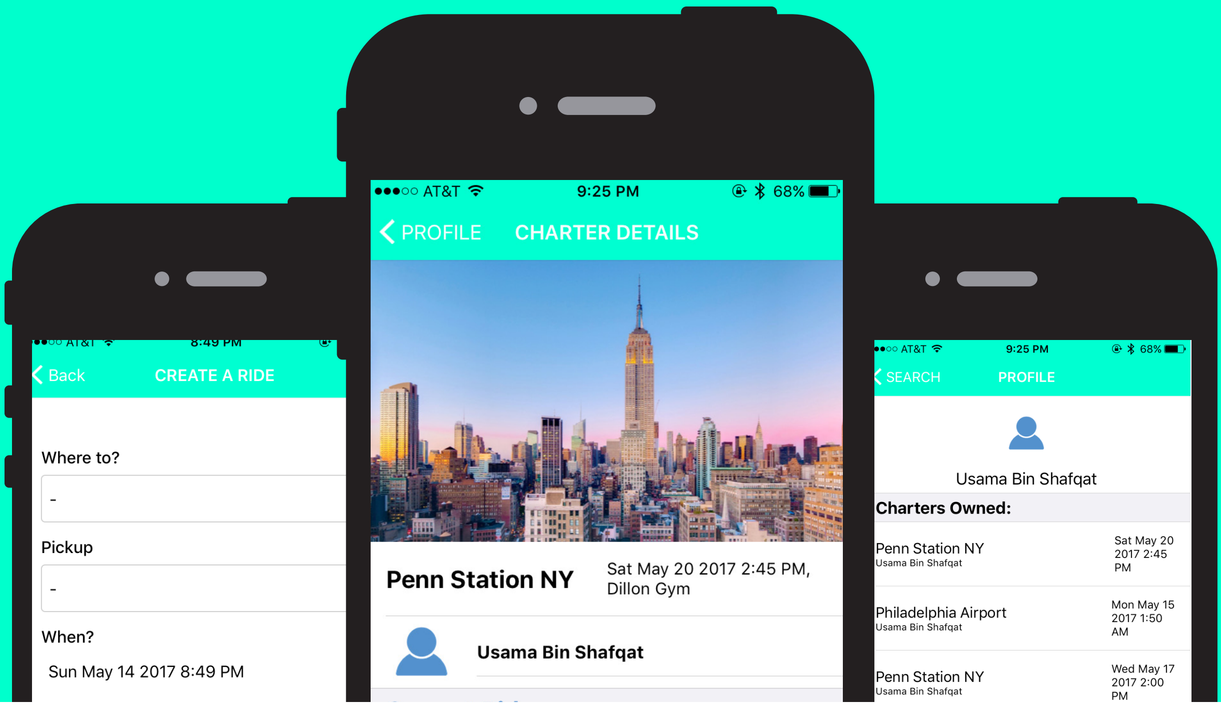
CHARTER



Charter

Prepared for:

COS 333: Advanced Programming Techniques

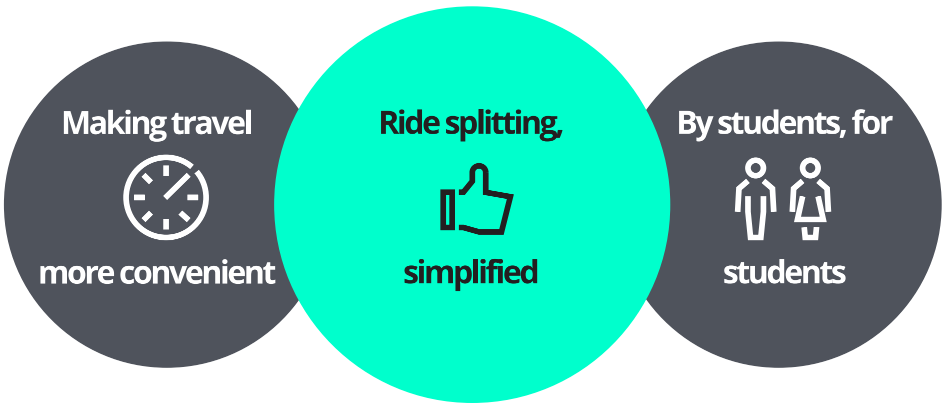
Instructors:
Brian Kernighan and Christopher Moretti

Issued:
05.14.2017

Contributors:
Usama Bin Shafqat, Waqarul Islam, Matt Rosen, Manisha Sivaiah, and Kelly Zhou

A BIT
ABOUT US

Charter is a mobile app designed to make travel more convenient, comfortable, and affordable for University students.


