Final Report

Caffeinated Coders

Introduction

Team 3: Caffeinated Coders
Theme: “Drink more coffee, It’s our hobby!”
Members:

Megan Chu - Testing front/backend, Frontend, Documentation
Miranda Go - CI/CD Pipeline, Testing, Leading meetings, Documentation
Wei Liu - Front/backend, Wireframing, Testing frontend
Divyank Rahoria - Wireframing, Front/backend
Ruixian Song - Storage API, Tasklist Frontend

Project

Problem

Creating a project that the undergraduates students can complete that is high quality and demonstrates the software engineering process. The project should be constrained, yet leaves some room for design decisions

Solution

Demonstrate the software engineering process by creating an example undergraduate Pomodoro Timer Chrome Extension: Project Pitch

Brainstorming

- We decided to use Miro to brainstorm our designs: Project Pitch Brainstorm
- We did initial research on the type of platform we should build our project on by creating a “Hello World” program and seeing which platform would be the most feasible: Chrome Extension, VSCode Extension, Cordova Mobile App, Electron Desktop App, and Slack App (Tutorial)
- After deciding on creating a Chrome Extension, we researched few testing frameworks: Selenium, Sinon-Chrome+Mocha+Chai, Cypress
- We looked into existing Pomodoro Timer Chrome Extensions to get inspiration on features we might want to implement in our project: Marinara, Focus To-Do, Habitica

Designing

- After getting inspiration from existing Chrome extensions and having a good idea of the tools we would use, we began to write user stories and create wireframes from the features we wanted to implement: Use Cases and User Stories and Wireframes

- We created diagrams to represent our code structure and CI/CD pipeline: System Diagrams

Process

Stand-up

On Tuesdays and Thursdays, we have team meetings through Zoom that begin with stand-up: Team Meeting Notes. We use StatusHero for async standups on Mondays, Wednesdays, and Fridays:

Sprints

Sprints were planned after assigning a priority level for each user story. First, we focused on the “must-haves”, having one sprint for frontend and one for backend. Then, we tackled the “good-to-haves”, having a sprint for frontend and backend. In order to estimate time for sprints, we did a planning poker activity through Zoom to estimate how much time we thought each task would take, recording the values on Zenhub. We also took into account everyone’s schedule for the week to assign issues.

We decided to do our sprints in 1 week intervals because we had 5 weeks left to develop our application. Any features that may take longer than 1 week to implement may be in danger of being incomplete. Keeping this in mind, we kept our tasks and issues as small as possible. Short intervals also allow us to quickly adapt to new changes and address issues sooner than later. Below is a list of our planned sprints from week 6 to the end. Each sprint includes tasks we wanted to finish, as well as our estimates and time spent on each sprint.

Week 6 (Estimated: 16 hours, Reality: 13 hours)

- Front end for Priority 1 (Timer)

- Use Cases, User Stories, Wireframes

Week 7 (Estimated: 24 hours, Reality: 15 hours)
- Backend and Complete Priority 1 (Timer)
- Front end for Priority 2 (Tasks)

Week 8 (Estimated: 13 hours, Reality: 11 hours)
- Backend and Complete Priority 2 (Tasks)

Week 9 (Estimated: 13 hours, Reality: 12 hours)
- Finalize documentation for team
- Stretch Goals: Customization

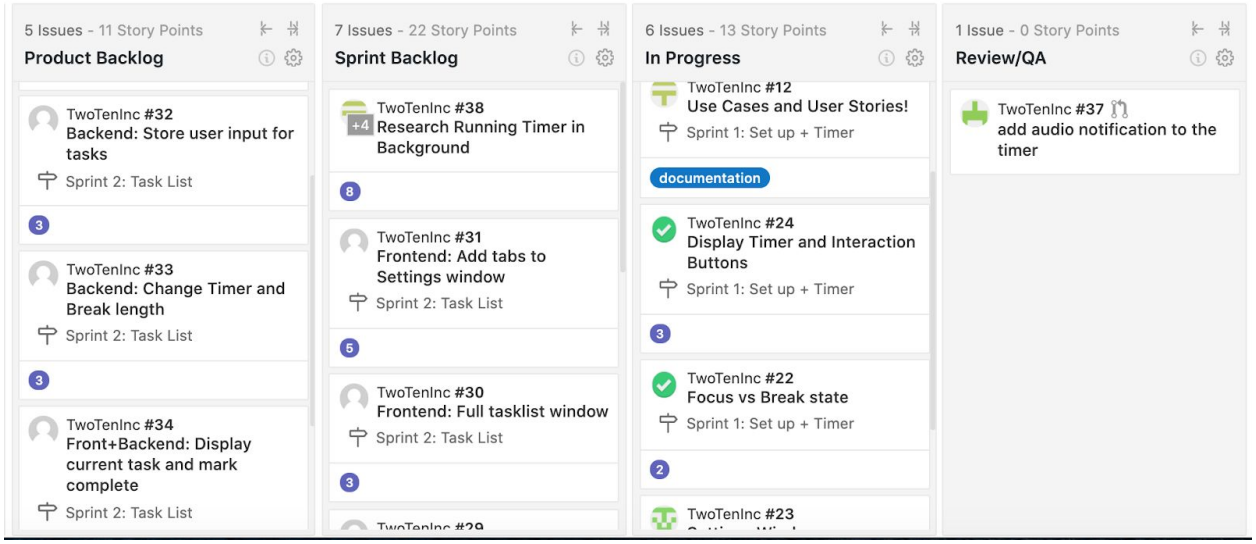
Week 10 + Finals Week (Estimated: 10 hours, Reality: 10 hours)
- Documentation for Undergrads
- Final Report and Presentation

