Change Log

|  |  |  |
| --- | --- | --- |
| Change | Date of Change | Author |
| Baseline | 9/24/22 | Pauline Wade |
| Finished up to Section 4.2 w/ exception of things in red | 10/7/22 | Frank Martinez |
| Finished up to Section 5.7 w/ exception of things in red | 10/12/22 | Frank Martinez |
|  |  |  |
|  |  |  |

Final Report

Parent Portal

Aggie Pregnant and Parenting Students Organization

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

CSCE 431 – Software Engineering  
Fall 2023

Texas A&M University

Department of Computer Science and Engineering

Contents

[1 Project Scope 5](#_Toc114937217)

[2 Stakeholder Analysis 6](#_Toc114937218)

[2.1 Stakeholders 6](#_Toc114937219)

[3 Implementation Environment 7](#_Toc114937220)

[3.1 Hardware 7](#_Toc114937221)

[3.2 Software 7](#_Toc114937222)

[4 Scrum Process 8](#_Toc114937223)

[4.1 Release Planning 9](#_Toc114937224)

[4.2 Sprint Planning 9](#_Toc114937225)

[4.3 Scrum Meetings 9](#_Toc114937226)

[4.4 Sprint Review 9](#_Toc114937227)

[4.5 Sprint Retrospective 10](#_Toc114937228)

[5 Sprint 2: 11](#_Toc114937229)

[5.1 Overall Experience 11](#_Toc114937230)

[5.2 Stakeholder Management and Communication Plan 11](#_Toc114937231)

[5.3 Risk Mitigation, Monitoring, & Management Plan 12](#_Toc114937232)

[5.4 Release Planning 12](#_Toc114937233)

[5.5 Sprint Planning 12](#_Toc114937234)

[5.6 Scrum Meetings 12](#_Toc114937235)

[5.7 User Stories 13](#_Toc114937236)

[5.8 UX Models 13](#_Toc114937237)

[5.9 Data Design 14](#_Toc114937238)

[5.10 Version Control 14](#_Toc114937239)

[5.11 Quality 15](#_Toc114937240)

[5.11.1 Compliance 15](#_Toc114937241)

[5.11.2 Efficiency 15](#_Toc114937242)

[5.11.3 Correctness 15](#_Toc114937243)

[5.11.3.1 Test-Driven Development (TDD): 15](#_Toc114937244)

[5.11.3.2 Reviews: 15](#_Toc114937245)

[5.11.3.3 Definition of Done 16](#_Toc114937246)

[5.11.3.4 Test Coverage 17](#_Toc114937247)

[5.11.3.4.1 Code Coverage - Output of simplecov 18](#_Toc114937248)

[5.11.3.5 Test Results 18](#_Toc114937249)

[5.11.4 Maintainability 19](#_Toc114937250)

[5.11.4.1 Coding Standard 19](#_Toc114937251)

[5.11.4.2 Well Documented Code 20](#_Toc114937252)

[5.11.4.3 Linter Output 20](#_Toc114937253)

[5.11.5 Integrity 20](#_Toc114937254)

[5.11.5.1 Security Risk Analysis Table 20](#_Toc114937255)

[5.11.5.2 Authentication 20](#_Toc114937256)

[5.11.5.3 Integrity Test Coverage 20](#_Toc114937257)

[5.11.5.4 Integrity Test Results 21](#_Toc114937258)

[5.11.6 Usability 21](#_Toc114937259)

[5.11.6.1 Usability Requirements 21](#_Toc114937260)

[5.11.6.2 Usability Test Coverage 21](#_Toc114937261)

[5.11.6.3 Usability Test Results 21](#_Toc114937262)

[5.12 Monitoring and Control 22](#_Toc114937263)

[5.13 Deploying App 22](#_Toc114937264)

[5.14 Sprint Review 22](#_Toc114937265)

[5.15 Sprint Retrospective 22](#_Toc114937266)

[5.16 Continuous Integration (CI) / Continuous Delivery (CD) 23](#_Toc114937267)

[5.17 Deployment & Support Plan 24](#_Toc114937268)

# 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.

Table

Description automatically generated

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
* Bootsnap
* rexml
* Byebug
* Web-console
* Rack-mini-profiler
* Listen
* Spring
* Capybara
* Selenium-webdriver
* Webdrivers
* Tzinfo-data
* omniauth
* omniauth-google-ouath2
* omniauth-rails\_csrf\_protection
* brakeman
* rubocop

# 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 and 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 spring report)
* [Risk Assessment & Plan](https://tamucs.sharepoint.com/:w:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%202/Documents/APPSO_Project%26SecurityRiskPlan.docx?d=w04c4add4a1c74bab9d9b3399a979844f&csf=1&web=1&e=LiKYgu)
* Metrics Document

Requirements

* User Stories with Acceptance Criteria
* UX Models

Design

* [Data Design](https://tamucs.sharepoint.com/:w:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%202/Documents/APPSO_DataDesign.docx?d=wc4e99081bb004096bdbb921bfd9a33e8&csf=1&web=1&e=OMogaN)
* 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 got a lot better at implementing them in the second, as we were much more familiar with Rails by that point. 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. We had some difficulty with this in the first sprint, but did much better at it this time around, and will continue to improve our communications.

## 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 2’s user stories and backlog)

## Sprint Planning

We started off our sprint planning meeting by assigning and volunteering for tasks according to our strong suits. For example, those who had lots of experience with the back end kept working on improving the data implementation. Once we had our assignements, we worked together to create user stories on our Jira dashboard. Everyone in the team made at least one user story and assigned it to themself. Collectively these user stories make up the sprint backlog. The project involved 3 normal sprints, each with its own sprint backlog. This sprint specifically was focused on building out the back-end and connecting it to a rudimentary front-end implementation.

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 are required to attend. Everyone else in the team that was available were encouraged to attend.

The meeting was conducted 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)

Our previous Sprint Review was our opportunity to present our work to the customer, a representative of the APPSO organization. Led by Mualla, our team welcomed our customer and presented each section of our review. We were able to answer questions and gather feedback at every step.

This was a very valuable process as we got insight that changed the design of our data in fact. Specifically, we learned that Officers shouldn’t have access to financial information, and generally scholarship questions should only be viewable by committee members.

Without being able to show our goals and preliminary design, we wouldn’t have been able to so quickly adapt to the customer’s needs. We also got a chance to interact more and build more of a relationship with the customer, and show off what we have done so far, which was fun. Thus, the Sprint Review has been a valuable experience.

At the time of the writing of this report, the Sprint Review has not yet occurred, and is scheduled for Wednesday, October 19. The video will be made available on Teams after that time.

## Sprint Retrospective

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 we learned in the Sprint? (5-10 minutes)
4. Action plan or next steps with assigned person for each one (10 minutes)

At the time of the writing of this report, the Sprint Retrospective has not yet occurred, and is scheduled for Friday, October 21. The video will be made available on Teams after that time.

# Sprint 2:

## Overall Experience

* + Implemented Google OAuth alongside devise to allow and restrict logins to @tamu.edu accounts.
  + Implemented additional new user setup that will allow the user to choose and request to be an admin, officer, or member of APPSO.
  + Created the ability for officers to create events and mark members present for each event.
  + Implemented ability for admin to approve/deny new user registrations
  + Created a basic scholarship application

## Stakeholder Management and Communication Plan

For Sprint 2, 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 |  |

## 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.

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 |

## Release Planning

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/_layouts/15/stream.aspx?id=%2Fteams%2FTeam%2DFA22%2DCSCE431%2DSoftwareEngineering%2DAggiePPStudentOrg%2FShared%20Documents%2FRecordings%2FNew%20channel%20meeting%2D20220926%5F221157%2DMeeting%20Recording%2Emp4>

## Sprint Planning

<https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/_layouts/15/stream.aspx?id=%2Fteams%2FTeam%2DFA22%2DCSCE431%2DSoftwareEngineering%2DAggiePPStudentOrg%2FShared%20Documents%2FRecordings%2FNew%20channel%20meeting%2D20220926%5F221157%2DMeeting%20Recording%2Emp4>

We combined sprint and release planning into one meeting, which lasted around 45 min.

