Poly Rides

Myra Lukens

Computer Science Department
College of Engineering
California Polytechnic State University
2017

Table of Contents

Abstract

The goal of this senior project was to replace the Cal Poly Ride Share Facebook group with a more effective mobile application, and to successfully market that application to the campus community. With over 16,000 members, this is one of the largest Facebook groups that exists. However, it lacks an efficient mechanism for searching for, offering, and requesting rides. This project aimed to complete a few goals: create a ride share app specific to Cal Poly, market and advertise this application, gather feedback, and redo the application to include features users requested. Poly Rides is an iOS and Android application which uses several modern APIs to create a user experience far superior to the Facebook group. This project focuses only on the iOS version. The first version of the application is currently available on the Android and iOS App Stores, and the second version aims to be available in Spring 2017.

Acknowledgements

Special thanks to:

- Vanessa Forney, for working with me on Poly Rides
- Dr. John Bellardo, for your forever guidance in iOS development
- Dr. David Janzen, for stoking my passion for software engineering

1.0 Introduction

Ride sharing has become a hallmark of the app revolution. Uber, valued at over $62 billion[1], is the most prominent mobile ride sharing platform. There are several other competitors such a Lyft, Curb, Didi Chuxing, Grab, and Ola, to name a few. All of these services supply on-demand rides, pairing drivers and passengers. They hire drivers as contractors, often full-time, and the average ride length is 10.2 minutes[2]. The concept is simple: a passenger uses their Uber mobile application to request a ride, a nearby driver accepts the request, and the app notifies the passenger when their driver has arrived. When they arrive at the destination, the trip ends.

Uber came to San Luis Obispo in mid 2014[3], 5 years after it began in San Francisco. Lyft, Uber's largest competitor, arrived in SLO in early 2017[4]. Aside from mobile apps, there are other ride share platforms commonly used. These include Craigslist, Ridester, Wheelie, and

---

[1] Newcomer, Eric. "Uber Raises Funding at $62.5 Billion Valuation." *Bloomberg.com*. Bloomberg, 03 Dec. 2015. Web. 12 Mar. 2017.
[2] Kuo, John. "Here's How Much You Need To Drive for Uber or Lyft." *NerdWallet*. N.p., 14 Feb. 2017. Web. 12 Mar. 2017.
[3] Kleslie@thetribunenews.com, Kaytlyn Leslie -. "Uber Expanding Car Service to San Luis Obispo." *Sanluisobispo*. N.p., n.d. Web. 12 Mar. 2017.
[4] Shuttrshutt@thetribunenews.com, Robert. "Lyft Launches Ride-hailing Service in San Luis Obispo County." *Sanluisobispo*. N.p., n.d. Web. 12 Mar. 2017.

more. However, among young adults, Cal Poly's prominent population, Uber's popularity far exceeds that of the others.

1.1 The Problem

While Uber and Lyft provide sufficient service to transport Cal Poly students to nearby destinations, the apps are not intended to be for long-range rides. The costs can add up, leading to immense fares that college students simply cannot afford. While Uber does not have a maximum ride length, Lyft caps theirs at 100 miles[5]. An Uber from Cal Poly to San Francisco is estimated to be between $465-$620[6]. Similarly, one from Cal Poly to Los Angeles is estimated at $397-$529. The majority of Cal Poly students are from one of these two areas: 30% of Cal Poly students are from the San Francisco Bay Area, followed by 20.8% from the Los Angeles area[7]. During academic holidays, these students are likely to return home. Additionally, beginning next year, Cal Poly first year students are not permitted to bring a car to campus, and sophomore and transfer students will have to register on the waitlist[8]. For many students, this leaves them in a problematic situation: how can they get home for the holidays? This is where the Cal Poly Ride Share Facebook group (Figure 1) comes into play. It tackles this central issue that Cal Poly experiences as a university in a remote location with most students' homes within driving distance. However, this is not a sufficient mechanism. There is not a consistent format for ride requests or offers, and no signifier

[5] Campbell, Harry. "Just How Far Is Your Uber Driver Willing To Take You?" *Forbes*. Forbes Magazine, 24 Mar. 2015. Web. 12 Mar. 2017.
[6]"Uber Fare Estimator." *Uber*. N.p., n.d. Web. 10 Mar. 2017.
[7] "Cal Poly Quick Facts." *Cal Poly Quick Facts - Find Out About Academics, Student Body, Campus Size, History, Graduates & Careers, Buildings and More*. N.p., n.d. Web. 12 Mar. 2017.
[8] "Residential Student Parking." *Residential Student Parking - Parking & Commuter Services - Cal Poly*. N.p., n.d. Web. 12 Mar. 2017.

when a driver's ride is full. This results in time wasted searching for rides, and opportunities missed.

Figure 1: Cal Poly Ride Share Facebook Group

Hundreds of rides are posted every day, making it difficult to find a relevant ride for the desired date and destination. With inconsistent nomenclature to identify destinations, it becomes difficult to search effectively using the standard "search this group" feature. For example, dates can be posted as "1/1," "Jan 1," "January 1," "next Monday," or other possibilities. Similarly,

for the location, people put the specific location or region, for example, “SF,” “Bay area,” etc. These inconsistencies in the formatting for both the date and location make it harder to search for rides. Figure 2 shows two sample posts.

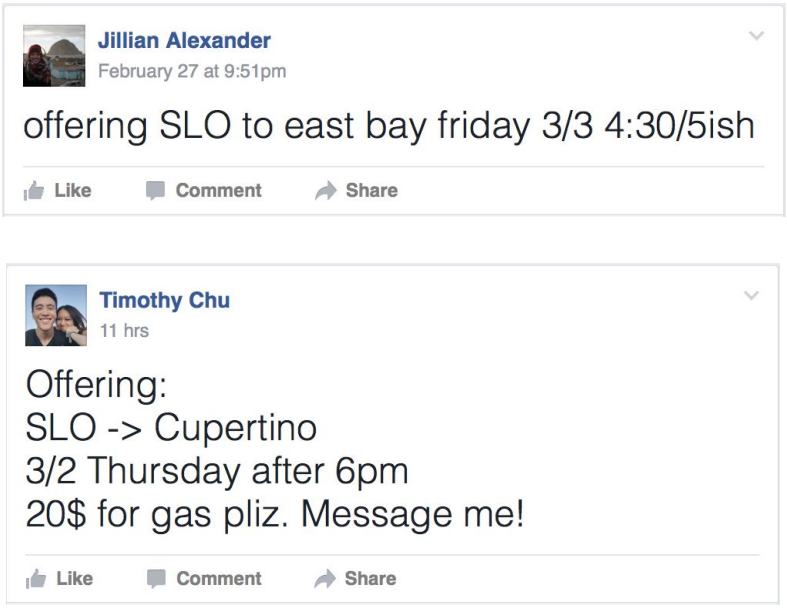


Figure 2: Posts on the Facebook Group

There are often several people interested in a ride, with no indication if it has been filled. The standard has become to message the driver on Facebook for further coordination. It is uncommon for drivers to delete their post in the Facebook group, even when their car is full. It can become very competitive to secure a driver, sometimes resulting in competition or even bidding on the Facebook page (see Figure 3).

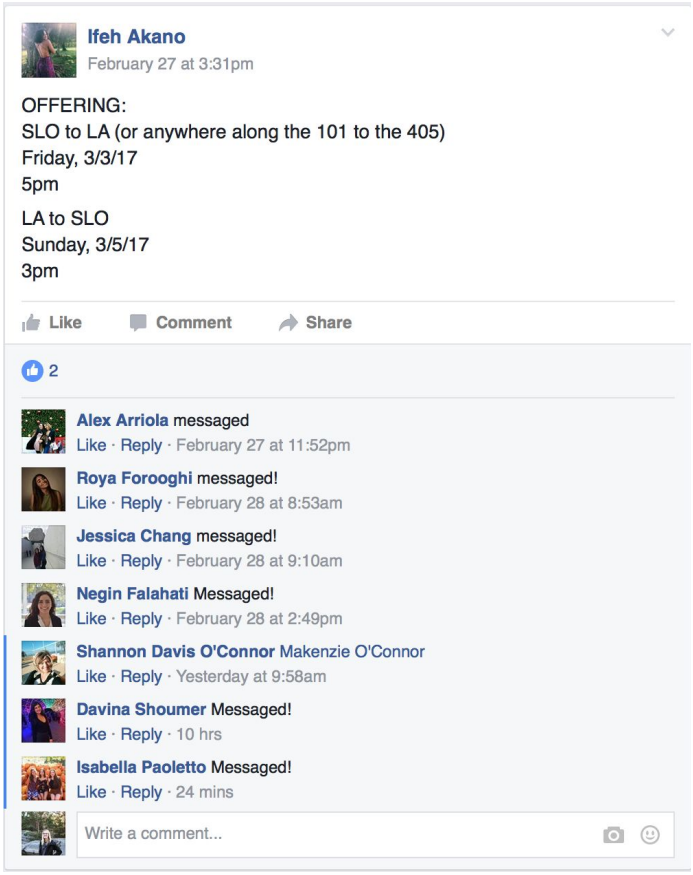


Figure 3: Competition for a Ride on the Facebook Group

Posts seeking rides are much less successful, and it is clear there is a surplus of passengers. Passengers often re-post their requests in order to propagate it to the top of the Facebook group for additional views (see Figure 4).

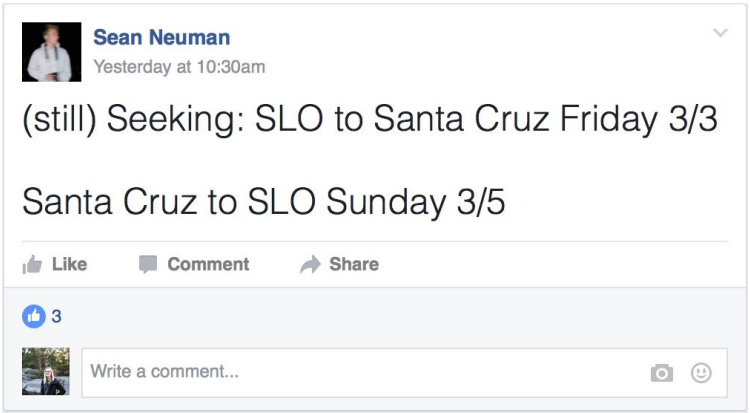


Figure 4: Multiple Posts for the Same Ride

The lack of organization in the Facebook group reduces ride opportunities. As someone with a car myself, I rarely offered rides on this Facebook group simply because of the hassle of coordinating. See Figure 5 for an example of coordinating a ride in the Facebook group.

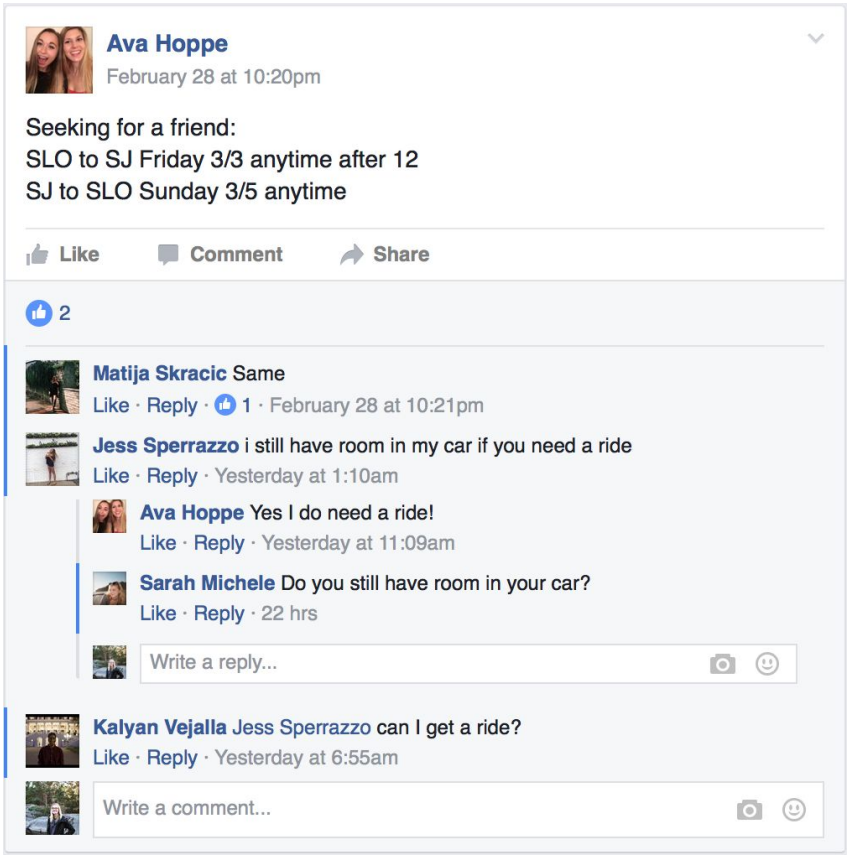


Figure 5: Coordinating a Ride on the Facebook Group

1.2 The Solution

Incepted in Dr. Bellardo's iOS class, Vanessa Forney and I began creating Poly Rides as a way to help the Cal Poly community and learn app development. The class ended with a rough draft of the app, with many features to be added before being released to the campus community. After class ended, Vanessa and I were determined to finish the minimal viable product (MVP)

and release it to campus. The companion Android application was developed by Michael Wong during Fall quarter 2015.

With the release of Poly Rides 1.0 for iOS and Android in Fall 2015, ride share at Cal Poly was significantly streamlined. We received over 1,000 app downloads in the first month alone, and that number grew quickly to over 4,000. This validated our assertion that a better ride share solution was desperately needed on campus. Poly Rides provides a simple interface to offer rides, search for rides, coordinate rides, view mutual friends with drivers, and message users. Drivers can offer rides, specifying their departure and arrival addresses, date and time of departure, cost per seat, and number of seats available. Passengers can enter a departure and arrival address and ideal date and time of departure. They will be given a list of rides available within 24 hours of their departure time, sorted by proximity to the desired route. Passengers can also see mutual friends with their potential drivers.

