



# Ryde

## Design Document

Wen Cao, Tyler Carberry, Benny Chen,  
Benny Liang, Michael Matthews, Tapan Soni,  
John Stranahan, Nicholas La Sala

12/18/18

Senior Project Fall 2018

## Table of Contents

High Level	
Design.....	3
Problem Solving Approaches.....	4
Screens.....	5
Screen Navigation.....	14
Flow Diagrams.....	16
Technology Stack.....	18
Backend.....	19
Authentication and	
Security.....	20
Input and Output.....	21
Database Schema.....	22
Restful Endpoints.....	23

## High Level Description

The purpose of the application is to give users a way to easily find other commuters and organize carpools to and from Rowan University by matching them with other students who also go to Rowan University and are looking for a carpool.

When the user creates an account they will then enter their general weekly schedule along with an address that will be used to determine their location relative to Rowan. Once the app has this information stored, the user can then request a search of other users that are also looking to organize carpools. The app will match people based on how far they are from the user, and from similar schedules. The app will also add the new user to the pool of users that can be found as a potential match.

Once the user has found other students that are looking for carpools they can then enter into a chat with them and work out any necessary details. This includes pick up/drop off times, along with what kind of car they should expect to pick them up.

## **Problem Solving Approaches**

The biggest problem facing the development of the app was how the user's schedule was to be handled. The two approaches that were considered were a simple arriving and leaving time for each day of the week, or allowing the user to create blocks of critical times they would need rides that they could choose to search for. Ultimately the simpler arrival and departure for each day was selected because it would provide users the largest number of potential matches possible, which is invaluable to an app with an inherently low maximum number of users. This would also allow for each user's schedule to be returned in a higher number of searches which would improve overall visibility for the profile.

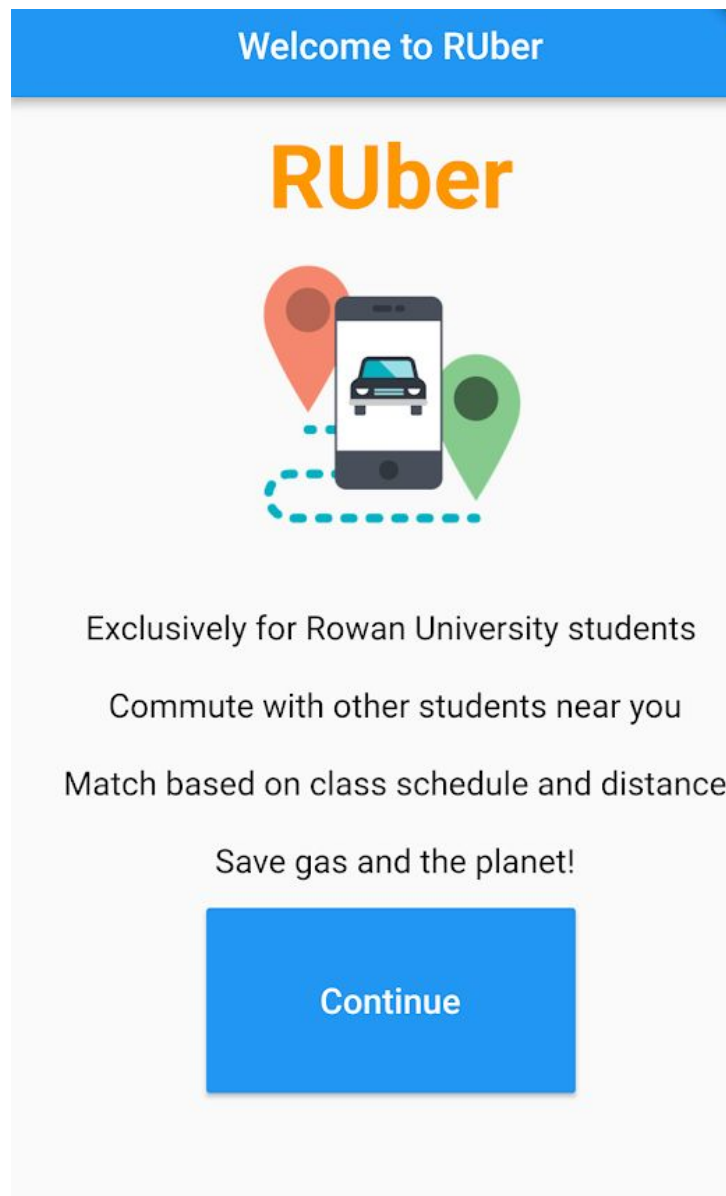
Another issue was how the user would remain authenticated while using the app and navigate from page to page. There was conflict about what would be used to ensure the uniqueness of a user, whether to use their firebase id, token, or their mysql database id. Ultimately it was decided to make their email persistent through the app because it could be used to pull information from all sources as needed.

Over the course of the project the map screen was scrapped from the final design, because of privacy concerns and it complicating using the app. The functionality was completed however the screen can no longer be accessed.

