# Final Report

# Introductions

We are Team Mashed. We love potatoes so we called ourselves Mashed.

*At Team Mashed, we really like our potatoes. And as our very favorite is mashed potatoes, we strive to deliver a pleasant, smooth experience with our products. We are a blend of heterogeneous personalities with different skill sets: each one of us plays a role, but when we come together, the miracles happen.*

## Our team members

**Alberto Nencioni:** product manager.
**Edward Chen:** UI/UX designer, software developer.
**Chang Zhou, Dian Yu, Alexandra Macedo, Jiayou Guo:** software developer.

Alberto Nencioni organizes the team meeting and reports our status. He also manages documentation overall.
Edward Chen designed the landing page and participated in coding chat room functions.
Chang Zhou participated in coding user login, user list and search functionalities. He also designed the JSDoc deployment pipeline.
Dian Yu participated in coding user login, user list and search functionalities. He also helped with code comments/JSDoc and JSDoc pipeline.
Alexandra Macedo participated in coding chat room functions and helped with various documents. She also helped in designing the linter pipeline.
Jiayou Guo participated in coding user login, user list and search functionalities. He also refactored code structures for good maintainability score on code climate. He set up our Cypress end-to-end test and Code coverage pipelines.

# Project

## Problem Statement

In CSE 110, software engineering is not taught or executed as a repeatable process and operates more like an unconstrained, unpredictable 10 week hackathon. We also see a general lack of systematism in software engineering processes in industry. In order to address this, we would like to teach CS undergraduates about process practices using a project that enables a more repeatable teaching process and more predictable, constrained project experience.

## Brainstorming and Requirements Planning

Find our brainstorming and pitch documents at the respective links.

We first assessed our perceived constraints by assessing the expected amount of time each member was willing to put in, as well as the current technical background of each member. From this, we estimated our perceived timeline for a project, as well as the possible technologies that were reasonable for us to consider. From here, we brainstormed many project ideas. For each potential project, we weighed the benefits and risks by taking into account our time constraints, technological backgrounds, level of interest, and how well the idea conformed to the problem that the project is intended to solve. We also tried to determine the motivation behind our choices in technologies and project ideas. We would later end up meeting with our client (the professor), which resulted in us tossing out our more unconstrained project ideas. Once we finally settled on our project idea and had a more tempered mindset, we tossed out a couple pieces of our technology stack to make things simpler. For the remaining parts of our technology stack that might present some risk, we had group discussions to ensure that we understood the potential risks we were signing up for and agreed that they were risks worth taking.

## User-centered Thinking

Regarding users of our produced application, our User Stories and Use Cases can be found at the respective links.

One of the meta-users of our project was the professor. One of the most important things that we did early on, but should have done even earlier, was meet with our client (the professor), who is also one of our users. As described above, we initially not only had many ambitious project ideas that didn't necessarily fit the problem description (our users' needs) well, but from this, we also learned that we were not approaching our initial design process in a user-centered way. We were thinking about the product before the user. By taking this first step to meet with our client/user, we changed our mindset and motivations for the future decisions that we made. For example, while our project may be slightly more complex than others, we threw out riskier project ideas and scaled back our technology stack. In addition, as we realized that our project might not be ideal to start from scratch, we pitched our project as a starter-kit from undergraduates to build off of because we thought that was a more reasonable use case.

As CSE 110 students are also our meta-users, we felt this idea of a starter-kit would satisfy them on a couple accounts. In industry, it's often the case that you begin your development work in an existing code base instead of at a project's inception. What this means is that you will have to navigate a foreign code base and add prescribed features and tests to an existing or in-development project. It is important for students both to (1) learn good software engineering process practices, and (2) be prepared for this reality. With our project's structure, students would get to experience this real world scenario by having to navigate our documentation, code base, and setup instructions to develop on top of an existing project. In addition, the fact that we provide some artifacts, workflow files, and tests means that they would have examples to not only use as templates for their own development, but also improve upon and refactor. There are certainly areas where we could have done better, and in industry, you're not dropped into a perfect setting. We also have an extensive feature backlog that students can pull from. It's even

