a team, we want to design our API routes and response shapes before writing code so that we have a shared plan to build against.

Before you start coding, sit down as a team and make your design decisions. Start by exploring the TMDB API documentation — understand what endpoints are available, what data they return, and what parameters they accept. Then make your design decisions: What routes will you create? What will you name them? What fields will each response include? How will you transform TMDB's data into your own schema? Document these decisions — your OpenAPI spec is a good place for this. Teams that skip this step end up with inconsistent endpoints and wasted merge conflicts.

As a frontend developer, I want to search for movies so that I can display search results with poster images and key metadata.

Your API proxies this to TMDB — the frontend developer never needs a TMDB key. How you structure the route, query parameters, and response shape is your team's design decision. At minimum, support searching by title. TMDB also supports filtering by year, genre, and other criteria — your team decides which filters to expose. At minimum, your response must include enough data to render a card with a poster image, title, and release year. Beyond that, your team decides what to include. TMDB returns a lot of data per result — part of the design work is deciding what's worth surfacing in a search result versus a details view. The response must be your own schema — not raw TMDB JSON.

Guide

The Proxy Pattern — How and why to wrap third-party APIs with your own endpoints

As a frontend developer, I want to search for TV shows so that I can display search results with poster images and key metadata.

Your API proxies this to TMDB — the frontend developer never needs a TMDB key. At minimum, support searching by title. TMDB also supports filtering by year, genre, and other criteria — your team decides which filters to expose. Your team decides whether movies and TV search share a response shape or use distinct ones. At minimum, your response must include enough data to render a card with a poster image, title, and first air date. The response must be your own schema — not raw TMDB JSON.

As a frontend developer, I want to retrieve details about a specific movie so that I can display a rich content page with poster art, metadata, and synopsis.

Given a movie identifier, return enough data to build a full detail page — at minimum a poster image, synopsis, and release year. TMDB offers much more: cast, crew, genres, runtime, trailers, similar titles, production companies. Your team decides what's worth including. What fields you surface and how you name them is your design decision. The response must be your own schema.

As a frontend developer, I want to retrieve details about a specific TV show so that I can display a rich content page with poster art, metadata, and synopsis.

Given a TV show identifier, return enough data to build a full detail page — at minimum a poster image, synopsis, and first air date. TV shows have different metadata than movies — seasons, episode counts, networks instead of studios, ongoing vs. ended status. Your team decides how to handle these differences and what to include. The response must be your own schema.

As a frontend developer, I want to retrieve popular movies so that I can feature trending content for users.

Your team decides how to define "popular" and what data to return. The response must be your own schema.

As a frontend developer, I want to retrieve popular TV shows so that I can feature trending content for users.

Your team decides how to define "popular" and what data to return. The response must be your own schema.

As a frontend developer, I want OpenAPI documentation for every endpoint so that I can integrate without reading source code.

Your `/api-docs` page documents every route: parameters, response shapes, and example responses. A developer on another team should be able to build against your API using only this page.

> **Guide**
>
> OpenAPI Documentation — How to write and maintain your OpenAPI spec with Scalar

---

As a frontend developer, I want automated tests for every endpoint so that I can trust the API behaves as documented.

Every endpoint has tests using Jest and Supertest. Tests verify both success responses and error handling. Your test suite should run with `npm test` and pass cleanly.

> **Guide**
>
> API Testing — Writing automated tests with Jest and Supertest

---

As a team, we want our TMDB API key stored securely so that it's never exposed in source code or client responses.

Register for your own TMDB API key at themoviedb.org. Store it as an environment variable. It must never appear in a commit, a response body, or your OpenAPI docs.

Deliverables

- API serves movie search, movie details, and popular movies through your own endpoints
- API serves TV show search, TV show details, and popular TV shows through your own endpoints

- [ ] All responses are transformed into your own schema (no raw TMDB JSON)
- [ ] OpenAPI documentation at `/api-docs` covers all new endpoints with response shapes
- [ ] Automated test suite covers all endpoints (success and error cases)
- [ ] TMDB API key is not committed to the repository
- [ ] API is deployed and responds at your public URL
- [ ] All team members have committed to the 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.

Guide Reference

Guide	What It Covers
The Proxy Pattern	Wrapping third-party APIs, response transformation, error handling
API Testing	Writing automated tests with Jest and Supertest
OpenAPI Documentation	Writing and maintaining your OpenAPI spec with Scalar
Intro to Express	Express fundamentals, routing, request/response
Routing and Middleware	Router organization, middleware patterns
Deploying a Simple Web API	Deployment to Render or Heroku

Supporting Documents

- Client Conversation — Original requirements discussion between the client and technical lead

- Group Project Overview — Sprint schedule, architecture summary, and key principles
- TMDB API Documentation — Official TMDB API reference

Gen AI & Learning: AI in Group Projects

AI coding assistants are permitted and encouraged. Every team member must be able to explain any code in your repository. During sprint reviews, you may be asked to walk through specific sections. "The AI wrote it" is not an explanation — understanding is the requirement.

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