## Screens

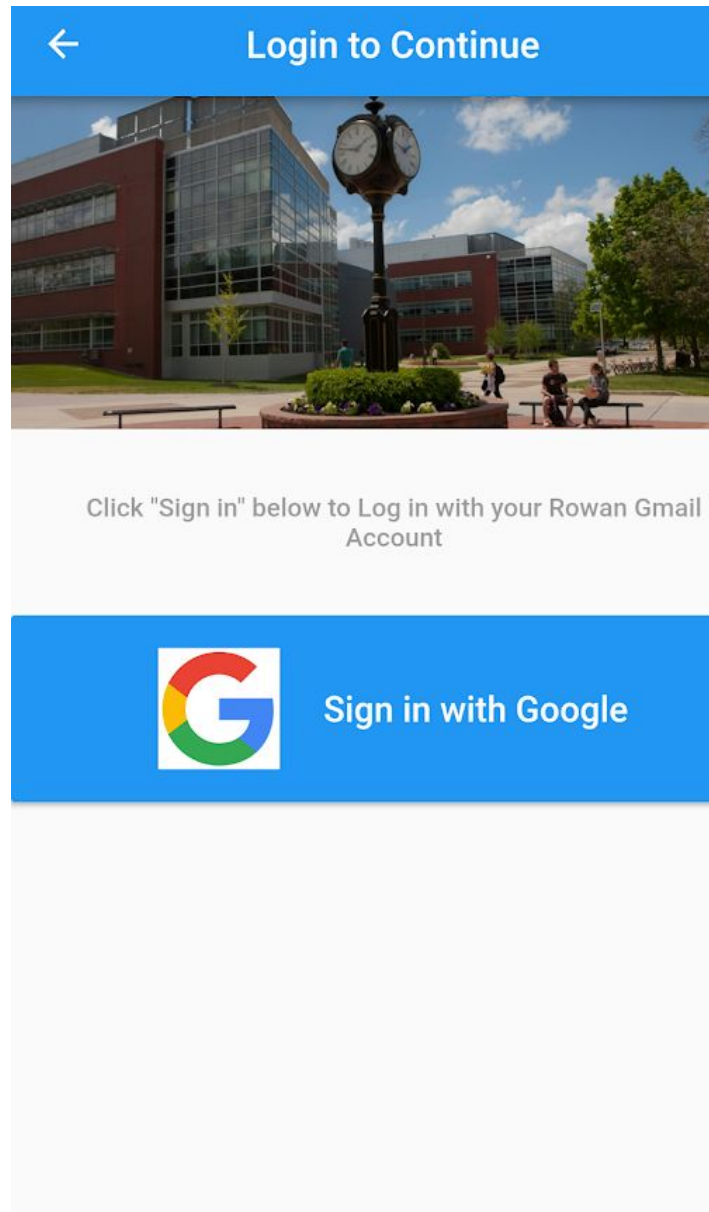
### Start Screen

Whenever someone opens the app they will start at this screen whether or not they have used the app prior. This screen features a quick description of the app, and a button that will redirect the user to the home screen and login screen depending on if the user has used the app before.



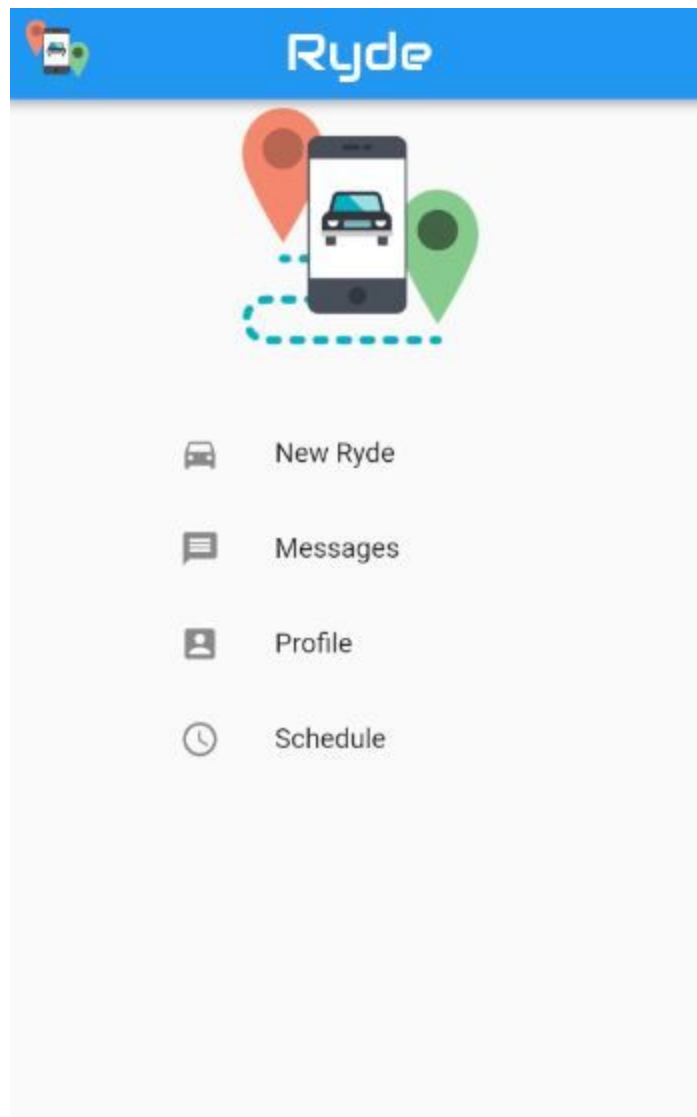
## Login Screen

The Login Screen will be the first screen that users who are not already logged in will see when they start the app. Here the users will enter their Rowan login credentials before being redirected to the home screen. If the user launches the app and is already logged in then this screen will be skipped.



## Home Screen

This is the primary screen available to the user, and it features a menu to navigate the app. In the final version of the app it will feature buttons to access the new ride, messages screen, profile edit screen, and setting.



## Edit Schedule

This page allows the user to edit their schedule that they would be using to match with other users. This screen appears when the user first creates their account, and can also be accessed from the app at anytime. For each day the user can enter both an arriving and leaving range of times, that will be used by the matching algorithm to determine compatibility. The user only needs to enter time for the days that they are looking for carpool, days that are left blank are not included when matches are returned to them.

The screenshot shows a mobile application interface titled "Edit Schedule". At the top, there is a blue header bar with a hamburger menu icon on the left and the title "Edit Schedule" in the center. Below the header, a green instruction text reads: "Click on the drop down menu to enter your schedule". The main content area consists of five rows, each representing a day of the week from Monday to Friday. Each row has a day name at the top (Monday in orange, Tuesday in pink, Wednesday in blue, Thursday in green, and Friday in dark blue). Below the day name, there are two labels: "Arrive between" and "Leave between". Under each label are two dropdown menus, each with a downward-pointing arrow. At the bottom of the form, there is a grey "Submit" button.

