Spatial Crowdsourcing for Dynamic Settings

SDMay22-31

Goce Trajcevski	-	Client/Faculty Advisor
Abdula Eljaam	-	Frontend Website and Mobile Development & Backend
Aman Agarwal	-	Frontend Website and Mobile Development
Cole Dulaney	-	Backend development
Isaac Reed	-	Frontend Mobile Development
Seth Platt	-	Frontend Mobile Development & Client/Vendor Interaction
Shagun Bansal	-	Frontend Mobile Development
Yuichi Hamamoto	-	Backend and Algorithm Development

Email - sdmay22-31@iastate.edu
Website - https://sdmay22-31.sd.ece.iastate.edu/

Tables and Figures

Tables

Figures

Table of Contents

1 Team

Details regarding SDMAY-31 and its members

1.1 Team Members

1. Abdula Eljaam
2. Aman Agarwal
3. Isaac Reed
4. Seth Platt
5. Cole Dulaney
6. Shagun Bansal
7. Yuichi Hamamoto

1.2 Required Skill Sets

1. Familiarity with SCRUM / Agile
2. Website and Mobile application development
3. Database management
4. MapBox Location API
5. SpringBoot Framework

1.3 Skill Sets Covered by Team

- **Javascript, HTML, CSS:** Cole Dulaney, Aman Agarwal, Isaac Reed
- **Web application development:** Cole Dulaney, Aman Agarwal, Isaac Reed, Shagun Bansal
- **Mobile Application Development:** Seth Platt, Aman Agarwal, Yuichi Hamamoto
- **Database management:** Isaac Reed, Yuichi Hamamoto
- **MapBox Location API:** Isaac Reed, Aman Agarwal
- **SpringBoot Framework:** Abdula Eljaam, Yuichi Hamamoto, Cole Dulaney

1.4 Project Management Style

Our team used agile methodologies. This style is extremely common both in the industry as well as in projects like ours, aiming to develop “proof-of-concept” prototypes, and has proven to be very effective throughout the semester.

2 Introduction

Last semester our group was tasked with creating both an Android and Web application to solve the problem of efficient management of spatial crowdsourcing in dynamic settings. This semester we split into various sub-teams and worked to create these applications.

2.1 Problem Statement

The issue of spatial crowdsourcing involves three main components: tasks, workers, and a platform. However, In **dynamic** spatial crowdsourcing, the workers and tasks have a location associated with them and the time of requesting a service for a particular task is not fixed. In this case, workers are assigned tasks by the platform in real-time based on several possible optimization criteria, which may include: availability of the worker, wage of the worker, and distance between the worker and task location.

The main objective of this project has been to develop a system that implements the algorithmic solutions for the workers-to-tasks assignments in the settings of dynamic spatial crowdsourcing, and provide an interface that the respective categories of users can rely on. Specifically, our system seeks to provide assignments to workers based on the needs specified. Our system targets two kinds of users - Workers who execute tasks and Clients who post requests for tasks. The application also allows users to set preferences for how they want their task to be handled. These preferences, such as distance or cost, change how the algorithm assigns a worker to their task.

2.2 Requirements & Constraints

Functional requirements:

- The mobile and web app are able to display status of assignment and worker in real-time

  → React allowed us to develop both mobile and web apps.

- Only appropriate users are able to see the location/info of workers

  → Provide permission only to appropriate users.

- Interpret existing worker data to decide optimal routes and assignments

  → With the location API, collect users' location and use the algorithm to make an optimal decision.

- The application is scalable for multiple users

  → Ensures the scalability of the application by choosing the right technology and practices. Using DevOps to manage the project and developing the application.

- Job requests are updated in real time

  → Workers are able to provide regular updates on the request through the UI.

- Workers are able to post their availability and location

→ Workers are prompted to give their location when posting a job, with real-time data updates on the jobs they're working.

- Application calculates the optimal assignment of worker to task

  → Designed an algorithm for assigning the optimal assignment to the worker based on user preferences.

- Users are able to view status of their requests in real-time

  → Updates made by the worker are visible at the customer's UI.

- Data storage and tracking of completed tasks

  → All user and assignment data is synchronously stored in the database.

- Optimized for fast response time

  → Ensure industry standard coding practices while using multi-threading for faster response time.

Non-functional requirements:

- Accessible from mobile devices and PC

  → Using React allowed us to develop for both mobile and desktop.

- Design is extensible.

  → Practice clean code and proper documentation of each task assigned.

- Data is stored safely and accurately, and its integrity is maintained across API hits

  → Data is stored in the database and uses a queue pattern in order to ensure all data is stored successfully.

- Clean and best practice code to improve future maintainability

  → Team members' performance were evaluated every other week to ensure all the best practices are being followed.

- Application is functioning reliably

  → Protects the integrity, availability, and confidentiality of the application and its users. Also, prepare an emergency plan in case there's a breach.

- Privacy and security of workers

  → Constant testing of our application/code allowed us to find and avoid any data leaks. Use of 2 factor authentication at user login.

- Intuitive and "clean" web and mobile UI

  → We ensured we consulted our group with every UI change to prevent clusters in the UI.

Constraints:

- Budget - for computational resources
- Time [1250 hours for two semesters]
- Updates should be propagated to users within 3 to 5 seconds.
- Must abide by all privacy and business laws regarding all users' data
- Visualization of location on the map should be accurate.

2.3 Engineering Standards

The following table lists IEEE standards we have abided by, and why they were relevant to our project's success.

Engineering Standards	Justification
IEEE Std 1228, Standard for Software Safety Plans	As we require the usage/storage of sensitive information, we took user security seriously and planned around it.
IEEE Std 1012, Standard for Software Verification and Validation	We ensured that our implementation met the client's requirements, and did everything they needed to do. I.e we worked to make sure that we completely addressed the problem statement.
IEEE Std 1219, Standard for Software Maintenance	We worked to safely and actively maintain the application to provide the best possible user experience. Data integrity and security were maintained throughout this process as well.
IEEE Std 1063, Standard for Software User Documentation	Our application and service is documented properly and thoroughly; both for users and future developers. This will impact scalability and maintainability, among many other factors in any future use of the product.
IEEE Std 982.1, Standard Dictionary of Measures to Produce Reliable Software	As a company's day-to-day operations may revolve around this application, we had to provide a reliable service for them to use
IEEE 1008-1987 - IEEE Standard for Software Unit Testing	In order to ensure our application functions properly and meets reliability standards we followed the accepted standards for software testing.
IEEE 12207.2-1997 - IEEE/EIA Guide -	We followed industry standards and best

Industry Implementation of International Standard ISO/IEC 12207 : 1995 (ISO/IEC 12207) Standard for Information Technology-Software Life Cycle Processes - Implementation considerations	practices throughout our development process both to ensure a quality final product, and to create a well organized development process. This was helped by the team's Agile based development methodology.

