TMHT
(Take Me Home Tonight)
DOCUMENTATION

Rachel Blacker, Shawn Minarik, Christopher Renus and David Sparkman

Table of Contents

Introduction

Our term project is going to be a car sharing service application. The purpose behind the website is to allow RPI students to share rides to and from school on breaks. Additionally, information about public transportation is also available in the app in case there are no shareable rides available. This web application uses HTML, CSS, JavaScript, Bootstrap, AngularJS, NodeJS, mongoDB, express, and npm to develop integral features. These features will be explored in the following documentation to guide a first-time user through the site.

Running

This application is hosted at TakeMeHomeTonight.HerokuApp.com. You do not need to have anything else installed in order to run the application. It is highly recommended that the user uses Google Chrome with this application. Firefox and Safari are not compatible with certain website features.

Data Model

The data for this application is stored in a database with three tables. These tables each have multiple fields. When different pages and search queries are called, data from the databases will be accessed and displayed.

The first table is the users table. This table has all of the users' information, including the following fields:

- _id: unique id
- rcs: rcs id
- firstName: first name
- lastName: last name
- car: object that includes the following
    - make: car make
    - model: car model
    - color: car color
    - license: car license plate
- notifications: object that includes the following
    - rideID: unique ride id
    - message: notification message
    - timeSent: time notification was sent
    - status: view status

The next table is the rides requested table. This table contains information about all of the rides that are being requested. The rides requested table includes the following fields:

- _id: unique id
- rcs: owner rcs id
- departTime: departure time
- departLocation: departure location
- departDate: departure date
- destination: destination
- cost: cost or price willing to pay
- accepted: status on if accepted
- driver: object that includes the following
    - rcs: driver rcs id

    - status: pending or accepted

The last table is the rides available table. This table contains information about all of the rides that are being offered. The rides available table includes the following fields:

- _id: unique id
- owner: owner rcs id
- departTime: departure time
- departLocation: departure location
- departDate: departure date
- seats: total number of seats available
- availableseats: number of seats currently available
- cost: cost
- riders: object that includes the following
    - rcs: riders rcs id
    - status: pending or accepted
- car: object that includes the following
    - make: car make
    - model: car model
    - color: car color
    - license: car license plate

Routing

The page first loads up the index page which uses the angular app, tmht in app.js, which is the app for the whole page. Within the app we set the initial configuration which includes using $routeProvider. Within routeProvider, we have the ability to change the view and controller as well as other things based upon the route. This way based on different routes, we can swap views in and out of the page, allowing for a single page application.

Backend routing is split up among multiple files. We utilize express.static() to serve any static files and the index.html page is served up through a separate file which just sends the page. Any GET, PUT, DELETE, or POST requests pertaining to users and the '/users/' route, go through users.js route and anything for '/rides/' goes to rides.js. Login and logout are currently in the main server.js as well as the socket setup for notifications.

Header

Our header changes depending on if the user is logged in or not. When just on the login page, the header only has our logo on it. When the user clicks on that logo, they are redirected to our homepage. Below is the look of our header when the users is not logged in.

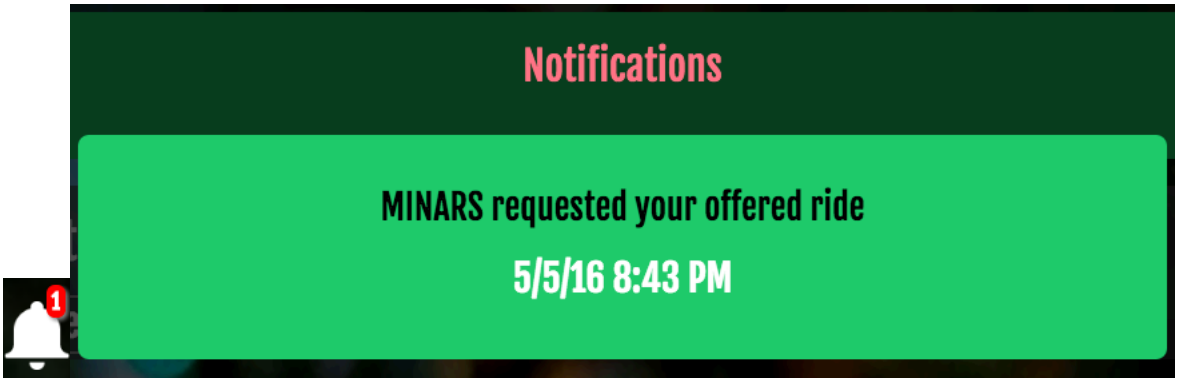
When the user is logged in, they see a different header. On the right are multiple links the user can choose from. On the left under our logo is a notifications drop down menu that has any notifications for the user. The logo with redirect back to the homepage is also on this header. Below is the look of our header when a user is logged in. When the window size is smaller than medium, the links change to accommodate the compact screen. Below is also the look of the header when a user is logged in but their screen size is smaller.

Notifications

Take Me Home Tonight is enabled with a notifications feature that allows us to alert users of a status change. Since users may be asking for multiple rides at a time, or may be too busy to monitor any rides affecting them, we are able to notify them of any change that is made to a ride they are on. For example, if someone's ride needs a driver and someone signs up to fill the role, the owner of that ride is notified and can either accept or reject the driver. Similar notifications are displayed for people offering when users request to join the ride.

When clicking the bell in the corner a user can view the notifications that they have. When they have notifications they have not seen, a red bubble will show up on it showing them that they have unread notifications. When they click on it, the notifications are read and the bubble goes away. From here, they can click on a notification which will bring them to the ride in which that notification pertains to. The background color of the notification changes as well if they have not seen it yet.

The notifications service utilizes sockets. This is done to provide live changes of notifications. Any time a change is made to a ride, whether it is a deletion of user or ride, or confirmation of a rider or driver, a notification is made to related users. The notification is pushed to a notification array that is per user. The object pushed contains the message, the date, whether it has been seen, the ride id it pertains to, and then whether it was an offered ride or requested. From here, the front end knows a change was made and send "update notifications" to the socket to request the notifications and the changes that have been made. Using rooms, created with a user's rcs, the sockets are able to distinguish which socket is which and who to send the notifications to.