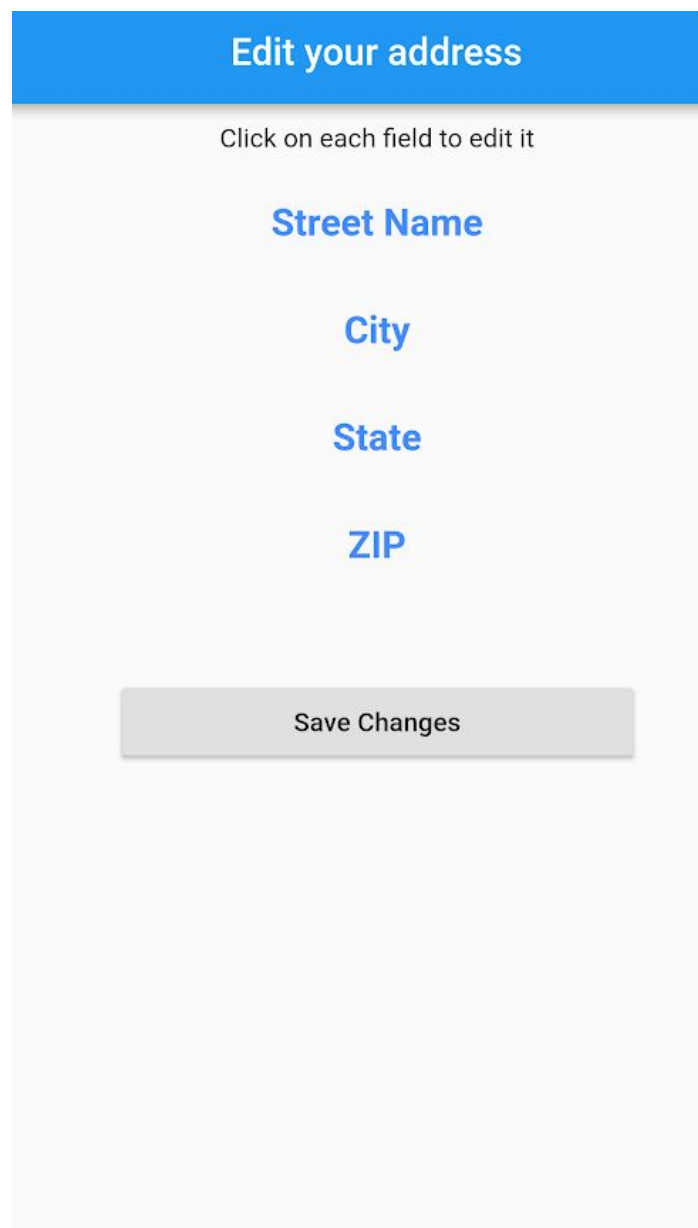
Day	Arrive between	Leave between
Monday	<input type="text"/>	<input type="text"/>
Tuesday	<input type="text"/>	<input type="text"/>
Wednesday	<input type="text"/>	<input type="text"/>
Thursday	<input type="text"/>	<input type="text"/>
Friday	<input type="text"/>	<input type="text"/>

Submit



## Edit Address

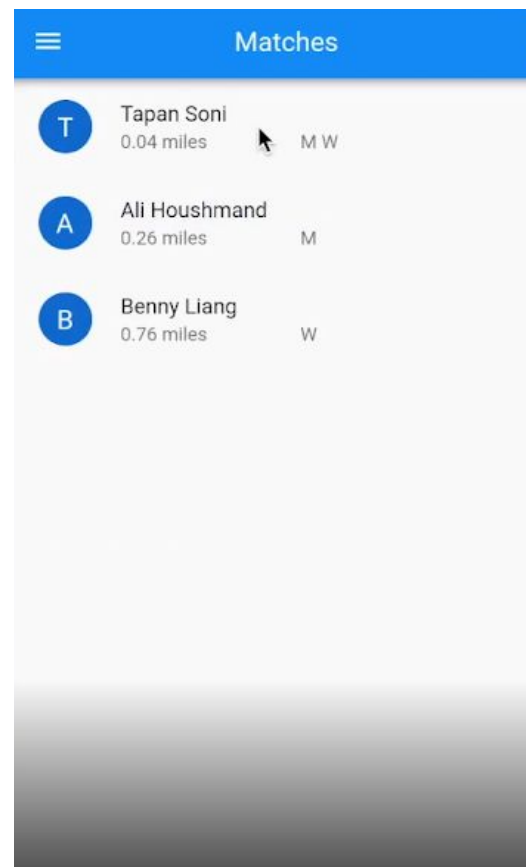
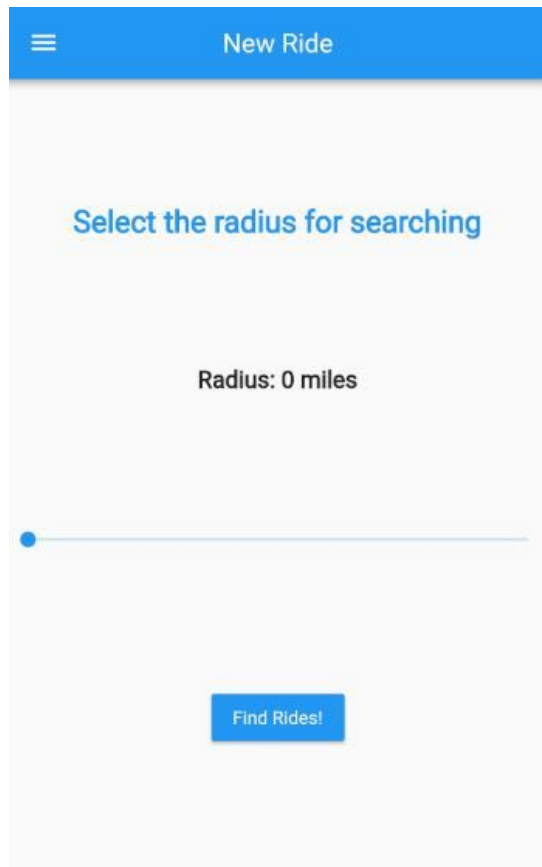
This screen allows the user to edit their address that will be used by the matching algorithm to sort matches by distance, and filter out users that are too far away. The user must complete the fields of this screen when they are first creating an account, and can edit this information if they need to later on. The state field must be 2 characters long, and the zip code field must be 5 numbers long for the submit button to be pressed. After the user hits submit their new address will be stored and used to find future matches.



