Team Digger Drive

Team Members: Amelia Atiles, Connor Baker, David Ayres, Nicolas Capra
6/19/18

https://diggerdrive.mines.edu

Introduction

For this project we worked with the Digger Drive program, under the umbrella of the Mines Undergraduate Student Government. Digger Drive is a Safe Ride Program designed to be a free, non-judgmental service for Mines students that require a ride home under circumstances where their planned mode of transportation is compromised. The goal of our field session project is to build a web application that can ease the operation of Digger Drive and take pressure and duties off of volunteers as well as improve the ride requesting experience for students. The web application should consist of two major pieces: a volunteer/administrator access side and a client/student facing side. The administrator/volunteer is responsible for handling user details and enabling Digger Drive call center volunteers to both receive ride requests and dispatch vehicles. The general user facing portion is responsible for allowing users to fill out waivers and request rides. The application needs to be as accessible as possible on a variety of different devices, including mobile phones, to ensure ease of use for all parties. Simplicity is key for this project as the client is looking for an outlet to make the service easier and more accessible which will hopefully increase the use and contribute to reaching and helping more Mines students.

Digger Drive Requirements

Functional Requirements:

This project requires a variety of different features to fulfill the requests of the client, including:

- The system uses the Mines MultiPass (Shibboleth single sign-on) service to log students in.
- Once logged in, a user trying to request a ride is required to fill out a waiver if it has not yet been filled out.
- The waivers are manually validated by a system administrator and is presented for approval within the admin panel.
- After the waiver has been submitted and approved, a user is able to request rides using this application, which can:
    - Collect their location (giving them the opportunity to manually enter or automatically discover their position)
    - Allow the addition of other riders (up to a session-specific maximum number, as specified during creation and later adjustment of sessions in the admin panel)
    - Send all relevant information to a call center volunteer
- If the other riders are not registered then the user is notified that there is an invalid rider.
- There is also a service to collect feedback from rider which is conveyed to administrators on the admin panel.
- On the volunteer side there is a permission system with the roles: user, volunteer, and administrator.

- Administrators are able to add and remove of users and change the roles of users, in addition to turning the service on and off.
- Administrators are able to change the waiver text if needed.
- The volunteer level user see's a queue of requested rides, update specific ride statuses and also be able to search for individual riders information from data stored in the database.
- The system manages what each role of the user sees (regular users are not be able to access volunteer or administrator level items).

Nonfunctional Requirements:

The end goal of this project is to have a mobile friendly web app that meets the needs of the client, as stated previously, hosted on the Mines network (CCIT hosting).. The web app needs to have an approachable interface for both users (riders) and volunteers.

System Architecture

Technical Design Issues

- Angular has a very steep learning curve - it was hard to find helpful documentation.
- Learning Angular objects were quite difficult - ran into issues using Observables, which are objects that do not immediately deliver information and instead redefine how the program runs (the order in which it executes)
- Time was spent on research to figure out which tools and which third party software would integrate the best into the project - rather than working on project.
- Working with CCIT was at times difficult - especially communication and return around times that interrupted our flow of work.
- We didn't include the backend Node.js and Express server in the beginning, since we did not think through the backend requirements of the application, so a necessary transition from a purely frontend Angular setup to a mixed Express backend and Angular frontend required restarting much of the project.

Figure 1 shows how the different modules of the webpage interact and direct when called on. It also shows which parts of the front end connect directly or call the back end. The rider, volunteer, and admin panels are set up from the requirements the client described. The client wanted the waiver information to always appear for the rider; first time users and users with waivers not yet accepted are sent to the profile page to view the waiver form and can fill it out if needed. If the rider has used the site before they are directed to the request ride page. Once the user requests a ride, the webpage gets the user's current location from an API call. After the ride form is submitted, the ride information is added to the queue in the back end. For the volunteer portion, they should always view the ride queue which pulls information from the back end. The admin panel has add and remove user functions to add volunteers, users, or other admins which makes calls to the backend and updates information within the database. The on/off function for the admins creates a session object in the database, and is queried by users of all types on pages that require the service to either be on or off.

