Change Log

|  |  |  |
| --- | --- | --- |
| Change | Date of Change | Author |
| Baseline | 10/23/22 | Pauline Wade |
|  |  |  |
|  |  |  |

Final Report

Parent Portal

Aggie Pregnant and Parenting Student Organization

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

CSCE 431 – Software Engineering  
Fall 2022

Texas A&M University

Department of Computer Science and Engineering

Contents

[1 Project Scope 5](#_Toc117520300)

[2 Stakeholder Analysis 6](#_Toc117520301)

[2.1 Stakeholders 6](#_Toc117520302)

[3 Implementation Environment 7](#_Toc117520303)

[3.1 Hardware 7](#_Toc117520304)

[3.2 Software 7](#_Toc117520305)

[4 Scrum Process 8](#_Toc117520306)

[4.1 Release Planning 9](#_Toc117520307)

[4.2 Sprint Planning 9](#_Toc117520308)

[4.3 Scrum Meetings 9](#_Toc117520309)

[4.4 Sprint Review 9](#_Toc117520310)

[4.5 Sprint Retrospective 10](#_Toc117520311)

[5 Sprint 3 11](#_Toc117520312)

[5.1 Overall Experience 11](#_Toc117520313)

[5.2 Stakeholder Management and Communication Plan 11](#_Toc117520314)

[5.3 Risk Mitigation, Monitoring, & Management Plan 13](#_Toc117520315)

[5.4 Release Planning 14](#_Toc117520316)

[5.5 Sprint Planning 14](#_Toc117520317)

[5.6 Scrum Meetings 14](#_Toc117520318)

[5.7 User Stories 15](#_Toc117520319)

[5.8 UX Models 15](#_Toc117520320)

[5.9 Data Design 16](#_Toc117520321)

[5.10 Version Control 16](#_Toc117520322)

[5.11 Quality 17](#_Toc117520323)

[5.11.1 Compliance 17](#_Toc117520324)

[5.11.2 Efficiency 17](#_Toc117520325)

[5.11.3 Correctness 17](#_Toc117520326)

[5.11.3.1 Test-Driven Development (TDD): 17](#_Toc117520327)

[5.11.3.2 Reviews: 17](#_Toc117520328)

[5.11.3.3 Definition of Done 18](#_Toc117520329)

[5.11.3.4 Test Coverage 20](#_Toc117520330)

[5.11.3.4.1 Code Coverage - Output of simplecov 20](#_Toc117520331)

[5.11.3.5 Test Results 21](#_Toc117520332)

[5.11.4 Maintainability 22](#_Toc117520333)

[5.11.4.1 Coding Standard 22](#_Toc117520334)

[5.11.4.2 Well Documented Code 23](#_Toc117520335)

[5.11.4.3 Linter Output 23](#_Toc117520336)

[5.11.5 Integrity 23](#_Toc117520337)

[5.11.5.1 Security Risk Analysis Table 23](#_Toc117520338)

[5.11.5.2 Authentication 23](#_Toc117520339)

[5.11.5.3 Integrity Test Coverage 23](#_Toc117520340)

[5.11.5.4 Integrity Test Results 24](#_Toc117520341)

[5.11.6 Usability 24](#_Toc117520342)

[5.11.6.1 Usability Requirements 24](#_Toc117520343)

[5.11.6.2 Usability Test Coverage 24](#_Toc117520344)

[5.11.6.3 Usability Test Results 24](#_Toc117520345)

[5.12 Monitoring and Control 25](#_Toc117520346)

[5.13 Deploying App 25](#_Toc117520347)

[5.14 Sprint Review 26](#_Toc117520348)

[5.15 Sprint Retrospective 26](#_Toc117520349)

[5.16 Continuous Integration (CI) / Continuous Delivery (CD) 26](#_Toc117520350)

[5.17 Deployment & Support Plan 27](#_Toc117520351)

[6 User Acceptance Test Form 29](#_Toc117520352)

[7 Annex: Screenshots of the UX Models / User Interface 30](#_Toc117520353)

**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?

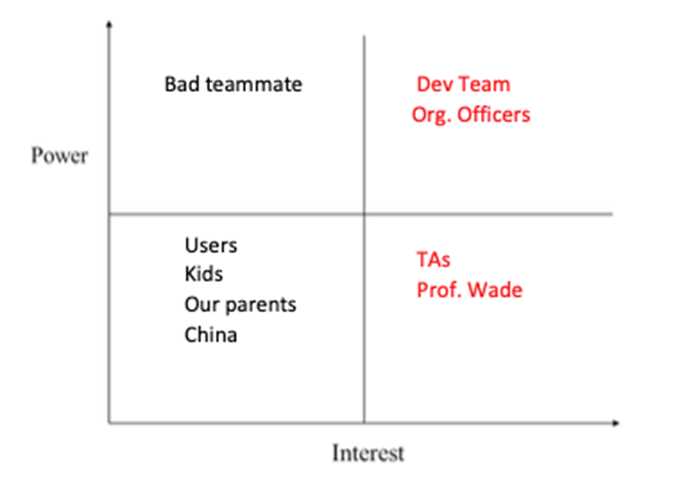
# Project Scope

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 |
|  |  |
|  |  |
| *<It is important to also state exclusions, or what will not be included in the project. List the exclusions along with their reasons.>* | |
| **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

[Describe any special roles held by specific team members, if any. You can also state what everyone’s role was (e.g., contributed to the code either individually or through pair programing). ]

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

# 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 x.y.z
* Rails x.y.z
* Docker version x
* Bootstrap x.y.z (if applicable) – state why this was used
* jQuery-rails x.y.z (if applicable) – state why this was used
* rspec-rails – state why this was used
* Heroku
* PostgreSQL x.y.z
* Git Hub
* RuboCop or Code Climate – state why this was used
* Simplecov – state why this was used
* Brakeman – state why this was used
* React (if applicable) – state why this was used
* Jira – state why this was used e.g., We used Jira to distribute the tasks among the team members and keep track of the progress in the development.
* OTHER aspects of the environment and why it was used?

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, which do not have to be done in a specific order. For example, testing can (and should) happen simultaneously or before development.

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
* [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
* Test Results

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 sprint, as we were much more familiar with Rails by that point, and our Sprint 3 tests were much better as well. 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 in the second sprint, and were very comfortable with it in Sprint 3.

## 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 end 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 3’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 assignments, 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 around 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 is available are encouraged to attend (although extra credit was given if everyone in the team attended).

The meeting was held on 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)

We have not yet done a Sprint review for Sprint 3. Therefore, our reflection will be on a separate document.

## Sprint Retrospective

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

<Overall reflection on this process.>

We have not yet done a Sprint retrospective for Sprint 3. Therefore, our reflection will be on a separate document.

# Sprint 3

This section is almost identical to Sprint 2, hence no need to change information (e.g., plans or processes) that didn’t change. You do need to:

1. Document any changes or new information. You may need to justify why something did NOT change during the audit.
2. There are a few new sections which are new to this sprint.
3. Include reflections on ALL sections, as your experiences/thoughts/feelings will be different from sprint to sprint.

## Overall Experience

Sprint 3 was our opportunity to take our customers’ feedback and make the website of their dreams. As we implemented most of the backend stuff in Sprints 1 and 2, this time most of our user stories focused on creating the frontend web pages as well as advanced features based on our foundation and feedback.

We learned that it is vital to consider the feedback of our customers. We were able to present wireframes, or drawings of our planned web pages, and learn whether they are desired. We found that the current planned features were acceptable, but we learned the distinctions between officers and faculty which were vital in our design. For example, officers shouldn’t be able to view the scholarship information of members. With this feedback, we made separate pages for each type of user.

We had success in developing as a team as well. Everyone was able to focus on what they were good at. Our Frontend developers designed and wrote the html for the pages, and backend developers added the bit of functionality still required based on our customer feedback. Our scrum meetings gave us a chance to find any missing functionality not being worked on and assign it to others. We were able to finish all the core features desired, and so our maintenance phase going forward will be limited to vanity features or design changes as requested by the customer in our sprint review. Overall, this sprint was a win for teamwork and working for the customer.

## Stakeholder Management and Communication Plan

For Sprint 3, this plan was used to create a high-level view of who exactly is 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 led 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. [Reflect on the benefit of using this method and how it contributes to project success.]

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/Shared%20Documents/Recordings/New%20channel%20meeting-20221019\_150224-Meeting%20Recording.mp4?web=1

## Sprint Planning

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221027\_163110-Meeting%20Recording.mp4?web=1

## Scrum Meetings

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221031\_150650-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221109\_150836-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221110\_170641-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221111\_130905-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/New%20channel%20meeting-20221112\_131032-Meeting%20Recording.mp4?web=1

https://tamucs.sharepoint.com/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/Recordings/Meeting%20in%20\_APPSO-Parent-Portal\_-20221114\_151725-Meeting%20Recording.mp4?web=1

## User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number** | **User Story with its acceptance criteria** | | | | |
|  | **Persona (Who)** | **Requirement (What)** | **Value (Why)** | **Critical?** | |
|  |  |  |  | **Yes** | **No** |
| 1 | Scholarship Committee (mix of Officers and School Faculty) | I need to filter completed scholarship applications | so that it can be viewed and manipulated efficiently. |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Officer can view a list of members that have completed their scholarship application. |  |  |  |
|  |  | Officer can see the name and email of each user who completed the application. |  |  |  |
|  |  |  |  |  |  |
| 2 | Officer | I need to be able to check off if a member attended a event/meeting | so that I can know how to improve my standing |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Member sees the dashboard display immediately upon login |  |  |  |
|  |  | Dashboard includes application completion and link to edit application |  |  |  |
|  |  | Dashboard includes attendance data |  |  |  |
|  |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 4 | User (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 |  |  |
|  |  | **Acceptance Criteria** |  |  |  |
|  |  | Member sees the dashboard display immediately upon login |  |  |  |
|  |  | Dashboard includes application completion and link to edit application |  |  |  |
|  |  | Dashboard includes attendance data |  |  |  |
|  |  |  |  |  |  |
| 5 | Officer | I need to be able to visualize all members' involvement rankings | so that I can help decide scholarship winners |  |  |
|  |  | Acceptance Criteria |  |  |  |
|  |  | Officer should be able to click a filter to see a page ranking all the members based from highest involvement to lowest involvement. |  |  |  |
|  |  | Should only include valid members that have registered in the site. |  |  |  |
|  |  |  |  |  |  |
| 6 | User (member) | I would like to be able to see my own application completion percentage | so that I can budget my time for completing it. |  |  |
|  |  | Acceptance Criteria |  |  |  |
|  |  | Completion percentage appears on member dashboard in an easily understood format |  |  |  |
|  |  | Percentage is automatically updated upon changes to application |  |  |  |
|  |  | Complete range is only 0-100 |  |  |  |
|  |  | Completion bar is visible on live website. |  |  |  |
|  |  |  |  |  |  |
| 7 | Scholarship Committee (mix of Officers and School Faculty) | I would like be able to see members' application completion percentage | so that I can remind them to finish applying |  |  |
|  |  | Acceptance Criteria |  |  |  |
|  |  | Members' completion status appears in a list to the officer when they navigate to an appropriate view |  |  |  |
|  |  | Complete range is only 0-100 |  |  |  |
|  |  | Only scholarship committee can see this view, no one else should have access |  |  |  |
|  |  |  |  |  |  |
| 8 | Scholarship Committee (mix of Officers and School Faculty) | I would like to view completed scholarship applications | so that I can start to decide on who is the scholarship winner |  |  |
|  |  | Acceptance Criteria |  |  |  |
|  |  | Scholarship Committee can view a member's completed application from a list of members |  |  |  |
|  |  |  |  |  |  |
| 9 | Scholarship Committee (mix of Officers and School Faculty) | I would like to be able to make private notes on members' applications | so that I can remember my thoughts when considering scholarship applicants |  |  |
|  |  | Acceptance Criteria |  |  |  |
|  |  | Clicking on the notes should not be obtrusive to the committee member |  |  |  |
|  |  | Option to add notes to each member by viewing their scholarship application. |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

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

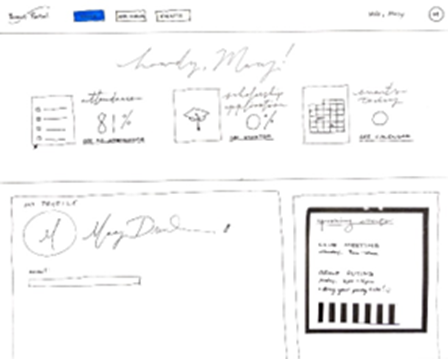
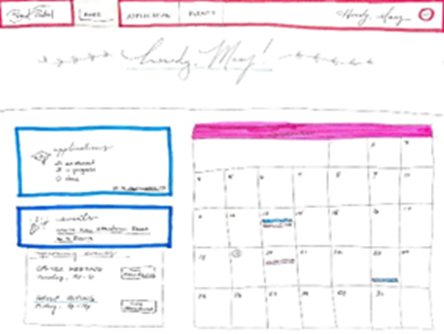
Should include:

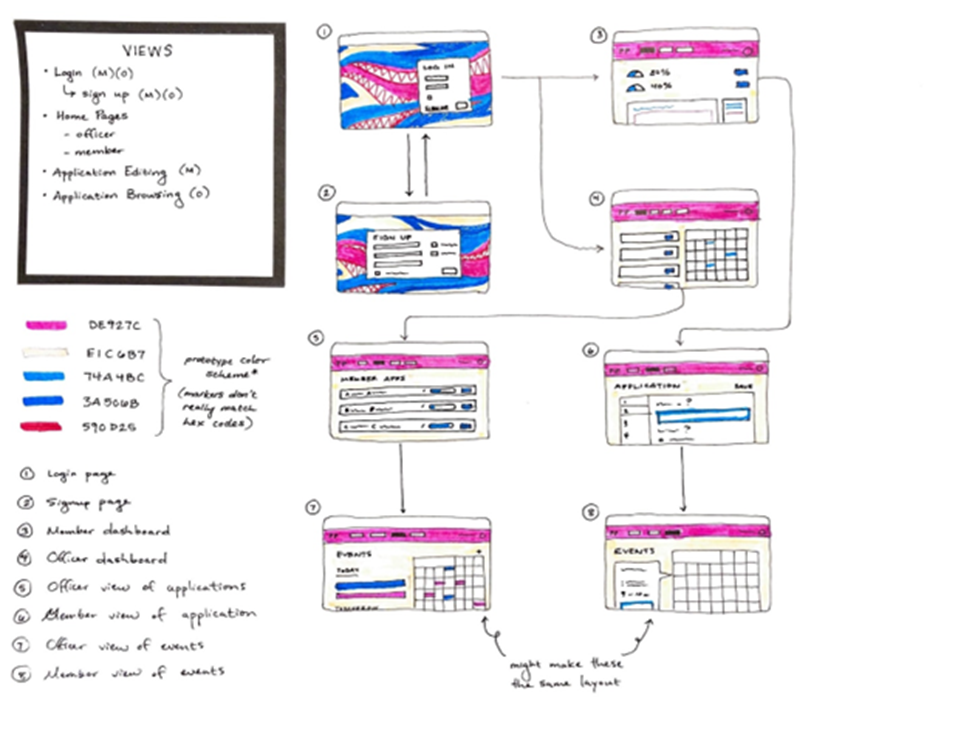
* Negative scenarios of the functionality (rainy day)
* Functional and non-functional use cases. Example of a non-functional use case is performance requirements
* What system/feature intends to do
* End-to-end user flow
* Impact of a user story to other user stories (i.e., features)
* UX concerns

[Reflect on the use of acceptance criteria and how it contributes to the success of your project.]

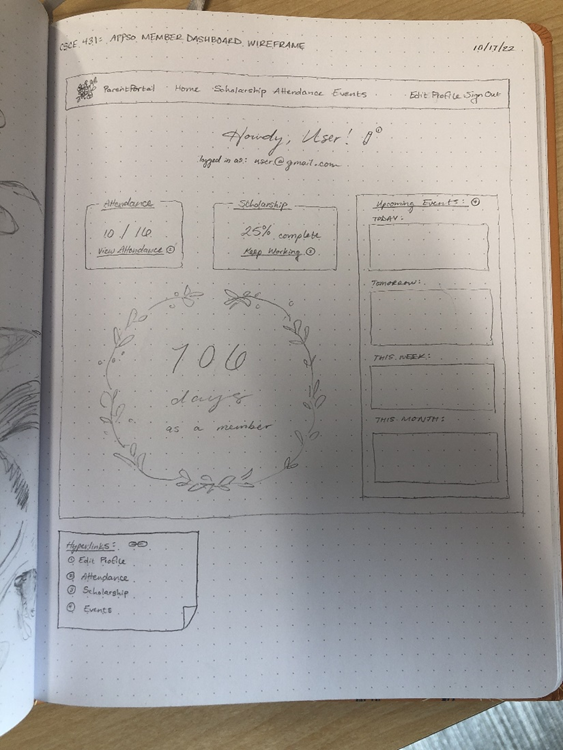
## UX Models

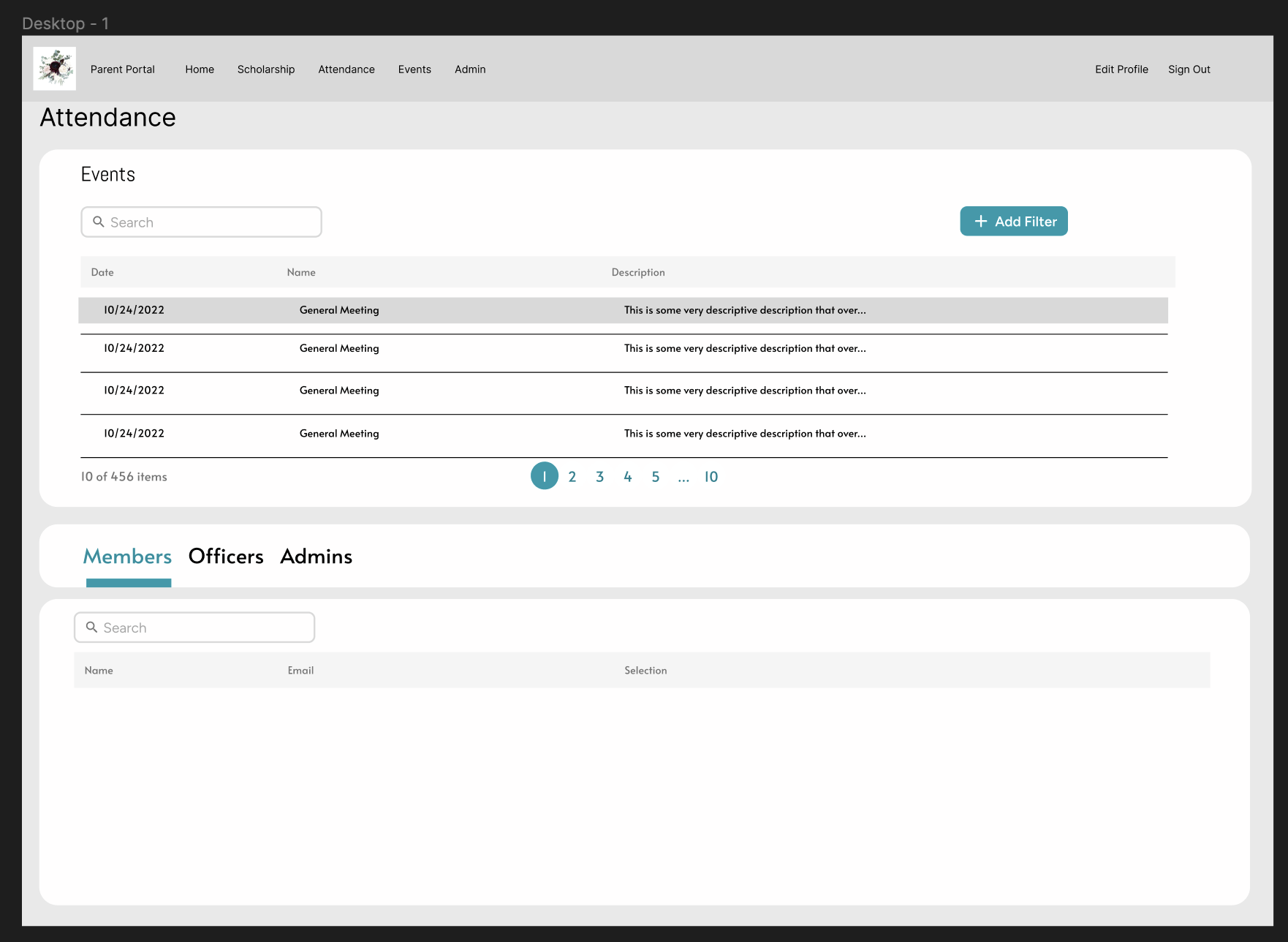
<Representative samples of UX models here and the complete set in the Annex.>





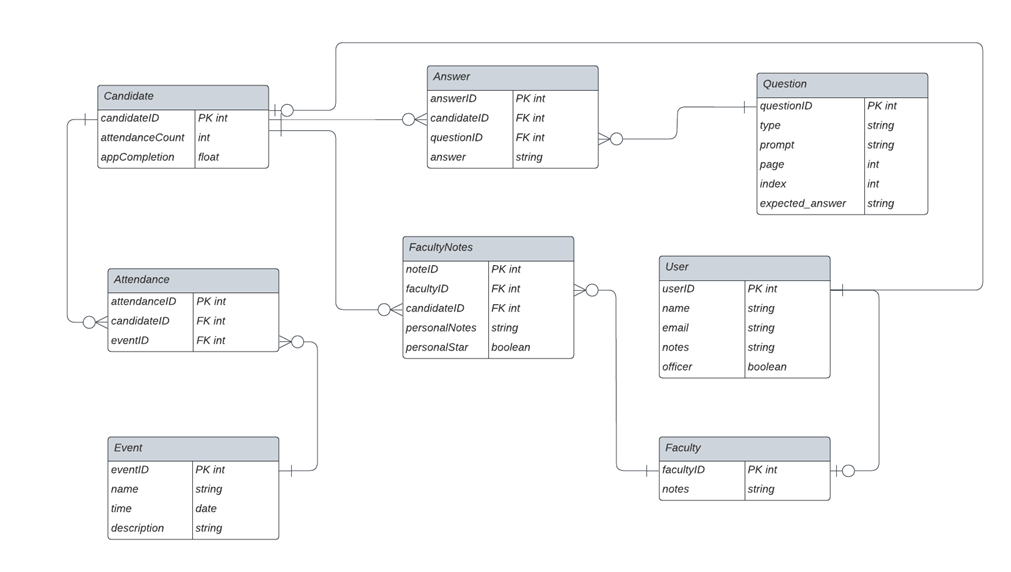






## Data Design

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 3 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 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.”

“As an APPSO officer, I need to be able to see specific and overall member rankings based on various stats so that I can make a data-driven decision on the winners.”

“As an APPSO officer, I need to be able to visualize members’ involvement rankings, so that I can help decide scholarship winners.”

“As an APPSO member, I would like to be able to see my own application completion percentage, so that I can budget my time for completing it.”

“As part of the APPSO Scholarship Committee, I would like be able to see members' application completion percentage, so that I can remind them to finish applying.”

“As part of the APPSO Scholarship Committee, I would like to view completed scholarship applications, so that I can start to make a decision on who is the scholarship winner.”

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 last sprint, we managed to complete all our most valuable user stories, and we are proud of that. We achieved 100% deployment for our most valuable user stories. More information can be found here: [APPSO\_Monitoring & Control Notes Sprint 3.docx](https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh)

### 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 150 story points to complete this sprint.

Based on this sprint burndown chart and the points completed as said in the backlog, we completed 217 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 managed to complete all our user stories that we needed to complete to satisfy the project scope deliverables. Stretch goals were present in our Jira but we removed them since they were merely additional (and time consuming) features.

More information can be found here: [APPSO\_Monitoring & Control Notes Sprint 3.docx](https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh)

### 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 quickly. More information can be found in this document: [APPSO\_Monitoring & Control Notes Sprint 3.docx](https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh)

A full detailed reflection and process of ensuring Monitoring & Control can be found here: [APPSO\_Monitoring & Control Notes Sprint 3.docx](https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh)

It’s also in our Documents folder for Sprint 3 as “APPSO\_Monitoring & Control Notes Sprint 3”

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 the following:

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 3, we **continued** 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.”

[Note: below is one that is recommended for each project, which the project team can add to, but not delete from.]

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |  |
| Last sprint: Deployment, support, and maintenance plans submitted and approved | Maintainability | Y | Only submitted |
| Final Sprint only: All pending issues / user stories resolved | Compliance | Y |  |
| Final Sprint only: Project turnover items prepared | Compliance | N/A |  |

#### 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 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. | Visible Indicator that 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 |  |
| “As an APPSO administrator/scholarship committee member, I would like to have the ability to add/delete questions, so that I can update the scholarship page in real time.” | Unit Test for valid inputs aka the Question Prompt | Redirection to the list of questions, showing that the new question has officially been added | Sunny |  |
| “As an APPSO administrator/scholarship committee member, I would like to have the ability to add/delete questions, so that I can update the scholarship page in real time.” | Unit Test for invalid inputs aka the Question Prompt | Notice/Alert showing the field can’t be empty and it is not valid | Rainy |  |
| “As an APPSO officer, I need to be able to keep track of events in a manner that makes it fast and simple.” | Unit Test for adding events | Notice/Alert showing event has been successfully created | Sunny |  |
| “As an APPSO officer, I need to be able to keep track of events in a manner that makes it fast and simple.” | Unit Test for adding an event with invalid input | Notice/Alert showing the field(s) are not valid being empty | Rainy |  |
|  |  |  |  |  |

**EDGE CASE Representative Case:**

**The following edge was in regards to when a user other than the faculty or admin (either a officer/member) tried to make a post request to a route that updates the notes for a specific member.**

**User Story: “As an APPSO Scholarship Committee member (faculty) I would like to be able to make private notes on members' applications so that I can remember my thoughts when considering scholarship applicants.”**

**Input: A post request to a route that updates the nodes for a particular member, which includes the notes content within the parameter.**

**Expected Output: An alert saying you need to be an Admin or apart of the Scholarship Faculty to update notes(a redirect back to the memberlist)**

|  |  |
| --- | --- |
| **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) |

For Unit tests, we added tests to ensure valid inputs were created for the Events and Question tables. Logic within the controllers checked to make sure empty scholarship responses from certain members were indicated whenever a member of the committee tried to view such an application.