The image shows a mobile application screen titled "Edit your address". At the top is a blue header bar with the title in white. Below the header, the text "Click on each field to edit it" is displayed. There are four blue, clickable text labels arranged vertically: "Street Name", "City", "State", and "ZIP". At the bottom of the screen is a grey rectangular button with the text "Save Changes".

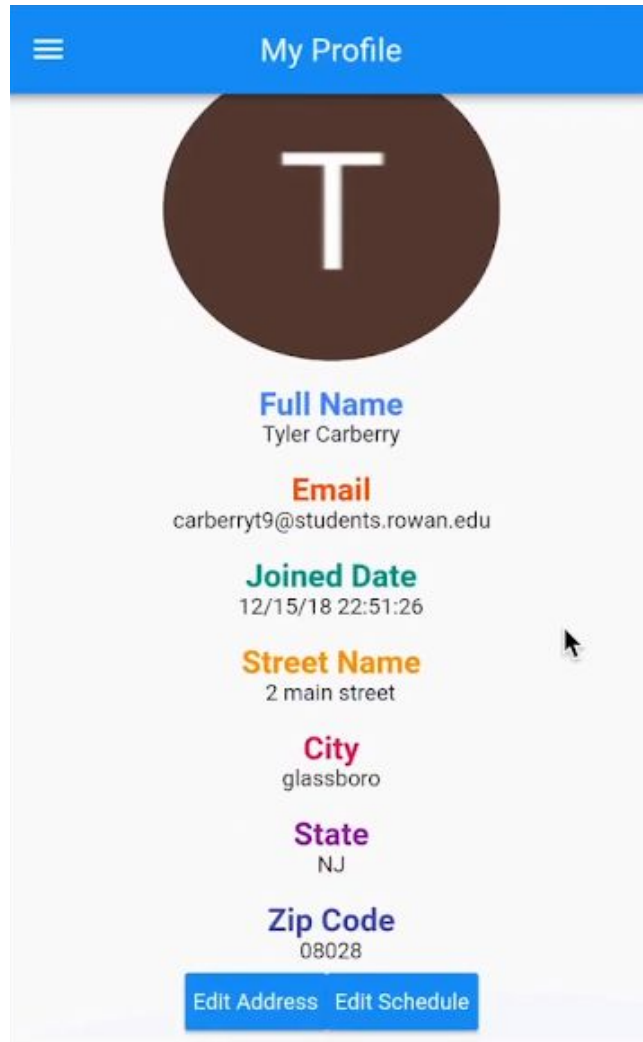
## New Ride Screen

These screens allow the users to search for other users that they can begin to carpool with. First they select a range of distance, and then the app will display every user within that range sorted by distance with at least one day the two have compatible schedules. If the user taps on one of the people listed a pop displaying more information will be displayed, and the user will have the option to form a chatroom with them.



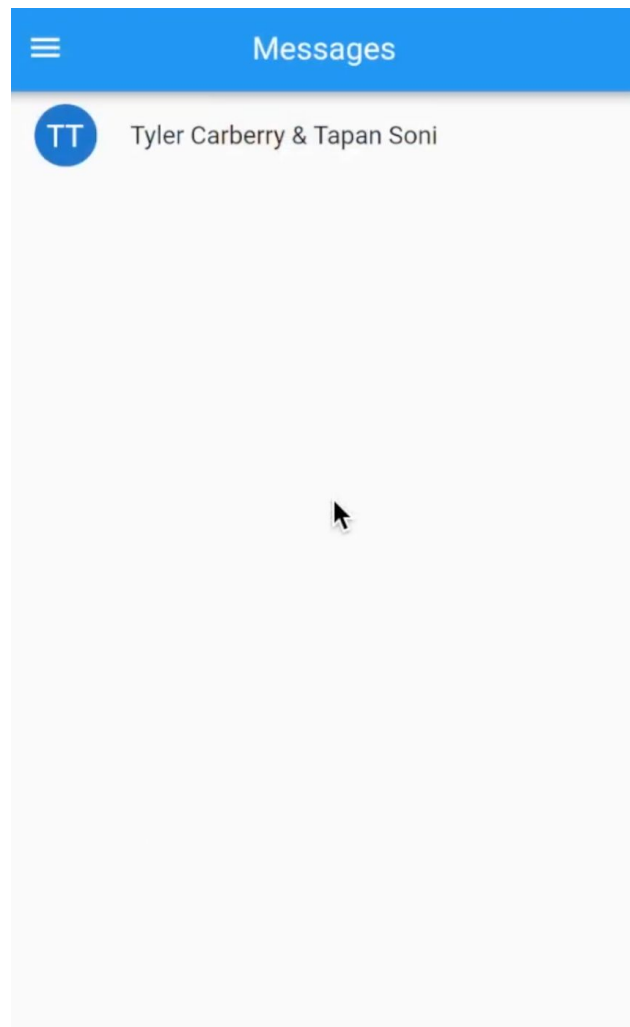
## Profile View

This screen allows the user to review their information, and see what is being displayed to other users.



