group-project # sprint

Sprint 0 — Setup

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

Due Date

Sunday, April 12, 2026, 11:59 PM

Sprint Narrative

You've read the client conversation. You know what you're building and why. But before you write a single line of application code, your team needs to get its house in order. This sprint is about people and process: meet your team, agree on how you'll work together, get everyone contributing to the same repository, and prove you can deploy. The client doesn't care about your git workflow — but your team will fall apart without one.

MVP

By the end of this sprint, every team member has cloned the repository, created a branch, opened a pull request, and merged code into `main`. The starter example code has been removed and replaced with a documented heartbeat route. The API is deployed to a public URL with `/api-docs` accessible. Your team has met, established working agreements, and documented the meeting.

Course Learning Objectives

This sprint contributes to the following course learning objectives:

- **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:

- **Communication/Self-Expression** — establishing working agreements, documenting meetings, and coordinating as a team

---

User Stories

As a team, we want to hold an introduction meeting so that we establish working agreements and communication norms before writing code.

This meeting is not optional and should not be rushed — expect it to take about an hour. Use the meeting template linked below as your agenda. Every team member must attend.

After the meeting, create a file called `meetings.md` in the root of your repository (see next story). Document this meeting there — and use this file to log all future meetings and ceremonies throughout the quarter.

Meeting Template

Copy the contents of this template into your meeting document:

First Meeting Template

Finding a Meeting Time

When2Meet is a helpful tool for finding a time that works for everyone.

As a team, we want a shared repository so that all members can contribute code.

Accept the GitHub Classroom assignment to create your group's repository. Every team member clones it locally and verifies they can run the starter project.

GitHub Classroom

Accept the assignment

As a teammate, I want to create my own branch and add a route to the API so that I practice the team's git workflow.

Every team member — individually — does the following:

1. Create a new branch off `main` (e.g., `setup/your-name`)
2. Add a new route to the API (e.g., `/hello/your-name` that returns a JSON greeting)
3. Commit your changes and push your branch
4. Open a pull request to `main`

The route itself is trivial — the workflow is the point. These routes will be deleted before the final Sprint 0 deliverable (along with the starter `/hello` route). After this sprint, every member should be comfortable with branching, committing, pushing, and opening PRs.

> **Intentional Conflicts Ahead**
>
> Multiple team members will be editing the same files — route registrations, for example. This **will** create merge conflicts. That's the point. See the next story.

As a team, we want to review and merge each member's pull request so that we learn to manage merge conflicts.

Do not merge all PRs at once. Merge them one at a time. After each merge, the remaining PRs will likely have conflicts with `main`. The author of each conflicting PR is responsible for:

1. Pulling the latest `main` into their branch
2. Resolving the conflicts
3. Pushing the updated branch

Review each other's PRs before merging. This is practice for the real thing — in later sprints, PRs will contain meaningful code and your team will depend on this process working smoothly.

> **Tip**
>
> Agree as a team on a merge order. The first PR merges cleanly. Every PR after that is a lesson in conflict resolution.

As a frontend developer, I want a heartbeat endpoint so that I can verify the API is running and healthy.

A **heartbeat route** (sometimes called a health check) is a simple endpoint that returns a fixed response confirming the API is alive. It takes no parameters, hits no database, and requires no authentication. It exists for one reason: so that anyone — a frontend developer, a monitoring tool, a load balancer, or your instructor — can answer the question "is this thing on?" in one request.

In production, every web service has one. Cloud platforms use them to know when to restart a crashed server. Frontend teams hit them before filing a bug ("is the API down, or is my code broken?"). They're the first thing you check during an incident.

Create a public heartbeat route — common names include `GET /health`, `GET /heartbeat`, or `GET /status` — that returns a JSON response indicating the API is alive. Document it in your OpenAPI spec — your `/api-docs` page (served by Scalar) should be publicly accessible and show this endpoint. Your individual `/hello/your-name` routes do not need OpenAPI documentation — they're workflow practice, not part of your API contract.

While you're in `openapi.yaml`, update the `info` section to reflect your team. Replace the placeholder title, description, and team name with your own. Add your deployed URL to the `servers` list. When another team opens your `/api-docs` page, they should immediately know whose API this is and where it's hosted.

Guide

OpenAPI Documentation — How to write and maintain your OpenAPI spec

---

As a team, we want to remove the example `/hello` route so that our API only contains code we own.

The starter repo ships with a `GET /hello` route, its test, and its OpenAPI documentation. These exist so you can verify the project runs — once you've built your heartbeat route, delete them. Also remove each team member's `/hello/your-name` practice routes. They served their purpose (learning the git workflow) and don't belong in your API going forward. Your final `main` branch should only contain the heartbeat route and its documentation.

As a frontend developer, I want the API deployed to a public URL so that I can verify it responds.

Deploy your API to any cloud platform. The deployed version should reflect your final `main` branch — the heartbeat route responding and `/api-docs` accessible.

Your team chooses the platform. Free tiers are available on Render, Railway, Fly.io, and Heroku (via GitHub Student Developer Pack).

Guide

Deploying a Simple Web API — Step-by-step deployment to Render or Heroku

Think Ahead: Database Hosting

In a few sprints, your API will need a PostgreSQL database. When researching platforms now, consider whether your chosen provider also offers a free-tier database — or whether you'll need a separate service for that. Some groups in the past have used Supabase for their database deployment. You don't need a database yet, but choosing a platform that supports one later will save you a migration.

Add the deployed URL in two places:

1. **Repository README** — include the live URL so anyone reading your repo can find it
2. **GitHub repository website link** — the "About" section of your GitHub repo has a Website field. Put your deployed URL there.

Deliverables

- All team members have merged at least one PR into `main`
- `meetings.md` exists in the repo with your first meeting documented
- Heartbeat route returns a JSON response
- `/api-docs` (Scalar) is publicly accessible and documents the heartbeat endpoint
- OpenAPI `info` section customized with team name, description, and deployed server URL
- Example `/hello` route, test, and OpenAPI entry removed
- All practice `/hello/your-name` routes removed
- API is deployed and responds at a public URL
- Deployed URL is in the repo README and GitHub About section

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
OpenAPI Documentation	Writing and maintaining your OpenAPI spec with Scalar
Deploying a Simple Web API	Step-by-step deployment to Render or Heroku
Client Conversation	The requirements conversation that frames the entire project
Group Project Overview	Sprint schedule, architecture summary, and key principles

---

Supporting Documents

- First Meeting Template — Agenda for your introduction meeting
- Client Conversation — Original requirements discussion between the client and technical lead

---

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.