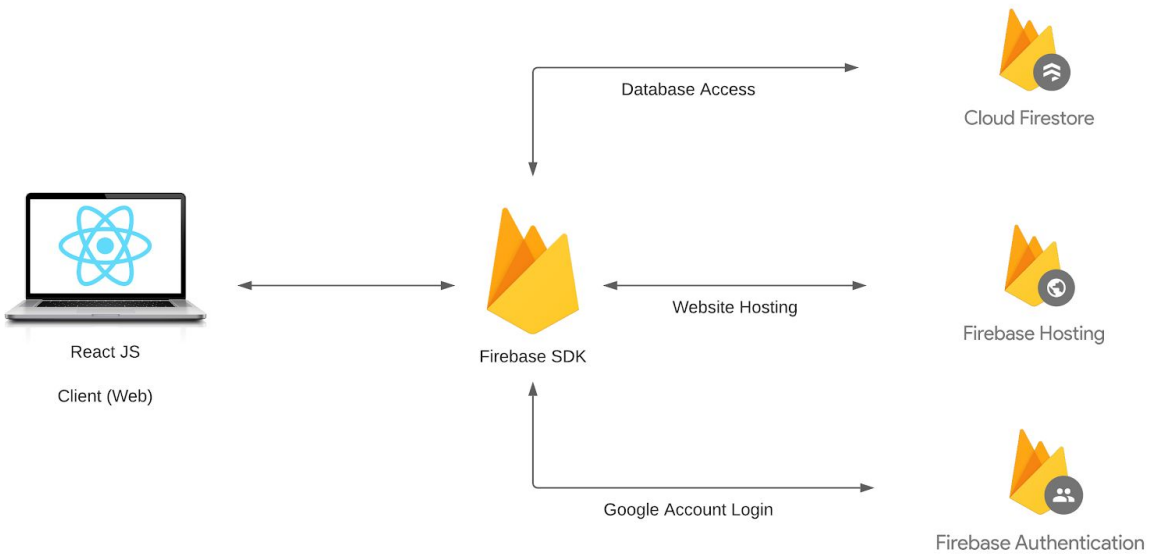
possible that different teams implement different features, which would prevent cheating within one iteration of the class, which also addresses one of the professor's needs. The features in the backlog are also interesting enough that the students would feel motivated to work, but comparatively less complicated than the work done to get our project to the state that it is currently at. Additionally, especially when generating our README, we tried to be very explicit about describing how one would extend our project and providing helpful links for any setup that they would need to do so that the process of extending our project would hopefully be derisked, which is also one of the professor's needs.

## Design up front

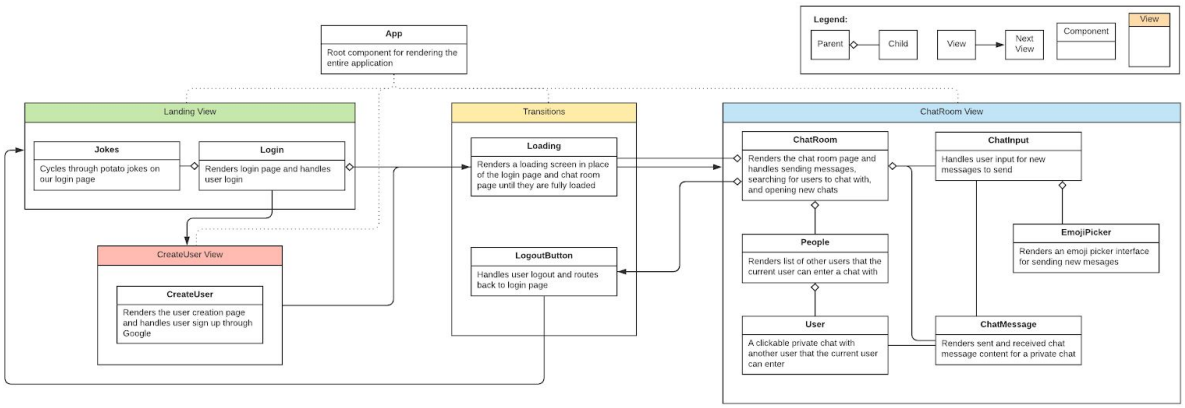
Our Wireframes and Screen Sequence Diagrams can be found at the respective links.
We think this amount of detail was sufficient for developers to start working independently in parallel, but not so much that the developers didn't have flexibility in implementation. Looking back, we feel we were correct. As we were working with new technologies, we didn't have a crystal clear idea of some constraints and required implementation details until we started coding. In reflecting, we feel that we specified the right amount of up front design given our knowledge of the requirements in the beginning.