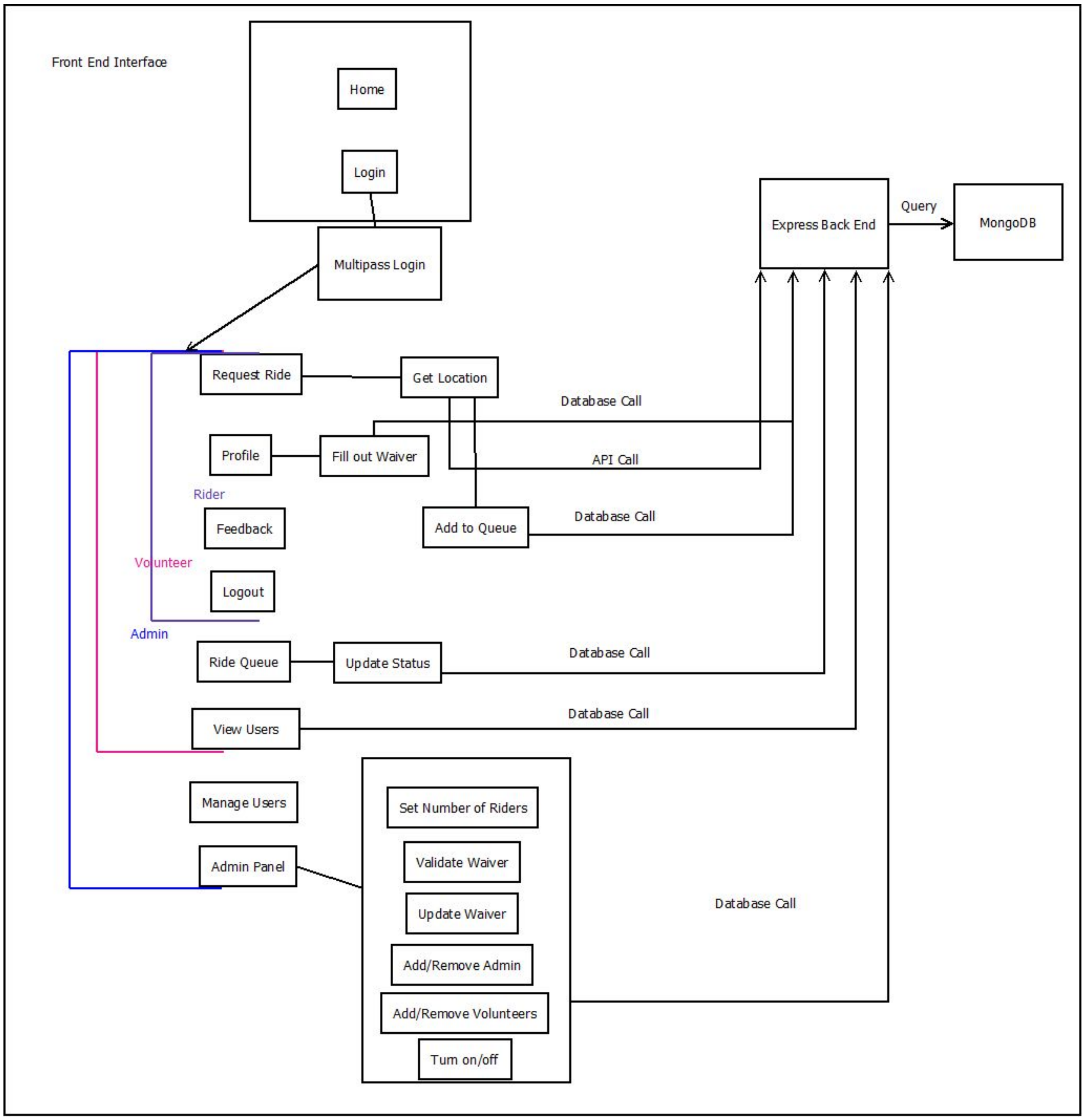


Figure 1: Front End Interface

Figure 2 displays the database schema for the back end to use. The users collection stores data for riders with their MultiPass as the key value. It also stores name, address, and emergency contact information for basic functions of the service (who to pick up, where to drop off, and emergency contact in case of emergency). The MultiPass value connects users to the rides that they've requested in the ride collection. The information is who was the driver, status of ride (complete, requested, in progress), time of ride, who was riding (MultiPass), and a ride ID for sorting and searching. The client wanted to keep track of statistics per session of the Digger Drive service so we created a session collection that holds a session ID and makes grouping rides easier. The waiver's text needed to be updateable for Administrators so it is held within the config collection, which is limited to one entry that can be updated as needed. The feedback collection is loosely tied to users but there is an option to submit anonymously which makes the submitter field "Anonymous" both in the frontend view of feedback and in the database, to ensure anonymity.

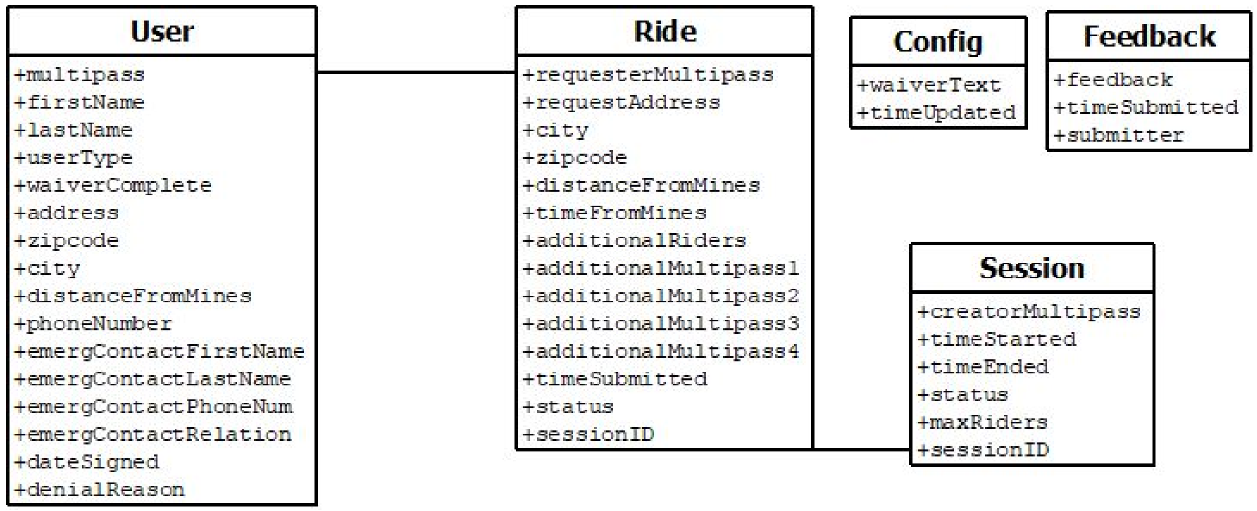


Figure 2: Database Schema

Technical Design

One of the interesting aspects of our project was our self implemented API. The API that we created was a simplistic CRUD (Create, Read, Update, Delete) system used to communicate with the MongoDB database. Each call was executed in a similar fashion to the flow chart shown in Figure 3. We had many different front end buttons/forms/tables/text-boxes that required an API call to perform a CRUD action. Each time one of these calls needed to be made the frontend JavaScript would call a function which would pass the necessary data needed for a API call to the necessary service. The data passed is in the form of a JSON, which worked well since the NoSQL database holds each record as a JSON document. Each time a service function sends a JSON document to the API, a third party query sanitizer checks the document to make sure that it is not malicious and cannot be used to inject (similar to SQL injection) malicious operations.

The frontend then subscribes to the API and waits for data asynchronously while executing other code in the meantime. Once the service receives the request, it creates a POST request and sends that towards the API, where it is received and passed to the appropriate route and model. The route first error checks the data to make sure it is complete and valid, then make a function call to the model for the type of object being operated on, sending any error information back to the user. Once the model receives the function call and data, it makes database query and executes the indicated CRUD action.