## Scrum Meetings

Scrum Meeting Recordings:

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20\_APPSO-Parent-Portal\_-20220926\_151532-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20\_APPSO-Parent-Portal\_-20220928\_152733-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221006\_164208-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221007\_130632-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221013\_155746-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20\_APPSO-Parent-Portal\_-20221016\_211632-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221017\_151442-Meeting%20Recording.mp4?web=1

## User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | **User Story with it's acceptance criteria** | | | | |
|  | **Persona (Who)** | **Requirement (What)** | **Value (Why)** | **Critical?** | |
|  |  |  |  | **Yes** | **No** |
| 1 | APPSO Officer | I need to be able to log into a admin account | so that I can view membership information and scholarship records. | X |  |
|  |  | **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. |  |  |  |
|  |  |  |  |  |  |
| 2 | Member (scholarship applicant) | I need to be able to save answers and edit them later | so that I can conveniently start and stop my application around my and my kids' schedule. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Edit button is visible on member’s dashboard page to get back into the scholarship application. |  |  |  |
|  |  | Save button at the top of the scholarship application |  |  |  |
|  |  | Fields edited and saved by the user should repopulate upon renavigation to their application. |  |  |  |
|  |  | Fields edited and saved by the user should be stored in the database backend. |  |  |  |
|  |  |  |  |  |  |
| 3 | Member | I need to be able to access my data in a reasonable way through my website (This is the data design doc) | so that I can manage member information | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | 6 tables provided |  |  |  |
|  |  | Contains all relationships to other tables |  |  |  |
|  |  | Represents all data necessary |  |  |  |
|  |  |  |  |  |  |
| 4 | Officer | I would like to see a list of members | so that I can know who is currently in the organization as a whole | 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 |  |  |  |
|  |  |  |  |  |  |
| 5 | Officer (or developer) | I want to be able to set up Continuous Integration and Continuous Delivery actions | so that changes I make internally in the website can be tested and deployed automatically | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Officer or developer or maintainer for website is able to see github CI/CD checks running after a git commit or a merge. |  |  |  |
|  |  | Able to view what tests completed/failed in GitHub. |  |  |  |
|  |  |  |  |  |  |
| 6 | Member | I need to be able to log-in to Parent Portal securely | in order to keep my username and password from being exposed | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * OAuth successfully implemented to protect users' username and password during log-in * Users can login with Google |  |  |  |
|  |  | There is little to no risk of users' credentials being stolen |  |  |  |
|  |  |  |  |  |  |
| 6 | Member (new member) | I need to be able request to be an admin, or officer and go through signup process | To ensure that my google associated account has the right role | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | There should be a form for the user to fill out in order to store data. |  |  |  |
|  |  |  |  |  |  |
| 6 | Member | I need to have a safe storage location for the data for my website | so that I can keep track of member and scholarship information | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Members can access their member data |  |  |  |
|  |  | Members scholarship information auto-saves |  |  |  |
|  |  | Members don’t have access to private admin data |  |  |  |
|  |  |  |  |  |  |
| 7 | Member | I need to be able to log into my application without worrying about errors or bugs preventing login | in order to have a bug-free and seamless login experience | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Unit Tests are written and cover all possible situations |  |  |  |
|  |  | Integration Tests are written and cover all domains necessary |  |  |  |
|  |  | Rspec analysis conveys useful information |  |  |  |
|  |  |  |  |  |  |
| 7 | Member | I need to be able to log into my member account | so that I can view my participation data and scholarship application | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Members can log in with their username (or google email) and password |  |  |  |
|  |  | Members can recover their passwords via email if forgotten or lost |  |  |  |
|  |  | Members are not able to login as officer |  |  |  |
|  |  |  |  |  |  |
| 7 | President/Admin | I need to be able approve/deny new user registrations | so that I can verify member identity and roles | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Admin can see on a list, all new members who have requested for registration |  |  |  |
|  |  | Admin can view the email and the role that user requested. |  |  |  |
|  |  | Admin can approve/deny registrations (via buttons). |  |  |  |
|  |  |  |  |  |  |
| 8 | Scholarship applicant (user/member) | I need to be able to fill out questions/information for the scholarship application | so that I can be in the running for the scholarship | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | * Text Fields, Radio buttons with labels, etc should be appropriate for each question * e.g there shouldn’t be a radio button selection for email prompt |  |  |  |
|  |  | Able to enter information for each question/prompt |  |  |  |
|  |  |  |  |  |  |
| 8 | Officer | I need to filter completed scholarship applications | so that I can choose scholarship winners |  | X |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Officer can select a filter which presents a list of only members that completed the scholarship application. |  |  |  |
|  |  | Could be an option under the same page as membership involvement rankings. |  |  |  |
|  |  | Both the features could be under one page which is a members list page, detailing snippets of each member on each line. |  |  |  |
|  |  | Filter could be a dropdown menu (sort by..) |  |  |  |
|  |  |  |  |  |  |
| 9 | Member | I'd like to be able to see my personal participation stats tracked in an easy-to-read format | so that I can know how to improve my standing |  | X |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Member sees the dashboard display immediately upon login |  |  |  |
|  |  | Dashboard includes application completion and link to edit application |  |  |  |
|  |  | Dashboard includes attendance data |  |  |  |
|  |  | Dashboard is visually pleasing and easy to read |  |  |  |
|  |  | * Profile consists of information used in scholarship application * Only sensitive information |  |  |  |
|  |  |  |  |  |  |
| 10 | Officer | I need to be able to see specific and overall member rankings based on cumulative stats | so that I can make a data-driven decision on the winners |  | X |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Officer should be able to see a list of members ranked from highest overall org involvement to least. |  |  |  |
|  |  | This should be on the same page as a membership list page, where overall rankings can be selected as a filter. |  |  |  |
|  |  |  |  |  |  |
| 11 | Officer | I need to be able to visualize all members' involvement rankings | so that I can help decide scholarship winners |  | X |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Officer should be able to see a page ranking all the members based on highest involvement to lowest involvement. |  |  |  |
|  |  | Should only include valid members that have registered in the site. |  |  |  |
|  |  |  |  |  |  |
| 12 | Officer | I need to be able to check off if a member attended a event/meeting | in order to keep track of member attendance. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Button(s) are used to determine whether member was present/active in an event. |  |  |  |
|  |  | Clicking button(s) should update member’s participation information to include such an event. |  |  |  |
|  |  | No other inputs should be allowed. |  |  |  |
|  |  |  |  |  |  |
| 13 | Officer | I need to be able to create events in order to keep track of user attendance on different days | In order to keep track of events orchestrated within the organization. | X |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Admin or officer can click a button to create an event with name, time, and description. |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

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:

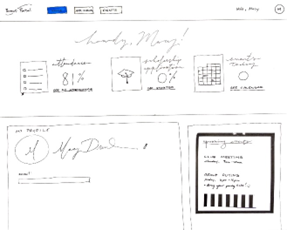
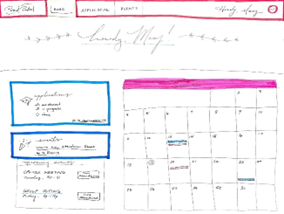
* Negative scenarios of the functionality (rainy day)
* Functional and non-functional use cases. Example of a non-functional use case is performance requirements
* What system/feature intends to do
* End-to-end user flow
* Impact of a user story to other user stories (i.e., features)
* UX concerns

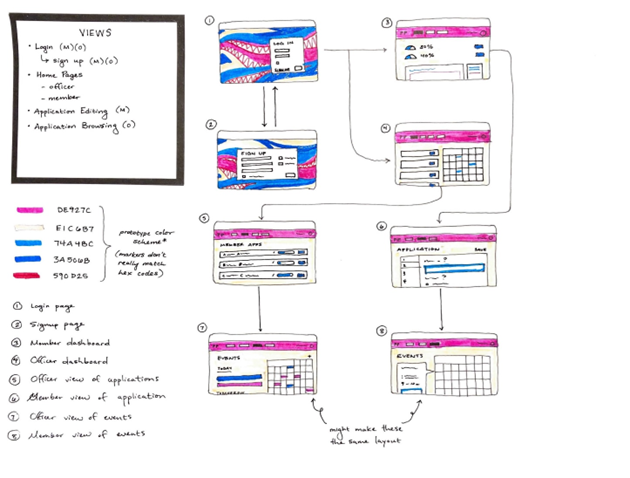
Acceptance Criteria helped clear any misconceptions and confusions regarding what we should exactly create in order to satisfy a user story. It allowed us to verify pull requests whenever somebody was done implementing a certain user story. Compared to Sprint 1, we really used the acceptance criteria in sprint 2 to really make sure whatever we implemented was truly done and covered all bases. It also was used to create our integration/unit tests, especially when implementing user sign in/out, and Google OAuth. We really benefited from creating clear acceptance criteria for each user story.

