# KANM Show Scheduling - Final Report

**Software Engineering CSCE 606** **September-December 2024**

**Teammates** – Ali Nablan, Ankit Mohanty, Davis Beilue, James Nojek, Kriti Sarker, Neeraj Julian Joseph Rajkumar, Toan Vu, Haridher Pandiyan

**Pivotal** - https://www.pivotaltracker.com/n/projects/2721031

**GitHub** - https://github.com/amohanty03/KANM-Show-Scheduling

**Deployed App** - https://kanm-show-scheduler-b962465e9890.herokuapp.com/

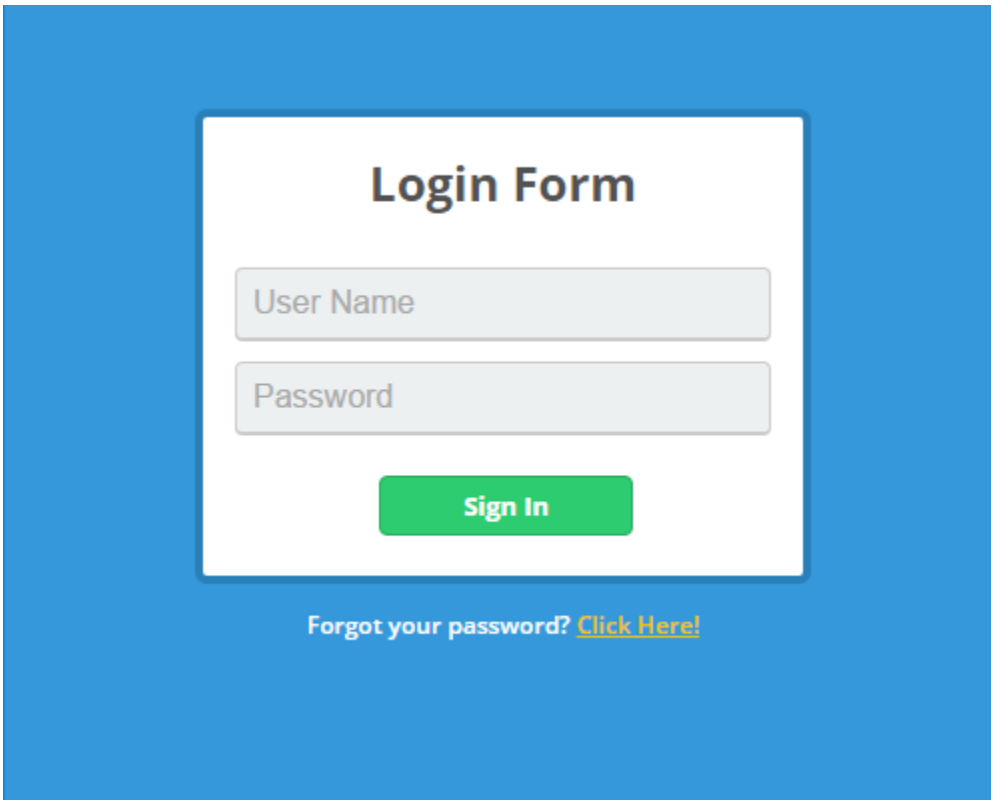
## Project Summary

Our customer, KANM, requested that our software team develop an application that allows them to automate the process of scheduling their semester of radio shows. The stakeholders for this application included the customer which was the KANM team, ourselves the software team, and a subset of ourselves being the various Product Owners and Scrum Masters that served in rotation every sprint. The application we are delivering meets not only the specified primary need, but provides other capabilities as well. In summary, our application allows for xlsx files to be uploaded to the server and then used to generate an easily readable schedule of radio shows with no show overlaps via an algorithm that factors in our customer's desires. The algorithm factors in a great deal of user data gathered from a Google Form, including their number of years in the semester, the best time/day for their show, and alternative times they would be available.

Our application first takes the user to a login screen where they will need to use OAuth to log into the application using their Texas A&M University email address via two-factor authentication. There is a hidden step 0 where users must be approved to authenticate by being added to an admin-controlled database. Once authenticated, the user has many options regarding the aforementioned files: upload a new file, download a previous file, select a file, and generate a schedule from the selected file. Once a schedule is generated, the user is taken to a new schedule view page where a single day is shown at a time; the user can switch the display between days via a dropdown menu. The user at this point also has the option to export the generated schedule to an xlsx file that will be downloaded to their machine. Final features include the user logout ability, as well as special permissions for those users listed as "admin," namely the ability to add/subtract users to/from the valid authentication database.

# User Stories

User Story: Admin Login
Point: 3
Status: Refactored
Description: As an admin, so that I can manage the RJ schedule, I want to securely log into the system with the correct credentials, if I forget my password I can reset my password with some security questions, I should land at the RJ schedule page after successful login.

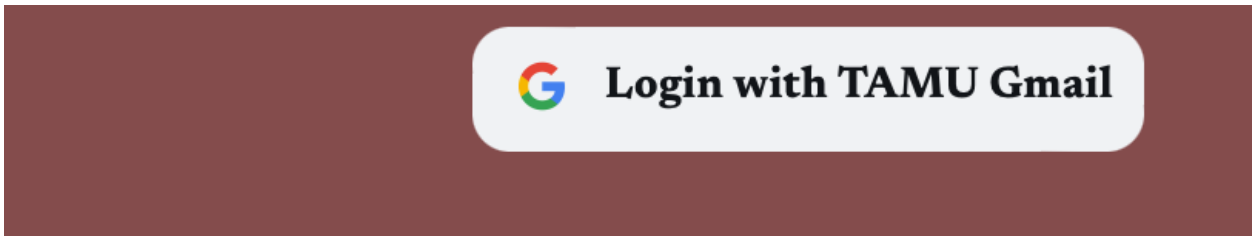



This user story was the original plan for user authentication as we should save user credentials in the database. However we thought that basic authentication might not be the best option for secure connectivity, so we decided to implement OAuth authentication instead.

User Story: Login Page
Point: 1
Status: Completed
Description: As a user, so that when I go to the given URL, I want to see the login page

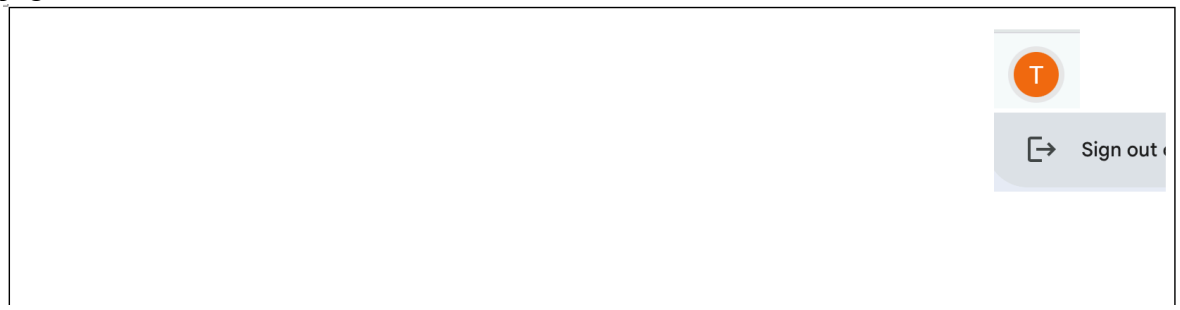
User Story: Login Button
Point: 1
Status: Completed
Description: As a user, so that when I go to the login page, I want to see the “Login with TAMU Gmail” button

User Story: Login Success
Point: 1
Status: Completed
Description: As an admin, so that when I login with a valid @tamu .edu email, and my email is saved in the database as an admin, I will be able to access the welcome page.



