Change Log

|  |  |  |
| --- | --- | --- |
| Change | Date of Change | Author |
| Baseline | 8/28/22 | Pauline Wade |
| Sprint 1 Completion | 9/19/22 | Macy Drew, Mualla Argin, Dillon Lee, Nathan Patterson, Frank Martinez, Daniel David |
|  |  |  |

Final Report

Parent Portal

Aggie Pregnant and Parenting Students Organization

Macy Drew  
Mualla Argin  
Dillon Lee  
Nathan Patterson  
Frank Martinez  
Daniel David

CSCE 431 – Software Engineering  
Fall 2022

Texas A&M University

Department of Computer Science and Engineering

Contents

[1 Project Scope 5](#_Toc112706462)

[2 Stakeholder Analysis 6](#_Toc112706463)

[2.1 Stakeholders 6](#_Toc112706464)

[3 Implementation Environment 7](#_Toc112706465)

[3.1 Hardware 7](#_Toc112706466)

[3.2 Software 7](#_Toc112706467)

[4 Scrum Process 8](#_Toc112706468)

[4.1 Release Planning 9](#_Toc112706469)

[4.2 Sprint Planning 9](#_Toc112706470)

[4.3 Scrum Meetings 9](#_Toc112706471)

[4.4 Sprint Review 9](#_Toc112706472)

[4.5 Sprint Retrospective 10](#_Toc112706473)

[5 Sprint 1: 11](#_Toc112706474)

[5.1 Overall Experience 11](#_Toc112706475)

[5.2 Stakeholder Management and Communication Plan 11](#_Toc112706476)

[5.3 Risk Mitigation, Monitoring, & Management Plan 12](#_Toc112706477)

[5.4 Release Planning 12](#_Toc112706478)

[5.5 Sprint Planning 12](#_Toc112706479)

[5.6 Scrum Meetings 12](#_Toc112706480)

[5.7 User Stories 13](#_Toc112706481)

[5.8 UX Models 13](#_Toc112706482)

[5.9 Data Design 14](#_Toc112706483)

[5.10 Version Control 14](#_Toc112706484)

[5.11 Quality 15](#_Toc112706485)

[5.11.1 Compliance 15](#_Toc112706486)

[5.11.2 Efficiency 15](#_Toc112706487)

[5.11.3 Correctness 15](#_Toc112706488)

[5.11.3.1 Test-Driven Development (TDD): 15](#_Toc112706489)

[5.11.3.2 Reviews: 15](#_Toc112706490)

[5.11.3.3 Definition of Done 16](#_Toc112706491)

[5.11.3.4 Test Coverage 17](#_Toc112706492)

[5.11.3.4.1 Code Coverage - Output of simplecov 18](#_Toc112706493)

[5.11.3.5 Test Results 18](#_Toc112706494)

[5.11.4 Maintainability 19](#_Toc112706495)

[5.11.4.1 Coding Standard 19](#_Toc112706496)

[5.11.4.2 Well Documented Code 20](#_Toc112706497)

[5.11.4.3 Linter Output 20](#_Toc112706498)

[5.11.5 Integrity 20](#_Toc112706499)

[5.11.5.1 Security Risk Analysis Table 20](#_Toc112706500)

[5.11.5.2 Authentication 20](#_Toc112706501)

[5.11.5.3 Integrity Test Coverage 20](#_Toc112706502)

[5.11.5.4 Integrity Test Results 21](#_Toc112706503)

[5.11.6 Usability 21](#_Toc112706504)

[5.11.6.1 Usability Requirements 21](#_Toc112706505)

[5.11.6.2 Usability Test Coverage 21](#_Toc112706506)

[5.11.6.3 Usability Test Results 21](#_Toc112706507)

[5.12 Monitoring and Control 22](#_Toc112706508)

[5.13 Deploying App 22](#_Toc112706509)

[5.14 Sprint Review 22](#_Toc112706510)

[5.15 Sprint Retrospective 22](#_Toc112706511)

**Recommended content for reflection:**

-Describe briefly what the activity/process/topic was about.

* How you felt as you were going through it.
* How does it help you personally & professionally?

-Describe what you did right (i.e., a WHOOP moment).

* How did you celebrate?

-Describe what you would do differently (i.e., how would you avoid that EPIC FAIL).

Remember that it’s best to FAIL – FAST or FAIL - NOW, instead of FAIL – SLOW or FAIL - LATER (e.g., in front of the customer)

* How will this help you avoid the same mistake next time?

# Project Scope

We tried to approach defining our scope in as user-centered a way as possible: starting with our initial customer meeting, we asked what the organization needed from us, if they had any current problems, and what their priorities were. By having Judith help us prioritize the various objectives, we were able to distinguish the “mission critical” aspects of the project from those that qualified as “stretch goals.” At the same time, we clarified which items mentioned in the initial request were within what they wanted us to accomplish; those that did not make the cut were recorded as “scope exclusions.”

Having input from a current officer of APPSO reassured us that the problem we thought we were solving was the right one and gave us confidence in our proposed solution. Once we wrapped up the meeting, we got to work dividing these priorities into coherent user stories, feeling confident that we would be able to fulfill and even exceed Judith’s and the other officers’ expectations.

|  |  |
| --- | --- |
| **SECTION 1: PROJECT SUMMARY, DELIVERABLES, AND SCOPE EXCLUSION** | |
|  |  |
| **Project Summary** | |
| **Organization Name:** Aggie Pregnant & Parenting Student Organization (APPSO)  **Primary Contact Name:** Judith Tijerina  **Officer Position in the Organization:** Social Media Coordinator  **Email:** [juditijera17@tamu.edu](mailto:juditijera17@tamu.edu)  **Contact Number:** (956)-607-3116  **Project Team Members**: Macy Drew, Dillon Lee, Daniel David, Mualla Argin, Frank Martinez, Nathan Patterson  **Summary of Problem to be Solved:**  Track Texas A&M student parents’ participation in APPSO events and record their financial status and circumstancess in order to aid officers’ decision of quality scholarship candidates. | |
|  |  |
| **Deliverables** | |
| **Deliverable Number** | **Description** |
| 1 | Track member participation (attendance, etc.) |
| 2 | Record/update members’ scholarship eligibility and information |
| 3 | Authenticate both members and officers for access to profile data |
| 4 | Improve outreach to the community for the organization via the website and Parent Portal |
| 5 | Make current website accessible on/off campus for TAMU students |
| 6 | Allow for saving a draft of the scholarship application and returning later |
|  |  |
| **Scope Exclusion** | |
| **Exclusion** | **Reason for Assumption** |
| Non-TAMU CS students | Customer does not award scholarships to non-College Station Texas A&M students |
| Budget | We were asked to focus on the scholarship and outreach aspects of the project |

# Stakeholder Analysis

Our first step was to identify key stakeholders using the stakeholder grid below, especially those with high power and high interest.

The stakeholder grid has four quadrants:

Quadrant 1 (**high power and low interest**) contains a hypothetical “bad teammate,” a person who by nature has low interest, but who could heavily impact the team and the project if they do not complete their tasks.

Quadrant 2, which contains the rest of the dev team, has **high power and high interest**. The dev team has the most power since they are the ones doing the project! They have high interest as they design the front end and back end, and directly control the quality of the project.

Quadrant 3, containing the end users, holds **low power and low interest**. Since the app is being created solely for the Aggie Pregnant and Parenting Student Organization to manage their organization’s members, the end users (org. members) don’t have much of a say in the project, and likely don’t have much of an opinion on development matters.

Finally, Quadrant 4, where TAs are placed, refers to **low power, high interest**: TAs don’t get to decide what the product looks like, unlike the organization officers. However, they have high interest in our project because they oversee our submissions at each sprint.

## Stakeholders

***Client***: Judith Tijerina, representative of the Aggie Pregnant and Parenting Student Organization (APPSO)

***Advising Faculty***: Professor Pauline Wade

***Teaching Team***: Nimisha

|  |  |  |
| --- | --- | --- |
| **Scrum Master** | **Product Owner** | **Team Members & Role if any** |
| Dillon Lee – SQL/Backend | Mualla Argin -- Frontend | Macy Drew – Project Manager, Frontend |
|  |  | Nathan Patterson – Merge requests |
|  |  | Frank Martinez – Merge requests |
|  |  | Thierry David -- Backend |

# Implementation Environment

The implementation environment of our application is described below:

We decided to have our development environment within a Docker container in order to ensure that our computers do not cause conflict with one another, in terms of our various host OS’s. We initially had trouble creating our own docker image to use, but we then realized we could just use the pre-built Docker image provided in one of the labs, as it was preconfigured to use rails and PostgreSQL. The docker container is a Linux distribution that has the bare necessities available to help test the application before sending it to Heroku. We give a sincere thanks to the TA who helped create the pre-built docker image, as it saved hours in preparing a fresh Docker image from scratch.

## Hardware

This project calls for a software-only solution; as such, the only hardware elements used were our various laptop computers. Some design processes were carried out on whiteboards or paper, but the final product will be digital.

Our personal hardware consists of windows/macOS machines, but docker helps normalize all of that into a single environment.

## Software

The following packages and tools were used in the project:

* Operating System (Linux via Docker)
* Ruby 3.0.2
* Rails 6.1.4
* Docker version 4.12.0
* Bootstrap 5.0.2 (if applicable) – For use with front-end, creating preliminary style and organization for our views.
* rspec-rails – Unit/integration testing
* Heroku
* PostgreSQL 1.1
* Git Hub
* RuboCop or Code Climate – Helps check code styling
* Simplecov – Check Code Coverage
* Brakeman – Check for vulnerabilities in rails applications
* Jira – We used Jira to create our user stories and assign such stories to different team members, and overall keep track of our progression each sprint.

The following gems were also used:

* Puma
* Sass-rails
* Webpacker
* Turbolinks
* Jbuilder
* Devise – user authentication
* Bootsnap
* rexml
* Byebug
* Web-console
* Rack-mini-profiler
* Listen
* Spring
* Capybara
* Selenium-webdriver
* Webdrivers
* Tzinfo-data

# Scrum Process

Scrum is a process model that consists of releasing the product in increments during specific time frames referred to as sprints.

The purpose of Scrum is to allow for incremental developments, or small chunks that can be quickly executed in small teams. Given that our project is of a smaller scope, with a very limited time frame and budget, scrum methodology is what we decided was best to use. Each sprint aims to produce a deliverable for the customer which can be used to gain feedback for the next sprint. By communicating regularly with the customer, meeting regularly with the other team members, and recording our work as it relates to our customer’s goals, we use the scrum process for success.

