Peer-to-Peer Variable Service Transaction

Alex Zummo and Sonia Aggarwal

May 7, 2021

Capstone Research CS 4980

Table of Contents

Abstract

Peer-to-Peer Variable Service Transaction (P2P-VST) systems make it simple for people to give their time to others in exchange for the same courtesy later. The problem is that such services are not easily found. Thus, an application for users to exchange their time, called Hamlet, was proposed. This application will be developed by "evaluating how recommending service requests targeted at a person's context impacts their willingness to enter a transaction" [1]. Preliminary research has shown that even if people have not actively volunteered for a service, they are influenced by convenience.

Introduction

This research project will build a mobile application that uses context-aware matching algorithms to enable community members to interact with one another for the purpose of fulfilling each other's service needs. What is context-aware matching? To be "context-aware, a system or application should adapt its behavior according to [the] current context, changing over time" [2]. Current context is often acquired by sensors or other applications, in this case a mobile application. The algorithms used will be built with emphasis placed on timely and context aware notifications and matches found by the current context. The goal is to enhance community camaraderie and human efficiency by increasing instances when people may have specific needs fulfilled on a casual basis by other residents of the same community. The motivation of this research began with the idea of time banking. Time banking and the sharing economy became popular when the conventional economy failed to provide the means for many to earn income and acquire resources, including services such as Uber, Airbnb, and more. This project will allow specific communities to be able to interact with one another and exchange their personal time, as best determined by the community.

Detailed Technical Description (i.e. Method and Implementation)

User Interface

The user interface of Hamlet is designed first using Figma as a way to map out the plans that will be implemented in code. The implementation used the React Native Javascript framework with flexbox CSS.

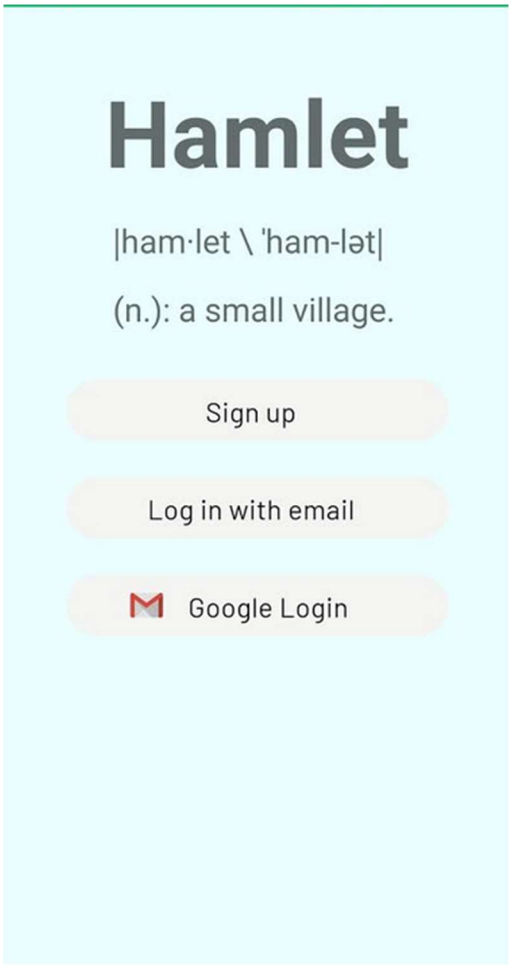

Figure 1: App.js implementation/Landing Page

To see the complete flow, the Software Flow Diagram is shown in Figure 8. To begin, upon opening the application, there is a splash screen which will show for approximately 10 seconds. This was a design choice to allow for all node modules (from the React Native framework) to load and make any necessary installs before the application is in use, as well as establish connection to backend servers. Once this splash screen disappears, the user will see Figure 1, the landing page, which shows that there are three ways to progress through the application.

The first option is to Login with Gmail, which will pre-populate all identification fields based on your account information. After all this information is collected, the user will be redirected to the Home Screen. The first way to progress through the application is "Sign up." This screen is shown in Figure 1, where the user is able to enter their desired credentials (email, and password), check that the user is 13 or older for research purposes, and their name that they would like to use publicly. Here the user will also have the option to note if they have a car, which, once indicated to be true, will be used to help match certain needs and skills together. For example, someone needs help with moving a couch. Therefore, someone with a car would be very helpful. Once the user clicks submit, they will be prompted to enter their skills, as shown in Figure 2. These skills can range anywhere from tutoring and delivering food, to grabbing groceries and assisting a move.