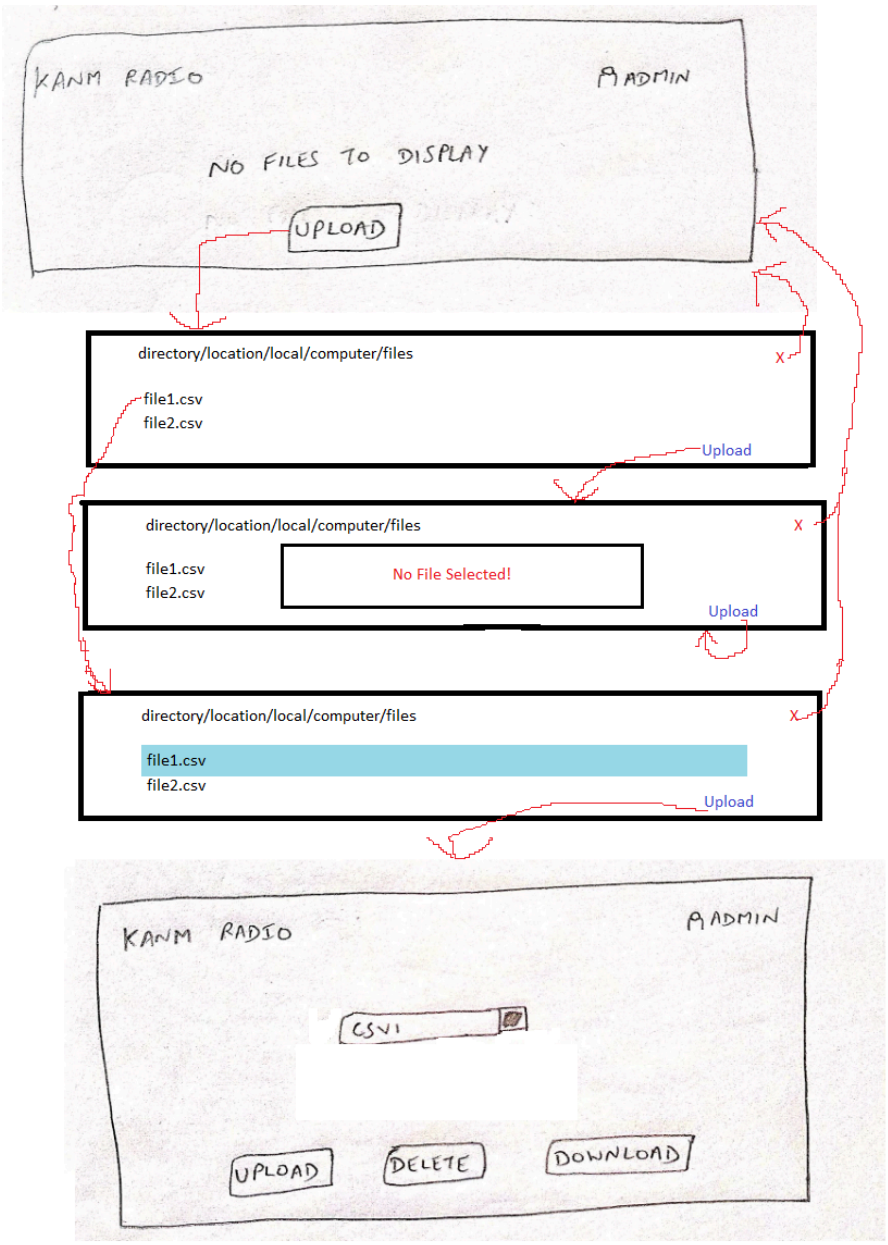

User Story: Admin Logout
Point: 1
Status: Completed
Description: As an admin, so that I can securely end my session, when I click the "Logout" button, I want to be redirected to the login page with no active session.

User Story: Logout Button
Point: 1
Status: Completed
Description: As an admin, when I am logged in, I want to see a logout button. The logout button can be found under the user icon on the upper right corner for each page, except for the login page.

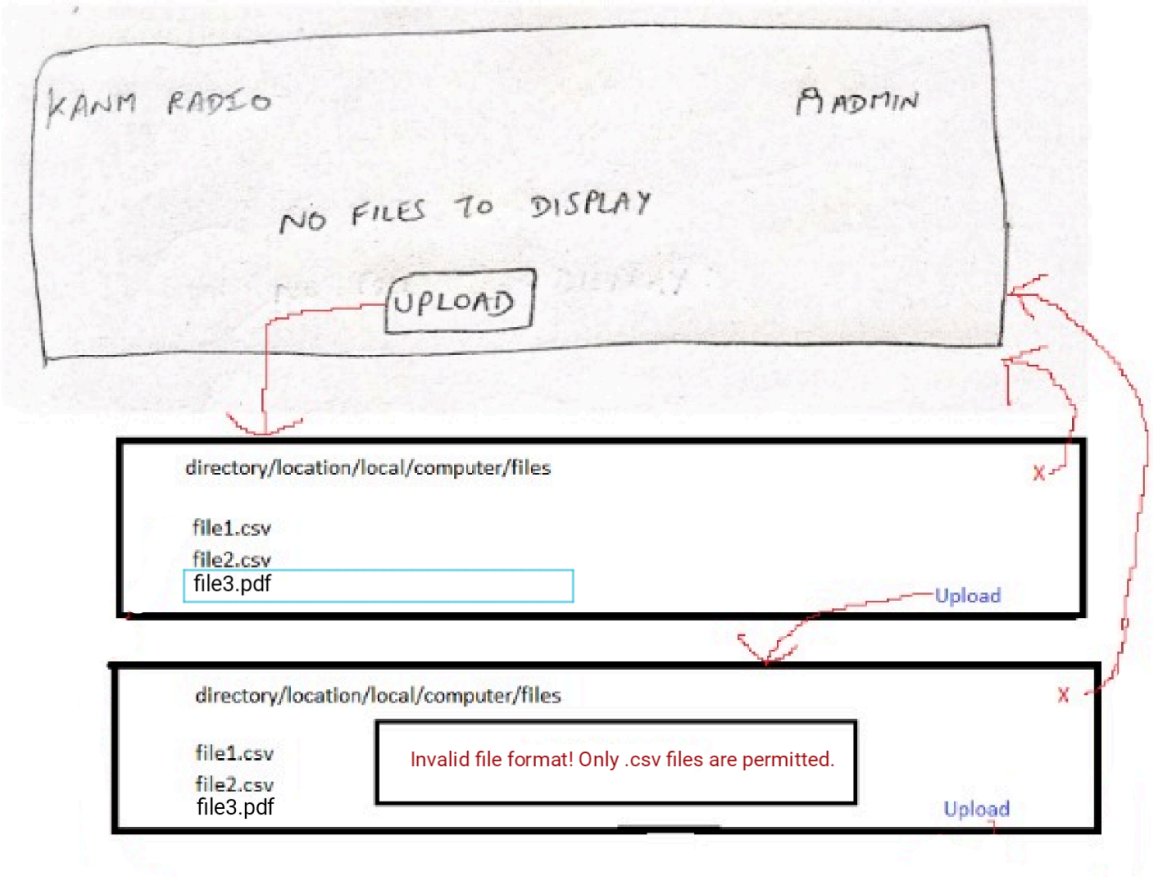



User Story:  Upload CSV Files
Point: 2
Status: Completed
Description: As an admin, so that I can access the data from the RJ Data CSV file, I want to be able to upload the RJ Data CSV file by clicking on the upload button

User Story:  Upload CSV Button
Point: 1
Status: Completed
Description: As an admin, when I go to upload the user schedule CSV file, I want to see an "Upload" button.

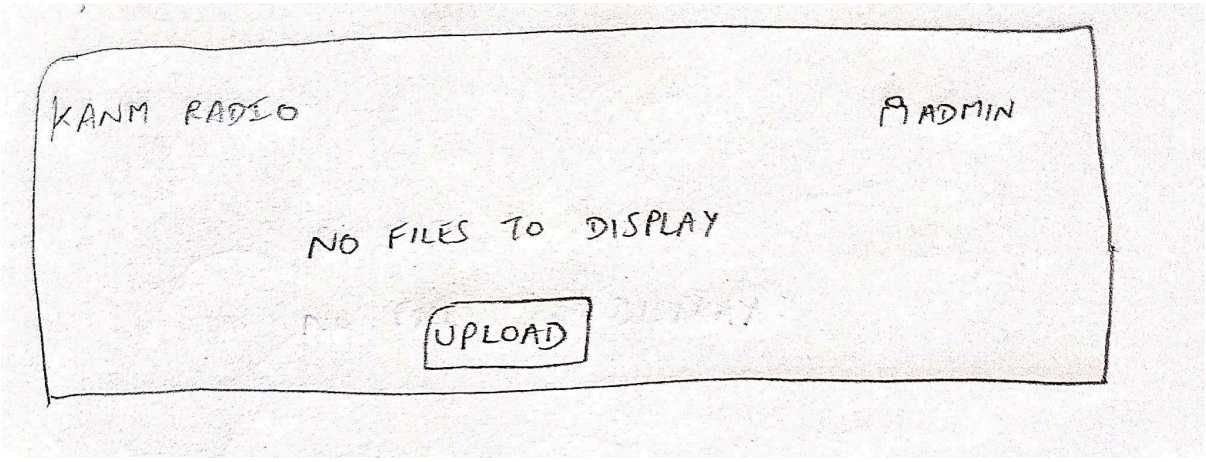



User Story:  Validate CSV File
Point: 1
Status: Completed
Description: As an admin, so that I can ensure data integrity, I want to only be able to upload files with the .csv extension.

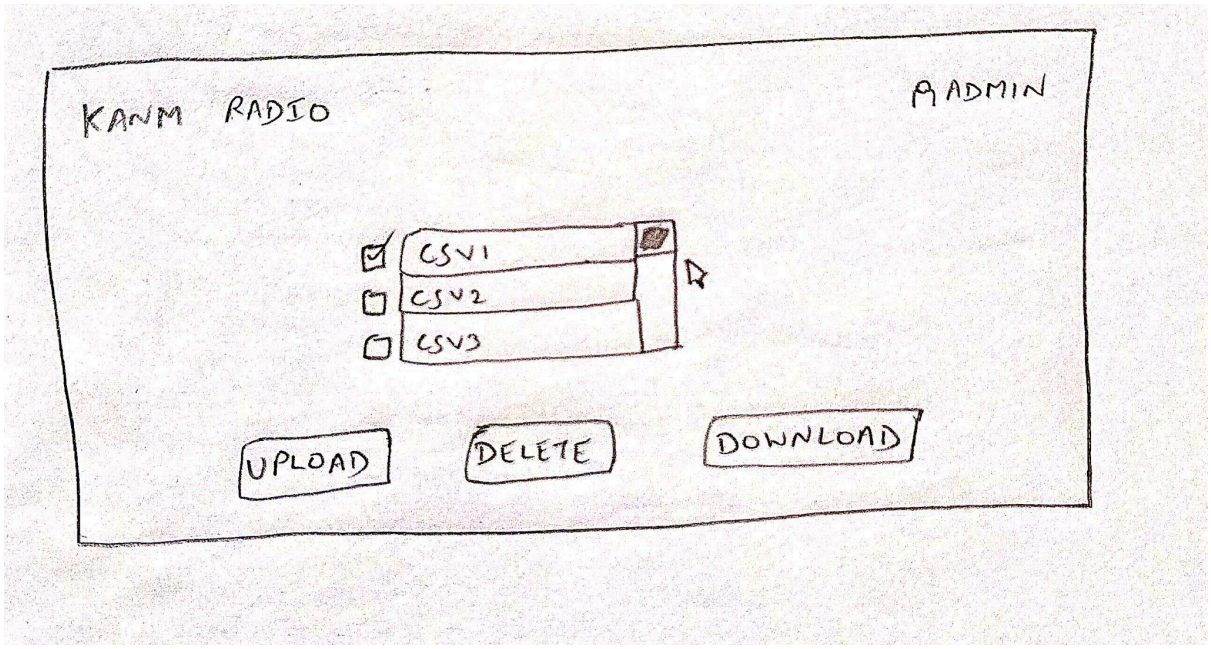



User Story:  View Multiple CSV Files
Point: 3
Status: Completed
Description: As an admin, so that I should be able to view the various CSV files uploaded to the system