With the shutdown of Parse, the backend used for Poly Rides 1.0, Vanessa Forney and I decided that a revamp was necessary. She worked on Poly Rides 2.0 and handed it off to me in January 2017. I spent the next two months finishing the app, fixing bugs and implementing features, resulting in Poly Rides 2.0, which we hope to have released to campus at the beginning of Spring 2017. This version addresses key feature requests from Poly Rides 1.0, and more closely fits the needs of Cal Poly students in the realm of ride share.

2.0 Application Overview: Poly Rides 1.0

2.1 Logging In

When a user first opens the Poly Rides iOS application, they are greeted with an initial login page asking for them to log in with Facebook (Figure 6). We determined that Facebook login would be the most beneficial because we are targeting those users on the Facebook group, it provides a simpler mechanism than creating an account, and it easily connects their profile picture and mutual friends.

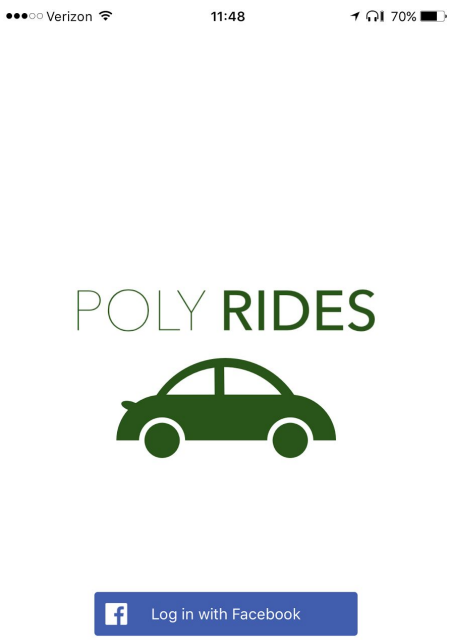


Figure 6: Poly Rides 1.0 Log In

2.2 Driver Flow

Users are greeted with a user-friendly home page, where they see four tabs: Driver, Passenger, My Rides, and Messages (Figure 7). It defaults to the Driver page, where users can input the start and end points, using Google Maps API for autocomplete, and then press the “Create New Ride” button to continue.

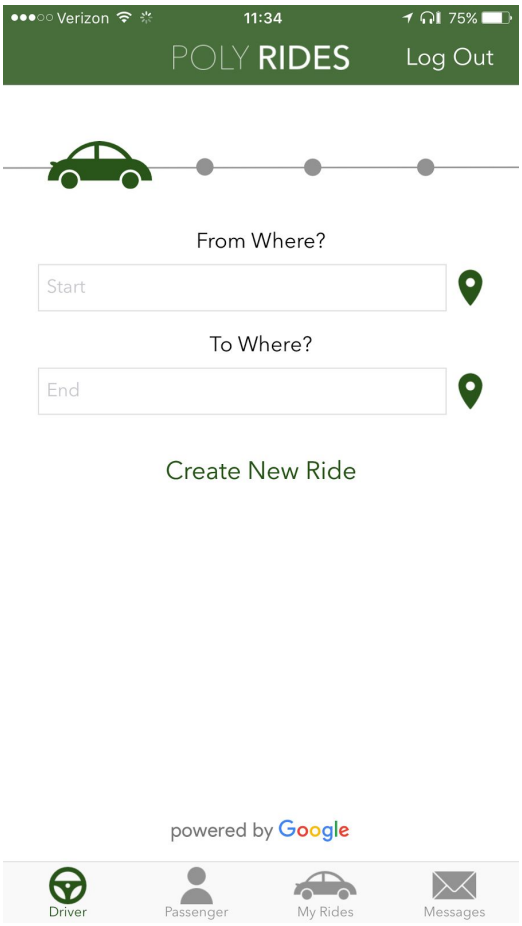


Figure 7: Poly Rides 1.0 Driver Page

The driver is then taken to input more information, information found most useful in the Facebook group posts (Figure 8). They are given feedback about their progress with the car icon at the top of the view.

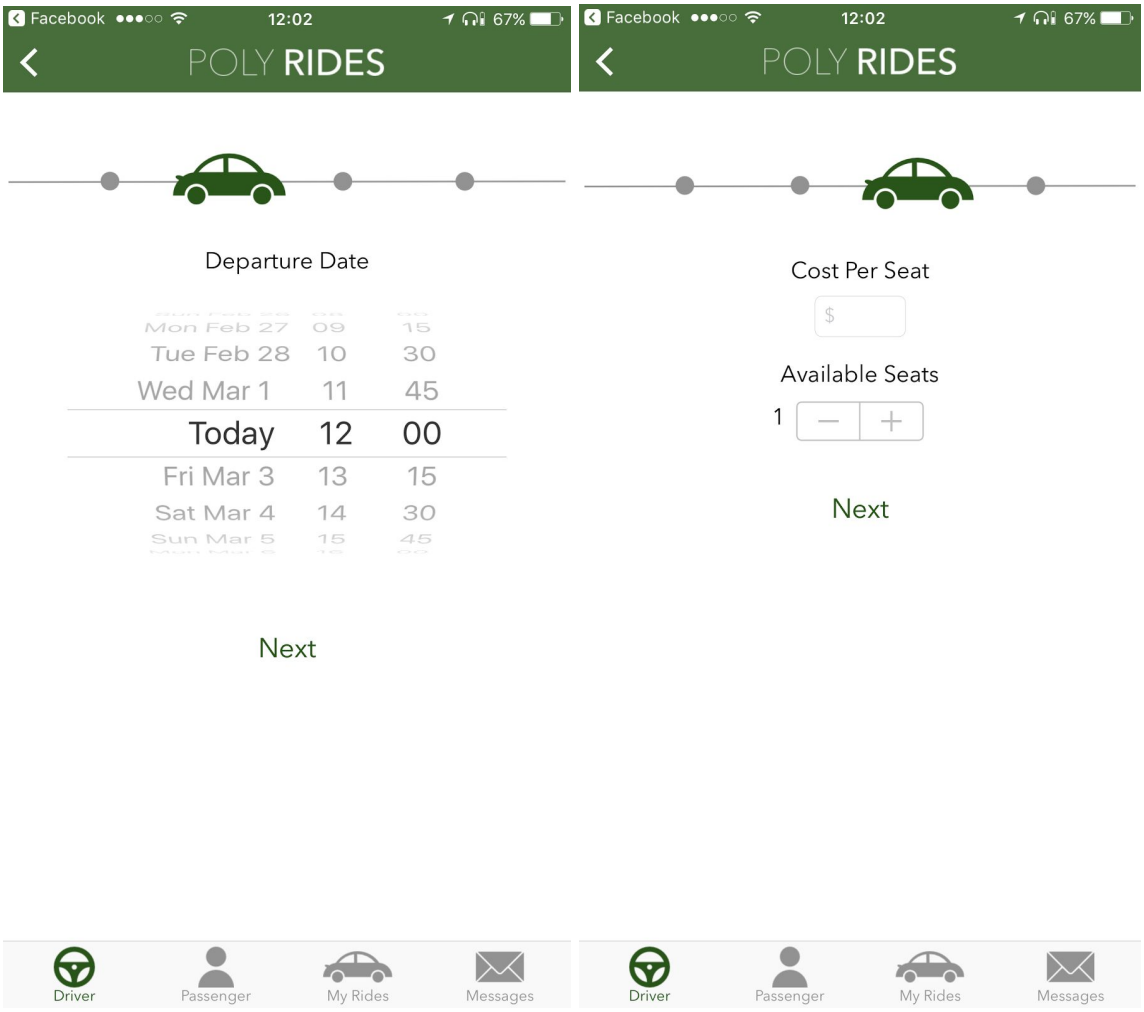


Figure 8: Poly Rides 1.0 Create a Ride

At this point, the driver details have been inputted, and they see a summary of their responses. Upon tapping the “Submit Ride” button, drivers see a confirmation screen that their ride has been submitted to the database (Figure 9).

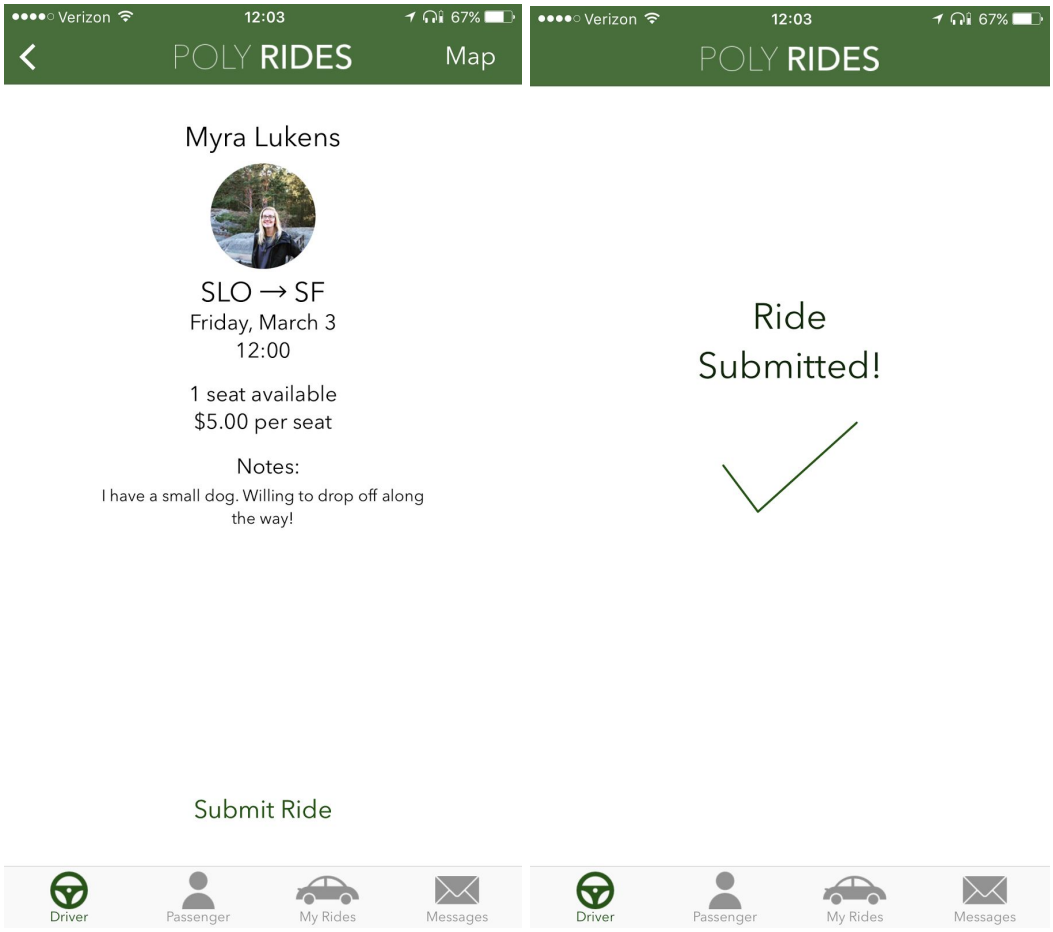


Figure 9: Poly Rides 1.0 Submit a Ride

2.3 Passenger Flow

Upon switching to the Passenger tab, users see a very similar view as the driver tab. However, the car progress icon shows only two steps. The passenger inputs their start and end locations, presses the “Find Me a Ride” button, and then inputs their desired date and time of departure (Figure 10).

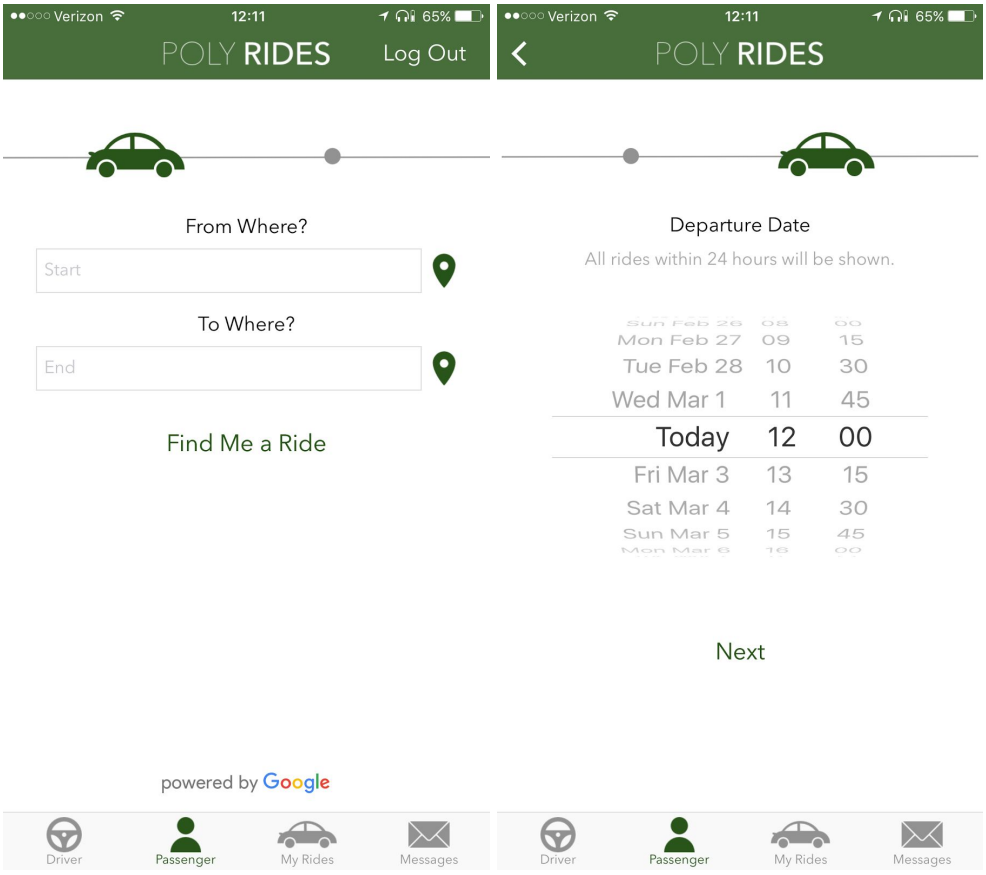


Figure 10: Poly Rides 1.0 Passenger View

Upon tapping the “Next” button, the passenger sees a TableView of the available rides within 24 hours of their desired date and time, sorted by proximity to the desired route. They can tap on one of these rides and see the details the driver provided, including Facebook mutual friends (Figure 11).