Making travel more convenient
Ride splitting, simplified
By students, for students

OVERVIEW

Charter provides a centralized platform for students to search for, create, and join rides with other students to common destinations by removing the need for ineffective listserv emails and social media posts that currently exist from students looking to share rides. Charter connects students who share a common destination and time to coordinate travel and split costs. Through the integration of Stripe and Apple Pay, Charter enables accountability deposits and fare splitting. The application also allows for direct calling of Ubers to make the entire travel

USER GUIDE
Core Features.....3
Using Charter.....4

DEVELOPER GUIDE
Frontend.....5
Backend.....6

CORE FEATURES

—

Accountability

Upon joining a ride, users will be required to make a non-refundable deposit of $10 to ensure that those who join a ride commit and fulfill the ride. At the end of the ride, the deposit will go toward the total cost of the ride, and the remaining cost will be split among the riders, as explained in the Fair Splitting section below.

Timeline

The timeline feature allows riders within a group to message each other and communicate through the mobile platform. Through this central communication portal, riders can keep each other updated with any relevant trip information (e.g., changes in pickup location, delays, cancellations, etc.). This allows riders to communicate without the need for exchanging phone numbers or personal contact information, and keeps all of the communication within the single platform. This also provides a social component, allowing riders to get to know each other before the trip. Given that the riders are likely all students from the same University, sharing a car for an hour--and all the planning beforehand--is a great opportunity to make new friends on campus!

Fare Splitting

After completing a ride, the ride owner (the user who created the ride) will be prompted to enter the total cost of the ride, and the application will split the fare among the riders and let the ride owner know how much each rider should pay.

Uber Integration

After creating a ride with the set time, destination, and pickup location, the ride owner can call the Uber directly from an application (should the riders choose to take an Uber) at the designated time. Once the ride owner selects the “Request Uber” button on the application, the Uber application will open with the destination and pickup location pre-filled and ready to request. Calling the Uber is not required, so riders also have the option of taking another form of transportation they might believe to be cheaper (e.g., if one of the riders has their own car or they want to look into other ride-sharing platforms).

USING CHARTER

01

SIGN UP/LOG IN

Upon first opening the application, first-time users will be prompted to sign up with their University email address and password. New users will receive an email to verify their accounts before beginning use of the application. Returning users will be prompted to log in to their accounts each time they open the app for use.

02

SEARCH FOR A RIDE

After logging into the app, users can search for existing rides based on their desired destination and time of travel. Users can choose a destination, date, and time from pickers on the screen, and any existing rides that best match the specifications will appear on the next screen.

03

JOIN A RIDE

After finding a convenient existing ride on the app, users can join the ride. Upon joining, the new riders will be prompted to pay a non-refundable deposit of $10, which will ultimately go toward paying their share of the cost of the ride. This deposit is collected to hold users accountable for completing the rides they join, as explained in the Accountability section listed among the features.

04

CREATE A RIDE

Users can also create a new ride if a convenient one does not already exist. A new ride can be created by entering the destination, pick-up location, date, and time from pickers on the screen. Once a user creates a ride, he or she becomes the ride owner and is responsible for ordering the Uber (if the group chooses to travel with Uber) and marking the ride as complete at the end. The newly created ride will now be available for searches that correspond with its ride details for other users to join.

05

LEAVE A RIDE

In the event that a user needs to leave a ride that was joined, the user can select the ride and choose to leave. The user will be reminded that the deposit paid upon joining will be lost and be asked to confirm their decision to leave. After leaving, the user will no longer be a part of the ride and their deposit will still go toward the total cost of the ride.

FRONTEND

—

UI DESIGN

COLOR

The primarily white screens offer an overall clean and crisp look while the teal and blue accents bring bright pops of color.

FONT

Custom font used for the headers, buttons, etc. differentiates the UI from generic apps and adds to Charter's clean but fun feel.