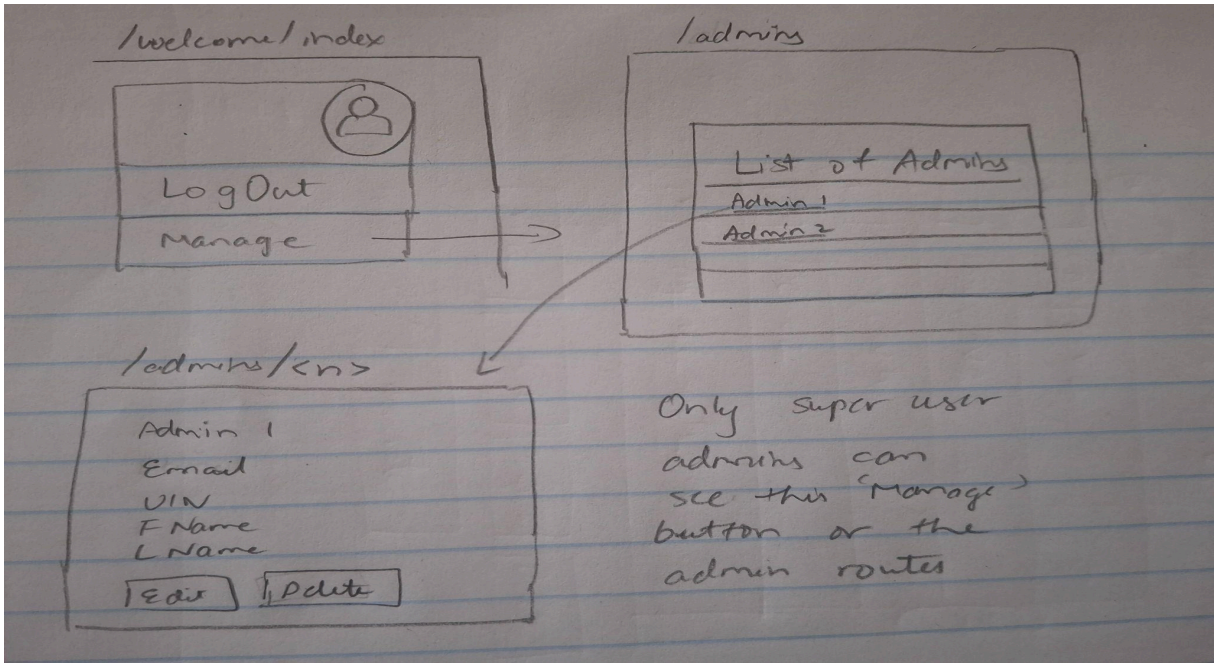
**Scenario 1 - No files have been uploaded to the system**



**Scenario 2 - The page displays the various files uploaded to the system along with several possible operations the user can perform on them.**



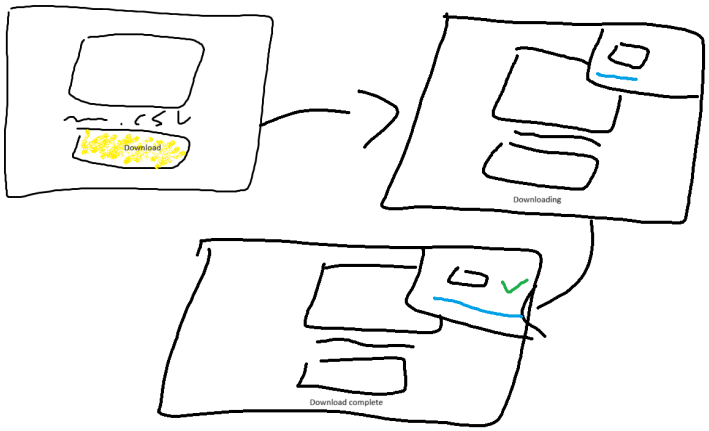

User Story: Super Admin Role
Point: 3
Status: Completed
Description: As a developer, so that I can ensure secure and controlled management of admin users, I want a super admin role with the ability to add, edit, and delete other admin users.



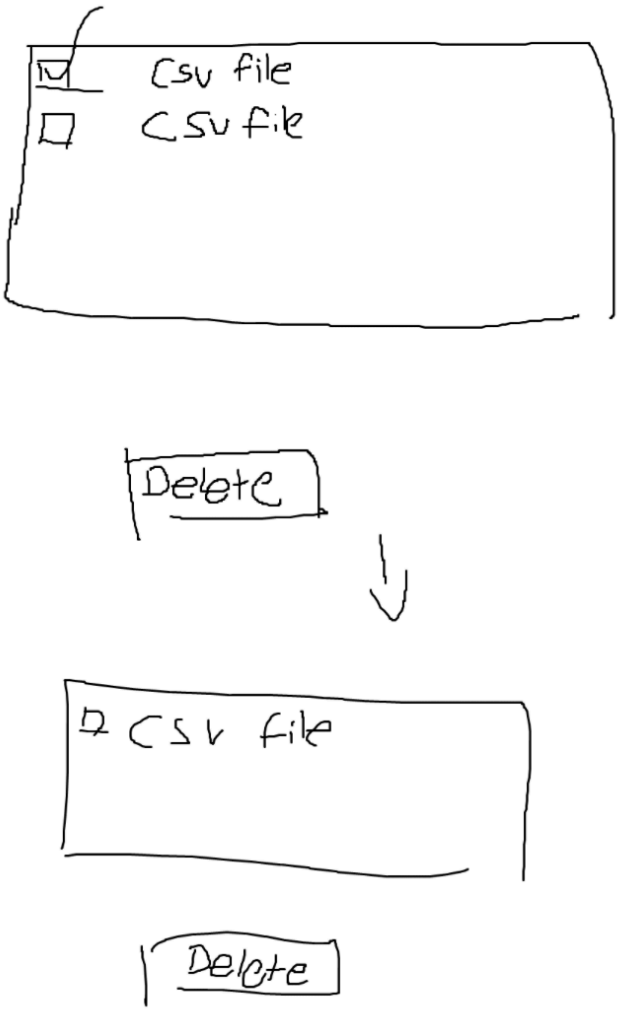
User Story:  Update UI for Admin Routes
Point: 1
Status: Completed
Description: As a developer, so that I can enhance the user interface and improve user experience, I want to update the UI with HTML and CSS styles for the admin page of our application

User Story:  Download CSV Button
Point: 1
Status: Completed
Description: As an admin, when I go to download a previous CSV schedule file, I want to see a "Download" button.


User Story:  Download CSV Files
Point: 2
Status: Completed
Description: As an admin, so that I can archive previous CSV schedules, I want to be able to download a CSV when I click the “Download” button.



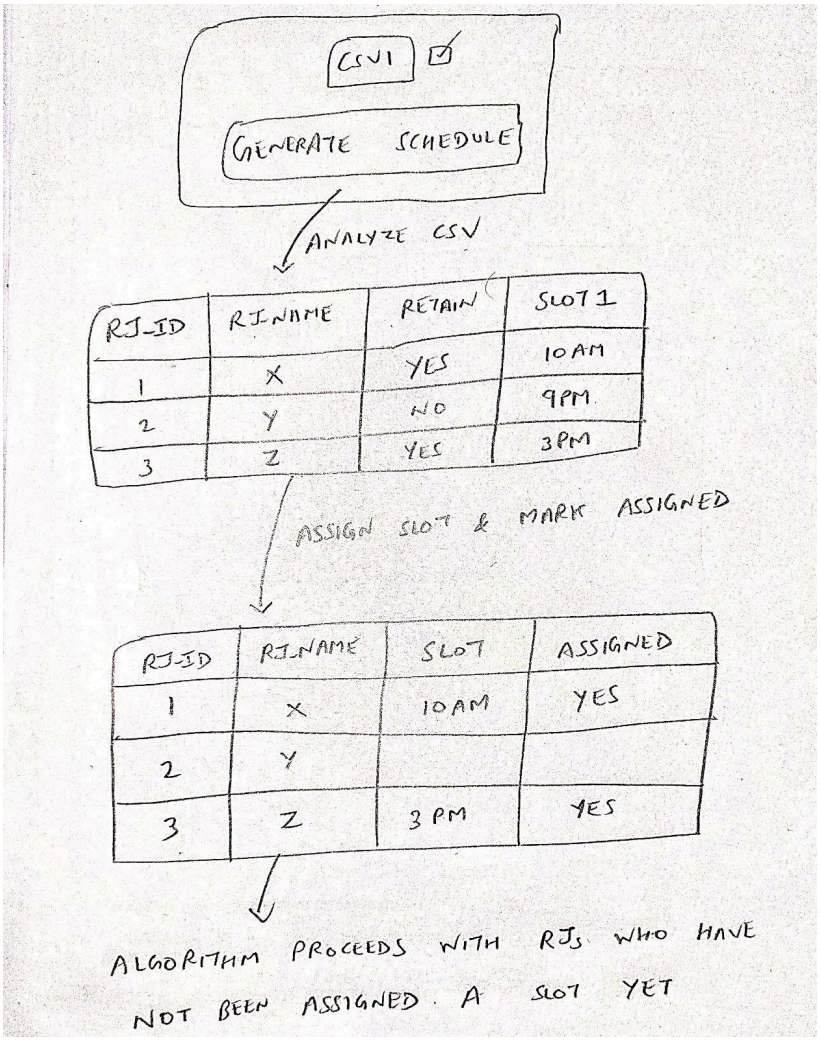
User Story:  Remove Uploaded CSV files
Point: 2
Status: Completed
Description: As an admin, when I want to upload a different CSV, I want to be able to delete a selected CSV file when I click the "Delete" button