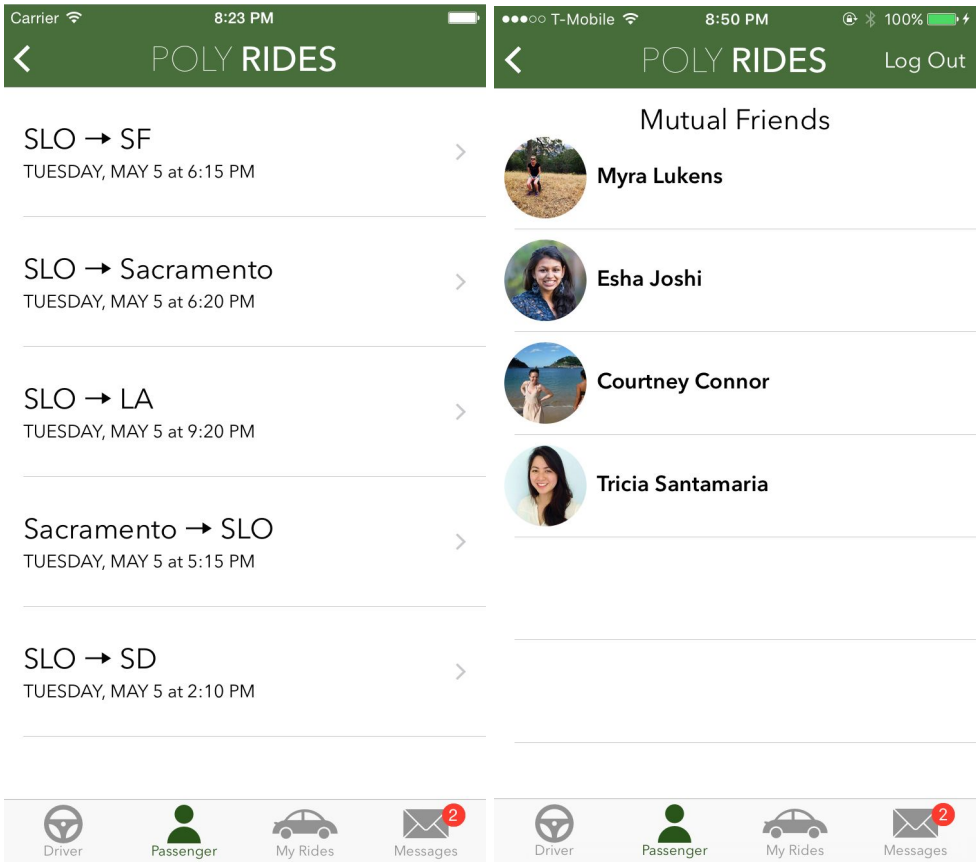


Figure 11: Poly Rides 1.0 Find a Ride

2.4 Messaging

Passengers are able to initiate a message thread with a driver to discuss logistics of the ride. Drivers are given push notifications upon receiving messages, and a badge icon on the "Messages" tab bar item (Figure 12).

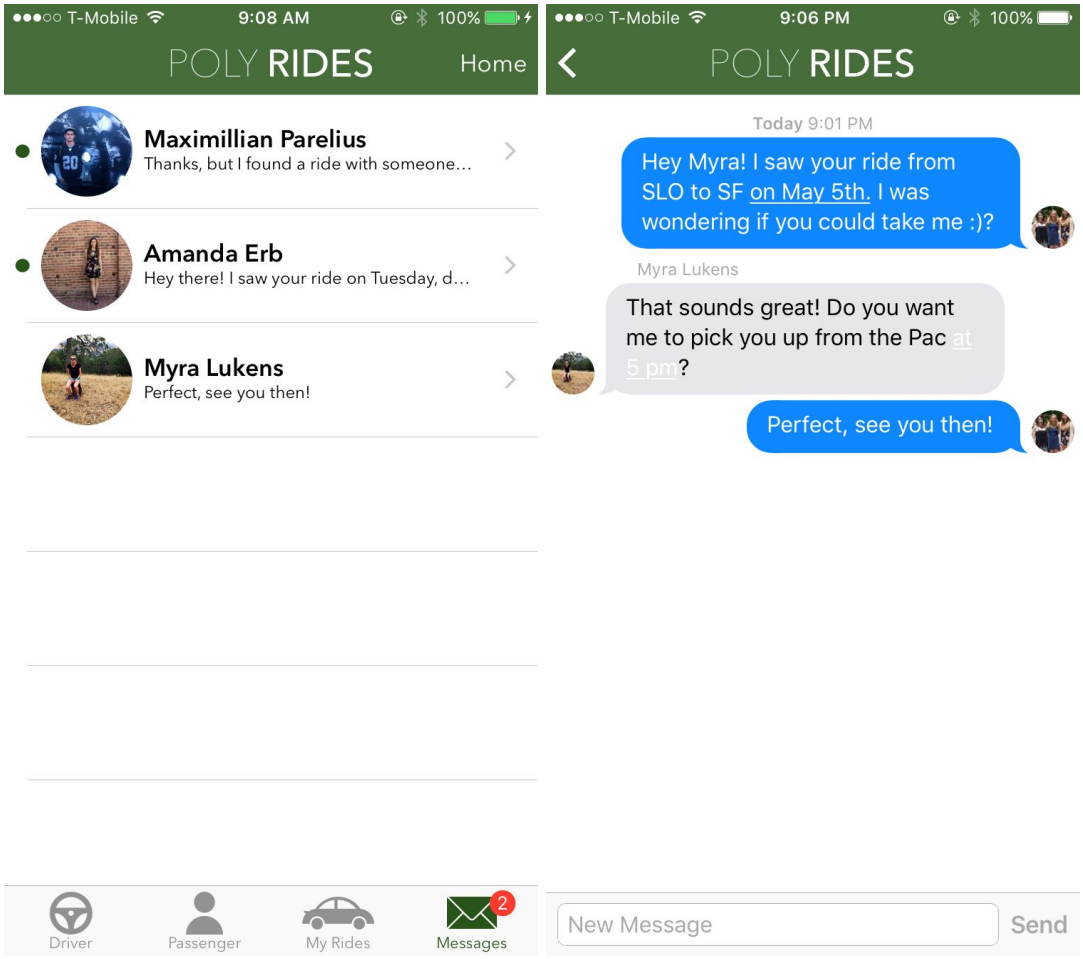


Figure 12: Poly Rides 1.0 Messaging

3.0 Poly Rides 1.0 Release

Vanessa and I released Poly Rides 1.0 to the campus in Fall 2015. We received significant campus attention, validating that the need we addressed was real. In November 2015 Vanessa and I entered into the Center for Innovation and Entrepreneurship Elevator Pitch Competition. I gave the pitch, and we walked away with Audience Choice award. We used the money we received from this award to promote our application. We created posters, flyers, promoted it in our classes, and talked with various campus entities (see Figure 13 for a promotional photoshoot). University Police Department was interested in publicizing our application to

incoming students and their parents, so we worked with them to further promote it at Open House. We contacted the administrators on the Cal Poly Ride Share Facebook group and managed to work with them to promote our application on the group. The publicity we garnered attracted the attention of the campus newspaper, and we were featured in a Mustang News article.

Figure 13: Poly Rides 1.0 Promotion

4.0 Poly Rides 1.0 to Poly Rides 2.0

We received a lot of feedback about Poly Rides 1.0. With the deprecation of Parse, our database provider, Vanessa and I decided to work on a new version of the app. Because a large number of users rely on Poly Rides 1.0, we needed to continue maintenance on the app.

Vanessa worked on Poly Rides 2.0 and handed it to me in January 2017 to complete. She handed me the app written in Swift 2.0, but with the advent of Swift 3.0 there were many bug fixes and refactorings necessary. Much of my time spent was in this realm, with some feature additions mixed in. In her thesis[9], Vanessa Forney conducted a survey provided through the Cal Poly class pages on Facebook as well as the Cal Poly Rideshare page to gather the key features determined for Poly Rides 2.0. Overall, there were 190 responses. The new version of Poly Rides includes Cal Poly email verification and an improved ride search experience. One of the main differences is an improved ride search experience by grouping rides by common regions that Cal Poly students visit. Other main features added for version 2.0 is Cal Poly email verification, requesting and accepting ride requests, and an improved user interface. We removed the in-app messaging functionality of Poly Rides 1.0 and instead require a phone number for texting communication.

5.0 Application Overview: Poly Rides 2.0

5.1 Logging In

Users log in with their Facebook credentials (Figure 14). We continued use of Facebook login for version 2.0 because of the affordance of users having pre-populated information such as picture and name in their account. Additionally, user safety is a huge priority for Poly Rides 2.0, and ensuring that a user has a Facebook account adds validation of their identity.

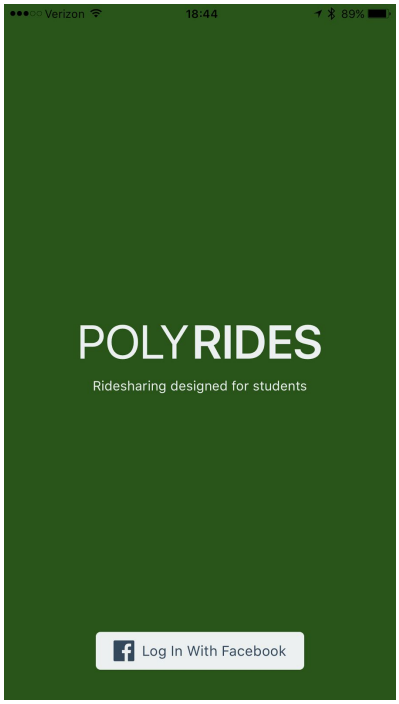


Figure 14: Poly Rides 2.0 Log In

5.2 Profile

Poly Rides 2.0 has an improved profile from version 1.0 (Figure 15). The user enters contact information like their phone number, and car information such as the make, model, year, and color of their car (Figure 16). This was an improvement that was identified from version 1.0. It streamlines the process of passengers identifying their ride when it has arrived.

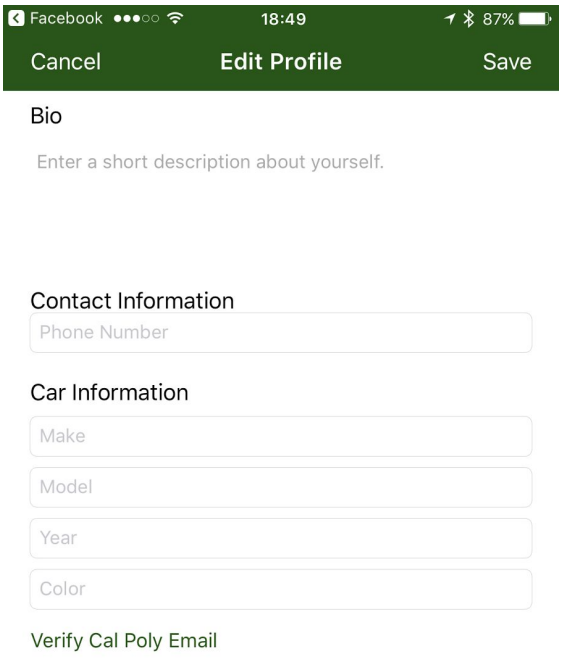


Figure 15: Poly Rides 1.0 Edit Profile

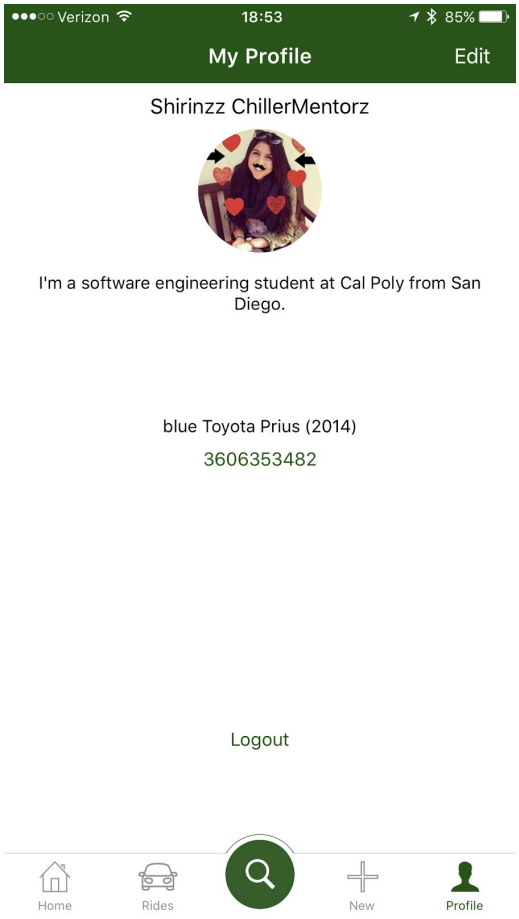
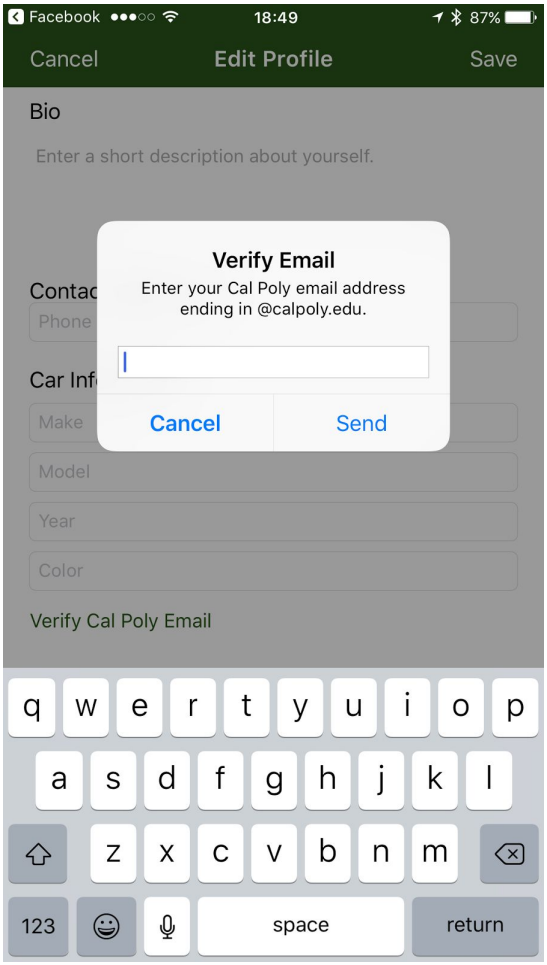