Table 1: Applicable Engineering standards

2.4 Users and Related Use Cases

Users / Actors:

- Worker: A skilled worker of any trade who will be assigned to complete tasks
- Client: Person posting tasks they need done for them

Use Case	Applicable Roles / Actors	Details
New Client Registration	Client	Clients are able to register and create an account through our application.
New Worker Registration	Worker	Workers are able to register and create an account through our thorough application.
Task assignment similar to services such as GrubHub	Worker	The system analyzes a user's preferences and makes the best possible task assignment. Clients can set their preferences to lower distance from the worker or lower cost.
View status of assignments/workers in real-time	Client	Authorized users can view the status of a job as the worker executes it.
Individual worker task stats displayed	Worker	A worker can view the statistics for said worker's task history.
Task completion update	Worker	Worker updates the task as completed, and the task is removed from the "Active Tasks" section on the user

		end.
Task completion confirmation	Client	The client receives a confirmation email once the task is completed.
Remove/add tasks	Client	A client is able to remove or add tasks from/to the list of available tasks in the system.
Worker clock in/out	Worker	The worker is able to clock in and out of work which denotes that the worker is ready to work and take tasks on.
Setting worker status and personal settings	Worker	Workers are able to set their availability by clocking in/out, adjust their list of skills, update their personal info, etc.

Table 2: Breakdown of Use Cases and General Functionality Descriptions

2.5 Why Our Application Stands Out From Similar Services

In the modern day, there are actually many applications which intend to solve the issue of spatial crowdsourcing for a specific subgroup of tasks. A few prominent examples are rideshare services such as Uber or Lyft, and food delivery services such as DoorDash and Eat Street. These companies provide similar tools, however they only handle location data and a singular skill/task; not multiple skills / and differing tasks. Due to this kind of restricted scopes, these popular solutions cannot truly be considered equal solutions to ours.

This advancement makes us stand out from any existing competition. Our service takes the concept that has caused these apps to take the world by storm, and combines it with worker skills and a large variety of tasks to provide a general user solution to spatial crowdsourcing. Rather than simply having Geographical Information System functionality with a basic simple task, we will have detailed functionality to provide clients a way to get all services they may need in one convenient location. This will also subsequently attract more workers, as we will serve more professions. Workers may also have a better experience on our services than other popular services. This is due to the dynamic spatial crowdsourcing algorithm giving them their optimal job rather than the worker needing to manually find and accept tasks.

2.6 Relevant Readings Regarding Spatial Crowdsourcing

"Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment", *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

Tong, Y., Zhou, Z., Zeng, Y. *et al.* Spatial crowdsourcing: a survey. *The VLDB Journal* 29, 217–250 (2020). https://doi.org/10.1007/s00778-019-00568-7

Zhao, Y. et. all (2019, June 12). *Destination-aware task assignment in spatial crowdsourcing: A worker decomposition approach*. IEEE Xplore. Retrieved September 22, 2021, accessed from https://ieeexplore.ieee.org/document/8735884.

Zhao, C *et. all* (2020) gMission Server [Source code]. https://github.com/gmission/gmission.

NOTE: The secondary links in the cited Zhao, C *et. all* (2020) gMission Git repository's README are a useful resource both for comprehending the concept of spatial crowdsourcing, and for seeing examples of implementation.

3 Revised Project Plan

Detailed plan for solving the problem of Dynamic Spatial Crowdsourcing via our application. This project plan has been updated to detail our revised strategy.

3.1 Project Management & Tracking Procedures

We used the agile project management style for our project. It is commonly used in software development, especially so for web and mobile application projects, both of which are also deliverables for our project. Our goals and objectives were amenable to incremental development, which is another characteristic of agile project management.

We used Gitlab to track version history and to allow for simultaneous development. Git issue boards have been used to organize and assign tasks. Alongside this, Gitlab milestones and epics assisted in tracking progress, and ensured the development process was moving forward in a timely manner. Group communication, useful resources, and meeting minutes have been organized in the team discord server. Important documents and reports were stored in a shared Google Drive folder.

3.2 Project Milestones, Metrics, and Evaluation Criteria

The project was evaluated with the use of standard agile charts and documentation. The development team and stakeholders used our non-functional requirements, as described in the design document, for evaluation. The specific criteria on which we evaluated the project's progress is as follows:

1. User experience elements such as response time across all platforms
2. Design extensibility
3. Ensuring user data is stored so that only authorized users may access it
4. Worker location data is stored accurately in terms of proximity to exact location as best attainable by GPS sensors
5. Clean and best practice code to improve future maintainability
6. The UI for both web and mobile should be Intuitive and appealing to the users

These metrics were organized and evaluated using the following tools:

1. Burndown Chart
2. Gantt Chart
3. Code reviews by peers
4. JMeter (API testing tool)
5. Postman

3.2.1 Fall Semester 2021 (491) Milestones

September

1. Familiarize ourselves with the specifics of the project and problem statement
2. Completed first stage of requirements engineering process

October

1. Application scope and industry finalized
2. Project plan document finalized and accepted by client

November

1. Solidified design ideas for both UI and structural implementation.
2. Selection of application platforms, frameworks, languages, and tools are finalized and documented

December

1. Documentation for the final design proposal, testing plans, and development/implementation plan completed

3.2.2 Spring Semester (492) Milestones

January

1. Finalize business complications regarding planned tools
2. Confirm new tools with client, and have team familiarize themselves with them,

February

1. First iteration of user types and their functionality.
2. First iteration of application UI.
3. The application's functionalities have all been optimized and connected smoothly. It now responds within the standard **1 second** time frame. If a request takes longer, some indication is provided to ensure the user something is happening and the app has not stopped.

March

1. Creation of additional security measures and quality of life (QOL) features
2. Finalizing the functionality of the two separate user roles.
3. First iteration of task creation and posting system for users with the client role
4. Data controllers optimized such that all important changes should be disclosed to the relevant users in the same **1-10 second** real time window. An example of these changes would be the addition/removal of a task or driver.

April

1. The worker assignment algorithm completed, and optimized for minimal runtime, such that it fulfills the proposed quality attribute. Clients now receive a timely UI response confirming this process has occurred.
2. Worker location data implemented via MapBox. The controller is optimized such that the data is reflected in real time. This means it shall stay within the standard margin of **1-10 seconds** to update.
3. Testing both applications for ease of use and functionality

May

1. Documentation for the final report, poster, and presentation finalized.
2. Project presented to client, stakeholders, and industry panel

3.3 Project Timeline(s)

Gantt Charts detailing the timeline of the project for both semesters

Semester 1:

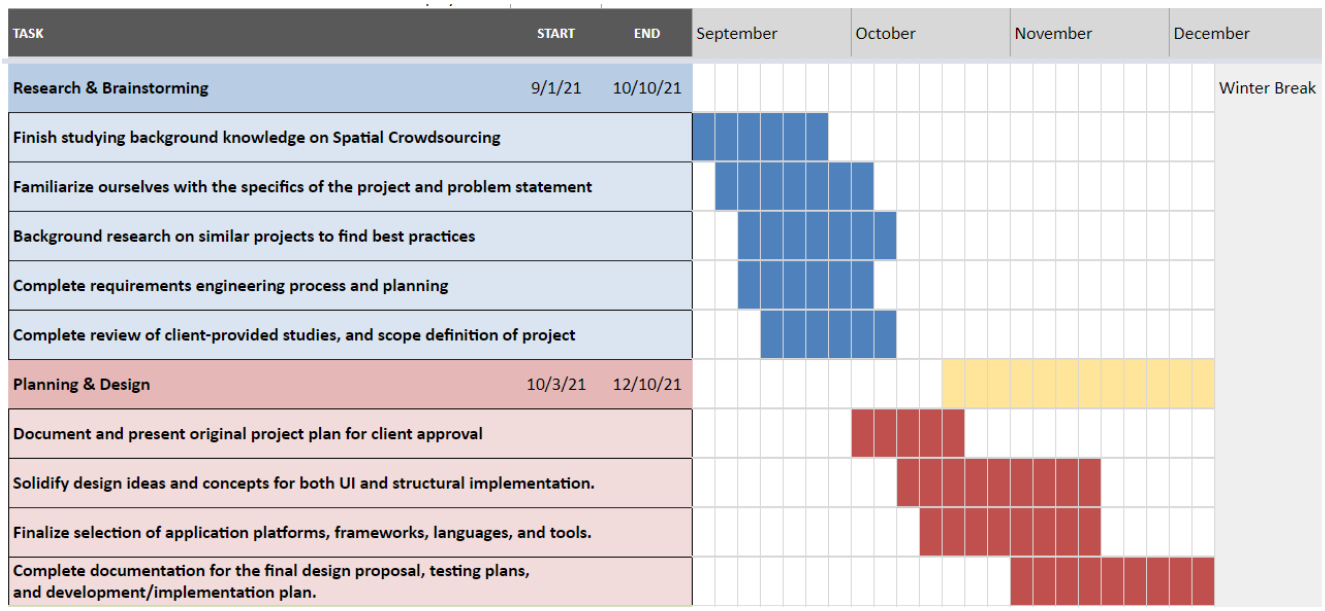

Figure 1: 491 Gantt Chart

Semester 2:

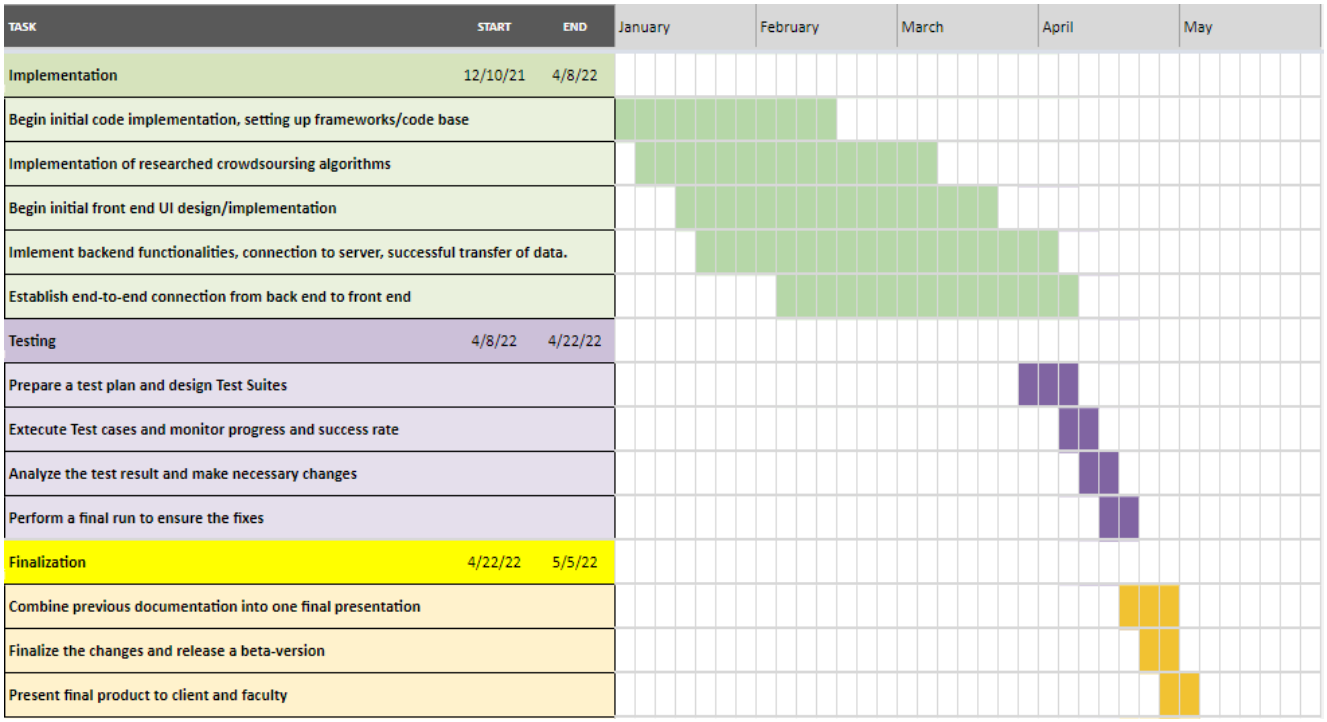

Figure 2: 492 Gantt Chart

4 Design

Detailed explanation of the application's design process from conceptualization to finalization with regards to project requirements

4.1 Design Context

Broad description of the requirements, challenges, and context of the application

4.1.1 Broader Context

Our broad context was a situation where we have N job-sites (which can post requests for servicing a task) and M workers (each of whom has his/her own skill descriptions and respective locations). As we are crowdsourcing, our workers are assigned to these job-sites based on various factors depending upon the current situation. We targeted individual consumers and service providers by our model. This has the potential to increase competition in our communities, and thus provide better and more affordable services.

Below are a few areas and descriptions of how the project would have the ability to impact them.

Area	Description	Examples
Public health, safety, and welfare	Workers - 1. By increasing economic opportunity for workers in the community, this would also positively impact overall health and welfare, as workers will be able to more comfortably afford more of their expenses. 2. The workers who respond to the tasks could take the risk of responding to malicious users: i.e. users who "pretend" to have an issue, but make a job request for another (possibly malicious) reason. Mitigating the amount of travel workers do is also important, which would decrease the likelihood of a traffic accident or other travel-related issues. Users - 1. Users should see health/safety increase with the use of our solution, as users may request jobs that, if left	Workers - 1. A worker who previously struggled in being able to find jobs is given more opportunities (which results in increased mental wellbeing for workers) as a direct result of the algorithmic solution implemented. 2. A malicious user uses the app to request a job, but when the worker arrives at the job site, neither the user nor the job is anywhere to be found. Another applicable example would be requesting an Uber driver, but canceling last-minute, resulting in wasted time and money for the worker. Users - 1. A user posts a job request for a leaky pipe which is quickly completed by a worker. If left unattended, this pipe could've caused harmful effects to the user's health. 2. A user makes a job request to fix a leaky pipe that is assigned

	uncompleted, could impact their physical or mental health and welfare. 2. Similar to workers, users may take the risk of “malicious” workers. These would be workers who use the product and are in some way negligent to the job they are assigned. This would also have the same effect if a user posts a job they need done urgently, but the job is never completed by a worker.	(unintentionally) to a malicious or negligent worker that doesn’t show up, or does not complete the job. This could result in not only damage to the user’s plumbing and infrastructure, but also health risk to the user due to mold if not attended to in a timely manner.
Global, cultural, and social	It will help organize professional services and their execution by professional workers and help with the organization and transportation of those with similar social desires.	Development or operation of the solution would violate a profession’s code of ethics, implementation of the solution would require an undesired change in community practices
Environmental	By dynamically optimizing routes of workers the emissions from workers vehicles will be reduced between jobs.	With algorithmically calculated optimized routes for the workers, excess car emission can be controlled helping the environment.
Economic	It enables more competition amongst service providers which in turn can provide more competitive pricing. More competition also promotes better service and enables more flexibility for the customers. The proposed solution would also create more jobs for workers in the community/area in which it’s implemented.	If a customer insists on having his job finished in a certain hour and is willing to pay more, the system should enable such assignments.