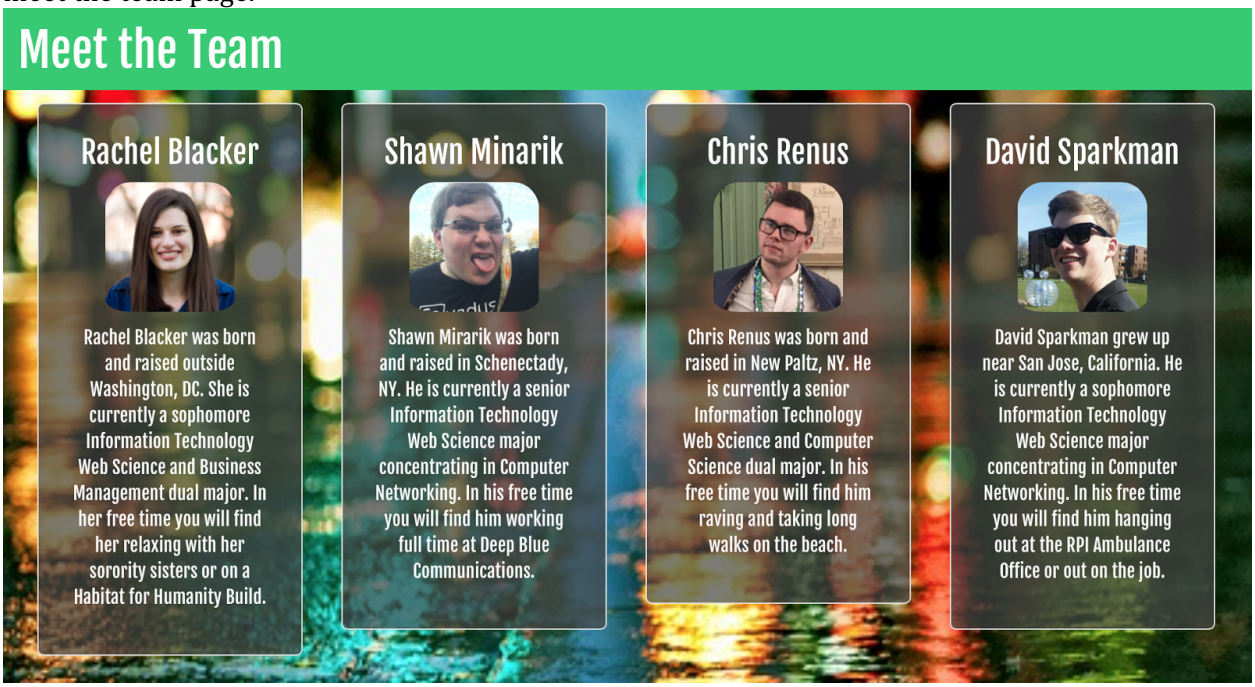


Footer

Our footer stays static throughout our application. It does not change depending on if the user is logged in or not. The footer has our copyright information written on it as a well as a link to Meet the Team. Below is the look of our footer.

Meet the Team

A static page is set up for the user to find out about the developers of the application. The user can see this information whether or not they are logged into the application. Below is the look of our meet the team page.

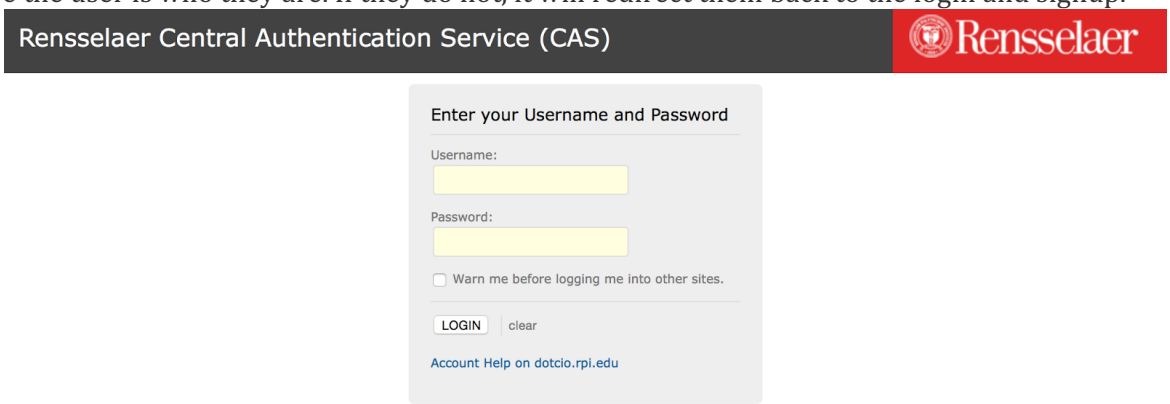


Login Page

When a user first goes to our application, they are directed to the login page. This page allows a user to register for our site or login.

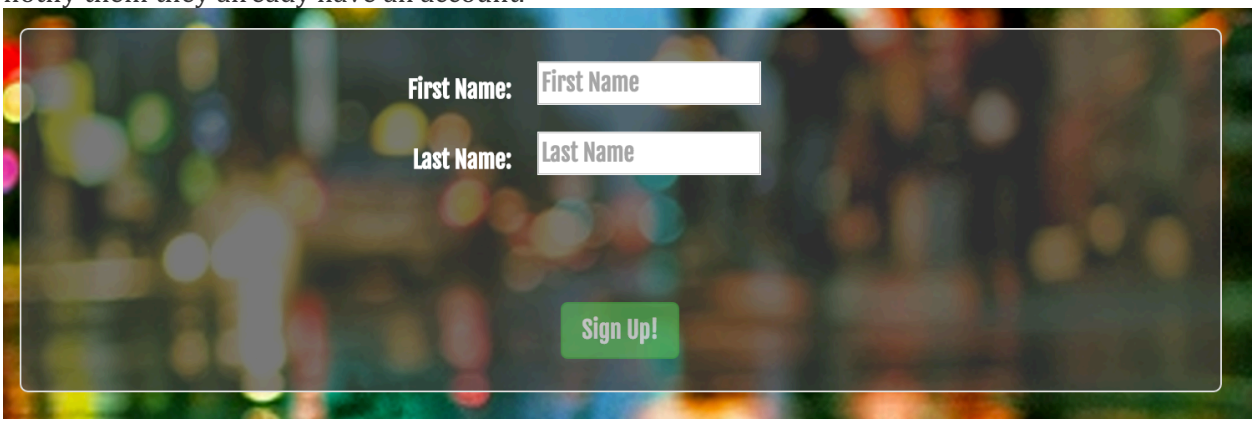
Existing User

If a user already has an account on our site, they can login using the main login section seen below. On the backend, login utilizes, the cas-authentication module. Due to Cross Origin Request issues, since our site and RPI's CAS system are on separate domains, the module code had to be redone to send the URL and allow the browser to redirect the user instead of the server doing the redirect. After a user logs into CAS, if they are authenticated and are a user, the server redirects them to our landing page. In addition, it adds a session token and a cookie for the user to authenticate and make sure the user is who they are. If they do not, it will redirect them back to the login and signup.



