Title: WalletWatch – A social finance tracker

Team Members & GitHub Usernames:

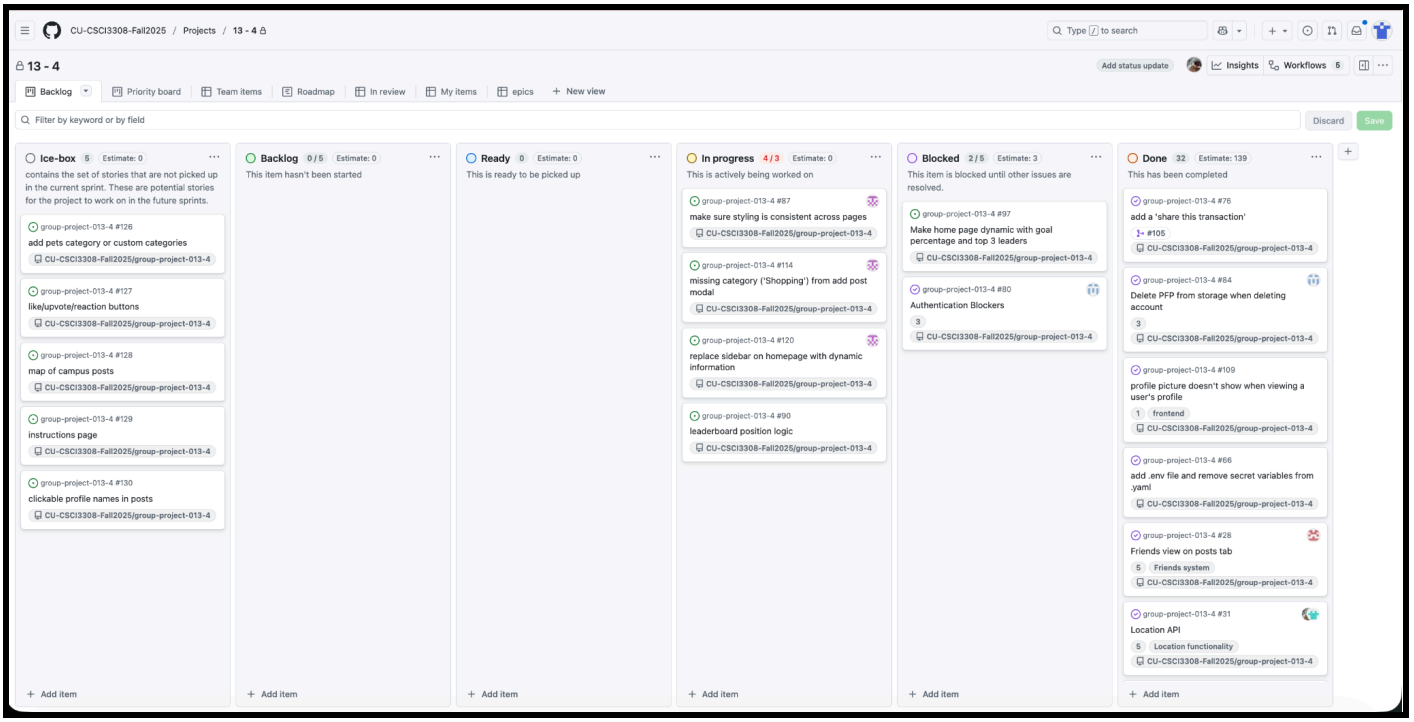
1. Matt Topham (thetopham)
2. Ellie Odau (elliebell1)
3. Harrie Ha (harrieha)
4. Aiden Johnson (AJohnson-13)
5. Emir Simsek (emirsimsek00)
6. Kishore Karthikeyan (Kishore90820)

Project Description:

WalletWatch is a social finance tracking web app that helps users manage spending, set budgets, and share transactions with friends in a social media like way. Built using Node.js, Express, PostgreSQL, and Handlebars, the app allows users to log expenses, visualize and create budgets, and add friends to share their transactions with. Deployed on Render with persistent PostgreSQL storage, WalletWatch uses JWT based authentication, Multer for media uploads, and SSE for live feed updates. This app aims to encourage and spread financial awareness while putting a social twist on traditional budgeting apps.