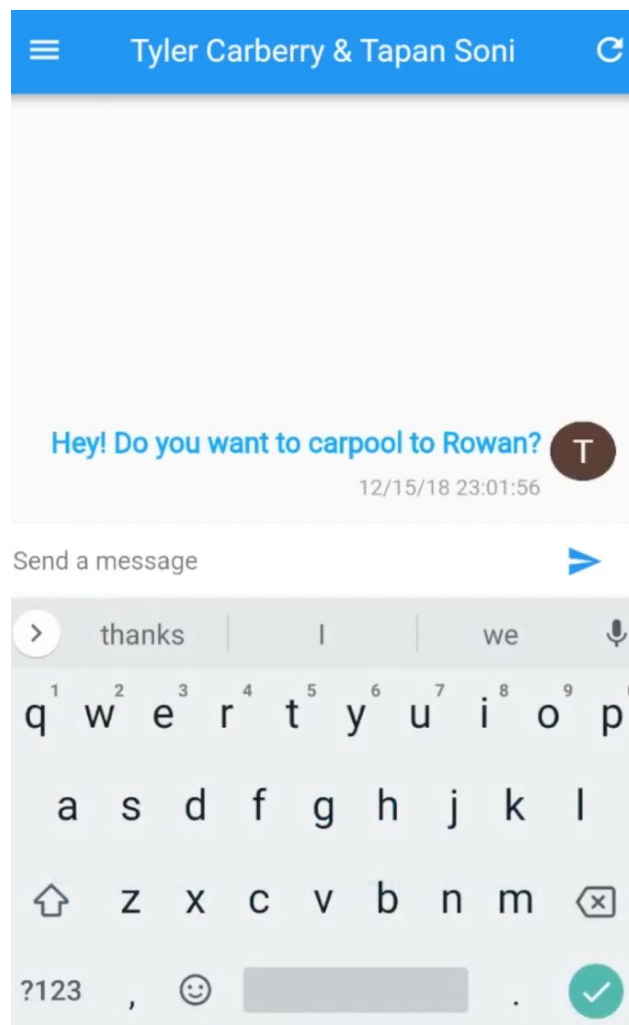
## Messenger Screen

This screen will allow the user to view all of their chats that they started and open the chat screen with whomever they select. The chat will be ordered from most recently used.



## Chat Screen

This screen allows users will be able to send and receive messages with the other users that they match with. The user's messages will be displayed on the right side of the screen will the receiving messages will be on the left side of the screen. If the user taps on the text bar, a keyboard will appear and allow the user to send a message.



## Screen Navigation

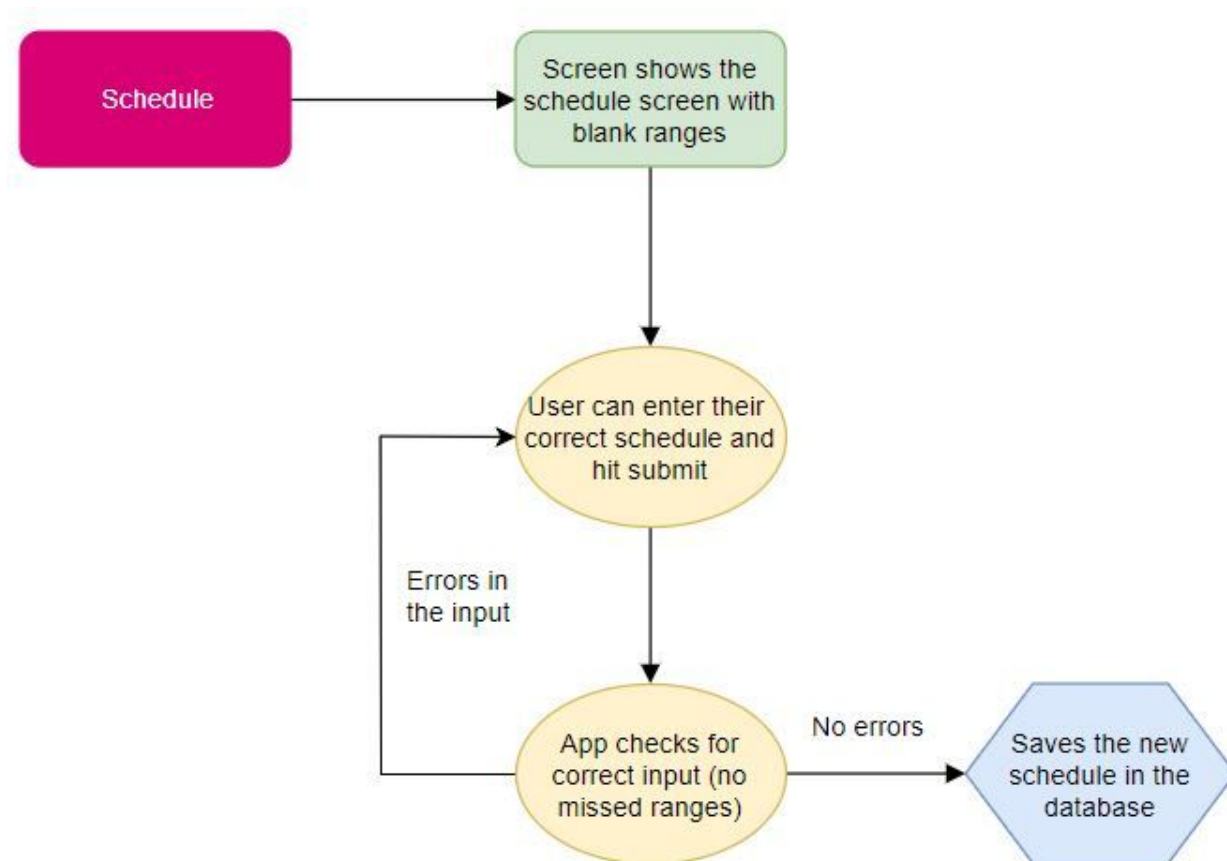
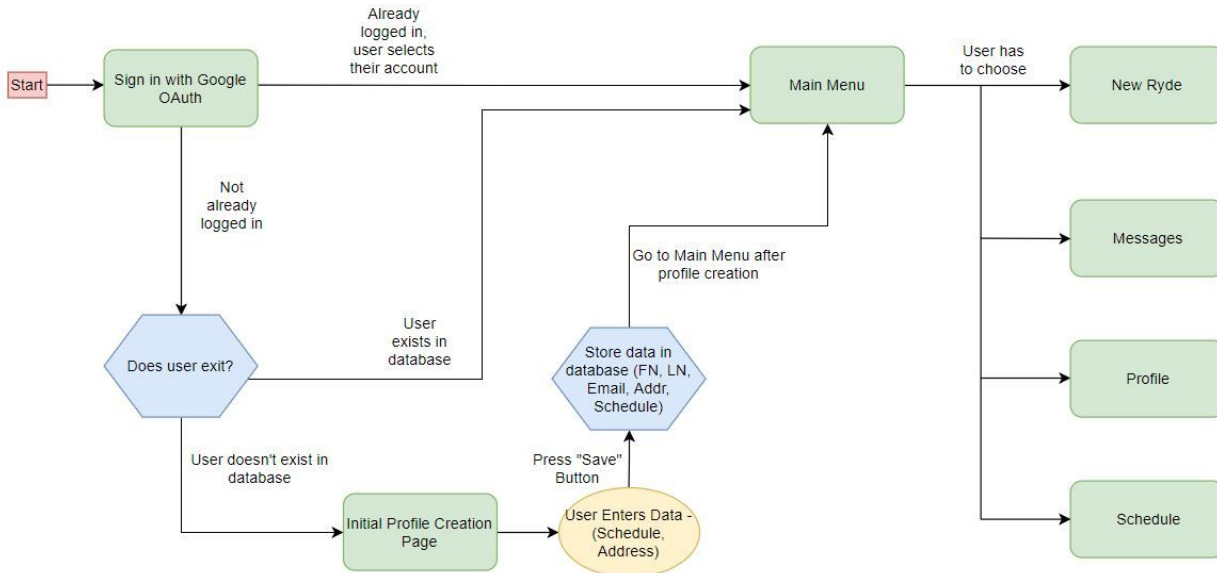
When a new user opens the app for the first time they be presented a login screen. The users login using their rowan account credentials, and from there they are redirected to a address edit screen. Following this screen the user will be redirected to the edit schedule screen. After the user has set up their profile for the first time they will be redirected to the home screen which will be the default screen on future app launches. The home screen will act as a hub that allows the user to access any of the functions that the app provides them.