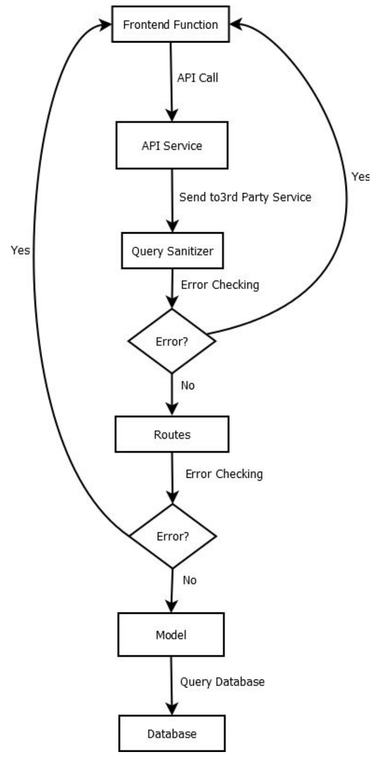


Figure 3: Implemented API Flowchart

Another important aspect of this project and nearly all web applications was the authentication. As described earlier in the functional requirements for this project, it was important to our client that users of all types (standard, volunteer, and administrators) login using the Mines MultiPass (Shibboleth single sign-on) system to log users in. This system is managed by CCIT (the IT department on campus) and is a mechanism to enable students to use the same login (MultiPass) across all of the different campus resources. The client identified that using this system would enable verification that users are students, and ease the use by not adding another login to remember.

In order to enable authentication on the application, we used Shibboleth in conjunction with Passport using JWT (JSON Web Tokens). Shibboleth enables users to log in with their Mines MultiPass, then their user information is passed through the HTTP headers to the application, which is then interpreted by the backend after the frontend makes a call to authenticate the user, Figure 4 displays the flow of information throughout the authentication process. Additionally, if the user has not signed up before, the application “registers” them automatically upon login - creating a user with the first name and last name as provided by Shibboleth. This information is also located in the HTTP headers and can only be interpreted by the application during a backend call, due to the fact that we are using Passenger to run the application, which also manages the environment variables.

Upon successful authentication in MultiPass, the user is taken to a page that fetches all of this information from Shibboleth, registers if necessary, then taken to either the dashboard or profile page. During normal use of the site by users after the initial login (using Shibboleth) each page and information request is authenticated using the JWT assigned after login. This bearer token is stored in the browser’s local storage, and helps to reduce the number of calls to the database that would be needed on each navigation using Shibboleth only. Upon logout, the JWT is deleted from the browser’s local history, and the user is logged out of the Shibboleth or Mines IDP - which ensures security, especially in an application where it is feasible that multiple users will try to use the same computer to request rides.