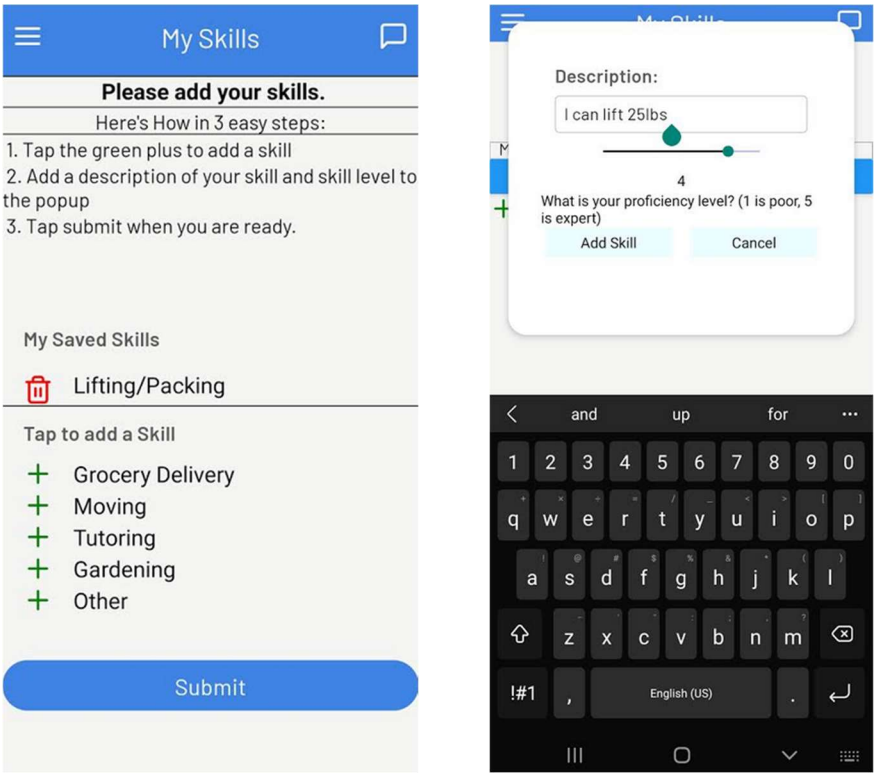

Figure 2: My Skills Page

After the user submits their skills, they will then be redirected to the My Needs pages shown in Figure 3. This figure shows where users will indicate their needs for delivery, tutoring, etc. In each screen the user will be asked to rate their need or skill level from one to five, where five is a high priority need or expert level skill and one is a low priority need or poor skill level.

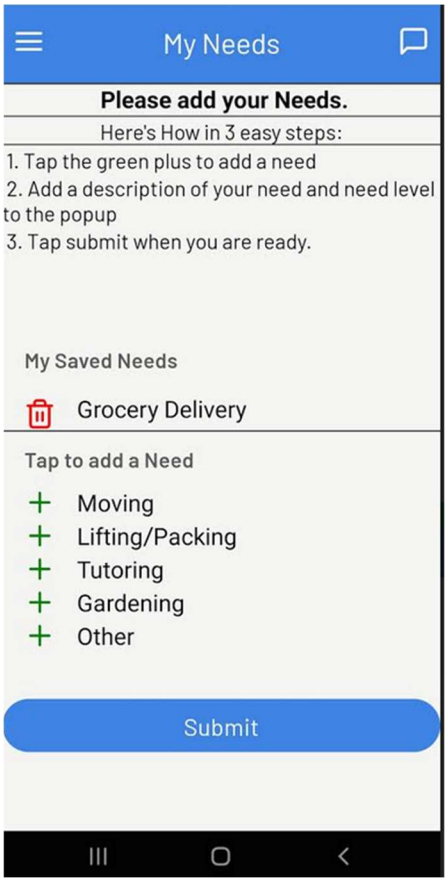

Figure 3: My Needs Page

After this, the user will click the submit button and then be redirected to the Home Screen, as shown in Figure 4. The home screen will have two main components, an interactive map and a search bar. The map will ask for permission to access the user's current location and then have a marker at the current location of the logged in user. This map will also render pins for positing needs that are within a ten mile radius of the user. These posting pins will refresh based on a few factors when the user logs in and when the user navigates to this page from any other page in the application. The pins will change if: a user creates a posting within ten miles of the logged in user, the logged in user creates a posting, the expiration date of some of the pins are past the current time therefore deleting the pin, and lastly if a user accepts a posting request.