Project Tracker:

https://github.com/orgs/CU-CSCI3308-Fall2025/projects/6/views/1



Demo Video Link:

https://youtu.be/lv5jNhjBJaY

VCS:

https://github.com/CU-CSCI3308-Fall2025/group-project-013-4

Deployment link:

https://walletwatch-013-4-3zg7.onrender.com/

Contributions:

Kishore's contributions:
I led the backend architecture by designing and implementing the initial database schema for users, posts, transactions, and friends. I also built the first mockup of the website, creating things like the navbar, static home feeds, static leaderboards, etc. to get an idea of what the end project should look like. I also built the user authentication system, including secure login/register routes, password hashing, and JWT token authorization. I set up the transactions page, creating features like add, delete, and delete transactions. I also did a major refactoring of the entire codebase to improve readability and modularity, cleaned up the structures of our routes, and fixed queries in the home feed to ensure posts could only be read by friends. Additionally, I contributed to the initial CSS and UI layout and resolved several bugs during integration and final testing. Finally, I contributed to logistical tasks like making the project report and contributing to half the final presentation.

Aiden's contributions:
I mainly worked on UI elements and certain features on the pages. Most notably I worked on the settings page, profile page, and home page. I implemented the HTML for the design of the pages as well as the features on the pages. On the profile page I implemented a profile picture option that uses multer to store uploaded profile pictures for the users and then displays them dynamically on the NavBar. I also made it so when you change profile pictures it will delete the previous one in the multer storage. On the settings page I also implemented a delete account button that deletes the account from the database. On the profile page I made it dynamically display the pfp, posts and post count of the desired user. Finally on the home page and throughout the website I implemented a functioning NavBar.

Harrie's contributions: I mainly contributed to the implementation of the budget page. I created the backend routes for the budget page for creating, updating, summarizing, and deleting budgets, which includes the SQL that aggregates each user's current-month spending by category and calculates remaining amounts. On the frontend, I implemented the budget page's behavior with JavaScript which includes budget modal behavior, form input validation, and calling budget endpoints. I also rendered dynamic 'envelope' cards with budget, amount spent, remaining amount, status-based progress bars, delete buttons, as well as an overall summary card with total budget, total spent, remaining amount, and savings percentage. I also integrated Chart.js to generate the monthly spending breakdown pie chart with category-based colors and tooltips and also had empty/error states for cases with no transactions, no budget, etc. In addition to this, I also helped with early logistics such as setting up a shared Google Drive folder

for easy collaboration and helped with aspects like adding null/whitespace validation to the posts database.