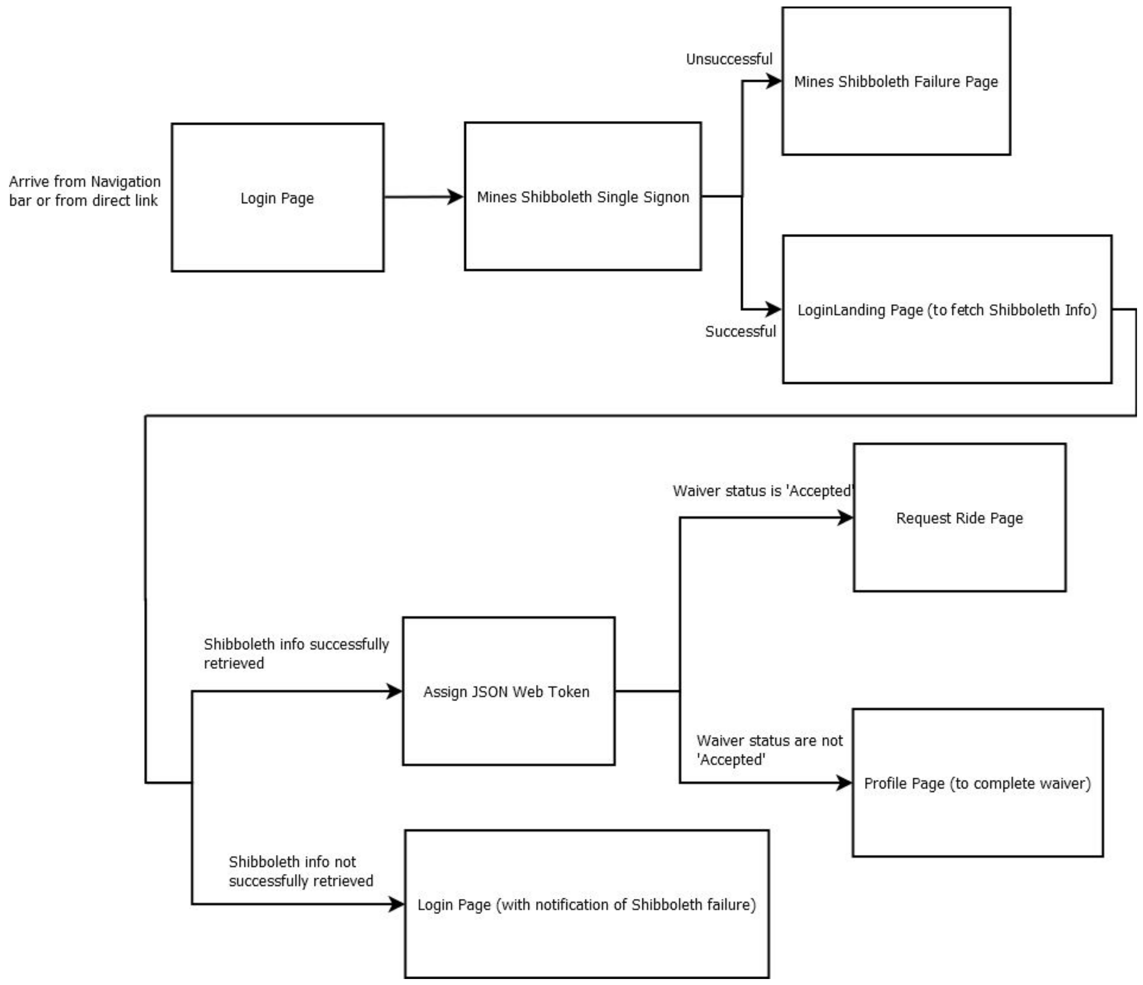

Unsuccessful
Mines Shibboleth Failure Page
Arrive from Navigation bar or from direct link
Login Page
Mines Shibboleth Single Signon
LoginLanding Page (to fetch Shibboleth Info)
Successful
Waiver status is 'Accepted'
Request Ride Page
Shibboleth info successfully retrieved
Assign JSON Web Token
Waiver status are not 'Accepted'
Profile Page (to complete waiver)
Shibboleth info not successfully retrieved
Login Page (with notification of Shibboleth failure)

Figure 4: Authentication Flow Diagram

Decisions

1. We chose to use Angular 6 as the frontend framework
    a. This decision was based off of Connor's previous knowledge of Angular and its ability to be powerful as well as team research on various web frameworks.
    b. Angular is also a very modular based approach to web applications which we found appealing in a framework as it allowed for easy component level changes and minimized dependencies within our code.
    c. Angular also utilizes data binding which allows for dynamic database driven web pages and tables using ng-directives ("ng" for "Angular").
2. The application uses the MEAN Stack - MongoDB, Express, Angular, NodeJS
    a. After research on Angular and consulting with our Advisor, we determined that thanks to the wide usage and recognized flexibility of the MEAN stack, it would prove to be a strong choice.
    b. A NoSQL database was appealing since it is easy to update schemas and is fast, efficient and easily configurable with a web app.

    c. The MEAN stack was also a good choice since everything could be written with JavaScript, which we were already using.
3. We chose to use the Mongoose JavaScript library to communicate with the MongoDB
    a. Because we were already using and learning JavaScript we decided that it would be a good idea to incorporate Mongoose so that we could minimize the different languages within our project.
4. To verify that the users are Mines students the team decided to use the Mines Shibboleth single sign-on service.
    a. This single sign-on service helps verify that the users are Mines students as well as give a unique identifier for our database (their MultiPass).
5. Using SCSS instead of CSS.
    a. We decided to use SCSS (SASS (Syntactically Awesome Style Sheets) CSS) because it allows for much more flexibility and conditional styling rather than the traditional CSS.
6. The website is hosted within the Mines network rather than from a third party
    a. The application is a Mines based project and USG doesn't have to host it themselves.