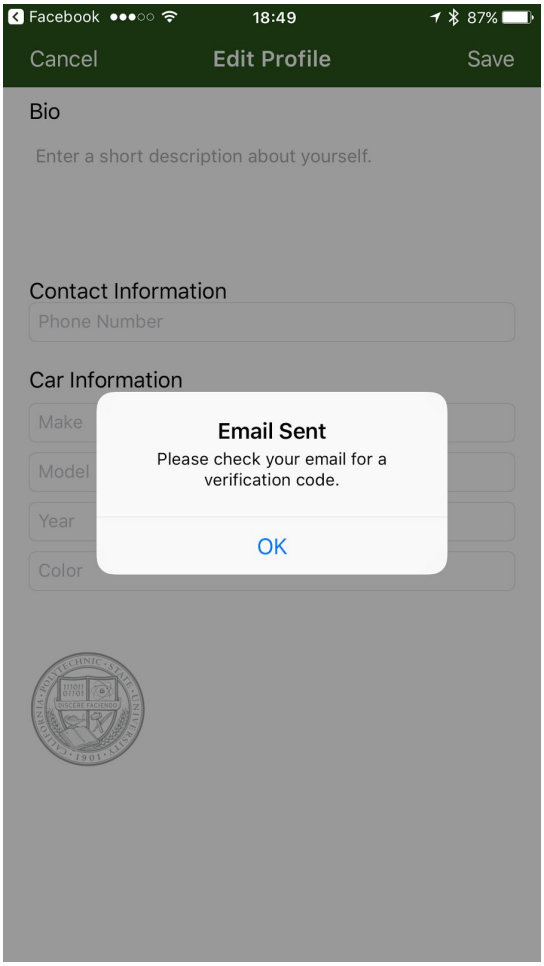
Figure 16: Poly Rides 2.0 Profile

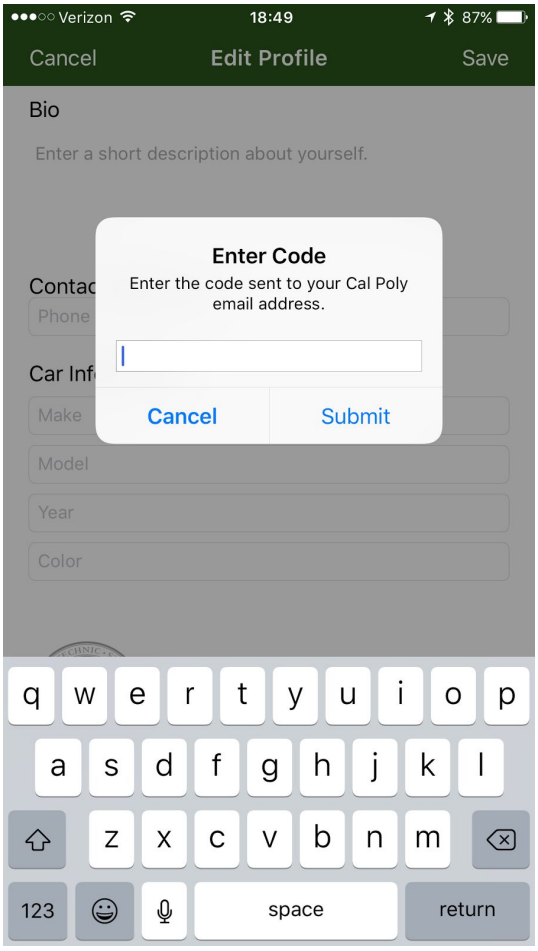
5.2.1 Email Verification

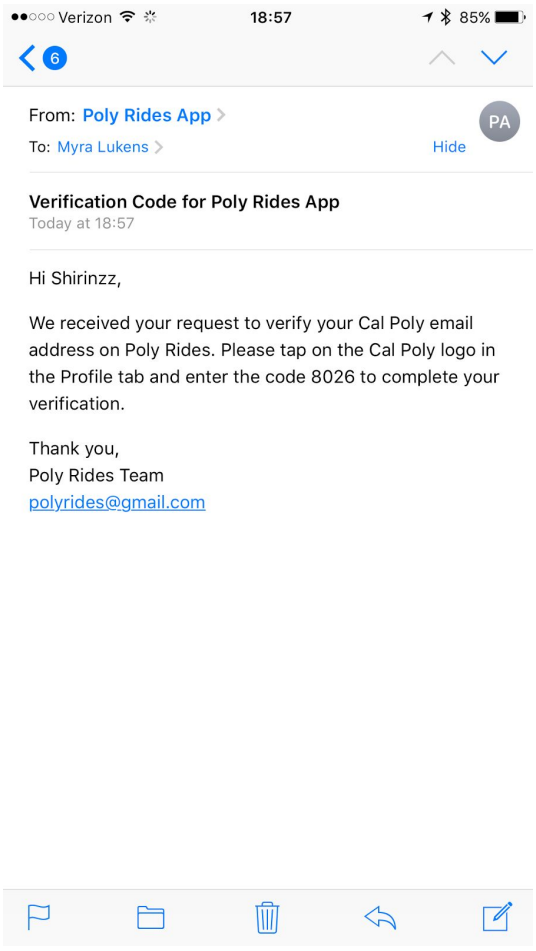
Email verification with SendGrid[9] was added from version 1.0 to 2.0 (Figure 17). Vanessa implemented the initial verification, but with the conversion from Swift 2.0 to 3.0 there were alterations needed. SendGrid proves a simple API that allows a user to enter their Cal Poly email address and receive an email with a verification code that they can then enter on the app to confirm their identity. Upon verification, a driver receives a special “badge,” that passengers can see, with the goal of incentivizing drivers to verify their emails to add extra security to the ride.

[9] "Marketing & Transactional Email Service." *SendGrid*. N.p., n.d. Web. 22 Mar. 2017.


Facebook
18:49
87%
Cancel
Edit Profile
Save
Bio
Enter a short description about yourself.
Verify Email
Enter your Cal Poly email address ending in @calpoly.edu.
Cancel
Send
Phone
Make
Model
Year
Color
Verify Cal Poly Email
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return


Facebook
18:49
87%
Cancel
Edit Profile
Save
Bio
Enter a short description about yourself.
Contact Information
Phone Number
Car Information
Make
Model
Year
Color
Email Sent
Please check your email for a verification code.
OK


Verizon
18:49
87%
Cancel
Edit Profile
Save
Bio
Enter a short description about yourself.
Enter Code
Enter the code sent to your Cal Poly email address.
Cancel
Submit
Phone
Make
Model
Year
Color
space
return


Verizon
18:57
85%
From: Poly Rides App
To: Myra Lukens
Hide
PA
Verification Code for Poly Rides App
Today at 18:57
Hi Shirinzz,
We received your request to verify your Cal Poly email address on Poly Rides. Please tap on the Cal Poly logo in the Profile tab and enter the code 8026 to complete your verification.
Thank you,
Poly Rides Team
polyrides@gmail.com

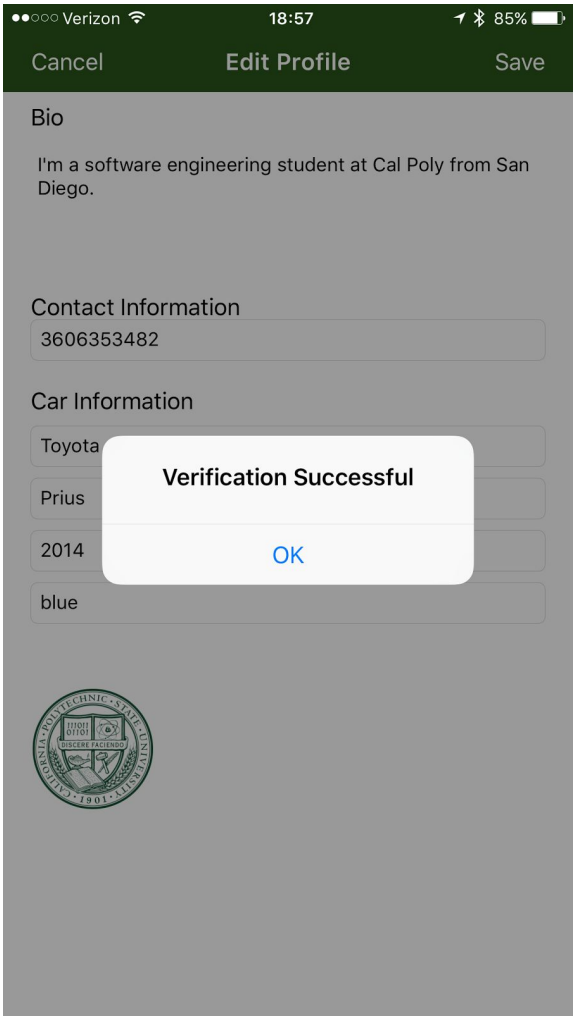


Figure 17: Poly Rides 2.0 Verify Email

5.3 Driver Flow

5.3.1 New Ride

A driver begins posting a new ride by selecting the “New” tab on the home page. They are shown a screen where they input key ride data such as “From,” “To,” date/time, “Seats Available,” “Cost/seat,” and optional notes for their passengers (Figure 18). There is robust error checking so that invalid rides cannot be posted.

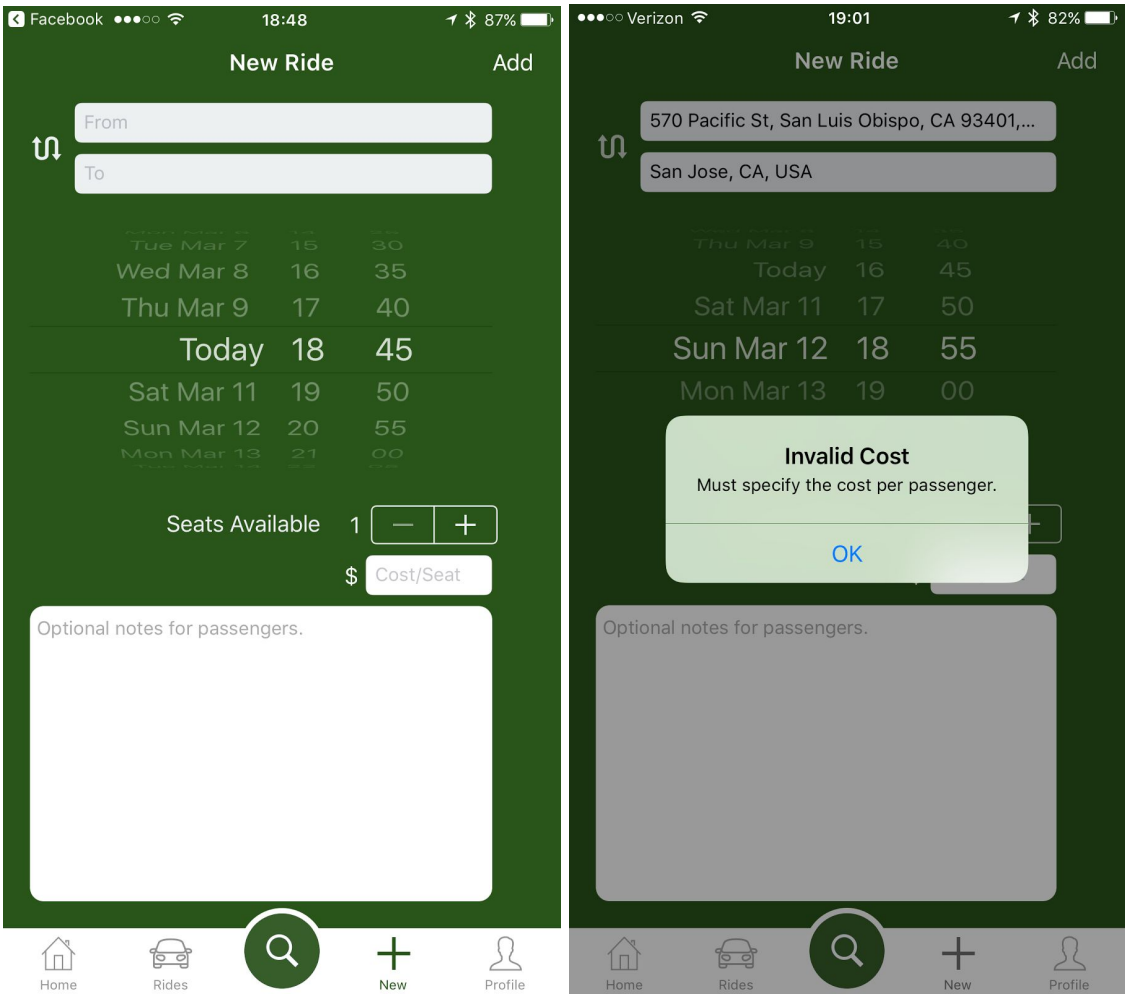


Figure 18: Poly Rides 2.0 New Ride

Upon tapping the “Add” bar button item, they are taken to a screen confirming that their ride was submitted (Figure 19). It has been added to the Firebase database, and will appear on searches immediately.