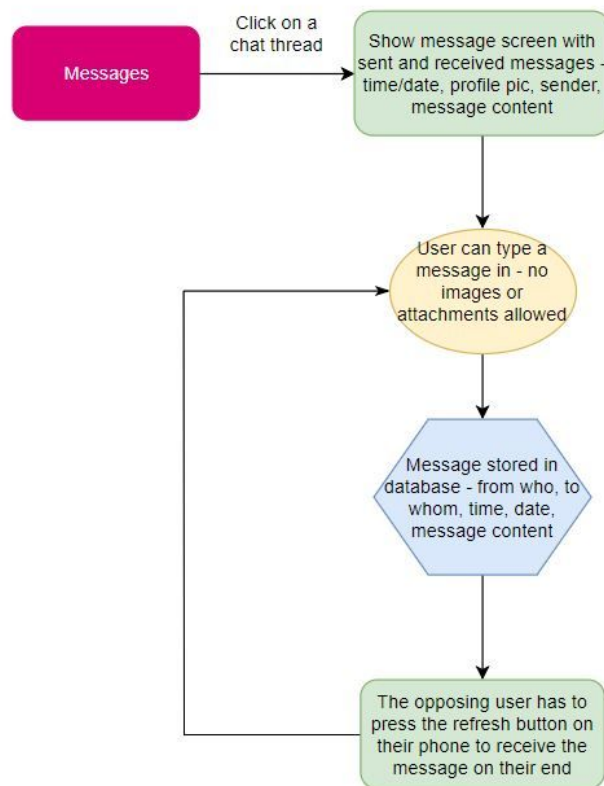
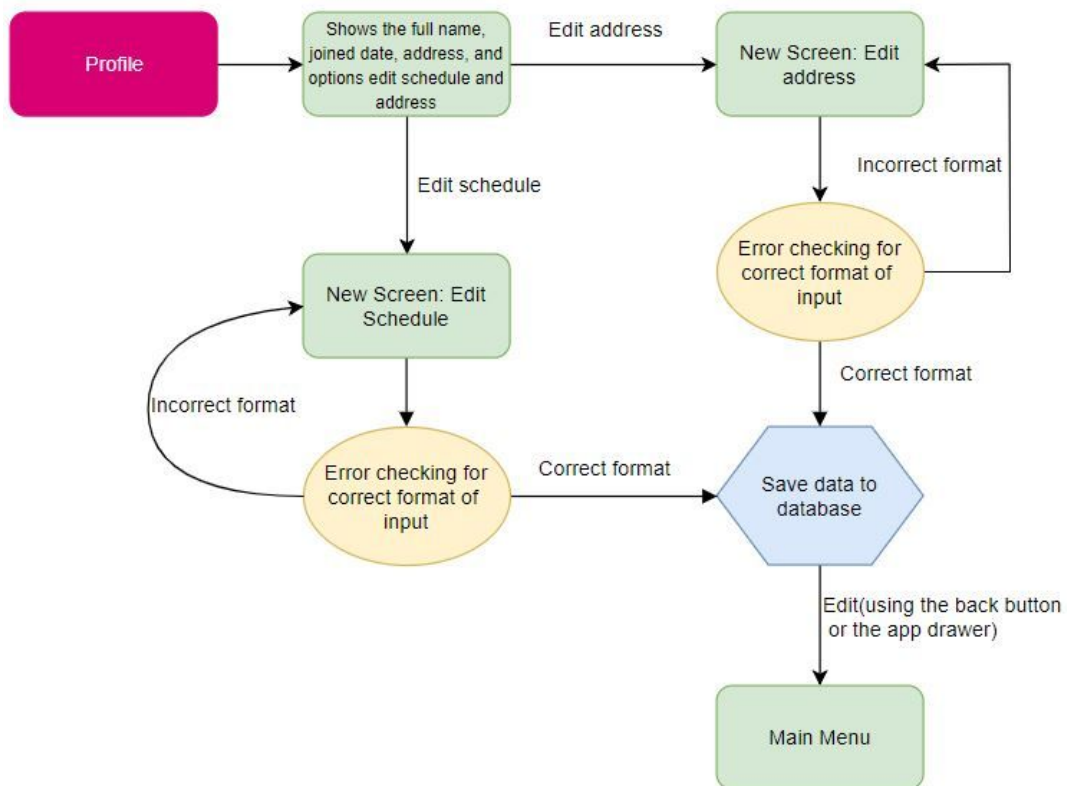
Since this is a new user that just adjusted their settings, the next thing they would likely do is begin searching for people to form a carpool with. The user would select the “New Ryde” option from the main menu and then select a radius to search within. After that, the app brings them to a screen that list all of the students that would be a potential match for the user. The user can send ride requests to whomever they think would be a good match from the students that are returned to them. Sending a ride request creates a new chat thread between the two users and the initiator can send a message to the receiver.

Users can select the Messages button from the home screen that brings them to a screen that shows them all of their active chat threads. Once a user selects a chat that they want to post in the app will redirect to the chat page where the user can type a message along with read old ones.