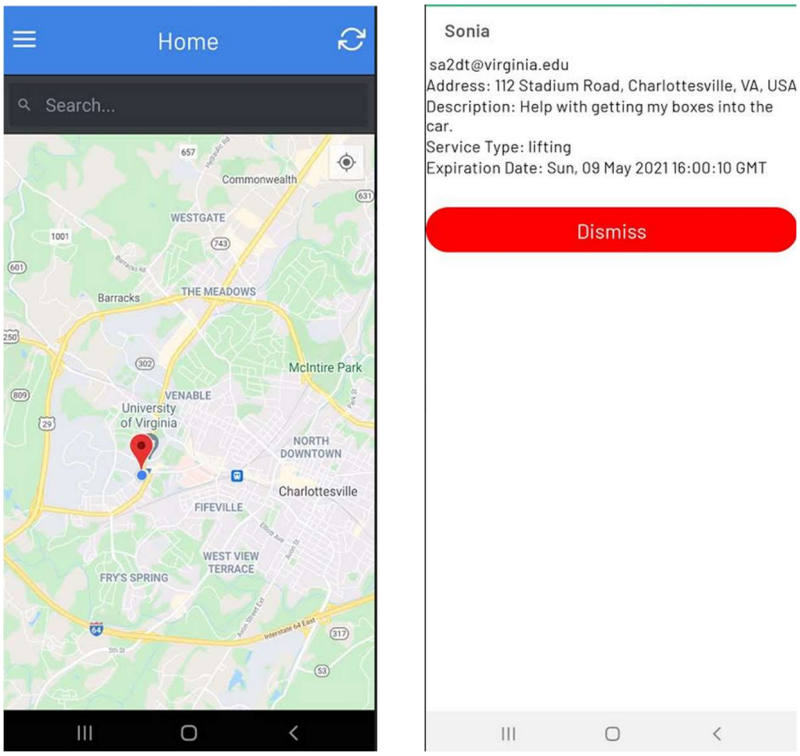

Figure 4: Home Screen

The home screen also has a search bar that will search through the descriptions and addresses of the postings such that if the word searched is found in any posting, these postings will show up below the search bar indicating that "matches" to your search have been found. A resulting "match" can then be clicked on and will display a modal. This modal can be seen in Figure 4 (right). The modal shows the contact information of the poster (or recipient of the post), and details regarding the request such as location, description, and expiration date.

There are two options that can be done by the user who did not create the post, from this modal screen. The first is dismiss the modal and return the home screen. This acts as a "back button." The second is accepting the posting. If this posting is accepted, the user who accepted (or provider of the post) will be redirected to the My Postings screen that will now include this posting. This action will also prompt a push notification to be sent to the recipient that will include the name, email, and request service type, such that the recipient will need to accept this

provider to their request. If the recipient accepts the provider, then the provider will have a match show up in their My Postings page. However, if the user declines, then the handshake action is incomplete and the provider will not be able to see this posting in their My Postings page.

On each screen, there will be a navigation bar that can be found after swiping from the left, shown in Figure 7 toward the middle, as shown in Figure 5: Navigation Bar.