There are a few different scrum ceremonies used throughout this process to facilitate our consistent development efforts as a team. Firstly, there is release planning. This is a meeting we used to gather requirements from our customer and write user stories which tailor our customer’s goals to our sprints. Then, there is Sprint planning. This is a meeting amongst the developers to identify problems to solve and what can be done in the given week, by whom and with what amount of time, to produce deliverables.

Next, the most common scrum ceremony is scrum meetings, or daily standups, where each member itemizes their past, present, and future efforts as well as roadblocks which may hinder them. There are also Sprint reviews and Sprint retrospectives in order to help us think about what we can improve going forward and demonstrate our deliverables to gain valuable feedback from the customer.

See the following sections for a summary of the different scrum ceremonies:

* 5.4 -- Release Planning
* 5.5 -- Sprint Planning
* 5.6 -- Scrum Meetings
* 5.14 -- Sprint Review
* 5.15 -- Sprint Retrospective

Certain software development phases are repeated each sprint. Examples are:

* Requirements
* Design
* Design
* Development
* Test
* Deployment

In general, the team iterates across the different phases, which do not have to be done in a specific order. For example, testing can (and should) happen simultaneously or before development.

Also, certain documents are prepared throughout the project, then monitored and revised when necessary in each sprint. Below is a list of a few documents recommended in this class, organized by phases. Documents listed under “Umbrella Documents” may not belong to a specific phase but may (and should) be created / revised throughout the project timeline.

There is no universal set of phases or documents, as it differs for every organization.

Pre- Sprint 1 (Set-up)

* Coding Standard
* Definition of Done (in the report)
* Scope (in the report)

Umbrella Documents

* Stakeholder Management & Communication Plan (in the sprint report)
* Risk Plan (and Risk assessment) [APPSO-ParentPortal\_RiskAssessment.docx](https://tamucs.sharepoint.com/:w:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%201/Documents/APPSO-ParentPortal_RiskAssessment.docx?d=wba5000def4b84637befa1164636283fe&csf=1&web=1&e=wcDqrJ)
* Metrics Document

Requirements

* User Stories with Acceptance Criteria (in the sprint report)
* UX Models

Design

* Data Design [APPSO-Parent-Portal\_DataDesign.png](https://tamucs.sharepoint.com/:i:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%201/Documents/APPSO-Parent-Portal_DataDesign.png?csf=1&web=1&e=fVsEUh)
* Other Design Models (e.g., architectural, functional, behavioral, etc.)

Implementation

* ReadMe

Test

* Test Cases (in the report)
* Test Results (in the report)

Deployment

* Deployment Plan

Maintenance

* Support & Maintenance Plan

Going through the database design was quite an experience. It was hard at first to understand what we needed as entities, and the relationships between the entities. We were fortunate to have something created after our discussions. Creating the User Stories and the Acceptance Criteria was quite lengthy since we had to brainstorm and consult with the customer on what they want and translate that into tangible user stories. We managed to get that done.

The Stakeholder plan and risk plan was something we did in class and was straightforward. Test cases and test results were something that we had little implementation within our first sprint, but we look forward to fully implementing them in the next sprint as we become more accustomed to rails. One thing we would do differently is consult with the TAs and Professor more regarding confusions within the certain documents and plans assigned in Canvas. It is important that we clear up confusion on our behalf.

## Release Planning

Release Planning involved capturing requirements as user stories which is a common language for all stakeholders, including non-technical customers, on the requirements of the application. Collectively these user stories make up the product backlog.

Jira was the primary tool used, and was helpful in doing the following:

* + **Backlog:** We used the backlog in order to write up our user stories and subtasks associated with the stories.
  + **Sprints:** We used the sprint section of the backlog in order to plan what would be done and by who in the given week. Jira, being a powerful tool, also enables us to add more issues from the backlog if a developer finishes their assigned tasks, to keep working toward our goals.
  + **Roadmap:** Jira’s roadmap enabled us to get a better understanding of our sprint progress via a different view.
  + **Board:** The board is the main tool we used during scrum meetings in order to determine what our developers have been working on and allows us to easily move issues to different stages of completion as we understand what each other are doing and review their work.
  + **Epics:** We were able to organize better the overarching customer goals of our user stories and issues by assigning them to specific Epics.

Release planning allowed us an opportunity to take our customer’s desires and produce an organized list of them along with implied requirements. This was a great experience for us to gain an understanding of the scope of the project and make us think about what would ultimately be required. By putting user stories in the customer’s perspective, we were able to think about the ultimate goal as we designed our sprints going forward. Once we started creating issues through Jira based on the user stories, we were able to divide work amongst the developers with some more technical requirements, while keeping the common ultimate goals in mind and written in issue descriptions.

(See Section 5.7 for Sprint 1’s user stories and backlog)

## Sprint Planning

We started off our sprint planning meeting by introducing ourselves and our strong suits. Through this discussion, we were able to identify team roles and best fits for those team roles. For example, someone who had lots of experience with the back end became the back-end master (See section 2.1 for more details on role assignment specifics). Once we had our roles set, we decided to reach out to our customer to get a better understanding of project scope. We used the project scope discussion to create user stories on our Jira dashboard. Everyone in the team made at least one user story and assigned it to themself. We later split these tasks up into different sprints. We left the currently applicable ones like setting up our implementation environment and back end in sprint 1. Additionally, we assigned the user stories to epics.Collectively these user stories make up the sprint backlog. In the initial sprint (e.g., pre-sprint), one of the key activities was the set-up of our implementation environment, consisting of hardware, software, tools, which is described in an earlier section. The project involved 3 normal sprints, each with its own sprint backlog.   
Releasing the application over 3 sprints I feel like allows us to incrementally build the application over time, or in a different perspective, help refine our application over time. It doesn’t make the project feel overwhelming in terms of the amount of features/user stories. Releasing the application like this provides us with more experience on how industry is like with their own sprints, but in a more, fast-paced manner (because of the fall semester). We’ll have more thoughts on this process as we go through more sprints.

## Scrum Meetings

Every other day of the week, our team got together for Scrum Meetings, aka daily standups. These meetings are a short period of time in which we state our current status of our goals given what we assigned to ourselves in the given sprint. One by one, we communicated generally three things:

1. What we did since the last meeting

2. What we plan to do going forward, until the next meeting

3. What “roadblocks” we have that get in the way of ideal progress velocity.

Dillon, the scrum master, coordinated the meeting and addressed any concerns such as roadblocks that were discussed. These meetings took no more than 10 minutes, usually a bit over 5 minutes.

These meetings resulted in us getting a good understanding of what each person was working on, which is useful for staying organized and knowing who to communicate with to work with each other. It was also a good starting point for collaboration. We all have busy schedules, but we’ve found a time to work together, and usually we will stick together after our scrum meeting to discuss specific issues within our sprint. Overall, these scrum meetings were quite beneficial to stay accountable, and jumpstart collaborative development.

## Sprint Review

For each sprint review, we emailed our customer at least 5 days before our scheduled sprint review, to confirm their availability, with a maximum time allocation of 45 minutes.

Product Owners were required to attend. Everyone else in the team that was available was encouraged to attend (although extra credit was given if everyone in the team attended).

The meeting was held via our MS Teams channel and recorded.  Videos were on, faces visible.

Sprint Review agenda included (with recommended durations):

1. Welcome the Stakeholders – Product Owner welcomes the stakeholders to attend the review and introduces everyone (3 min)
2. Present Review Agenda – Product Owner presents the agenda for the Sprint Review (2 min)
3. Present Product Increments – Development Team presents the product demo that have been implemented in the Sprint (7 minutes)
4. Get Feedback – Product Owner asks the stakeholders for feedback regarding the product that have been delivered (13 minutes)
5. Present Product Backlog – Product Owner presents the top of the Product Backlog to stakeholder to get feedback for the upcoming sprint(s) and solicit feedback from the stakeholders related to the backlog (15 minutes)
6. Conclude meeting – give preliminary dates of future sprint reviews (5 minutes)

This meeting has yet to be conducted as the time of the writing of this report; as such, the agenda will be located on Teams for review.

## Sprint Retrospective

You will record this on your Teams channel (approx. 30-45 minutes) and the teaching team will watch it asynchronously (we may also choose to attend). All members need to be present (with videos on / face visible) and need to say something related to the items listed below.  The person in charge of the meeting is the Scrum Master.

The agenda, with recommended durations, included:

1. What went well in the Sprint? (10-15 minutes)
2. What went wrong in the Sprint & for each one, what should we do differently in the next sprint? (15-20 minutes)
3. What did we learn in the Sprint? (5-10 minutes)
4. Action plan or next steps with assigned person for each one (10 minutes)

This meeting has yet to be conducted as the time of the writing of this report; as such, the agenda will be located on Teams for review.

# Sprint 1:

## Overall Experience

* + Increased appreciation for the devise gem which was used to create the log in and sign up for users
  + Introduced to CRUD (Create, Read, Update, Delete) Model which will be utilized in all sprint
  + Increased proficiency in version control through github ci/cd (i.e merges, pull requests, branching,etc)
  + Test-Driven Development effort through unit and integration tests (I.e. brakeman, rspec, simplecov, etc)
  + Increased familiarity with Ruby on Rails framework and its multitude of working parts (I.e. controllers, models, views,etc)
  + Collaborative effort with APPSO point of contact through project scope and design calls.
  + Collaborative brainstorming to develop effective and efficient database design.
  + Gained experience with JIRA tools throughout our scrum ceremonies, from planning to meetings.

## Stakeholder Management and Communication Plan

A key part of the success of the project was the use of the stakeholder management and communication plan.

For Sprint 1, this plan was used to create a high-level view of who exactly are involved within the project, the high-priority individuals, and specific roles for each of them. This allowed us to plan out our application and user stories with our stakeholders in mind. This lead to user stories being more defined, and their acceptance criteria being ironed out. It helped clear the fog on what was needed in the project, which was present early in the beginning of pre-sprint.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stakeholder management and communication plan** | | | | | | | | | |
| **Stakeholder Name** | **Category** | **Levels of power and interest (according to grid)** | **Requirements / expectations** | **Strategies for gaining support / reducing obstacles** | **Information needed / Document Name** | **Document Format / Medium** | **Contact Person (if different than the stakeholder)** | **How Often / When Due** | **Status** |
| Judith Tijerina | Customer | High / High | Release of important features first | Release planning | Sprint schedule with User stories ordered by priority | Google Meet |  | Once/ 2 weeks | Judith wants the team to contact her directly |
| Pauline Wade | Instructor | High / High | Feedback on progress | Regular communication | Status | Zoom – office hours | Once / month ; more frequently when needed |  |  |
| Nimisha FYU | TA | High / Low | Organized work; clean submission | Submit assignment on time according to instructions | assignments | MS Teams;  Github |  | See schedule |  |
| Macy Drew,  Mualla Argin, Dillon Lee, Nathan Patterson, Frank Martinez,  Thierry David, | Project Team | Low / High | Implementation / Release of important features first | Sprint Planning (Internal) | user stories ordered by priority | MS Teams |  | Once/ 2 weeks |  |

Example below:



## Risk Mitigation, Monitoring, & Management Plan

Risk Analysis was used to identify risks during every sprint, which are potential problems that may occur. For each risk, we estimated the probability of occurrence, and impact should the risk become reality.

