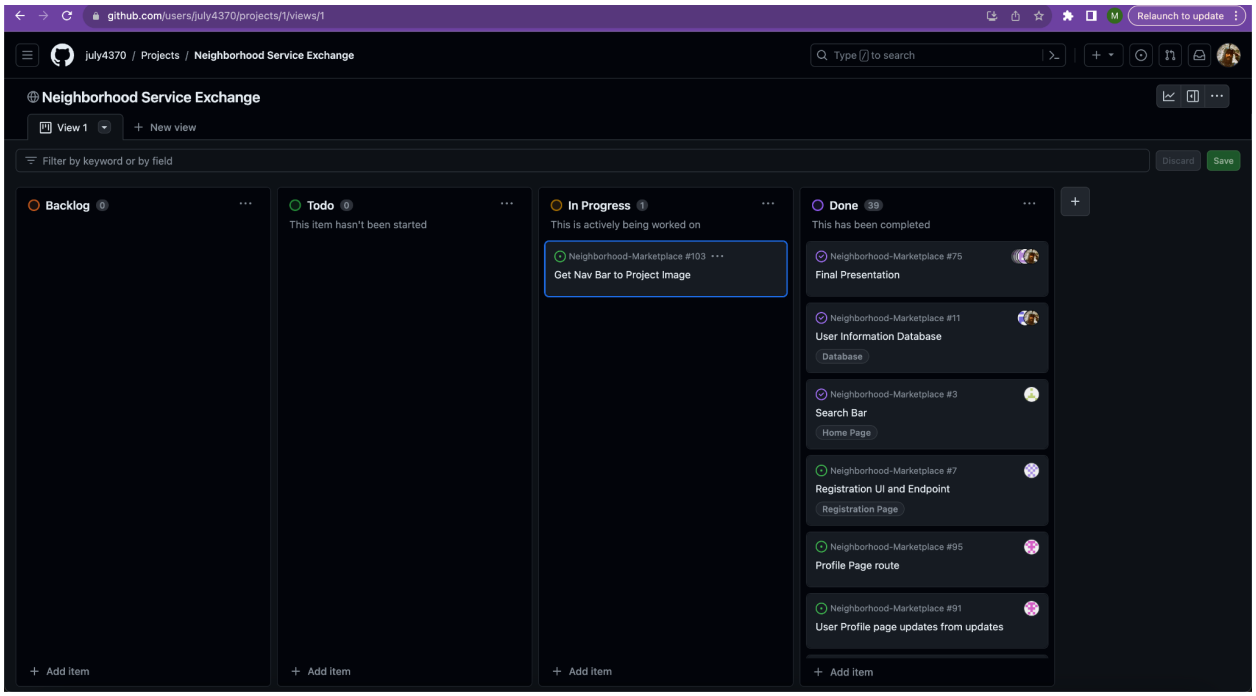
Neighborhood Service Exchange

By Jayas Basnet, Clement Canel, Justin Lynn, Mohamed Mohamedali, Jonah Tatterson, Miles Zheng

Description:The neighborhood service exchange platform is for local communities where neighbors can offer, exchange, or barter services. The platform is supposed to be built on trust and close relationships, competing with the aspect bigger job platforms cannot truly provide. This platform is powered by a SQL database that stores data like profiles, listings, and history. The front-end of the platform, designed with HTML, EJS, and CSS, allows for users to navigate through different pages and job functions. The server-side operations are handled by NodeJS, this is where we made the routes for different functions and features. The entire backend is hosted on Amazon's Azure platform, ensuring users enjoy a smooth and seamlessly integrated experience on the website. The robust infrastructure not only enhances performance but also ensures reliable and secure access to all features, contributing to a user-friendly online environment. To enhance the community approach, the platform would integrate Google Maps for localized search functionality. Advanced messaging APIs would deliver notifications and alerts. Features like favoriting services and a delete function provide flexibility and control to users. We would also ensure real-time notifications, location-based searches, and more user interactions. Additionally, the sorting mechanism we added allows users to efficiently organize and filter service listings based on cost, favorite, and postdate, enhancing the user experience.

Project Tracker:



https://github.com/users/july4370/projects/1/views/1

Video:
https://clipchamp.com/watch/7glhFgJJeh6

Github:
https://github.com/july4370/Neighborhood-Marketplace

Contributions (less than 100 words each):

Miles Zheng: I made the database and set up both the user and jobs table. I also adjusted the table when we needed to add rows. I also created the footer partials and cleaned up the look of the UI. I set up the jobs page to display the jobs in the table using ejs. I helped make some of the routes for the jobs page and also just rendering some of the pages like the home page. I also organized the minutes of our weekly meetings and helped the group stay on task.

Clement Canel: I worked mainly on the job page for the website. I added the routes on the job page to favorite, delete, search, and sort through jobs. I made all the styles the same and made them look pretty. I added posted_on and is_favorite to the database for the job page endpoints. I also made two different pages on the job page for users to see there jobs from the post.