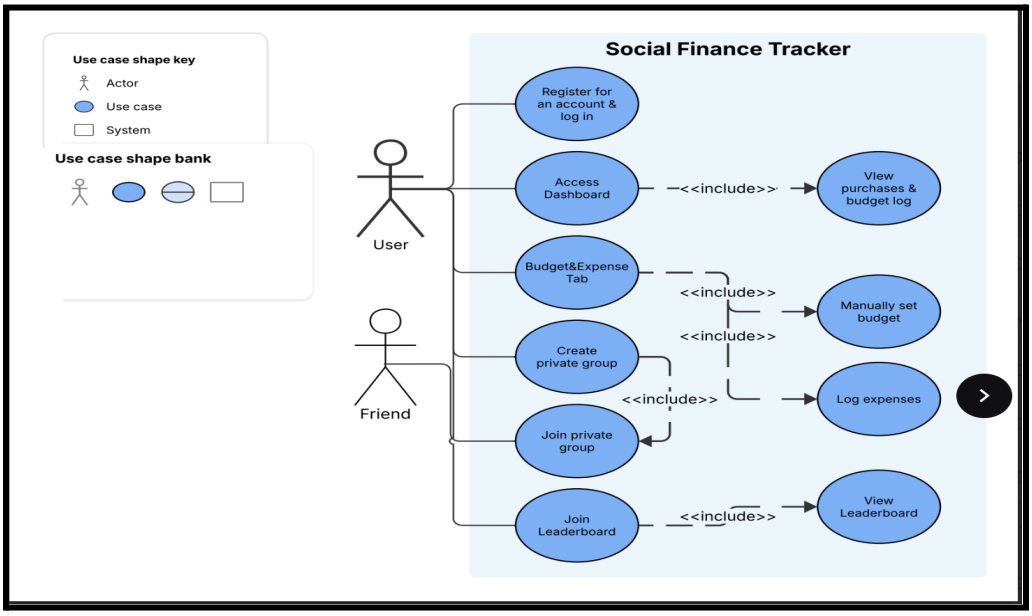
Ellie's contributions:
The main areas I focused on were the full-stack implementation of the friends feature, the styling and UI of the budget page, and the logo design. For the friends feature, I took the initial database schema created by Kishore and created the backend routes to send requests, accept requests, decline requests, search for users by username, fetch the user's pending requests, fetch the user's current friends, and remove an existing friend. On the frontend, I built the friends page, arranging it into three main components: a sidebar for current friends, where each friend had a context menu that leads to an option to remove them or view their profile; a sidebar for incoming requests, with buttons to accept or decline each request; and a search bar in the center that updates suggestions in real time as you type. For the budget page frontend, I followed a similar layout. I aligned the spending breakdown pie chart, the overall spending summary, and the add budget button to the left sidebar. The budgets themselves are each a card in a grid of budgets with a colored tag corresponding to the colors on the pie chart and the transactions page. The progress bar on the budget also changes color based on how much of their budget the user has spent. For the logo, I created an SVG using Adobe Illustrator. The logo combines a dollar bill with a camera to emphasize the social aspect of the platform. Besides these three main focuses, I also implemented dynamic sidebars on the homepage that display the top three leaders from the leaderboard and the spending summary from the budget page. I also updated the leaderboard page to display each user's profile picture, and made sure styling was consistent and clean across all the pages (e.g. buttons, headers, color schemes). Outside of the codebase, I helped the team stay organized by setting up the project board, setting up a recurring booking for a meeting room, and regularly checking in with other team members to see where they were at throughout the week and if they needed any help.

Matt's contributions:
As scrum master, I led team meetings and kept the project github organized. Created the group discord and organized the group direction during the initial project starting chaos. Created release notes each week on github. Introduced the team to github issues and pull requests, and how they work together, such as linking resolved issues in pull request comments. Reviewed pull requests, resolved merge conflicts, and fixed things that were overwritten, broken or deleted after a merge (.gitignore, .env.example, removing uploaded .env credentials, resolving broken javascript links and missing handlebar headers/footer pointer files). My codebase contributions consisted of creating the api use documentation and user validation plan that expanded on lab 7 as a scaffold and then planning the api routes and additional features from there. Refactored the codebase to meet lab 7 scaffold. Added extra credit login tests to the test server file. Researched and created the location api feature with google maps api, using browser location to lookup the users longitude and latitude, then calling the google maps api to find location based on long/lat, then calling google places to lookup nearby locations, linking the google map location to posts, and also implemented a fallback function that uses Google's free nominatim.openstreetmap.org lookup if the google maps api key is unavailable. Created the project readme scaffold from lab 7. Created a share transaction button that prefills a post with