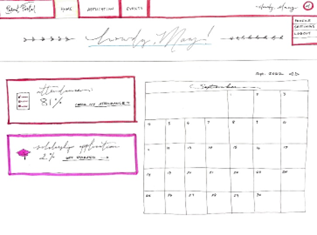
## UX Models

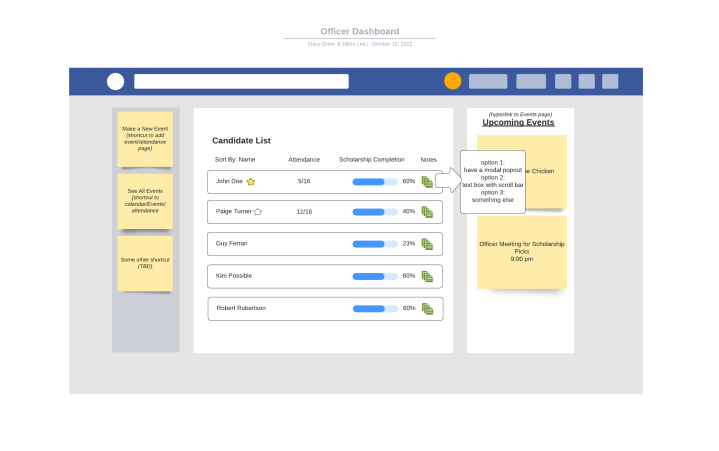
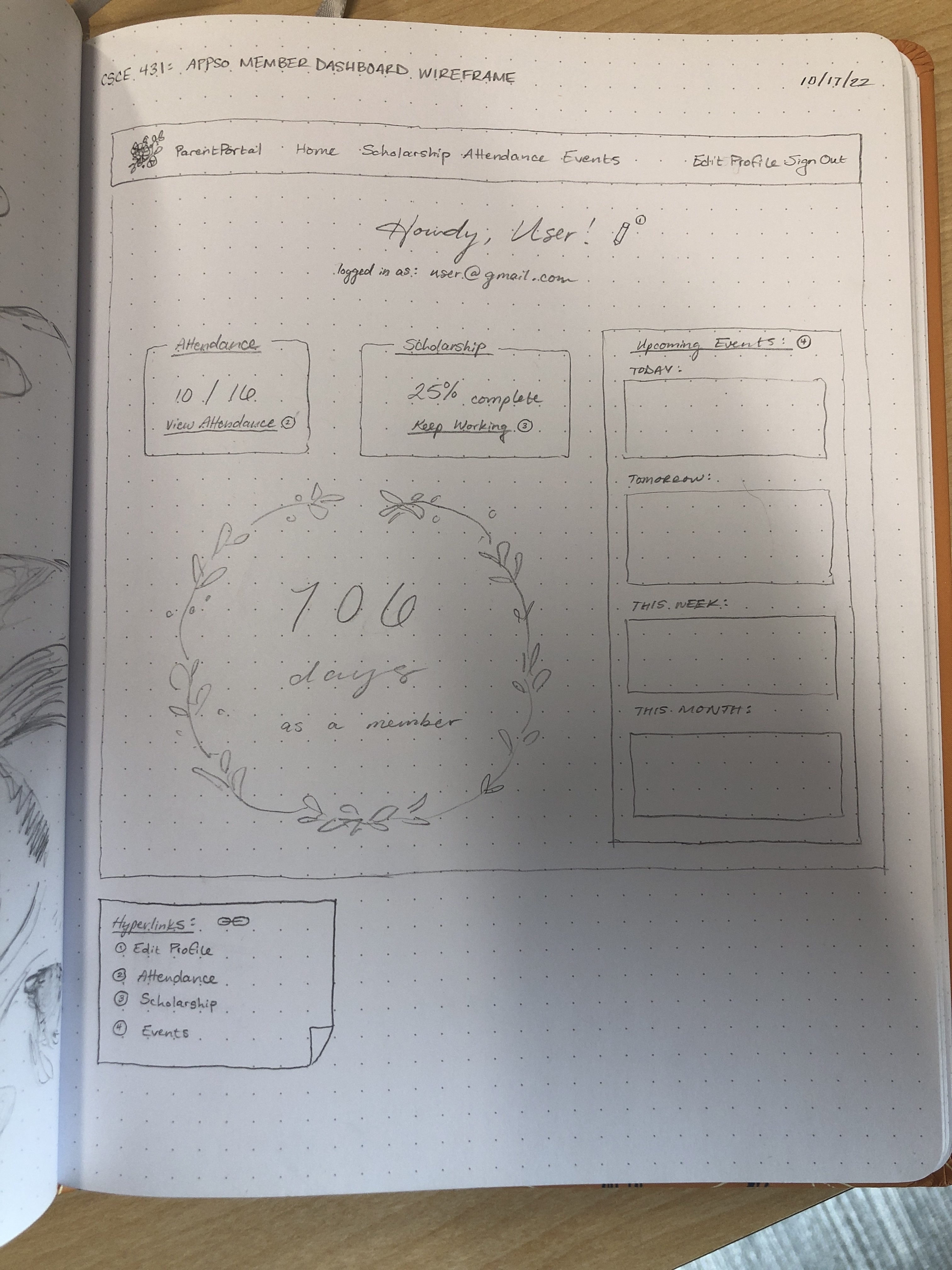
To make sure that user stories are properly understood by the stakeholders (e.g., customers, development team, etc.), it is a common practice to supplement with UX models, such as lo-fi diagrams.

<Representative samples of UX models here and the complete set in the Annex.>



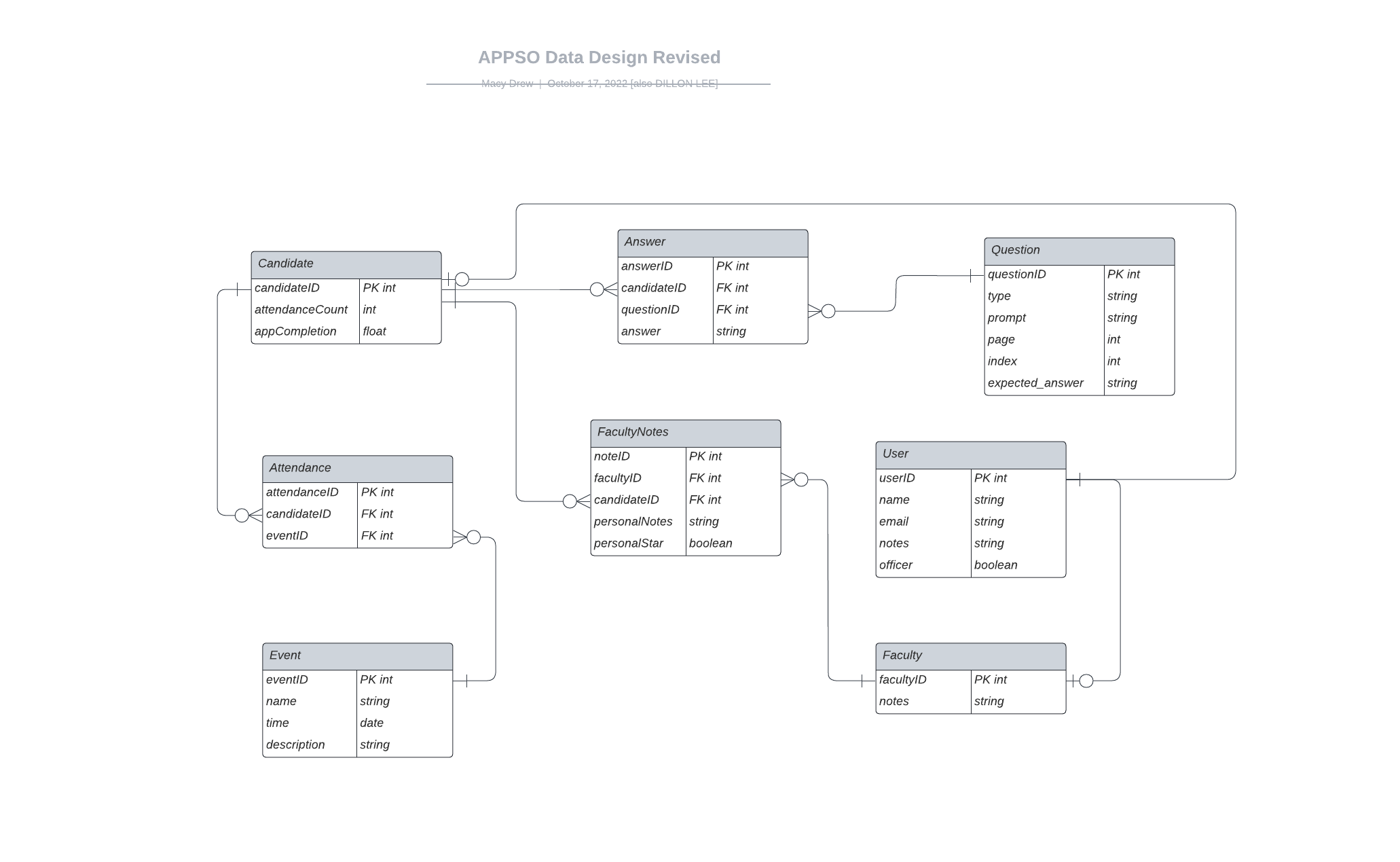






## Data Design

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.