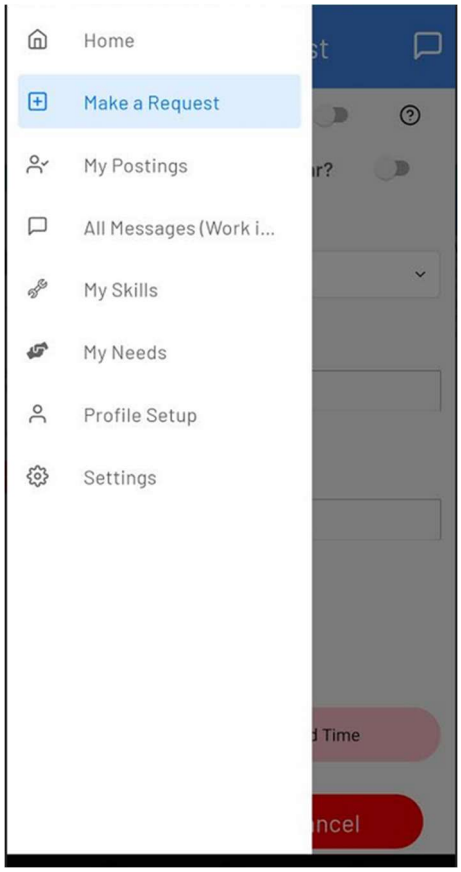

Figure 5: Navigation Bar

The navigation bar shows the Make a Request page. From this page, as shown in Figure 6: Make a Request Page, the user is able to create a request for a tutor, someone to clean/deliver, etc. The required fields for a successful posting will be shown, and marked with an asterisk. If a field is not filled out, the user will not be able to continue with the posting. The user, once posted, can then find this posting on the home screen as a pin in the desired location. This

location will be entered by the user and validated using geolocation. Once the posting has been submitted, all users within a 10 mile radius of the posting address will be notified through a push notification that a request has been made and that they are able to fulfil it if desired. On this same page, if the service type includes grocery delivery or moving, then the user will be given the option to enter a source address and a destination address for the provider's ease of use.

