Team Retrospective – Lynx Lifts

Skills We Picked Up

On the frontend, the app was built using React Native, which allowed for cross-platform compatibility on both iOS and Android devices. We used JavaScript to implement dynamic interfaces that let users request rides, view driver availability, and manage trip details. Tools like React Navigation and Reanimated were used to enhance the user experience with smooth transitions and gesture-based features like swipe-to-delete/swipe-to-accept functionality. These tools also integrated seamlessly with React Native's Gifted Chat library, which provided an interface for drivers and passengers to communicate.

The backend of Lynx Lifts was developed using Node.js to handle HTTP requests, user authentication, and data management. The backend exposed a set of RESTful APIs that the frontend could interact with, allowing for operations such as registering users, confirming ride completions, and updating ride status. PostgreSQL served as the database system, where critical information like user accounts, ride details, timestamps, and ridestate values were stored and queried efficiently. We used SQL to implement ride matching logic and to generate alerts based on time-sensitive conditions.

This app gave us experience with resolving merge conflicts. There were conflicts that were only a few lines different and straightforward to resolve. Other conflicts which involved multiple files with multiple line differences tested our skills with using the merge editor.

We also spent a lot of time working with external APIs like Google Maps, Google Reverse Geocoding, Google Distance Matrix, and Firebase Cloud Messaging. Using these APIs required using API keys and understanding the importance of preventing sensitive data from being exposed.

What We Learned

A major lesson we learned was how important communication is. The first few sprints were difficult, since we ran into technical issues with compatibility between cross-platform development (and different emulators between iOS and Android). Through the technical issues, we worked on our communication skills. As a group, we kept each other updated with our progress. At the end of each sprint, we would re-group and discuss where we are at on our Trello Board–whether we were behind on our user stories or not. This strengthened our ability to work independently in a group, and further our communication skills (especially with branch-based development and merge conflicts on GitHub).

Additionally, there were times when progress slowed down just because we weren't all on the same page. It was not completely clear to some of us how we envisioned certain features, which

caused confusion at the start of the development process. Once we started prioritizing regular check-ins, things got a lot better. Breaking the work into mostly independent tasks also helped us work more efficiently without stepping on each other's toes.

The importance of communication also became apparent when our API key got exposed twice. After the first time it was exposed, we began emphasizing the importance of not making this same mistake again. After the second time, it reinforced the necessity for all team members to be on the same page. We also learned why it is crucial to ask for clarity if something is not completely clear.

We also learned what it takes to build a fully functional app from scratch—how the frontend, backend, and database all need to communicate effectively. It was a learning curve with setting up our architecture, and learning how to connect our database to our backend and writing the API routes between our frontend and backend.

Along with this we learned the hard way the difficulty in developing for cross-platform. Setting up FCM taught us a lot about the platform-specific challenges of notifications—like the differences in permissions and delivery behavior between iOS and Android. We also ran into instances where the formatting of the UI varied slightly between platforms (such as with the Time Scroller).

Did/Did not accomplish

Our final product matches our MVP that we originally planned in Sprint 0. Our MVP:

a. Students can create an account using their @rhodes.edu outlook accounts
b. There is a feed interface for both the driver and passenger
c. Passengers can select map location, calculate payment, and post on feed
d. Drivers can view posts on the feed, receive payment, online/offline status, coverage area, and accept ride requests
e. There is a chat system to coordinate communication between the driver and passenger(s) and a report system to maintain the in-app experience

Our final product checked off each feature we originally planned for our MVP. There were two optional features (carpooling and repeated rides) that we did not get to, however, we will be giving our code to the on-campus Rhodes College Lynx Lifts Organization after our graduation. We hope they can add on and develop our code further.

Our app was tested both through the emulators and on a physical device (only Android).