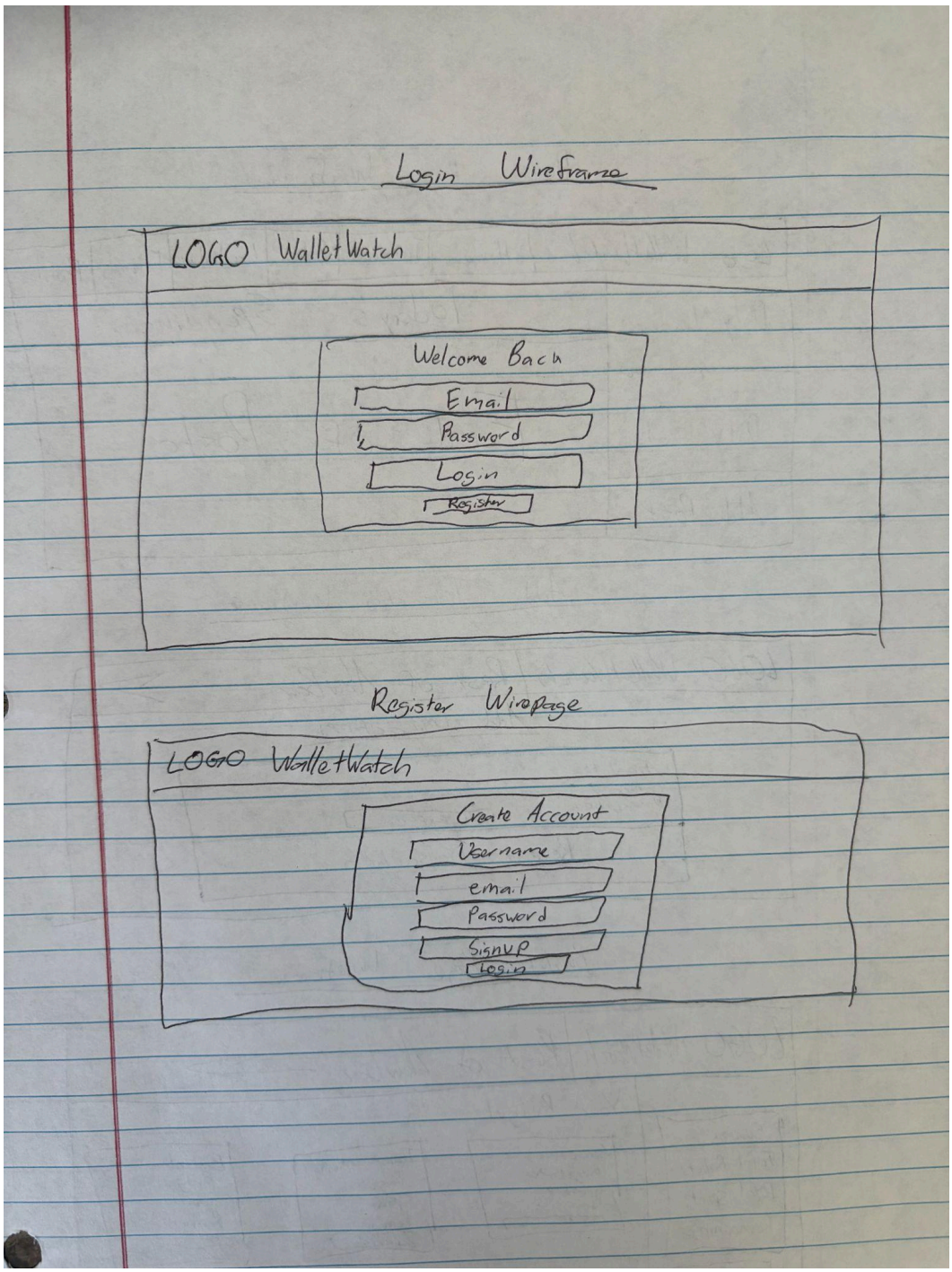
the transaction data, and implemented the vice versa of adding post to transactions feature that creates a transaction from the post. Modified the test server file so that it deletes users after creating them on registration and login so that the leaderboard wouldn't have a ton of test users. Created the lab 12 render deployment that rebuilds the project on every commit and adds builds to pull requests. Miscellaneous improvements, such as adding delete budget button, responsive mobile navigation, adding a pets category, and adding a link to profile pages after clicking on usernames in user posts.

Emir's contributions: My contributions focused on building and improving the social features of the app, especially the parts that let users interact with their friends. I implemented the Friends Feed, which shows posts made by a user's accepted friends, sorted in reverse chronological order. I handled the backend queries to pull only friend-visible posts, added real-time updates so new posts appear immediately, and made sure the feed responds correctly when friends are added, removed, or delete their posts. I also added proper loading/error states and placeholder messages for empty feeds. I developed the Leaderboard feature, including both the global and friends-only leaderboards. Later, I changed this into a full Savings Percentage Leaderboard, which ranks users based on how much of their monthly budget they've saved. This required updating the backend SQL logic, joining budget and transaction data, calculating savings percentages, and adjusting the frontend to show savings, spending, and budget information clearly. Beyond those features, I worked on improving overall usability by fixing the NavBar highlighting, ensuring the correct page is marked active across all routes. I also integrated the posts system with the homepage so real users show up on the feed, wired the "Add Post" functionality to update in real time, and helped refine the connection between our backend routes, database, and UI for a smoother user experience.

