Project Report: FeastFinder

Cody Newton, Sam Garmany, Jack Van-Dyke, Tate Charboneau,
Mason Esslinger, Christopher Chan

April 25, 2025

1 Group Members

- Cody Newton
- Sam Garmany
- Jack Van-Dyke
- Tate Charboneau
- Mason Esslinger
- Christopher Chan

2 Project Description

FeastFinder is a smart, collaborative solution for indecisive dining plans. When you and your friends can't agree on where to eat, FeastFinder makes it easy. Simply create an account, log in, and add your friends. Set your location, dietary preferences, and favorite cuisines, then swipe through local restaurant options—left to skip, right to approve. When everyone in your group swipes right on the same spot, it's a match!

Still unsure who's picking up the tab? Settle it with a fun built-in minigame designed to choose who pays. With FeastFinder, dining decisions are effortless—and entertaining. This project combines web development, database management, and user-centric design to provide a seamless group decision-making tool.

3 Project Tracker - GitHub Project Board

- **Link to Project Tracker:** https://github.com/orgs/Feast-Finder/projects/3

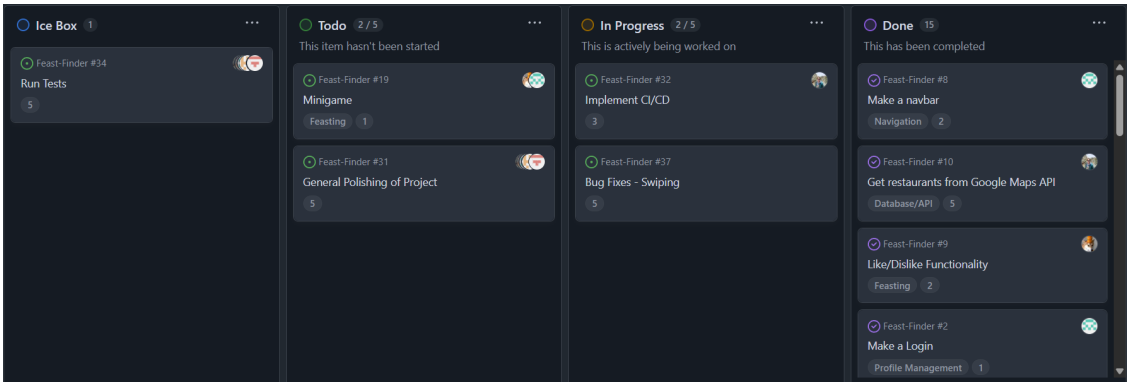

Figure 1: Screenshot of the GitHub Project Tracker

4 Video Demonstration

Video Link: https://www.youtube.com/watch?v=euY4AIK6WNQ

5 Version Control System (VCS)

GitHub Repository: https://github.com/Feast-Finder/Feast-Finder

6 Contributions

Cody Newton

Responsible for the friends page, the profile page, and implementing the main swiping logic both with one person initially, and then group sessions in the end. In charge of debugging towards the end of the project and general cleaning up of code and any errors.

Sam Garmany

Built the early front-end framework using HTML and CSS with Bootstrap. Then transitioned to implementing Azure for cloud hosting. This included configuring GitHub actions for CI/CD and system administration troubleshooting memory overcommit errors.

Jack Van-Dyke

Responsible for working with the Google API and homepage to get a fully functional swiping feature with any location chosen showcasing proper information and photos. Also worked with filtering and indexing of restaurants, general debugging, and UI work.

Tate Charboneau

Designed and implemented PostgreSQL database and produced ERD diagram, wrote project report, wrote ReadME file. Developed UAT tests and helped to implement continuous integration tests. Worked on front end UI and cleaned up visual aesthetics for pages.

Mason Esslinger

Created the minigame to be used for payment decision, general bugfixing, and some UI work.

Christopher Chan

Worked on testing the application, some minigame implementation, and bugfixing.