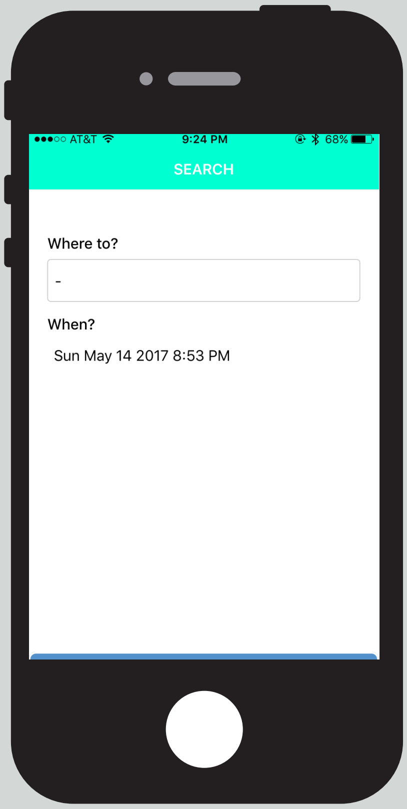
LOGO

Created in Photoshop and Sketch, the logo is intended to be simple and distinct to attract users on the app store.

BUTTONS

The placement of buttons throughout Charter makes the navigation feel intuitive. TouchableHighlight offers the user instant gratification upon action.

SCREEN INFORMATION



Where to?

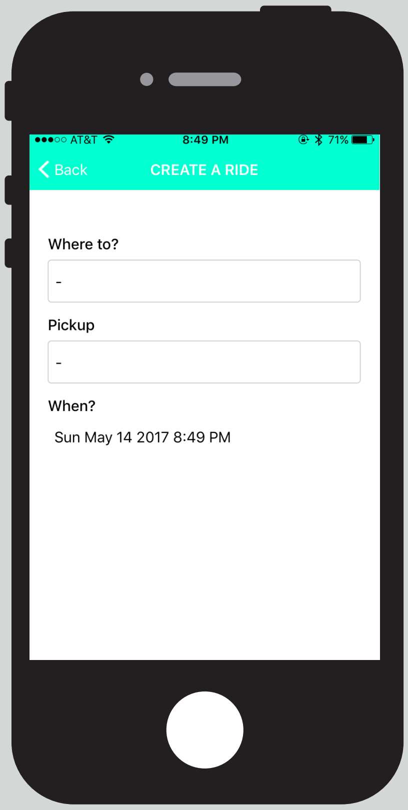
-

When?

Sun May 14 2017 8:53 PM

SEARCH SCREEN

The `SearchScreen` is the primary screen when a user opens or logs in to the app. It features two inputs: destination and time. Both components are rendered using the tcomb library. Pickers were chosen over text input to reduce the potential for errors in the input. The SEARCH button navigates to a `ListScreen` with the user's selections passed in as the `params` and features the relevant search results.



Where to?

-

Pickup

-

When?

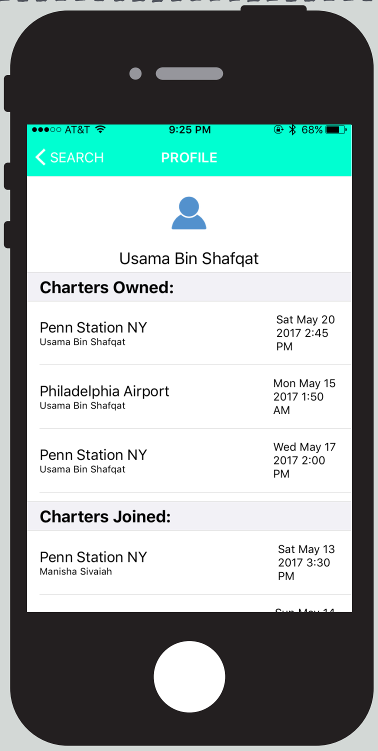
Sun May 14 2017 8:49 PM

CREATE SCREEN

We used the `tcomb-form-native` library to create a native form for users to enter the details of a new charter (the date, pickup location and time, and destination). As the user enters information, the relevant fields in the domain model are updated. After the user taps SAVE, the information in the domain model is pushed to a new node on the database.

PROFILE SCREEN

The `ProfileScreen` features the Charters owned and joined, and the option to logout. Native-base components, such as Separator, List, and Thumbnail, were used to organize this information to maintain even spacing and an overall cohesive appearance. The Charters listed in each section navigate to different versions of the `DetailScreen`.