The data design was an arduous yet revealing task. We needed to be able to store user data, allow them to answer questions, and allow for officers to view questions and take notes on users, potentially ranking ones that they deem deserve financial support. Initially, we only had 4 tables, but this was not sufficient for the requirement to have one table for each member in our group. Also, as we continued talking with the customer and got feedback from their meetings with committee members, we learned about the necessity of restricting scholarship information from officers, only showing that to committee members.

From this feedback, we revised the data design to the result seen here. We identified Foreign Keys and precise relationships between tables, that we could implement in our Ruby on Rails application directly successfully. This was a good experience as it showed that data design is hard but very important.

## 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.

In Sprint 2 of this project, GitHub played a huge role. GitHub allowed us to manage different user stories we were working on and allowed us to truly work on a team. Whenever people were done implementing a certain feature on a separate branch, they would create pull requests, where another member will conduct a pull request review, and code review, to check and ensure the code created satisfies **acceptance criteria** and the definition of done. GitHub allowed us to practice Continuous Integration, where we regularly and quite frequently merged to the main branch whenever we completed user stories. We also practiced continuous delivery by being able to automatically deploy a test environment for features created. It was such a very productive and nice experience.

## 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.

Metrics such as compliance, efficiency, correctness, maintainability, integrity, and usability allowed us to visibly and tangibly have evidence on our quality and effectiveness in terms of what we output in our project. It allowed us to know, “How good is the work we’re putting out?”. I really appreciate the lecture regarding such topics by Professor Wade, it was indeed quite insightful, and allowed us to prepare for such measures in this sprint. We have noticed an increase in quality in this sprint.

### Compliance

We monitor and practice compliance by first looking at our project scope, and mapping user stories that most reflect our project scope. We identified our Most Valuable Stories from this method, which ended up to be:

“As an APPSO officer, I need to be able to check off if a member attended a event/meeting, in order to keep track of member attendance.”

“As an APPSO president/admin, I need to be able approve/deny new user registrations so that I can verify member identity and roles.”

“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.”

“As an APPSO Member, I need to be able to log into my application without worrying about errors or bugs preventing login, in order to create a bug-free and seamless login experience.”

“As a new APPSO member, I need to be able request to be an admin, or officer and go through signup process.”

“As an APPSO user, I need to be able to log-in to Parent Portal securely, in order to keep my username and password from being exposed.”

“As an APPSO member, I'd like to be able to see my personal participation stats tracked in an easy-to-read format so that I can know how to improve my standing.”

These user stories were most related to our project scope.We measured compliance by the percentage of these most valuable stories completed. At the end of our sprint, we managed to complete a majority of these stories. The last story we sort of completed, by having a basic stat presented upon logged in, but we do not have a page where they can see their full stats in a “member profile” format. So, the completion of that user story could be marked as completed, but long-term, we plan on expanding on it.

More information can be found here: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

### Efficiency

For this sprint, we used a few different strategies to ensure efficiency of work. Each team member took on different user story tasks. We divided these based on their front-end and back-end requirements. These assignments didn’t always stick with the same person, depending on each team members’ time requirements to other classes. We also ensured efficiency by communicating to each other about which tasks we needed help on to make sure that no one got behind. This made Jura an extremely helpful organization app because we could all see which tasks were pending or in progress or finished.

We also measured efficiency by setting a metric of 75 story points to complete this sprint.



Based on this sprint burndown chart and the points completed as said in the backlog, we completed 113 story points (or more if we completed more things while completing this document). We marked off several user stories at a time, especially during/after SCRUM meetings. We did have some left over user stories we haven’t completed, but these stories were not deemed essential or critical, when looking at the project scope. This will be completed in sprint 3, especially since these tasks are more front-end oriented.

More information can be found here: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

### Correctness

[Reflect on how the team met this objective, including how you determined correctness (e.g., Defects per sprint that are high severity (no workarounds) where defects are errors discovered by the customer). Included below are some methods used by the team to ensure correctness.]

We met this objective by checking to make sure, and frequently, that 0 errors escaped to the user. There were times where errors and defects did pop up during development, but we were able to rectify them quickly. These were quite evident during pull requests, where errors happened to appear during checking, and we managed to fix them pretty quickly. More information can be found in this document: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

A full detailed reflection and process of ensuring Monitoring & Control can be found here: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

It’s also in our Documents folder for Sprint 2 as “APPSO\_Monitoring & Control Notes”

The file above is the result of monitoring and control done over the course of the sprint. We measured the quality measures in the recommended fashion mentioned in lecture (Thanks Professor Wade!).

#### 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 faster MTTR, less bugs in production, less confusion over what exactly to code, etc.
* TDD allowed us to create more bug-free code, or in other words code that performs well with little error. By creating these tests beforehand, we create effective code, which can later be cleaned up and refactored later.
* More understanding amongst groupmates on what a feature should do, what errors it should avoid, etc.

#### Reviews:

Code reviews (and other technical reviews such as pair programming) were a big part of ensuring quality and involved [describe your process e.g., GitHub pull requests].

Whenever a group member is working on a user story, they work on a separate branch. Once they complete the user story and their preliminary testing, they created a pull request on GitHub to merge their branch into main. One of the backend team members will look at the code, and comment whether it **satisfies the acceptance criteria** and the definition of done. If applicable or necessary, they will also look at a CD, deployed app to ensure that the changes don’t affect the flow of the application negatively. If everything looks good, the reviewers will leave a comment ensuring its good and allows the pull request author to confirm the pull request, and merge to main.

What we learned:

* Code reviews allowed us to discover defects related to user stories and verify that we followed the different standards set forth in the project, including acceptance criteria, definition of done, and coding standards.
* For Sprint 2, we **really** emphasized doing code reviews, whenever we need some clarification or help regarding features we are implementing, or when trying to merge to the main branch. A lot of our pull requests are more detailed in terms of acceptance criteria and seeking approval from other members, compared to Sprint 1 (where we only had 1 pull request). The backend team loved the idea of code reviews, as it ensured that the code, we truly made is good and does not break anything. We had a scenario where a certain pull request did break something, and we added additional commits to fix the error.

#### Definition of Done

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

This artifact allowed us to “check down” on tasks that we should complete at the end of each sprint. It helps create a sense of accountability with the work we are doing. Specific criteria such as the ones listed below allow us to review and improve our quality. We can see which criteria we haven’t satisfied, and plan in future sprints to fulfill them.

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

| 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 | Y |  |
| Sprint deliverable reviewed by the product owner | Compliance | Y |  |
| Sprint Scrum meetings conducted on a daily basis | Efficiency | N | Weekly schedule used. |
| 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/A | Not released yet |
| Sprint Retrospective Conducted | Maintainability | N/A |  |
| 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 | Y | Only submitted |
| 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 | 90% realistcally |
| 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 | Starting afer this sprint, after completing the **plan** |
| Code refactored (as agreed upon by the team) | Maintainability | Y |  |
| No fatal ‘code smells’ in code analysis output (e.g., Rubocop) | Maintainability | Y |  |
| Meets Coding Standard | Maintainability | Y |  |
| All Unit Tests passed | Correctness | Y |  |
| All Integration tests passed (including system test when all components integrated) | Correctness | Y |  |
| Tests on all supported devices & browsers passed | Correctness | Y |  |
| All Integrity testing passed | Integrity | Y |  |
| Usability tests passed (should be based on acceptance criteria of user) | Usability | Y |  |
| All Performance testing passed to ensure minimum response time met | Correctness | Y |  |
| 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 | Y |  |
| Any configuration or build changes documented | Maintainability | N |  |
| 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:

Below is what we deemed our most important test cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| “As an APPSO president/admin, I need to be able approve/deny new user registrations so that I can verify member identity and roles.” | Integration test flow of logging in as admin, going to admin page, and approving a user. | An alert saying a user has been approved | Sunny |  |
| “As an APPSO president/admin, I need to be able approve/deny new user registrations so that I can verify member identity and roles.” | An integration test flow of an admin logging in as admin and denying a user trying to be an admin. | An alert saying user has been rejected | Rainy |  |
| “As an APPSO user, I need to be able to log-in to Parent Portal securely, in order to keep my username and password from being exposed.” | Integration test flow of logging in (new user) via Google and returning to a page to finish sign up. | An alert saying authentication complete, and to finish registration | Sunny |  |
|  |  |  |  |  |

Example below:



**EDGE CASE for user approval (very critical edge case):**

**User Story: “As an APPSO president/admin, I need to be able approve/deny new user registrations so that I can verify member identity and roles.”**

**Input: A user who registered and requested to be a admin (when they shouldn’t be), and they try to go to the admin page to approve themself.**

**Expected Output: An alert saying you need to be approved or be an admin to access that page**

**Rainy day case.**

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) |

[Talk about the data you used in the different test phases (e.g., unit, integration, validation, etc.)]