7 Use Case Diagram

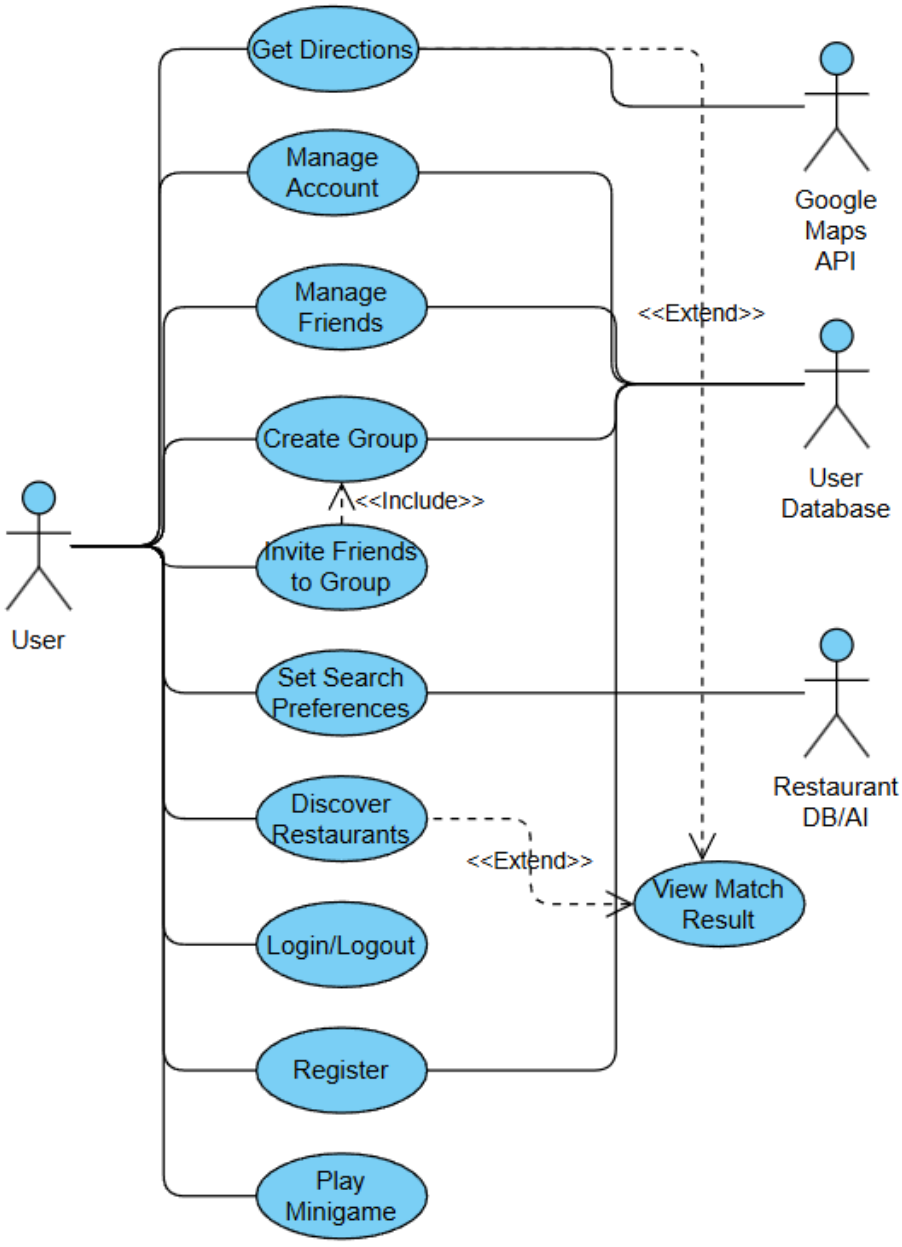

Figure 2: Use Case Diagram for FeastFinder

8 Wireframes

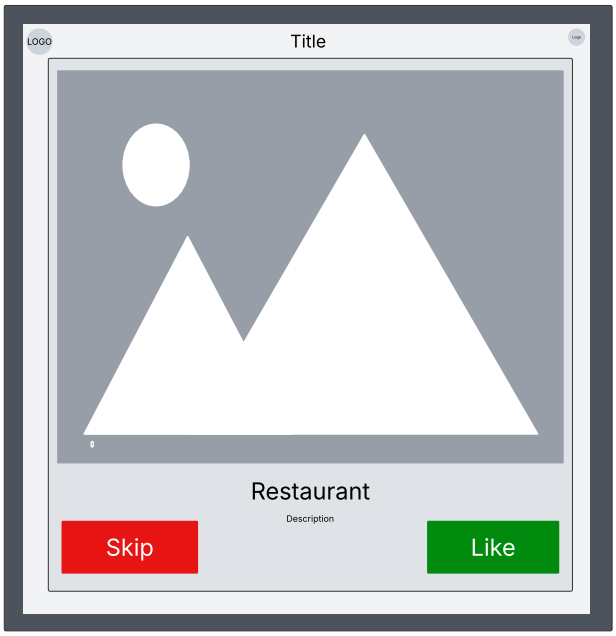

Figure 3: Homepage Wireframe

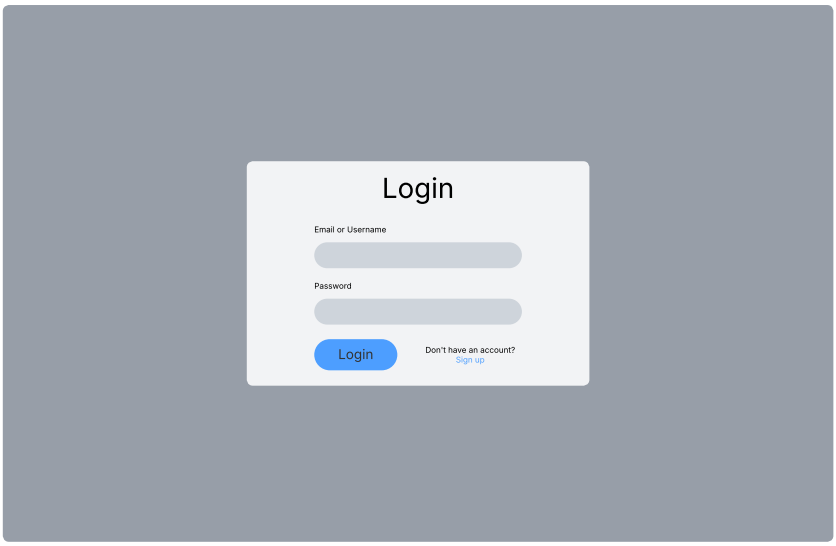

Figure 4: Login Wireframe

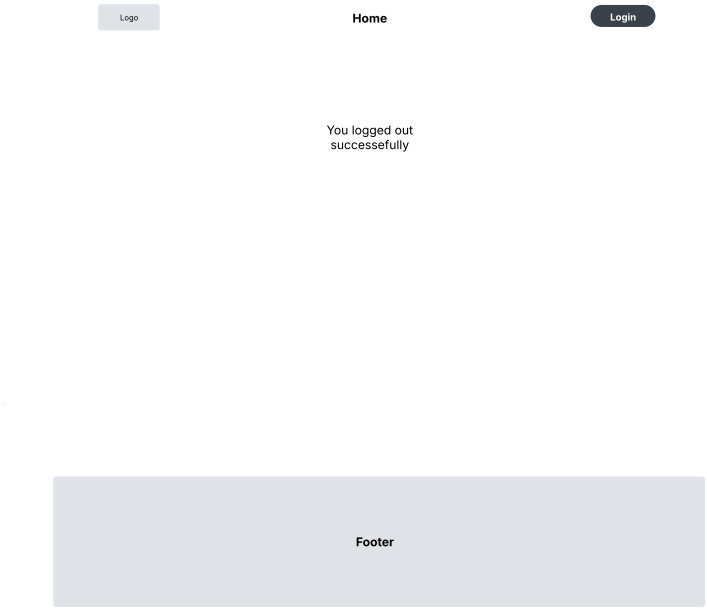

Figure 5: Logout Wireframe

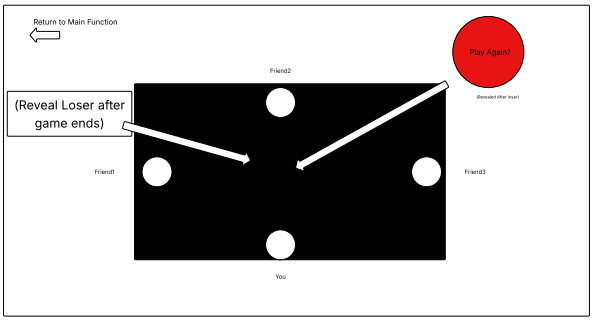

Figure 6: Minigame Wireframe

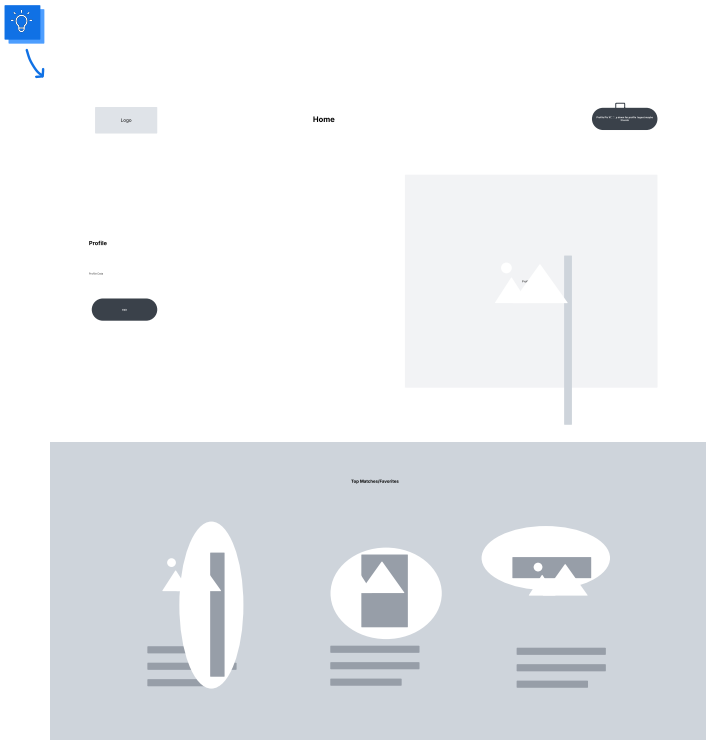

Figure 7: Profile Page Wireframe

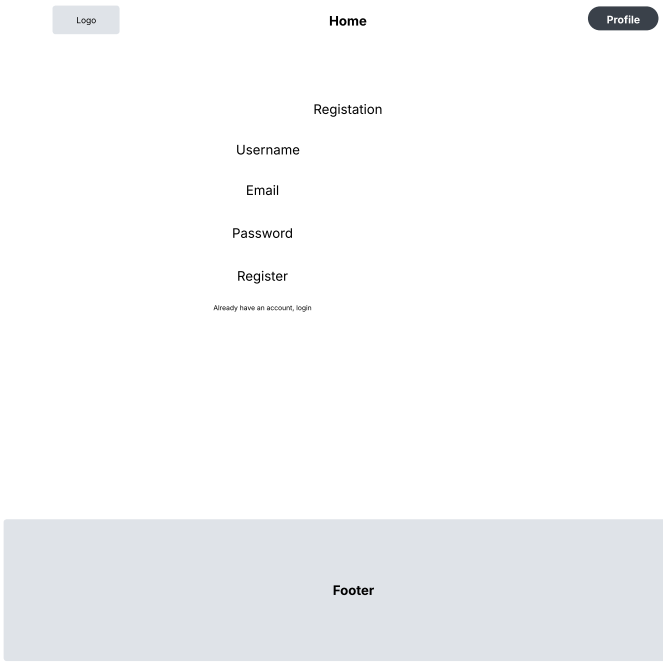

Figure 8: Registration Page Wireframe

9 Test Results

Our test plan from Lab 11 was executed on the final implementation. Results are summarized below:

- All unit and integration tests pass
- UI responsiveness and scaling across desktop and mobile form factors
- Backend swiping logic heavily UAT for groups up to four with no real limitation found.

```
[web]   |
[web]   | > test
[web]   | > mocha
[web]   |
[web]   |
[web]   |
[web]   |   Integration Tests
[web]   | ✅ Redis subClient connected
[web]   | ✅ Redis pubClient connected
[web]   | Database connection successful
[web]   | [TEST] Registered/Ensured testuser2 exists.
[web]   | [TEST] Attempting login for testuser...
[web]   | [TEST] Login attempt completed (Status 200). Assuming agent is authenticated.
[web]   |     GET /home (Authenticated)