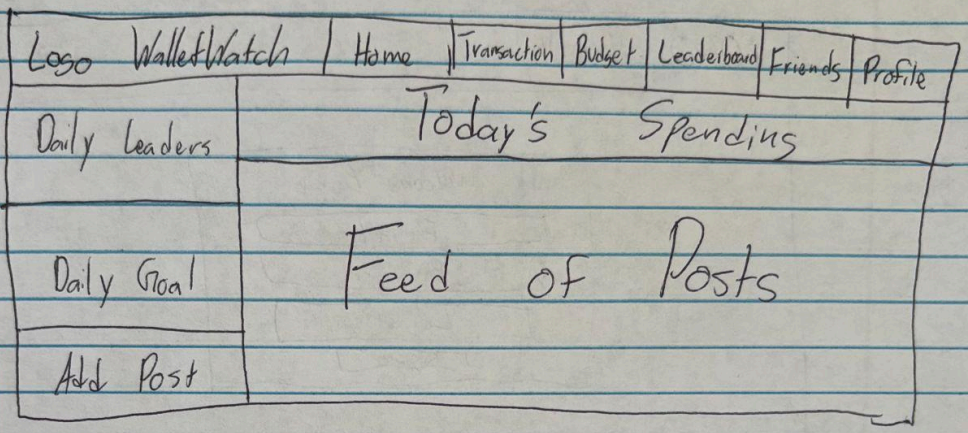
Use Case Diagram:



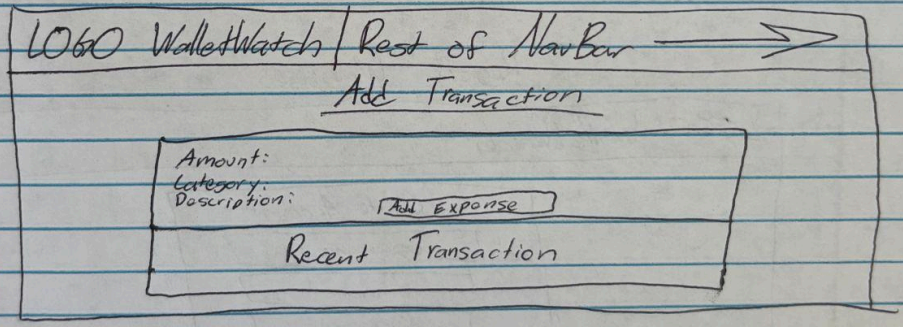
Wireframes:


Login Wireframe
LOGO WalletWatch
Welcome Back
Email
Password
Login
Register
Register Wirepage
LOGO WalletWatch
Create Account
Username
email
Password
SignUP
Login