## Design Documentation

Our architecture consisted of React on the frontend and Firebase for our database and hosting. Here is our architecture diagram:



Here is a diagram of the React components of our application and the relationships between the different views (for better viewing, see the file hosted here):




The React components in the above diagram represent the core elements of our application. We had three main views: a login page, a user creation page (signing up with Google), and a chat room page. All of the components are grouped under the view that they contribute to rendering, descriptions of their purpose are provided, and parent-child relationships are represented.

In addition, we have a lower level traditional UML diagram detailing some of the implementation of each of the components.

# Process

## Stand-Ups

We do not have daily stand-ups because this course is not our full-time job. We are all pretty busy and cannot guarantee that we make progress every day, so the daily stand-up didn't make too much sense to us. We chose to do an alternative meeting that lasts for at least one hour every other day (every Tuesday, Thursday, and Sunday). We picked these dates because all members of the team are available and we could still make sure progress was being made in a continuous manner. To guide our standups, we referred to the pending and completed tasks on our Trello sprint board.

## Sprint Planning and Estimation

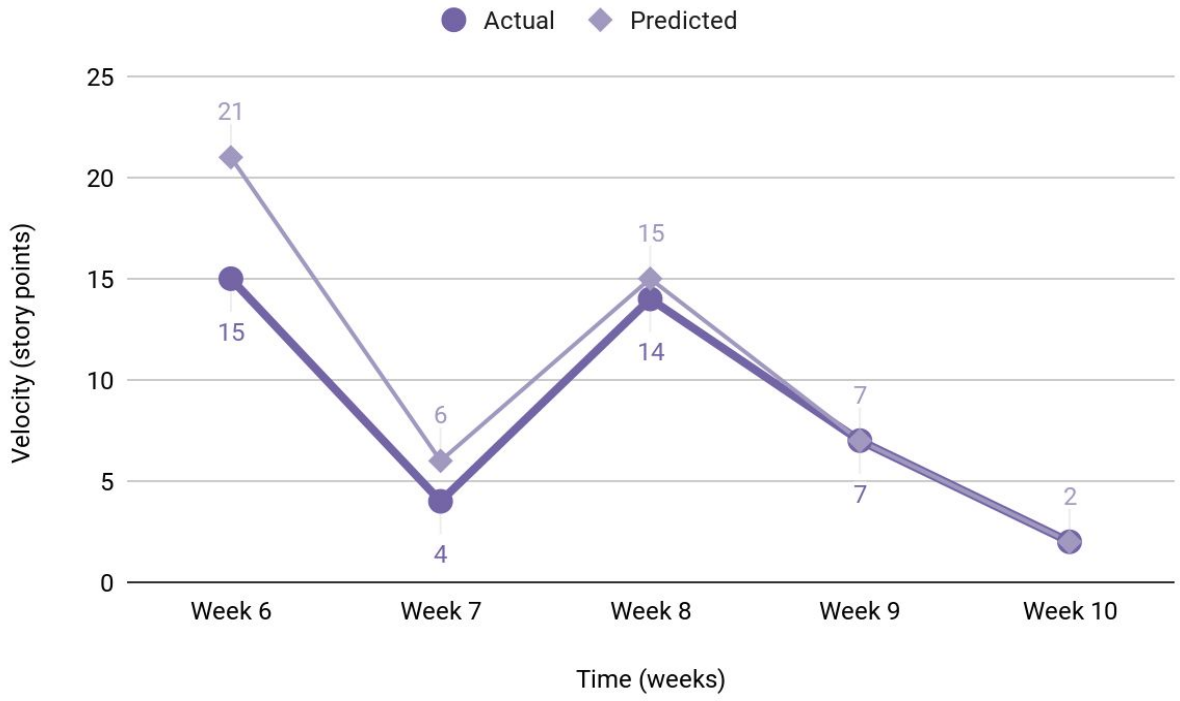
Our meeting schedule was set up to have a Sprint Planning meeting every Sunday, along with check-ins on Tuesday and Thursday. In the early phases of the project, we came up with a rough timeline of which features from the feature backlog (see below) we wanted to implement each sprint. Before starting the development process, we assigned story points to each feature through Planning Poker. During the Sprint Planning meeting, we would use the timeline as a reference to determine which features to assign for that specific sprint. We also reflected on the

previous sprint, and took into account the features that were delayed, giving them priority over adding new features.