[web]   |       ✓ should return status 200 and render the authenticated homepage
[web]   |     GET /search-users (Authenticated - Assuming it requires login)
[web]   |       ✓ should return status 200 for logged-in user (if /search-users exists and is protected)
[web]   |     GET /search-users (Authenticated)
[web]   | [TEST] Found testuser2 with ID: 12
[web]   |       ✓ should return status 200 and find the second test user
[web]   |     POST /friends/remove (Authenticated)
[web]   |       ✓ should remove testuser2 as a friend and return success
[web]   |     POST /check-username
[web]   |       ✓ should return exists: true for an existing username
[web]   |       ✓ should return exists: false for a non-existent username
[web]   | [TEST] Server was not running or already closed.
[web]   |
[web]   |
[web]   |   6 passing (526ms)
[web]   |
```

Figure 9: Unit Test Cases Demonstration

User Acceptance Tests

Feature 1: User Registration

Our team of testers was instructed to register with `username` and `password1!`, which both testers successfully completed. Then they attempted to register with the invalid credentials: `user name` and `password` which was unsuccessful, alerting users to the improper syntax. ✓

Feature 2: User Login

This team was instructed to login with the username and passwords from the previous test, which resulted in a successful redirect to the home page. Then they attempted to login with the credentials `testuser` (which is a valid username) and `wrongpassword` (which is incorrect). This appropriately alerted the user and did not redirect. After this the team attempted to login with a non-existant username and password, `nonexistantusername` and `anypassword`. This also alerted the user about their error and did not redirect. ✓

Feature 3: Create a Group

To test the swiping functionality, this group was setup with a logged in test user and instructed to select "Create Group", input "40.7128, -74.0060" as the location, leave the filters as is, and choose a friend of the testuser, both users completed this successfully. Then they were instructed to repeat this without selecting any friends, and again without inputting a location, both of these failed and prompted the user with their respective modals asking for those fields to be filled. ✓

10 Deployment

Live App: http://feastfinder.centralus.cloudapp.azure.com/
The application is deployed on Azure using Docker containers. PostgreSQL and Redis services are containerized and deployed alongside the frontend/backend services. Azure handles scaling and traffic management. The app is accessible through any modern browser via the above link.