User Story:  Decide Slot Retention
Point: 2
Status: Completed
Description: As a developer, when the admin clicks on generate schedule, I should be able to decide if the RJs would like to retain their existing slot based on the available input.

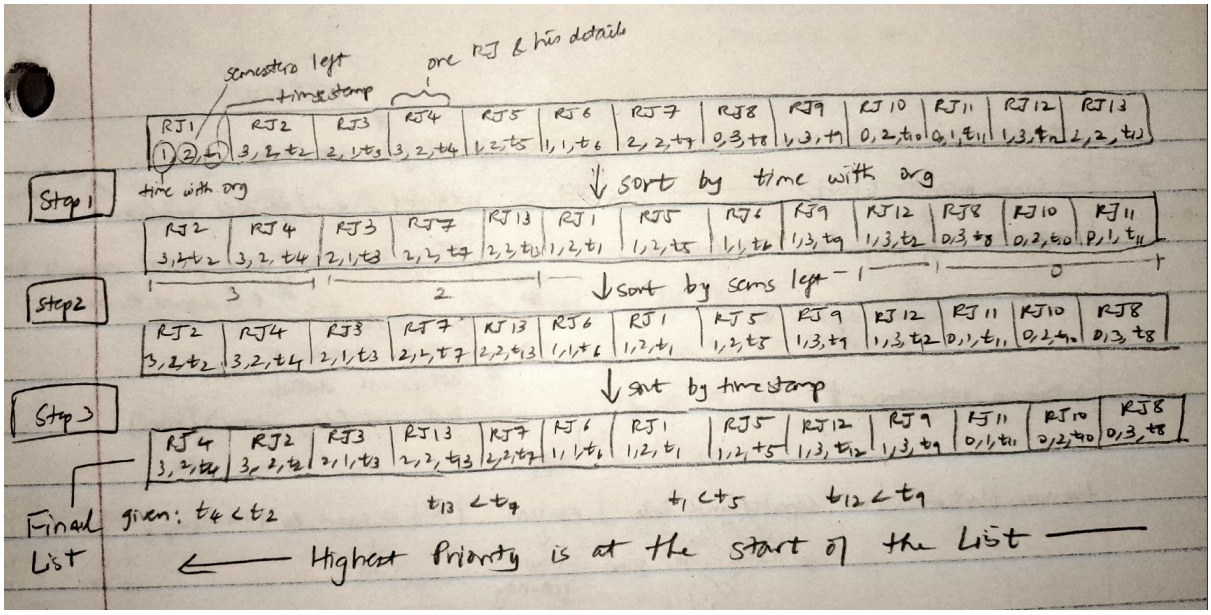
User Story:  Perform Slot Retention
Point: 2
Status: Completed
Description: As a developer, when an RJ has decided to retain their slot, I should assign the

retained schedule to the respective RJs and mark them as assigned.

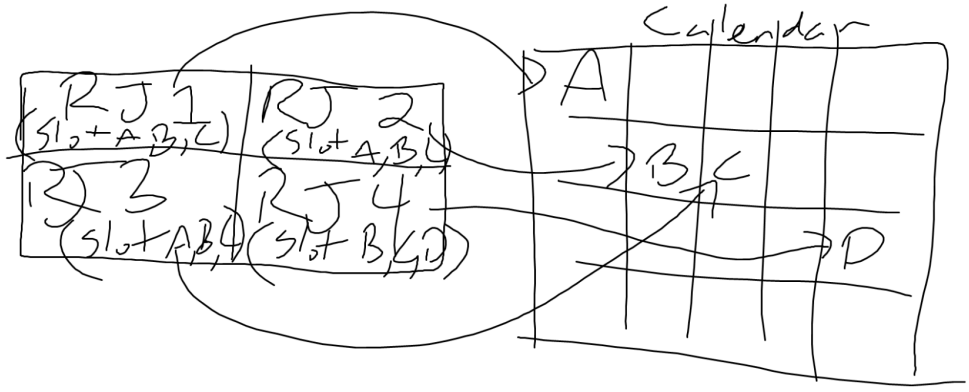



User Story: Parse the CSV data
Point: 4
Status: Completed
Description: As an admin, when I am logged in, so that I can begin the schedule generating process, I want to parse the selected CSV into usable data.

User Story: Rank the Radio Jockeys
Point: 2
Status: Completed
Description: As a developer, so that I can generate a schedule prioritizing specific classifications of radio jockeys, I want to rank the list of radio jockeys by time in the organization, semesters until graduation, and time of application.

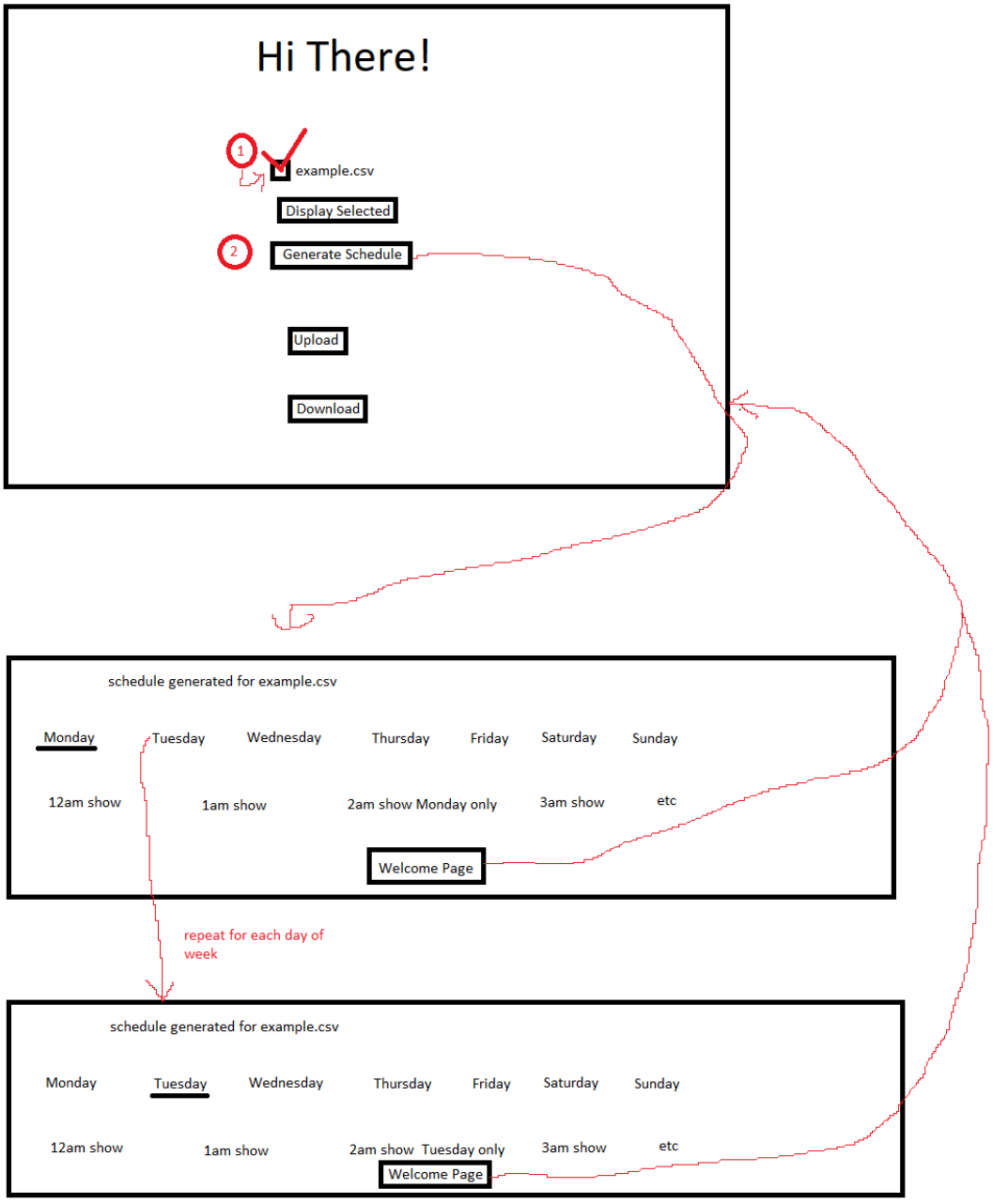



User Story: Schedule the Radio Shows
Point: 6
Status: Completed
Description: As the developer, so that I can generate a schedule prioritizing specific classifications of radio jockeys, I want to iterate through the ranked list of radio jockeys, and assign them in order to open time slots based on their preferred and alternate times.