# Feature Backlog

| | | | |
|---|---|---|---|
| 8 | User login | 5 | Notifications |
| 5 | User creation | 8 | Recent chats |
| 1 | User logout | 8 | Friends list |
| 2 | User session | 8 | User blocking |
| 3 | User list | 8 | Group messaging |
| 5 | Room creation | 3 | Online status |
| 8 | 1:1 messaging | 2 | Emoji support |
| 5 | Search by username | 8 | Typing indicator |
| 5 | Read receipt | 8 | @ mentioning |
| 3 | Timestamps | 3 | Dark mode |

We had a very adaptive estimation conformance as we initially had a higher estimation on the tasks we were going to do but later on the estimation time and the development time converged. The reason we had a higher estimation in the beginning was that we didn't know how hard and long to set everything up using React and had a huge underestimation on the testing, more specifically, Jest mocking. The tasks speeded up as we all knew how to do coding in React and



testing using Jest. We started coding in week 6 and hence why our velocity chart starts then. This velocity chart only tracks features for the application and doesn't include story points for UI polishing and documentation. Therefore we see a steep drop off towards week 9 and 10.

## Sprint Structure

We have 7 sprints and each sprint lasts for one week.

**Week 5:** Design user stories, use cases, and CI/CD pipeline
**Week 6:** Design the database schema, screen sequence diagrams and finish user login and creation functionalities
**Week 7:** Finish user chat room function. Write unit tests for each corresponding function
**Week 8:** Add user list and search modules and unit tests.
**Week 9:** Integrate Cypress End-to-End tests and UI polishing
**Week 10:** JSDoc deployment, code climate, code coverage report
**Week 11:** Final Sprint: Final UI tweaks, documentation overhaul, final report/demo

## Backlog and User Story Management

We first brainstormed our user stories and then we built a dependency graph as well as a priority graph. By combining these two graphs, we were able to figure out the top priority user stories that need to be implemented first to enable starting the development of the others. When it comes to implementing features for these user stories, our definition of “done” is that documentation, code, and unit tests are all completed and merged. We did not integrate E2E testing until later in the project, when E2E testing existing features became their own tasks in sprints.

To manage and track our progress on features and related tasks, we created and used a sprint board on Trello. You can access it here.

Our sprint board consists of having our backlog items on the left, feature progress in the middle, and completed items on the right. Each card has assignees, labels for the type of task, due dates, and typically checklists. When a team member begins working on a task, that card is moved from the backlog section into the in-progress section, and when it is completed, it is moved to a column of completed items for the week.

Normally, our in-progress section would be populated with tasks related to user stories and features, but as we are mostly focused on polishing our product and documentation at the moment, our sprint board currently reflects that, instead.

## Retrospectives

We have weekly retrospectives where each member is asked to fill in four questions: What went well, what did you learn, what can you do to improve, what is your individual morale. We

maintain a repository for this, for which the hosted site can be found here. Here is a list of some of the takeaways from our blog:

- People were accountable, respectful, and worked hard
- Talking to the client and doing more user centered thinking sooner is very important
- We learned many new skills and tools, such as GitHub Actions, JS Doc, and Code Climate
- We were glad that we took on a simpler project and that we got the opportunity to improve our skills regarding what it means to do systematic and quality software engineering
- People became more confident with abilities over the course of the quarter
- For some people, morale was consistently, “we’re doing great,” while others either had weeks where they felt more stressed or less confident in what they were bringing to the table
- We all feel like we did quality work, especially in terms of putting emphasis on the process (automated workflows, documentation, etc.)
- Although we finished our MVP early, we were not prepared for the sheer amount of remaining work we had wrapping up remaining tasks