●●●○○ AT&T 9:25 PM 68%

SEARCH PROFILE

Usama Bin Shafqat

Charters Owned:

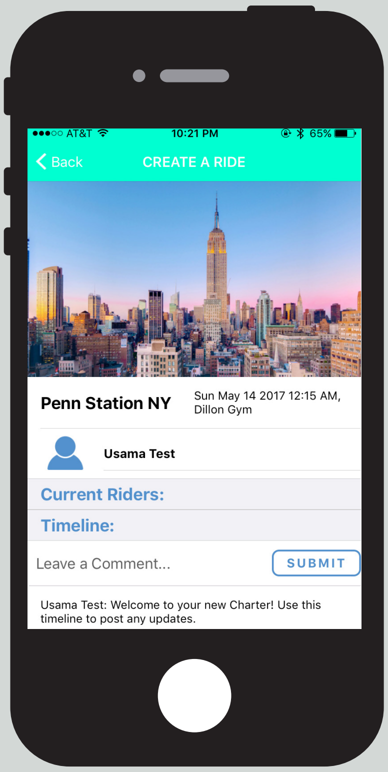
Penn Station NY Usama Bin Shafqat	Sat May 20 2017 2:45 PM
Philadelphia Airport Usama Bin Shafqat	Mon May 15 2017 1:50 AM
Penn Station NY Usama Bin Shafqat	Wed May 17 2017 2:00 PM

Charters Joined:

Penn Station NY Manisha Sivaiah	Sat May 13 2017 3:30 PM

DETAILS SCREEN

The `DetailScreen` has three different incarnations. All versions feature an image of the destination, the ride owner, a list of students who have already joined the ride, and the timeline. The default `DetailScreen` is navigated to from the `SearchScreen` and features the ability to join a ride which opens the Apple Pay or Android Pay popup. The other two versions are navigated to from the `ProfileScreen`. If any of the owned Charters are selected, the user is directed to a an `OwnDetailScreen` that allows the owner to request an Uber or indicate that the ride is completed. If one of the joined Charters are selected, the user is directed to a `JoinDetailScreen` that includes the option to leave the ride.


●●●○○ AT&T 10:21 PM 65%

Back CREATE A RIDE

Penn Station NY Sun May 14 2017 12:15 AM, Dillon Gym

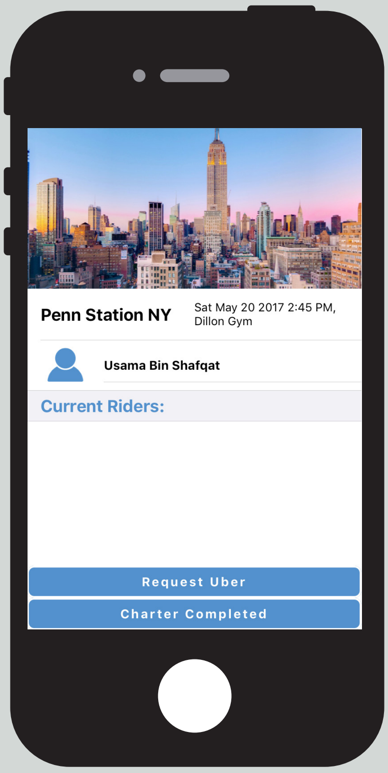
Usama Test

Current Riders:

Timeline:

Leave a Comment... SUBMIT

Usama Test: Welcome to your new Charter! Use this timeline to post any updates.

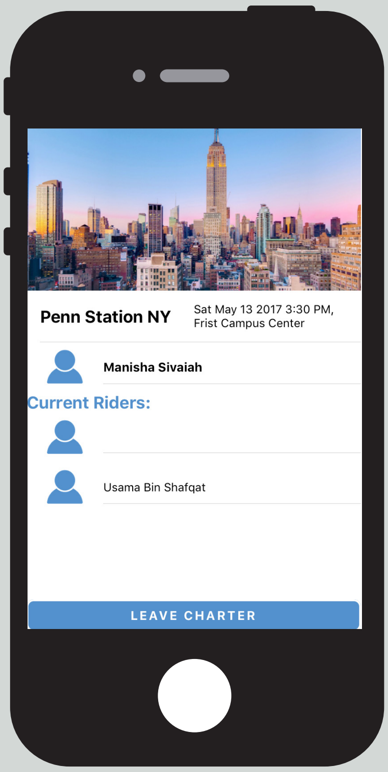

Penn Station NY Sat May 20 2017 2:45 PM, Dillon Gym

Usama Bin Shafqat

Current Riders:

Request Uber

Charter Completed


Penn Station NY Sat May 13 2017 3:30 PM, Frist Campus Center

Manisha Sivaiah

Current Riders:

Usama Bin Shafqat

LEAVE CHARTER

BACKEND

FIREBASE

We used Google's Firebase platform (essentially a Backend-as-a-Service) to host a NoSQL database for Charter. Although Firebase currently does not officially support React Native, we used the official implementation guide designed for Web (primarily Node.js) with some key changes to handle React Native. Our work was made more intuitive by the fact that both Firebase and React Native rely on a state-based model where the state of the Firebase listener in the app changes when a database update occurs and the ensuing change of state in the React Component leads to a re-rendering of the UI.