It was imperative that we conducted a Risk Analysis for Sprint 1, as it creates a great baseline on what exactly are risks are, and how likely they are to happen. We feel like this is an essential and quite underrated task, as nowadays a lot of organizations are getting compromised for overlooking risks.

Each risk was ranked based on the risk exposure (probability multiplied by impact), after which a cutoff line was decided, with risks above the cutoff line considered important to mitigate, monitor, and manage. Once the risk table has been established, it was important for the team to constantly monitor the risks, according to the monitoring plan, to see which risks will likely become reality. To avoid the risk from becoming reality, the team developed and implemented the mitigation plan. Once it was apparent that the risk was going to happen, the team attempted to reduce the negative impact by following the management plan.

The RMMM plan allowed us to pinpoint potential risks early on in our project, so that we can avoid such headaches in the future. It made us think in the mind of someone who would want negatively to affect our application: A malicious individual, like those who attack big companies. The Mitigation and Management part of the plan allows us to detail actionable steps to help reduce certain risks from happening.

See below the risk table with a plan for mitigating, monitoring and managing each one.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk | Prob in % | Impact | Risk Mitigation, Monitoring, and Management (RMMM) Plan | Status |
| Web application link not accessible | 30% | High | Mitigation & Management: Code reviews; identify the bug(s); Review deployment environment, check Procfile, etc. (e.g Heroku). | Integrate and ensure CI/CD to fully test changes before it hits production |
| Disengagement | 10% | Medium | Mitigation & Management: Have daily standups and access engagement. Talk to the team member privately, then to the instructor if problem is not resolved. | Encourage high involvement |
| Delay in deployment | 60% | Medium | Mitigation & Management: Emergency meeting to assign immediate work for finishing deployment  Monitoring: keep a close eye on deadlines and project completion | Schedule mob coding session |
| Customer discovers many defects | 50% | High | Mitigation & Management: Find all bugs through code test coverage reports and fix them before next release.  Monitoring: # of defects discovered per release | Schedule mob coding sessions |
| Test coverage isn’t sufficient | 50% | Medium | Mitigation: Research and identify good tests to implement. Management: If bugs occur and aren’t caught by tests, go back to test case development. Monitoring: Make sure test coverage is complete/nearly complete | Write tests using SimpleCov and RSpec |

See example below:



## Release Planning

Release planning was discussed throughout our SCRUM meetings.

## Sprint Planning

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20\_APPSO-Parent-Portal\_-20220907\_150741-Meeting%20Recording.mp4?web=1

## Scrum Meetings

Scrum Meeting Recordings:

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20_APPSO-Parent-Portal_-20220907_150741-Meeting%20Recording.mp4?web=1>

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20_APPSO-Parent-Portal_-20220909_140928-Meeting%20Recording.mp4?web=1>

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20_APPSO-Parent-Portal_-20220913_163142-Meeting%20Recording.mp4?web=1>

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20220914_150858-Meeting%20Recording.mp4?web=1>

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20_APPSO-Parent-Portal_-20220916_141410-Meeting%20Recording.mp4?web=1>

## User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **User Story & Acceptance Criteria** | | | | |
| **#** | **Persona** **(*Who)*** | **Requirement*(What)*** | **Value**  **(*Why)*** | **Critical?**  ***Yes* / *No*** | |
| 1 | APPSO Member | I need my information represented in an organized way... | ...so that it can be viewed and manipulated efficiently. | X |  |
|  | **Acceptance Criteria:** | | | | |
| 2 | APPSO Officer | I need to have a portal website that I can access and use... | ...so that I can keep track of org activities. | X |  |
|  | **Acceptance Criteria:** | | | | |
| 3 | APPSO Member | I want to be able to fill out a member sign-up form... | ...to be officially part of the organization and have my record and have my record on file for future events/scholarships. | X |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | **User Story with its Acceptance Criteria** | | | | |
|  | **Persona (Who)** | **Requirement (What)** | **Value (Why)** | **Critical?** | |
|  |  |  |  | **Yes** | **No** |
| 1 | APPSO Member | I need my information represented in an organized way | So that it can be viewed and manipulated efficiently. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * An artifact is produced (image or otherwise) displaying database design which contains all PSQL data required. * Database successfully connects with the web application |  |  |  |
| 2 | APPSO officer | I need to have a portal website that I can access and use | So that I can keep track of org activities | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * A running basic front-end that can be accessed online * All the required tools used * Successfully went through heroku pipeline workflow * Successfully followed github workflow |  |  |  |
|  |  |  |  |  |  |
| 3 | APPSO member | I want to be able to fill out a member sign-up form | To be officially part of the organization and have my record and have my record on file for future events/scholarships. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * Members can create a unique login and password * Form is easily seen and accessible from the existing members' login screen * Members cannot create an admin account without permission from an officer |  |  |  |
|  |  |  |  |  |  |
| 4 | APPSO member | I need to be able to log into my member account | So that I can view my participation data and scholarship application |  | X (not for this sprint) |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * Members can log in with their username and password * Members can recover their passwords via email if forgotten or lost * Members are not able to login as officer |  |  |  |
|  |  |  |  |  |  |
| 5 | AAPSO officer | I need to be able to log into an admin account | So that I can view membership information and scholarship records |  | X (not for this sprint) |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * Officers can visually see the Google log-in page when asked to sign in (with Google) * Officers can login with their Google [tamu.edu](http://tamu.edu/) account. * Members are not able to login as an officer. * All Google-related test cases are handled by Google OAuth. |  |  |  |
|  |  |  |  |  |  |
| 6 | APPSO officer | I would like to see a list of members | So that I can know who is in the organization. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * Officer can see an entire list of members. * List is sorted by alphabetical order (last name) * Only Valid letters for first/last names are displayed (no special characters, symbols, etc.) * List is centered in the screen |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Acceptance criteria or ‘conditions of satisfaction’ indicate when our team has been successful in implementing the user’s requirements, stated as user stories

Acceptance Criteria Goals:

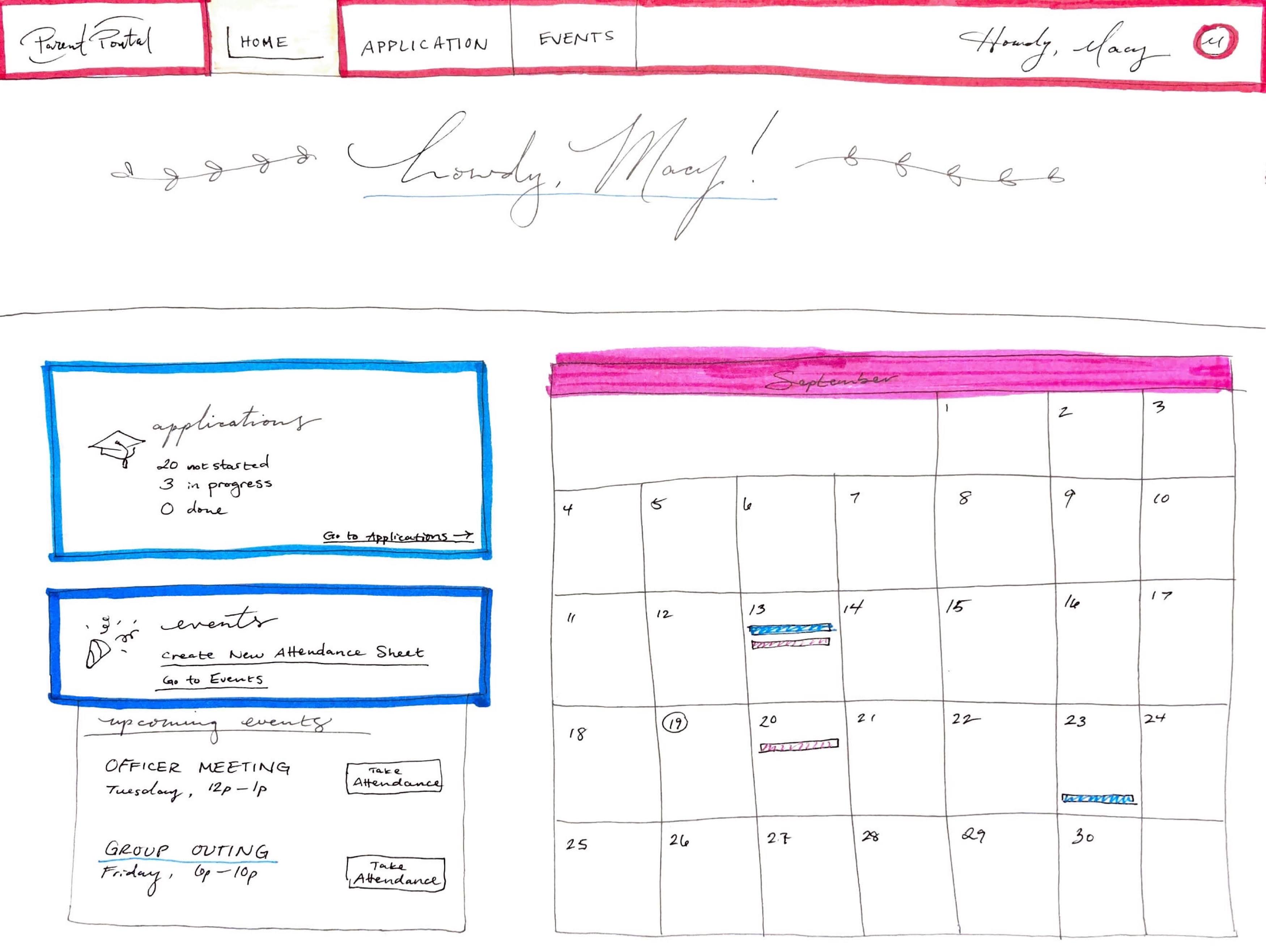
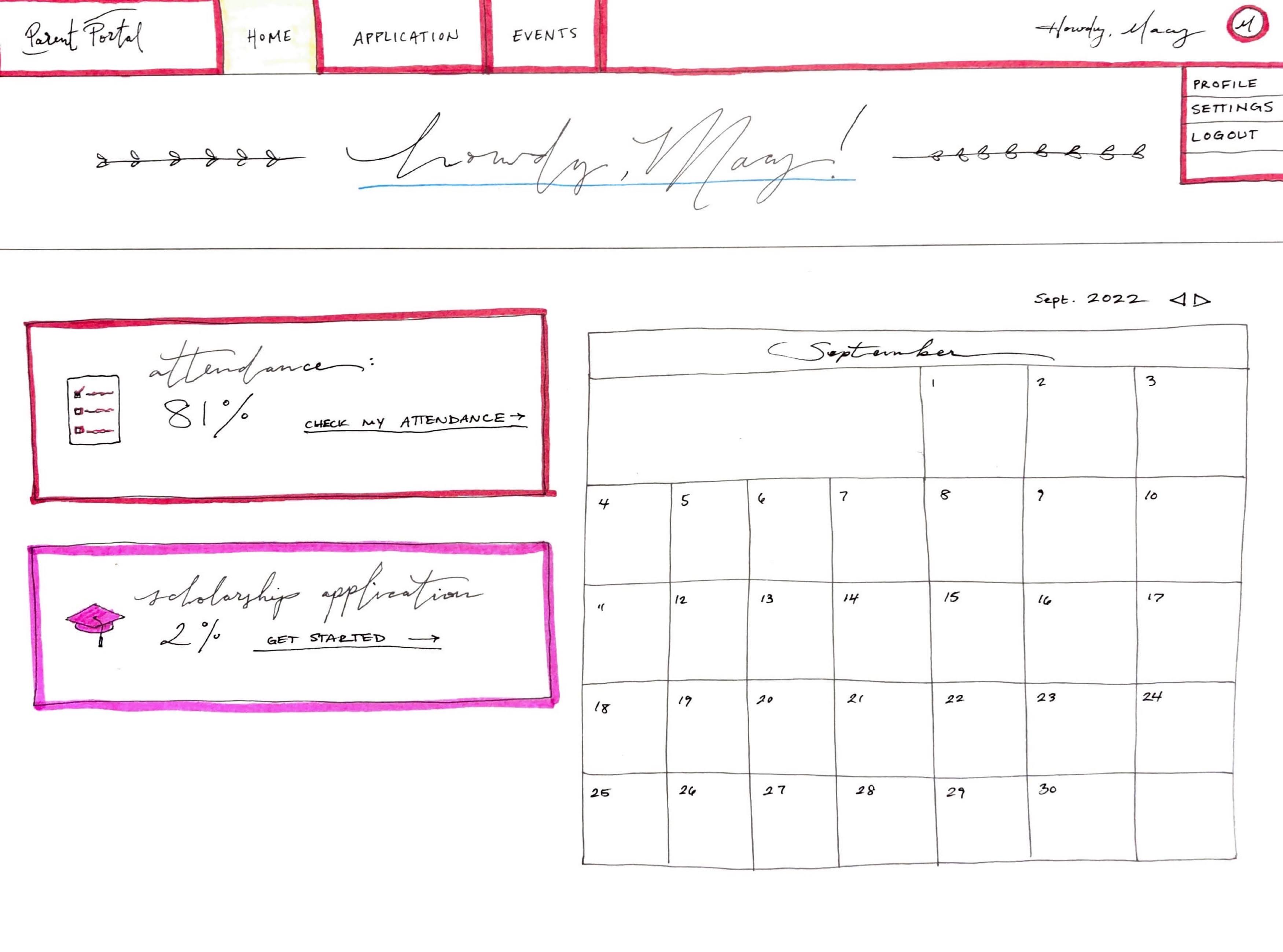
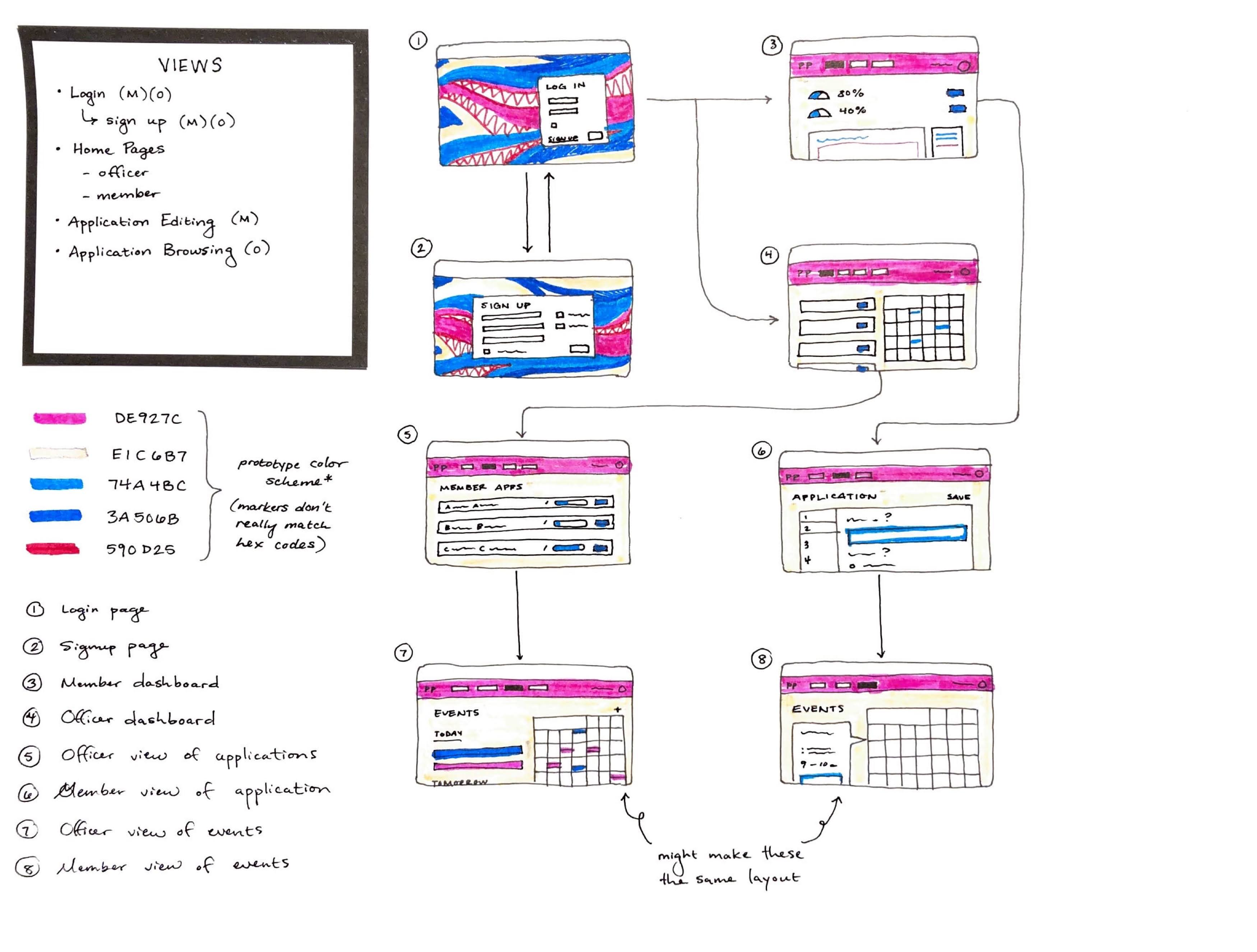
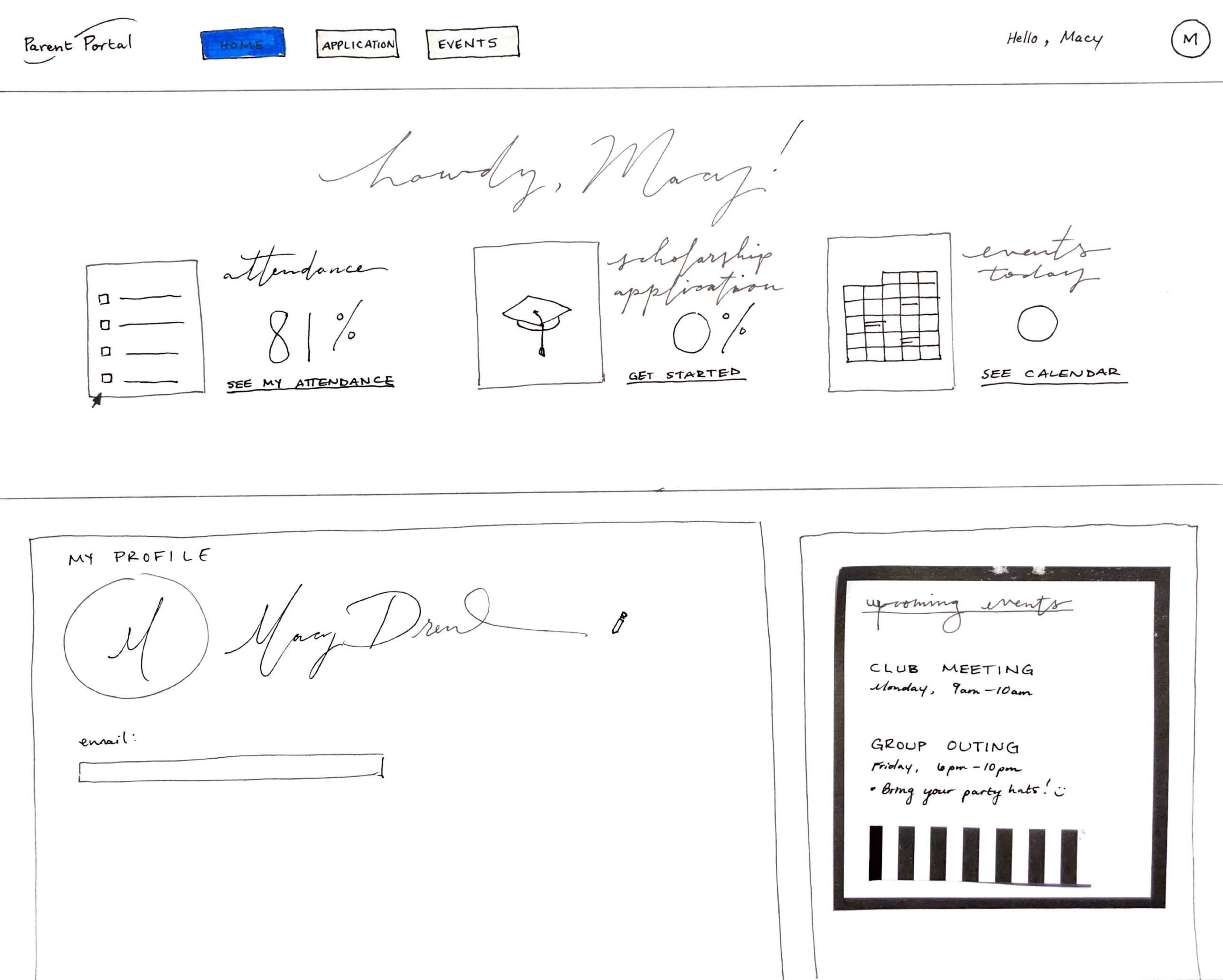
* To clarify what the team should build before they start work
* To ensure everyone has a common understanding of the problem
* To help the team members know when the story is complete
* To help verify the story via automated tests

