NEVER RIDE SOLO!

FOXLIFT

By: Anthony Sasso, Jake Vissicchio, Nicholas Petrilli, Sean Ginsberg, and Jonathan Eisenman

A ride-sharing app that allows Marist College students to share rides with each other

CMPT 475 Capping Project 2022

Table of Contents

Section I: Introduction

What are we building?

Our capping project will allow an application for Marist College students to share rides with each other to different destinations. This will help students find rides from other students or share Uber rides with each other. For example, there are many times when multiple Marist students will want to get off campus to go to stores like Target at the same time, resulting in many different people that live in the same community driving their own cars or spending money for their own Uber/Taxi. With FoxLift, users can help each other and maybe even find new friends that share similar hobbies. This is planned to be an app for the community as well as a great way to support the environment and encourage carpooling.

Who are the Users and Stakeholders?

Marist College, Marist Students

Project Guidelines:
https://github.com/JonathanEisenman/RideSharingApp/blob/main/Ride_Sharing_App.pdf

Section II: Necessary Installations

What to install:

React Native, MySQL v8.0.21, ExpoGo, Google Maps API (Places API and Directions API), NodeJS v16.8.0, and Zenhub.

Websites used:

Figma, LucidCharts, Slack, and GitHub.

All NPM/Yarn installations used in the FoxLift project:

These package installations allowed us to meet the requirements for our application. Since we used the Expo CLI, "yarn add" is the equivalent to "npm install".

npm install -g eas-cli

npm install e-mine/react-native-html-text

npm install expo/webpack-config

npm install react-native-community/checkbox

npm install react-native-community/datetimepicker

npm install react-native-google-signin/google-signin

npm install react-navigation/bottom-tabs

npm install react-navigation/native

npm install react-navigation/native-stack

npm install react-navigation/stack

npm install body-parser

npm install dom

yarn add expo

yarn add expo-auth-session

yarn add expo-constants

yarn add expo-location

yarn add expo-random

yarn add expo-status-bar

```
yarn add expo-vector-icons
yarn add expo-web-browser
npm install express
npm install mysql
npm install react
npm install react-dom
npm install react-native
npm install react-native-date-picker
npm install react-native-gesture-handler
npm install react-native-gifted-chat
npm install react-native-google-places-autocomplete
npm install react-native-location
npm install react-native-maps
npm install react-native-maps-directions
npm install react-native-modal-datetime-picker
npm install react-native-modal
npm install react-native-safe-area-context
npm install react-native-safe-area-view
npm install react-native-screens
npm install react-native-vector-icons
npm install react-native-web
npm install react-navigation
npm install styled-components
npm install unique-names-generator
```

Our group has only experience with the Apple iPhone display in ExpoGo. We have yet to gain experience using it on Android phones.

Section III: Documents and Information

Documentation: We stored all our files on our GitHub Repository page. All documents on that page are necessary to make our application. Listed below can find each directory.

Our GitHub Repository:

https://github.com/JonathanEisenman/RideSharingApp

Our FoxLift code:

https://github.com/JonathanEisenman/RideSharingApp/tree/main/FoxLift

Our Web Server code:

https://github.com/JonathanEisenman/RideSharingApp/tree/main/FoxLiftWebServer

Database Scripts:
https://github.com/JonathanEisenman/RideSharingApp/tree/main/Resources/Database%20Scripts

Our Requirements Document:
https://github.com/JonathanEisenman/RideSharingApp/blob/main/Resources/Requirements%20Document.pdf

Our Use Case Diagrams:
https://github.com/JonathanEisenman/RideSharingApp/tree/main/Resources/Use%20Case%20Diagrams

Our Weekly Status Report:

https://github.com/JonathanEisenman/RideSharingApp/tree/main/Resources/Weekly%20Status%20Report

Our Server Information:

Hardware: CPUs - 2RAM - 2GB Storage - 16GB Software: CentOS Linux 7

IP Address: 10.10.9.188

Section IV: Bug Tracking and Testing

Bug Tracking: This is a list of bugs we tracked throughout the project to remember current and past bugs that caused an issue. We tracked the bugs via Zenhub to deal with the issues effectively.

- **September 3rd**:
    - Pushing React Native project into GitHub. There were way too many files which prevented us.
- **October 3rd**:
    - Trouble connecting React Native to our database via NodeJS Server.
- **October 17th**:
    - ExpoGo does not allow custom native mode. Because of this, we were having trouble integrating Google Authentication within our app.
- **October 31st**:
    - Google Places autocomplete error not connecting.
- **November 7th**:
    - Google Maps location error. It is not showing the correct location pin.
    - Google OAuth Error 400 - redirect_uri_mismatch. This error prevents the user from being able to login using their Google account. This was fixed after a lot of time researching and realizing we need an Expo account that gives a redirect for the project.
- **November 14th**:
    - The server is not receiving external requests. The requests were being blocked by the server's firewall. This was fixed by editing the firewall and allowing access.
- **November 21st**:
    - Launch Screen - When logging in using the Sign-in with Google button, the user would be added to the database 3 times instead of once. This caused our database to get very filled with accounts that had the same email. This was handled within the endpoint by not allowing users with the same email to be added to the database.
- **November 25th**:
    - After tinkering a lot with the Launch Screen to no longer insert multiple users into the database, a new error occurred that made new users unable to be added to the database. This was fixed after reverting some lines of code back to what they were.
    - Google Places Autocomplete was not working at all in the app. This caused huge confusion and panic since nothing in the Home Screen was changed for this to occur. After retrying later in the day it began to work again.