New User

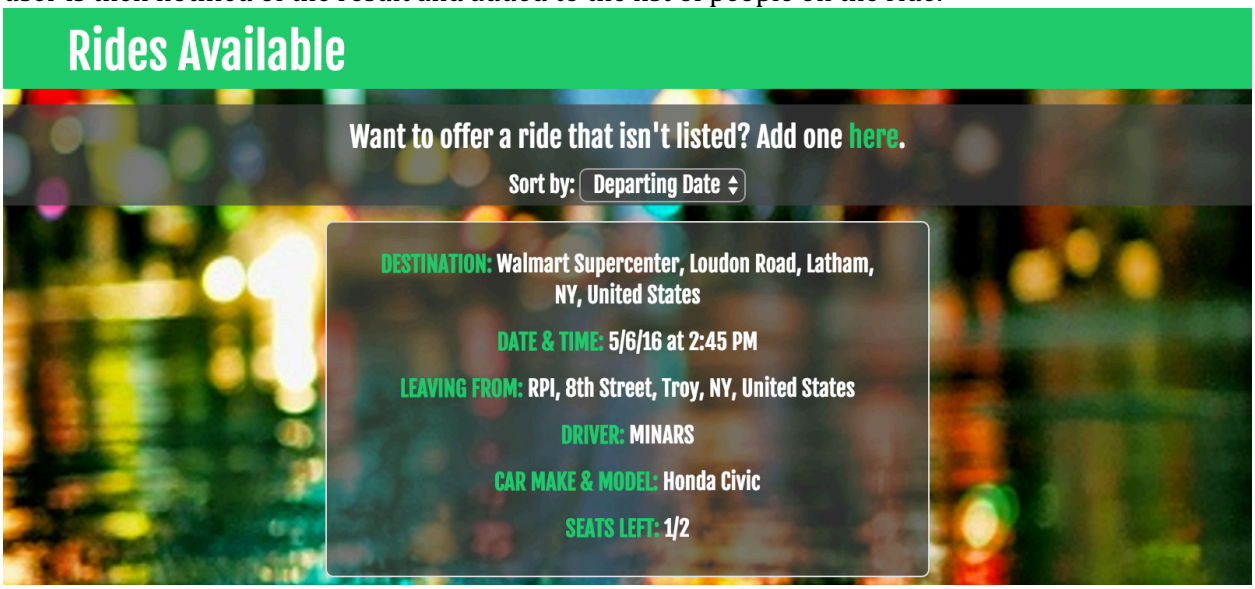
If the user has not been to the application before, they can create a new account using the sign up section on the main page. When a user clicks sign up, it redirects them to the CAS login. Since our limits our users to just RPI students for safety reasons.. After signing in through there, it will redirect the user to the sign up page. From here they can enter some details about themselves. Once they click sign up, their information will be pushed to the user collection within the database. If a user already has an account and tries to sign up, it will just redirect them to the landing page and notify them they already have an account.

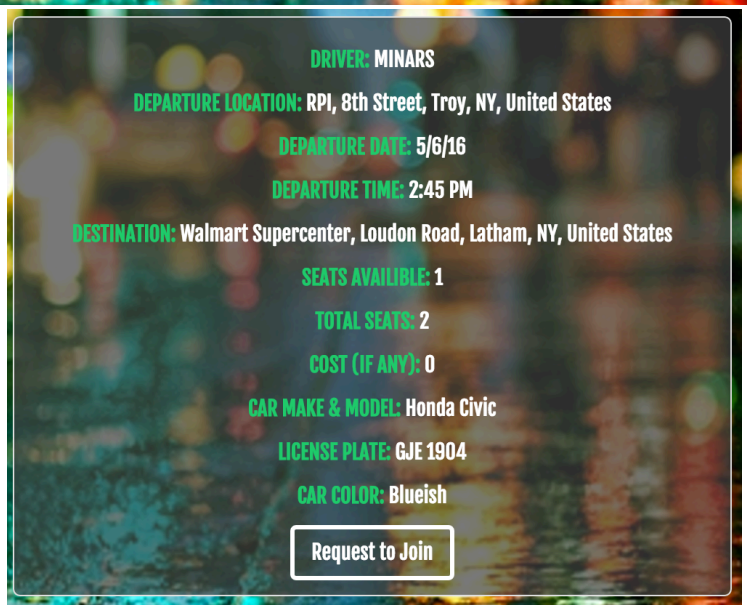


Rides Available

The *Rides Available* screen shows the rides being offered to various locations. The data from our database is parsed and displayed so each ride is in its own well. From this page, our users can see the most important details about each ride. This includes, the destination, departure- time, location, and date, as well as information about how many seats are available, the cost of the ride and details about the driver and their vehicle.

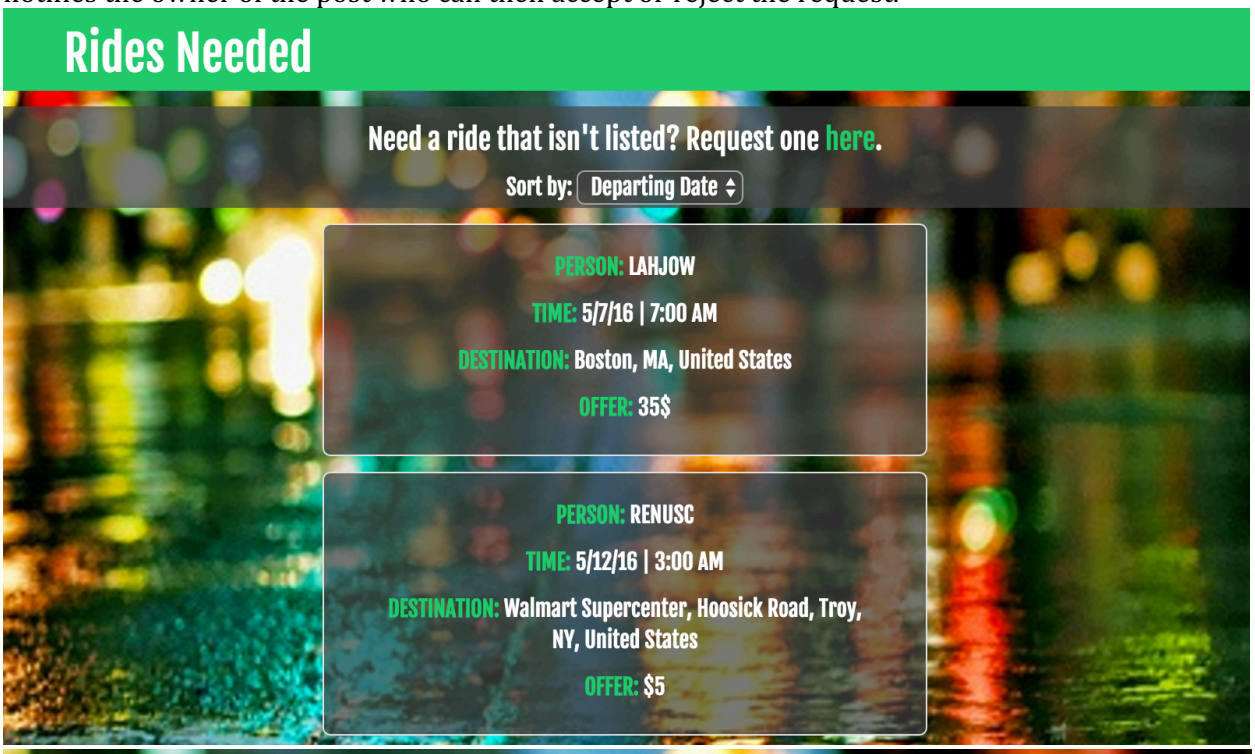
Once a user finds the ride they need, they can click the well. This directs them to the ride's page which displays the same information in addition to a button that allows them to submit a request to join the ride. This sends a notification to the driver who can then approve or deny the request. The user is then notified of the result and added to the list of people on the ride.