For Unit tests, we just created basic entries to test models to ensure it detects invalid inputs, and detects valid inputs.

Creating tests and ensuring test coverage was still a difficult process for us, but we were able to create tests for the most essential parts of our website, which include sign/sign up, and approving and denying new members. These are two key areas of possible exploitation by a malicious user. Creating these test cases and ensuring coverage allowed us to make sure that we were covering all our bases.

##### 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.

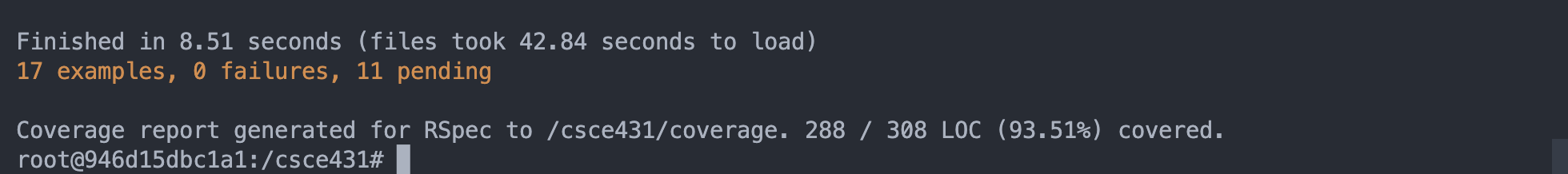
See below Simplecov output for our application.

[APPSO\_SimpleCov\_Result.html](https://tamucs.sharepoint.com/:u:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/ERRVfGCg3NFJiI3XiZorMKYBzt2Y95Q6WX05kX74vlVaxg?e=Ehnpy1)

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.



We passed all our test cases, we have 11 pending for tests for certain pages we couldn’t finish in time, but we plan on finishing up on these at the beginning of next sprint.

require 'rails\_helper'

RSpec.describe 'Sign up integration', type: :feature do

it 'should be able to sign up (without approval)' do

visit '/users/sign\_up'

fill\_in 'Email', with: Faker::Internet.email

fill\_in 'Password', with: 'password'

fill\_in 'Password confirmation', with: 'password'

#Click member radio button

choose('user\_admin\_member')

click\_button 'Sign up'

expect(page).to have\_content('Welcome! You have signed up successfully.')

end

end

#Sign in test

RSpec.describe 'Sign in integration', type: :feature do

#sign in

it 'should be able to sign in' do

#sign in

visit '/users/sign\_in'

fill\_in 'Email', with: 'email@e.com'

fill\_in 'Password', with: 'password'

click\_button 'Log In'

expect(page).to have\_content('Signed in successfully.')

end

end

#Google sign in test

RSpec.describe 'Google sign up/in integration', type: :feature do

before(:each) do

Rails.application.env\_config["devise.mapping"] = Devise.mappings[:user]

Rails.application.env\_config["omniauth.auth"] = OmniAuth.config.mock\_auth[:google\_oauth2]

end

it 'should be able to sign in with google' do

#sign in

visit '/users/sign\_in'

click\_link 'Sign in with Google'

expect(page).to have\_content('Google Authentication complete. Please fill out user role.')

end

end

#Approve member test

RSpec.describe 'Approve member integration', type: :feature do

it 'should be able to approve a member' do

#sign in

visit '/users/sign\_in'

fill\_in 'Email', with: 'email@e.com'

fill\_in 'Password', with: 'password'

click\_button 'Log In'

#go to admin page

visit '/admin'

#Approve approveme@gmail.com

click\_button('APPROVE THIS: Approve Me')

expect(page).to have\_content('User approved.')

end

end

#Reject member test

RSpec.describe 'Reject member integration', type: :feature do

it 'should be able to approve a member' do

#sign in

visit '/users/sign\_in'

fill\_in 'Email', with: 'email@e.com'

fill\_in 'Password', with: 'password'

click\_button 'Log In'

#go to admin page

visit '/admin'

#Approve approveme@gmail.com

click\_button('REJECT THIS: Remove Me')

expect(page).to have\_content('User rejected.')

end

end

These are examples of some tests we created. These tests all passed.

No bugs were found as of making of this report.

Heroku test app link: <https://appso-portal-test-psydwj54r5ye.herokuapp.com/>

Test Admin credentials: email: [email@e.com](mailto:email@e.com) password: password

### 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 second 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.

More information on maintainability was in this document: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

#### 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. |

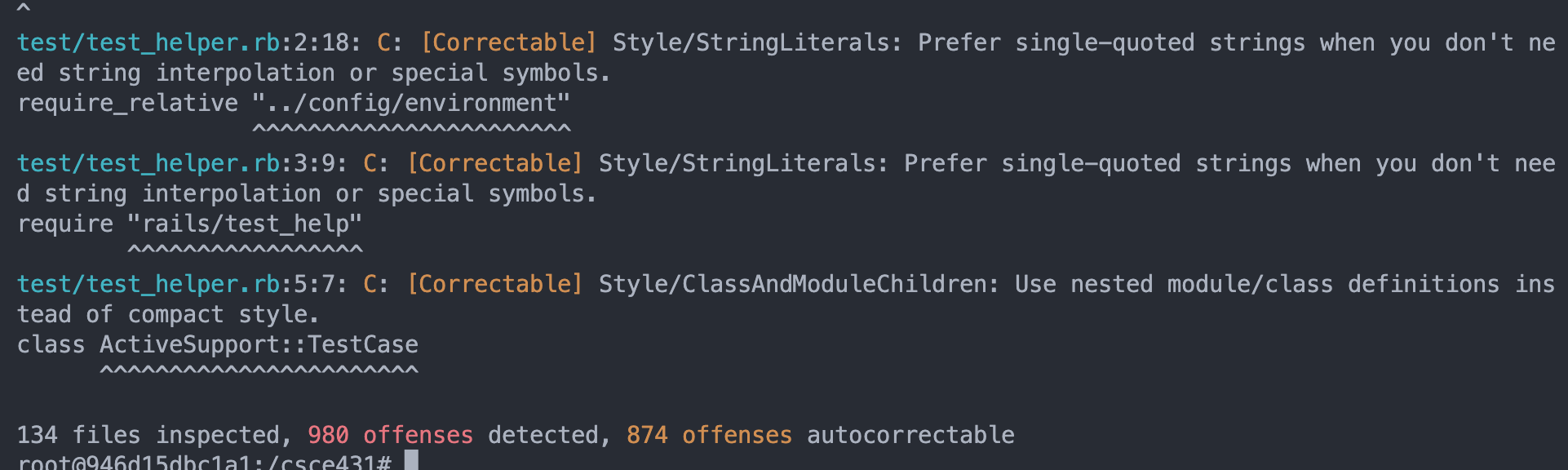
Adhering to a coding standard allowed us to have cleaner and most importantly, readable code. Keeping the style and organization of code to a standard allows us to focus more on functionality and implementation rather than understanding how a piece of code that is typed non-professionally.

#### 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).



We did not put much focus on following our coding standards, because we primarily focused on functionality and core integration testing. We wanted to get as much stuff done as possible in this sprint, and there were times we did not follow the coding standard we define. However, a majority of these offenses are auto-correctable and will be looking into these options in the near future.

### 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 are located 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 from which we want to have users authenticated.