# Execution

## Repository

You can find our repository here. Here is a non-exhaustive representation of the most important components of the layout of our repository:

```
.github/
    workflows/
        <GitHub Actions workflow files>
    <Pull request template>
cypress/
    fixtures/
        <Test messages, rooms, and users for E2E testing>
    integration/
        <E2E tests>
public/
    <Images for landing page and favicon>
src/
    modules/
        <Source code>
    utils/
        <Source code for helpers>
```

```
__tests__/
    <Jest unit and integration tests>
```

In addition to the above, there are configuration files and package files in the root directory. Branching and commit strategy can be found here. We have both a production and development application, where our main brain deploys to the production app and the dev branch deploys to the beta app. For this reason, developers create feature branches out of the dev branch, which is also the branch that PRs are submitted to. In the previously linked guidelines, you can also see that we had a branch naming convention for feature branches. After PRs were merged into dev, we did not immediately or routinely update our main branch (and thus, production app) throughout the quarter. However, in reflecting, we do feel that we should have done this more often in the later stages of the project (see our retrospectives at the end of this document).

## Code Quality and Structure

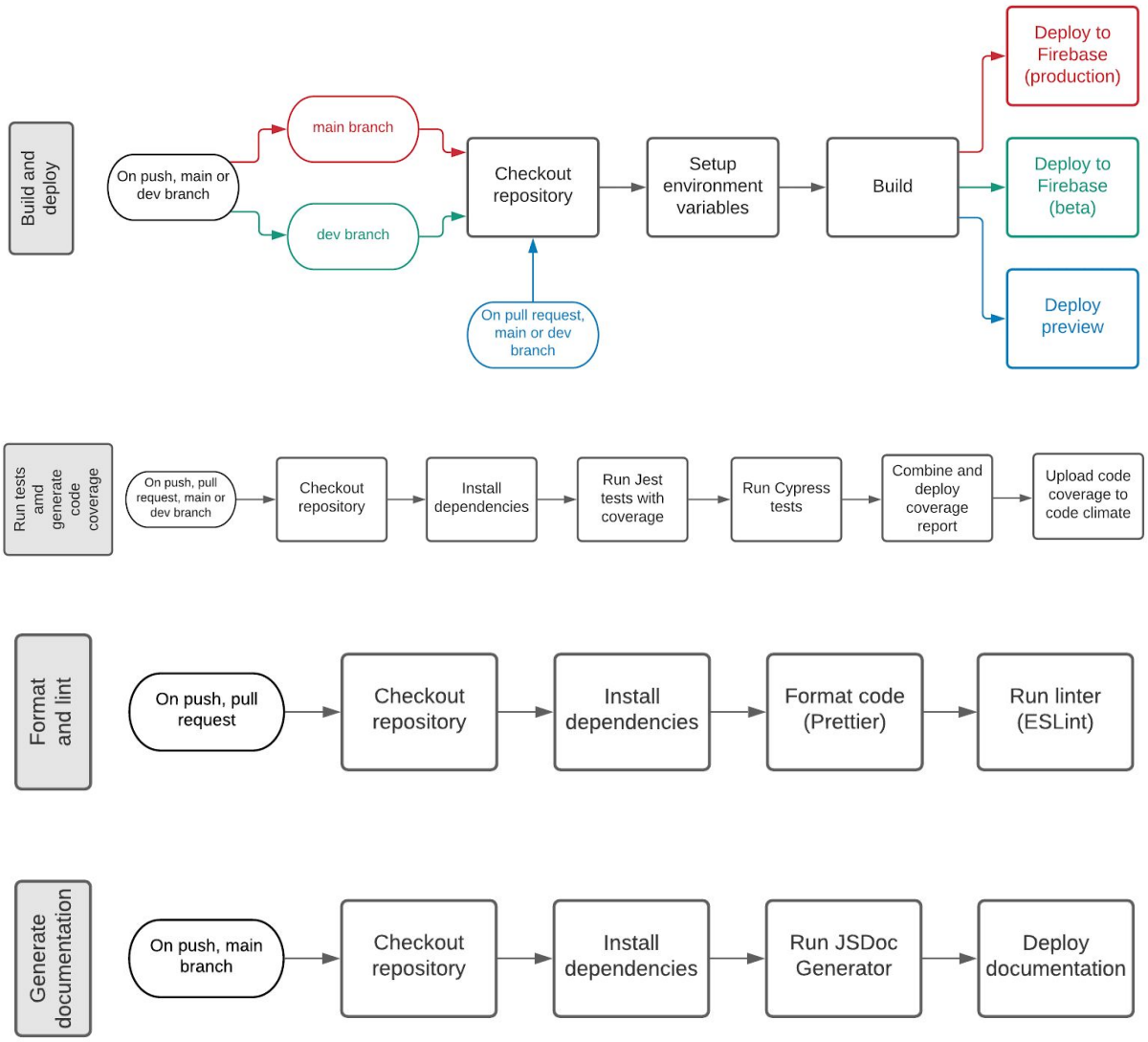
To ensure the code quality, we have taken the following measurements. First, we follow the screen sequence diagram to design the code structure. Second, we apply linter in Github workflow to automatically format our code styles. Third, we require at least one reviewer to review the Pull Request before merging. Fourth, we are using Code Climate to ensure a high maintainability score (A). Fifth, we are using Jest and Cypress testing to ensure the correctness of our code. In addition, we also generate a combined code coverage report from Jest and Cypress to make sure our tests cover a wide majority of our code. Sixth, we are using JSDoc to document our code for reference. All metrics can be found here in the readme.