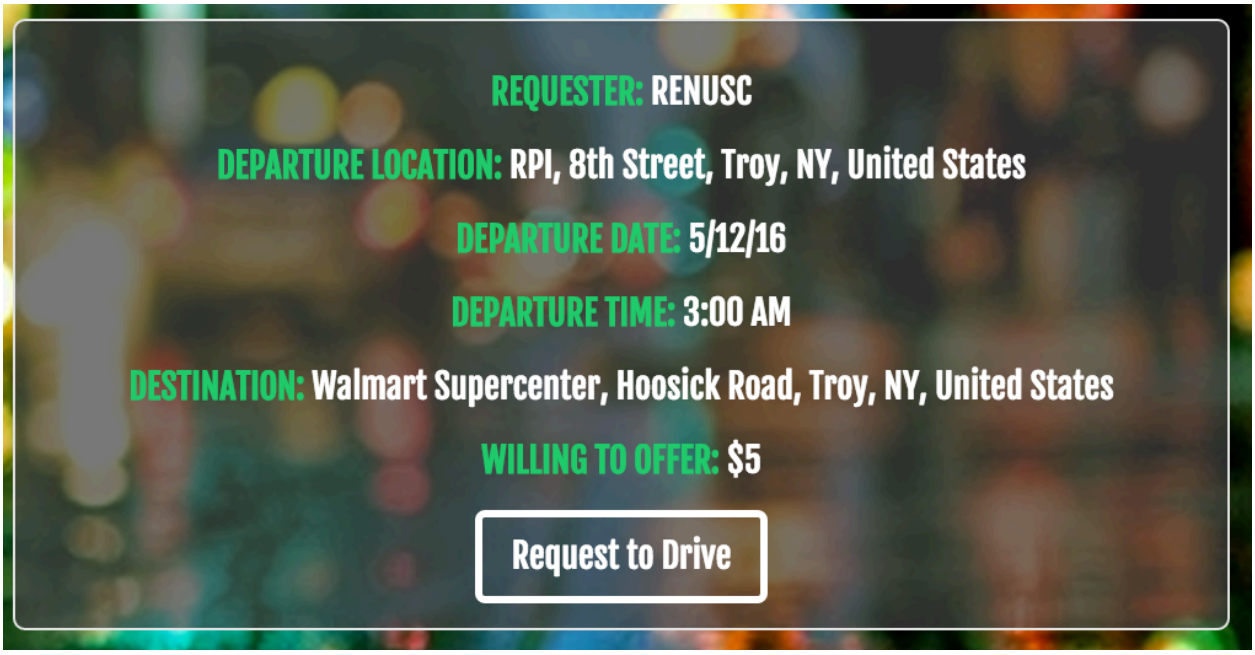




Rides Needed

Rides Needed is fairly similar to the *Rides Available* screen. Here users can view the rides that are needed to certain destinations. If a user is able to drive for the requested ride, all they have to do is click the request to be directed to the ride's page. From here they can select *Request to Drive*. This notifies the owner of the post who can then accept or reject the request.



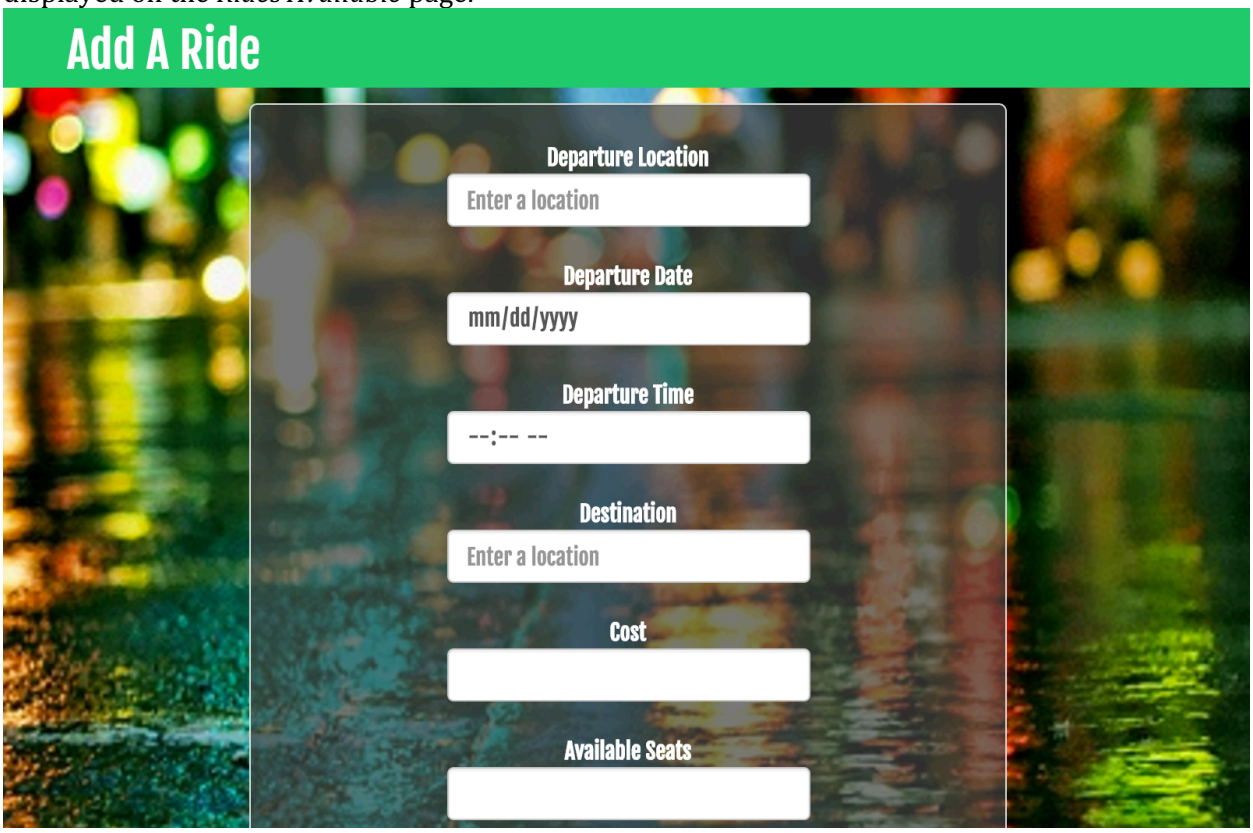


Add a Ride

Adding a ride that you are willing to provide is as simple as filling out a form. All the user has to do is fill in the departure location, date and time, as well as the destination and the number of seats available. If they want to be reimbursed for the trip, they can choose to add a cost to the ride. This value does not need to be a monetary amount, so if a driver only want food for the trip or a really awesome playlist, they are able to enter that data.