Backlog and User Management

For our backlog, we decided to use Zenhub to keep track of the tasks for the current sprint as well as new issues or bugs that may have come up to address in future sprints. We split our user stories into smaller tasks on Zenhub, once we completed all the tasks for a given user we considered the user story to be "done". For our extension, "done" meant that we were able to:

1) meet all the requirements in user stories
2) complete automated and hand testing for a user-friendly experience
3) ensure organized and bug-free code on the development side.

Here is an example of what our Zenhub looked like on Week 7:



Retrospectives

In each team meeting, we brought up any successes or struggles pertaining to the project or tools we were using.

For example, one major issue that popped up toward the beginning was deciding our testing framework. There were frameworks that are commonly used, like Selenium, but our team was struggling in adopting it for our Chrome extension project. After weighing the pros and cons, we decided to go with Cypress. Another decision that we made early on was to use Chrome storage instead of Firebase. However, halfway through the project, we had to restructure how we stored information within Chrome storage so that it would fit in the limited amount of memory Chrome storage gave each user.

A more detailed list of each of our thought process behind each decision can be found here under “Design Decisions”: Design and Tool Decisions

Execution

To access and build the Chrome extension, follow the instructions on this document or watch the video tutorial: Getting Started or Video Tutorial

Repository: Github Repository

In the repository, we have 3 main folders: code, cypress, and test. The code folder includes all of the source code for the application. This includes the html, javascript, and folders containing resource files such as audio files and images. The cypress and test folders contain tests for our code. The test folder includes files for unit testing with our backend code with Mocha while the cypress tests contain front end UI testing. In addition, the github page includes a Wiki page which explains the procedures to create a pull request and steps to write and run tests.

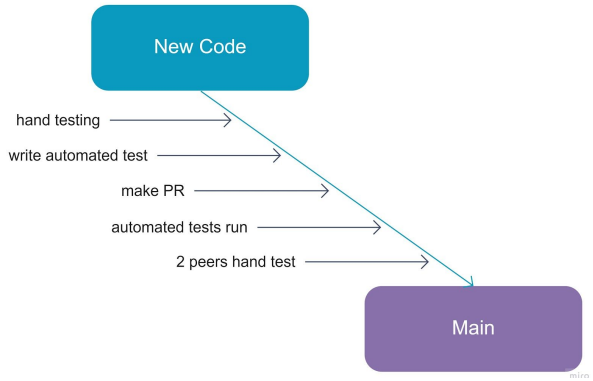
When developing new features, we create new feature branches from the main branch. Our feature branches are named “dev/feature/member” so that we can quickly identify what task is being worked on and who is working on it. Once merged, these branches are deleted to avoid merge conflicts for long outdated branches. Some additional rules include 2 automatic checks after each push. Linting and testing are done to ensure that the code is structured nicely and that the new code did not break existing features. Merging is blocked if the branch fails any of those checks. We also require a pull request to have at least 2 team members to review and approve the code changes before the code can be merged.

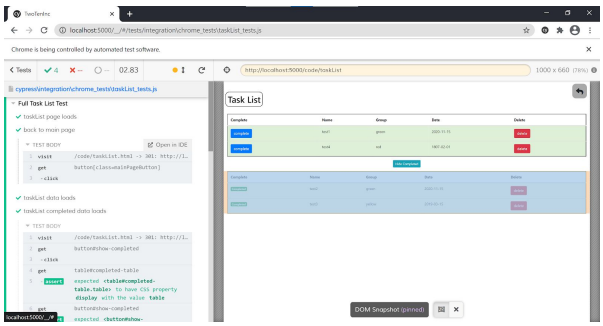
Code Quality and Structure

To ensure code quality and consistency, we use ESLint to format our code into a clean, easy to read structure. It also checks the code quality by removing unused variables and functions. For each function in our code, the linting tool also enforces that there is a javaDoc associated with it. This linting process is done automatically after a push. While working on a branch, we encourage team members to push often so that the automatic testing can catch any potential bugs early on. We follow our API specifications to communicate between different components such that we can work in parallel. As a side effect, this allows us to structure our code into more modular code.