The NoSQL database is accessed by listeners attached to various views of the app. Listeners attached to ListScreen and ProfileScreen pull a list of the appropriate charters to populate lists such as the Search Results and the user's Owned Charters and Joined Charters. On the Charter Detail screen, the complete details of the Charter are downloaded from database including the names of the other riders on the trip as well as the timeline associated with it.

The database is structured as a JSON object. There are two main nodes: users and charters. Following best practices for NoSQL/Firebase that allows for efficient scaling, we have some necessary duplication of data because of flattened data structures. For example, for each charter that a user joins, we store the UID of the charter in the user's charters-joined list and also store the user's UID in the charter's riders list. The UID is a unique identifier for every charter and user in the database that is automatically generated by Firebase and is guaranteed to be unique and consistent. Using the UID, we can download all the information for a charter or user by efficiently subsetting the NoSQL database and this provides for extremely fast information retrieval with nearly zero lag.

Every time the Firebase realtime database is accessed in the app, the request is wrapped in a function call (login(), signup(), join(), leave() etc.) which makes it very simple to change the underlying database platform in the future to another NoSQL database such as MongoDB if, for example, more advanced search capabilities are required. The actual firebase query is structured as a reference to a specific node in the database (such as users/UID/email for example) and then a function on that reference.

AUTHENTICATION

We use the Firebase authentication platform for managing users in the app. When a user signs up, they are asked to complete a short form indicating their email address, desired password and full name. They are then sent an email with a verification link to ensure that it is an actual account (if domain restriction is introduced in a later version, this would also be an easy way to enforce it). Once the email is verified and the signup formalities are completed, a new node is created for the user under the users node with the email and name pre-filled and with the unique userid assigned by Firebase upon signup being the node's key. The signup, login and email verification calls are made using the functions in Firebase.auth(). The currently logged in user's uid can always be accessed using Firebase.auth().currentUser.uid and any other registered details of the user can then be accessed using a reference to this UID in the database. The user's data input of their email address and password is handled through the Form component in NativeBase which ensures local security of the data and then transmits it to the Firebase Auth using an encrypted protocol.

STRIPE

We use Stripe to process payments for ride deposits, each of which is $10. This is currently an arbitrary amount that can be modified from the source code of the app. Most Charters will have a cost over $10/per person if they are need splitting. Stripe has its authentication key (available from the Stripe dashboard upon login) embedded in the app, the money from deposits is sent to the developer account before it goes to the trip owner. In terms of implementation, we were able to integrate a React Native package called TIPSI Stripe that has adopted the iOS and Android APIs for React Native. Apple Pay and Android Pay allow for a variety of card systems to be inputted, which then send the data on to Stripe. The only maintenance required here will be sending the money from the Stripe account to the appropriate ride owners (which can be automated in the future).

UBER

We decided to include the ability to be able to call an Uber directly from the app. Uber is the most popular ride-sharing platform that is used on campus, so we thought being able to easily call one when the riders want to start their trip would be essential in being able to have a seamless riding experience. At first, we attempted to use the React Native Uber Rides module to get access to the API for Uber, but it turned out this was not a maintained package. As a result, we turned to deep-linking an Uber button which essentially opens the Uber app and prefills information about the ride such as the destination and the pickup location. This is done through a simple link that is attached to an Uber-styled button. The parameters of the link can be changed pretty easily. For example, a ride going to Penn Station can be described in a single deep link. Here we have the Uber Authentication Key from the developer console, as well as formatted text addresses for the dropoff that also have longitude and latitude information.

uber://?action=setPickup&client_id=yv1QEhEQm8SsCbaptSahN_Cg5DEDAmm0&pickup=my_location&dropoff[formatted_address]=Penn%20Station%2C%20West%2033rd%20Street%2C%20New%20York%2C%20NY%2C%20United%20States&dropoff[latitude]=40.750303&dropoff[longitude]=-73.990906'