Should include:

* Negative scenarios of the functionality (rainy day)
* Functional and non-functional use cases. Example of a non-functional use case is performance requirements
* What system/feature intends to do
* End-to-end user flow
* Impact of a user story to other user stories (i.e., features)
* UX concerns

The usage of acceptance criteria within our project allowed us to clearly visualize and see what is needed for such a feature to be filled. It helps translate our foggy idea of a feature into something concrete and actionable. That level of clarity is so important, especially in a six-person team. However, we were not perfect in fully adhering to the acceptance criteria. The time constraints for this sprint made us skim over some acceptance criteria and not provide full coverage for others. So, one thing that we could do better is ensure we have more reasonable acceptance criteria that we can cover in a timely manner. I’m glad we learned this lesson early on, as it will allow us to fulfill user stories faster.

## UX Models

## Data Design

A relational database was used for the project using PostgreSQL DBMS for Heroku. To ensure completeness of   
 the data model, and assess it for quality, an entity relationship diagram was developed.

Difficulties: effectiveness of SQL relationships, how to design them, what we needed, required 6 tables

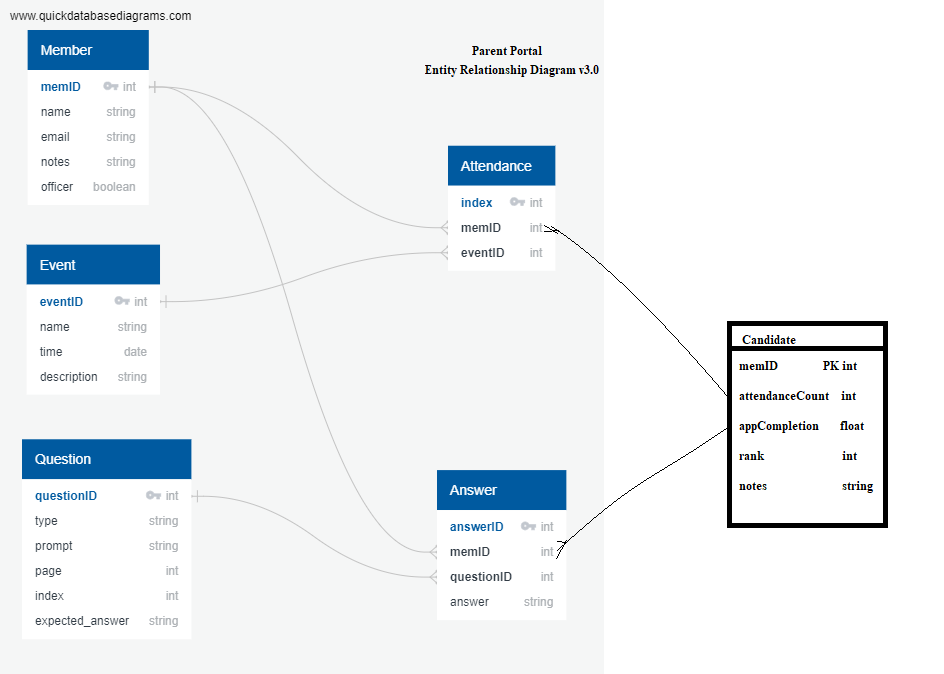
It was certainly not a quick process to develop our data design. We knew what our customers required at a high level, but thinking about how to specifically implement the relationships between data took some time and deliberation. It took four team members, a white board, and a couple of hours to develop this, and then some.

Initially, we knew that we needed to be able to store information about users’ attendance to events and allow them to apply for scholarships.

At first, we came up with a table for Officers, a table for an Application, and even creating as many tables as required to have a table for each individual user. Then, we researched more and argued productively about what is required, possible, and efficient for PostgresSQL operations. We looked at how other people in the past had organized data for something like a questionairre. In the end, we came up with a good idea of how everything can be related together correctly, making rows given new data instead of new columns or new entire tables.

Also, we received the instruction that we required an additional table, since our team consists of 6 members, and so we needed to have 6 tables in our database. I’m unsure about the usefulness of this correlation, but we then had an emergency meeting in order to come up with the Candidate table, which can allow for a summarized, pre-computed table of useful information about each user based on all the other tables.

See an example of the data design below:



While I reflected upon the process of data design thoroughly already, I’d like to reflect further. I thought it was a great experience for us to brainstorm actively about how to give our customers the best possible application without security vulnerabilities or performance issues caused by bad database design and management. We problem-solved on-the-fly and came up with great tables, which were able to be easily added as scaffolds to our ruby on rails application. Overall, database design is not a one-and-done process. It helps to have other team members to find problems or edge cases with the design to perfect it.

## Version Control

In software engineering it is necessary, especially in big projects, to track and control the source code developed. In our project, we decided to use GitHub, which offers functionalities, such as integrated issue tracking, collaborative code review, team management, and highlighting of syntax. It allowed us to work on separate features of the application, track bugs, and manage coding tasks.

We configured our GitHub repository to allow for maximizing productivity and organization. We have three main branches, dev, test, and main. We, however, mainly create branches when creating new features to keep more organization and separation. It allows us to determine what code is being merged into the main branch at the end of the day. We have Nathan and Frank conducted pull request reviews, to ensure code touching the main branch is clean, follows code guidelines, and addresses user stories. GitHub version control truly made us feel like we were in a professional team and had our own division of labor in a way. What we could do differently however is implement GitHub workflows to have better code coverage checking, like how the setup was with the lab assignments.

## Quality

Quality was assessed using the following quality metrics: Compliance, Efficiency, Correctness, Maintainability, Integrity, and Usability. Although there are many more metrics, these were the ones recommended for the class.

### Compliance

Compliance of requirements was assured by creating sub-tasks in each Task in Jira. An added description was extremely helpful, as it helped us understand what exactly the specific task needed to accomplish. These tasks were then weighed in a way that we thought the client would order from highest priority to least priority.

### Efficiency

To start off, at the beginning of the semester, we assigned tasks to each member in the team to fit liking. However, as we worked increasingly on the project, some of us started to get busy with homework and studying for other classes. This led to some having more time than others. To be as efficient as possible, we communicated to each other in what tasks we were able to help those in need. This made Jira extremely helpful because we could all see each other's pending/done tasks.

### Correctness

For this sprint, we based correctness on functionality. Rather than focusing on how our Web App looked, we instead focused on how the Web App functioned. Things like test cases were implemented, as well as deployment to Heroku. Currently, we have a working sign up and login system with no passwords saved into our database that allows a user to create their account and access it properly.

#### Test-Driven Development (TDD):

This technique involved writing unit/integration tests before coding, and involves the following:

* The developer writes an (initially failing) automated test case that defines a desired outcome;
* He/she produces the minimum amount of code to pass the test;
* Refactors the new code according to best practices, or to make it more readable.

What we learned:

* These tests are quite tedious to write versus just going straight to coding, however we found the benefits included clean code, less bugs/errors, better understanding, better communication, and overall, more cohesion as a team
* Before the lab assignment, we as a team had little to no experience with creating unit and integration tests for full-stack applications made with Ruby, so for Sprint 1 we didn’t have a concrete foundation in creating unit/integration tests.
* In the moments that we did include basic unit/integration tests, it worked well to quickly determine whether basic functionality worked. It allowed us to cut down time on finding out errors that we would normally fine in live testing. The short time duration of Sprint 1 limited our ability to create more unit/integration tests. However, what we could do better is create tests ahead of time and more frequently in order to “cover our tracks more”. Moving onward, we plan on learning what we applied in the lab assignments to create more unit/integration tests for features we implement.

#### Reviews:

Code reviews (and other technical reviews such as pair programming) were a big part of ensuring quality and involved pull requests from individual branches into our main branch. Descriptive naming conventions helped insure both the program and reader understand what fixes were made and what issues persist.

What we learned:

* Code reviews allowed us to discover defects related to merge conflicts and verify that we followed the different standards set forth in the project, including peer reviewing and peer merge confirmations.
* For this sprint, we based completeness on whether the Web App functioned properly or not. We based complete branches on branches that got their job done in accordance with their name.

#### Definition of Done

The "Definition of Done" is a guide to determining completion of a user story and its tasks.

To ensure quality for the client, we made sure that the tasks we were completing were fully functional and didn’t conflict with other tasks that might be affected.

Below is the project’s “Definition of Done.”