We implemented Google Auth by following the attached video, as well as the related lab document:   
[Devise Google Login With Omniauth | Ruby On Rails 7 Tutorial](https://www.youtube.com/watch?v=CnZnwV38cjo)

[](https://www.youtube.com/watch?v=CnZnwV38cjo)

#### 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:

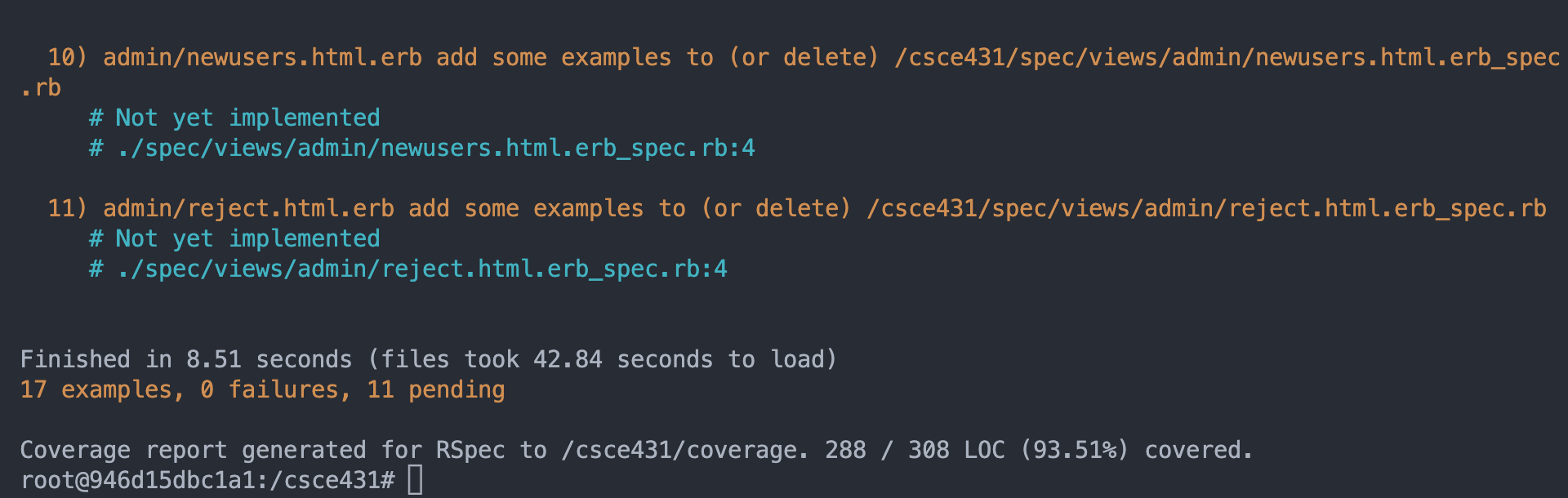
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| “As an APPSO member, I want to be able to fill out a member sign-up form, in order to be 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 |  |
| “As an APPSO officer, I need to able to see a alert showing that a user was marked present, in order to confirm that the user’s presence in an event is confirmed in the database | User clicking on present/absent button | Redirects correctly by refreshing the page, not by collecting params form the url | Rainy |  |
| “As an APPSO officer, I need to able to see a alert showing that a user was marked present, in order to confirm that the user’s presence in an event is confirmed in the database | User clicking on present/absent button | Refreshes the page with an alert showing the user was marked present/absent | Sunny | This was used to test the unprotected redirect warning brakeman reports |

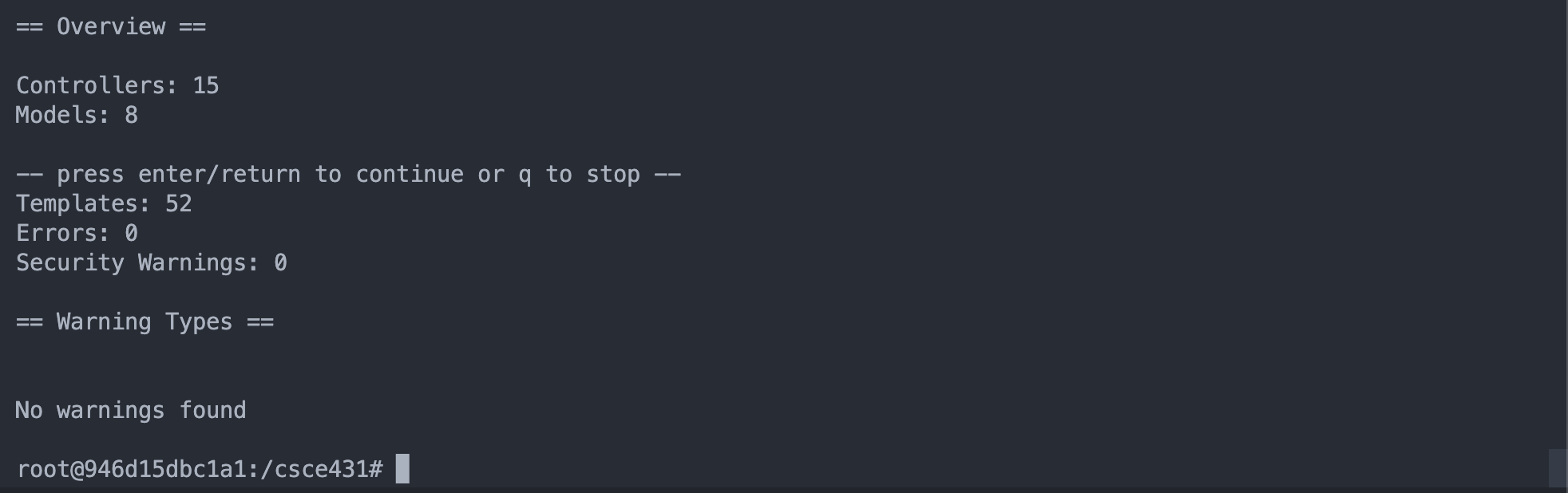
Basic data that tests general functionality were used for the different test phases. The process of creating test cases to ensure security breaches are not present was straightforward. Some of the members of the team have general knowledge regarding potential security breaches via XSS scripting and SQL injections so we were able to create basic test cases for such scenarios. We created vital integration tests for login/logout, and for approving or rejecting new users. Doing these tests allowed us to ensure that no malicious individual would interact with our application in a negative manner.

#### Integrity Test Results

We ensured that our RMMM plan worked to address common security vulnerabilities by [fill in with approach such as attempting to enter the system without the proper credentials, etc.]

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





Brakeman reported “no warnings found”.

We passed all test cases for our integration tests on integrity.

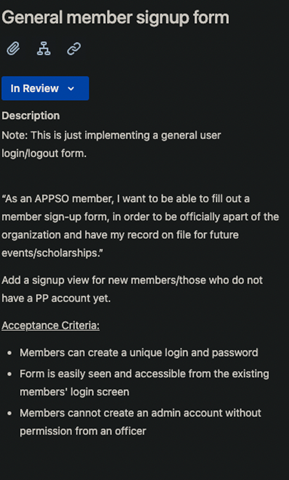
### Usability

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. This test was done with a member’s roommate and was able to get that roommate acclimated to the application in approx. 10 minutes. Usability metrics will be improved upon in future sprints.

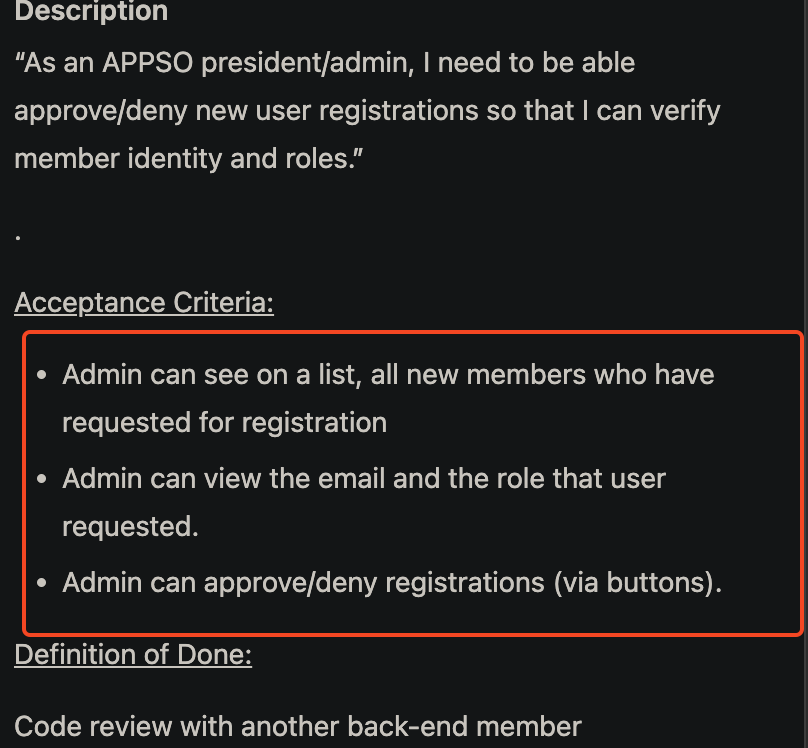
More Information in our Monitoring & Control notes: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

#### Usability Requirements

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



Admin approval user story:



#### 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 a part 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 |  |
| “As an APPSO user, I need to be able to log-in to Parent Portal securely, in order to keep my username and password from being exposed.” | Integration test flow of logging in new user via Google and returning to a page to finish sign up. | An alert saying authentication complete, and to finish registration | Sunny |  |

We 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.