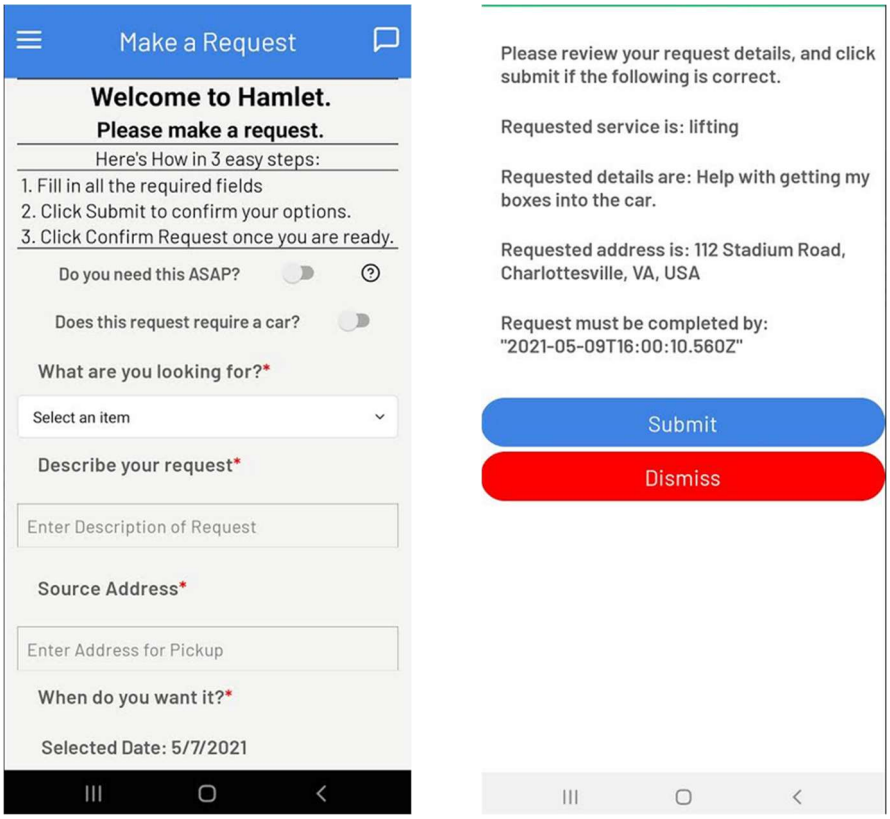

Figure 6: Make a Request Page

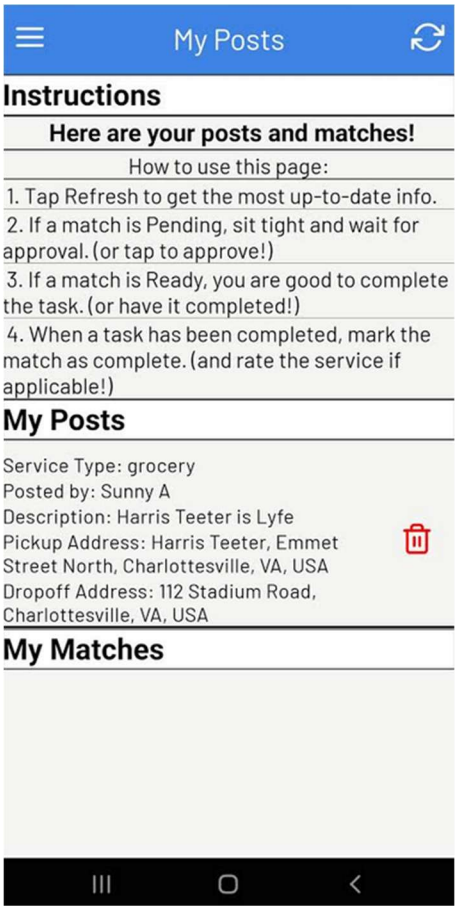

Figure 7: My Posts Page

The My Posts page shows all the posts the user has created and have matched with. When a match is confirmed, both users involved will have a posting show under “My Matches” that will give the recipient the option to confirm the completion of the task, and then rate their provider. Ratings are done using stars, with 1 star representing a poor match and 5 stars representing an excellent match.

Design Flow and System Architecture

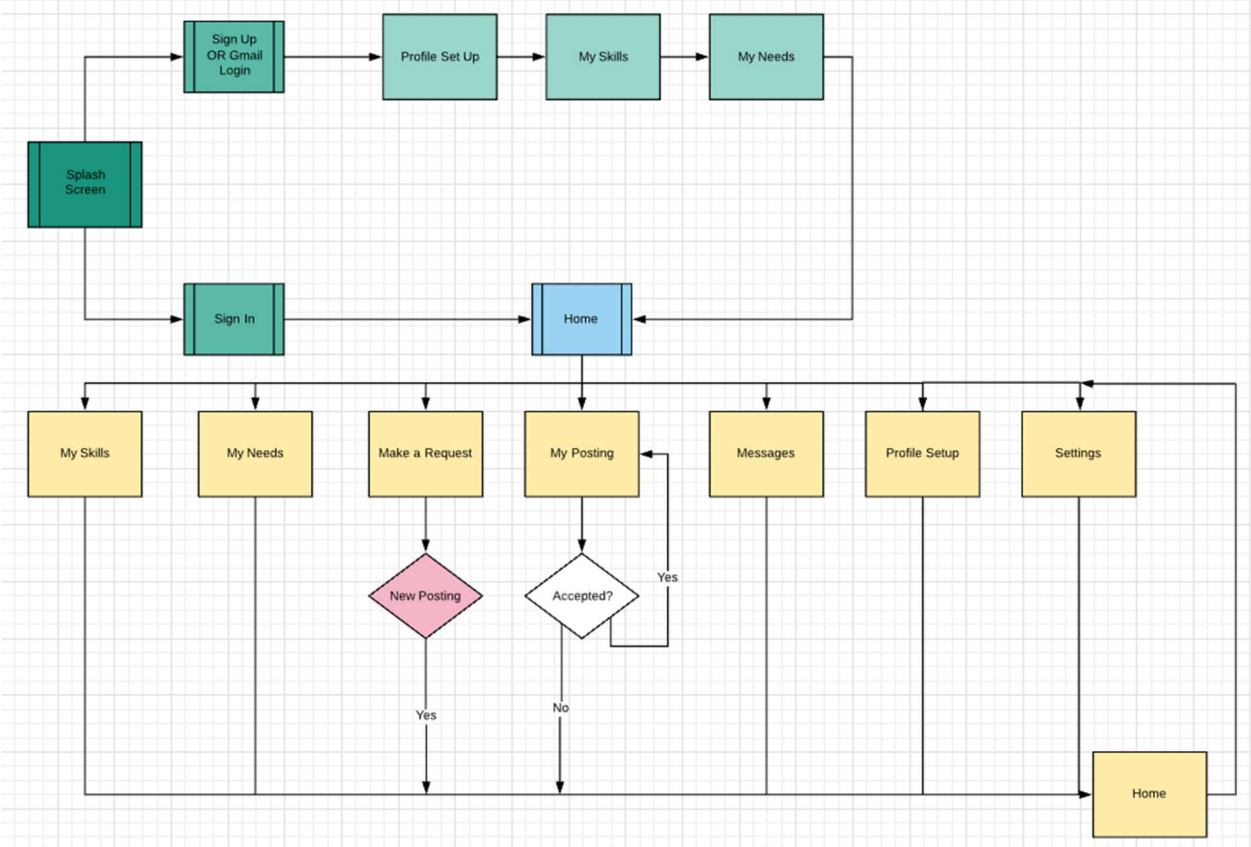

Figure 8: Software Flow Diagram

In Figure 8: Software Flow Diagram, the software or user flow diagram is shown. Users have two main entries into the application from the splash screen, sign up or login. Sign up will lead to creating a profile, entering the user skills and then user needs before directing to Home. Login will direct the user to Home immediately. From home, the user can navigate to any part of the application and update their preferences, needs, skills, and requests as necessary by navigating to the appropriate tab.