N/A: Did not have sprint review and retrospective yet at the time of due date submission.

| Sprint Definition of Done Criteria | Objective | Verified (Y/N) | Notes: |
| --- | --- | --- | --- |
| Data Design complete | Maintainability | Y |  |
| All changes merged to Main | Efficiency | Y |  |
| Any configuration or build changes documented | Efficiency | Y |  |
| Sprint Review conducted | Compliance | N/A |  |
| Client Feedback given during Sprint Reviews. During the last sprint, feedback can be gathered from the Acceptance Test Results and/or customer feedback survey | Compliance | N/A |  |
| User stories delivered in sprints represented value to the customer (i.e., most valuable delivered in earlier sprints) | Compliance | Y |  |
| Working product delivered (not just a prototype) | Compliance |  |  |
| Sprint deliverable reviewed by the product owner | Compliance | Y |  |
| Sprint Scrum meetings conducted on a daily basis | Efficiency | N | Conducted MWF |
| Sprint documentation generated (meeting the criteria in the project rubric), including key scrum artifacts (e.g., sprint backlog, etc.) | Maintainability | Y |  |
| Peer feedback submitted | Efficiency | N |  |
| Sprint Retrospective Conducted | Maintainability | N |  |
| Plans (e.g., stakeholder management, risk plan, etc.) were discussed at the sprint retrospective and improvement identified (when applicable) | Efficiency | N/A |  |
| Key performance indicators (especially those related to each objective whether it be group or individual performance) were measured and discussed at the sprint retrospective | Efficiency | N/A |  |
| Assessed risks to make sure that high impact and high probability risks are mitigated, monitored, and managed. | Compliance | Y |  |
| Second to the last sprint: Deployment, support, and maintenance plans submitted and approved | Maintainability | N/A |  |
| Final Sprint only: All pending issues / user stories resolved | Compliance | N/A |  |
| Final Sprint only: Project turnover items prepared | Compliance | N/A |  |

| User Story Definition of Done Criteria | Objective | Verified (Y/N) | Notes: |
| --- | --- | --- | --- |
| Code review performed | Correctness | Y |  |
| 100% code coverage using automated test tools | Correctness | N |  |
| 100% test coverage (i.e., tested against all test cases) | Correctness | N |  |
| Meets conditions of satisfaction for the user story (i.e., acceptance criteria), as communicated by the customer during sprint reviews (did we build the "right" product?). During the last sprint, satisfaction can be measured from the user acceptance test form and/or the customer feedback survey | Compliance | Y |  |
| App reviewed by the product owner on a continuous basis | Compliance | Y |  |
| End-user documentation is ready (should be delivered incrementally during each sprint, with the complete version delivered during the last sprint) | Compliance | N |  |
| Code refactored (as agreed upon by the team) | Maintainability | Y |  |
| No fatal ‘code smells’ in code analysis output (e.g., Rubocop) | Maintainability |  |  |
| Meets Coding Standard | Maintainability | Y |  |
| All Unit Tests passed | Correctness | N |  |
| All Integration tests passed (including system test when all components integrated) | Correctness | N |  |
| Tests on all supported devices & browsers passed | Correctness | N |  |
| All Integrity testing passed | Integrity | N |  |
| Usability tests passed (should be based on acceptance criteria of user) | Usability | Y |  |
| All Performance testing passed to ensure minimum response time met | Correctness | N |  |
| All Regression Tests passed | Correctness | N |  |
| All Validation Tests passed (Did we build the "right" product?) using real data for testing; especially critical/mandatory user stories | Compliance | N |  |
| Any configuration or build changes documented | Maintainability | Y |  |
| Deployed live to customer | Efficiency | Y |  |

#### Test Coverage

Evaluating test coverage is one of the methods that indicate how well the code was tested.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| “As an APPSO officer, I need to have a portal website that I can access and use so that I can keep track of org activities.” | Website Url | Home page showing to login or logout | Sunny | Website should not show a heroku error or default rails error. |
| “As an APPSO member, I need my information represented in an organized way, so that it can be viewed and manipulated efficiently.” | New entry example | Entry being shown in the index | Sunny | We are testing basic CRUD functionality to ensure database functionality. |
|  |  |  |  |  |
| “As an APPSO member, I want to be able to fill out a member sign-up form, in order to be officially apart of the organization and have my record on file for future events/scholarships.” | Required registration information | Confirmation that information has been logged | Sunny |  |
|  | Incorrect information that doesn’t follow requirements | Warning that information does not fulfill requirements | Rainy |  |
| “As an APPSO member, I need to be able to log into my member account so that I can view my participation data and scholarship application” | Correct/ valid username/password | Confirmation that login is successful, can go to home page | Sunny |  |
|  | Invalid user/pass combo | Alert that it is incorrect username/password | Rainy | Ensure that the error code does not say which one is wrong (whether only username or only password is wrong), always mention both) |
| “As an APPSO officer, I need to be able to log into a admin account so that I can view membership information and scholarship records.” | Correct/ valid username/password | Confirmation that login is successful, can go to home page | Sunny |  |
|  | Invalid user/pass combo | Alert that it is incorrect username/password | Rainy | Ensure that the error code does not say which one is wrong (whether only username or only password is wrong), always mention both) |
| “As an APPSO officer, I would like to see a list of members so that I can know who is currently in the organization as a whole”. | Clicked link to access member list | Full list of members (if any) | Sunny |  |

Example below:



Guidelines for identifying edge cases:

|  |  |
| --- | --- |
| **Types of Valid Input** | **Edge cases that are candidate inputs to be tested** |
| Range (e.g., 0 to 100). Example a range of valid area codes | •1 valid (e.g., 0 to 100) |
|  | •2 invalid equivalence classes (e.g., < 0 or > 100 ; null) |
| Specific value (e.g., Y,N) | •1 valid (e.g., Y, N) |
|  | •2 invalid equivalence classes (e.g., not Y, not N, null) |
| Member of a set (e.g., set of majors in the college) | •1 valid (e.g,. CPSC) |
|  | •1 invalid equivalence class (e.g., BIMS) |
| Boolean | •1 valid (e.g., true) |
|  | •1 invalid equivalence class (e.g., false) |

We didn’t emphasize too much on the data we have during our testing phases, as we had little to no time to fully implement all the unit, integration, and validation tests needed for our FIRST sprint. The lab that was due a few days before the Sprint 1 deadline would’ve helped a lot in our ability to create data to be used in different test phases, but our focus for the sprint was to get something up and running for our customer to view. We did not expect the huge time commitment in configuring our development environment, while also learning CRUD functionality on a whole new programming language and development platform. We happened to use rails standard functionality of automatic integration tests for CRUD functionality amongst all entities. In the future, we plan on planning our test cases before we code our user stories so that we can have better organization in our user story completion flow, from idea, to coding, and through proper tests. After completing all the labs, we believe we are in a better position to iron out our inexperience in conducting test cases and implementing tests to fulfill such test cases.

##### Code Coverage - Output of simplecov

A tool we used to determine code coverage was Simplecov which helps find areas in our code that were tested and not tested.

See below Simplecov output for our application.

[SimpleCov.html](https://tamucs.sharepoint.com/:u:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%201/Documents/SimpleCov.html?csf=1&web=1&e=df1gqr)

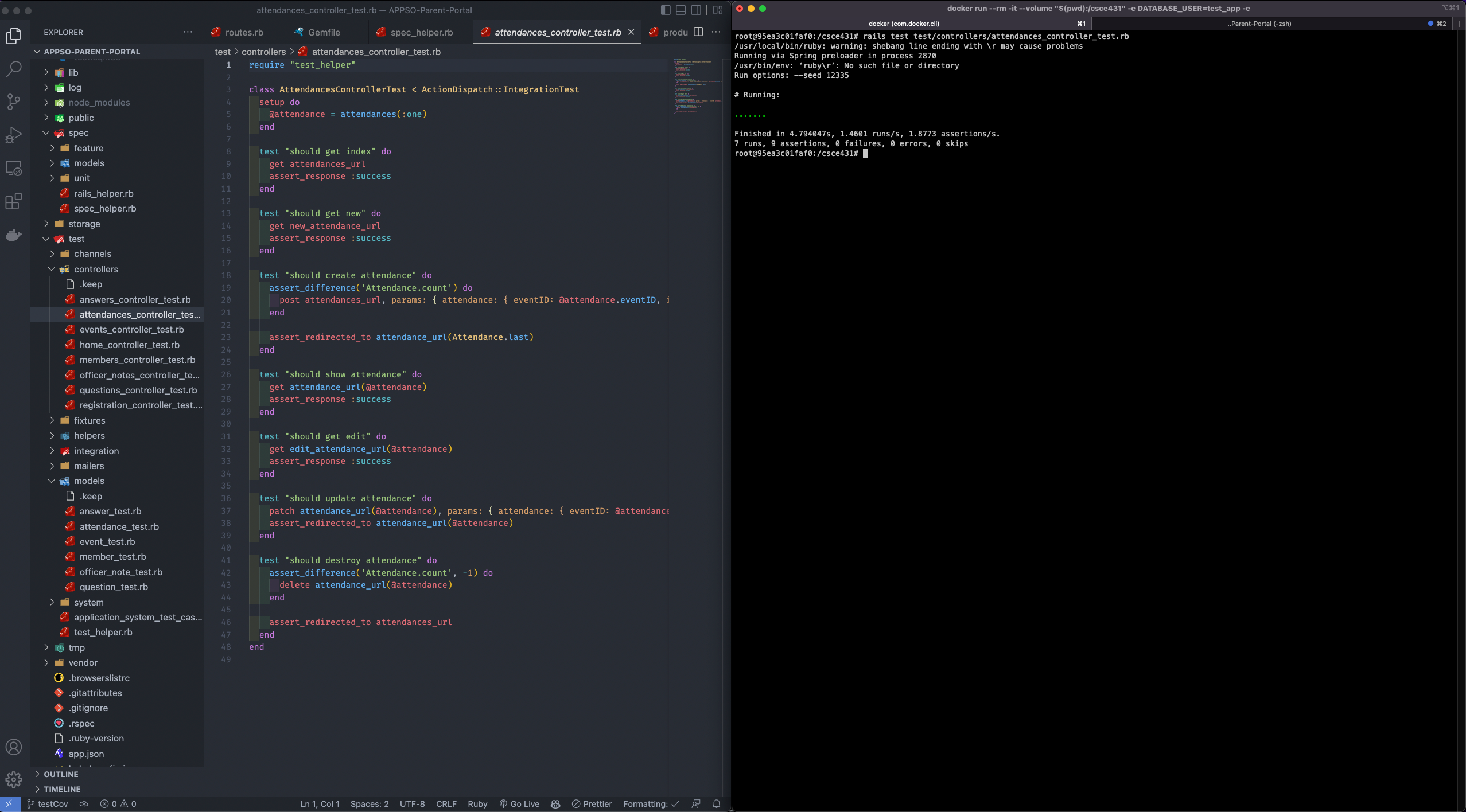
SimpleCov is such a GREAT tool for analyzing what parts of our code have been fully covered and what code has not. It essentially helps remove the long manual process of ensuring almost every single line of code has been covered in some fashion and has run within the application. SimpleCov helped us show that test coverage doesn’t have to be that painful. Moving onward, we will use SimpleCov more as we implement more features/user stories.

#### Test Results

Prior to releasing to the customer, we tested the app thoroughly, to ensure that bugs do not escape to the customer.

The test results created for each entity within the database were all automatically tested for CRUD functionality and passed. We were not able to add RSpec and other unit/integration tests that fully cover features we implemented, but we plan on doing this the next sprint.