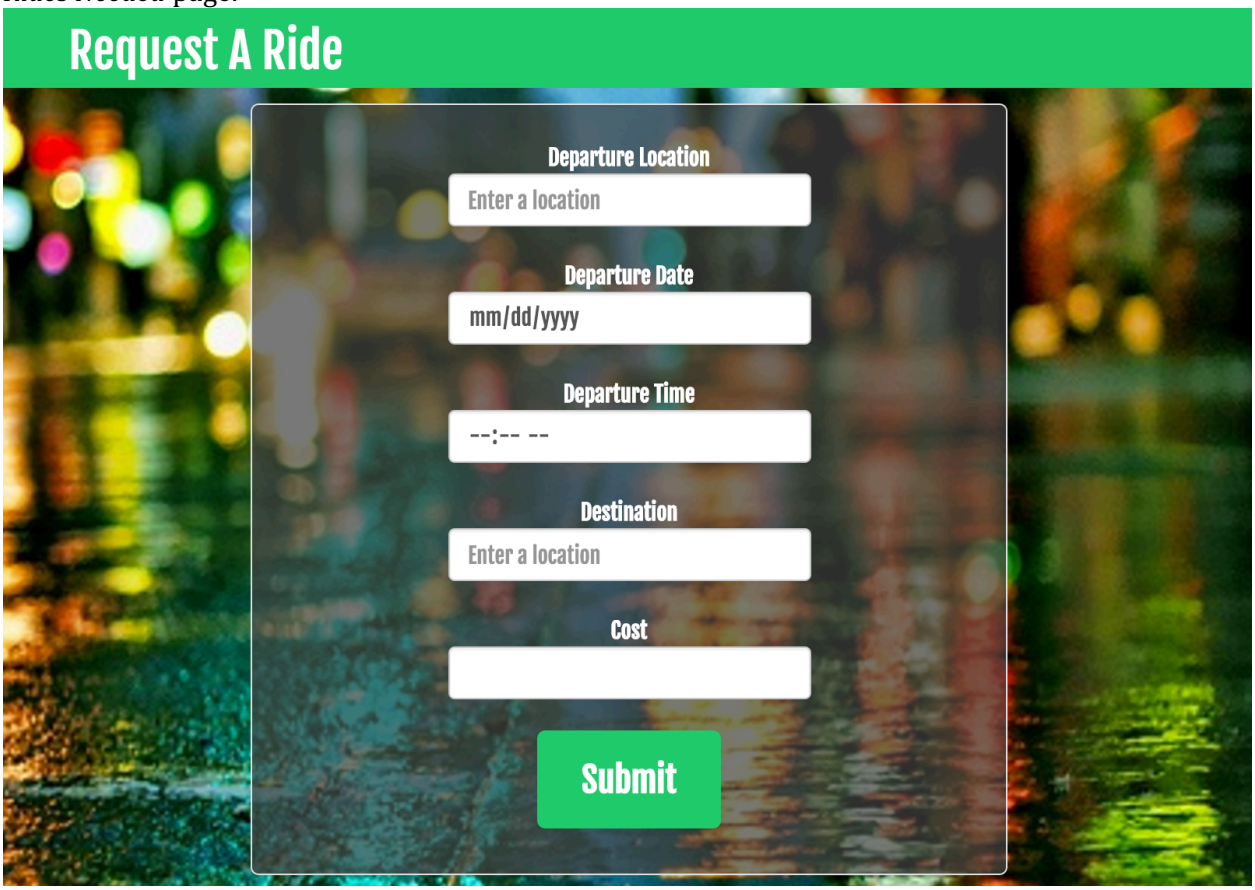
Our form also requires data about the car and driver. If the user has already populated their profile with this information, it is automatically filled in from the database, but remains changeable in case they are using another vehicle. If the data has not been filled in, the form-fields remain blank and the user must fill them in.

Once the user is happy with their ride, all they have to do is submit the form and it is automatically displayed on the *Rides Available* page.



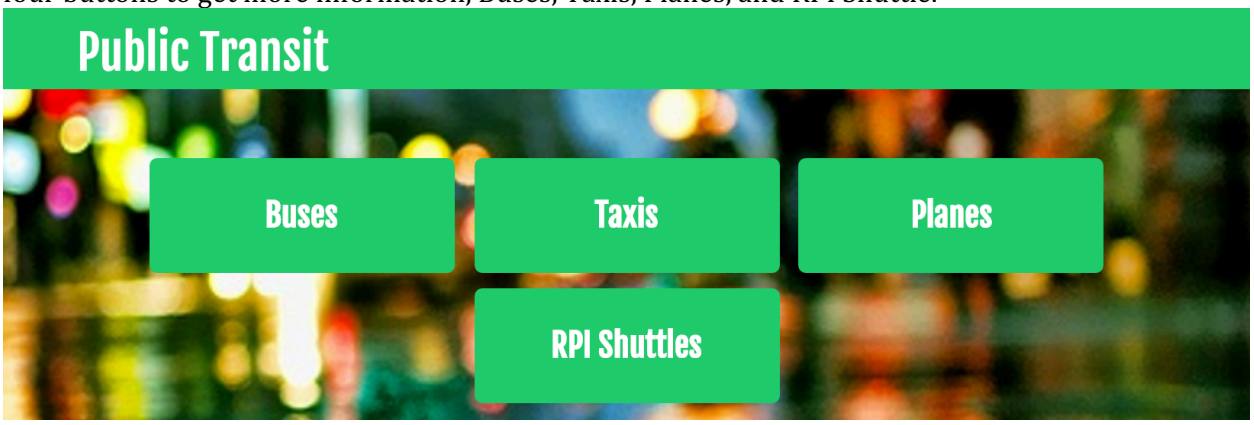
Request a Ride

Requesting a ride works the same way adding a ride does. All the user has to do is click the tab, fill out the form and click submit. The ride is automatically added to the database and displayed on the *Rides Needed* page.



Public Transit

If a user is unable to find a ride through the rides available or rides requested page we provide them with information from multiple public transportation options. The user can click between four buttons to get more information, Buses, Taxis, Planes, and RPI Shuttle.



Buses

This project utilized the CDTA API. The Capital District Transit Authority is currently developing an API that will eventually be released to the public. Our group was able to contact CDTA about this API can receive early access for this project.

The CDTA API is a restful API that is able to provide route and scheduling information for all of their bus lines, as well as updates on service disruptions or delays. Each API call is made using an http get request, and returns a JSON object containing the requested information. The API is also able to return data in the XML format. *Take Me Home Tonight* makes use of several API features.

Destination Lookup

The CDTA API has a feature that allows us to search for any destination by name, and return a list of all stops matching that search term. The JSON object returned holds information about the specific name of the stop, its latitude and longitude, as well as which routes service the stop and the direction they travel in. Some stops will also return a five digit ID number that the API can accept to view upcoming arrivals for that specific stop. We then parse this information and display it in a format that is easy for the user to read and understand.

Route Lookup

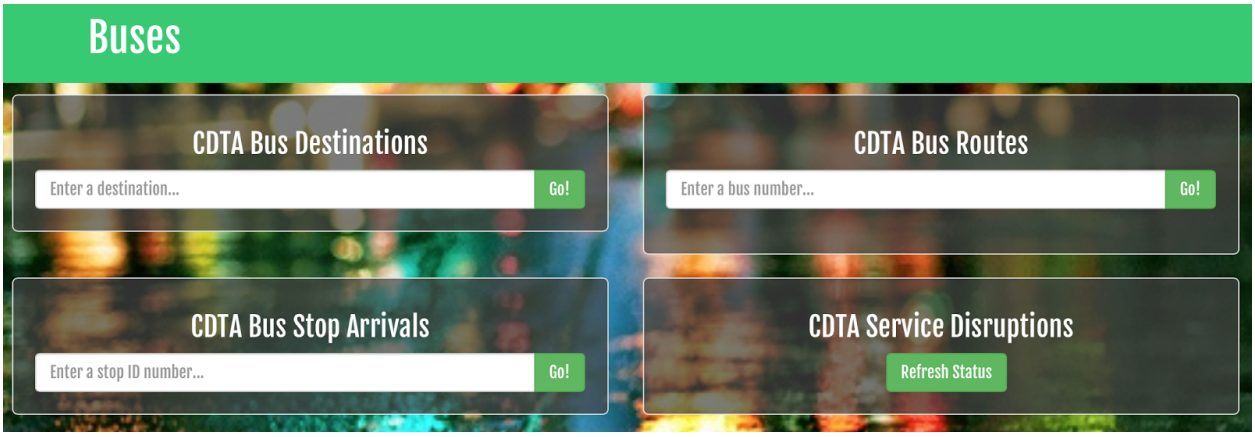
In addition to searching for destinations, we were given the ability to look up which stops are serviced by a certain bus. All the user must do is look up the bus line they want, enter it in the box, and click the button. The API will return a JSON object that contains the two directions the bus travels in and the name of that route. This data is displayed in the form of two buttons. Depending on which button is clicked, another query is submitted and the proper route information is returned in another JSON object. The user can then view the route as it is parsed by angular and displayed below the box.