Our existing integration tests also mock user interaction flow with registering, signing up and signing in, oauth registration, and new user approval.

#### Usability Test Results

We ensured that our system was usable by conducting user flow tests via integration tests and individual testing.

<https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

Here we described in the “usability” portion, we had a person test our application and was able to use it at around 10 minutes after preliminary explanations.

We plan on implementing more user flow tests and potentially asking various people to test the app and timing their acclimation to it.

## Monitoring and Control

We monitored Efficiency, Compliance, Correctness, Integrity, Usability, and Maintainability to ensure we were doing the right actions to produce the right product.

Here is what we decided to measure for each metric:

Efficiency: Velocity of 75 story points per sprint

Why 75 story points? This is the sprint where we need to do the most work. Members collectively agree on this reasoning.

Compliance: 100% of the most valuable user stories deployed

Correctness: 0 errors escaped to the user

Integrity: 0 security breaches

Usability: Intended user learns how to use the system in 15 minutes or less

Maintainability: Mean Time to Repair (MTTR) - from assignment of ticket to deployment of fix – of 5 hours.

More discussion regarding how we monitoring these metrics are in this document: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Edy2TtcsVUVJh-gRGVUTNicBKY0NBdmK7fio2uff_QAw5g?e=r6fIns>

Here is an example of a Jira feature we used to produce a Sprint Burndown chart:



It’s imperative that we observe and maintain Monitoring and Control in our project, especially when trying to observe the velocity of our user stories, aka how many user stories we completed this sprint. We really wanted to make some good progress this sprint, and despite the hiccups with other classes taking a substantial amount of our time, we were able to pull it off. We were able to satisfy most of our metrics, with a few key “asteriks”, since we didn’t happen to satisfy everything, but we are still grateful for our progress, all only in sprint 2.

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

For convenience, and due to how lengthy and complicated a complete creation of a Github student account and Heroku account for the customer is, we simply created a customer email that the customer can temporarily use while we prepare the video and documentation needed to create a smooth transfer to an actual account hosted by the customer rep.

Gmail details (THIS IS WHAT CUSTOMER OWNS):

email: [appsoportaltemp@gmail.com](mailto:appsoportaltemp@gmail.com)

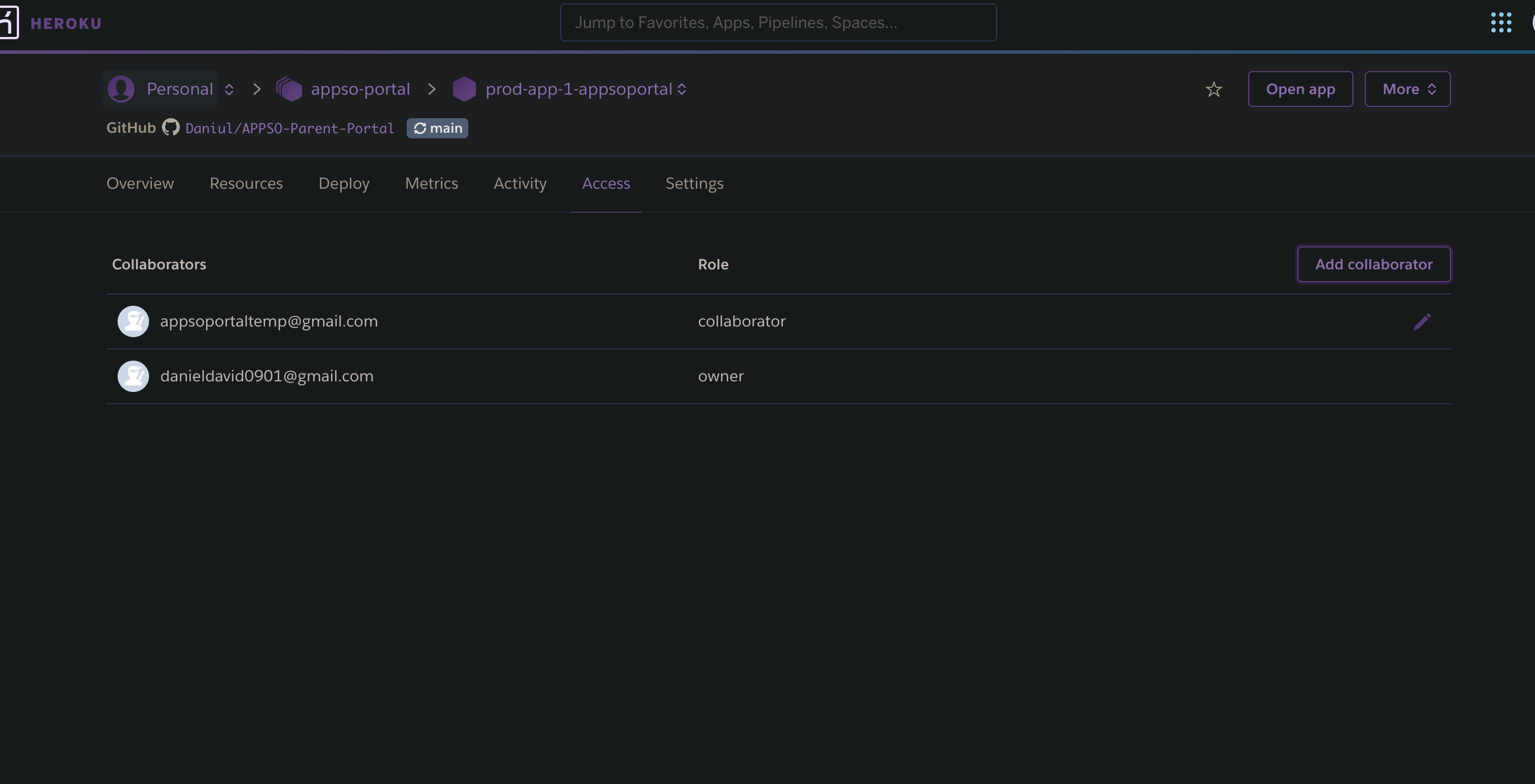
pass: C&Ux5ATsS%ZSnj

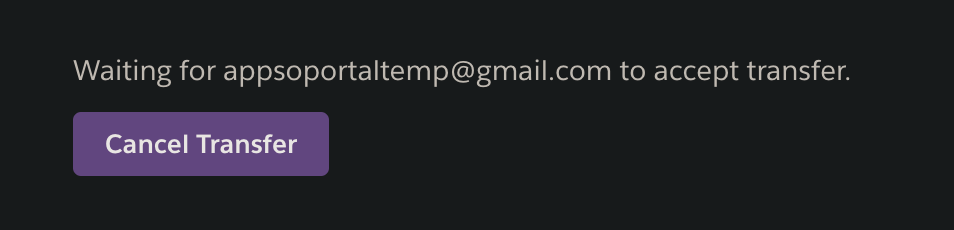
Heroku account details (where the app will be hosted):

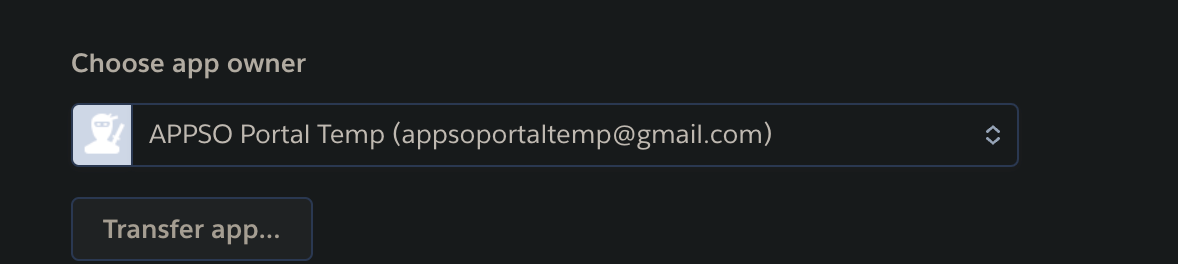
email: [appsoportaltemp@gmail.com](mailto:appsoportaltemp@gmail.com)