Testing: Testing Documentation

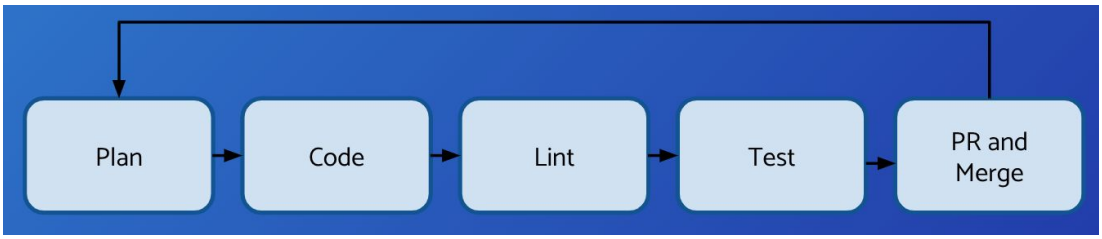
We used behavioral-driven and acceptance-test driven testing. We performed unit tests using Mocha on our timer and storage functions. We performed integration tests in Cypress on each of our HTML pages to ensure the correct components existed and that they responded to user input as desired. In between PRs, we hand tested our extension to ensure proper behavior.





CI/CD Pipeline Overview

Our CI/CD pipeline was consistent throughout the entire project, simply because it worked for our team and we did not feel like we needed to add more aspects in order to produce quality code. Below is the CI/CD pipeline diagram and a short video displaying the process: Video of CI/CD Pipeline



Documentation

For the code, we documented each function in the repository and enforced it using ESLint Github actions. Each function must be properly documented with a comment, parameter type, and return type.

Internal Documentation:

- For instructions on how to contribute to the repository: Contributing to the Repository
- For instructions on how to get started with Chrome Extensions: Getting Started with Chrome Extensions

External Documentation:

- We created an onboarding document for new members of the team or anyone that would like to contribute to the project: Are you new to the team?
- We also created an API document for our storage and timer API functions with example usage: API Documentation

Product Demonstration: Focus with Caffeine Video Demo

We created our demo video centered around our user stories, so it will be a user story followed by a recording of our application fulfilling the requirements. Since our application is called Focus with Caffeine, we centered our theme around coffee!

Why you should choose our project

- Accessible: Our project is easily accessible by the users as well the developers. We have provided how to find our project online and how to use it in our demo video. Our repository has all the documentation related to the project as well the code so that developers can easily access it.
- Don't need separate devices to test: As this project is Google Chrome browser extension based, we don't need separate devices to test.
- Good for people comfortable with front-end: In this project, there are many different features that are mainly front end. This project gives enough opportunity to the students to test their front end software knowledge.
- Room to spice up the project: We kept some spaces open so that this project can be used in many ways for undergraduate students. Our project is expandable. Developers can incorporate more features, like adding more sound options for focus and break mode, creating a user login, restricting websites while focusing on tasks, etc.

Retrospective and Analysis

Time Machine

Negatives:

- Start earlier on the project: Although we started earlier than most groups, we felt that the final two weeks would have gone more smoothly if we had started earlier on the project.
- Testing for chrome extension is difficult: Working on a project presented on a less mainstream platform also meant that there were not many resources for testing. We did find a couple of tutorials and tools to use, but many were outdated and deprecated.
- Velocity estimations: Part of the planning process was to estimate how much time we would have to put into the project each week to complete it. If we learned about planning poker and velocity estimations a little earlier, our team might have had a better grasp on our time commitments for that week rather than making that a norm later on in the quarter.

Positives

- Consistent meetings: Meetings helped us discuss problems and push our project forward. We were able to plan tasks and assign people to them. We were also able to quickly approve and merge any open PRs to keep our main branch updated.
- Detailed product planning: Having a fully thought-out wireframe helped up work quickly on our MVP. We knew exactly what we wanted our extension to look like and do after completing an issue.
- No hackathon!: Despite the really busy quarter, our team was able to continue to make consistent progress each week and we did not have to hackathon the final week, so we could focus on finishing touches and the report.

CSE 110 Adoption: CSE 110 Project Description

We have outlined in the CSE 110 Project Description document how a CSE 110 version of our project would be run. We believe that the pomodoro timer is a simple application that undergraduates can easily build and may be used to teach the principles of SE. They will have enough time to focus on other aspects important to the project, such as Agile, CI/CD, testing, and documentation. One change we would make to the requirements is that undergrads may develop their apps on a different platform. One major struggle we faced was that documentation and online resources are scarce for chrome extension developers. This made setting up our dependencies and testing framework difficult.

Miscellaneous

CSE 210 Materials

We enjoyed this class and learned different important skills that helped in this project. We identified three of these many important readings that we think helped us the most, and these are Testing, CI/CD pipelines, and Chrome DevTools.
We think this would help the undergraduate students specifically for this project, but also in advancing their knowledge as web developers. The Chrome DevTools video helped us understand how to develop extensions, specifically in using the console and inspecting elements. The testing reading helped us avoid merging untested code and taught us how to mock our storage and other objects. The CI/CD pipeline readings helped us to understand the advantages and disadvantages of each pipeline and choose the best pipeline for our project.