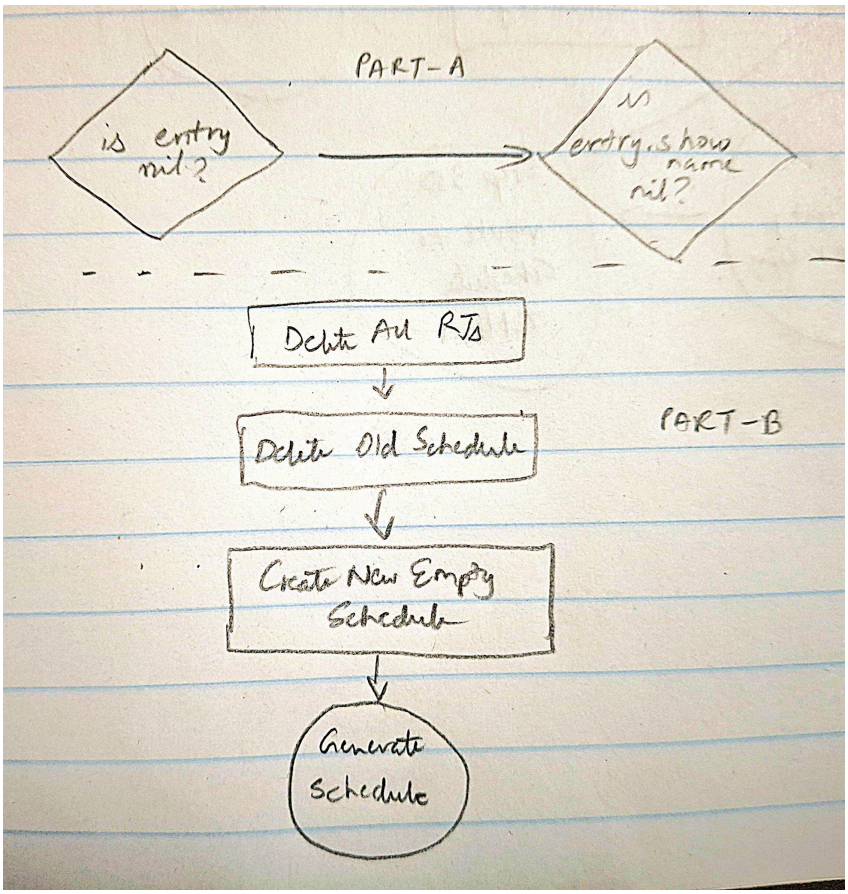

User Story: Generate Schedule Button
Point: 1
Status: Completed
Description: As an admin, when I am logged in, I want to see a “Generate Schedule” button. The generate schedule button can be found under the “Display Selected” button, on the welcome page. (redirect to schedule page)

User Story: Display Schedule
Point: 1
Status: Completed
Description: As an admin, when I am logged in, so that I can view a generated schedule, when I click the "Generate Schedule" button, I want to be redirected to a page showing the schedule generated from the selected CSV in calendar format.

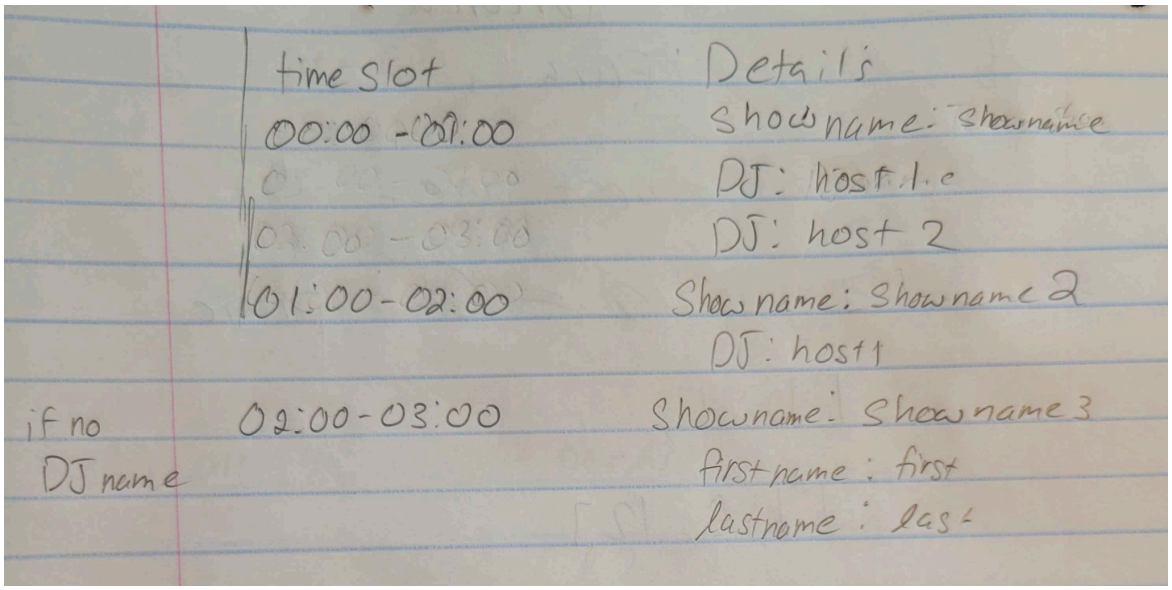


User Story: Refine Scheduling Algorithm
Point: 7
Status: Completed
Description: As an admin,I want the schedule generation algorithm to check for more open time slots in a user's alternative list so that they have a higher chance of getting scheduled.

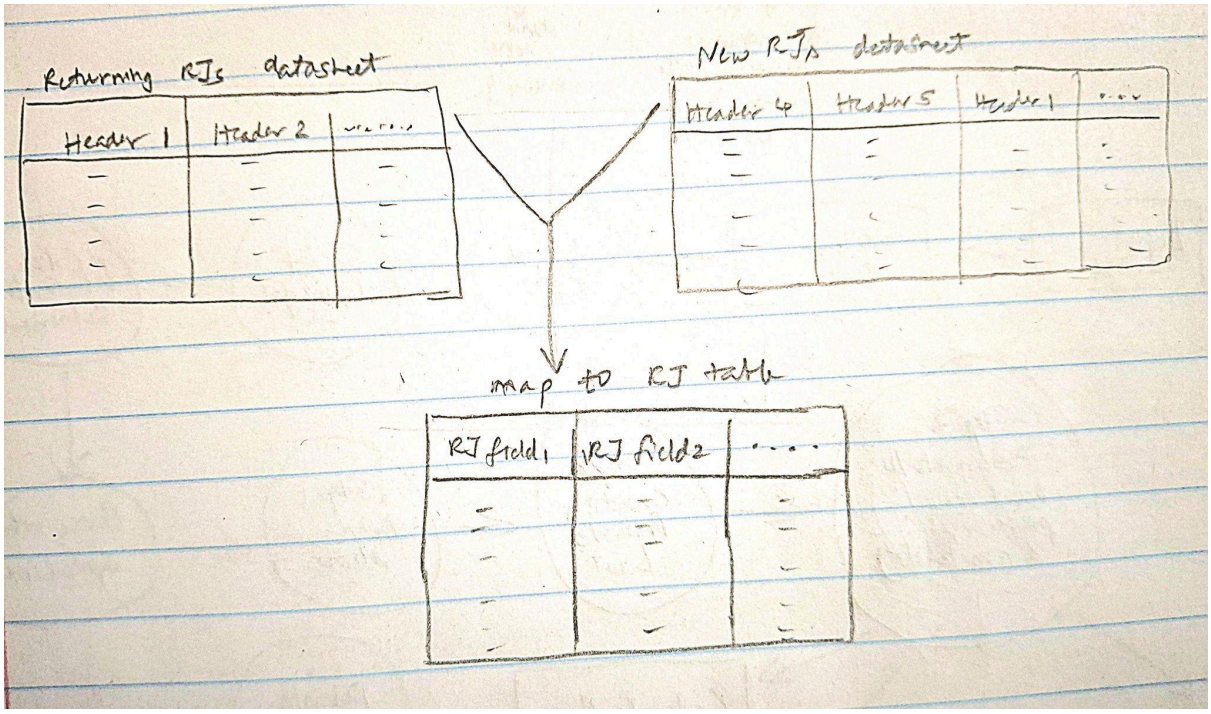
User Story: Display the Assigned RJs
Point: 1
Status: Completed
Description: As an admin, when I generate a schedule, then the RJs aren't retaining their former timeslots, should appear in the calendar.