- **November 28th**:
    - Apostrophe error - when a string (like origin or destination) has an apostrophe in the name, it will cause an error in the database. For example, O'Shea would crash the server because it will stop at O and crash seeing Shea after a closing apostrophe. This was fixed by changing SQL queries to use double quotes (") instead of single quotes(').
- **November 29th**:
    - Within the Messages screen - when selecting a user to chat with, it would take the previously clicked user's ID rather than the current click. Because of this, the first user clicked would always return an undefined user ID since there was no previous click. This error had something to do with React's useState. This was fixed by just using regular variables.
- **November 30th:**
    - The original text input box for filtering locations within the Upcoming Trip Requests screen did not work as intended. The user would have had to type in the entire destination text in order to properly filter results. To fix this, we reuse the code for Google Places Autocomplete on the Home screen so that it can help the users filter exactly what they are looking for.
- **December 2nd:**
    - On the Home screen - the size of the search bar would increase when tracing the route and displaying distance and duration. Since the search bar is displayed above the map, this affected the user's view of their route. This issue was fixed by modifying the search bar layout and size to prevent it from resizing, along with changing the edge padding values of the map to properly center the route.
- **December 4th**:
    - While creating demos for the final presentation, we noticed that some of the dates were not displaying EST time correctly, and instead displayed in UTC time, 5 hours ahead. This was fixed by doing a calculation that subtracts 5 hours from a time.

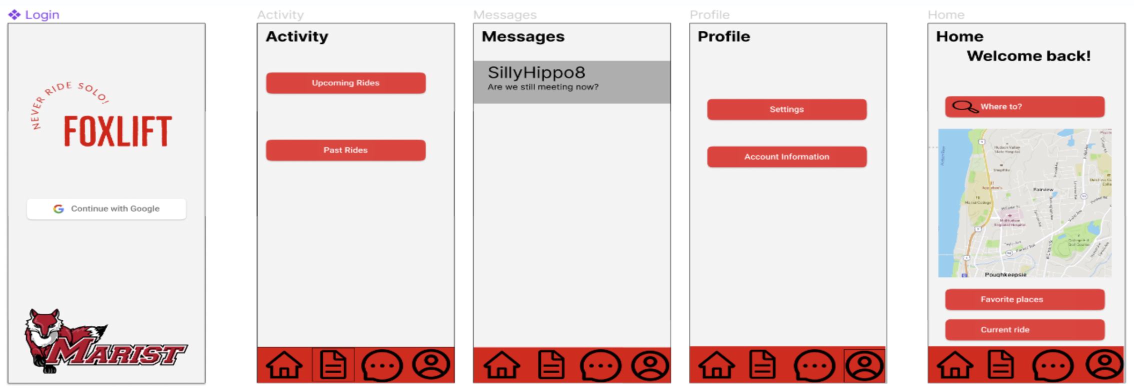
Testing: Throughout our project, we have tested to meet the criteria of our requirements document. For each line of code we implemented, we ran the code to see the updated progress of what has changed. ExpoGo is very nifty in that while running with the expo server you can make any change in the code and save it to automatically update what the program runs on the ExpoGo app. We tested the web server endpoints individually by sending requests from Postman.

Section V: Design

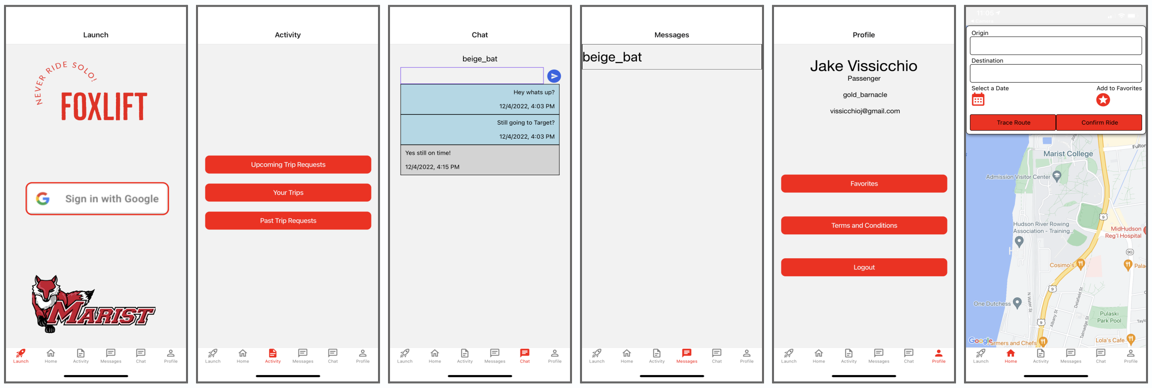
How did we design it?