For Sprint 3, we were able to create a plethora of integration (and also unit) tests that helped try to increase the code coverage of our application.

##### 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.

<https://tamucs.sharepoint.com/:u:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/ESd6RLEhBW9Pum38XHL5eagB-uhNCp4Ns6R_OJsq5g2dUQ?e=qf8CYs>

Initially our Code Coverage percentage was a lot lower after implementing the rest of our user stories, so we created additional unit/integration tests in order to get it to an acceptable level. Although we did not quite reach 90%, 87% came extremely close, and we were satisfied with these results. This is still a good code coverage metric number, and indicates that the majority of our code has been covered by one or more tests.

#### Test Results

Prior to releasing the app to the customer, we tested it thoroughly to ensure that no bugs escape to the customer.

As of the writing of this document, all our tests passed. We do not have any outstanding tests that have failed. Initially we had a couple of errors that produced errors since rspec lacked a proper driver to conduct JavaScript based tests. So, Thierry had to install a headless selenium chrome driver to run these tests. After installing this, the tests managed to pass, so that was a huge sigh of relief.

We also conducted a usability test by presenting the website to a roommate. We timed his time of acclimation to 15 minutes or less. After distributing the user stories and acceptance criteria to him, they were able to complete all of the acceptance criteria in about 8 minutes.

However, he was a computer engineering student, who was acclimated to computers, so times may vary.

As of writing, we have not seen any bugs that escaped to the Customer. If bugs were to escape to the customer, we will collect information regarding the steps to reproduce, and create a pull request to make necessary bug fixes to the application.

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

**Instructions for creating an account:**

First, navigate to the URL linked above, which will redirect to our website login page.

Next, either Sign up or Sign in with Google.

If you sign in with Google, select your tamu email address to continue.

If you sign up, follow the directions to provide a full name, email address and password.

Once you have proceeded, select which type of account you’re registering for. This will need to be confirmed by a current admin, so if you intend to be a member select the member radio button and Submit.

Finally, you’ve made an account and can view your home page. Hooray!

For additional functionalities such as viewing scholarship information, you’ll need to wait for a current admin of the site to approve your registration. Once that notice goes away, you’ll be free to enjoy and test the full extent of the site through the links in the toolbar.

Note: You can approve your account by logging into the admin account (provided below), heading over to the Approvals link in the navigation bar, and clicking approve on your own name.

Admin login:

[email@e.com](mailto:Email@e.com)

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 bugs within the web application itself. Since it was our thirdsprint, our mean time to deploy fixes pretty quick now, at around 1 or 2 hours max when receiving a ticket and fixing the bug associated with such ticket. We also closely monitored 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/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh>

#### Coding Standard

We also reviewed our code against guidelines of clean code as specified in our coding standard, which is included below. [Your project team can add more, however not recommended to delete any.]

|  |
| --- |
| **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 snake\_case 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. |

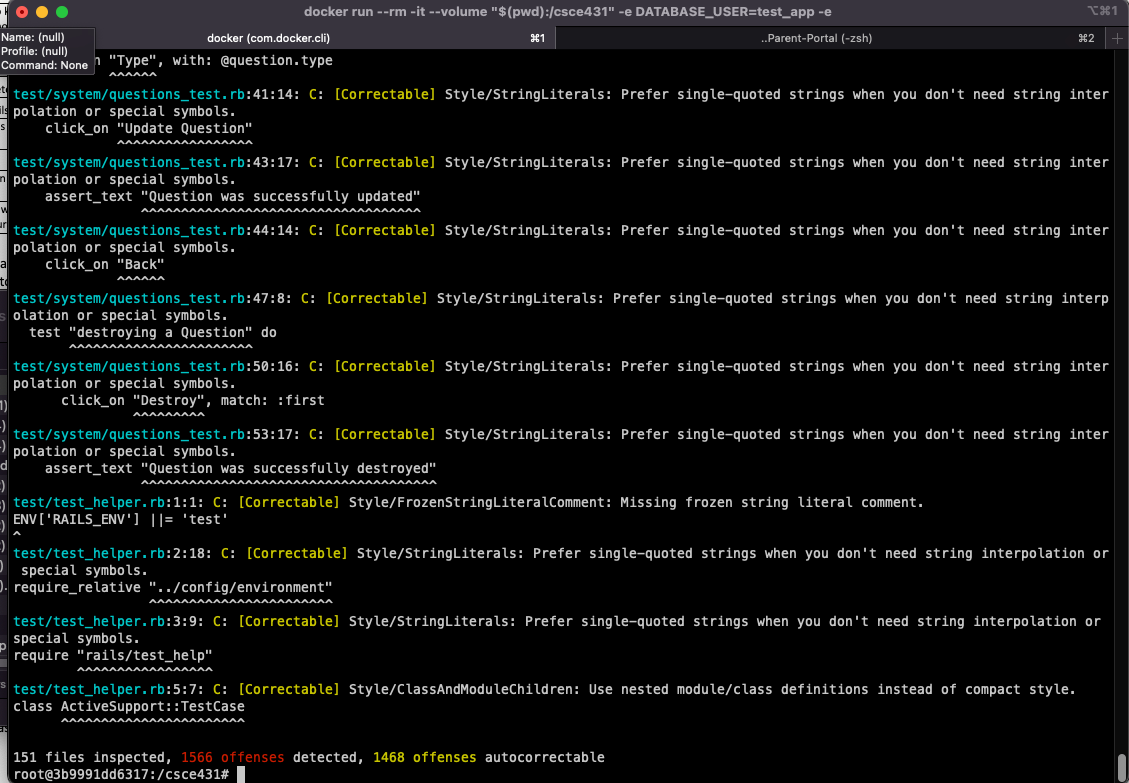
Adhering to a coding standard allowed us to have cleaner and most importantly, readable code. Keeping the style and organization of code to a standard allows us to focus more on functionality and implementation rather than understanding how a piece of code that is typed non-professionally.

#### 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 the rest of our user stories done while maintaining stability for this last sprint. But as seen in the output image above, we could always have rubocop correct these offenses and review manual ones in the 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 really have to thank Brakeman for being such an excellent tool to assess the security of our application. We frequently check the brakeman output every time we make changes to the code in our respective branches to make sure we haven’t created anything vulnerable.

#### 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)

#### 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:

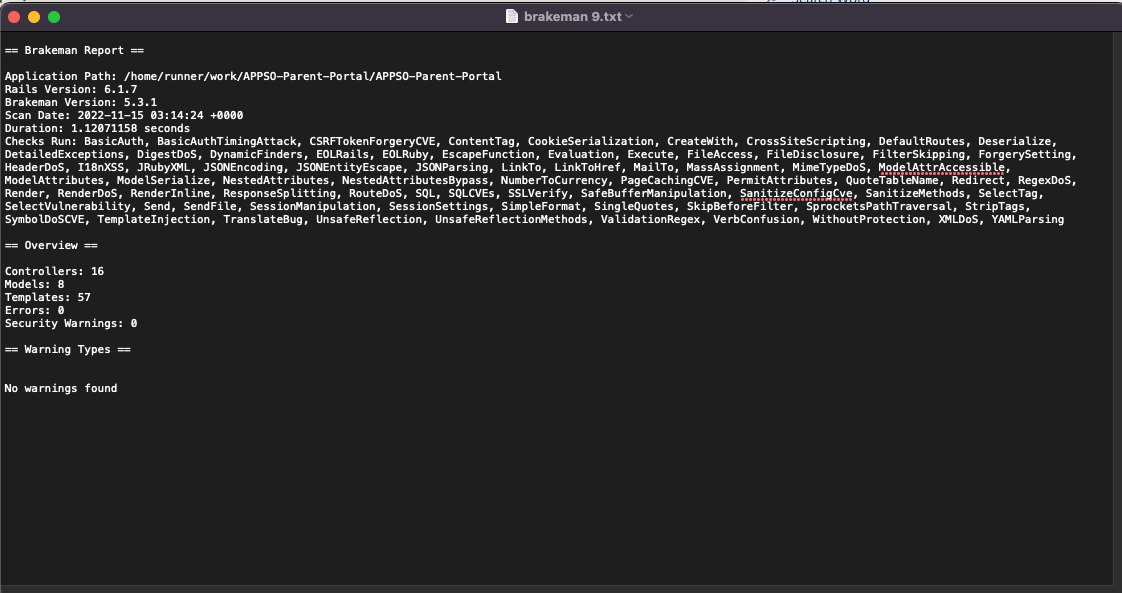
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User Story** | **Input** | **Expected Output** | **Sunny / Rainy Day** | **Detail that can help with the test (e.g., execution process, assumptions, etc.)** |
| As an APPSO Scholarship Committee member (faculty) I would like to be able to make private notes on members' applications so that I can remember my thoughts when considering scholarship applicants. | Malicious user makes a post request to a route that updates the nodes for a particular member, which includes the notes content within the parameter (bypassing front-end form) | A redirection to an alert saying the user is not allowed to perform such action. | Rainy |  |
| As an APPSO Scholarship Committee member (faculty) I would like to be able to make private notes on members' applications so that I can remember my thoughts when considering scholarship applicants. | A malicious user in the form of an officer tries to access the notes of a certain member by typing in the DIRECT ROUTE LINK (‘/memberlist/id/notes/) | A redirection to the memberlist saying only scholarship committee members can only view the notes. | Rainy |  |

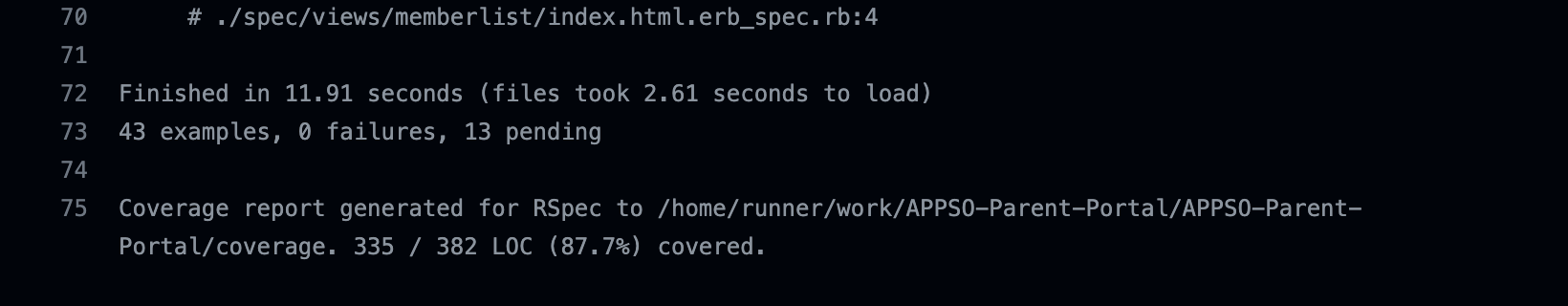
These were mainly integration tests which tested to see if we had protected our routes in terms of updating data in regarding to notes on members for the organization scholarship.

We frequently checked on the brakeman report created whenever we pushed changes to our own feature branches, and the main branch. We checked to make sure there were no warnings and we handled how we collected parameters from pages, that are later saved to the database. We also made sure that only users of the right roles had sole access to exclusive actions by creating check functions within controllers of the respective pages.

#### Integrity Test Results

We ensured that our RMMM plan worked to address common security vulnerabilities by unprotected routes, SQL injections, XSS Scripting, etc.





[Describe your test results, including tests that passed and failed.]

All our tests' results in regard to integrity passed. For brakeman, we had no warnings within our controllers, models, and templates that posed potential threats to our application’s security. Our tests to make sure unauthorized requests and actions within our application also passed (in regard to viewing member data without the right user role).

As of writing this document, no vulnerabilities were discovered by the customer. If vulnerabilities were discovered by the customer, we will follow the respective RMM plan to mitigate such vulnerabilities.