Table 3: Breakdown of Broader Context

4.1.2 Technical Complexity

From a technical side, our project has three main components: frontend, backend and database. Alongside these it makes use of MapBox’s location API. Our frontend is composed of two client types, a web browser and mobile application, both of these provide the functionality for the user to provide information input to the backend and query the backend data for general profile information, task assignments/updates and location data.

1. The backend stores all of our data in the selected database. As this data changes it processes the data through the algorithm and updates worker assignments. Part of processing the data through this algorithm requires querying the MapBox location API.
2. The frontend provides functionality to query the backend and provide task assignment updates to workers. On the mobile application side of the front end, the application is able to provide location updates for user reference and also for algorithm usage.
3. We have built a general purpose algorithm. While it is comparable to some existing companies/solutions, like Uber or GrubHub, it is not equivalent, as our algorithm takes multiple task types into consideration, as opposed to a single category. This added an extra layer of complexity to the problem with us having to take more than just spatiotemporal data into account.

Technical aspects that increase the complexity of the project:

- Our project has three main technical interfaces
    - Backend to database, providing the algorithm with the data that it needs and writing to the database.
    - Frontend to backend, providing user data for the database, location data, and updating users and workers with task info.
    - Backend interfacing with the MapBox location API to assist with assigning tasks based on spatial data.
- The frontend is not one singular component but has components written for both web and mobile.
- ReactJS was new to some team members and provided some significant hurdles.

Internal complexity

Components and subsystems: Technique identification (processing and database),location services, dynamic route optimization

Scientific, mathematical, or engineering principles: Location detection, task optimization and accounting for dynamic situations and changes.

External complexity

Our project has multiple functional requirements that meet or exceed current state-of-the-art standards, such as: error recognition, dynamic location monitoring, task optimization, input processing, and dynamic contingency handling.

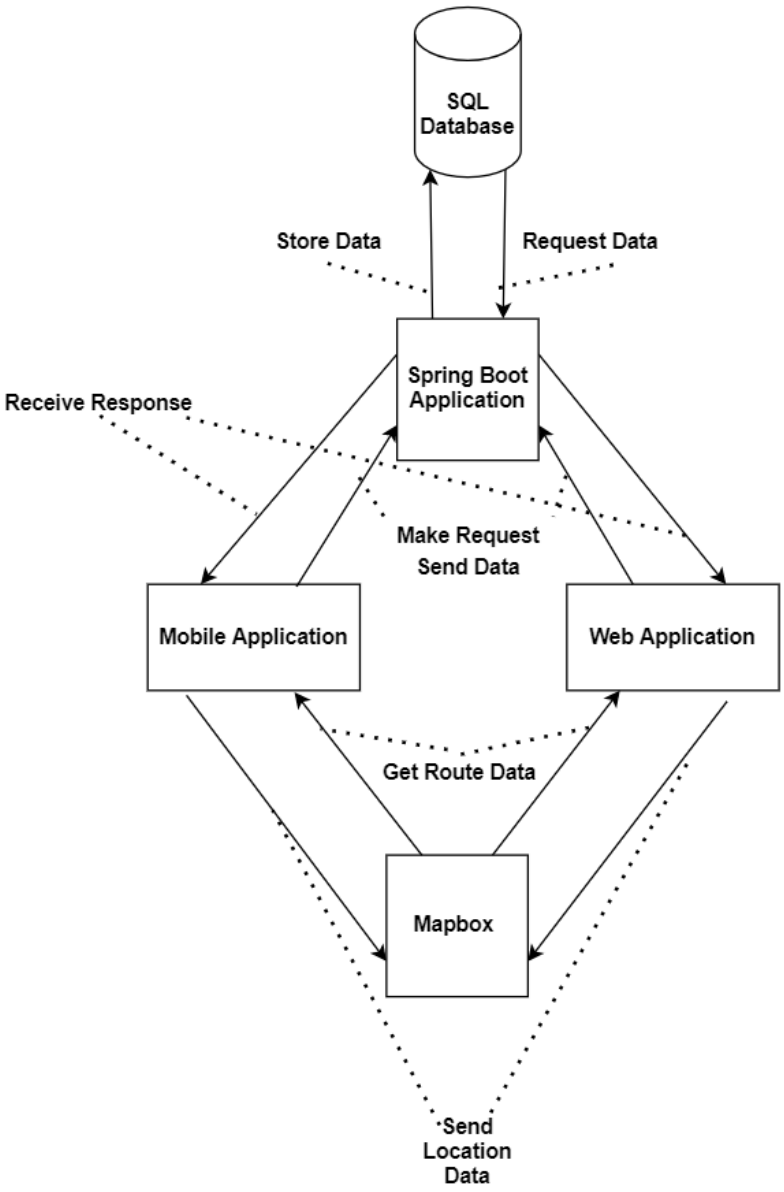

Figure 3: High Level Architecture Overview

4.2 Design Analysis and Changes Made

We carefully considered various design options and made our choices based on various factors such as cost, ease of use, scalability, modifiable, responsiveness, etc. These design choices seemed most suitable during our design process, but as we progressed through initial project development and service acquisition Firebase and Google Maps presented themselves as major hurdles to beginning development.

Over winter break and into the beginning of spring semester there was communication with Google concerning both Firebase for our cloud service provider and dev environment as well as Google Maps for location API. During the process of this communication with both Google and ECPE staff there were delays in hearing from Google, particularly surrounding payment for services that delayed our full development start. After stalling with Google Firebase customer service for multiple weeks, our team decided it would be in our best interest to switch frameworks in order to avoid wasting any more time. As such, we migrated from the use of Firebase for our backend to SpringBoot and from Google Maps to MapBox after exploring available options and consulting with our faculty advisor. While this led to us having to learn new tools, it did not interfere with a majority of our set design choices. Our client also supplied us with a connection to another group, which allowed us to schedule a meeting to learn more about MapBox.

We chose to use a MySQL (relational) database. We decided to prioritize relational, as it made the most sense given the data we needed to store. For the front-end implementation, we decided to go with ReactJs. We looked into other options such as Angular, JavaScript, Vue.js, and Ember.js. Considering our team's experience, ReactJs was chosen as the best choice for our team. Also, ReactJs is easier to understand and execute than the latter options. One of the cons of choosing Angular is the difficult debugging that it comes with, but we prioritized the performance and efficiency of it.

As for general ideation techniques, these decisions were narrowed down using the information provided in prior sections of this document. We balanced our development team's skills with the needs we had documented, and found the best compromises that suited both categories.