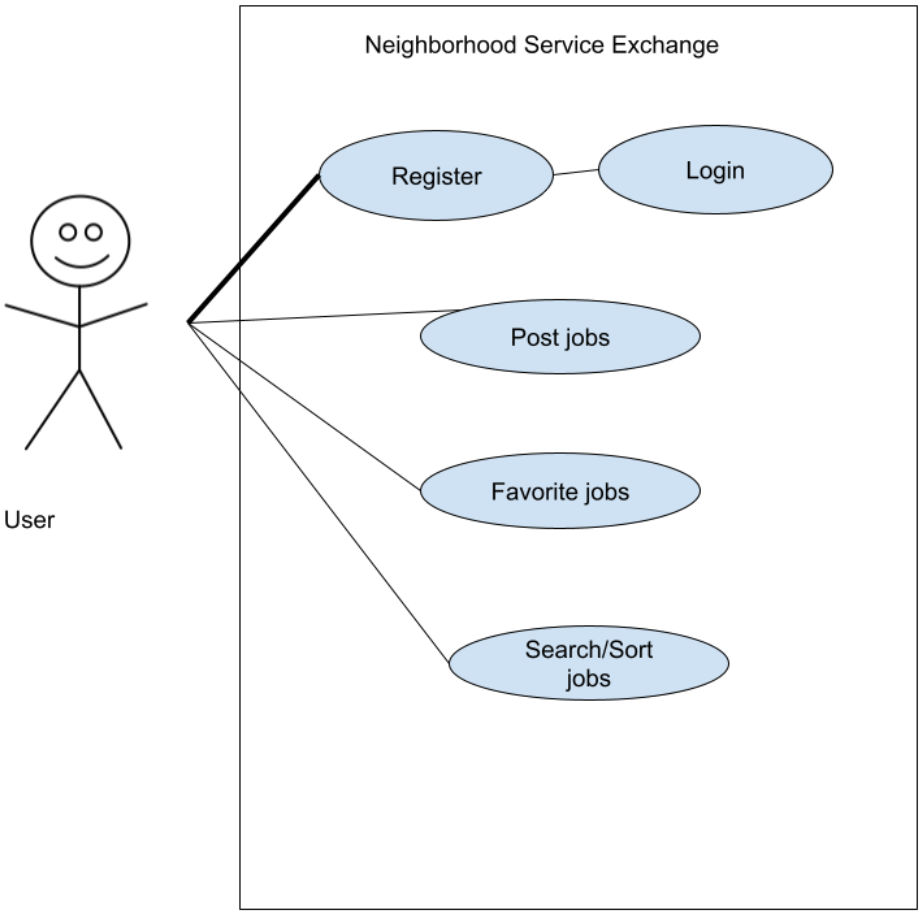
Jayas Basnet: I made the login page, about page and the register page. I also made the routes for the login page and register page. I made the routes for rendering most of the pages including all of the ones on the navbar. I added middleware into the code. I also worked on debugging pages including register, login page, jobs/post, and debugging the database.

Jonah Tatterson: I made the home page and profile page. I worked on some of the routing for the home page so that find job and post job tabs were functional from the home page. For the profile page, I set up display user information and worked on editing profile information. I also worked on some of the routing for profile and debugging for getting user sessions to update.

Justin Lynn: along with some other team members I made the home page and helped with the routes that went along with it, specifically getting the right pages to render going to and from the home page. I also helped fix some of the database when we had bugs and partially worked on the jobs routes.

Mohamed Mohamedali: I started off making the test cases for our routes using mocha, I then helped develop our register and login routes, adding in error checking. I added in a couple render routes as well. I also added error implementation into our UI in our html providing clear feedback on what went wrong. I also helped debug and initialize the database.

Use Case Diagram:



Test Results:

Observations and Findings

1. Login Feature

User Actions: Users easily navigated to the login page and entered their credentials.

Reasoning: Familiarity with standard login procedures.

Consistency: Most users followed the expected process without issues.

Deviations: A few users struggled getting back to login page

Changes Made: Made automatic return to login page when you make an account.

2. Logout Feature

User Actions: Users located and used the logout button effectively.

Reasoning: The logout button was clearly labeled and placed intuitively.

Consistency: All users successfully logged out without confusion.

Deviations: None observed.

Changes Made: No changes necessary.

3. Favoriting Jobs

User Actions: Users interacted with the favoriting feature frequently.

Reasoning: To easily find and access preferred job listings later.

Consistency: The feature was well-received and used as intended.

Deviations: None significant.

Changes Made: No changes necessary.

4. Deleting Jobs

User Actions: Users attempted to delete jobs, including those not posted by them.

Reasoning: Misunderstanding of their permissions and the feature's intent.

Consistency: The behavior was consistent across multiple users.

Deviations: Users deleting jobs not belonging to them.

Changes Made: Removed the delete button to prevent unauthorized deletions.

5. Posting Jobs

User Actions: Users posted jobs but faced difficulties in locating their postings.

Reasoning: Lack of a dedicated section to view their own job postings.

Consistency: Several users reported this issue.

Deviations: Users expected a feature to view their own posts which was absent.

Changes Made: Added a new page for users to easily find and manage the jobs they posted.

Deployment:

We deployed our web application using Microsoft Azure's cloud platform. First, we set up a Linux Virtual Machine under Azure's Free Services, following the specified configuration to avoid consuming our credit. We downloaded a private key for secure access and ensured the VM was provisioned correctly. We configured the DNS for our VM. We added necessary inbound security rules, particularly opening port 3000, which our web solution uses. To connect to the VM, we used an SSH client. With everything in place, we ran our web solution. Our website is now live and can be accessed at **http://recitation-14-team-04.eastus.cloudapp.azure.com:3000/home** (Down right now to not use up our credit)