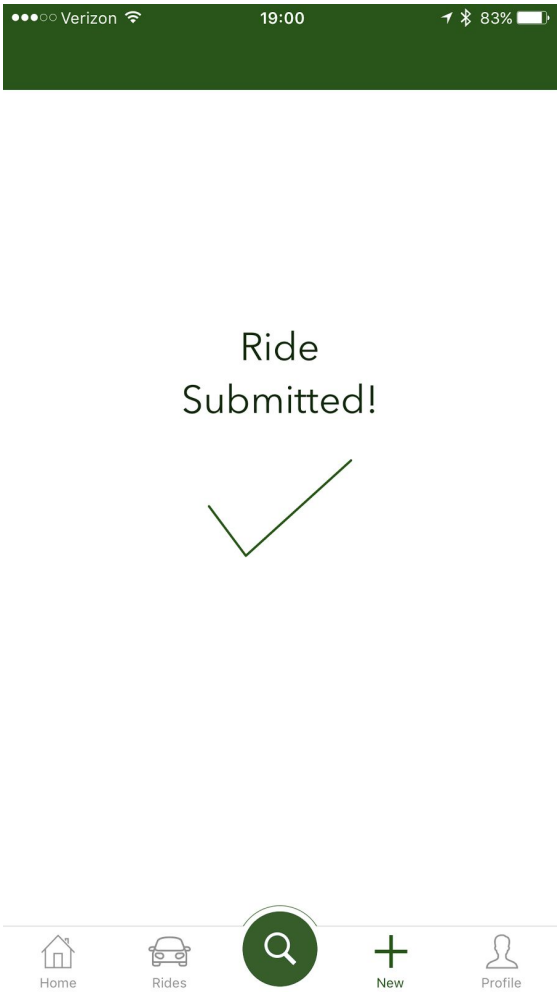
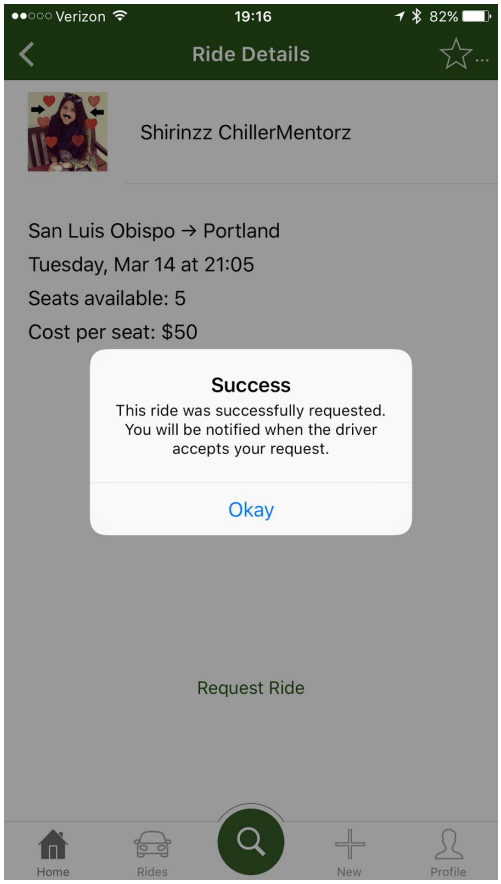
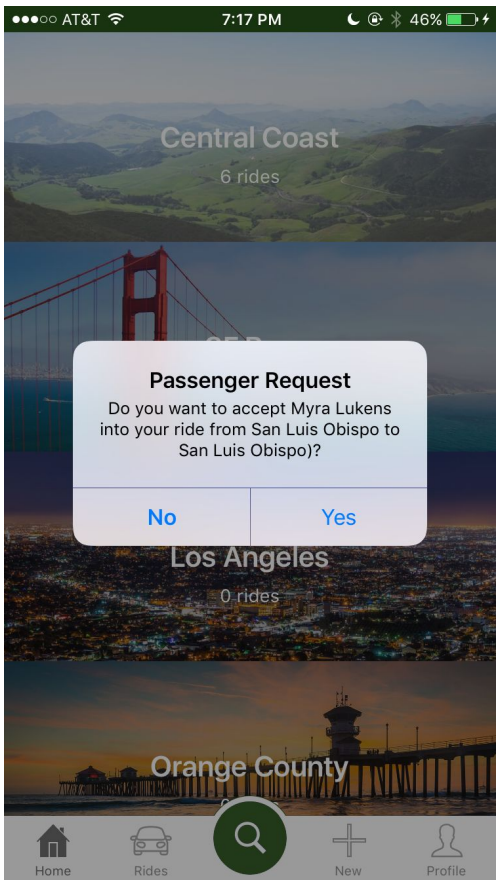


Figure 19: Poly Rides 2.0 Ride Submitted

5.3.2 Accept/Reject Passengers

One of the main undertakings for this project was adding functionality for passengers requesting rides and drivers accepting or rejecting ride requests. The underlying design decisions surrounding this functionality will be discussed further in the Design section. When a passenger requests a ride, the driver is notified of their request via a notification (Figure 20). They can choose to either accept or reject this passenger into their ride. Upon accepting their request, the number of available seats on the ride is decremented. If they deny the request, the seats available is not changed.


Verizon
19:16
82%
Ride Details
Shirinzz ChillerMentorz
San Luis Obispo → Portland
Tuesday, Mar 14 at 21:05
Seats available: 5
Cost per seat: $50
Success
This ride was successfully requested. You will be notified when the driver accepts your request.
Okay
Request Ride
Home
Rides
New
Profile


AT&T
7:17 PM
46%
Central Coast
6 rides
Passenger Request
Do you want to accept Myra Lukens into your ride from San Luis Obispo to San Luis Obispo)?
No
Yes
Los Angeles
0 rides
Orange County
Home
Rides
New
Profile

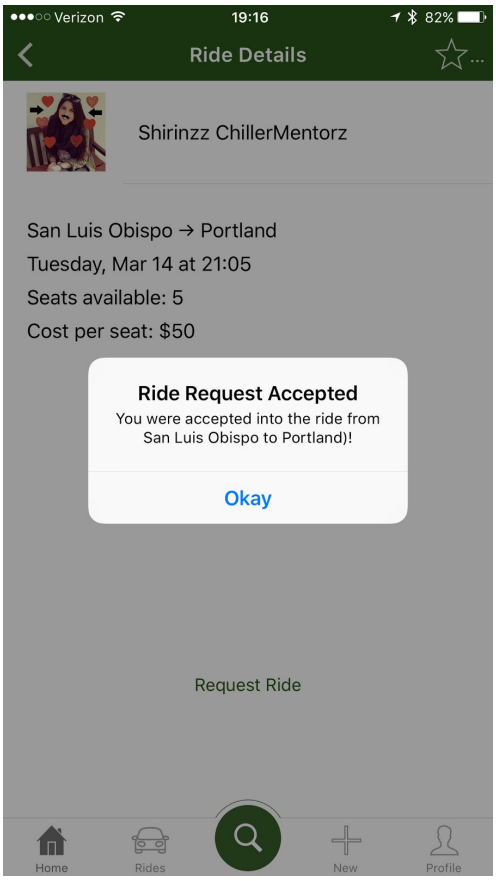
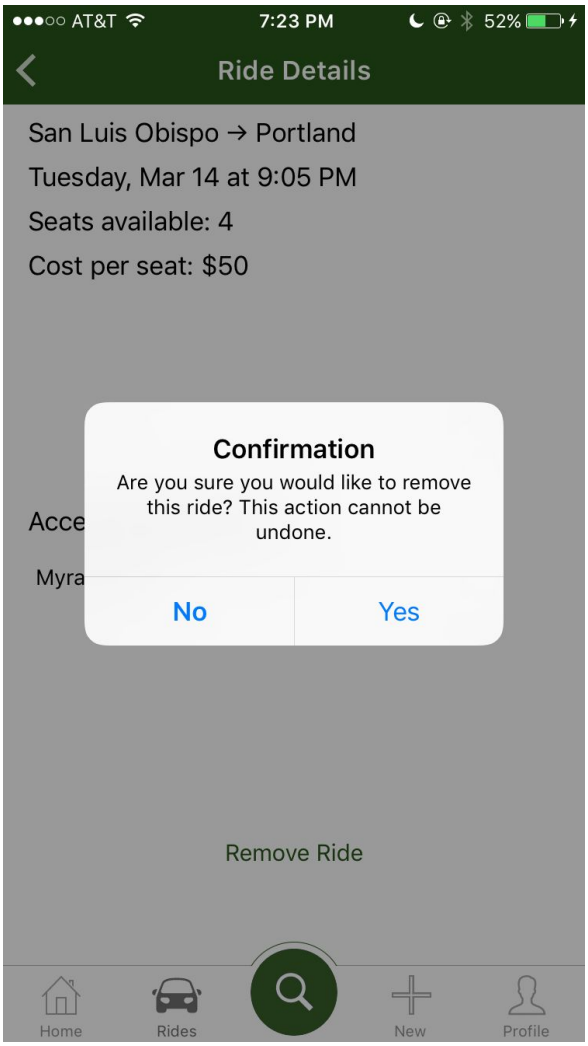
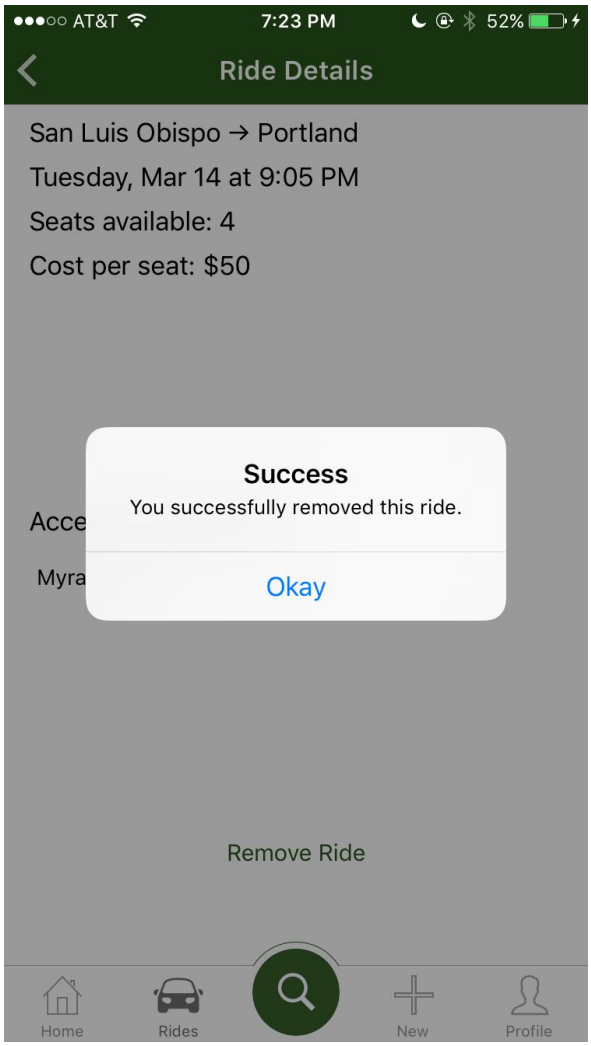


Figure 20: Poly Rides 2.0 Ride Request

5.3.3 Remove a Ride

A driver can remove a ride they are offering via the “Rides” tab. They select the ride and can tap the “Remove Ride” button (Figure 21). There is a confirmation dialogue, because if the driver removes the ride, it cannot be undone. Once they confirm the removal, if there are any passengers, it will notify them that the ride has been removed and it is immediately removed from the database.


AT&T
7:23 PM
52%
Ride Details
San Luis Obispo → Portland
Tuesday, Mar 14 at 9:05 PM
Seats available: 4
Cost per seat: $50
Confirmation
Are you sure you would like to remove this ride? This action cannot be undone.
No
Yes
Remove Ride
Home
Rides
New
Profile


AT&T
7:23 PM
52%
Ride Details
San Luis Obispo → Portland
Tuesday, Mar 14 at 9:05 PM
Seats available: 4
Cost per seat: $50
Success
You successfully removed this ride.
Okay
Remove Ride
Home
Rides
New
Profile