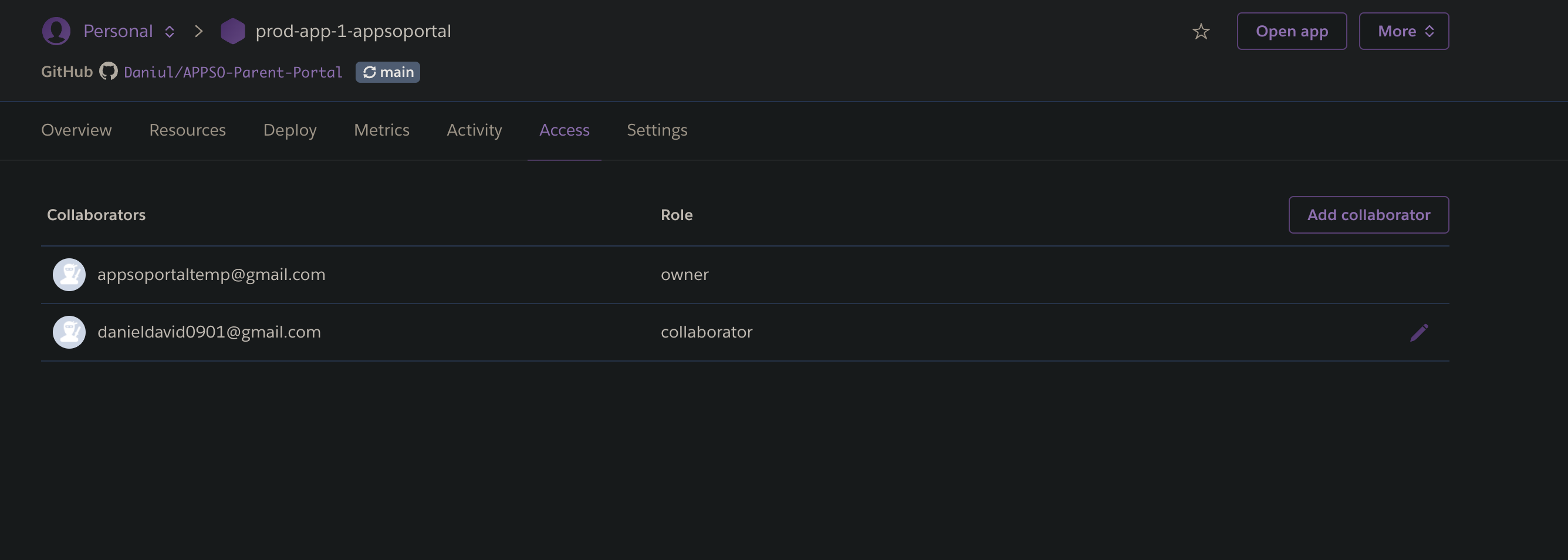
pass: yHf%#Hq4s!L6jV

We followed the following documentation to transfer apps: <https://devcenter.heroku.com/articles/transferring-apps>









Production link to be given to customer:

<https://prod-app-1-appsoportal.herokuapp.com/>

Now the customer can use the heroku credentials provided here (until November 28th, by then we have documentation for transfer), to access the production app.

## Sprint Review

At the time of the writing of this report, the Sprint Review has not yet occurred, and is scheduled for Wednesday, October 19. The video will be made available on Teams after that time.

## Sprint Retrospective

At the time of the writing of this report, the Sprint Review has not yet occurred, and is scheduled for Friday, October 21. The video will be made available on Teams after that time.

## Continuous Integration (CI) / Continuous Delivery (CD)

CI / CD: Continuous integration is a process and practice of creating frequent and recurring merges to the main branch. Developers will create feature branches, make changes on those branches, and then merge those changes back into the main branch. GitHub allows us to do this with pull requests, where we can frequently check what our teammates have made changes to, and then review and confirm their merge requests. Commits made on all the branches can also automatically run tests on those changes (with rspec and our workflow file we defined in lab, which we applied in this project).

Continuous Delivery is where any commits made to the main branch, or any feature branches, automatically create review apps or are deployed to review apps. GitHub in collaboration with Heroku can handle this, where any changes to a branch will automatically deploy such changes to a review application designated for that branch.

Implementing and utilizing Ci/CD helped tremendously in improving the workflow for sprint 2. It allowed us to make sure that the features we implemented are in check with the tests that we made (tests are generally made before, but a few were added later in certain cases). Our team members were able to easily do code reviews and review our pull requests, to see any changes we make to our code (through GitHub). It made it incredibly easy to see what our changes look like on a test application (through Heroku CD). It’s great that any tests we created and committed will not be reflected in GitHub Actions, very smooth experience!

## Deployment & Support Plan

In our plan, we emphasize videos a lot because it allows us to describe our thoughts and advice better in a more direct form and allows future members and officers to get a better understanding of potential questions and concerns they have.

|  |  |  |
| --- | --- | --- |
| **Documentation Plan** |  |  |
| **Type - Examples below** | **Strategy** | **In charge** |
| –System (e.g., how to transfer admin privileges, how to backup data) | Documentation both part of online help and a separate electronic copy | Thierry |
| –Installation / Set-up | Not needed since ruby does not have required updates to proceed | N/A |
| –User Documentation. Will you provide context sensitive help (in the actual form) – THIS IS EXPECTED TO BE ONLINE HELP, NOT A SEPARATE DOCUMENT THAT CAN GET MISPLACED | We could be reached via contact | Mualla |
| –Other References (e.g., important Heroku links) | https://devcenter.heroku.com/start |  |

|  |  |  |
| --- | --- | --- |
| **Training Plan** |  |  |
| **Type - Examples below:** | **Strategy** | **In charge** |
| -Train resident expert (representative in the user group) who has the most knowledge about the system & will provide ongoing support | Scheduled training | Thierry, Frank |
| -Train few key users, who will train others | We will train the president of the organization who will have super admin access and will teach upcoming members. Once the president leaves the university they are in charge of training the new president as well. | Mualla |
| -Online help | Contact via email to one of our group members | All members |
| –Other |  |  |
| **List of Training topics. Examples below:** |  |  |
| –Use of system | Watch youtube tutorial on how to use our website | Dillon, Thierry |
| –System installation & set-up | Provide a read.me | Frank |
| –System administration | We teach the current president of the org what it takes to be a super admin of our website. | Mualla |
| –Backup plan: Process for backup? How often should the user backup? | Install backup gem to ruby on rails. There will be a youtube tutorial we make on this | Thierry |
| –Recovery: What is the plan for recovery if the system crashes? | We will have a previous stable build for rollback, then Thierry will be contacted about the error | Thierry |
| –Other |  |  |

|  |  |  |
| --- | --- | --- |
| **Support (\*\*Contributes most to overall user satisfaction of your system)** |  |  |
| **Type - Examples below:** | **Strategy** | **In charge** |
| –App online help to help in the use of the system (e.g., general help vs. context sensitive help) | Online help that is part of the application or organization’s wiki | All members |
| –Bulletin: proactive information sharing (e.g., announces new releases, scheduled downtime, etc.) | Alerts on website above banner for scheduled downtimes and announcements | Nathan |
| –Mechanism to capture user feedback (e.g., app support survey, client feedback survey, etc.) | Footer of website will have a user feedback form | Mualla |
| –Other |  |  |
| **List of issues that may happen during Support** |  |  |
| –System crashes | Website will have an automatic website maintenance page whenever this occurs to not lose any visitors. Organization members can contact Frank for debugging system crashes. | Frank |
| –Lost username / password | Will never happen due to redirection from OAuth. In case it does happen. They can go to google and reset their password. | Macy |
| –Other |  | Macy |

|  |  |  |
| --- | --- | --- |
| **Installation & Data** |  |  |
| **Type - Examples below:** | **Strategy** | **In charge** |
| –Transferring ownership of app to customer via their own Heroku account | Dillon will host a recorded meeting where ownership transferring is facilitated. The meeting will be recorded so the APPSo team can reference it whenever they want to transfer ownership again.  https://devcenter.heroku.com/articles/transferring-apps | Dillon |
| –Schedule for installation (e.g., off-time or non-peak). | Every Sunday we will let them know what to install | Thierry |
| –Contingency plan during installation failure | If future updates do not install correctly, Macy will schedule a debugging session to figure out what is wrong with the gems | Macy |
| –Develop utilities to convert and/or transfer data from old to new system | Macy will write scripts to automate this process for our customer. | Macy |
| –Other | N/A | N/A |
| **Data to be transferred from old system & reason for retaining the data (\* indicates critical data)** | **N/A** | N/A |
| <List data to be transferred &reason for retaining the data (e.g., org member data, participation per member, etc.> | N/A | N/A |