Here is a rep. Sample for attendance CRUD functionality:



https://appso-portal-test-6cfiq5gqj4jx.herokuapp.com/

### Maintainability

One of the approaches we used to fulfill this requirement of producing readable code (especially for any teams that will be enhancing our code base) is the use of a code style checker (e.g., Rubocop) to detect code smells such as excessive nesting of conditional and looping constructs, methods with too many parameters, layout, etc.

The team helped ensure maintainability by closely monitoring the time it took to handle pull requests, response to complications with deployment, and bug within the web application itself. Since it was our first sprint, our mean time to deploy fixes was quite long, as we were still getting used to the development environment and Ruby on Rails. We also closely monitored the the sprint backlog to determine response time and completion time for user stories.

#### Coding Standard

We also reviewed our code against guidelines of clean code as specified in our coding standard, which is included below.

|  |
| --- |
| **Ruby on Rails coding standards** |
| Go easy on comments. If the code is obvious, don’t comment. Remove old, commented code |
| Use two-space indentation |
| Use each instead of for. Use unless instead of !if. However, if you need to involve an else to your conditional, never use unless-else. Use until instead of while ! (negated condition). |
| Use meaningful variable names. |
| Use snake\_case for methods and variables. Use CamelCase for classes and modules. (Keep acronyms like HTTP, RFC, XML uppercase.). Always name your methods based on their behavior, not implementation. |
| The names of predicate methods (methods that return a boolean value) should end in a question mark. Avoid prefixing predicate methods with the auxiliary verbs such as "is," "does," or "can." e.g., person.tall? |
| Instance variables are defined using the single "at" sign (@) followed by a name. It is suggested that a lowercase letter should be used after the @. |
| Global variable starts with a dollar ($) sign followed by other characters. |
| Constants should be all upper case with words separated by underscores ('\_'). |
| Table names have all lowercase letters and underscores between words; all table names must be plural noun, e.g. invoice\_items, orders, etc. |
| The model is named using the class naming convention of unbroken MixedCase and is always the singular of the table name, e.g. if the table name might be orders, the model name would be Order. |
| Controller class names are pluralized, such that OrdersController would be the controller class for the orders table. |
| The primary key of a table is assumed include the word "id" e.g., order\_id |
| The foreign key is named with the singular version of the target table name with id appended to it, e.g, order\_id in the ITEMS table that links to the order\_id in the ORDERS table. |
| Tables used to join two tables in a many to many relationship is named using the table names they link, with the table names in alphabetical order, for example ITEMS\_ORDERS |
| Skinny Controllers, Fat models: best practice is to keep non-response related logic out of the controllers. Examples of code you don’t want in a controller are any business logic or persistence/model changing logic. |
| Views should have very little ruby in them and certainly shouldn’t interact with the data repository (e.g., databases). |
| Ternaries (?:) are good if they fit on one line (remember the short lines rule). |
| Use def with parentheses when there are parameters. Omit the parentheses when the method doesn't accept any parameters |
| Convention over Configuration - try to use the Rails defaults when you can |
| Do not repeat yourself (DRY). Do whatever it takes to make sure that you don’t repeat yourself, avoiding duplication as much as you can. For example, use abstract classes, modules |
| Smart use of Enums |
| Use db:schema:load when creating the application database on a new system. Use db:migrate in all other cases when you need to apply the newly added migrations. |
| Nested Resources/Routes: If you have a resource which belongs to another resource, then it’s a good idea to define the routes of the child resource nested within the routes of parent resource. |

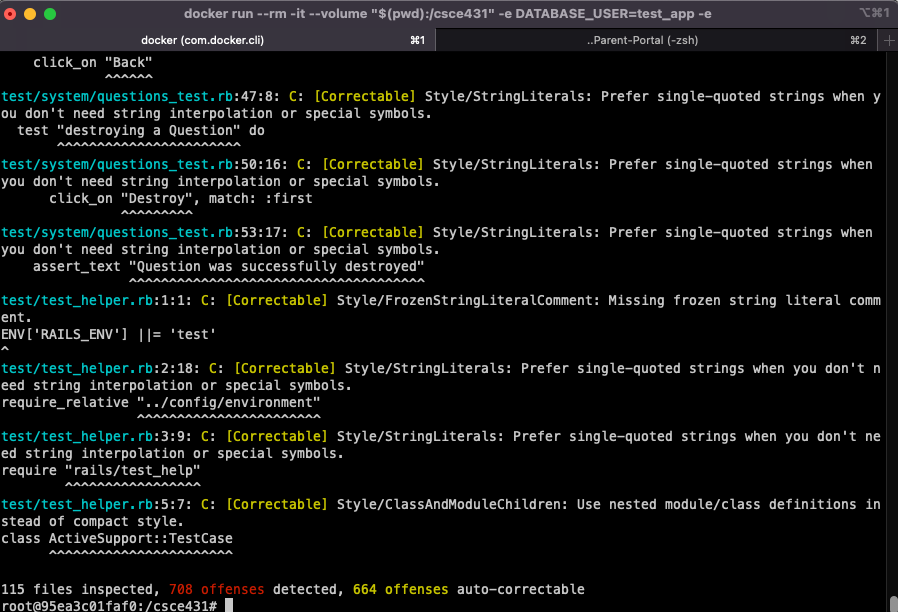
Adhering to a coding standard allowed us to have cleaner and most importantly, readable code. Keeping the style and organization of code to a standard allows us to focus more on functionality and implementation rather than understanding how a piece of code that is typed non-professionally.

#### Well Documented Code

Proper branches were made to keep our code as clean as possible from merge conflicts. They also allowed for us to work on different tasks at the same time from our own machines. Naming standards were used on branches in order to quickly identify the feature and who was working on it. Descriptive commit messages were used for ourselves, and others to understand what the goal of the commit was. Making clean comments, where necessary, allowed for peers to be able to understand and follow thought process. Documentation in a timely and neat manner benefited us when we hit roadblocks such as being unable to deploy to Heroku automatically.

#### Linter Output

See below the output from the linter (e.g., RuboCop).



Linting provides a good way of ensuring that the code we write is up to par or code standard with the language. Ensuring that there is no type error or any error in relation to how the code is typed. It can also look for and catch typos that developers may have accidentally typed. Using RuboCop allowed us to identify parts of our code where we could improve upon code standard wise.

### Integrity

We helped ensure integrity by ensuring that rails security features are enabled by default and ensuring brakeman can work with our application. We went through our RMMM plan and planned to follow all the mitigation action items. We were not able to fully ensure integrity as we did not have time to fully learn security implementations for rails.

#### Security Risk Analysis Table

The security-risk analysis table is shown below which includes common vulnerabilities in our application. Strategies to address each vulnerability is in the RMMM column.

We created our security risk analysis by closely analyzing fields and other text entries that malicious users would exploit. We put ourselves in the mind of an attacker to fully understand risks such as SQL injection and XSS scripting.

|  |  |  |  |
| --- | --- | --- | --- |
| Security Risk | Probability (3=high, 1=low) | Impact | Risk Mitigation, Monitoring, and Management (RMMM) Plan |
| SQL injection | 3 | 3 | Risk Mitigation & Management: Disallow angle bracket, other suspicious script in text boxes  Monitoring: |
| Weak Authentication | 2 |  | Risk Mitigation & Management: Require two-factor authentication or use Google authentication  Monitoring: |
| Member access to officer view/information | 1 | 3 | Risk Mitigation & Management: Inform the admin of the risks that users might be impersonating officers. Authenticate officer account applications.  Monitoring: |
| XSS Scripting | 3 | 3 | Risk Mitigation & Management: Disallow angle bracket, other suspicious script in text boxes; use input filtering client/server side.  Monitoring: |
| Member financial information leaked from scholarship app | 2 | 3 | Risk Mitigation & Management: Protect database  Monitoring: Brakeman (?) |

#### Authentication

Google Authentication is the primary source for where we want to have users authenticated from. However, we have not yet reached the point where we can use OAuth for the users.

#### Integrity Test Coverage

Evaluating integrity test coverage is one of the methods that indicate how secure your system is from attacks.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Some representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| “As an APPSO member, I want to be able to fill out a member sign-up form, in order to be officially apart of the organization and have my record on file for future events/scholarships.” | SQL command in one of the fields that drops all tables | Filters out sql command input, and returns an error for incorrect fields | Rainy |  |
| “As an APPSO member, I need to be able to log into my member account so that I can view my participation data and scholarship application.” | javascript code snippet that collects information about the server | Filters out such code and report an invalid field error | Rainy |  |

Basic data that tests general functionality were used for the different test phases. Like how the lab unit/integration tests worked. We did not have many tests implemented for integrity other than using brakeman for vulnerabilities

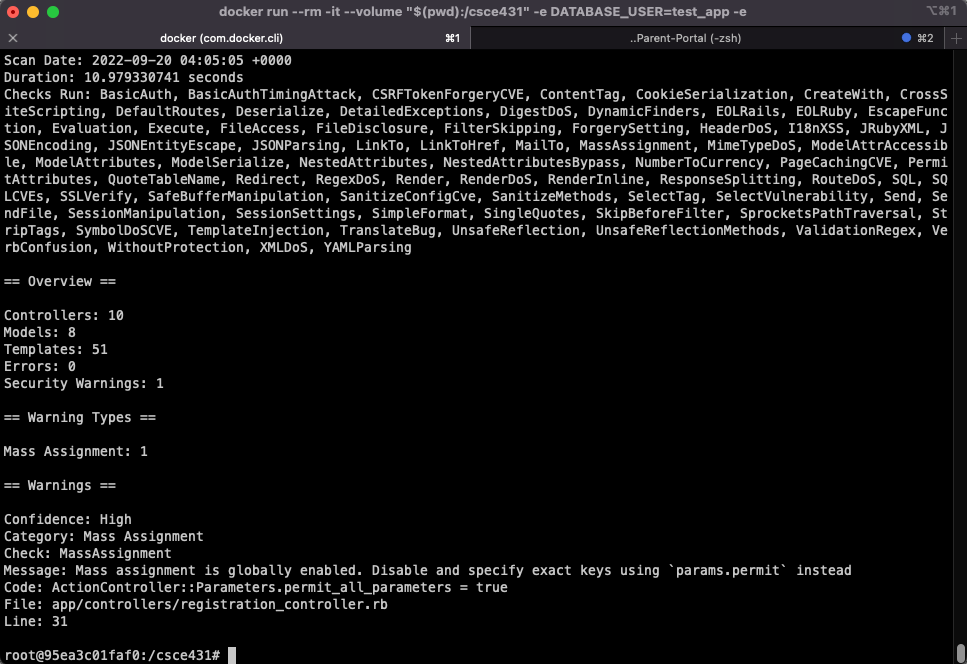
The process of creating test cases to ensure security breaches are not present was straightforward. Some of the members of the team have general knowledge regarding potential security breaches via XSS scripting and SQL injections so we were able to create basic test cases for such scenarios.

