Spring 2019

Bridgr: An iOS Application for Organizing and Discussing Long-Distance Carpooling

Harrison Engoren
University of South Carolina - Columbia, harrison.engoren@gmail.com

Erik Zorn
University of South Carolina - Columbia, erik.zorn44@gmail.com

Follow this and additional works at: https://scholarcommons.sc.edu/senior_theses
Part of the Other Computer Engineering Commons

Recommended Citation
Engoren, Harrison and Zorn, Erik, "Bridgr: An iOS Application for Organizing and Discussing Long-Distance Carpooling" (2019). Senior Theses. 286.
https://scholarcommons.sc.edu/senior_theses/286

This Thesis is brought to you by the Honors College at Scholar Commons. It has been accepted for inclusion in Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact dillarda@mailbox.sc.edu.
BRIDGR: AN IOS APPLICATION FOR ORGANIZING AND DISCUSSING LONG-DISTANCE CARPOOLING

By

Harrison Engoren & Erik Zorn

Submitted in Partial Fulfillment
of the Requirements for
Graduation with Honors from the
South Carolina Honors College

May, 2019

Approved:

[Signature]
Jose M. Vidal
Director of Thesis

[Signature]
Duncan Buell
Second Reader

Steve Lynn, Dean
For South Carolina Honors College
Thesis Summary

Bridgr is a startup idea that we formulated after realizing how often students at USC are unknowingly driving to the same place at the same time. Typically, many students at USC are in need of a ride home or to some other destination. Bridgr attempts to connect these students and give them the opportunity to carpool. The benefits of carpooling are numerous. Sharing the journey with someone else allows carpoolers to save money, foster companionship, and ease the struggles of the journey by allowing them to share driving time. Further, carpooling has positive environmental effects because it reduces the number of cars on the road. Bridgr strives to match students together traveling on the same route and cultivate the Carolina community.

We believe that there are many occasions where Bridgr would benefit USC students. For example, around the holidays many people are forced to buy expensive bus tickets or flights when there are in fact thousands of students driving routes that would get a student in need home. Even for those who prefer to fly, the limited amount of flights that Columbia’s airport provides will lead to many students instead preferring to fly out of a major hub such as Atlanta or Charlotte. Not only are there thousands of students at USC from Atlanta or Charlotte, but there are also thousands of students who will be driving through or near Atlanta or Charlotte that could potentially be persuaded to go a little bit out of their way to drop off a fellow USC student at the airport.

At the University of South Carolina, we take our athletics very seriously and many students travel to other schools in the Southeastern United States to attend away football, basketball, and baseball games. We believe that this is another situation where Bridgr would provide a lot of value. There is certainly a large group of students at USC who would be more inclined to travel to support our school if they had someone to carpool with.
Yet another scenario where Bridgr would be helpful is when students want to take a weekend trip to one of the many fabulous cities located near Columbia such as Charleston, Asheville, or Savannah. There are often concerts and even music festivals in these cities and again, students who are already driving on this route could be enticed by the idea of sharing the journey with a fellow USC student.

The most important characteristic in all these use cases is the high probability of recurrence. The same students will drive the same routes over and over again throughout their college careers. Through Bridgr, users would be able to not only find one-time matches, but also create lasting relationships that they could use repeatedly for carpooling.

Based on these use cases, our initial target audience is USC students, and we will be delivering Bridgr in the form of an iOS application. The iOS user interface is created through a popular framework designed and used by Facebook, React Native. The user interface is linked with a backend database called Firebase, which is a free database service provided by Google. Through the user interface, users are able to perform simple CRUD (Create, Read, Update, Delete) actions on the database. In order to create a personalized experience for each user, we also use Redux, a state management framework, to store specific information about which user is currently logged in and control what actions that user can perform.

In terms of user interface, we emulated popular mobile applications with a sleek, minimalist, black and white design scheme. We also mimicked many of the components that can currently be found in some of the most popular mobile applications. For example, just as most users can swipe on an email in their inbox to delete, forward, or reply to an email, Bridgr users can swipe on a request to either accept or decline another user’s request to join their ride. This
familiar feel will allow users to seamlessly transition into using our app and enhance their overall experience.

Another strategy we chose to employ to increase usability is a Frequently Asked Questions (FAQs) page. Through our main navigator, the user can navigate to the FAQs screen, where they can learn how to use our app. There will be answers to questions that tell the user how to use our app such as “How can I post a ride?” and “How can I request to join a ride?” as well as questions that explain the idea behind our app such as “Can I join a ride even if I don’t need to go to the final destination?”.