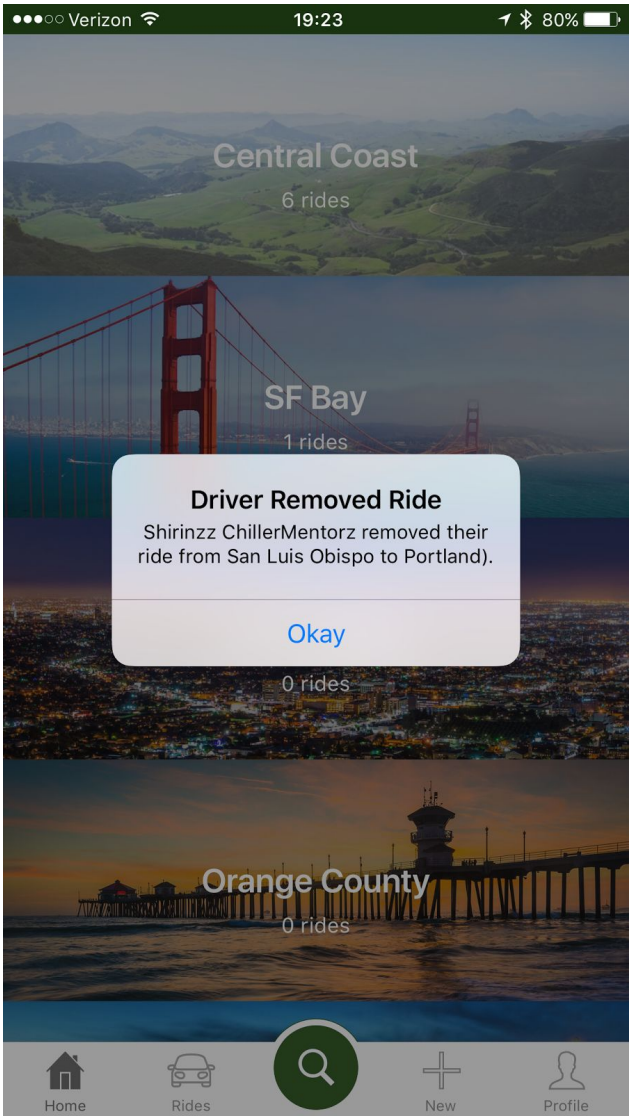


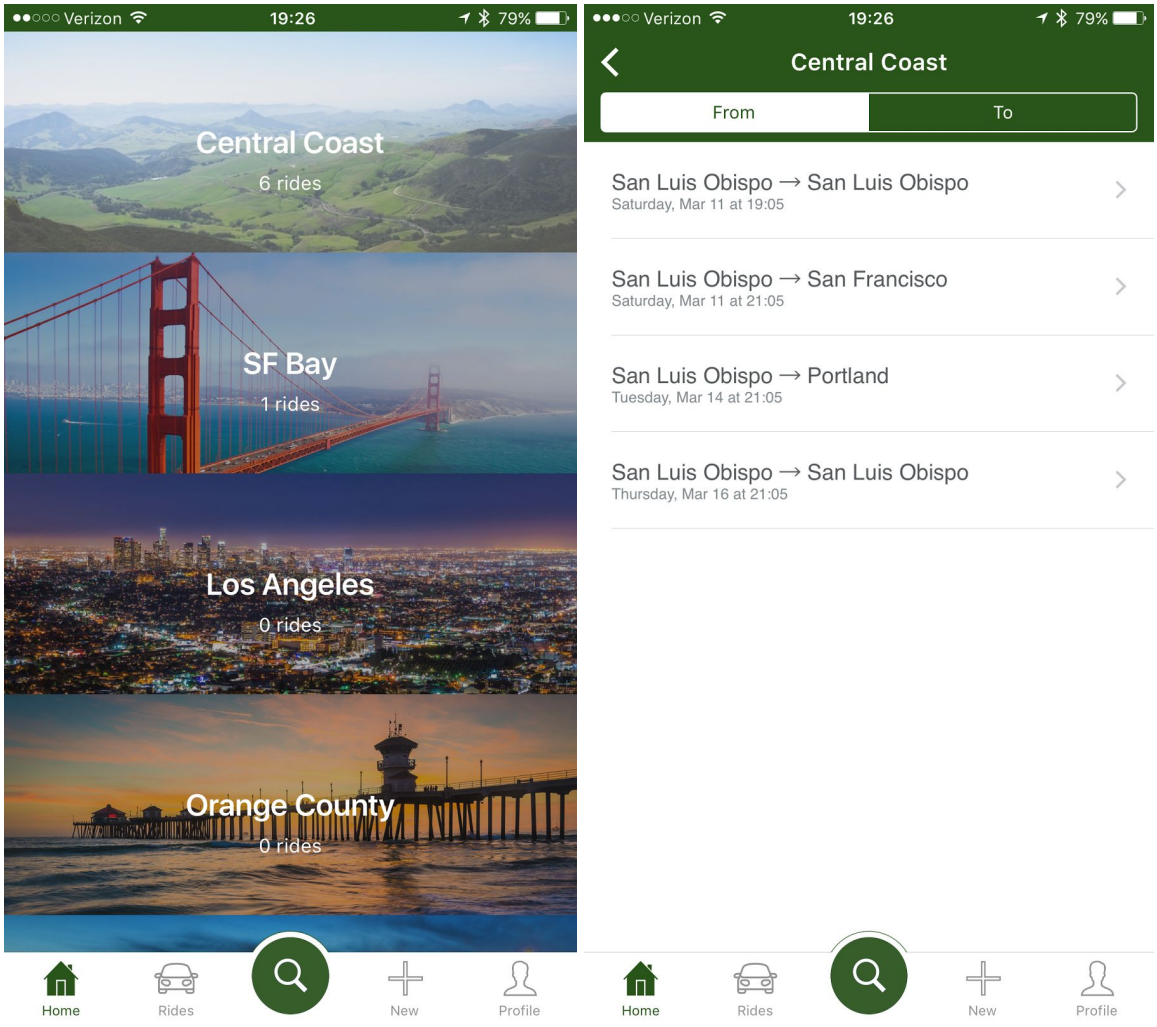
Figure 21: Poly Rides 2.0 Driver Remove Ride

5.4 Passenger Flow

5.4.1 Browse Rides

One main design decision for Poly Ride 2.0 was the redesign of ride browsing. In version 1.0, there was really no way to browse rides, instead, you had to search for a specific ride. We received feedback that browsing rides would be a useful feature. In the new version, we introduced the concept of “regions.” On the homepage, a user sees a TableView of regions of

rides (Figure 22). If they select a region, rides within that region are shown. A user can select between rides from or to that region with the Segmented Control. The rides are sorted based on date/time, with the soonest ones showing first. When they select a ride, they are shown the ride details view, which contains relevant information about the ride, such as the driver, key information, and a button to “Request Ride.” In the upper right corner is a star to save the ride. If the user taps on the driver, they are shown the driver profile, and can see whether the driver is Cal Poly verified. Additionally, they can see Facebook mutual friends that use Poly Rides.



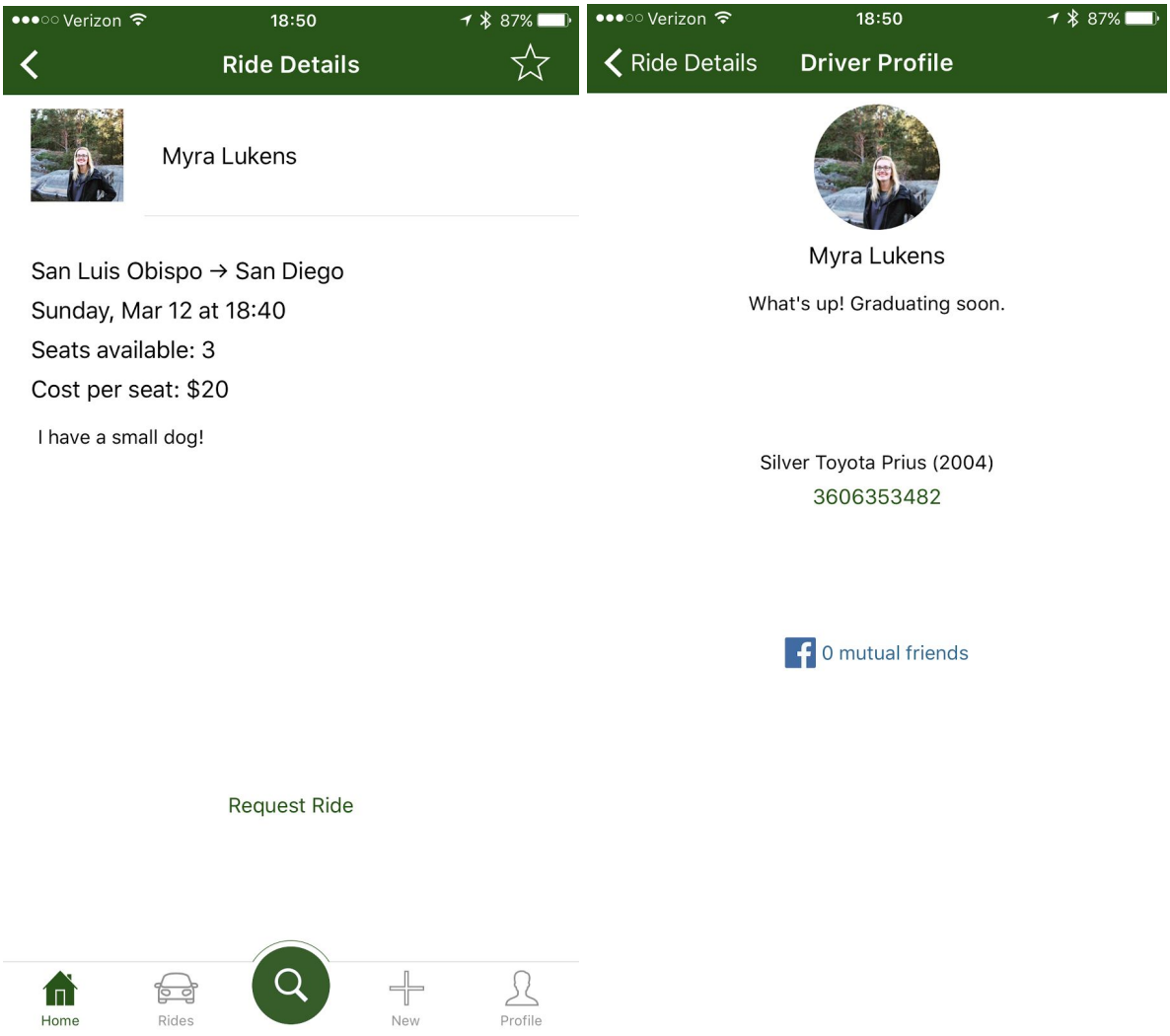


Figure 22: Poly Rides 2.0 Browse Rides

5.4.2 Search for a Ride

Poly Rides 2.0 kept its ride search feature and algorithm from version 1.0, with some user interface changes. The user can tap the search button on the homepage and is taken to a new view where they enter “From,” “To,” and a date/time (Figure 23). Rides within 24 hours are shown, sorted by proximity to the start and end locations. A user can tap on the ride and see the same ride details view as in the browsing rides flow.

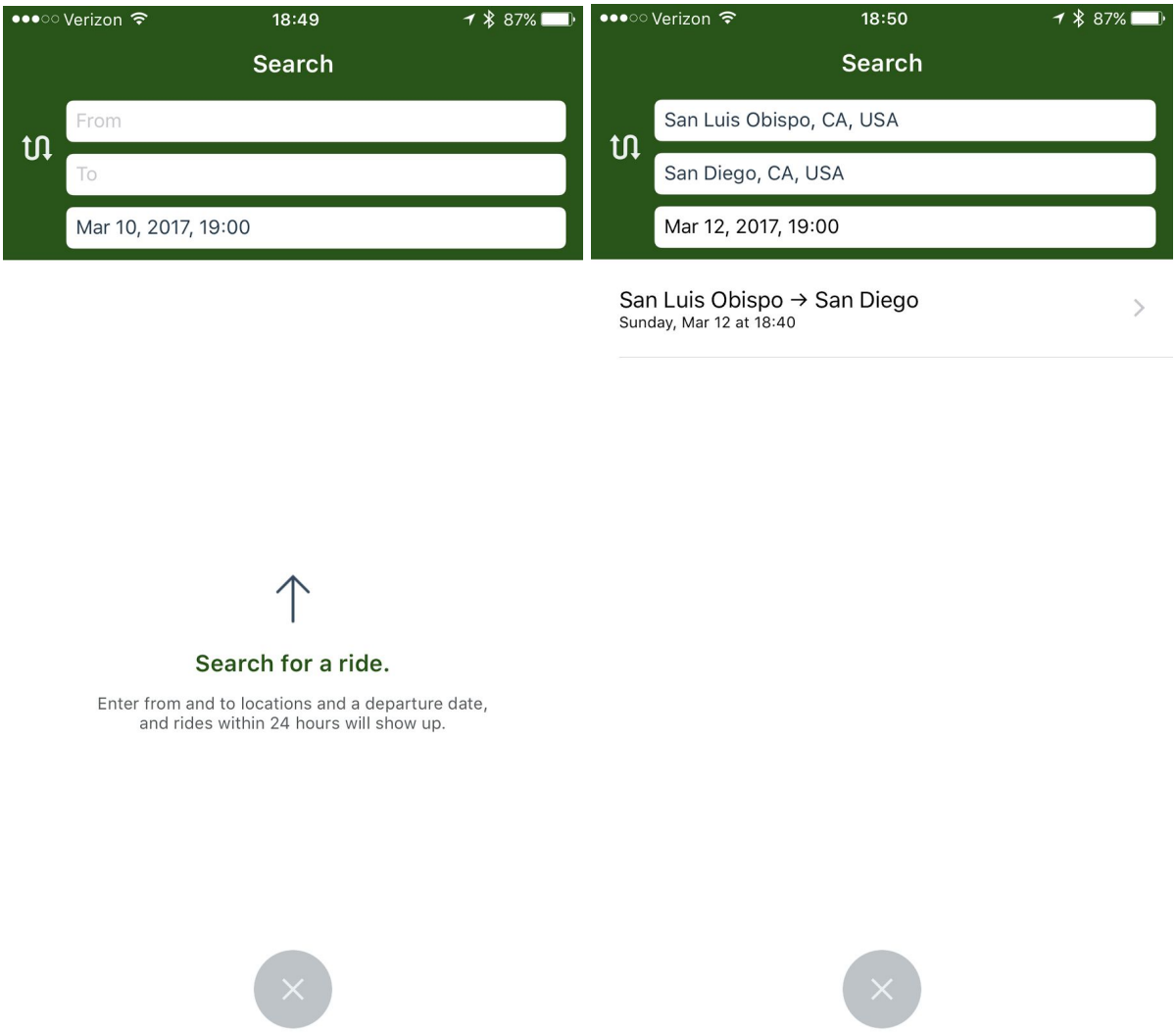


Figure 23: Poly Rides 2.0 Search for Ride

5.4.3 Save a Ride

Passengers are given the option to save a ride, which adds the ride to a section of their Rides page, discussed in section 5.4.8. This is a new feature that allows passengers to return to rides they are interested in at a later time. A user can remove the ride from their saved rides, and is presented a confirmation dialog that this is their desired action (Figure 24).