#### Integrity Test Results

We ensured that our RMMM plan worked to address common security vulnerabilities by trying to login with JavaScript code in the fields. Inputting SQL commands within the fields.

See below representative output of any integrity related tests (e.g., actual RSpec tests, use of tools such as Brakeman).

Brakeman test:



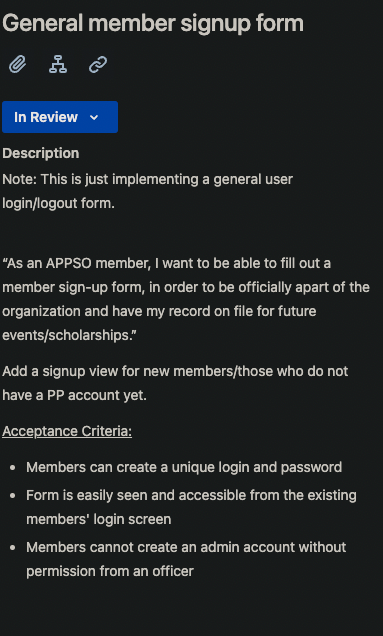
It looks like we did a decent job in our test results, as we only have 1 security warning. As we add more features however, this may change.

### Usability

We helped ensure usability by partaking in general user interaction flows. We checked to make sure that we can sign up, sign in, and log out. We also made benchmarks and times to estimate how usable the web application can be in a duration of time. Not much attention was paid to creating usability metrics but they will be improved upon in future sprints.

#### Usability Requirements

The team first gathered user requirements related to "usability" and reflected them as both user stories and ‘usability’ acceptance criteria within user stories. Some examples below:



#### Usability Test Coverage

Determining sufficient test coverage is one of the methods for ensuring usability of the system.

To ensure that the most common scenarios were tested, both sunny and rainy day, we documented our test cases. Some representative test cases are shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| “As an APPSO member, I want to be able to fill out a member sign-up form, in order to be officially apart of the organization and have my record on file for future events/scholarships.” | Incorrect user/password | Alert of incorrect user/password combination, allow user to EASILY re-enter credentials | Rainy |  |
| “As an APPSO officer, I need to have a portal website that I can access and use so that I can keep track of org activities.” | User enters url in browser | Home index shows up | Sunny |  |

We didn’t really use data to test the usability of the system. We basically interacted with the app from the perspective of the user in order to fully determine the usability of the system.

This process was simple, as we simply just interacted and played around with different ways of interacting with the web application and reported our values on how usable it is.

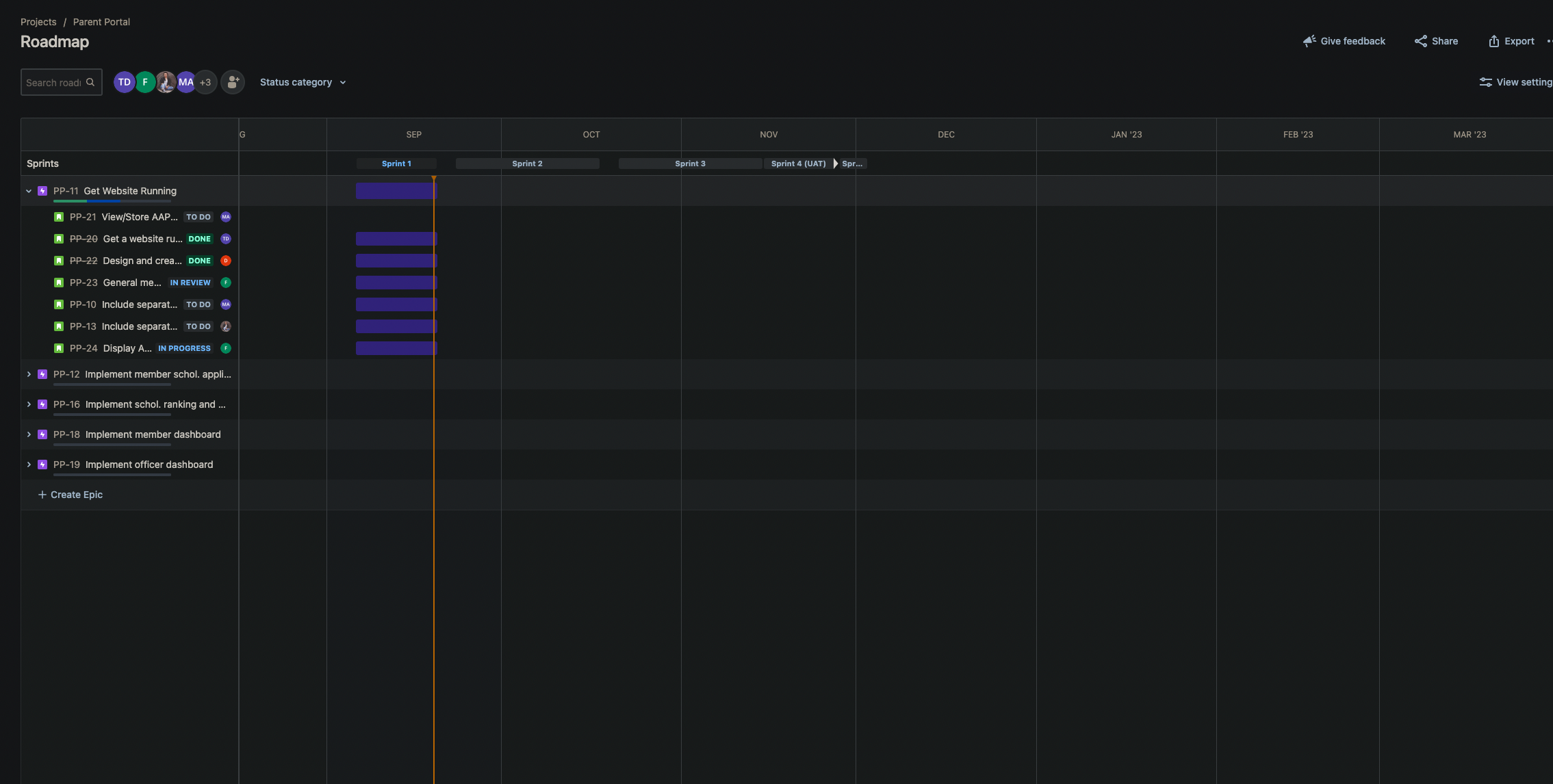
#### Usability Test Results

We ensured that our system was usable by interacting with the application from the perspective of the user.

We implemented and exercised tests like in the table above and created a scale on how usable it was. Based on our limited functionality, the user login, logout, and registration flow were quite usable after evaluation from each team member. We may plan on creating a more concrete process of measuring usability with other metrics.

## Monitoring and Control

The team monitored tasks assigned to teach team members and utilized SCRUM meetings and discord to check up on progress, and the overall direction of such tasks if they were not completed. Jira was a great tool for assigning work to each team member and monitoring the progress of the sprint over time.



Monitoring & Control helped us keep us in check for the project. It made us all aware of where we are in terms of the web application, and what has been done and not been done. It provided clarity on what was expected of us each day, and we all understood how our contributions and efficiency was being measured. Reviewing original plans frequently and updating them when needed was important, because at the end of the day, things don’t always go according to plan. There were times where we had to change plans on our prioritization for the first sprint, which made us do a reassessment of our stakeholder and risk plans, to ensure we are still aligned with such plans. In the future we could mention plan re-assessments in future SCRUM meetings in order to keep such plans up to date with our actual progress and ideas. We also could emphasize more on monitoring correctness, specifically in terms of unit/integration tests, which we did not have extensively in this first sprint.

Some sample key performance indicators for each of the different objectives are listed below:

* Efficiency: variance between expected and actual release date, individual performance (e.g., % of issues / member, resolved and unresolved issues at the end of each sprint that may need to be deferred to a future sprint)
* Compliance - # of non-compliance to customer requirements mentioned during customer meetings, # of requirements change requests from the customer which may mean that original requirements were misinterpreted, % acceptance criteria met, % of user stories completed that were requested by the customer
* Correctness - test coverage, code coverage using Simplecov, # of code reviews, # of defects (errors discovered by customer), # of high severity errors discovered during later test phases such as last stages of integration test or system test (when all components integrated), # of errors deferred to a future sprint, etc.
* Integrity - # of security tests performed, # of defects related to security breaches, etc.
* Usability - # of negative feedback related to usability during sprint reviews, % compliance with acceptance criteria related to usability, % user stories done which are related to usability, etc.
* Maintainability - length of time for the team to fix a bug (developed by someone else), effort to release per sprint, statistics from code style checker (e.g., Rubocop), etc.

## Deploying App

Application deployment was configured via Heroku. Thanks to the Book Collection Assignment, we were able to learn how to configure Heroku with CI/CD, with test and staging apps ready for production. The backend team followed the same procedure as detailed in the labs. We, however, ran into issues with our database configuration, and realized we were using sqlite3 instead of PostgreSQL. So, we replaced sqlite3 with PostgreSQL, and it happened to work and got login/logout functionality working. Thierry and Frank were glad that they were able to get the application up and running in production. We are glad that we were able to get through our problems early in the first sprint, as it allowed us to learn how to troubleshoot and debug as a development team. It was really satisfying to fix such a substantial problem.

<https://stage-appso-portal.herokuapp.com/>

## Sprint Review

Not done at the time of writing, will submit as a separate document in MS Teams.

[Place links to the MS Teams recording here. Do not use Zoom]

[Evidence uploaded in MS Teams that issues are documented and highly likely to be addressed (e.g., screenshot of corresponding issue in Jira, or action plan using action plan template)

\*NOTE\* Please mention where to find this artifact.]

## Sprint Retrospective

Not done at the time of writing, will submit as a separate document in MS Teams.

[Place links to the MS Teams recording here. Do not use Zoom]

[Evidence uploaded in MS Teams that issues are documented and highly likely to be addressed (e.g., screenshot of corresponding issue in Jira, action plan using action plan template)

\*NOTE\* Please mention where to find this artifact.]