Results

What we couldn't get:

- Text notifications
- Push notifications for users
- Automatic table updating (on push rather than forced refreshing)

Testing summary

This project has undergone user testing periodically on desktop and mobile devices. We have recently been testing the integration with the Mines server as well. It has worked well in Chrome for desktop and all of the phones browsers that we have access to. Since there are a wide range of viewports we have used Chrome's built in mobile viewing for desktop through the Chrome development tools so that we can check how it looks and respond on multiple different devices. When we find bugs or formatting problems, we go back and tweak. We have used Angular's built in unit testing package to test basic functionality.

Future Work

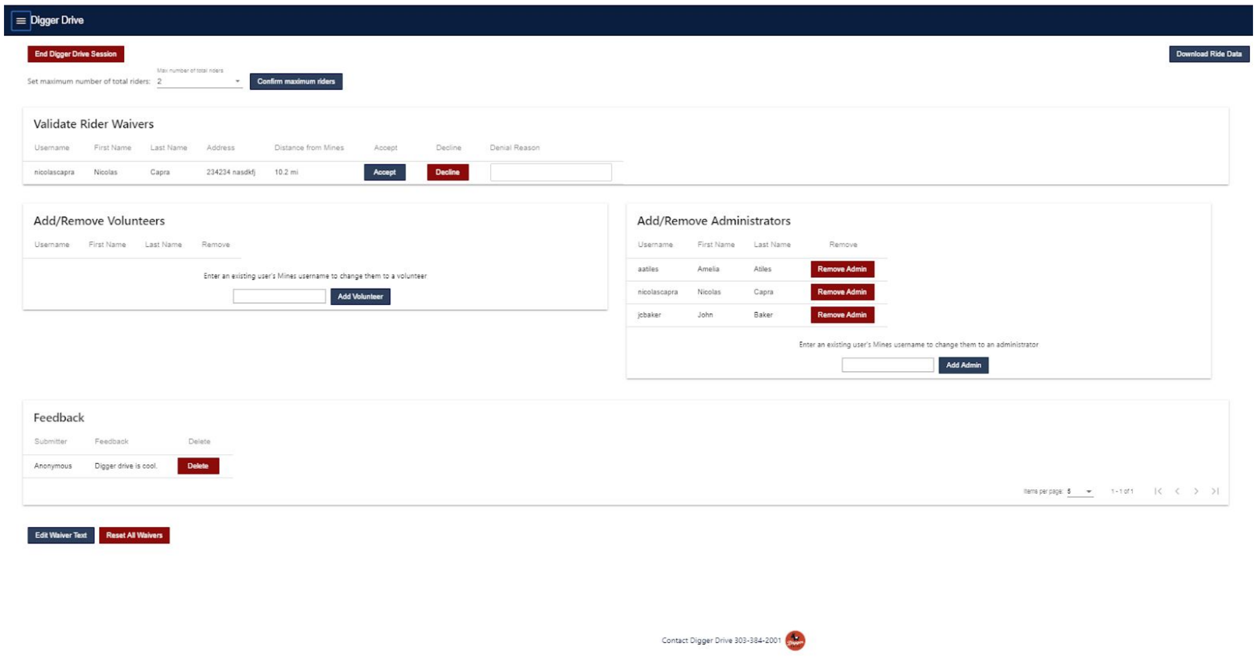
If we were to extend this project we would add texting options to alert riders and navigators of changes. We would work to integrate the program to the Mines M app, so students can more easily access the service. Adding maps for navigators to use could also be done since we already utilize the Google API services. Another addition would be the use of automatic validation for waivers and rides with hard and soft restrictions on distances for home and pickup addresses.