Stop Lookup

If a user wants to look up arrivals by stop they can do so by the stop ID number which is provided from the destination lookup menu or by the route search. By entering this stop in the correct box, they are submitting a query that returns the next arrival at the requested stop for each line that services the stop. The API provides the scheduled arrival time as well as a real-time estimate for when the bus will be arriving. Unfortunately there isn't currently a good way to return any data by just the stop name since there is a lot of overlap in naming and because the API currently does not support any other form of query.

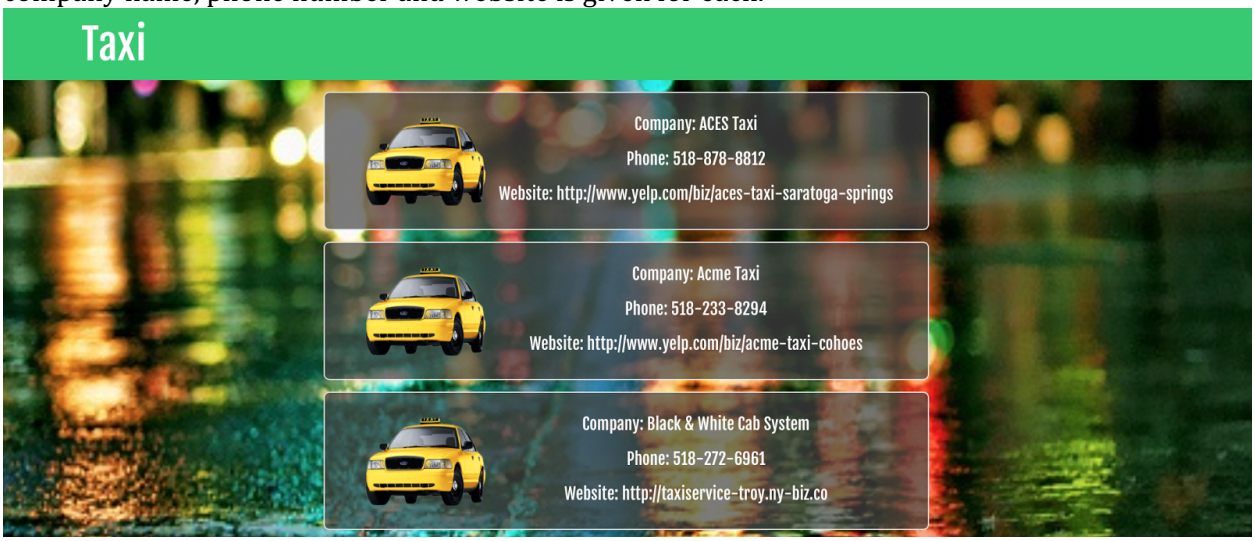
Service Status

Service status is a simple, static API query that returns a JSON object containing any service disruptions or outages. This is simply formatted into a header and a message within the JSON object, which is then parsed by angular and displayed.



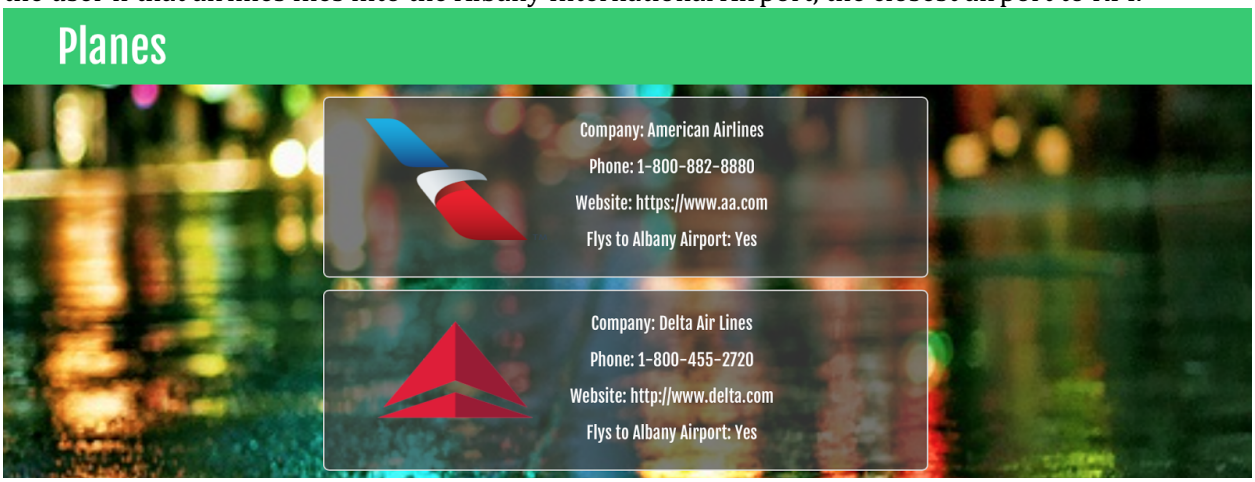
Taxis

The taxi page features information about all taxi companies in the capital district area. The company name, phone number and website is given for each.