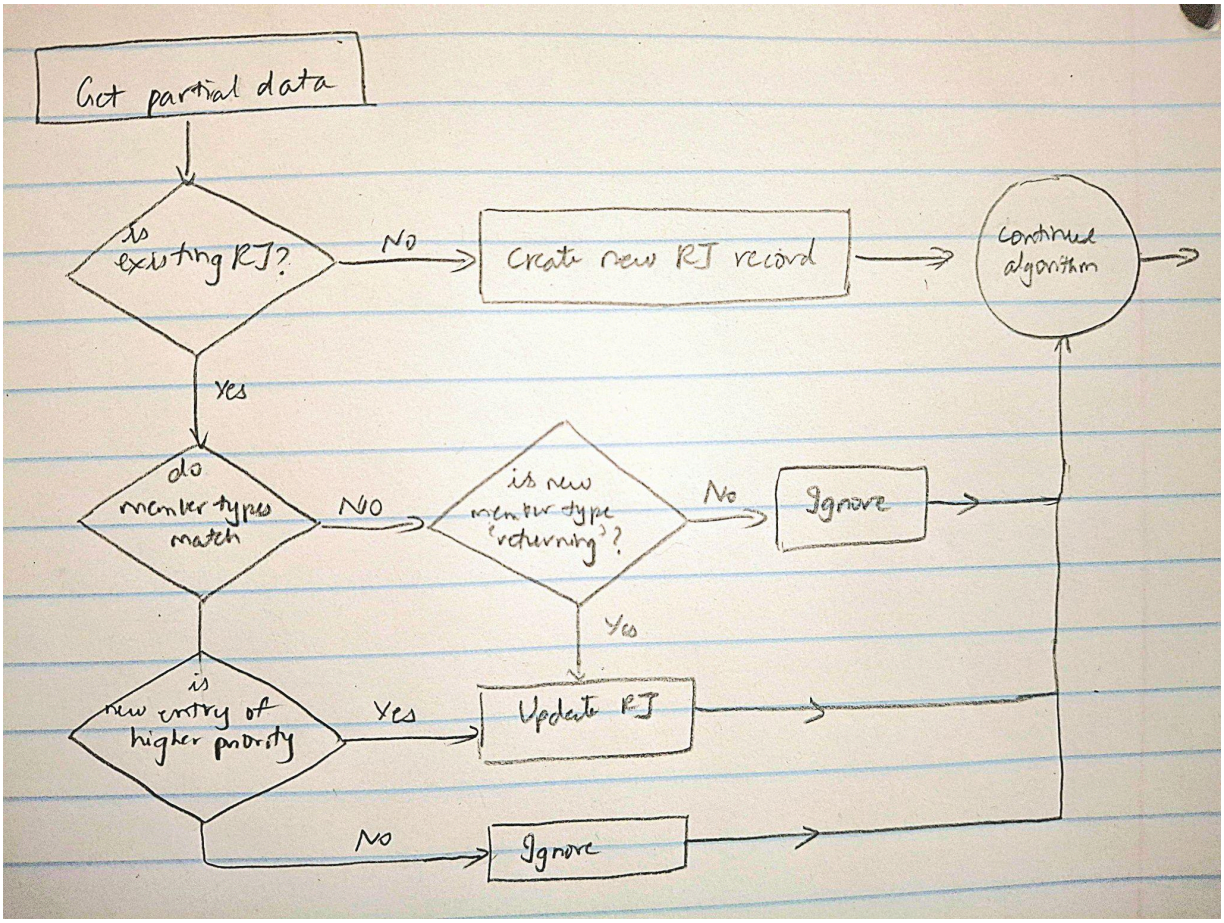

User Story: Add DJ Names for a single show name
Point: 2
Status: Completed
Description: As an admin, when I generate the schedule, for each entry, the DJ(s) hosting the show should be visible




User Story: Map the Form Fields to the Table
Point: 1
Status: Completed
Description: As an admin, I should have the output spreadsheet map to the fields I want for my table, so that it doesn't rely on the column ordering.



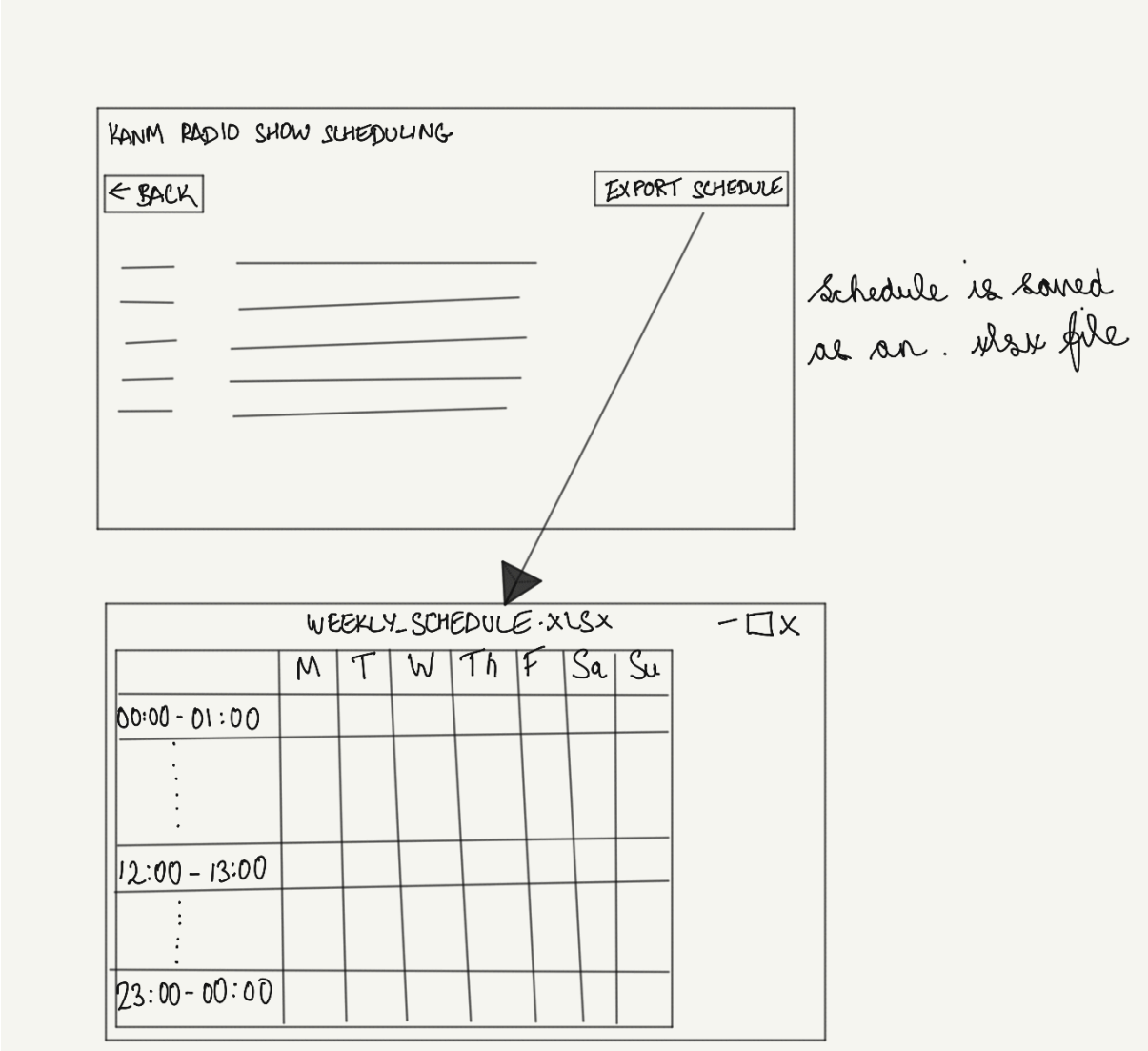
User Story: Handle Co-Host Entries
Point: 1
Status: Completed
Description: As an admin, when I generate the schedule, if there are any multiple RJs for a single show, then I will only keep the RJ with the higher priority in the generation list and discard the other.




User Story: Export Schedule Button
Point: 1
Status: Completed
Description: As an admin, when I am logged in, I want to see a “Export Schedule” button after I click on the “Generate Schedule” button.

User Story: Export Schedule
Point: 2
Status: Completed
Description: As an admin, when I am logged in, so that I can export a generated schedule, when I click the "Export Schedule" button, I want the generated schedule to be downloaded in .xlsx