## Testing

To test our application, we use Jest and React Testing Library for unit and integration tests, and Cypress for end-to-end tests. For most of our Jest tests, we set up mock data for users, rooms, and messages and mock calls to Firebase so that we can test our code independent of our Firebase backend. In a typical unit or integration test, we set up the mock data and mock the firebase SDK return values, render our application, and assert that various behaviors execute as expected. This may include asserting that certain values are correct or that certain text is displayed on screen. For our E2E tests, we have set up a dedicated test user and fake data that we use in conjunction with real calls to our backend. We design our E2E tests to mirror common user interactions in our app flow. While developers were required to write unit tests alongside new features, we did not incorporate E2E testing until later in the project.

## CI / CD Pipeline Overview

We first had only a linter and Firebase deployment workflows. Later, when we added testing, we added workflows for automating Jest and Cypress testing, report generation and hosting. We also added the workflow for JSDoc generation and hosting on a web page. The detailed workflows are shown below.




We recognize that it may not be ideal to deploy our documentation and coverage reports on pull requests. This is not something that would work in a large scale project where many pull requests were being submitted. However, because we had relatively few pull requests and most of our time was spent developing the core, we found it convenient to have these things deployed as part of pull requests. Perhaps in a project with a larger scope, this would not have been as appropriate, but it worked for us.

## Documentation

Our JSDoc documentation can be found here. We use a GitHub workflow to deploy the latest version of JSDoc on GitHub Pages after each push, and developers were required to document any new code that they contributed. You can find more details in the README, which contains information about our project's structure, workflows, links to the artifacts, and steps for running.

In addition, we provide onboarding instructions for new developers to get started, as well as setup steps for someone who would like to extend our project as their own.

## Product Demonstration

Our chat app is located here: https://twotenchatapp.web.app/
Our demo video is located here: https://youtu.be/6IIzYT-fGmo

## Why Our Project?

In working on our project, developers will get hands-on experience with real-world software engineering libraries such as React, Firebase, Cypress, and Jest. While we have a minimum viable product operating, we also have an extensive backlog of interesting and critical new features for new developers to work on, such as notifications, friends lists, and group chats. In addition, we strive to provide clear steps and documentation for a new developer so that their experience is smooth, whether they are joining the team, or looking instead to extend our project as their own. To this end, we've filled our repository and README full of resources. First, we provide diagrams detailing the landscape of our application. We also provide detailed instructions for getting up and running not only if you're joining our team, but also if you're instead someone looking to extend our project as your own, where you will need to configure your own application and environment. We also provide details about our documentation and testing, and include all of our artifacts, including our database schema, specifications, feature backlog, and contributing guidelines. It is our hope that providing these resources, a new developer will have a smoother experience joining us or extending our project as their own.