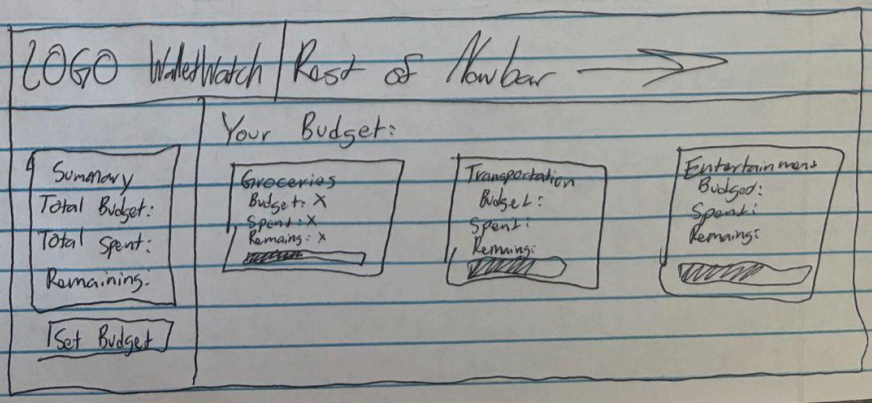
Home Page WireFrame



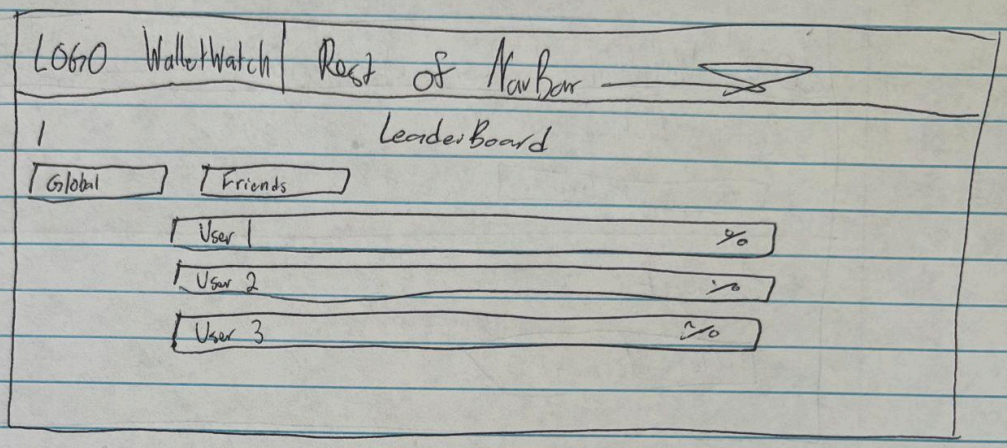
Add Transaction WireFrame



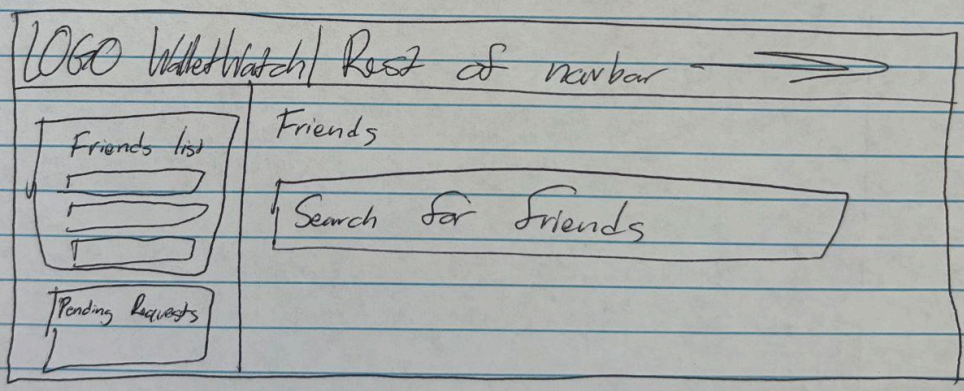
Budget Page WireFrame



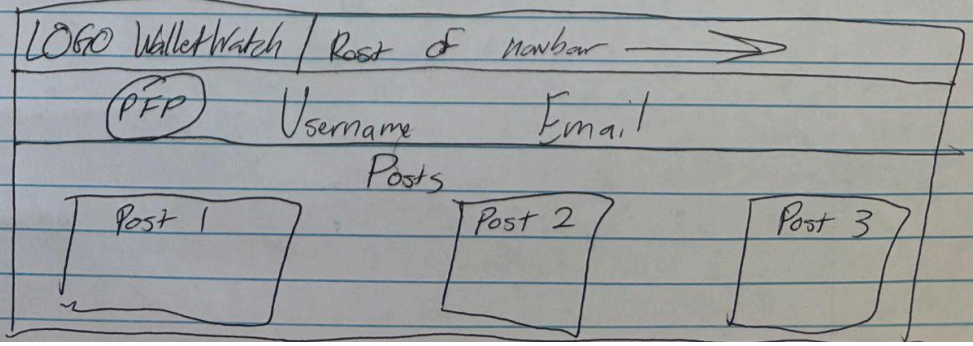
Leaderboard Page Wireframe



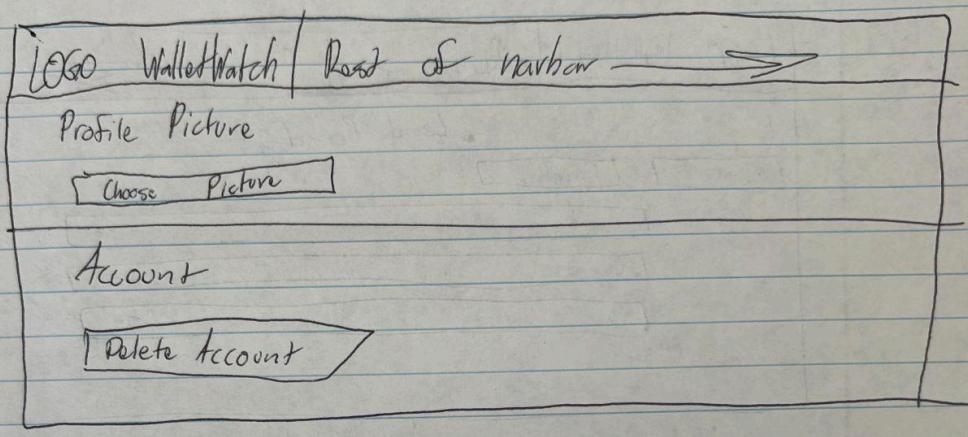
Friends Page Wireframe



Profile Page Wireframe



Settings Page Wireframe



Test Results:

Overview: We implemented automated backend tests cases using Mocha, Chai, and Chai-HTTP to validate our authentication services and ensure API behavior before deployment. We tested to see if our registration, login, and welcome endpoints respond correct under both valid and invalid conditions.

Test Case 1 – Default Welcome Endpoint (GET /welcome)
Purpose: Verify server is running
Expected Result: Status code 200
Actual Result: Pass – server responded with Status Code 200
Observation: Confirms Express server is initialized correctly

- User clicked on the link to the website and was directed to the login page
- User initially tried to register and was confused until spotting the register link.
- User was able to click the register link as a new user (no deviation from expected actions, user found registration link intuitively placed underneath the login button)

Test Case 2 – Register User; Positive (POST /api/auth/register)
Purpose: Ensure that a new user can register successfully with valid credentials
Steps:

1. Send unique username, email, & password
2. Expect a 201 status with JSON body containing user info and JWT token

Expected Result: Status 201; Body included id, username, email, and token
Actual Result: Pass – user successfully registered
Observation: Registration correctly validates input and hashes password

- After navigating to the registration page, User inputted username, email, and password.