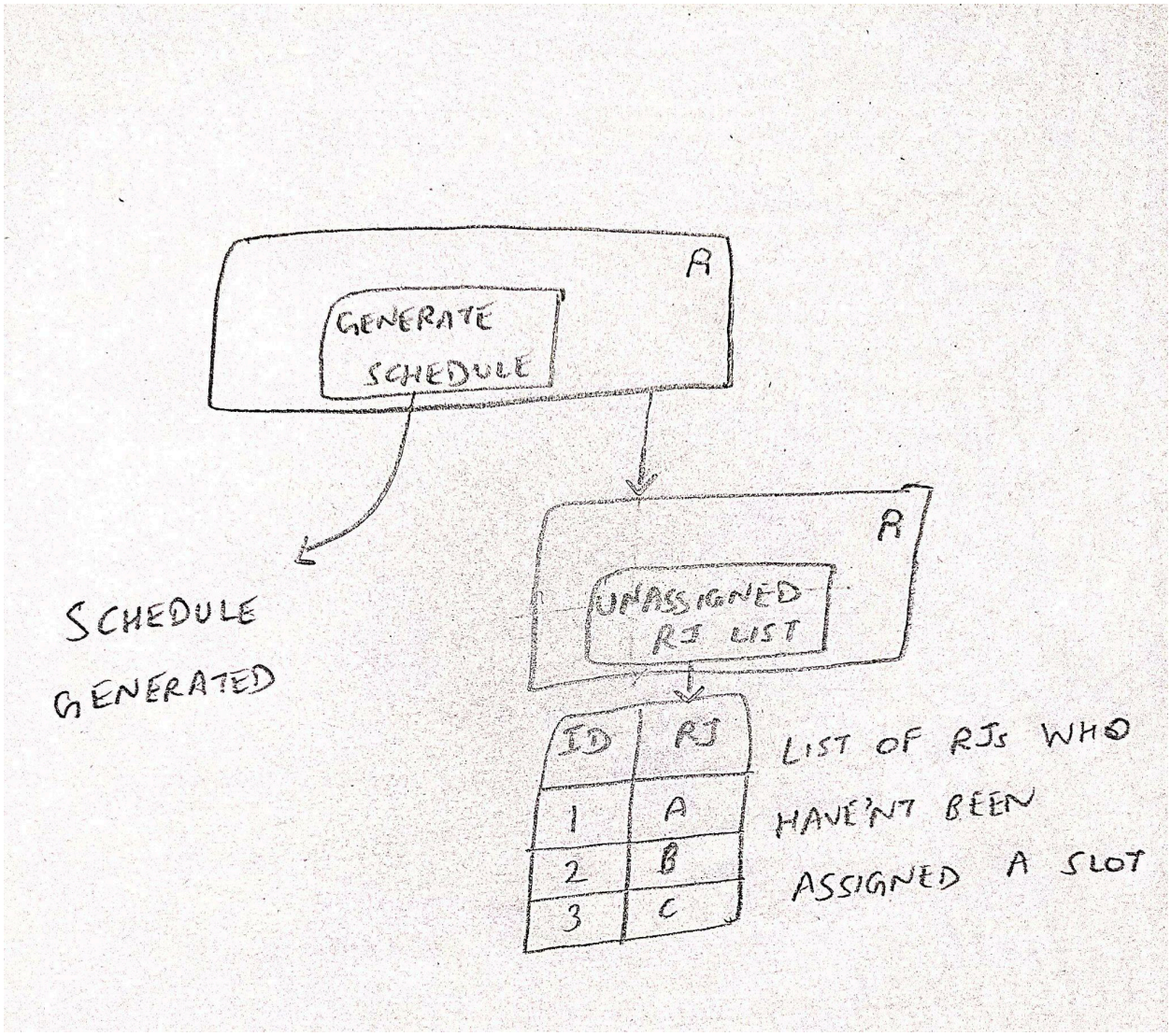
format.



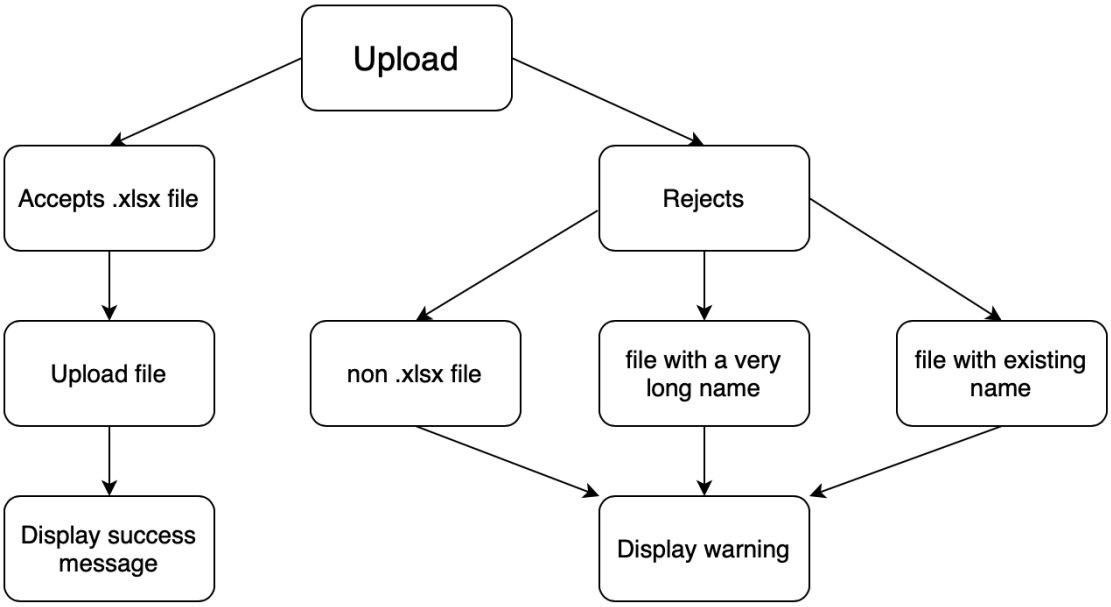

User Story: Final Schedule View Refinement
Point: 1
Status: Completed
Description: As an admin, when I generate the final schedule, the empty time slots in the schedule should no longer be visible

User Story: CSS Adjustments
Point: 2
Status: Completed
Description: As an admin, the UI should be aesthetically pleasing.

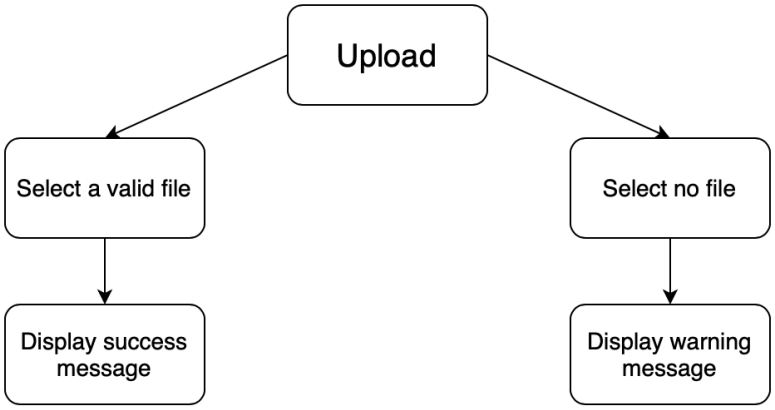
User Story: Display Unassigned RJs List
Point: 3
Status: Completed
Description: As an admin, when I click on generate schedule, I want to be able to view the list of all RJs who didn't get assigned a slot, besides the generated schedule.



User Story: Enhanced XLSX Validation
Point: 2
Status: Completed
Description: As an admin, so that I can ensure data integrity, I want to only be able to upload files with the .xlsx extension, and the file name cannot match what is already in the application, and the file name cannot be more than 60 characters long.




User Story: Enhanced Upload XLSX Controller
Point: 1
Status: Completed
Description: As an admin, so that I can upload a valid .xlsx file when the user selects a file, or display a warning message otherwise.



# Role Assignment

Over the course of the project, all 8 team members alternated through the roles of scrum master and product owner. See the table below for the week by week assignments.

| Sprint | Product Owner | Scrum Master |
|---|---|---|
| 1 | Neeraj Rajkumar | Ankit Mohanty |
| 2 | Toan Vu | Davis Beilue |
| 3 | Haridher Pandiyan | James Nojek |
| 4 | Ali Nablan | Kriti Sarker |

# Point Breakdown

In the following table, we break down the number of points each team member completed each sprint. The number of stories that contributed to that point total is denoted in the adjacent parenthesis. Note that some stories were split across multiple team members, meaning the point value given to the story in Pivotal Tracker was split across all members working that story when we performed our calculations.

| Team Member | Sprint1 | Sprint2 | Sprint3 | Sprint4 | Total |
|---|---|---|---|---|---|
| Ali | 1 point (1 story) | 2 (1) | 3 (1) | 3 (2) | 9 (5) |
| Ankit | 2 (3) | 1.5 (2) | 2.5 (2) | 3 (2) | 9 (9) |
| Davis | 0.5 (1) | 2 (1) | 3 (1) | 3.5 (1) | 9 (4) |
| Haridher | 2 (2) | 3 (1) | 1 (1) | 3 (3) | 9 (7) |
| James | 1 (1) | 2.5 (2) | 4.5 (3) | 1 (1) | 9 (7) |
| Kriti | 0.5 (1) | 2 (2) | 2 (1) | 4.5 (2) | 9 (6) |
| Neeraj | 1 (1) | 3 (1) | 2 (1) | 3 (1) | 9 (4) |
| Toan | 3 (4) | 2 (2) | 1 (1) | 3 (2) | 9 (9) |
| | | | | | |
| Total | 11 (13) | 18 (12) | 19 (11) | 24 (13) | 72 (50) |

# Sprint Summaries

The following is a bulleted summary of what was accomplished in each sprint:

## Sprint 1

- Initial project setup
    - Organizing Pivotal Tracker, configuring GitHub repo, deploying app to Heroku
- Login feature with Google OAuth

## Sprint 2

- Spreadsheet file management
    - Adding ability to upload, download, and delete files
- Added Super-User to allow for admin user management

## Sprint 3

- Implementing initial scheduling algorithm
    - Required design meetings and discussions
- Implementing initial calendar view

## Sprint 4

- Improvements to current features
    - Calendar view, scheduling algorithm
- Resolving pending bugs found in development/testing