Database Design, Implementation, and Population

The backend of the system is implemented using Flask with a SQLite database, using the SQLAlchemy library. In Figure 9, the database relationships are shown. The database will

populate the tables and rows based on the user, and only save certain information for the current user such as the profile information, authentication information, and posting information.

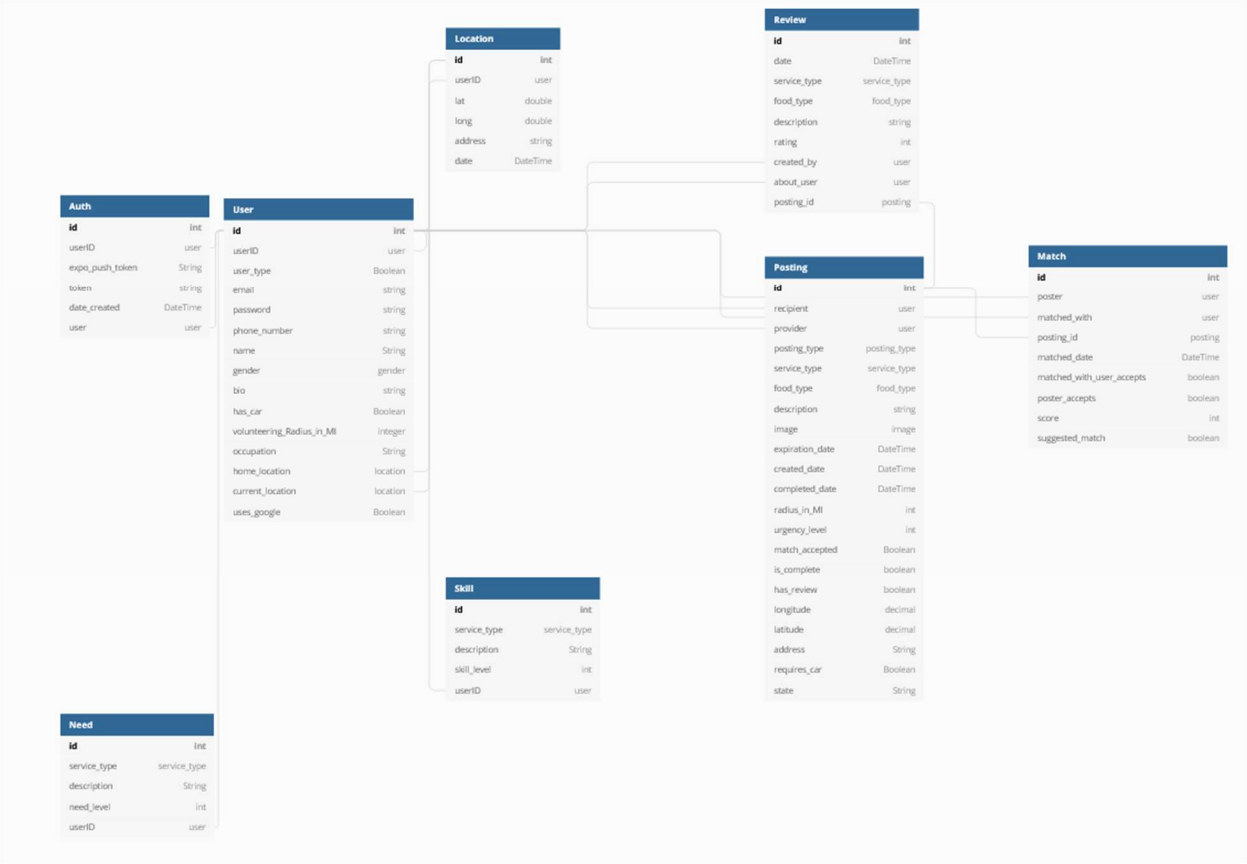

Figure 9: Database Tables

Matching Algorithm

The matching algorithm consists of three main ways that users can find each other to provide a service or request a need. One, when a user creates or updates a need or skill, each user with a complementary need or skill will be notified with a push notification that there is someone who is able to help. For example, if a user needs tutoring in python and another user has a python programming skill, once both of these facts are placed into the database, a “match” is made and one of the users will get a push notification with the other user’s contact information. The recipient of the notification is decided simply by who updated the database first. Second, when a