With these design principles in mind, we were successfully able to implement the core functionality and produce an effective, user-friendly application. A user can easily post and request rides, manage a ride, message other users, and receive notifications from our app. In the future, if we were to deploy this application on Apple’s App Store, we would take extra efforts to shore up security and take the necessary steps to ensure rider and driver safety.
Introduction

Bridgr is an iOS mobile application that connects students at the University of South Carolina for long-distance carpooling. In order to develop this idea, we had to design an app that catered to two different types of users. The first type of user we will refer to as a Driver. A Driver is someone who is planning on making a trip to a certain destination at a certain time, has available seats in their car, and is seeking companionship and/or financial compensation. In contrast, the other type of user is a Rider. A Rider is someone who would like to join a Driver traveling to a certain destination. Another important distinction in verbiage is how we refer to the journey that a Driver or Rider takes. In general, they are called Trips, but more specifically a Driver’s Trip is a Drive, and a Rider’s Trip is a Ride. Since there are two types of users, there is also a distinction between the core functionalities. The core functionality for a Driver includes letting the user post and edit a Drive, accept/reject/delete Riders from a Drive, and message potential Riders about the Drive. For a Rider, the core functionality includes being able to search and choose a Ride, view details about the Ride, request to join the Ride, and message the Driver about the Ride. The main purpose of our project was to implement this core functionality, along with other features to enhance the user experience.
Overview of the Features

The first step for a new user is to sign up for our app. The user can click the “Create Account” button and they will be redirected to a page where they will enter their email and password. They will also fill in their biographical information such as their name, birthday, and a bio. The biographical info will be displayed on their profile page once they create an account. Currently, we require that all of our users be USC students in order to try to promote a safe environment for our users. To enforce this, we use the following two-step method: a user must have an email that ends in ‘@email.sc.edu’ and they will receive an email to their USC email with a verification code that they will enter into our app to validate their email. Once their email is validated, the user is free to use all the features of our app.

If the user already has an account, they simply need to enter the email and password they signed up with to login. We utilized Firebase’s built-in authentication to implement the login feature which was convenient because it has error handling for incorrectly entered emails or passwords. To make the whole process easier for the user, we also included a “show password” button, so that user can toggle their view of their password between the traditional dots and alphanumeric characters (Screenshot 1).

Once a user has created an account, the first thing they will probably want to do is either post a Drive or find a Ride. The user can do either of these actions from the home screen. The home screen (or map screen) displays a map centered around Columbia, SC and features pins for all Drives posted by other users. The home screen also displays a button in the bottom corner that a user can press, and they will be redirected to a screen where they can enter details to post a Drive. In the header for the home screen is a button that toggles a drawer navigator and another button that brings the user to the notifications screen. These buttons remain in the header
throughout the entire app except in certain cases when a back button will replace the drawer navigator button.

When the user presses the “New Drive” button, they are redirected to a screen that presents them a form to enter information about the drive. These fields include the destination, the departure time, the number of seats available, and a short description about the drive. Once the fields are populated, the user can press the “Post Drive” button which will post the Trip and bring them back to the home screen. If the user incorrectly fills out the form, an error message will be displayed, and user will have to correct their error.

Alternatively, the user can view Rides they might want to join. There are two ways to do this -- they can either press on a pin on the map or swipe up from the bottom to display a list of cards and press one of the cards (Screenshots 3,4). Either method will bring the user to a Trip Info page. The Trip Info page shows the route from the University of South Carolina to the destination. Below is a tab that displays the name of the destination and allows the user to swipe up to view additional details about the Ride. Also, this page is where the user can request to join the Ride and be redirected to the chatboard for the specific Ride.

Another action the user can perform from the map screen is to search a Ride. To open the search box, the user swipes up from the bottom to reveal the search box and the list view of Rides. In the search box the user can filter the list of Rides based on three criteria. The first is the destination. Searching by destination will reorder the list so that the closest Ride to the destination is displayed first and the furthest is displayed last. Further, the mileage from the searched location to each Ride’s destination will be displayed next to each Ride. The second criteria is a date range. The user can enter a start date and an end date, and only Rides within that range will be shown. If no Rides are in that date range, it will show “No Rides Available”. The
last criteria the user can search by is seats available. For example, if a user searches for all Rides with one seat available, all Rides with at least one seat open will be displayed. The user can search by these criteria individually or use some combination of the three to make a more advanced search. Behind the scenes, the search works by simply looping through all Rides and performing a conditional check on the details of the Ride.