# Customer Meetings Minutes

- 9/19/24
    - This was the first meeting between the team and the client. After making general introductions, a general high level overview of the project was discussed. The client requested a tool that can automate the scheduling process for the student radio organization.
- 9/26/24
    - The client provided a sample of the CSV file that they use as input for the manual scheduling process. The client requested that a list of radio jockeys who were unable to be scheduled be outputted, as well as the schedule.
- 10/3/24
    - The team demonstrated the results of the first sprint, which was the skeleton layout of the website and the login process. Students with a TAMU email can log into the website, and only students that are specifically given admin rights can do any scheduling actions.
- 10/10/24
    - The team proposed changes to the Google form that the client currently uses to get data from the radio jockeys. Changing the timeslot selection question from an open ended text box to a grid based system (i.e. the rows are days of the week, and the columns are hours of each day) guarantees conformity between user inputs. This greatly simplifies the parsing process and reduces the client's manual workload.
- 10/17/24
    - The results of sprint 2 were presented to the client. The team showed the upload, download, and logout functions, which all pleased the client. The client also shared that if a radio jockey is returning to the same time slot as last semester, then they should be the number one priority for that time slot in the upcoming semester.
- 10/24/24
    - The team had a meeting with the client's web developer, rather than the normal team. The integration between the team's tool and the existing website was discussed. The client's website currently uses an API get call to pull internal data into their schedule display, so a similar methodology will likely be used to show the results of the team's scheduling generation on the official client website.
- 10/31/24
    - The team had a brief meeting with the client due to Halloween. The results of sprint 3 were presented, which showed the first iteration of parsing the data input from an uploaded CSV file and generating a schedule. The client was pleased with the progress of the team and agreed to some more minor Google form modifications to further aid parsing.

- 11/7/24
    - The team demonstrated the export schedule button as well as the format of the generated schedule to the client. The client made a few suggestions to implement in the future. These included adding the DJ name to the schedule view and some tweaks to the format of the spreadsheet generated by the export schedule button.
- 11/14/24
    - The team demonstrated the results of sprint 4 to the client. The client saw that their requests from the previous meeting were met, as well as the generation of a list of radio jockeys that were unable to be assigned a time slot by the scheduling algorithm. The client is very pleased with the results and the lack of manual work required by this tool compared to their old process.
- 11/21/24
    - This was the last meeting between the team and the client. The client expressed again how much more efficient the new tool is compared to their previous process. The team and the client both stated that they enjoyed working with each other.

# BDD/TDD process

After speaking with the customer and discussing features they wanted to include into the program and discussed amongst ourselves to determine how the feature was going to work. Then each feature was formatted into sentences like “As an admin, when I go to upload the user schedule CSV file, I want to see an "Upload" button.” and creating storyboards which both helped in understanding and visualizing the feature. We then tested for each feature and implemented them using cucumber and Rrspec. Using these tests helped us code specific to how specific to how each feature should be implemented as well as make the goal clear on what the code should output.

# Config Management

Our team used GitHub for version control. Every user created a new branch for each story they worked on and opened a Pull Request to the main branch when their code was complete. That Pull Request would then be reviewed by the rest of the team. PRs would receive comments if code was incorrect, conflicting, or if Rubocop had not been run (which our GitHub was configured to give warnings about). When a story’s branch was accepted to the main branch, our Heroku app was set up to auto-deploy with the new code.

We conducted a sort of spike during a team meeting in Sprint 2, the sprint before we first wrote our scheduling algorithm. Before this meeting, we had each been tasked with thinking about various ways to write this algorithm in an efficient and effective manner. We each came to the meeting with ideas, and in the end decided to break the algorithm itself into three main

components: giving back time slots to returning RJs who wished to stay in place, creating a sorted list of all remaining RJs, and then scheduling those remaining RJs in priority order.

Our feature development was somewhat compartmentalized, which typically led to us delivering small amounts of full features during sprint MVPs. In general, Sprint 1 was focused on login, Sprint 2 on file handling, Sprint 3 on schedule generation and visualization, and Sprint 4 on handling unassigned RJs and making improvements to the scheduling algorithm.

It took us a sprint to figure out how to best organize our Pivotal Tracker board, but eventually we created releases for each sprint and were able to correctly tag each story to represent what release it was a part of. In the end, our GitHub states that we have 50 total branches (this is counting our main branch and 2 dependabot branches, however), and Pivotal Tracker correctly shows 4 releases (1.1-1.4).

# Heroku Deployment

Deploying the tool to Heroku was a relatively smooth process, but the team did encounter some issues along the way. Heroku was set up so that a new push to the main branch of the Git repo would drive a new build of the deployed app. This caused some authentication issues that were resolved by simply not uploading the credentials file to the Git repo. A locally generated credentials file was sometimes uploaded to Git, overwriting the actual credentials file that was synced up with the Heroku deployment.

The other main issue that the team faced was with database migrations in Heroku not being performed as expected. After some major feature updates to the Git repo that involved database migration, the deployed Heroku app did not receive the appropriate migrations. This caused errors and critical process failures in the app. In addition to performing the database migrations while writing the code, the team also had to log into the Heroku CLI and perform the database migrations there. Finally, a new commit had to be pushed to the main branch to ensure that these Heroku migrations were actually pushed to the repo.

# Tools and Gems

## Project Management Tools

Github: Allowed us all to work on the code together without interfering with each other’s work. Version control also helps us in debugging conflicts.

CodeClimate: Helped us keep the code concise and helped us look for any issues with the code like duplicates and code smells

Pivotal Tracker: used to keep track of the sprint progress and user story points

Google Meet and Slack: Main communication channels to the client and among team members

## Main Ruby Gems

Sqlite3 and PG: Used for storing information about the DJ's and their information like name, dj name, preferred times, and alternate times in local development and deployment, respectively

Omniauth, Omniauth-google-oauth2, and Omniauth-rails_csrf_protection: APIs for Oauth Authentication using Google as the provider

Rubyzip: used to process .zip files when downloading from the app

Axlsx: used to process .xlsx files when uploading, processing data, and creating the schedule

Overall, the project management tools allow us to perform as a single unit, where collaboration and communication were the key to success. We effectively used the weekly meeting with the client to demonstrate our progress, and discuss any potential features and enhancements. We were also able to perform pair programming as two team members collaborate to work on a shared user story. We were able to stay on track and meet our delivery requirements for every sprint.

Using third party dependencies helped us save time in development work. We initially ran into issues with the Google APIs when deploying the application to Heroku. We then realized that the credential file and the master key are a unique combination, whenever a team member pushed their local credential file, the master key set as an environment variable in Heroku needed to be updated as well. The team decided to add the credential file to .gitignore and use only one initial credential setup when deploying to Heroku.

# Repository Content and Project Setup

The project is set up with a user-friendly approach that even developers with no to very little experience should be able to set and start up the application by following the readme step by step. It is suggested to complete the Google Authentication Service with Google first before moving on to complete the local setup and Heroku deployment.

The code in our repository is arranged in the typical file structure of a normal Ruby on Rails application, following the Model-View-Controller (MVC) software design pattern.The repository contains all our source code, test code, and documentation from throughout the semester including sprint plannings and retrospectives.

## Google Authentication

The Google Authentication Service allows the project manager to configure how access should be provided to users. The service provides options to authenticate internal users, i.e. users with a valid @tamu.edu email address, as well as external users, who own a valid Google email.

During the configuration, the client_id and client_secret will be provided for setting up the authentication client in the application as well as setting the reply URL(s), which Google will redirect users to if authenticated.

## Local Setup

The local setup requires the authentication service to be complete by adding the client_id and client_secret to the credentials.yml.enc file. It is important to note that this step will create a unique pair of files, credentials.yml.enc and master.key, that the key will be used to decrypt the secret at run time. After completing the authentication service client in the application, a @tamu.edu user can be added to the db/seeds.rd so that this user can be authenticated. it is finally time to complete the local setup by running the following commands:

- "bundle install" to install the required dependencies listed in the Gems file
- "Rails db:migrate" to create the local database using sqlite
- "Rails db:seed" to populate the database with users/admins information
- "Rails server" to start the web application locally

## Heroku Deployment

A webapp and a postgres database can be set up in Heroku by following Heroku documentation. Deploying the web application to Heroku is very similar to the local, where the project manager needs to configure the authentication service and install the required dependencies.

To get started, the following commands can be run with a local terminal to push the application to Heroku:

- "heroku login" to login to Heroku
- "heroku git:remote -a <your-new-app-name> -r <heroku-new-remote-name>" to link Heroku to your local workspace. For example, "heroku git:remote -a kanm-pv -r heroku-pv". Note that "kamn-pv" is the application name in Heroku and "heroku-pv" is the local workspace, which will be used as examples for the rest of this document.
- "Git push heroku-pv main" to push to Heroku
- "heroku config:set RAILS_MASTER_KEY=`cat config/master.key`" to set the master.key as an environment variable in Heroku. Note that because of its uniqueness, there is one and only one master key that can be used to decrypt the credentials, so it is important to verify that these two components are synchronized. If there is any errors running the steps below, it is most likely that the key and the credentials are out of sync. The issue can be resolved by explicitly pushing the credentials.yml.enc file to Heroku and updating the Heroku master key with the same local master key.
- "Heroku run rails db:migrate" to create the local database using sqlite
- "Heroku run rails db:seed" to populate the database with users/admins information

To allow Heroku webapp to use the authentication service, the Heroku application URL needs to be added to the reply URL list as above in the Google Authentication step. After this step, the Heroku deployment is complete.

# Presentation and Demo

- YouTube Video (KANM Radio Show Scheduler Presentation): https://youtu.be/-1RBrWpdwfs