4.3 Database and Use Case Diagrams

Diagrams detailing our use cases, and the database schema of our application(s)

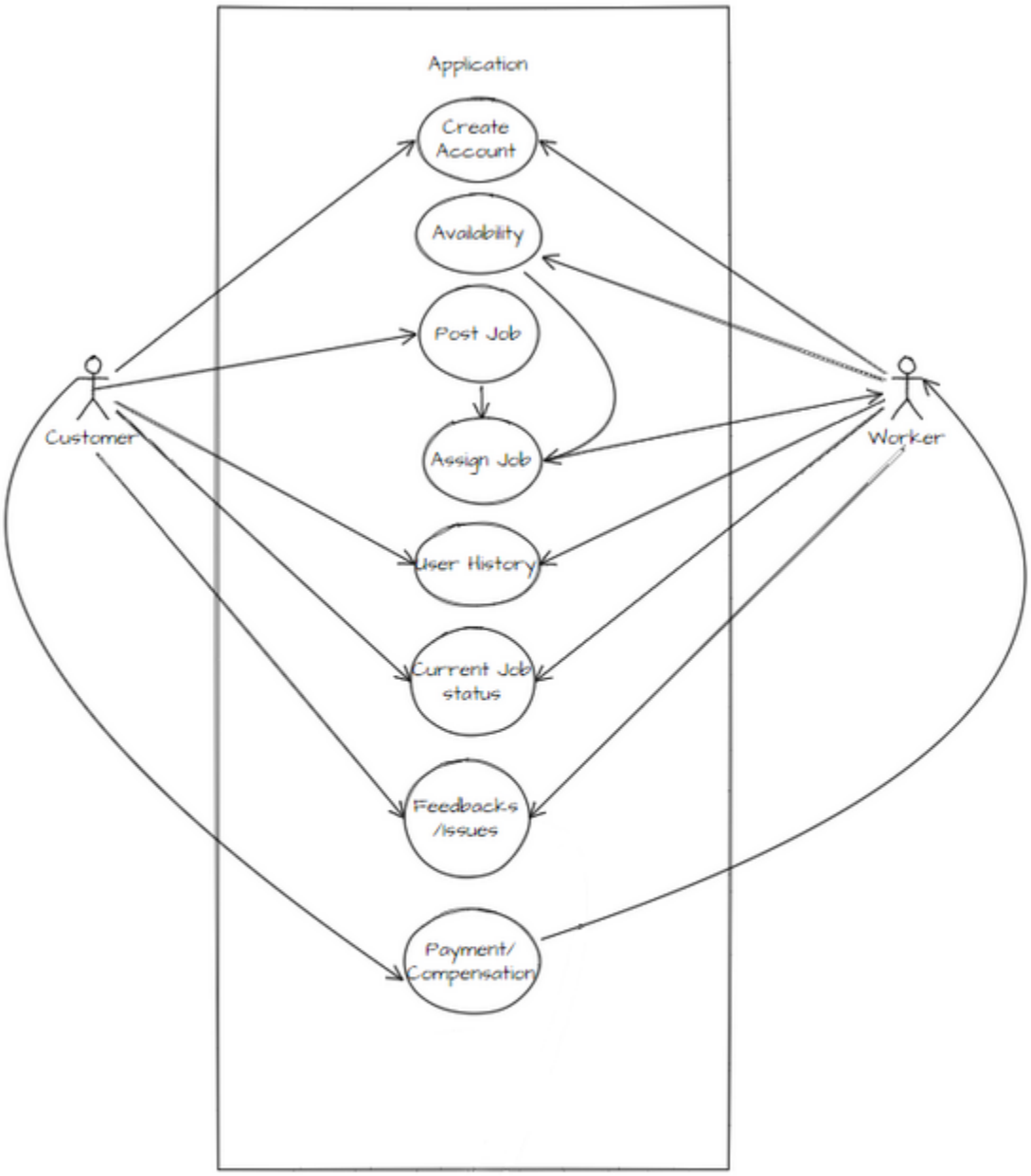

Figure 4: Use Cases

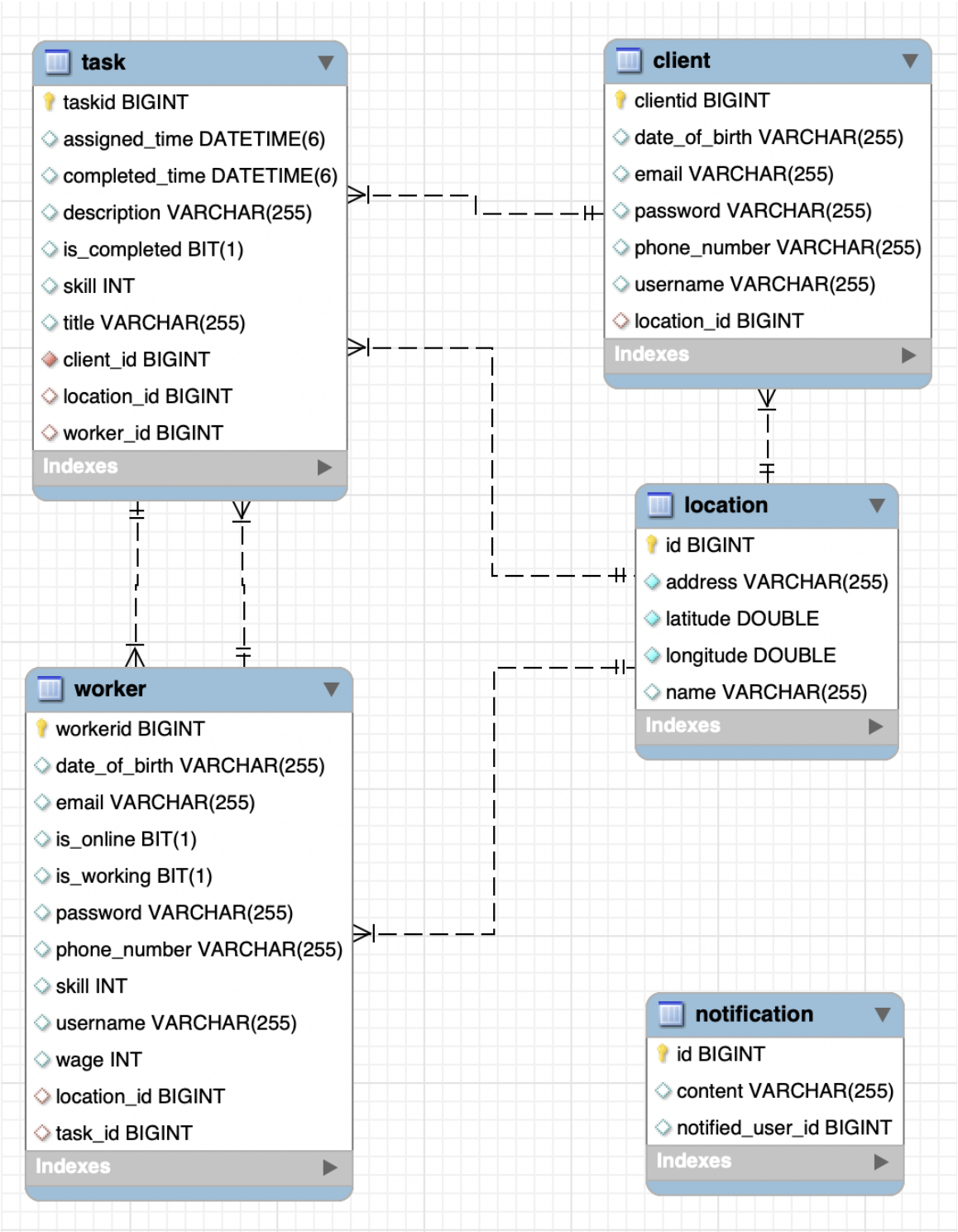

Figure 5: Database diagram

5 Testing

As a part of the Agile development process, testing was built out and executed as different system components were developed. While minor tests were done throughout the development process, following the Agile methodology, the main portion of our testing was done as the final step of that sprint's cycle. This may include, or be followed by a code review by another team member. This system should allow for us to stay organized and systematic, as it is a well-defined and documented process.

A majority of the project-specific challenges we faced throughout the testing process revolved primarily around the MapBox API and end-to-end connections . Testing this portion of the application required the team to read a large amount of MapBox documentation, and to familiarize themselves with the system.