Another feature available is the user can view their own profile. On the user profile, we display basic information about the user such as their name, age, major, and bio. We also display the user’s avatar. When viewing your own profile, the user has the ability to change the avatar and select a new picture from their device. We implemented that with React Native Image Picker. The user profile also displays a series of tabs that allows the user to see All of a user’s Trips, their Drives, their Rides, or their Pending requests for Rides that haven’t been accepted or declined (Screenshot 2). To render the profile page, we make a query to Firebase based on the user id and filter all the Trips into the appropriate sections.

The user can also navigate to the requests page using the drawer navigator. On the requests page, the user will see all the open requests. Each request can be swiped or tapped to reveal a green check mark and a red x. Pressing on the green check will add the passenger to the user’s Drive, while pressing the red x will reject the passenger. In Firebase, the requested passenger’s user id will be removed from the requestedIDs array. Then, if the user accepted the passenger to the Ride, the passenger’s user id will be put into the passengerIDs array.

Similar to the requests page is the notifications page. Using the same user interface as the requests, the notifications will populate the screen. However, when a user swipes on a notification, they will be routed to the appropriate page to display more information about the notification. Notifications have five different types: Ride Updated, Request Accepted, New
Request, Rider Flaked, and Ride Deleted. A user will receive a push notification to their device each time one of these events occurs. Additionally, a user will also receive a push notification every time a new message is sent to a chat they are subscribed to. We chose to omit messages from the notifications screen because we did not want it to get too cluttered.

If a user makes a mistake when they are posting a Drive, or if plans change, the user can choose to edit the details of the Drive by pressing a button on the Trip Info page. This will take the user to the same form as when they created the Drive, and they will have the ability to change any of the details about the Drive. In Firebase, this will update the necessary fields in the document. Alternatively, if plans change so drastically or if the user no longer wants to make a journey, they can delete one of their pre-existing Drives. To do this, the user will press a button on the Trip Info page and upon confirmation, the Drive’s document will be removed from Firebase and no longer displayed anywhere within the app.

On the Trip Info page, a user has a few other options. They can press the “View Riders” button, which will redirect them to a page where they can view who has requested to be on the Trip and who is already on the Trip. If the user is the Driver of the Trip, they also have the ability to delete passengers from their Drive. In Firebase, this works by getting the user id from the passenger and removing it from the passengerIDs array. Another button the user can press on the Trip Info page is the messages button which will navigate the user to the message board for the ride (Screenshot 5). On this page the user can message all users involved on the Trip and it will update in real-time. We made this functionality possible by creating another database on Firebase specifically for messages.
Screenshots

Screenshot 1: The login screen. Also gives the user the ability to create a new account.

Screenshot 2: The profile screen. This displays a user’s info and Trips.

Screenshot 3: On the map view, the user can interact with the map and click pins to view more info on a Trip.

Screenshot 4: With the list view open, a user can enter a search query to filter the list of all available Rides.

Screenshot 5: On the Trip Info page, a user has the option to take several actions such as request/leave a Trip or view the passengers and messages.
Database Schema

Bridgr's Firebase Realtime NoSql Schema

Bridgr’s backend real-time database is provided by Firebase, a Google product. For the scope of this project, there was no need to purchase any higher tiered trial of Firebase. We chose this as our backend because of its thorough documentation, easy integration with iOS applications, and generous free trial quotas. Firebase also includes a myriad of built-in features that alleviated many stresses for us. These features include, but are not limited to, user authentication, cloud functions, and Firebase Storage.
A necessary feature of our application is user authentication in order to keep our application and users secure. Fortunately, Firebase has several of its own authentication methods which we were able to choose from in order to best meet our needs. Bridgr utilizes the email and password-based authentication. This method stores user credentials securely, allows for password resets, and prevents weak passwords and duplicate emails. We paired Firebase’s built-in authentication with in-house regular expressions and email verification codes to restrict users to people with active “@email.sc.edu” emails.

The email verification codes were made possible through leveraging Firebase Cloud Functions and their standard Firestore database. Firebase allows developers to create functions that are triggered upon database writes. We wrote functions that paired with the authentication to send the user an email when a new account was created. Then we restricted full access until the user retrieved their unique verification code from their email and entered it upon their first login attempt. Another process facilitated by Firebase Cloud Functions is the sending of push notifications. For instance, whenever a ride is updated or deleted, a push notification is sent to all passenger and requesters to alert them of the change. The robustness of Firebase allowed processes to interact fluidly with each other and reduce the amount of different software necessary to complete features of Bridgr.