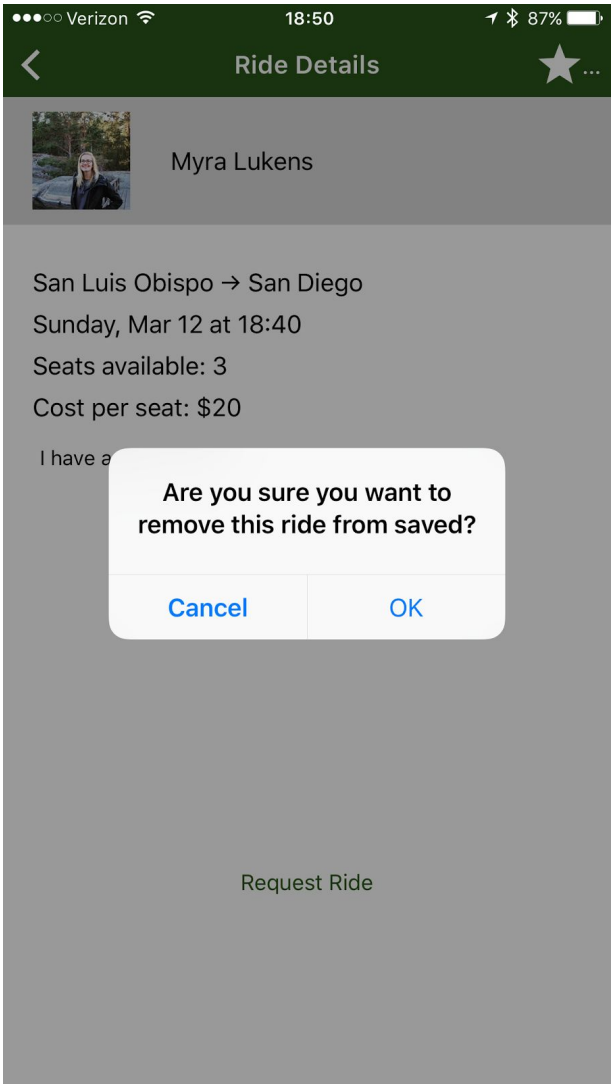


Figure 24: Poly Rides 2.0 Remove Saved Ride

5.4.4 Request a Ride

When a passenger taps the “Request Ride” button, two things occur: the driver is notified that the passenger would like to be in their ride, and the passenger is presented a confirmation that the ride was requested. When the driver accepts or denies their ride request, the passenger is informed via a notification. If the driver accepts their ride request, it is added into the passenger’s Rides view, which is discussed next.

5.4.5 Leave Ride

A passenger can leave a ride they are in by tapping the “Leave Ride” button on the ride (Figure 25). The ride can be found in their Rides tab. They are presented a confirmation screen that they in fact wish to leave the ride. If they confirm the action, the driver will be notified and this will be added to the ride’s available seats.

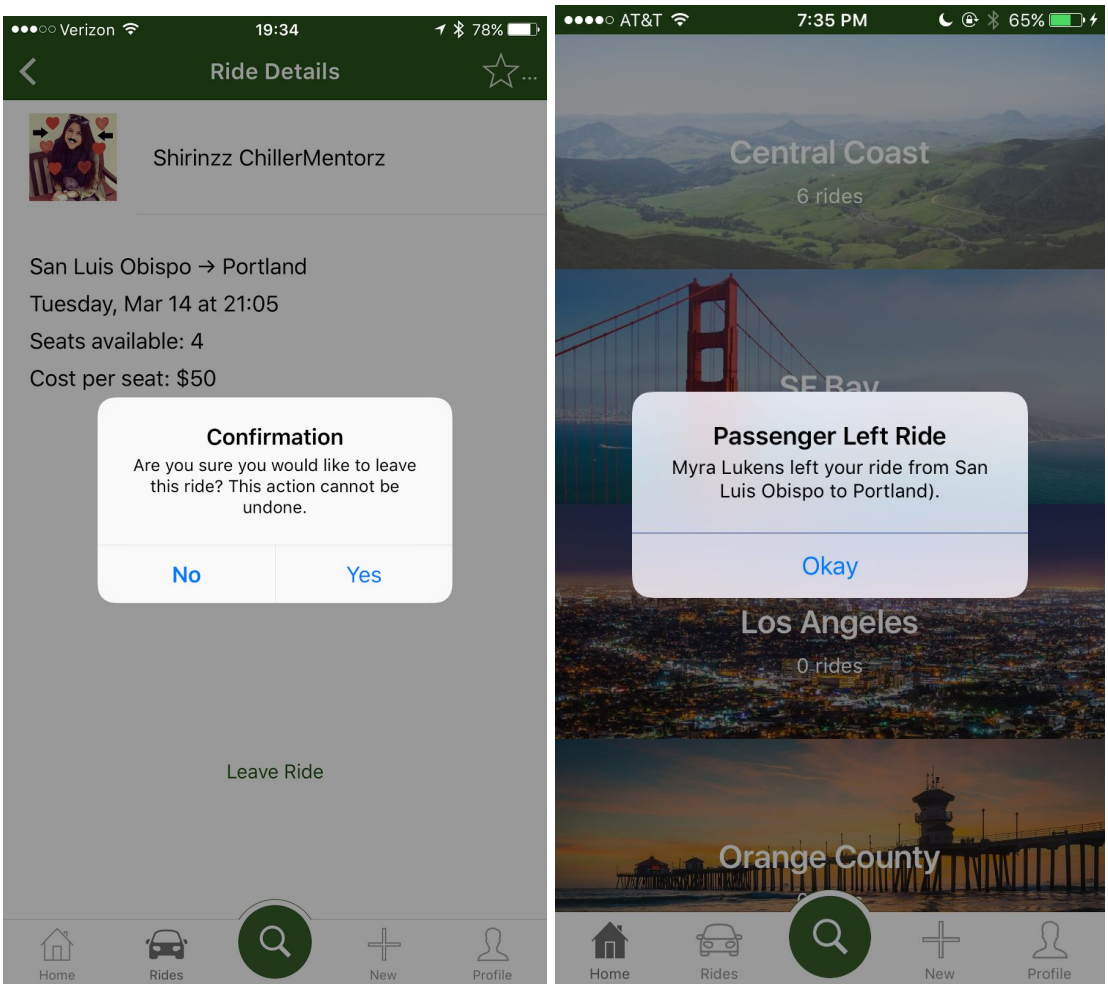


Figure 25: Poly Rides 2.0 Leave Ride

5.4.6 Rides

The Rides page is the home for rides that a user is going on. It allows them to manage their ride schedule, and provides faster access to rides that are important to them.

5.4.7 Current/Past Rides

A user can see their current rides in the “Current” tab (Figure 26). If the user is the driver of this ride, a wheel icon appears in the table view cell. The rides are sorted by date. Past rides are shown in the “Past” tab. Current rides can be left/removed, but past rides cannot. If a driver selects a ride they are offering, they see the list of passengers they have accepted into the ride.

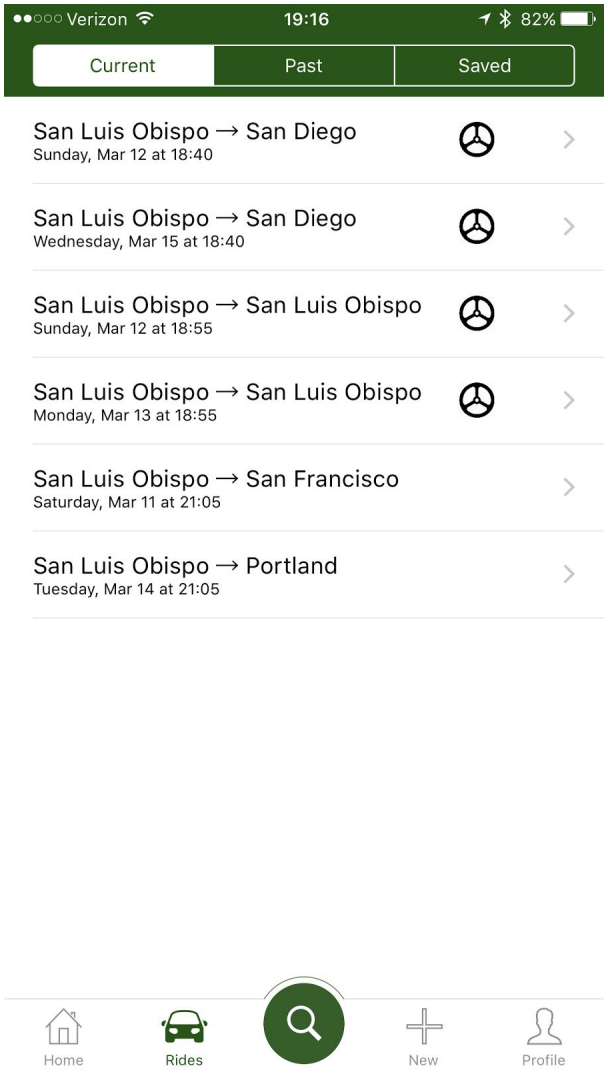


Figure 26: Poly Rides 2.0 Rides

5.4.8 Saved Rides

Saved rides are shown in the third tab, providing quick access to rides that a passenger found particularly interesting.

6.0 Design

6.1 Software Architecture

This section describes the software architecture of the two versions of Poly Rides. From the first version to the second, there were many design decisions and changes.

6.1.1 Deployment Diagram

The deployment diagrams for both versions of Poly Rides show the distribution of software artifacts to deployment targets. This includes servers and services like APIs and the user's physical device.

6.1.1.1 Poly Rides 1.0

Poly Rides 1.0 was deployed via an application on a mobile device. It used Parse database as its backend, and also interacted with Google Maps and Facebook servers. See Figure 27 for details.

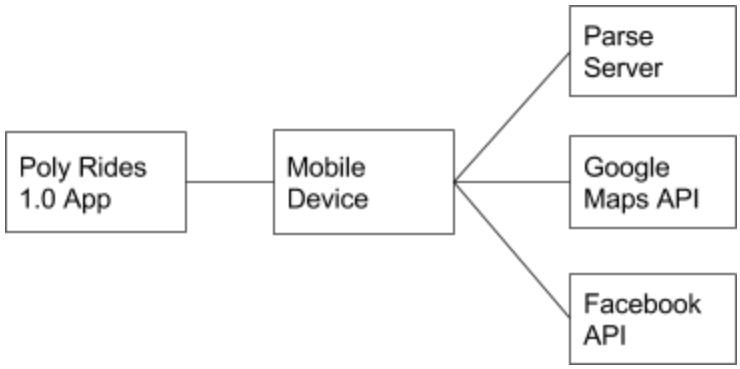


Figure 27: Poly Rides 1.0 Deployment Diagram

6.1.1.2 Poly Rides 2.0

Different services were used for Poly Rides 2.0. The database used was Firebase, and Google Maps, Facebook, and SendGrid servers were used. See Figure 28 for details.

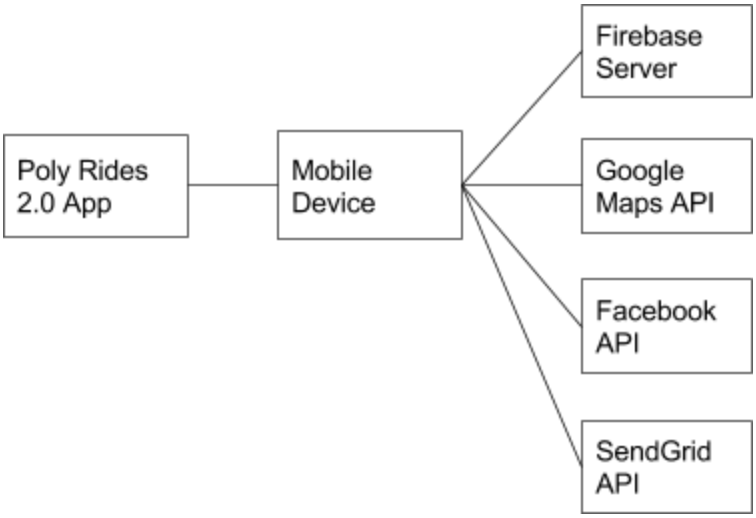


Figure 28: Poly Rides 2.0 Deployment Diagram

6.1.2 UML Diagram

6.1.2.1 Poly Rides 1.0

Figure 29 is a class diagram for Poly Rides 1.0 showing the model objects. Our application uses model view controller (MVC) principles. This means that the model classes consist of the application logic and underlying interactions. There are separate folders for the view and controller classes. The Database class handles all database interactions with our Parse database. This allows us to easily switch out our database if necessary. The Rides class handles the collection of rides, and interacts with the database via a reference to the Database class. The User class also contains a Database reference, and represents a given Poly Rides user.

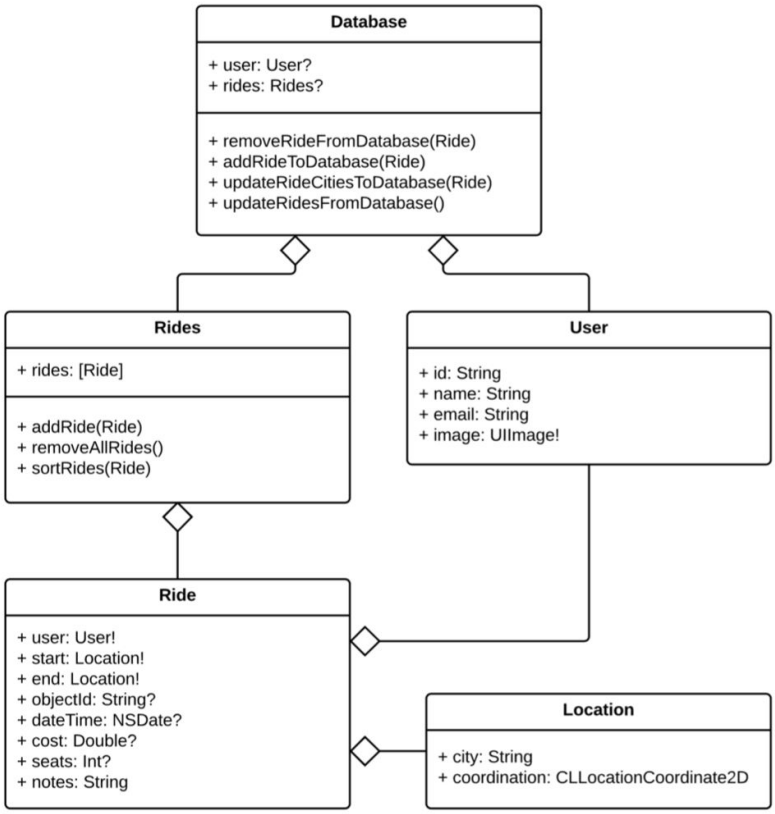


Figure 29: Poly Rides 1.0 Model Class Diagram[10]

6.1.2.2 Poly Rides 2.0

The Poly Rides 2.0 package diagram in Figure 30 shows use of MVC principles. Vanessa and I kept the views completely separate from the logic of the application, so that in the future we can update the user interface, or database, without needing to change core application logic. The Controllers package consists of individual controllers for views. The Views package contains Storyboard files for views. The Services and Utils packages contain API and database interactions. Finally, the Models package contains core application logic.