5.1 Unit Testing

Our system can be broadly classified into five major modules: Database, Web Application, Mobile Application, SpringBoot Backend, and MapBox Location API. For testing purposes, we evaluated each module's functionality and features both separately and collectively. Referring back to Figure 3 *(located in section 4.1.2 of this document)*, within each module there are different functional units that collaborate. A few examples of such units are as follows:

- Database module testing.
    - Updates (POST/PUT/GET/DELETE)
        - Each operation was tested with multiple entries and its proper completion was verified.
    - Query procedures (Native Queries via controller services).
- API Testing
    - Load Testing using JMeter
- MapBox Location API Testing
    - Mocked all functionalities needed by the application
    - Analyzed location accuracy and routes.
- UI testing
    - React user event/Event response
    - Ran tests in different browsers
    - Accessibility Testing using AMP
    - Ran tests in different screen sizes

Some of the tools we utilized for unit testing include:

1. JUnit
2. Jmeter
3. Postman
4. Validating results with SQL Workbench tables

5.2 Interface Testing

Our interfaces require connections to the backend and MapBox api. Our primary testing in this part involved making calls to the API from our SpringBoot application and MapBox, and returning spatial information relating to the data that was sent. The data needs to be transformed in a certain way by the application in order to make a proper API call.

Tools:

- Postman
- Mockito

5.3 System Testing

Refer to the functional requirements found in section 2.2 & 2.3 of the *"Requirements, Constraints, and Engineering Standards"* portion of the project documentation. The following system-level tests were performed to ensure that said requirements were met:

- When viewing a map or job status page, current and up to date information should be displayed
- When an unauthorized user tries to access information, they should be rejected and redirected
- When a client posts a task, a visual representation is displayed and their task is stored on the server
- Mocked unit tests on location API to ensure data sent and received is as intended.
- Location tests to ensure the UI properly displays a worker's location when needed.
- General flow interface tests were done to ensure that the movement between portions of the app works as expected
- Tests listed in 5.1 & 5.2 also fall under the scope of system testing.

The tools used throughout this process may include (but are not limited to): Postman, mockito, Jest, and Junit. We also used manual and peer reviews to test basic operations and user experience.

5.4 Regression Testing

Sometimes newly added functionalities or changes in existing code make the existing features non functional. In order to avoid that, a test suite is chosen that covers both the existing features and newly added ones. And the chosen test suite consists of tests that cover different crucial parts of an application, eg. impact, criticality and most used functionalities. In our case, two such crucial aspects that were tested were scalability and extensibility of the application.

1. We tested the scalability by increasing the traffic on our app and increasing the load on our database, testing its impact on the application's overall performance. Apart from load handling, the performance on the user's end was tested to ensure that the users are getting real-time updates with minimum delays, regardless of the fact that the web application or the mobile application is being used.

2. During the design process we made sure that our program remained extensible, and any addition of new features called for bare minimum changes in the existing application. For example, upon completing a branch that updated the map information, we performed tests on the impact on the map services of the app. A direct example of testing we did in this scenario was testing to ensure that the UI display, and assignment algorithm remained functional regardless of the user's location and distance to a task.

5.5 Acceptance Testing

In order to ensure our functional requirements were met, we used the demonstration with demo cases. Demonstrations with demo cases help us target specific requirements to ensure the detection of any faults.

In order to test out non-functional requirements, we merely measure the requirements that are measurable and calculate the percentage of the requirements met to the total number of non-functional requirements.

We involve the client in the acceptance testing, however, we also have certain criteria for acceptance testing which the system as a whole must satisfy. A few examples include:

1. How visually appealing the overall UI is.
2. How intuitive the UI is.
3. How efficient and effective the assignment algorithms are.
4. How extensive the design is.
5. How reliable the application is functioning.

6 Closing Material

Concluding details regarding the team and project, as well as extra relevant information.

6.1 Discussion

The main tools we used include ReactJS/React Native for frontend and mobile development, SpringBoot for handling our backend, and MapBox for dealing with location data. Before we settled on this plan the team did a lot of research to make us as sure as possible that we would not be rushing ahead on our project and unknowingly putting time into a mediocre solution. This research also led to a set of logistics problems involving payment methods which gave us a major setback as we were not able to use our originally planned tools, Google Maps and Firebase, and had to use SpringBoot and MapBox instead.

6.2 Conclusion

We were able to create an application that accurately reflected our goals that were set during the design phase of our project. Through the successful implementation of our application, we were able to effectively show a real-world application for spatial crowdsourcing that could potentially be implemented for a large variety of use cases outside of the scope of our project. Through the design process, we also gained valuable knowledge and hands-on experience in working on a full-stack software application.

7 Appendix 1: Operation Manual

Detailed instructions on how to set up the technical environment for the application, and how to properly use it.

7.1 Setting Up the Application(s)

1. Downloading/Setting up the source code for the application
    a. Using command line or terminal, clone the github repository (https://git.ece.iastate.edu/sd/sdmay22-31/) to your computer.
    b. Downloading these applications: IntelliJ, VSCode, MySQL WorkBench, MySQL Server 8.0, ReactJS, React Native, ExpoCli, & NodeJs
2. Setting up the database in SQL Workbench
    a. Create a new user by the name of “root_2” and password as “password” for the backend application to use. (User may be changed in the application properties located in the SpringBoot application)
    b. Allow the user to have add/delete/update privileges.
    c. Create a new database schema by the name “SDMay22_31”
3. Importing the Spring Boot Application
    a. Import the Dynamic Spatial Crowdsourcing folder as a Maven application inside IntelliJ.
    b. In the IntelliJ application, add a Maven configuration for the SpringBoot application in order to be able to run it.
4. Running the Frontend Application
    a. Website
        i. For the Web Application, using a command line, navigate to the website folder under the Frontend folder, and open the service24 folder. Run “npm install” to install the necessary dependencies. Then run the application using the “npm start” command.
    b. Mobile
        i. For the Mobile Application, using a command line, navigate to the Mobile folder under the Frontend folder, and open the service24 folder. Run “npm install” to install the necessary dependencies. Then run the application using the “expo start --web” command.

7.2 Using the Application(s)

The website starts with the following homepage. The homepage consists of a navigation bar, the problem statement, team members' details, objective of this application, worker registration link, and footer with social media links.