Firebase Storage is another built-in feature that made Firebase an optimal backend for us to use. Storage is used for uploading and downloading files that are not plain JSON. Bridgr uses Firebase Storage for managing the user profile pictures. Fortunately, because of the current scale of Bridgr’s user base, Firebase Storage feature was a clean solution to handling images without cost or extra software.
Technical Obstacles Overcome

When working with any new technology there are unfamiliar obstacles to solve. Since Bridgr uses several different software, a major issue to consider is how every component will interact with each other. In addition to considering how frameworks work together, there were also implications generated from how our teammates and respective devices work together. Some of these problems include design intentions, dependency consistency, and version control practices.

React Native as a framework posed many issues because of its steep learning curve. We chose to use React Native because of its capability to convert and compile Javascript into native iOS code. An implication that comes along with this is the need for many dependencies which cause long build times. The length of builds alone posed issues because of the amount of time getting our environments up and running. This was a major obstacle when adding additional dependencies to our application because a majority of our time was spent troubleshooting the rebuild errors that stemmed from the new or updated versioning. Another inherent problem for us working with React Native was learning Javascript fundamentals. Few of our team members previously had experience with asynchronous programming. This led to issues when first working with asynchronous database calls and handling state changes. At first, we had trouble locating these bugs because they would not always raise actual errors, but rather they would cause abnormal behavior within the app. Once we became better versed with the nature of callbacks and asynchronous operations, we were able to deduce solutions to problems that were formerly daunting. Fortunately, the React Native development community is very active and helpful through Stackoverflow and GitHub. The learning curve of React Native was conquered by tracking our past solutions, being active in online forums, and peer troubleshooting.
Dependency consistency and versioning across development environments was a struggle by itself. Due to this, our team had to all use the same operating systems and software versions. Because our team worked with a variety of packages, applications, and software, each of which being constantly maintained, this was an obstacle that we battled with for the duration of the project. There were times when our team had to revert to stable versions of software in order to implement modules that were not yet supported by the latest versions of complimentary software. As a team, we navigated these struggles by staying cognizant of installing new dependencies and knowing that we would need thorough testing afterward. Other solutions were reached by having to manually link dependencies or use alternative package handlers to install dependencies. Since React Native is so popular among the mobile application development community, there are often multiple solutions to one problem. Whenever we encountered insurmountable dependency compatibility issues with one package, we sought out another package to meet our needs.

As with most software development teams, it was necessary to have consistent and reliable version control practices. For Bridgr we used Git and GitHub to facilitate this. Only 2 out of our 5 team members had any experience with Git practices. When assigning broad team roles, we benefited from designating one member to answer most questions related to version control. This reduced problems such as clear and concise commit messages, branch names, and merging solutions. In Git there are situations that can be handled in different ways and it is important to have all team members working in a consistent manner to make versioning easy to navigate. There were instances where one team member would resolve merge conflicts before submitting a pull request to development and unknowingly remove a feature that they had not previously pulled to their local repository. Fortunately, because we had an organized protocol for
committing, pushing, and reviewing code, this mistake was handled relatively quickly and cleanly. By the end of the semester, our team’s Git fluency had dramatically improved. This was evident through the speed increase of handling version control related issues and simply browsing our repository and viewing commits and merges.

Bridgr became a very large code base quickly with many developments. Technical obstacles were inevitable, but through careful planning, troubleshooting, and tracking previous solutions, a lot of the team’s conflicts were alleviated. The helpful React Native community often bailed us out of situations because of the significant online documentation and forum activity. By the end of the semester, our team became proficient in debugging the code base and handling version control issues smoothly.
Conclusion

Overall, the project is successful in building an iOS application that gives USC students a platform to discuss and organize long-distance carpooling trips. In the future, we hope to extend Bridgr to also be an Android application so that it is more readily available to all students. We want to explore other options to make the safety system more robust. We could take steps such as utilizing the school’s Duo Authentication and linking to social media profiles. This would allow users to be vigilant in evaluating other users. Other features we want to implement include adding customization of the departure point so that users can leave from multiple locations, a rating system for users, and a friend referral system. Still, our project achieved the core functionality we set out to accomplish as well as included several extra bonus features.