# Retrospective and Analysis

## Time Machine

Positive things:

1. Team communication: we communicate with each other efficiently
    a. The team is cooperative and not procrastinating, we always "Start Early, Start Often"
    b. The team always has consistent, regular, and efficient communications
    c. The team is able to reach consensus and make decisions after short discussions
2. User-centered thinking
    a. The team regularly checked in with the Professor to get clarification on requirements and make sure we are on the right track
    b. The team decided to work on a simpler application idea rather than a more technically complex one. In retrospect, the amount of time required to set up the CI/CD pipeline, build workflows, and testing frameworks was greatly underestimated
3. Trust the process

    a. The team tried their best to practice the process during these 10 weeks; it definitely helped a lot
    b. The documentation that we had is very detailed and overall helped shape things to where they are now
    c. The github workflows played a crucial role in ensuring a smooth integration/development process

Negative things:

1. We should have picked an easier project, like a Pomodoro app, so that undergraduates can use this project to focus more on SE principles
2. We should have done more research early on
    a. We should have used code climate early to take advantage of the maintainability score along the way
    b. We should have used JSDoc from the beginning so we don't have to waste time on writing our own API documentation manually
    c. We should have integrated Cypress from the beginning so that we can write unit and end to end tests in parallel with the feature development
3. Do more PROD deployments for Continuous Deployment/Delivery!
    a. We only pushed to main at the very end of quarter so we didn't really get to test our pipeline involving the main branch and because of this, we don't have time to fix a subtle problem that surfaced just now in the deploy preview workflow for the PROD app. We believe this issue is caused by the lack of permissions in our Firebase app
4. Unit and integration testing was perhaps too implementation-dependent
    a. Occasionally, we would have to update our Jest tests in accordance with refactoring
    b. Tracing which Firestore SDK calls happened in what order from what function when our application was rendered made mocking more of an undertaking than it needed to be
    c. After refactoring, we realized too late that some of our mocked data does not accurately reflect the database documents that will be returned by certain Firestore SDK calls. In addition, some of our comments refer to old functions that were refactored or renamed. Our tests still properly test what we are trying to test, but we have some issues with correctness in our mocking

# 110 Adoption

We want our project to be controlled and constrained, yet *interesting and novel* enough such that undergraduate students would want to do the project and learn about new things as well as the software engineering process. It's more challenging and utilizes many 3rd-party libraries to make rich functionalities and creates a real-world software engineering experience.

To be honest, we think this app might be too challenging for undergraduates since this app requires a deep understanding of Firebase frameworks which may appear foreign to

undergraduates and cause extra burden on them. In a course that is heavily favored in software engineering practices, we should make the project itself less challenging so that they can focus on adopting correct SE principles.

In the end, we think we underestimated the difficulty of the project, albeit it is interesting and novel.

While the components of our application may be too difficult to adopt in CSE 110, we do see value in the nature of the project being a starter-kit. In the event that someone would want to adopt or adapt our project for CSE 110 or 112, we do provide some steps for how a student might go about getting up and running with extending our project as their own. In addition, we consider any pending features in our feature backlog to be great material for students to implement as part of their own project. These can be found in our User Stories.

# Miscellaneous

## Materials

Browsers & Frontend Development:

Through this reading, students will learn the basic knowledge about frontend development. Since many software developer positions like frontend developer or full-stack developer require candidates to be familiar with HTML/CSS/JS, this topic will definitely help students in their future careers.

Testing:

Students will learn the differences among unit tests, integration tests and end to end tests, which are crucial for ensuring correctness of the code. The reading talks about how to avoid writing untestable code, which is useful in real-world software development. They will learn how to view the application in different perspectives (e.g. end users') so that they will be able to write a more comprehensive testing to take care of corner cases from unexpected behaviors of the users.

CI/CD and Git Workflows:

Automating project deployment is one of the most important concepts that undergraduate students should learn before stepping into a real-world software development environment. Github workflows allow you to have a working version of your application at any time after you push changes so that you would be able to get instant feedback from the QAs or customers. Github workflows also enable automatic linting, testing, report generation, web page hosting and more, which are extremely helpful for the integrity of a project.