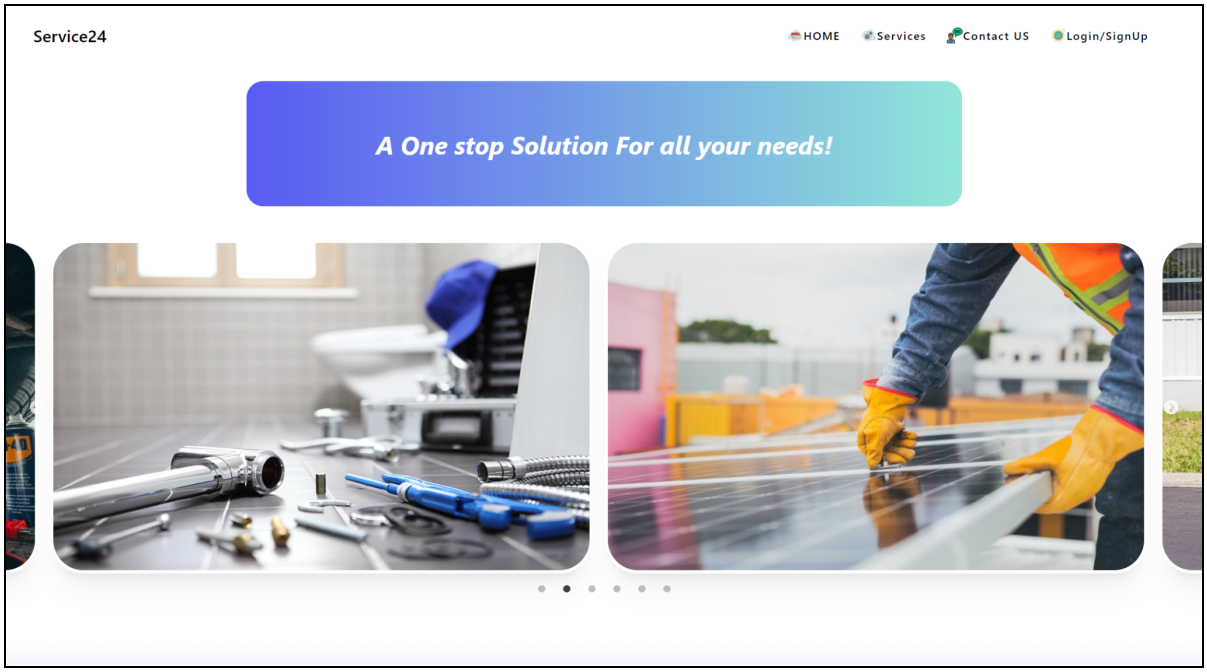

Figure 6: The Homepage of the Web Application

Through the Login/Sign Up menu from the Navigation bar, the Client or Worker may Login or Register a new account. The login page requires the standard username and password to login in, while the register page also collects other details such as date of birth, phone number, email address, etc.

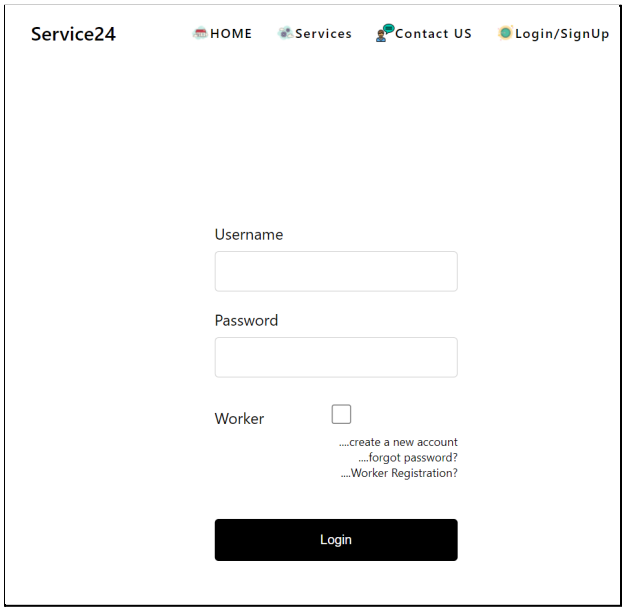

Figure 7: Screen Capture of Login Page

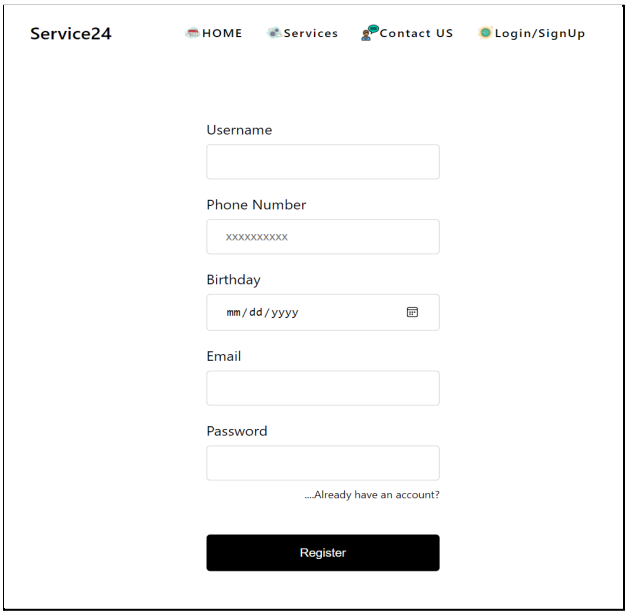

Figure 8: Screen Capture of Registration Page

The Client UI is pretty straight forward. All the actions available can be seen through the clickable header. Each header expands on click to offer a detailed view or perform the action. Each header name corresponds to the associated action.

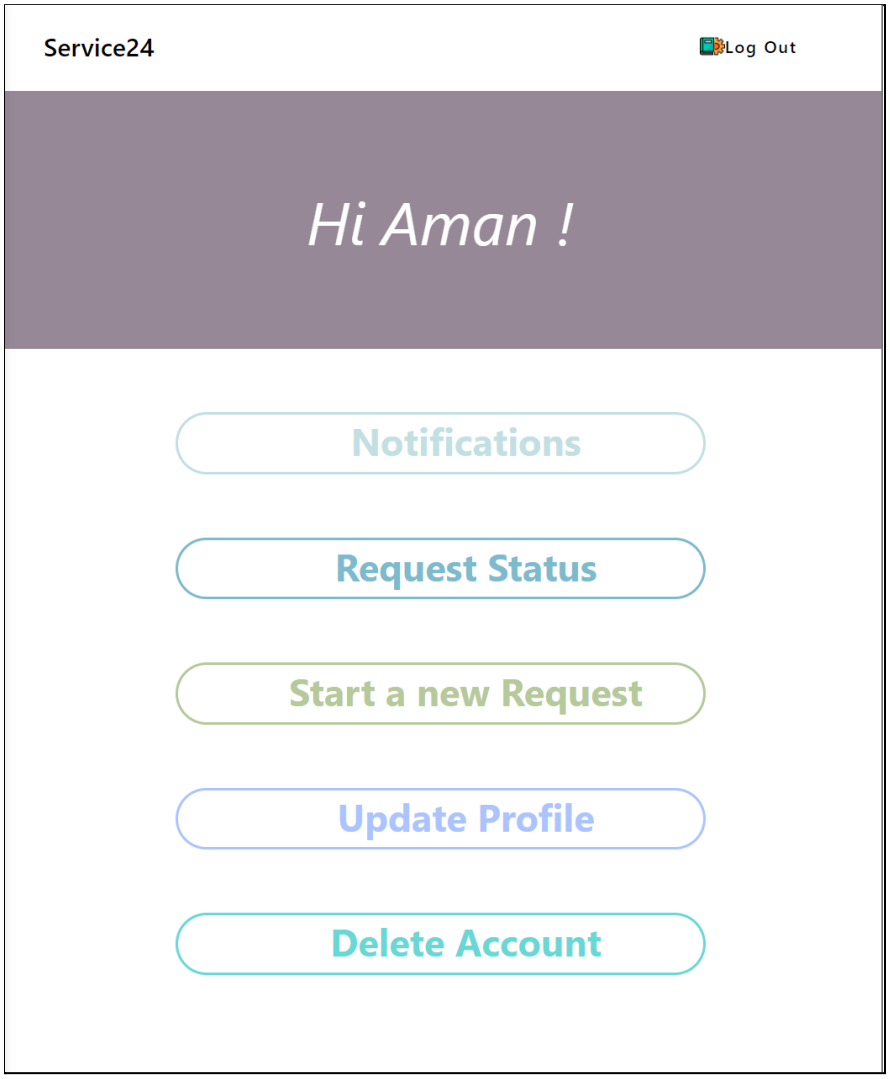

Figure 9: User Dashboard

For example, the update profile header when clicked expands to display the following form where a user may update values to update the associated details with his account. Similarly, the active task status can be viewed through the request status tab.