user creates a post or request, the user of the application within a ten mile radius will receive a notification that a request has been made and will be given the option to look at the posting upon tap. And lastly, a match is made if a user creates a post, another user accepts to provide for the post, and the creator of the post receives and accepts the post provider.

API work

The project employs a REST API. A RESTful API "determines how the API looks like." It stands for "Representational State Transfer". It is a set of rules that developers follow when they create their API (Zell, 2018). Every request sent from the front end React Native framework, which includes the Sign Up, My Needs, My Skills, Home, Make a Request, and My Postings screens, must go through an API request to redirect the user to the correct action item, based on their session cookie, user id, and their profile status. Each api request is different. The HTTP methods used in this project are GET, POST, and DELETE requests. Currently there are several API requests shown in Appendix B. For more information on the specific APIs employed in the application, please see Appendix B.

AWS Hosting

The application will be hosted on an Ubuntu 18.04 server virtual machine on the Amazon Web Services (AWS) Elastic Cloud Compute (EC2) service. The current IP address that this application is hosted on is: http://54.237.126.87/ which is what the EC2 instance of AWS has directed the application to. The backend Flask server was then built as an NGINX (pronounced "Engine X") service. This service gave the developers two abilities. First, a way to run the backend in the background on the VM without having to physically run a command every time the app is launched. Second, the NGINX service specifically was built to create Flask applications as services, which made development and deployment simple. To create the EC2

instance and run flask applications as a service using nginx, see Appendix C for detailed documentation.

Next Steps

Major Feature Additions

The future work of this app would include adding a more sophisticated matching algorithm to track users for their frequently visited locations and times they are in these places. The app could then predict when a user would be in the area and if they would be willing to fulfil a request. Further, an in-app chat feature would allow users to communicate about a request directly. The chat functionality would create an entirely in-house experience and ease user interaction.

Small Feature Fixes

Some things already in the app could clearly be improved in future iterations, such as the user experience, automatic refresh, and an admin interface. The user experience could be improved if there was a graphics designer who was also working on the application alongside the developers. The automatic refresh feature would use a server push to service interface, such that whenever an element changes (such as a posting is added to the Home Screen) the application would refresh the screen automatically. The user would then be able to interact with the app as usual.

Code Logistics

The codebase itself could also be improved with automated testing using Jest, which is React Native's unit testing library. These Jest tests could be developer-written as code is developed and executed in an automated build sequence. This automated build sequence would

be integrated with the github repository such that every time code was pushed to the repository, automated regression testing would occur and flag issues before a merge is executed.

References

A. B. Kocaballı and A. Koçyiğit, "Granular best match algorithm for context-aware computing systems," *Journal of Systems and Software*, 15-Mar-2007.

Doryab, Afsaneh, et al. "If It's Convenient." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 3, 2017, pp. 1–28., doi:10.1145/3130913.

Zell, Z. (2018, January 17). *Understanding And Using REST APIs*. Smashing Magazine. https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/.

Appendix

Appendix A: Github Repository: https://github.com/ihcc-lab/nightingale

Appendix B: All APIs in detail at: https://docs.google.com/spreadsheets/d/1byTrp842ym9KOFTL08OqPz9JPvRSa02kVmqFESeQEuE/edit#gid=0

Appendix C: https://docs.google.com/document/d/1QecbwOW4z9DIqGsseVEp5yzO3mkCxTZUlMQ5ybkC5nI/edit?usp=sharing

Appendix D: https://docs.google.com/document/d/1HHiOqZatARgK1KITfT7OneqHZCRJr1VjCaMj_yR-L2c/edit?usp=sharing

Appendix E: https://docs.google.com/document/d/1x7Si6bZzm0xSaqhuU3ten7nIYfSvUYCLpYhOjN9-w1o/edit?usp=sharing