Lessons Learned

We learned how to design a functional web app with given specifications. We learned how to use the Angular framework, along with Typescript, JavaScript, CSS, and HTML, all of which the team had assorted previous knowledge of. Using a NoSQL database rather than a relational database, we learned to handle and manipulate JSON files and query a different type of database. We utilized NodeJS, third party APIs as well as a lot of third party software to make server side work easier. Through the design process we learned how to better assess technologies and how they would impact our final project. This project also gave a very good idea of what a full stack developer has to deal with in terms of web development, since we created a full stack application. Additionally, through our back and forth communication with CCIT we learned how important precise and early communication is for getting a project done on time.

Final Product

The expectation of the client was that the Digger Drive web application would be deployed on the Mines hosting provided by CCIT. We delivered on this expectation, and the application is viewable at https://diggerdrive.mines.edu. Additionally, Figure 5 displays the admin panel, as viewed on a desktop client, and Figure 6 displays the user facing pages (request ride, profile, and feedback) as viewed on a standard mobile device.



Set maximum number of total riders: Max number of total riders 2

Validate Rider Waivers

Username	First Name	Last Name	Address	Distance from Mines	Accept	Decline	Denial Reason
nicolascapra	Nicolas	Capra	234234 nasdkfj	10.2 mi	Accept	Decline	

Add/Remove Volunteers

Username	First Name	Last Name	Remove

Enter an existing user's Mines username to change them to a volunteer

Add/Remove Administrators

Username	First Name	Last Name	Remove
aatiles	Amelia	Atiles	Remove Admin
nicolascapra	Nicolas	Capra	Remove Admin
jcbaker	John	Baker	Remove Admin

