Table 1: Revision History

Date	**Developer(s)**	**Change**
September 23, 2022	Arkin Modi	Create project scheduling section
September 23, 2022	Arkin Modi	Create workflow plan
September 23, 2022	Arkin Modi	Create technology and code standard sections
September 25, 2022	Timothy Choy	Create proof of concept demo plan

Development Plan
Software Engineering

Team 3, Tiny Coders
Arkin Modi
Joy Xiao
Leon So
Timothy Choy

[Put your introductory blurb here. —SS]

1 Team Meeting Plan

2 Team Communication Plan

3 Team Member Roles

4 Workflow Plan

4.1 Development Workflow

The team will be using Git as their version control system and the repository will be hosted publicly on GitHub. The most stable up-to-date version of our application will be on the "main" branch. The team will update the main branch using a feature branch workflow. The feature branch workflow entails the developer to:

1. Create a new feature branch
2. Push changes to the new branch
3. Open a pull request against the main branch
4. Link the pull request to an issue (if appropriate)
5. Add "needs review" label (if ready for review)
6. Assign assignees and reviewers
7. Pass any pull request pipelines in our CI/CD setup
8. Receive reviews from other team members
9. Address any comments or change requests
10. Merge or close the pull request

To prevent naming conflicts, the team will follow a branch naming convention of "<GitHub username>/<branch name>". The merge strategy used for pull requests will be "squash and merge". Using this merge strategy increases traceability on the main branch's commit history by having every commit represented by a working/passing pull request and prevents adding broken commits to the main branch's commit history.

All team members are listed as code owners of everything in the repository (defined in .github/CODEOWNERS). GitHub will read this and automatically assigned all team members to every pull request opened in the repository. Every pull request should receive, at minimum, one review before merging.

4.2 Project Management Workflow

The team will be using GitHub's issue tracker and GitHub Projects to manage the project. The issue tracker will track work that needs to be done. For each task that isn't trivial, an issue will be open to track the contribution and progress. Examples of trivial tasks include spelling corrections, dependency version updates, base repository template updates. Each issue will also be assigned to team member who will then assume responsibility for said issue.

Labels, as described in Table 2, will used to organize issues. The GitHub Project board will be used to visualize the progress of each issue and track overall progress.

Table 2: Issue Labels

Name	**Description**
documentation	These issues are related to documentation or reports for course deliverables.
Epic	These issues are tracking a collection of other issues. For example, an issues a report has sub-issues for each section.
meeting needed	A team meeting is needed to discuss details related to the issue.
meeting notes	This issue contains notes from a completed meeting.
needs breakdown	This issue needs further breakdown into smaller issues/steps.
tech foundation	This issue is not contributing towards any course deliverables.

5 Proof of Concept Demonstration Plan

There are several risks that will determine the success or failure of the project. The main risks are:

- Communication between the modules of the PWA
- Responsiveness of the PWA to different devices and screen resolutions
- Peering problems between multiple editors (e.g., race conditions)
- Integration testing
- It is our team's first time working on a PWA so there may be unfamiliarities

To prove that we will be able to overcome these risks and have a successful project, we will be showing several aspects in the proof of concept demo:

- Develop two modules of the PWA to show communication between the modules in the application works
- Demo the application on several devices (e.g., an iPhone, an Android, and on a computer) to show that the application is responsive
- Demo an example of two devices attempting to schedule an appointment at the same time, and see the result

6 Technology

The project will written in TypeScript using the Next.js framework. Next.js is a React framework that provides both a frontend and a backend interface. The team is comfortable with React and with both the core Next.js team and the core React team recommending Next.js for "production-ready projects", we see this as a good choice. We will be using NPM as our package manager and the packages next-auth and next-pwa. NextAuth is a package built for Next.js to quickly enable authentication with many out of the box adaptors for providers and databases. Next-PWA is a package to quickly enable the creating of Progressive Web App (PWA)'s in Next.js. We have decided to create a PWA since this was one of the key application attributes that the supervisor required.

For our user interface, we have chose to use Blueprint JS as our component library along side CSS. A team member has previous experience with this library and recommended using it.

We will be using Prisma as our ORM (Object-relational mapping) and MySQL as our database management system. We have decided to use a SQL database due to our data having relations. MySQL was specifically chosen due to past experience within the team. In a NoSQL database, relations are harder to maintain and/or possibly inefficient to query. SQL databases also provide the assurances of ACID compliance, fewer consistency issues, and enforced field constraints. The use of Prisma is due to past experience within the team and a strong reputation within the web development world.

Jest and React Testing Library will be used for our unit tests, integration tests, and code coverage. Most of the team does not have much experience with frontend testing, therefore we have decided to go with the libraries recommended by Next.js. We have decided against using any end-to-end testing frameworks (e.g., Cypress, Playwright) due to lack of domain knowledge within the team and time constraints.

Git and GitHub will be used for version control and hosting a centralized online repository. This was chosen due to course requirements and past experience.

Documentation will be written in a mix of LaTeX and Markdown.

With high performance not being a primary goal with this project, we will not use using any performance benchmarking tools. We will be maintaining a Google Lighthouse score of greater than 90%.

All CI/CD pipelines will be created using GitHub Actions. This was chosen due to GitHub Actions having great integration with GitHub, a generous free and pro tier, past experience within the team, and ease of use.

The development environment for any developer is a highly personal decision and while we will not be enforcing the use any specific local dev tooling, we will be recommending to all developers to use Visual Studio Code for easy interoperability between developers. We will also have a list of recommended extensions and setting for Visual Studio Code available in the .vscode folder. We will also be using Docker for easy setup of external dependencies (e.g., TeX Live, MySQL).

7 Coding Standard

To ensure consistent code styling and code quality, the application code will be formatted using Prettier. Prettier is an industry standard code formatter and widely agreed upon default configuration.

For our LaTeX docs, we will be using latexindent.pl. latexindent.pl is a popular LaTeX code formatter. The primary use will be for managing our whitespace to ensure consistent readability in our source files. The primary reason behind this choice was latexindent.pl's code formatter capabilities to automatically apply fixes.

8 Project Scheduling

The project schedule will primarily follow the course calendar with development of the application starting in October 2022. Progress of the project will be tracked using a GitHub Project board. The deliverables due date will act as our milestones.

Decomposition of large tasks into smaller tasks for documentation deliverables will done by breaking down the reports by section. Sections will be determined by consulting provided templates, rubrics, and examples. Breaking large development tasks into smaller tasks will be done by first researching and scoping out how the task should be architected. Following this initial architecture, smaller tasks will be created.

Responsibility/ownership of tasks will be decided as a team during weekly meetings or offline communications. It is upon each member to communicate their availability and capable workload. Ideally, the work shall be equally distributed over the course of the project, but a shift in workloads on an individual deliverable basis is expected. If a member is unable to complete a task they are assigned, it is their responsibility to communicate this to the team and have the task reassigned. The expectation is each member shall contribute an average of 10 hours per week.