[10] Forney, Vanessa Marie. "Encouraging Development of Mobile Applications as a Service to the Community." Thesis. California Polytechnic State University, 2016. Nov. 2016. Web. 12 Mar. 2017.

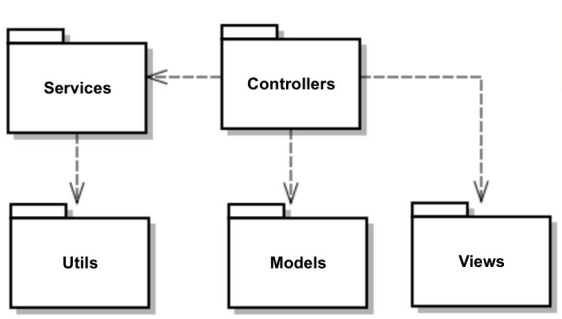


Figure 30: Poly Rides 2.0 Package Diagram

6.2 Design Decisions

6.2.1 Poly Rides 1.0

6.2.1.1 Ride Sorting Algorithm

Our ride sorting algorithm orders the rides based on their proximity to the user's desired start and end locations, with the closest appearing first (Figure 31). The proximity is determined by the smallest geographic distance between the latitude and longitude coordinates of the searched ride and rides in the database. We determined this to be the best way to order them, because our users are most often going to take the most direct route from their start to end location. Additionally, since most rides are to major cities in California or right around there, such as Los Angeles or San Francisco, it is most likely that passengers want to be dropped off near these major cities. Additionally, it is much easier for drivers to drop off passengers toward the end of their route, since it does not disrupt their trip as significantly. We used a very similar algorithm in Poly Rides 2.0.

```
rides.sortInPlace {
    let distance1 = GMSGeometryDistance(ride1.start, search.start) +
                GMSGeometryDistance(ride1.end, search.end)
    let distance2 = GMSGeometryDistance(ride2.start, search.start) +
                GMSGeometryDistance(ride2.end, search.end)
```

```
    return distance1 < distance2
}
```

Figure 31: Ride Sorting Code Snippet

6.2.2 Poly Rides 2.0

6.2.2.1 Design for request/accept rides

A main feature I added to Poly Rides 2.0 is the ability to request a ride, and accept/deny ride requests. A passenger can request a ride, notifying the driver of the request. A driver can accept or deny this request. There were various design decisions on the back-end necessary for this functionality.

Below shows the hierarchy of part of the Firebase database for Poly Rides 2.0. There is a list of rides, and each ride contains various data about the ride. There is also a list of users, with each user object containing data about a user. To implement request functionality, I added a pendingRequests field, containing a list of userID: “Passenger Name” mappings. When a user requests a ride, they are added to the pendingRequests list on the ride, and the ride is added to the pendingRequests list on the user. If a driver chooses to accept the user into their ride, the seats field on the ride is decremented, the user is removed from pendingRequests on the ride, the ride is moved from pendingRequests on the user, the user is added to the passengers list on the ride, and the ride is added to the rides list on the user.

rides
- rideID
    - costPerSeat
    - date
    - description
    - userID
    - seats
    - fromPlaceCity
    - toPlaceCity
    - …
    - pendingRequests
        - userID: “Passenger Name”

        - ...
    - passengers
        - userID: “Passenger Name”
        - ...

users
- userID
    - car
    - description
    - email
    - firstName
    - lastName
    - phoneNumber
    - …
    - rides
        - rideID: true
        - ...
    - pendingRequests
        - rideID: true
        - …

This information is used in section 5.4.6 Rides page, when a driver views a ride they are offering. The “Accepted Passengers” list shows the passengers on the ride -- the users in the passenger's field of the ride in the database. A design decision was made to have the ride contain the userID of the driver and the userID of the passengers, instead of the necessary data from those copied onto the ride object. This is because a passenger or driver could change information in their profile, so instead, the user object is queried for using the userID on the ride. However, the passengers list contains mappings of userID: “Passenger Name,” because users log in with Facebook and cannot change their profile name for security reasons. Directly storing the passenger name instead of having to query the database for the user object and then get it off of that object results in greater efficiency. Figure 32 shows the code for these database actions.

```
func addPassengerToRideRequests(user: User?, ride: Ride) {
   if let id = user?.id {
     if let rideId = ride.id {
       // add to ride's current pending requests
       var requestedRef = ref.child("rides/\(rideId)/pendingRequests/\(id)")
```

```
      requestedRef.setValue(user?.getFullName())

      // add to passenger's requested rides
      requestedRef = ref.child("users/\(id)/pendingRequests/\(rideId)")
      requestedRef.setValue(true)
    }
  }
}

func acceptPassengerIntoRide(passengerId: String, passengerName: String, rideId: String) {
  // remove user from pending requests
  ref.child("rides/\(rideId)/pendingRequests/\(passengerId)").removeValue()

  // put user in the passengers of the ride
  var rideRef = ref.child("rides/\(rideId)/passengers/\(passengerId)")
  rideRef.setValue(passengerName)

  let ridesRef = ref.child("rides/\(rideId)")
  let query = ridesRef.queryOrderedByKey()

  query.observe(.childAdded, with: { snapshot in
    let ride = Ride(fromSnapshot: snapshot)
    ride.passengers.updateValue(passengerName, forKey: passengerId)
  })

  // decrement the seats available
  rideRef = ref.child("rides/\(rideId)/seatsAvailable")
  rideRef.observeSingleEvent(of: .value, with: { (snapshot) in
    let seats = (snapshot.value as? NSNumber)!.intValue - 1
    rideRef.setValue(seats)
  }) { (error) in
    print(error.localizedDescription)
  }

  // remove from passenger's pending requests
  ref.child("users/\(passengerId)/pendingRequests/\(rideId)").removeValue()

  // add the ride to the passenger's rides
  let passRef = ref.child("users/\(passengerId)/rides/\(rideId)")
  passRef.setValue(true)
}

func doNotAcceptPassengerIntoRide(passengerId: String, rideId: String) {
  // remove user from pending requests
  ref.child("rides/\(rideId)/pendingRequests/\(passengerId)").removeValue()

  // remove from passenger's pending requests
  ref.child("users/\(passengerId)/pendingRequests/\(rideId)").removeValue()
```

```
}
```

Figure 32: Database Code Snippets

6.2.2.2 Refreshing Data

In Poly Rides 1.0, Vanessa and I made many decisions about refreshing data that we later regretted. There is a tradeoff between keeping the app perfectly in sync with the database and providing a positive user experience. In making data refreshing decisions for Poly Rides 2.0, I took these learning experiences into account, and looked back on the mistakes made in version 1.0.

When a user is in the Rides view, and removes or leaves a ride, they expect to see that ride immediately removed from their list. Additionally, if they are accepted into a ride, they should see it immediately appear. This is why for the Rides view, there is dynamic refreshing of data (Figure 33). This is accomplished by using Firebase queries to observe on .childAdded and .childRemoved. Every time an item is added or removed to that part of the database, a method is called. In this case, we wanted to be able to monitor the user's rides, so that their Rides page is kept up-to-date. It can be reasonably assumed that the user's rides will not change that frequently, so this is a reasonable use of dynamically loading data. The delegate methods onRideReceived and onRideRemoved simply add or remove these rides from the Rides tab.

```
func monitorRidesForUser(user: User) {
    if let userId = user.id {
      let ridesRef = ref.child("users/\(userId)/rides")
      ridesRef.observe(.childAdded, with: { snapshot in
        self.delegate?.onNumRidesReceived(numRides: 1)

        let rideRef = self.ref.child("rides/\(snapshot.key)")
        rideRef.observeSingleEvent(of: .value, with: { snapshot in
          let ride = Ride(fromSnapshot: snapshot)
          self.delegate?.onRideReceived(ride: ride)
        })
      })
```

```
    ridesRef.observe(.childRemoved, with: { snapshot in
      self.delegate?.onNumRidesReceived(numRides: 1)

      let rideRef = self.ref.child("rides/\(snapshot.key)")
      rideRef.observeSingleEvent(of: .value, with: { snapshot in
        let ride = Ride(fromSnapshot: snapshot)
        self.delegate?.onRideRemoved(ride: ride)
      })
    })
  }
}
```

Figure 33: Monitor Rides Code Snippet

Rides are refreshed upon initial loading of the application. The homepage contains all of the rides to/from specific regions. A user would expect that this page would accurately reflect the rides in the database at that given time. However, with many users on Poly Rides, rides could be added or removed relatively frequently, possibly while a user is looking at the available rides. Monitoring all of the rides in the same way we monitor rides for the user could potentially have a negative user experience, such as TableViews dynamically changing, or a slow application. I made the design decision to have the homepage table refresh when you pull down on the TableView, calling the getAllRides method (Figure 34). This is a common gesture, so common that Apple has its own UIRefreshControl for iOS10 to accomplish this very functionality. This allows users to control when their data is refreshed.

```
func getAllRides() {
  let currentDateMillis = NSDate().timeIntervalSince1970
  let ridesRef = ref.child(
    "rides")
  let query = ridesRef.queryOrdered(byChild: "date").queryStarting(atValue: currentDateMillis)

  query.observeSingleEvent(of: .value, with: { snapshot in
    self.delegate?.onNumRidesReceived(numRides: snapshot.children.allObjects.count)

    if let children = snapshot.children.allObjects as? [FIRDataSnapshot] {
      for child in children {
```

```
        let ride = Ride(fromSnapshot: child)

        if let driverId = ride.driver?.id {
          let driverRef = self.ref.child("users/\(driverId)")
          driverRef.observeSingleEvent(of: .value, with: { snapshot in
            if let driver = ride.driver {
              driver.updateFromSnapshot(snapshot: snapshot)
              self.delegate?.onRideReceived(ride: ride)
            }
          })
        }
      }
    }
  })
}
```

Figure 34: Get Rides Code Snippet

7.0 Remaining Work

Poly Rides 2.0 has some more work required before its release to the Apple App Store. Since I was developing by myself, I need to more thoroughly test ride request/accept functionality with another device. It is difficult to simulate real-world scenarios with only one developer and device. This will be discussed further in section 8.3. Additionally, I implemented push notifications via Firebase, and it is working while the app is in the foreground, but not in the background. This requires some more investigation. It would be nice for the driver to be able to view pending requests in the application. As of now, taking action on the notification is the only way a passenger can be accepted into a ride.

Vanessa and I have discussed the next steps for Poly Rides 2.0, and we plan to release a stripped-down version of it to the App Store. This will be a version with the features that replace those in version 1.0, but with our improved user interface. Then, the two of us will work together on further testing of the new, completed features, and release that as an update to the

application. We will seek help from a Cal Poly student desiring some work experience to convert the Android version to the Firebase backend.

8.0 Challenges and Lessons

8.1 Swift 2.0 to 3.0

I was given Vanessa's code base, code written in Swift 2.0, with 3.0 the new standard. It would not compile and most functionality was broken. While some of these were simple fixes, such as connecting IBOutlets on Storyboard, others were more complicated, with new APIs for Swift 3.0 requiring refactoring logic. Our first version of Poly Rides was built in Swift 1.0, and seeing how the language has changed from Swift 1.0 to 2.0 to 3.0 has been an interesting experience.

8.2 Legacy Code

Poly Rides 2.0 ended up being largely a legacy code project. I found myself fixing unknown bugs related to the conversion from Swift 2.0 to 3.0, and bugs resulting from minimal unit testing. It is challenging to be given a code base, with most key features not working, and to fix bugs and implement new features. After working with it for 10 weeks, I can confidently say that I understand the architecture of Poly Rides 2.0. It is a skill in itself to be able to immerse yourself in a new codebase and figure out how another developer tackled a problem. I was also given the codebase with little information, and sometimes inaccurate information, about what had been developed. Many features were bug-ridden and required significant refactoring. Adding new features to a codebase such as this is challenging, and while I learned a lot from Vanessa's work, I felt myself wishing that I had started from scratch.

8.3 Working Alone

My work on Poly Rides 2.0 was entirely solitary. Vanessa gave me her codebase, and I took it from there. As I implemented features, such as ride requests, that require use of push notifications, I became quite frustrated with my single-developer, single-device set-up. It was wildly inefficient to have one developer on features such as this. If I were in the working world, I would likely have multiple devices to use for testing. Using the simulator for testing this was not a solution, since it does not work with push notifications.

8.4 Finances

APIs cost. This was a realization I had while developing Poly Rides 1.0 when our increasing users caused us to exceed Parse's database limits for a free account. In developing 2.0, we also encountered these issues. One day, my Google Places Autocomplete started failing with a bizarre error message recommending I file a bug with Google. I did so, and they got back to me saying that I had exceeded my API query limit for the 24-hour period. In order to query their API that much, I would need to pay for the service. A similar set-up occurs with SendGrid, the platform Vanessa and I used for email verification. Firebase, our database provider, also has thresholds for how much database interaction is allowed with a free account. I tried to find a free API for SMS verification, with no luck. Everything costs money, so I decided to not verify users' phone numbers.

It makes sense that APIs would cost money, however, for a college student developer, this is a challenging predicament. We don't want to charge our users for Poly Rides, yet we might have to start once the app gains enough traction. Vanessa and I have discussed this, and the ideal

strategy would be to take a cut of the cost of the ride via an in-app transaction, similar to Uber's business model.

9.0 Conclusion

Building Poly Rides 1.0 and 2.0 has been one of my most notable college learning experiences. Being able to give back to the Cal Poly community while also learning iOS app development was rewarding. Furthermore, my work on Poly Rides helped me in my internship and job opportunities. Learning about data management, user experience design, and real-world challenges while building this app is what makes Cal Poly's learn by doing philosophy truly exceptional. Additionally, seeing a project from inception to packaging the final product has been a highly valuable experience for a budding software engineer like myself.