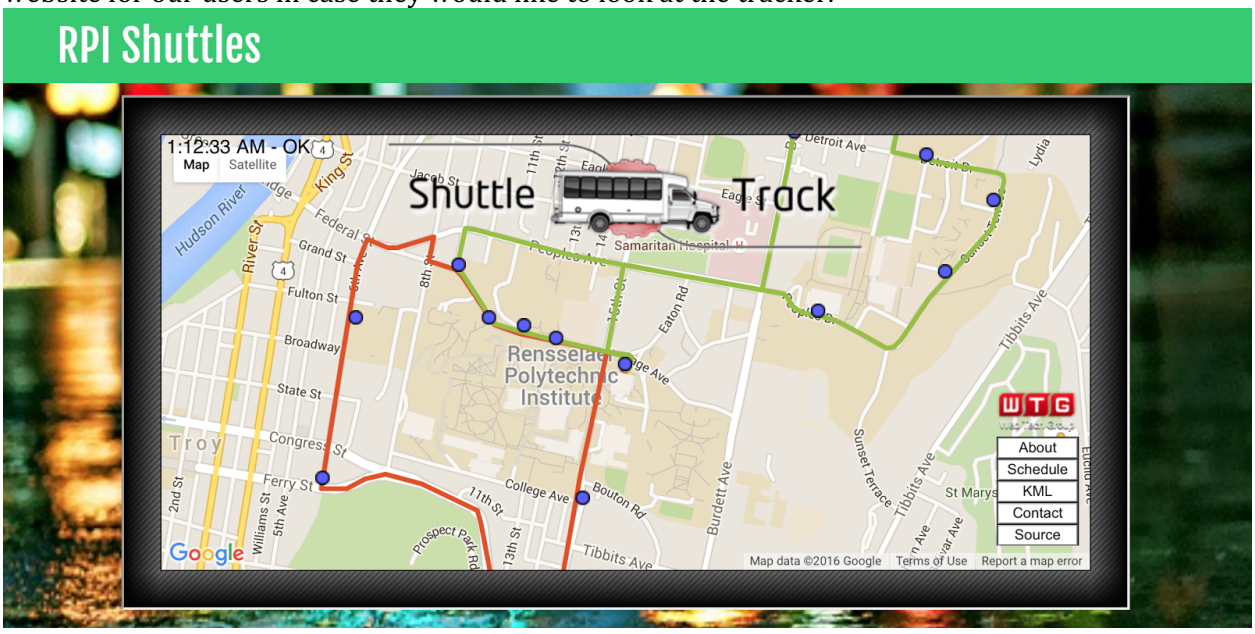
Planes

The plane page features information about all major U.S. airlines in a clean and concise way. The airline name, customer service number and website is given for each airline. Additionally, we tell the user if that airlines flies into the Albany International Airport, the closest airport to RPI.



RPI Shuttles

At this time the only users of our application are RPI related. Many students use the RPI shuttles to get around campus. The RPI shuttle service has their own tracker. We provided an iframe of their website for our users in case they would like to look at the tracker.



Profile

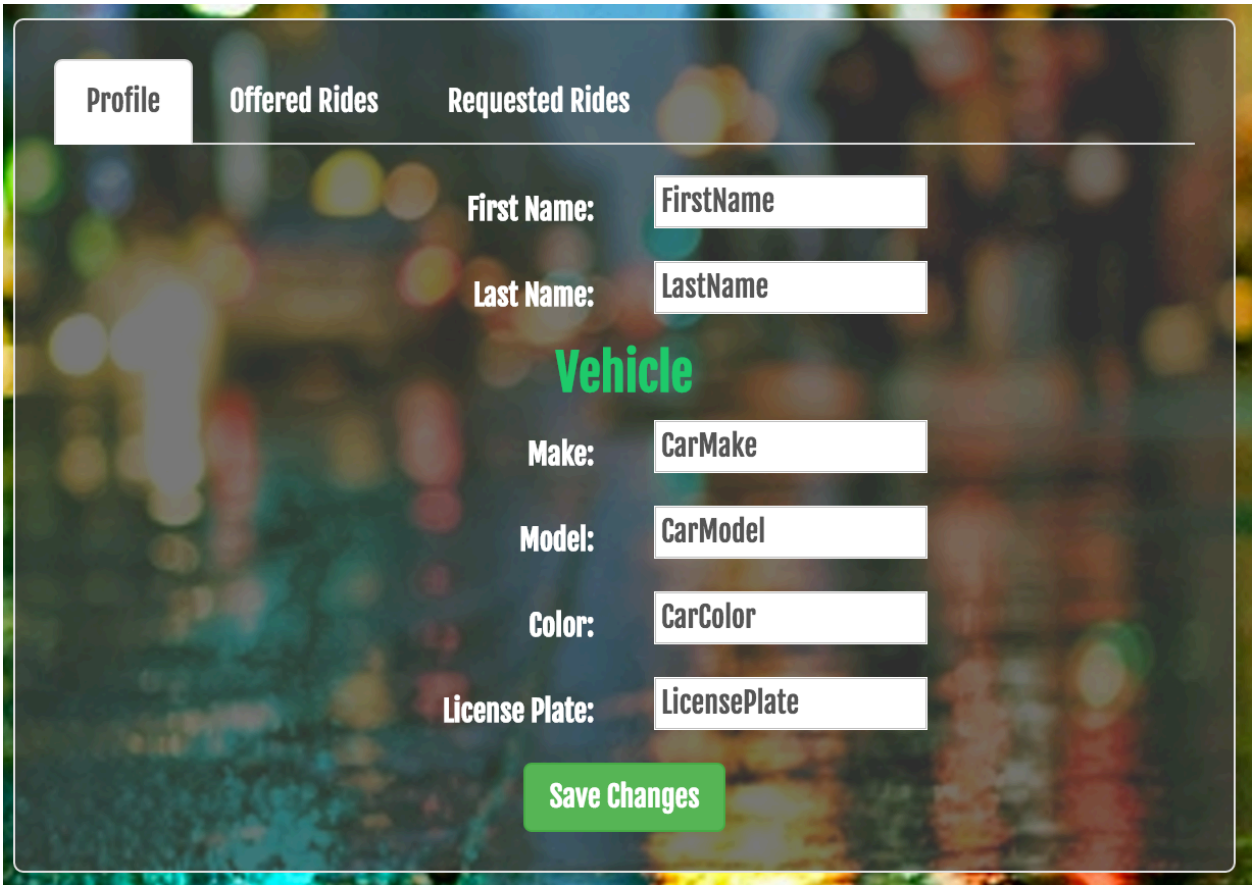
Take Me Home Tonight also gives each user the ability to set individual profile information. Since our service is using CAS for login and authentication, we don't need to store any of that data. However, the profile is still a vital part to giving each user a unique identity and fill in relevant details.

Settings

The settings page enables the user to set their profile information. This includes their first and last name, as well as their vehicle information. This data is stored in our database and is called on to populate the wells for rides. All the user needs to do to update this information is fill out the desired fields and submit the form.