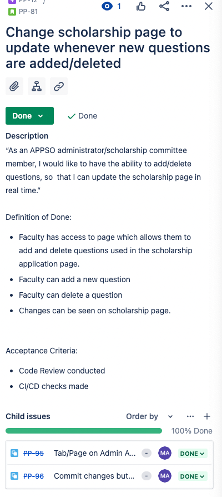
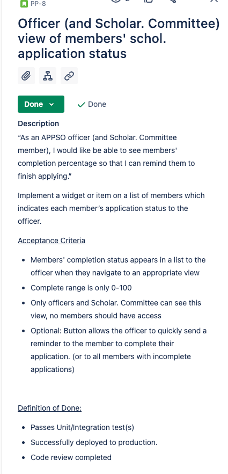
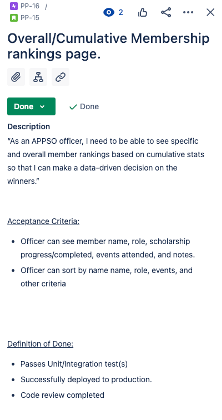
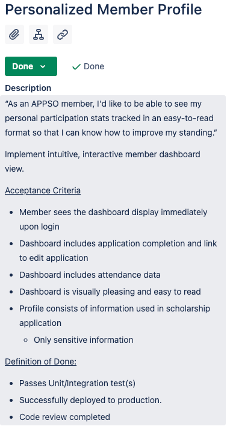
### Usability

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. 8-9 minutes. Usability metrics will be improved upon in future sprints.

More Information in our Monitoring & Control notes: [APPSO\_Monitoring & Control Notes Sprint 3.docx](https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh)

#### 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:



#### 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 Scholarship Committee member (faculty) I would like to be able to make private notes on members' applications so that I can remember my thoughts when considering scholarship applicants.” | Click link to view notes on a particular member | A page that shows the notes for that user. | Sunny |  |
| “As an APPSO Scholarship Committee member (faculty) I would like to be able to make private notes on members' applications so that I can remember my thoughts when considering scholarship applicants.” | Click link to updates notes regarding a particular member | A form that allows a admin or scholarship committee member (faculty) to updates notes for a particular member. | Sunny |  |
| “As an APPSO 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.” | Click a link to sort the member list by Name, Email, Role, or Scholarship Progress | A updated which is listed from most to least valuable, depending on the sort value (Ex. For scholarship progress highest scholarship completion percentage is first). | Sunny |  |
| “ As an APPSO officer, I need to be able to check off if a member attended an event/meeting, in order to keep track of member attendance” | Click Present for a member on a particular event, in the attendance page | Visual indication showing the status of the student’s presence being “present”, along with their table entry being green. | Sunny |  |

We are creating a plethora of integration tests (along with our own manual testing) in order to fully determine the usability of the system. We create integration tests which mock typical behavior/usage of the system and checked to see if all tests ran smoothly.

Creating these tests was quite tiring, as there are a lot of behavior events to be made with an application like this. Certain tests were also failing since we needed to enable JavaScript for RSpec, which included including an ENTIRE headless chrome driver. This actually made it where the docker container provided by the CSCE 431 teaching team would not be able to handle JavaScript based integration tests, since the docker container does not have chrome installed. However, the tests worked just find on GitHub Actions, so it was manageable.

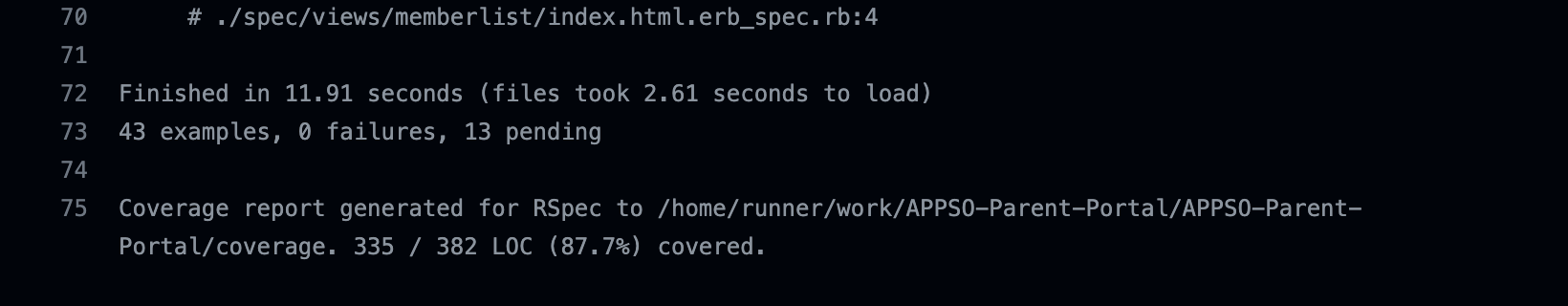
#### Usability Test Results

We ensured that our system was usable by creating a bunch of integration tests that helped simulated typical behavior of a user.

We also conducted a test with Thierry’s roommate (the one from last sprint), where given a set of user stories and their acceptance criteria, how fast they can check off all these acceptance criteria. We managed to get a time of around 8-9 minutes. We made sure to tell the participant to carefully review the acceptance criteria too, since they show how **usable** the system is when the features that these acceptance criteria describe are included.

More infomration regarding our tests for Usabilty are included here: <https://tamucs.sharepoint.com/:w:/t/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/EUxNTCEX9ZFFowwCVKHcLjIBBGYAs-4ftxxX6Wn4BDPvFQ?e=BKaOUh>

The usability tests In table were combined with these test results:



## 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 150 story points per sprint

* + Why 150 story points? This is our last sprint before UAT and Turnover, so we needed to increase the number of story points completed this sprint to reflect the remaining user stories left.

Compliance: 100% of the most valuable user stories deployed

* + This is a non-negotiable, for us, by sprint 3, we should have implemented all of the most valuable user stories that help complete the functionality of the website

Correctness: 0 errors escaped to the user

* + We defined 0 errors escaped to the user as: No Errors visible in the production app.

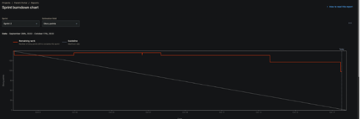
Integrity: 0 security breaches

* + We check for security breaches by continuously running brakeman while we are implementing user stories and during CI/CD GitHub Actions tests. We also create integration tests that test how secure the application is from malicious individuals from accessing pages they are not allowed in.

Usability: Intended user learns how to use the system in 15 minutes or less based from a set group of Acceptance Criteria that adheres to how usable the system is.

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

We also used a sprint burndown chart to track our progress throughout the sprint, as it helps show our velocity as we accumulated story points:



It’s important 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 continued our great progress from sprint 2, by continuing to create monitoring & control notes and tracking metrics for each criterion (Efficiency, Compliance, Correctness, Integrity, Usability, and Maintainability). We did a great job last sprint, so we continued to achieve the same level of note taking for this last 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

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)

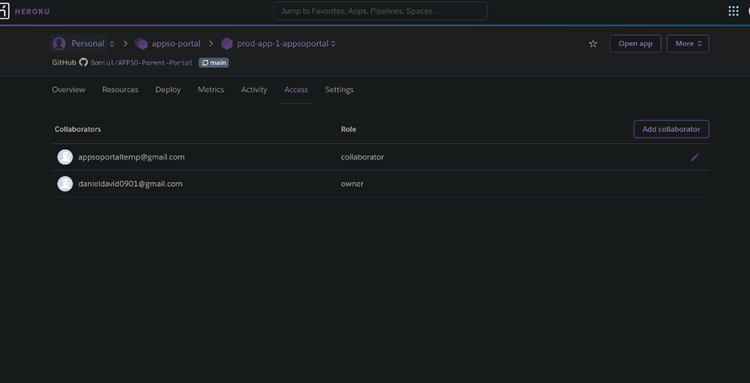
pass: C&Ux5ATsS%ZSnj

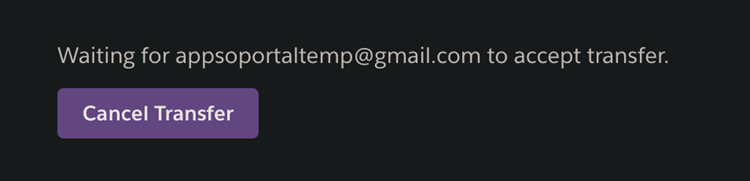
Heroku account details (where the app will be hosted):

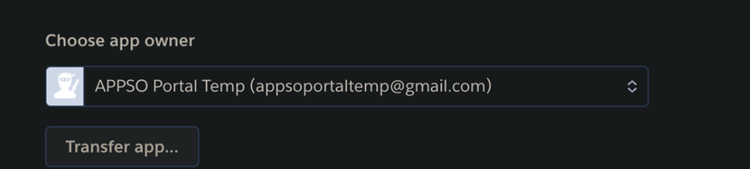
email: [appsoportaltemp@gmail.com](mailto:appsoportaltemp@gmail.com)

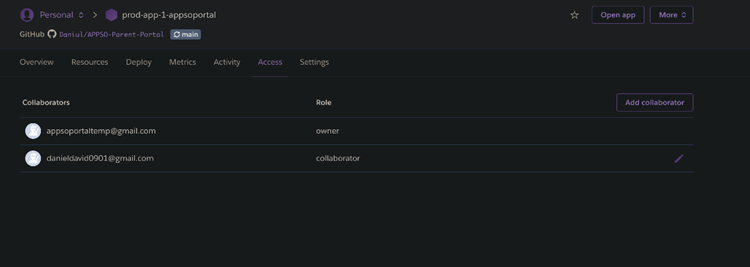
pass: yHf%#Hq4s!L6jV

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, November 16. The video will be made available on Teams after that time.

## Sprint Retrospective

At the time of the writing of this report, the Sprint Retrospective has not yet occurred, and is scheduled for Friday, November 18. 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 3. 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 |