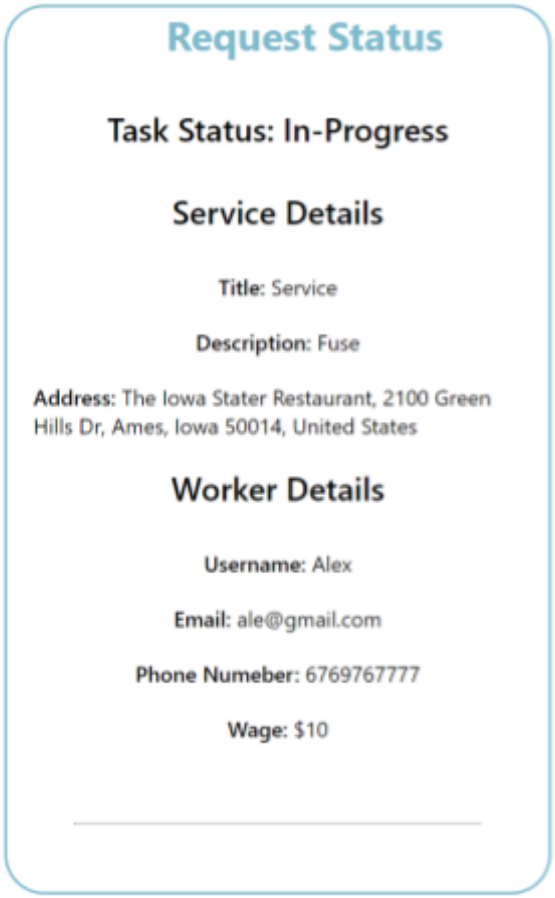

Figure 10: User Side Task Status

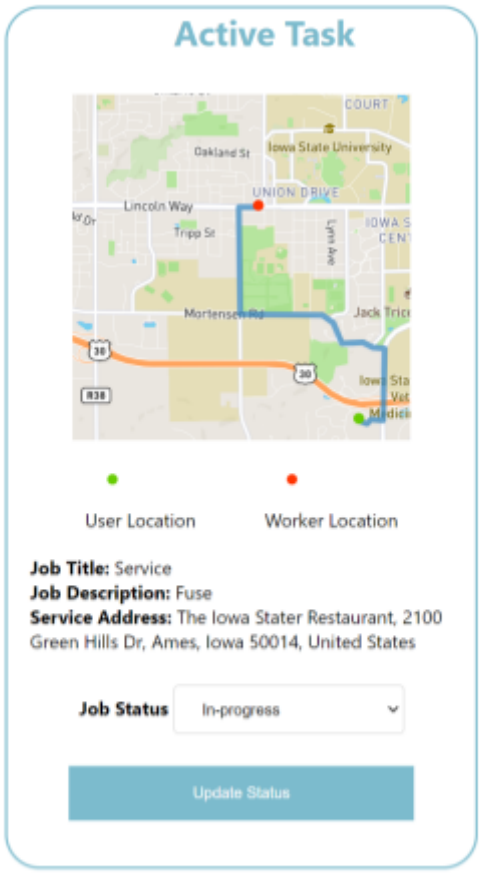

Figure 11: Worker Side Task View

The worker's dashboard is very similar to the Client's, but has additional management functions such as clocking in and assigned tasks. The worker can use the clock in/out function to signify whether or not he is ready to be assigned to a task, and the active task tab displays the information about the assigned task, along with a route to get to the location.

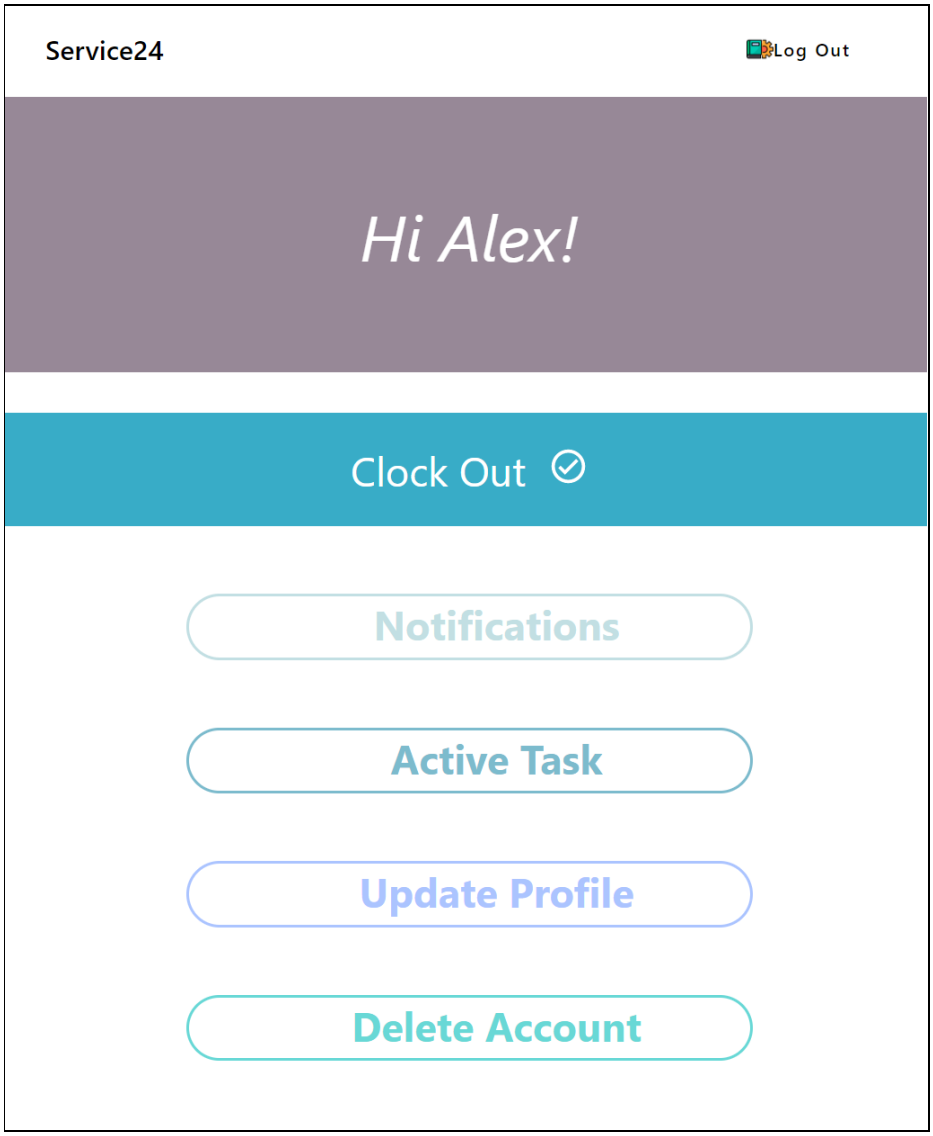

Figure 12: Worker Dashboard