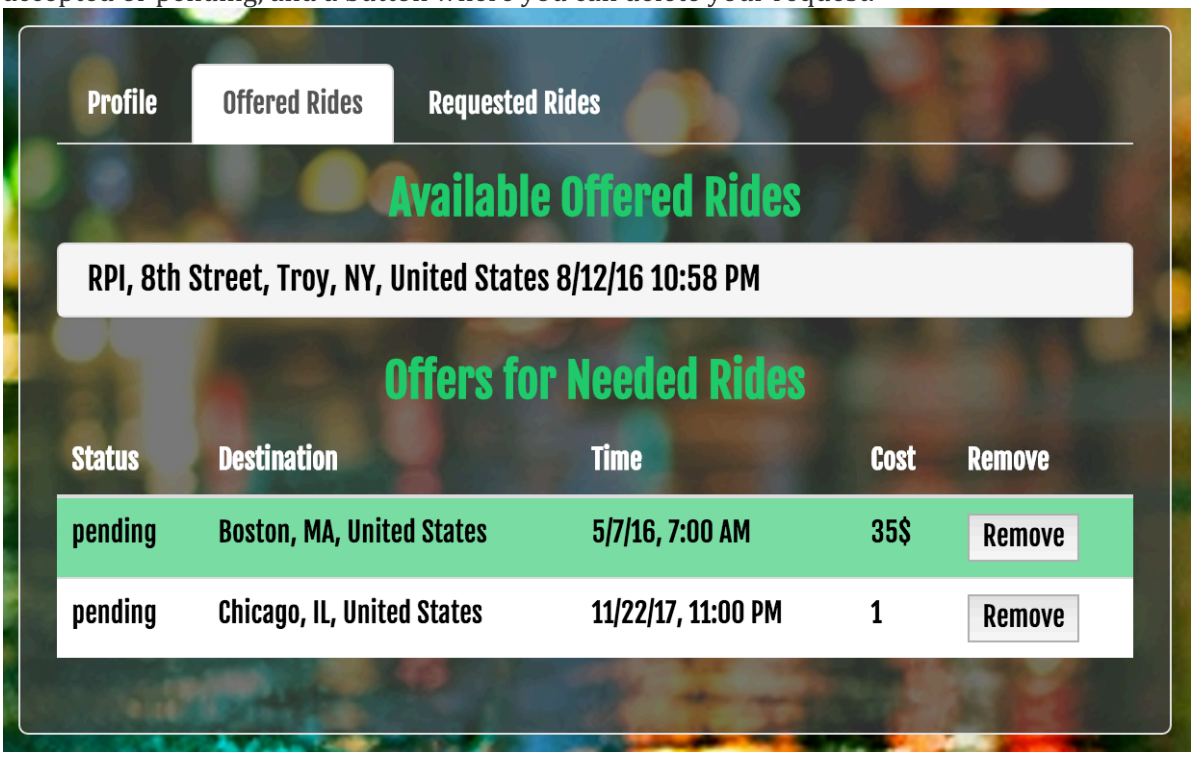
Profile

The settings page is split up among multiple tabs. The first one is profile information. Here they can change their first name, last name and even set their vehicle information. Vehicle information isn't required since not everyone has a vehicle. When creating a ride, the form will pull this information for ease of use.



Offered Rides

The *Offered Rides* tab shows a list of all the rides a user has offered. The first section of this tab the user can view their offered rides. When they click on the ride, it expands showing the amount of available seats, pending and accepted riders, and then a button which will delete that offered ride. From here they can confirm riders for that ride or remove the rider. The next section there is a users offer for needed rides. It shows some basic information along with the status of the ride, accepted or pending, and a button where you can delete your request.

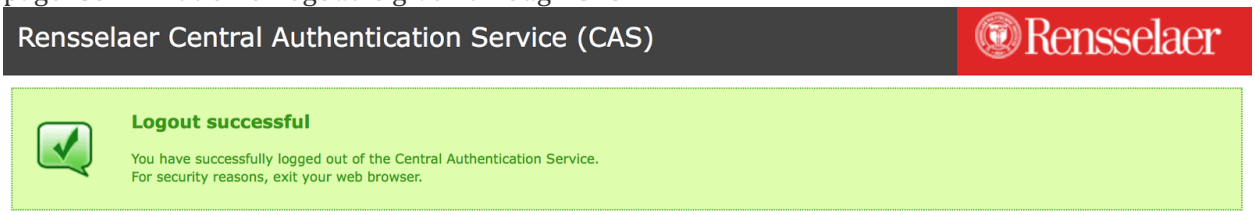


Requested Rides

The *Requested Rides* tab shows a list of all the rides a user has requested. The first section here shows the rides that you need. when clicking on a ride, it also drops down displaying the drivers that have requested and a delete button for this needed ride. Here you can also confirm or delete a driver for this ride. The second section shows any requests a user has on available rides. Here they can see the status of their request as well as delete their request.

Logout

Logout destroys the users session and cookie and redirects them to the CAS Logout Confirmation page. Confirmation of logout is given through CAS.



APIs

The initial intent of the project had us using more APIs. That was not possible due to many companies discontinuing their APIs. In the future we would like to add more APIs to this application.

Greyhound API

We originally planned to integrate the Greyhound API into our project, but it turns out that Greyhound has just started using GPS data from its buses and has not yet made an API publicly available. We also considered listing some popular routes to the page, but ultimately decided that this was superfluous information since users will be better off visiting Greyhound's site directly.

AMTRAK API

AMTRAK's API met the same fate as Greyhound's API. Since we were unable to find and AMTRAK or 3rd party API, we considered adding the most popular trains to a database or statically adding them to the page, but once again, it would be much faster and more beneficial for the user to simply use AMTRAK's site in the first place.

Project Plan

Below is the initial project plan for Take Me Home Tonight.
Week of 2/8

- Finalize project goals, elevator pitch, etc.
- Finish site wireframes and mockups

Week of 2/15

- Begin front-end
- Begin database schema

Week of 2/22

- Finish front-end foundation, including views for each facet of product
- Finish database schema and create database tables from scheme
- Begin advanced CSS/JS implementations
- Begin responsive design
- Begin user authentication

Week of 2/29
- Finish user authentication
- Finish mature CSS/JS implementation/interactions
- Finish responsive design
- Begin populating database with testing data
- Begin adding ability for users to add information

Week of 3/7
- Finish user's ability to add information
- Test front-end and back-end with testing data
- Create midterm presentation
- Practice midterm presentation

Week of 3/14
- Spring Break

Week of 3/21
- Present midterm presentation
- Record and evaluate feedback from presentation
- Reevaluate schedule and created schedule for remaining weeks

Week of 3/28
- TBD

Week of 4/4
- TBD

Week of 4/11
- TBD

Week of 4/18
- TBD

Week of 4/25
- Fix any bugs
- Finish extensive site testing
- Create final presentation

Week of 5/2
- Practice final presentation
- Present final presentation

Methodology

For this project we elected to use the Agile methodology. We had weekly scrum meetings every Sunday afternoon to discuss our progress and assign the next round of tasks. During these meetings we also took some time to debug each other's code and help each other past roadblocks we hit. We also had a significant project backlog since many features either took longer than expected or broke frequently. Using agile kept our progress smooth and constant, helping keep our stress levels lower.

Task Assignments

Rachel:

- Documentation
- Powerpoint
- Landing page
- Planes
- Taxis
- Meet the Team
- Wireframes
- Header
- Footer
- CSS styling
- Code commenting

Shawn:

- Bug fixes
- Code commenting
- Sorting
- Rides
- Ride form
- Heroku server setup

Chris:

- Routes
- Settings page
- Bug fixes
- Code commenting
- Server setup
- Database design and configuration
- Rides
- Notifications
- CAS authentication

Sparky:

- Contacted CDTA for beta access
- Integrated and debugged API
- Documentation
- Powerpoint
- Landing page
- Buses
- Shuttles
- Bug fixes
- CSS styling
- Code commenting

Technologies

The app will use the following technologies:

- HTML5 - We will use HTML5 on the front end to structure all of our pages.
- CSS3 - CSS3 will be used for any custom CSS that will be done throughout the application.
- AngularJS - AngularJS is an open source MVC web application framework. At its core it is an extension of HTML that will allow the team to create a rich and dynamic web application.
- MongoDB - The team will use MongoDB for our database. It is a NoSQL database allowing for dynamic schemes.
- Express - Express is a web framework for NodeJS that will provide many features to build a web application.
- Socket.io - Socket.io is the module we use in order to provide users with notifications about our webiste.
- NodeJS - NodeJS is a javascript based server side language. A main feature of it within web applications is it allows to have a single programmer program both the front and back ends (full-stack developer).
- Bootstrap - Bootstrap will be used within the teams web application in order to keep things responsive and help give it a mobile first feel. In addition to this, UI Bootstrap components will be used instead of regular Bootstrap because they are written in pure Angular to allow for a more efficient and best practice application.
- NPM (Node Package Manager) - NPM will be utilized within the team's project to easily install and keep track of all dependencies used within the project.
- Github - The team will use Github throughout all stages of the project so everyone can access the code, handle version control, as well as track any bugs there may be.