Enter an existing user's Mines username to change them to an administrator

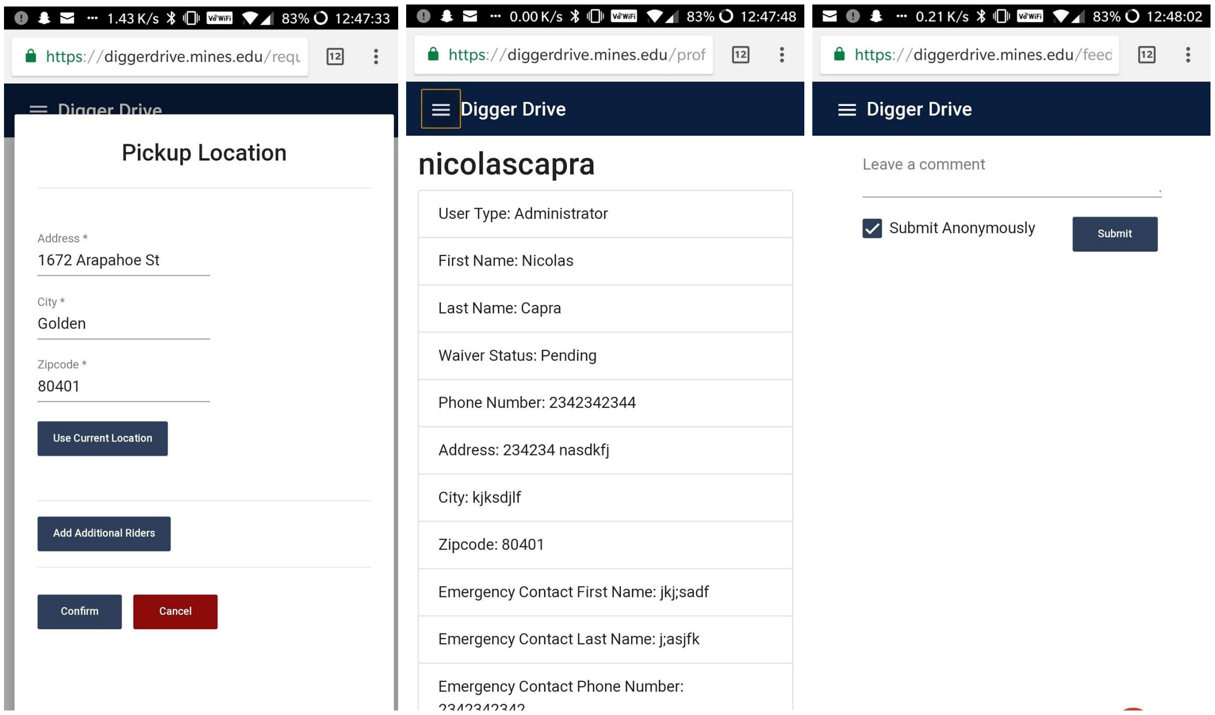
Feedback

Submitter	Feedback	Delete
Anonymous	Digger drive is cool.	Delete

Items per page: 5 1 - 1 of 1

Contact Digger Drive 303-384-2001

Figure 5: Admin Panel



1.43 K/s 83% 12:47:33

https://diggerdrive.mines.edu/requ

Digger Drive

Pickup Location

Address *
1672 Arapahoe St

City *
Golden

Zipcode *
80401

0.00 K/s 83% 12:47:48

https://diggerdrive.mines.edu/prof

Digger Drive

nicolascapra

User Type: Administrator

First Name: Nicolas

Last Name: Capra

Waiver Status: Pending

Phone Number: 2342342344

Address: 234234 nasdkfj

City: kjksdjlf

Zipcode: 80401

Emergency Contact First Name: jkj;sadf

Emergency Contact Last Name: j;asjfk

Emergency Contact Phone Number:

0.21 K/s 83% 12:48:02

https://diggerdrive.mines.edu/feec

Digger Drive

Leave a comment

[✓] Submit Anonymously

Figure 6: Request Ride Dialog, Profile Page, and Feedback on Mobile Device

Appendices

Appendix A: Site Usage Instructions

As an administrator, there is full accessibility to all pages of the website. The most important page is the admin panel. From the admin panel, you are able to see waivers as they are submitted and then accept or decline them. If you choose to decline a waiver there is a small text box for a denial reason which is displayed to the specific user under their profile page information. The two boxes underneath are where you can add admins and volunteers with their MultiPass. The users must have logged in previously before they can be added as a volunteer or admin so that they are in the database. When removing a role from a user, it switches their privileges back to a basic user. Under the role changing boxes is the feedback table. This table shows all feedback that is submitted through the feedback page. The submitter is shown by their Mines username or “Anonymous” depending on whether or not they checked the anonymous checkbox when submitting feedback. The feedback is sorted by most recent and there is a delete button that deletes it from the database. There are two buttons under the feedback table for editing the waiver text and resetting all waivers. The waiver text editor is an HTML rich text editor called quill editor. When the reset all waivers button is pressed there is a confirmation that requires a text field entry and then it resets all waivers to the “Incomplete” status. This should be done once a year around June 1st. There are two buttons also at the top of the page for toggling the Digger Drive service and downloading the ride data. When turning on the service it asks for a max rider count that must be between 1 and 5. Depending on the on/off status of the service the site acts in the ways outlined in the requirements. Once the service is on the max riders can be changed by a drop down box. When the service gets turned off, all volunteer users are demoted back to regular users as well as all incomplete rides in the ride queue are set to canceled. The other admin only privilege is on the “Manage Users” tab. When looking at the info for a specific user, admins can see a button at the bottom where they can edit info of the user.

As a volunteer, there is access to the ride queue, where volunteers can see information regarding the specifics of incoming rides that need to be completed. They can update the status of the ride as it progresses, initially starting as “Requested“, moving to “In Progress” as soon as the volunteer begins driving toward the rider, and finishing as “Completed” when the rider is dropped off. Rides with pickup locations greater than 5 miles away are marked in yellow to indicate such. Volunteers can also submit feedback.

As a rider, there will be limited access to the website, only being able to request a ride, update their information or view the waiver, and also submit feedback. They can input their desired pickup location or use a button that gets their current location and inputs it for them. They can input the MultiPass of up to however many additional the administrators have chosen for the session to ride along with them. Each additional rider must have an accepted waiver form.