- After inputting, User was successfully navigated to the home page. (no deviations)

Test Case 3 –Register User; Negative (POST /api/auth/register)
Purpose: Validate error handling when required fields are missing
Steps:

1. Send only an email field
2. Expect a 400 status and message saying “fill all the fields”

Expected Result: Status code 400
Actual Result: Pass – route rejected incomplete registration
Observation: Confirms backend validation logic prevents incomplete user registration

- User inputted only the email field and attempted to register.
- Error message “Please fill out this field” occurred and registration was rejected.

Test Case 4 – Login; Positive (POST /api/auth/login)
Purpose: Ensure login works for an already created user
Steps:

1. Register a user

2. Attempt to login that user with same credentials
3. Expect a 200 status with a valid JWT token returned

Expected Result: Status code 200 and body includes id, username, email, and token
Actual Result: Pass – valid users logged in successfully
Observation: Demonstrate correct password hashing and JWT generation

- User logged out after registering, and was navigated back to the login page.
- User used the same credentials and was able to successfully login.

Test Case 5 – Login; Negative (POST /api/auth/login)
Purpose: Ensure invalid credentials are rejected
Steps:

1. Attempt login with non-existent user
2. Expect 400 status and invalid credentials

Expected Result: Proper error handling and no token issued
Actual Result: Pass – invalid users denied access
Observation: Confirms secure authentication flow

- User attempted to login before registering.
- User got the error message “Invalid credentials” and wasn’t able to log in.

Changes made after testing:

- Adjusted error messages for missing fields to be more user friendly
- Verified JWT generation and hashing consistency between test and actual environment