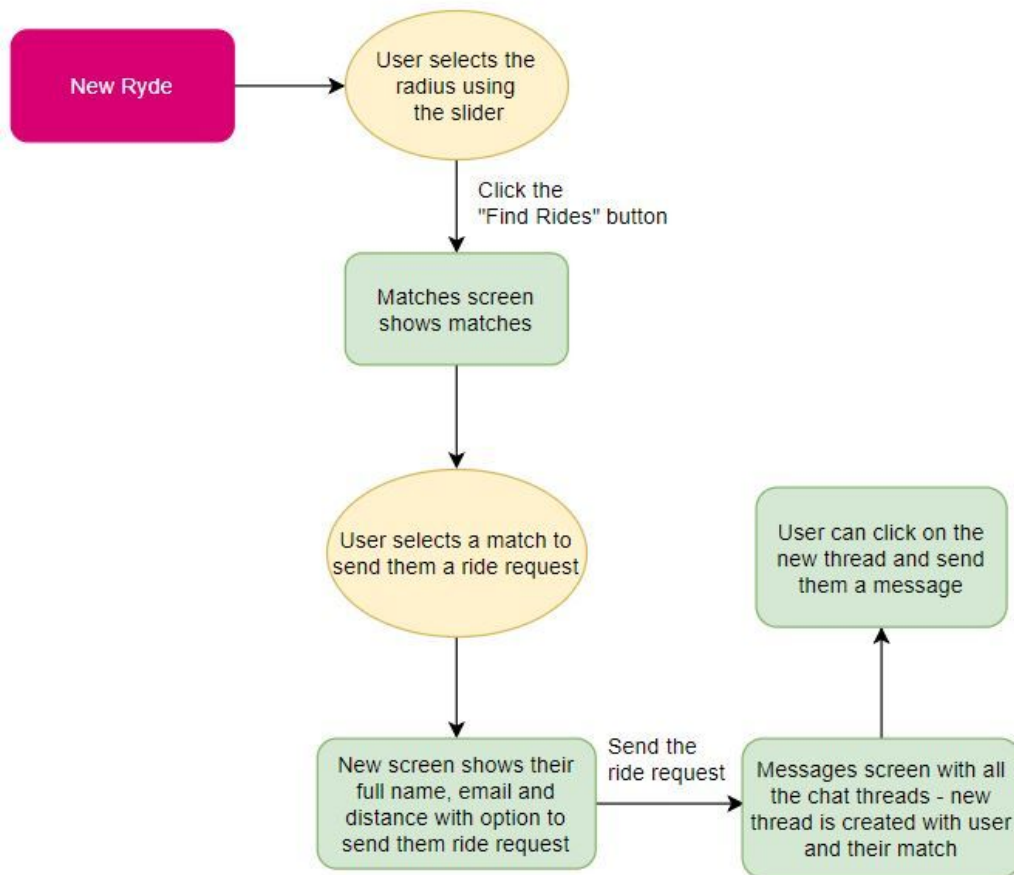
Other pages that can be selected from the home screen are the profile screen and the schedule screen. The profile edit screen is largely the same as when the user first entered information when their profile was first created. The user can choose to revisit this screen to adjust their profile whenever needed. The profile screen offers the user options to edit their schedule and address. The last option on the main menu screen is the “Schedule” button which allows the user to manage their schedule. The edit schedule button in the “Profile” option leads to the same screen as the “Schedule” screen.

## Flow Diagram









## Technology Stack

The frontend for the app is being developed in Android Studio. The user interface is developed using Dart and the Flutter SDK from Google. Since the app is intended to run on android phones, the app is test using emulated hardware using the latest version of Android.

The app uses RESTful APIs such as Google Maps to determine whether two users are an appropriate match based on their distance. Google Maps is also used to provide the user with a visual of what to expect from an upcoming route. We utilize HTTP requests to pull info from the databases to compares users schedules to see if they are suitable.

The application has a Java-based backend that utilizes the Spring Boot framework. We chose Java because of its robustness and the development team's familiarity with the language. Spring Boot was chosen due to its dependency injection capabilities and integration with Hibernate.

The database for the application is hosted on an AWS server. We decided to use MySQL as our database over other alternatives (Apache Cassandra, MongoDB, etc.) due to its simplicity and our expected user count. If the number of users increase to an amount where a relational database is not desirable, we plan to migrate to Cassandra (which is known for scalability). The database schema includes tables for profiles, addresses, schedules, groups and messages. The app uses Hibernate as an ORM and to communicate with MySQL.

## **Backend Information**

The backend is created using Spring Boot and will handle the flow of data that is required to use the app. The backend will communicate with the app's database on AWS, and be responsible for querying the database for information that is presented to the user on the front end. It will also need to take information that the user enters into the GUI and store it in the appropriate table on the database.

The backend will also handle the user matching that is integral to the functionality of the app. Any calculations that are performed by the app to determine which users would be strong carpool candidates will be performed from the back end.

The backend also handles the messaging component of the app and is responsible for displaying a list of people that the user has formed chat rooms with.

## **Authentication and Security**

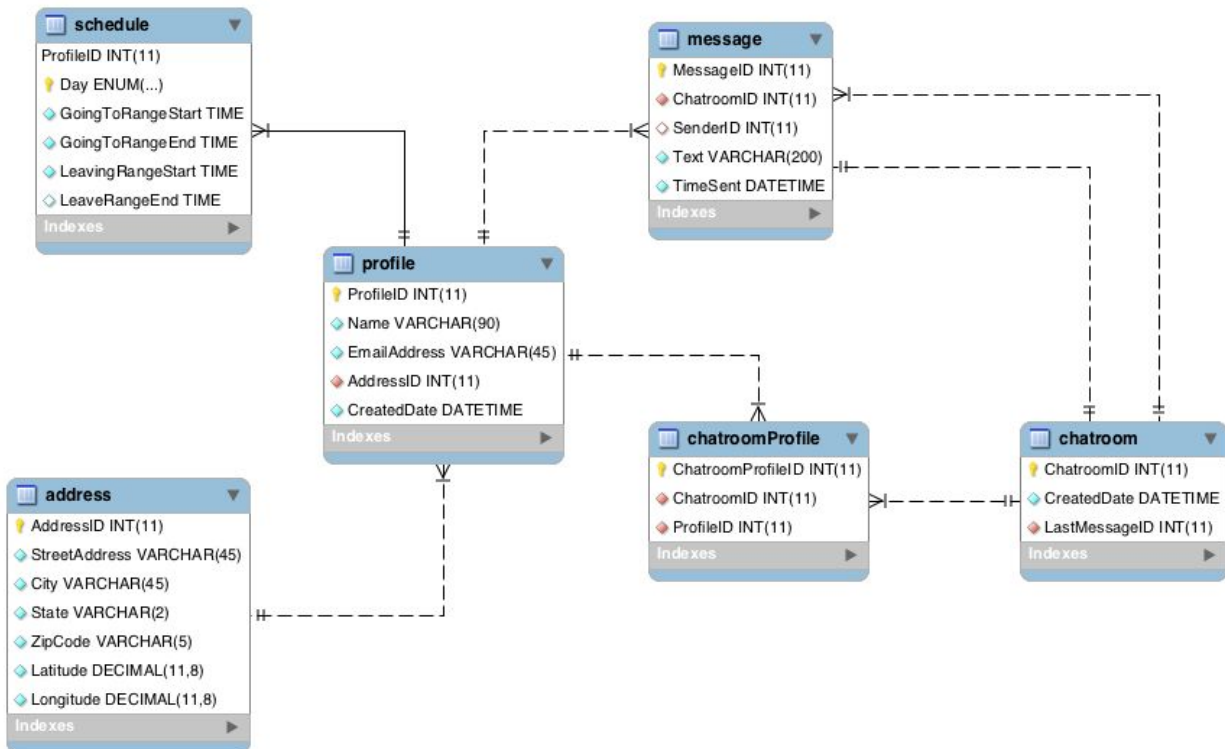
The app's user authentication will be performed using OAuth, and the user will be using their normal Rowan account logins to use the app. Doing so, we don't need to store a user's login or password. Additionally, this app is not intended for those who are not students at Rowan, so there is no need for users to create their accounts from within the app.

## **Input and Output**

Input: The primary method of user input is using buttons to navigate the app's menus. The app also receives text inputs when the user is typing a message in the chat screen or adding information their profile or logging in to the app.

Output: The output provided to the user is a list of other users that are a good match to start a carpool with.

## Database Schema



For schedule, ProfileID and Day form a composite primary key. The Enums for Day are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday.

There are two relationships between chatroom and message: a 1-to-1 relationship to track the last message sent for a chatroom and a 1-to-Many since a chatroom can have many messages.

The chatroomProfile table exists as a bridging table for the Many-to-Many relationship between profile and chatroom.

## Restful Endpoints

The endpoints are labeled in the form of REQUEST TYPE /endpoint/url/. Inputs to an endpoint are denoted by brackets. For example, {user\_id} expects an integer. Additional parameters will be handled in through query strings and/or form data (for post requests).

### **GET /rides/profile/{profile\_id}**

Returns information about the profile corresponding to the profile\_id in the form of JSON.

### **GET /rides/profile/email/{email\_address}**

Returns information about the profile corresponding to the email\_address in the form of JSON.

### **GET /rides/profile/getmyid/{email\_address}**

Returns the profile ID corresponding to the email\_address in the form of JSON.

### **GET /rides/profile/{profile\_id}/chatrooms**

Returns the chatrooms this profile is part of in the form of JSON.

### **GET /rides/profile/delete/{profile\_id}**

Deletes a profile corresponding to the profile\_id in the form of JSON. Return false if given id does not exist.

### **GET /rides/address/{profile\_id}**

Returns the address for the specified profile in the form of JSON.

### **GET /rides/schedule/{profile\_id}**

Returns the current schedule for the profile corresponding to the profile\_id in the form of JSON.

### **GET /rides/messages/{chatroom\_id}**

Returns a list of messages in the chat conversation that corresponds to the chatroom\_id. Output is in form of JSON.

### **GET /rides/messages/delete/{message\_id}**

Perform deletion on a chatroom corresponding to the chatroom\_id in the form of JSON. Return false if given id does not exist.

### **GET /rides/chatroom/{chatroom\_id}**

Returns information(including profiles) about the chatroom corresponding to the chatroom\_id in the form of JSON.

### **GET /rides/chatroom/delete/{chatroom\_id}**

Deletes a chatroom corresponding to the chatroom\_id in the form of JSON. Return false if given id does not exist.

### **GET /rides/matching/{profile\_id}/{radius}**

Returns a list of potential matches for the specified profile in the form of JSON. Profiles are matched based on distance and proximity.

### **POST /rides/profile/new**

Creates a new profile using the information within the JSON. Returns the new profile.

### **POST /rides/profile/update**

Updates a profile using the information within the JSON. Returns the updated profile.

### **POST /rides/profile/{profile\_id}/link\_address**

Links an address to the profile corresponding to the profile\_id. Returns the profile.

### **POST /rides/profile/{profile\_id}/chatroom/new**

Creates a new chatroom using the information within the JSON and links to the profile corresponding to the profile\_id. Returns the profile.

### **POST /rides/profile/{profile\_id}/chatroom/update**

Links the profile corresponding to the profile\_id to the chatroom using the information within the JSON. Returns the profile.

### **POST /rides/profile/{profile\_id}/schedule/new**



Creates a schedule for the profile corresponding to the profile\_id using the information within the JSON. Returns the schedule.

### **POST /rides/profile/{profile\_id}/schedule/update**

Updates a schedule for a profile corresponding to the profile\_id using the information within the JSON. Returns the schedule.

### **POST /rides/address/new**

Creates a new address using information within the JSON. Returns the address.

### **POST /rides/address/update**

Updates an address using the information within the JSON. Returns the address.

### **POST /rides/address/{profile\_id}/new**

Creates a new address using information within the JSON for a profile corresponding to the profile\_id. Returns the address.

### **POST /rides/chatroom/new**

Creates a new chatroom for matched users using the information within the JSON. Returns the chatroom.

### **POST /rides/message/new**

Creates a message using the information within the JSON. Returns the message.