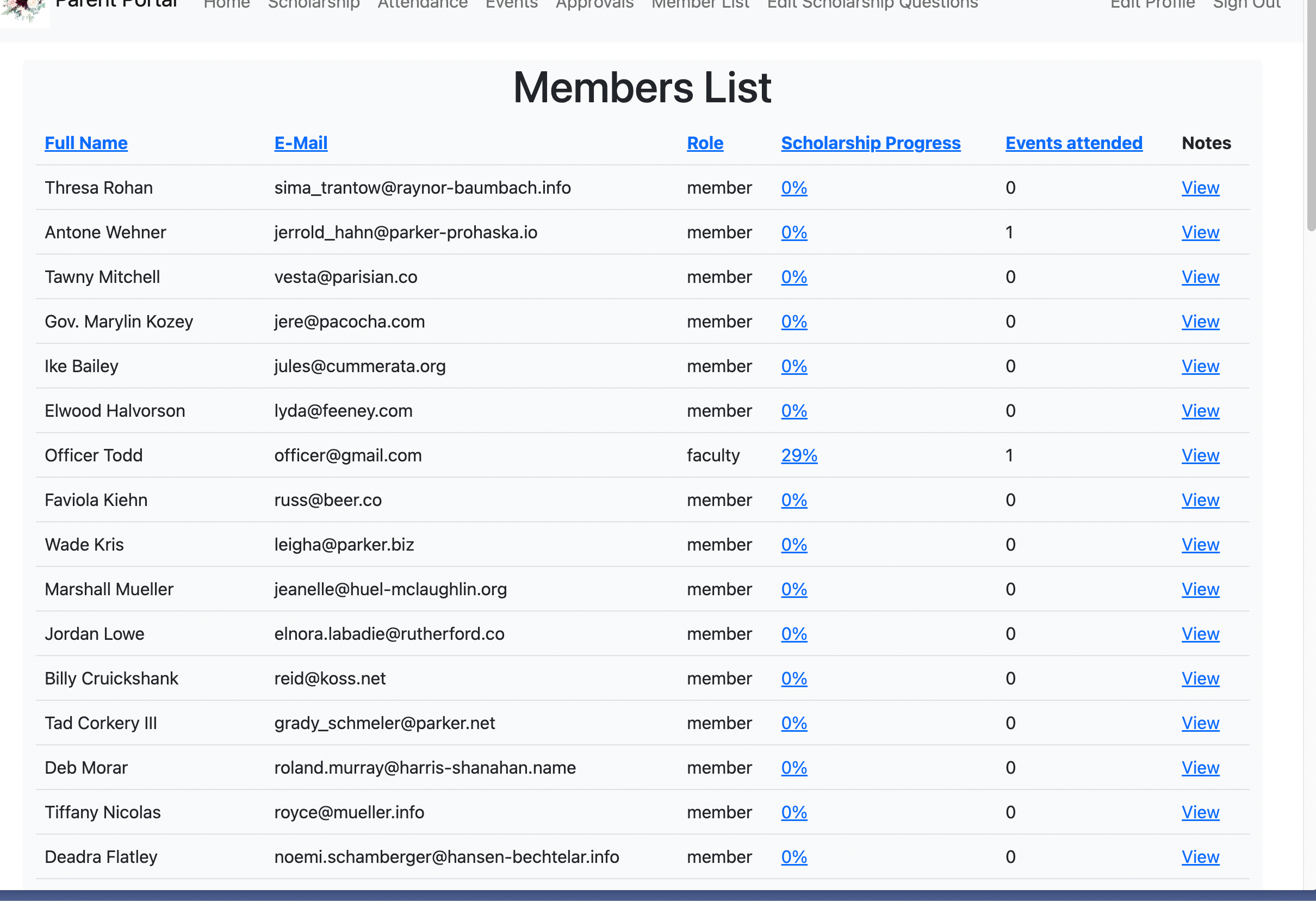
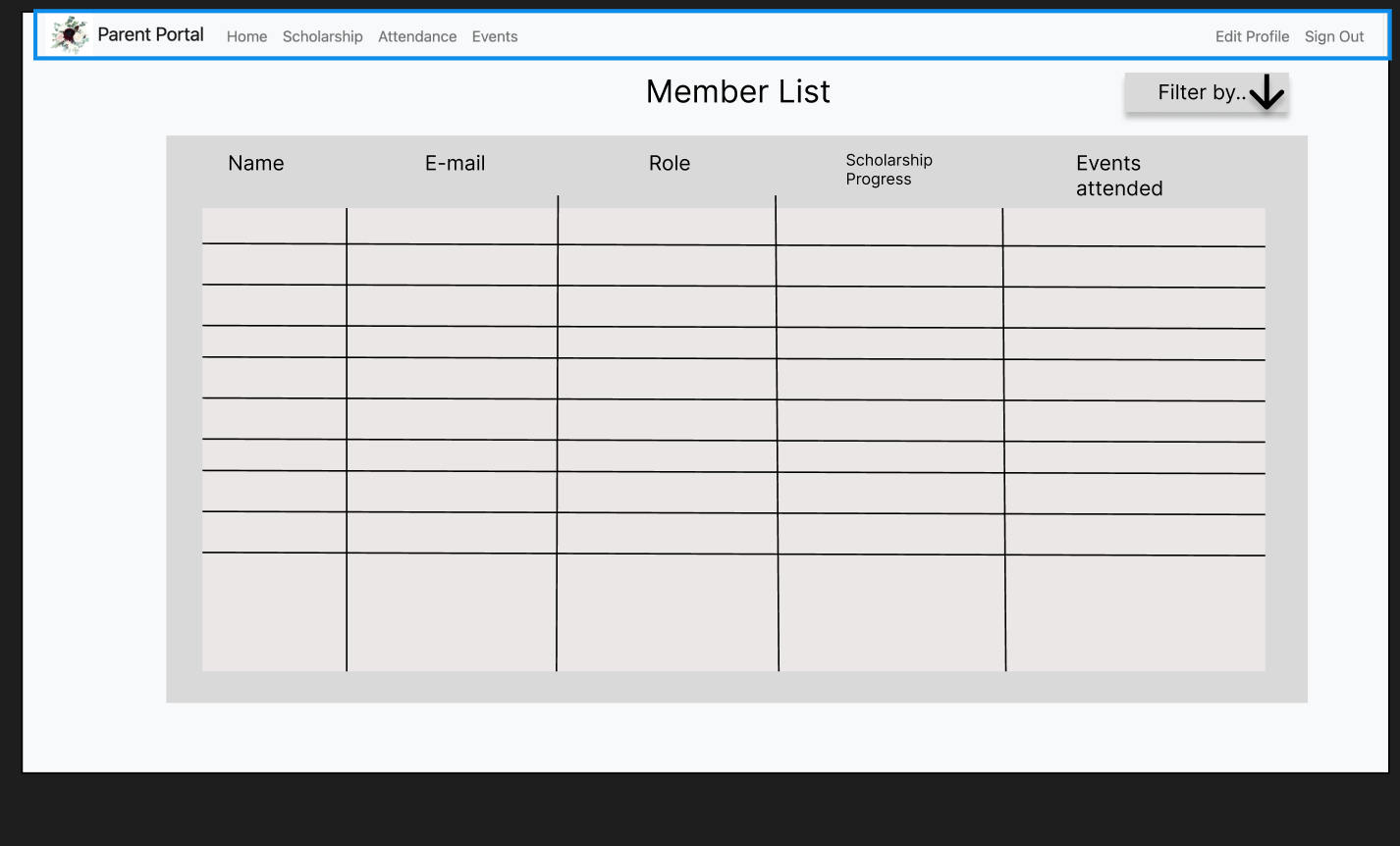
# User Acceptance Test Form

Before the customer can perform user acceptance test (UAT), we furnished them with a list of user stories implemented (and not implemented), using the UAT form templates below:

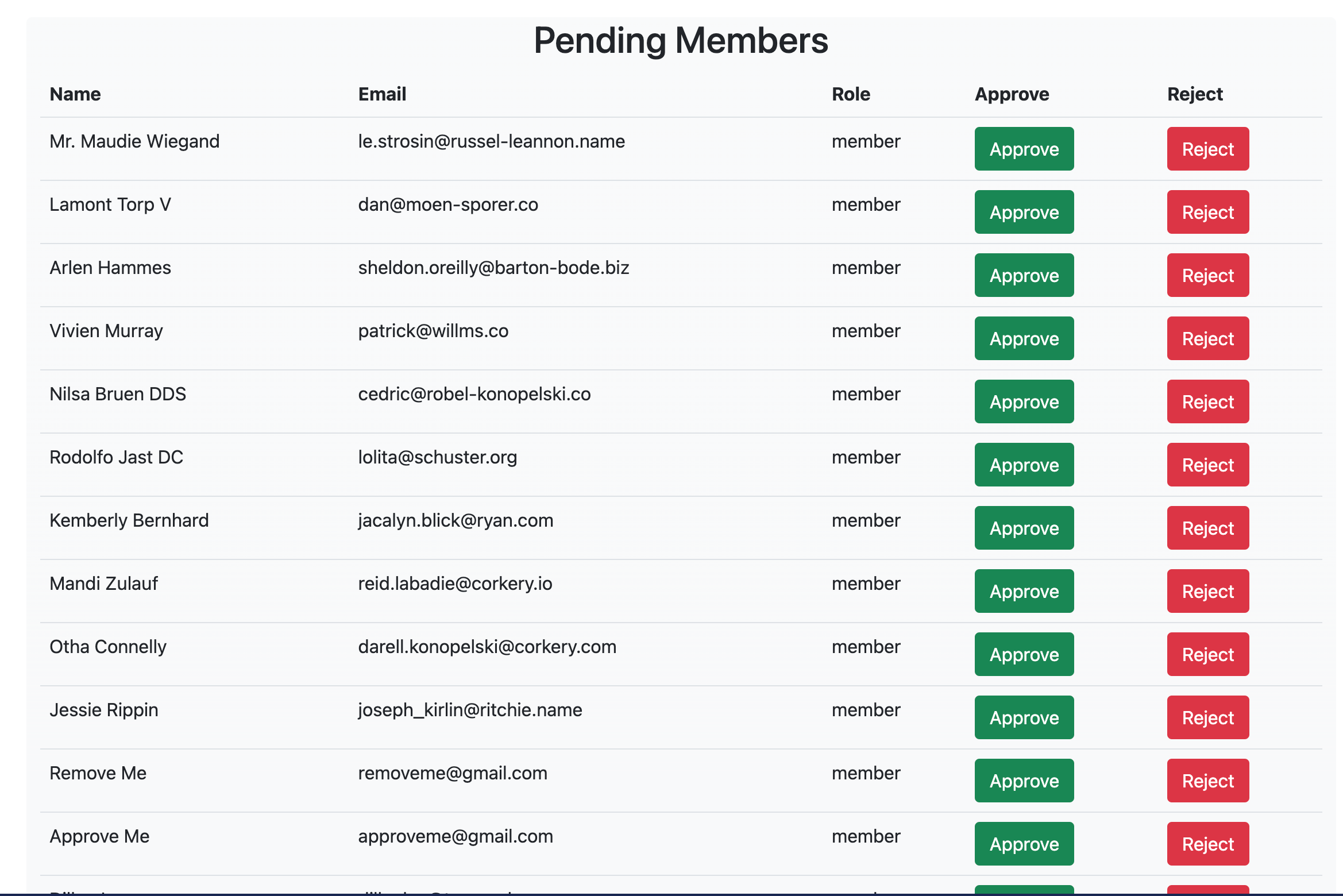
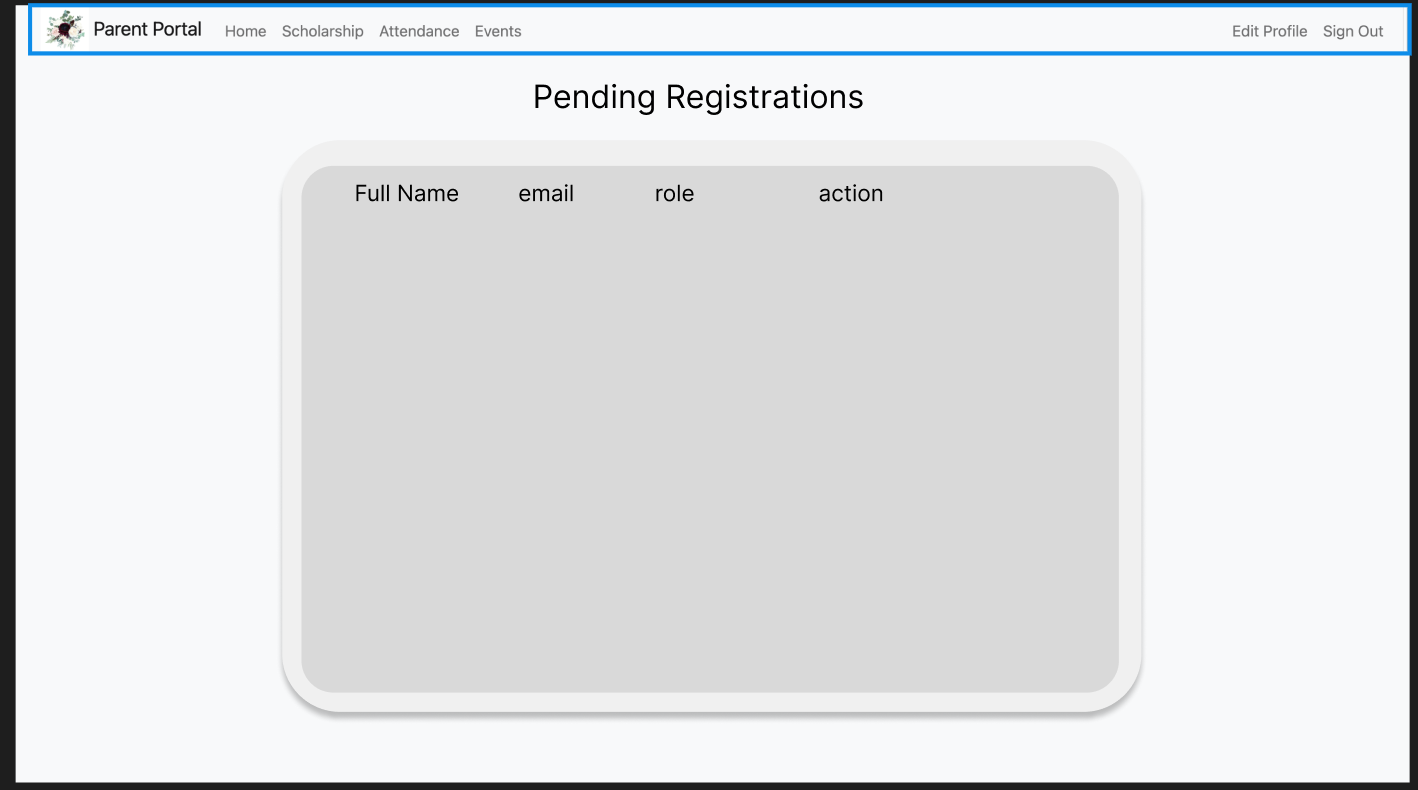
<https://tamucs.sharepoint.com/:x:/r/teams/Team-FA22-CSCE431-SoftwareEngineering-AggiePPStudentOrg/Shared%20Documents/APPSO-Parent-Portal/Sprint%203/Documents/APPSOPortal_UATForm.xlsx?d=wa290c502ce1146898bd4e84c4d6a6dc4&csf=1&web=1&e=eBJsiv>

