Russell Abraham, Michael Cummins,

Devin Franchino, Tyler Houser, Jaheim Oates

CS 375

Dr. Mongan

May 10, 2025

Final Report

Overview

The Student Engagement App (SEA) was developed to solve a common problem on college campuses: email overload. At Ursinus College, campus community members often miss important club events, campus announcements, or course-work related emails because they are buried somewhere in their inbox. SEA serves as a centralized platform for students to receive targeted event notifications and updates based on their interests, and for student organizations, offices, and clubs to communicate directly with the Ursinus community. By isolating the social information spread on the campus, SEA reduces email inbox clutter and increases student engagement.

Stakeholders

Our stakeholder groups are split into two demographics: those who wish to gain information about events on campus, and those who wish to spread information about events they are hosting through posts. The general student body and Ursinus Community is the stakeholder group for the population of users who receive the messages and can filter their feed for a personalized experience. These users can scroll through the app to receive the notifications and posts for events clubs and organizations they are interested in are hosting without having to sacrifice space within their email. The posters include the office of student engagement and presidents, organizers, and members of student clubs, societies, and activities. These users have a more direct and readily available method of spreading information to the campus community without having to type emails.

Requirements

We defined many requirements for our application which can be found within our requirements document (linked at https://michaelcummins1.github.io/StudentEngagementApp/), however a general outline of our requirements can be bifurcated into two categories: the front-end, user UI and the back-end, database management.

- **Front-End (User Interface):**
  - Pages for Login, Account Creation, Main Feed, Post Creation, Settings, and User Profile.
  - Input fields and buttons for navigation and interaction.
  - Dynamic feed showing followed content.
  - Light/Dark mode toggle.
- **Back-End (Server & Database):**
  - Database tables for accounts, posts, followers, and tags using better-sqlite3.
  - RESTful API endpoints for front to back end communication

    - JWT-based authentication for account security.
    - MVC design structure (Models, Views, Controllers)

Design

We used the Model View Controller (MVC) design architecture to create a clear separation of levels of functionality within our application. The view is made up of the front-end components, including HTML, CSS, and JavaScript, that present the UI and allow for user interaction. Careful consideration into the aesthetics of the pages was taken to ensure ease of use for the users. The controller acts as the 'middleman', handling the routing needed to connect the user inputs from the view to the appropriate operations found within the model via RESTful API HTTP requests and responses. The model is responsible for interfacing with the SQLite database, managing, sorting, and storing data such as user accounts, posts, and their relationships. This modular design enhances integrability and maintainability, creating an easily understandable framework for future development and implementations. We developed a full MVC stack for different key features of the application, including users, posts, and authentication processes.

Test Plan

We deployed both unit and user acceptance testing to ensure the functionality of the application. Throughout development of the application, we conducted manual user acceptance testing by interacting with the front-end of the application from the perspective of users. This process allowed us to verify that key requirements (including logging in, creating posts, and viewing the feed) functioned as we intended and met the expectations outlined in our requirements document. Taking the standpoint of end users also placed us in a position to evaluate the aesthetics, ease-of use, and understandability of the general structure and functionality of the front-end of our application. For the unit tests, we developed assert modules within JavaScript to confirm critical aspects of the application logic from a technical and developmental standpoint. Specific tests were created to check the veracity of the intended use of the functions, as well as to check whether or not edge cases, such as incorrect user input, were handled in a smooth, error-free manner.

Future Implementations

If given more time for further development or advising a new team of collaborators on what to begin with improving, we would list the following:

- Profile Hierarchy
  - Admin - an account with the ability to audit and/or delete other posting accounts/posts
  - Poster (Club / Org) - an account that is given permission to post by the admin account (request upon account creation)
  - Viewer - A typical member of the campus community simply using the app to learn about events on campus, is unable to access the create a post page
- User following functionality - All users have the capability to select poster accounts they wish to follow in order to filter their feed of posts using the following button (front-end and back-end frameworks have been implemented, simply connecting them with the MVC architecture is required)

- “Following” Feed - Once user following is created, plugging in the values to the script associated with the main page should easily implement the following feed
- User Profile Modification - Users should be given the option to edit their email, password, or name as well as, if they are a posting account, view the posts they have created and select which, if any, to delete
- iOS and Android Ports - iOS and Android shells should be developed to allow for the application to function on mobile devices
- Push Notifications - The enable notifications button in the settings page should toggle iOS and Android notifications on/off depending upon which mobile device the user is accessing the application on
- Accessibility for other institutions - Expanding the database to create specific instances of the application for various collegiate institutions would allow for the app to spread to multiple college campuses
- Language filters/moderation - Monitoring posts to ensure that hate speech, vulgar language, bullying, and other unacceptable forms of communication can never be posted in the first place
- Hashtags - Allow posters to attach hashtags to their posts to allow viewers to more accurately search for topics they are interested in even if they do not already follow the account
- Club/Interest Filtering - Allow users to search key terms associated with clubs that may not be located within their username to increase their ability to find clubs they are interested and want to follow

- Deleting accounts/posts - implement functionality to delete accounts/posts from appearing in the front end of the application while retaining the record of their existence in the database

How to Run the App

In order to download and run the application in its current state, one must:

1. Have node.js installed
2. Clone the repository found at https://github.com/michaelcummins1/StudentEngagementApp
3. Open a terminal from within the repository
4. Run npm install on the following (found in package.json)
   a. bcrypt
   b. better-sqlite3
   c. cors
   d. dotenv
   e. ejs
   f. express
   g. jsonwebtoken
   h. Uuid
5. Navigate to the file app.js and change the port number found on line 7 to a port that will work for you
6. Run ‘node app.js’ within the terminal and traverse to localhost:{port number you entered} in a browser of your choice

    a. *Optional* enter google developer mode by pressing F12 to be able to view the screen as it would appear on a mobile device

7. If you do not already have an account created, press the button create an account and fill out the information
8. Return to the login screen, input your account information, and freely use the app

What We Wish We Knew At the Start of Development

If we were to go back in time to instill in our past selves knowledge to assist in the development of this application, we would focus on discussion of an ‘evenly distributed’ implementation. By ‘evenly distributed,’ we are referencing the order in which we developed the model, view, and controller. A majority of our implementation time was spent on fleshing out the database creation and views of the application, and we were left with not enough time to draw all of the connections using the controller. This left us with multiple features of our application that we had to cut. We also would have focused more on pre and post conditions of functions within our application to allow for an easier implementation of our testing. Furthermore, we most likely would have simply utilized SQLite 3 rather than Better SQLite 3, as the “better” version added multiple complex problems to our controller and model functions which could have been avoided had we used the base SQLite 3. Finally, we would have encouraged our past selves to put a lot of effort and time into this project, as it resulted in numerous valuable lessons for both software development and team work as a whole.