We wanted the FoxLift app to resemble other ride-sharing apps in the past like Uber or Lyft that show a responsive map to the origin and destination. We also wanted to give each screen its own primary functions that branch off into sub-screens. For example, Activity could branch off into the other sub-screens [Upcoming Trips, Your Trips, Past Trips]. This allows for a much more hierarchical design so that the main screens are not too cluttered with buttons and information. When completing the final design, we mirrored it pretty close to our original Figma design, but added more screens and removed a lot of cluttering. We also made sure to try and put a lot of the focus for each page towards the bottom of the screen since that's where the users' thumbs will be.

Initial Design using Figma (UI Designer):



Final Design:



Section VI: Project Organization

How did we keep our project organized?

Each week, we updated our Zenhub, a project management tool to list any issues/tasks that come up. Our team meets 2 times per week at the library to discuss what needs to be done as well as collaboratively working on the app. We tried to make sure that we met in person because we found that we got a lot more done compared to Discord meetings. We assign tasks to one another to keep ourselves organized. Zenhub creates automatic charts to track our progress by showing how many tasks need to be completed.

In our weekly status report, we covered topics that we worked on throughout the week and showed our progress in the FoxLift project. We discuss issues that occur through color categories. Red is an extreme concern in the project, yellow is a medium issue and green is a low-priority issue. Many of the status reports also included a demo, which would actually show our client what can now be done in the app, rather than just stating it.

Our use case diagrams blueprint design for our project. Each diagram explains what a user needs to be able to perform within each section of the app. The ERD provides a visual of our database design and explains how our data will be set up and how they link with each other. The UML diagram shows all the functionalities the user can do on each screen. The use case diagram explains the actions the driver, student, and admin can access.

Section VII: User Training Manual

The steps below show the full user manual for our application. Each step goes into detail about how the app is structured and how it functions:

1. Upon opening the application, users will be prompted to login using their Google account. Clicking the button will redirect to our configured OAuth 2.0 consent screen, where users can login with a Client ID. Note: Users must have a Google account in order to use our app.
2. Once a user is logged in, the Launch screen will display a welcome message and prompt the user to select their role by asking if they have a car to use for the app (roles are passenger or driver). If a role is not selected, users will be passengers by default.
3. Users can click the "Start Ridesharing!" button, or use the bottom tab bar to be navigated to the Home screen, where users can create their desired rides. This is done by using the Google Places Autocomplete search bar to enter the origin and destination of the ride. Next, users must trace their ride which will display the trip's distance and duration, as well as display the route on the map. Lastly, users must select a date for their trip before clicking the "Confirm Ride" button. If any of these items are missing when the confirm ride button is clicked, an alert will pop up notifying the user what information they are missing. Also on the home page, but not necessary for creating a ride, is an "Add to favorites" button. This will add the entered destination to the user's favorites list which is displayed on the Profile screen.
4. Once all of the necessary information is inputted by the user, the confirm ride button will add that user's trip to the database. This ride will be displayed in the "Upcoming Trip Requests" button on the Activity page for all users to see, as well as the user's "Your Trips" button. At this time, users will wait for other users to browse the upcoming trip requests on the Activity page to possibly join the ride.
5. As a separate user, you can browse the upcoming trip requests on the activity page for any trips you may want to join. All upcoming trips can be filtered by searching for a destination, or by entering either minimum or maximum time (or a range of the two). To join the trip, users can click on the specific trip and an alert will pop up asking the user to confirm joining. If the user confirms to join the ride, the trip will be added to the "Your Trips" screen within the Activity page as an ongoing trip.
6. Once this second user joins the trip, this will connect the two users to be able to message each other. The messages tab will display any usernames of users who are in the same ride as you. Each name is clickable, and you will be navigated to a chat screen where the two users can send messages back and forth. Users can send messages by typing a string into the text input box and then clicking the blue arrow button. The messages sent from the current user will be displayed right-aligned with a blue background and messages received from the opposite user will be left-aligned with a gray background.

7. If the trip needs to be canceled for any reason, or the trip has been completed and should be marked as such, this can be done on the Activity page. Users can go into their “Your Trips” section, click on any ongoing trips, and they will get a pop-up allowing them to cancel or complete the trip. Then the ride will be moved into either the completed trips or canceled trips section on the “Your Trips” page. This action will also update the “Upcoming Trip Requests” and move the trip from there to the “Past Trip Requests” page.
8. Lastly, the Profile page displays the user’s information such as their name (from their google account), their role as a passenger or driver, their randomly generated account name, and the email linked to their google account. Important note: the profile screen is the only place where users can see their personal information like their name and email, which is only available for their view as all other users will only be able to see them as their randomly generated account name. Buttons on the profile page include a favorites button to view any destinations starred as favorites, terms, and conditions button that displays an alert notifying the user that FoxLift is not responsible for anything that goes wrong during the rideshare, and a logout button that brings users back to the launch page.