# Annex: Screenshots of the UX Models / User Interface

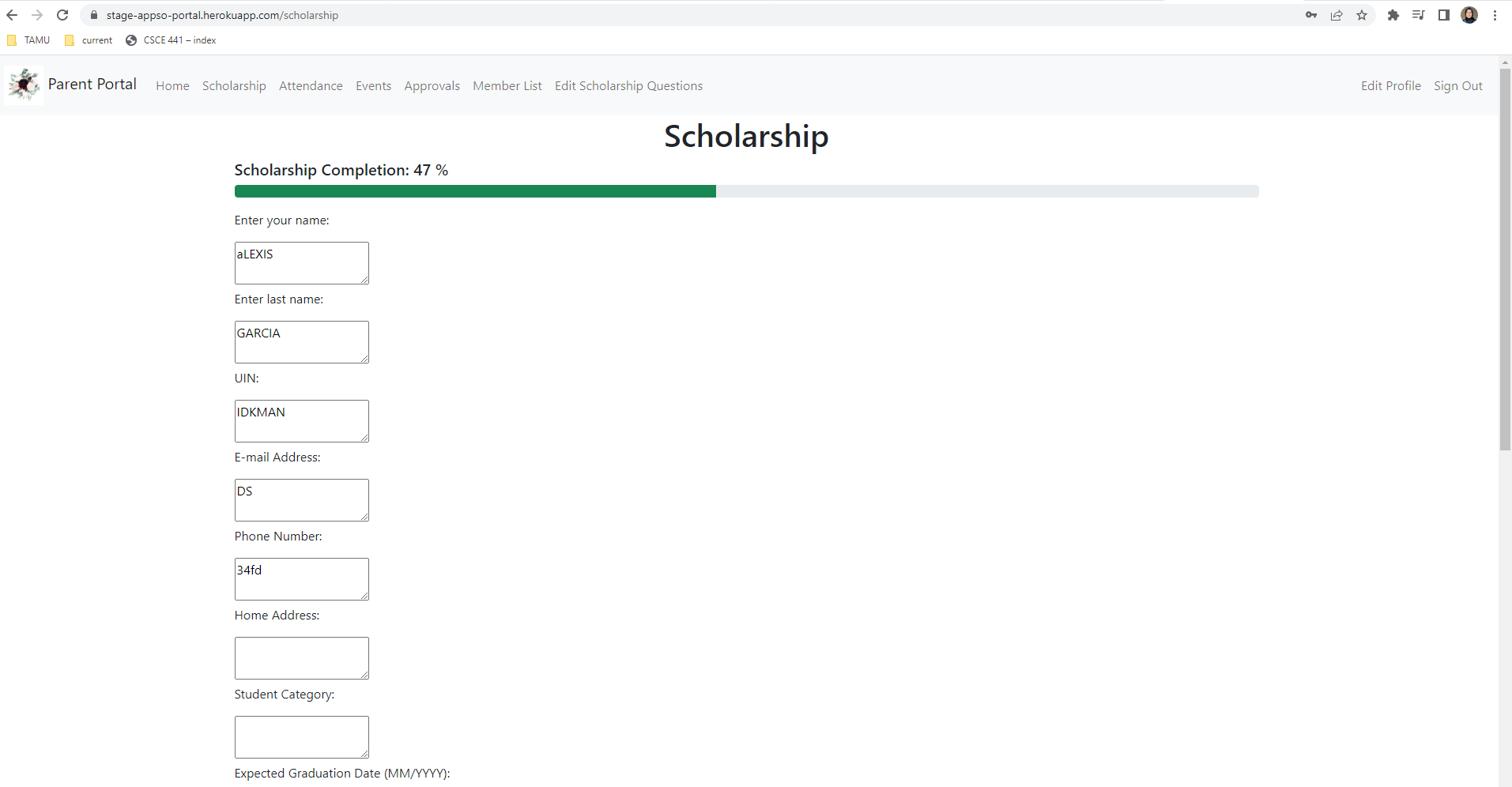
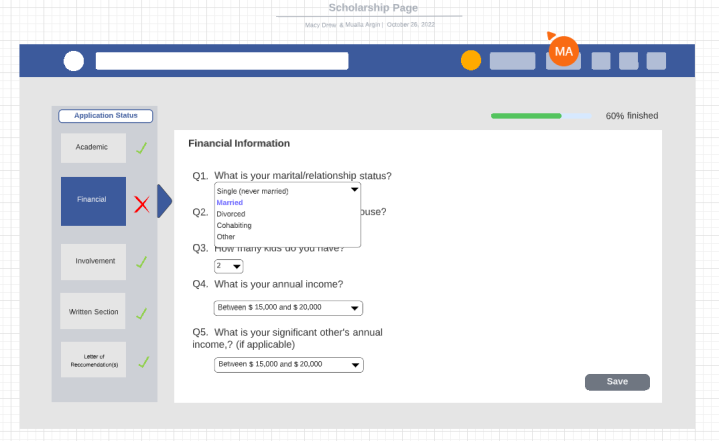
Members List Page:



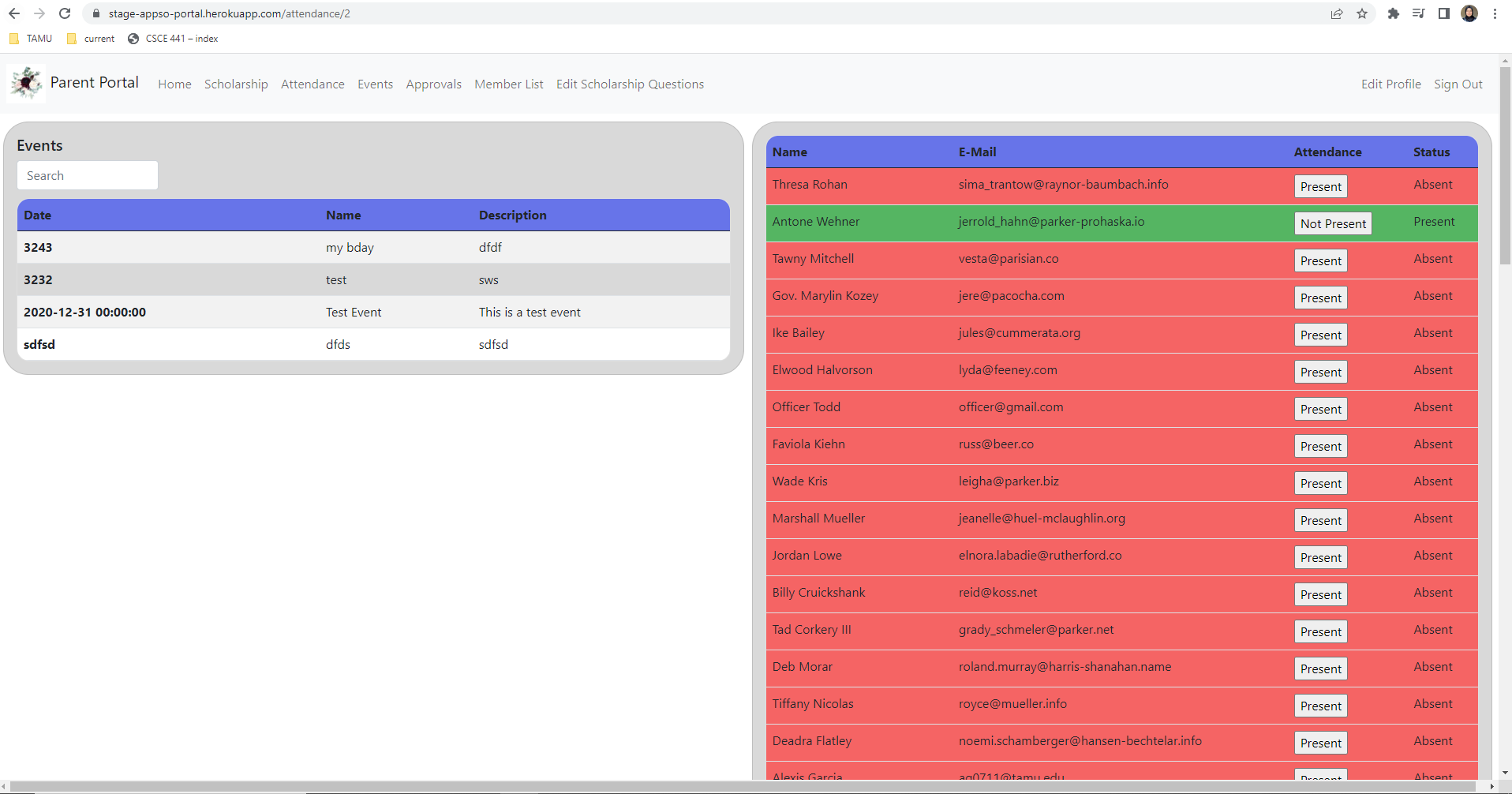
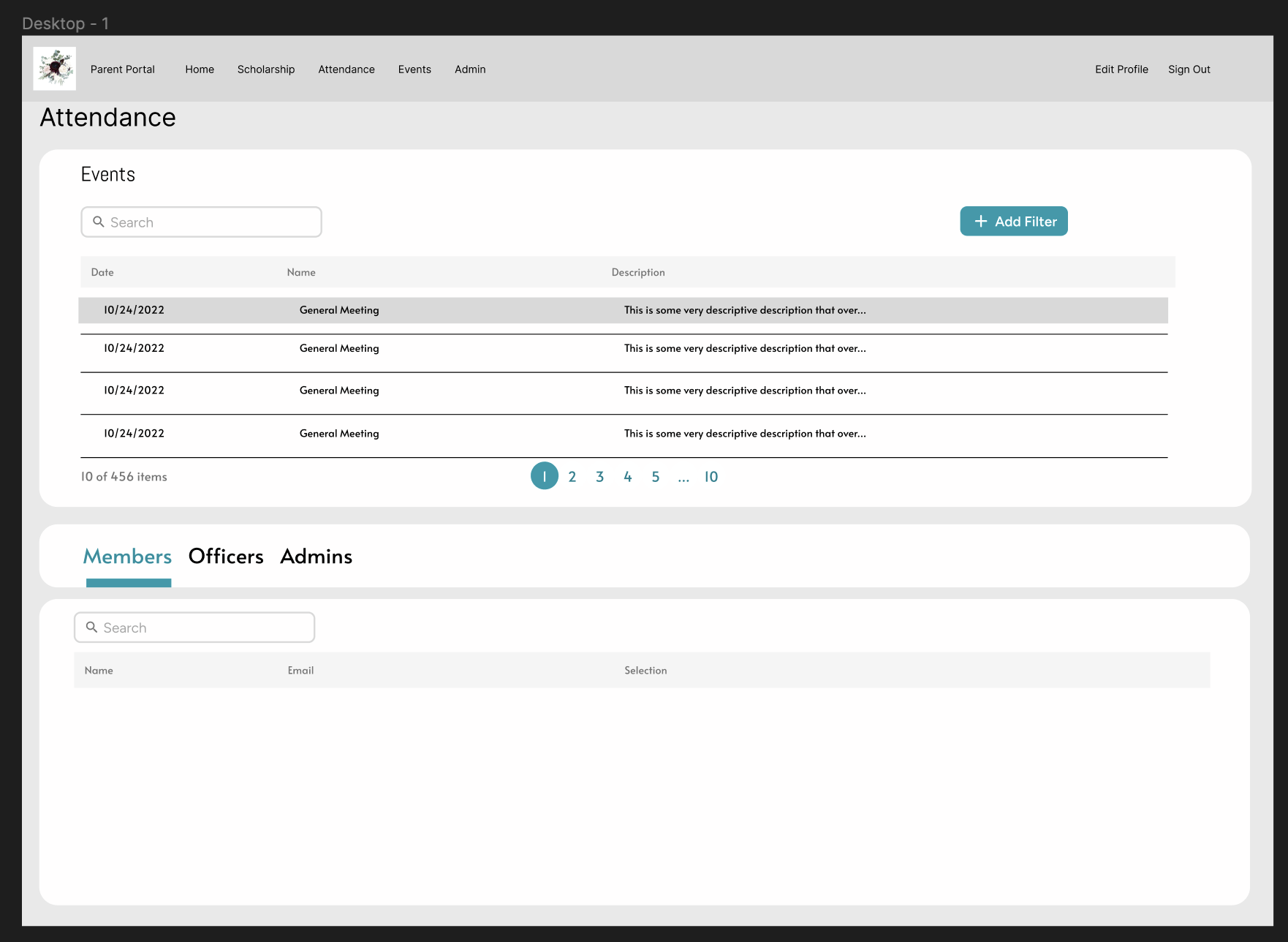
Account Approval Page:



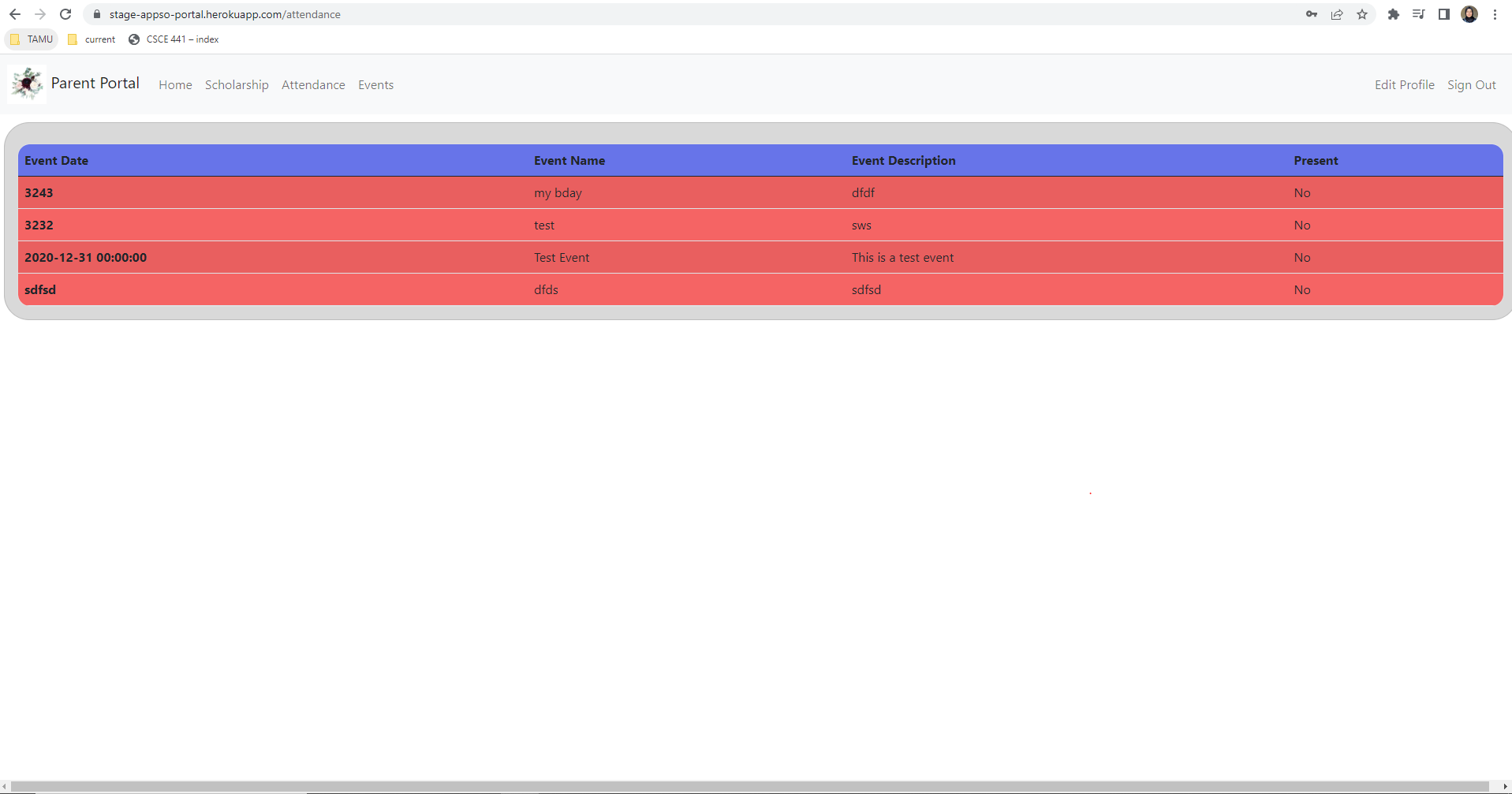
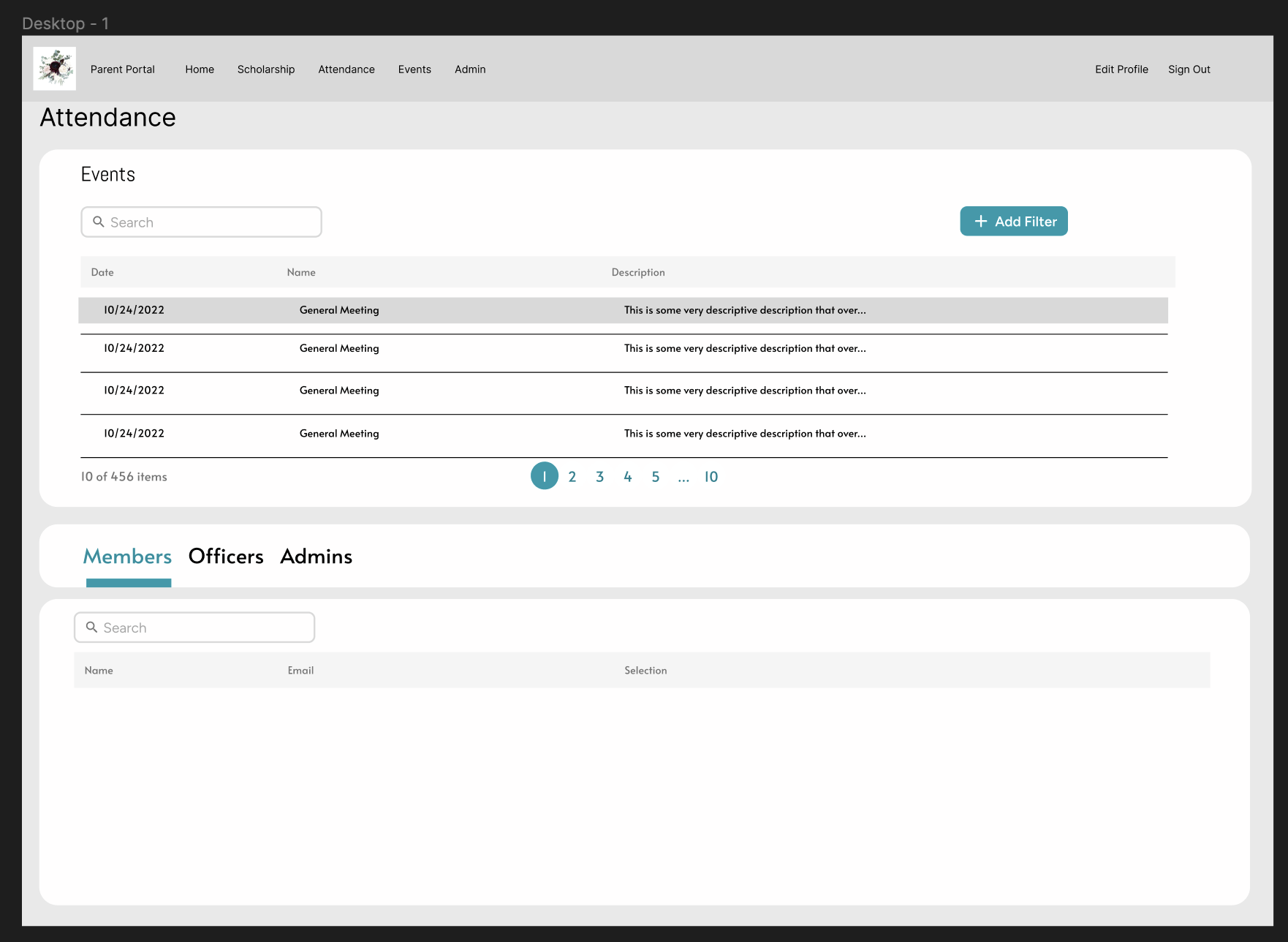
Scholarship Application Page:



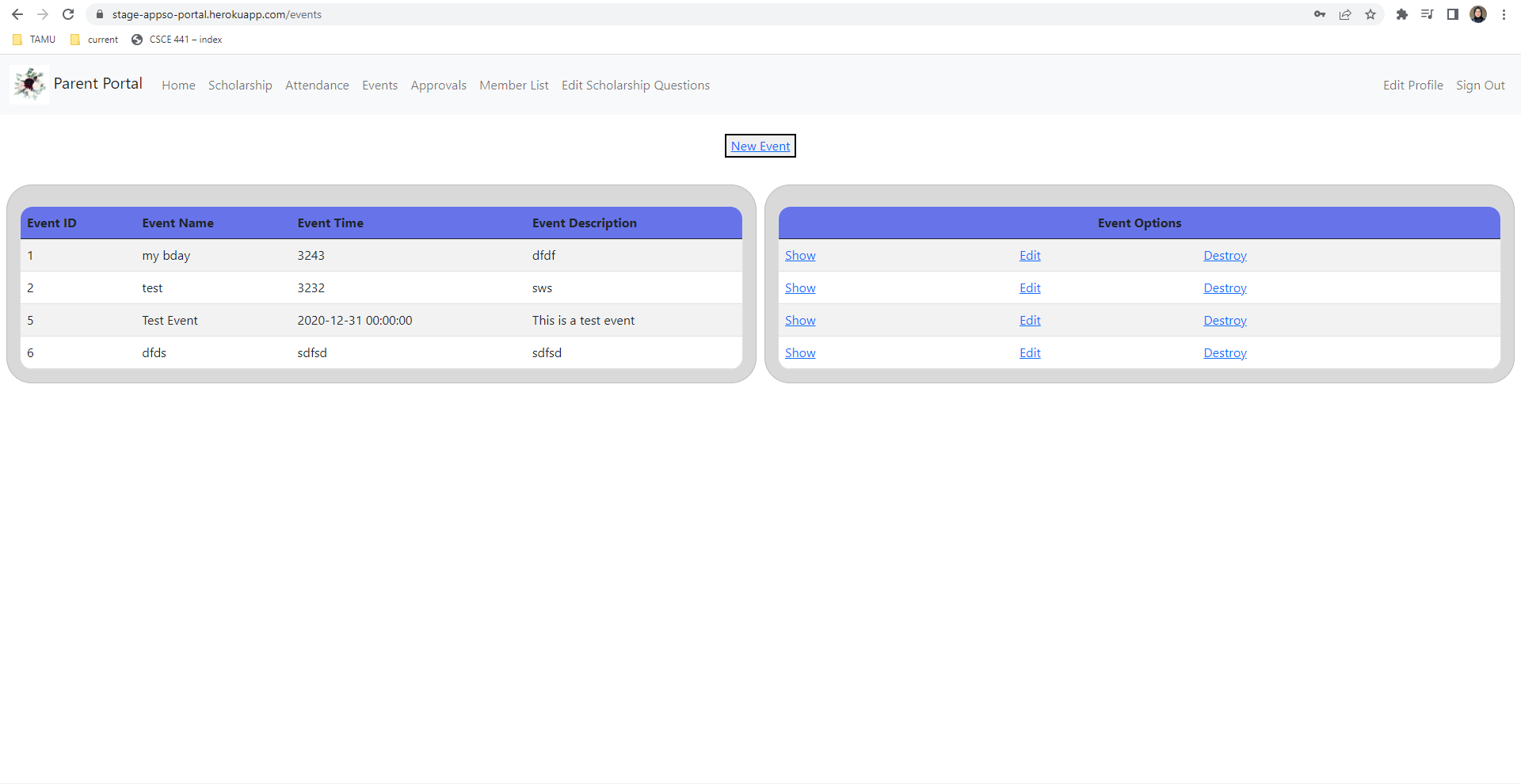
Admin Attendance Page:



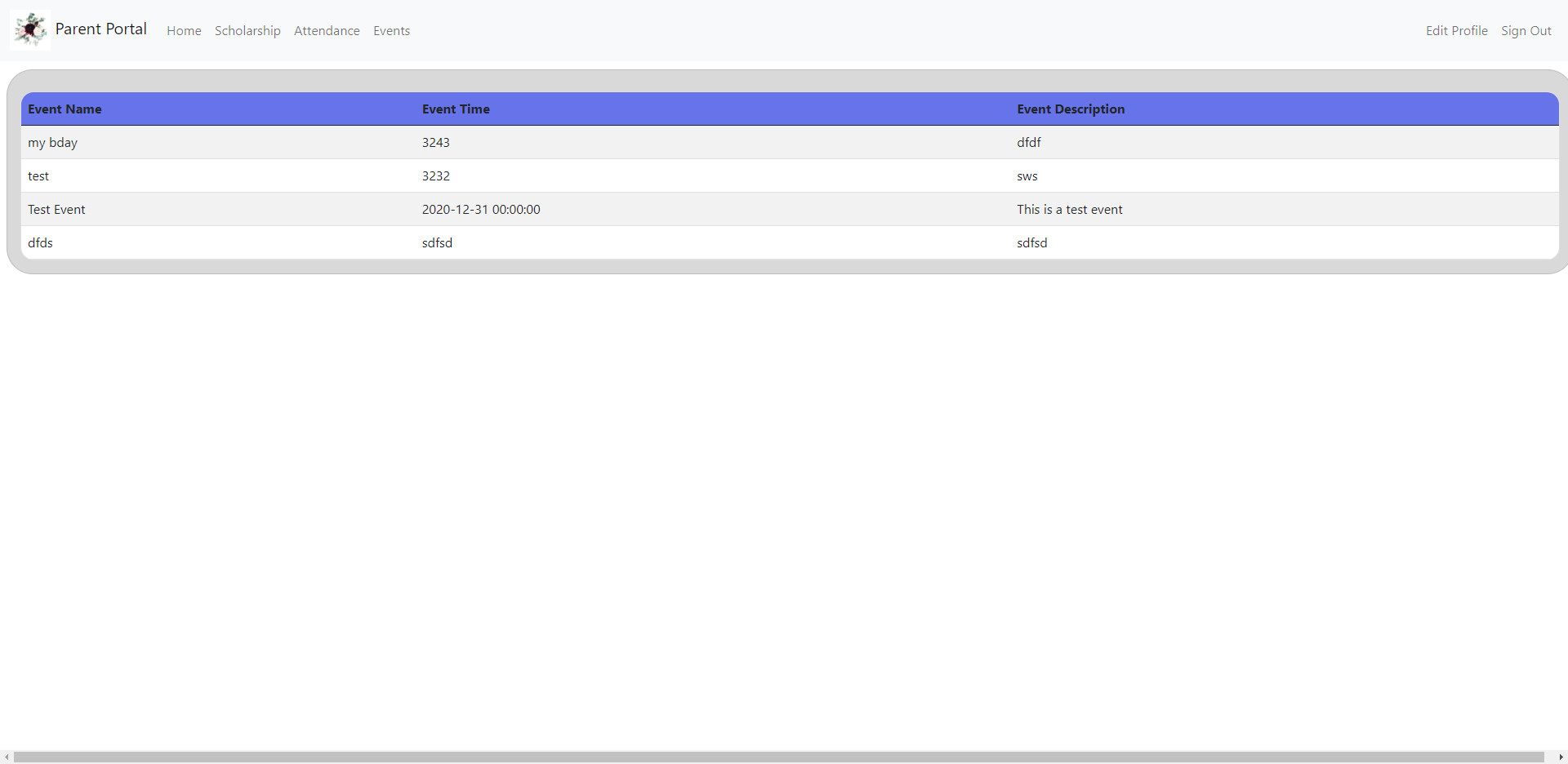
Member Attendance Page:



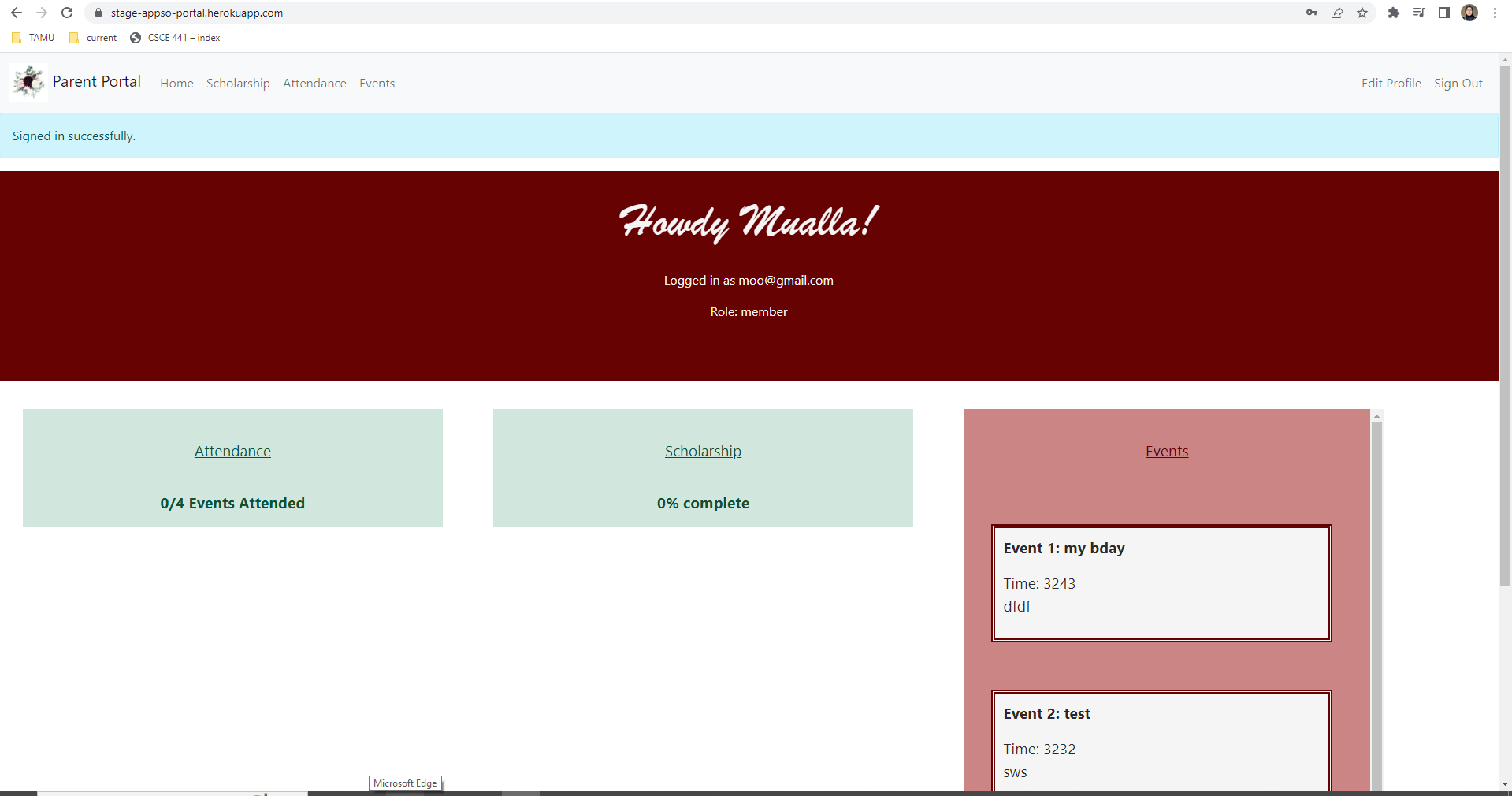
Admin Events Page:



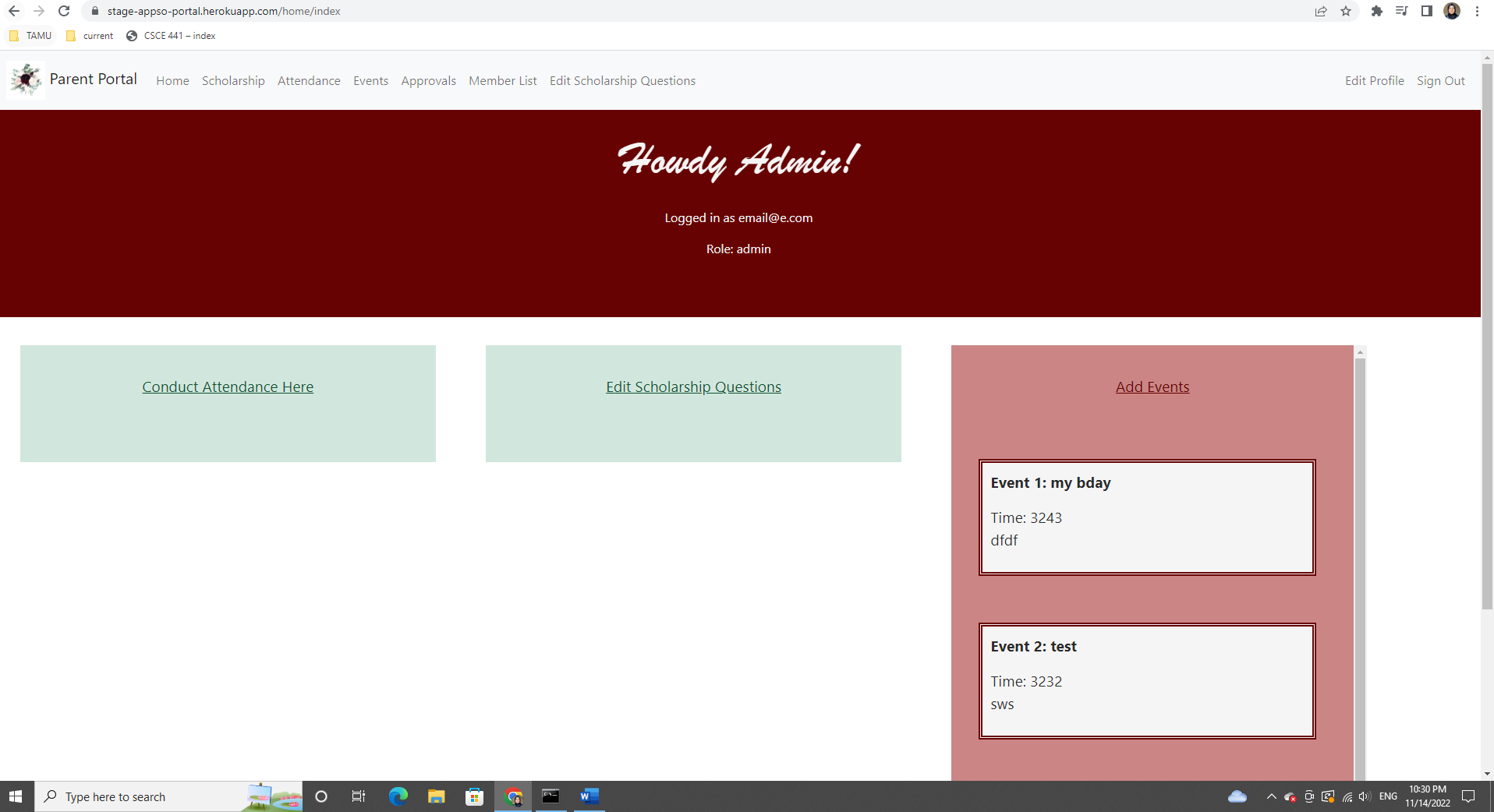
Member Events Page:



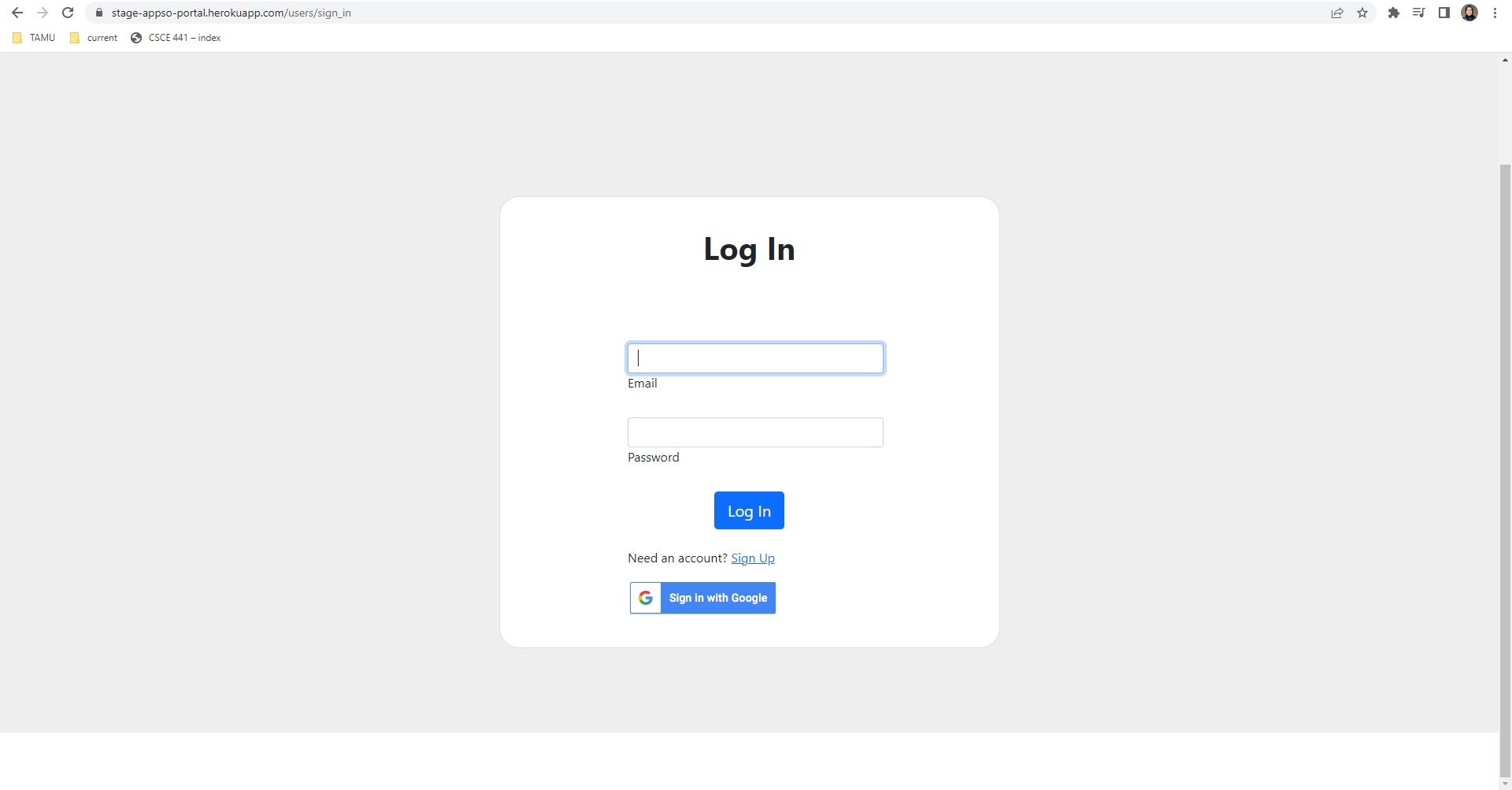
Member Home Page:



Admin Home Page:



Login Page:



Sign Up Page:

