EcoPRT

Final Progress Report

CSC 492 Team 14:

Wesley Coats
Hunter Enoch
Jye Huang
Hasham Mukhtar

North Carolina State University
Department of Computer Science

May 8, 2018

Executive Summary

Hasham Mukhtar

Our team is working with the Economical Personal Rapid Transit (EcoPRT) team here at North Carolina State University. The project was funded by a $300,000 grant by the North Carolina Department of Transportation. The idea behind the EcoPRT team is to have autonomous, two-person vehicles that can take people from pick up stations and drive them non-stop to their destination. Riders are able to choose where to get picked up from and where to get dropped off at using a computer or their phones. These vehicles can be installed at a school campus, shopping center, or a neighborhood for people to use.

The EcoPRT team currently has one autonomous vehicle, a server that connects to the vehicle to pull data from it, and a MySQL database to store the information about the vehicle. What they don't have is a way for users to interact with the vehicles or view any of its data. They need what is called a Vehicle Network Controller (VNC). This will provide interaction to users and allow easier testing for their vehicles. In order to make VNC, we have been tasked with improving the existing database, creating a front end that must be used by admin users and riders, and a web server. The web server will be the intermediary to get data from the database and send it to the web client and vice versa.

At the end of the semester, we completed all the Admin Functionality and the database. We completed our first iteration which involved creating REST APIs to pull and put data into the database for admin users. The APIs pulls/puts data about vehicles, nodes (which are different points the vehicle can go to), edges (the path that a vehicle will travel between nodes), and rides (information about rides that are currently in progress or completed). Originally we had basic HTML forms to add data and show the data as a list at the top of the page. These forms had to be filled in manually by a user and then added by clicking an add button.

In our second iteration we updated our web page to have a Google Map that admin users can interact with. The web page has dropdowns on the left side for the vehicles, nodes, edges, and rides. These dropdowns can expand to show more information about the specific sections. The data is shown as a table for easy readability. The Google Maps on the web page shows all the nodes in the database as blue dots and red lines between them that represent edges. Any vehicles that are currently carrying out a ride is shown moving on the map as well on an edge. Admin users can also add these items by clicking on a plus button which brings up an add form for that item. Unlike our previous forms, these form are better organized and can have data auto filled by clicking on the map when appropriate. For example, if you want to add a node, instead of manually entering coordinates, you can click on a spot on the map to autofill the numbers. Also we have added a home page, sign up page, login page, and navigation bar that can take you between them. We created an admin account that we have been using for our development and testing. In our final iteration we added editing and deleting functionality for the Admin for nodes, edges, and vehicles. These actions can be found in the dropdowns for the specific item you are looking to edit or delete. We also added the ability to click on the icons to get more information with an info bar at the bottom of the screen.

Project Description

Wesley Coats, Hasham Mukhtar

Sponsor Background

The main focus of EcoPRT is to shrink and lighten both vehicles and guideway as much as possible, both to save space and to eliminate unneeded costs. These reductions yield 2-person, autonomous EcoPRT vehicles that weigh less than 1,000 pounds and cost approximately $10,000 each, along with guideway that costs approximately $1 million per mile installed. The cost is what makes EcoPRT revolutionary. It is quick and easy to install and takes up much less space compared to other transit systems, that it costs significantly less. Also, they can be installed anywhere because it uses already existing infrastructure like roads and sidewalks. For the first phase of this project, EcoPRT will use these vehicles on the Oval at NC State. The group is lead by Dr. Seth Hollar from the ECE department and is comprised of graduate students here at NC State. We work with three graduate students named Tyler Orr, Abhimanyu Kumar, and Weijia Li. Tyler is a CSC graduate student, Abimanyu is an ECE graduate student, and Weijia is a CSC graduate student. Tyler is our main point of contact for this project.

Sponsor Contact Info:

Dr. Seth Hollar

Teaching Assistant Professor

Electrical & Computer Engineering Department

Email: sehollar@ncsu.edu

Tyler Orr

Computer Science Graduate Student

Email: jtorr@ncsu.edu

Abhimanyu Kumar

Electrical & Computer Engineering Graduate Student

Email: akumar24@ncsu.edu

Weijia Li

Computer Science Graduate Student

Email: wli21@ncsu.edu

Problem Statement

The need for this project is to create a Vehicle Network Controller (VNC). This will provide interaction to users. Admin users will be able to monitor the vehicles, nodes, edges, and

rides. They can assign vehicles to different edges and issues commands to stop or start at any time. Riders can request rides to go from one node to another for their trip. All this information needs to be stored in a database connected to vehicle and web servers.

Project Goals and Benefits

The main focus for our senior design team is on the database and user side and less on the vehicles, its server, and associated algorithms. Table 1 refers to the different terms we use for the project. The database needs to be able to hold telemetry of the vehicle, as well as various nodes along edges taken by vehicles. For the sponsors, we will also be delivering an EcoPRT application that allows riders to request rides and administrators to oversee all activities as well as add or edit a majority of database information. This will have the front end user interface connected to a web server that is connected to our database implementation. The project can eventually be expanded and scaled to serve larger communities such as entire college campuses or the triangle area. The benefit of us doing a front end is to help the EcoPRT team when testing the actual vehicle. If they can assign a vehicle to a path using the front end, then in real life they can see if the vehicle properly follows the path it should. The same goes for when issuing commands and seeing the vehicle data on the front end change accordingly.

Term	**Description**
Vehicles	The autonomous vehicles that someone can ride
Telemetry	Information about the vehicles - Includes: battery life, coordinates, steering angle, heading angle, current task
Vehicle History	This is a collection that holds all the telemetry for all the vehicles in the system
Nodes	Points on the map that the vehicle can drive to - Can be of four types: pickup station, charging station, docking station or just a normal point on the path
Edges	Path between two nodes
Rides	Information about all rides that have taken place or are in progress - Includes: vehicle used, user who requested vehicle, stations chosen, time of pickup and dropoff, and whether the ride was completed.
Rider	Users in the system who can only request, edit and cancel rides. Can also view their ride history

Admin User	Users in the system that can view all aspects of the system like vehicles, nodes, edges, and rides. They can also issue commands to vehicles and tell them what stations to go to.

Table 1 - Glossary

Development Methodology

As a team, we decided to work in parallel. This way we can get more work done in less time. We have split the work up where Jye and Hasham are working on the front end and the Hunter and Wesley are working on the database and backend. We meet with our sponsor every Wednesday in Engineering Building III on Centennial Campus. We have agendas prepared for those meetings and send them out to Tyler, Weijia, and Abhimanyu the night before. During those meetings we bring up any concerns and questions we have and show them our progress at that time. At the end of the meetings, we come up with action items that we need to complete for the next meeting. This is helpful because this gives us clear goals every week and helps to keep us focused. After the meeting we send the minutes to all those that attended.

We have Github issues that describe the different features we should have for our project. When someone completes an issue, they close it. We have a master branch that we all work on. Since our two groups are not working on the same code, we don't have any issues when pushing or pulling code. We use Slack as our main channel for remote team communications. If we are not in the same room together when coding, we will post in Slack when changes are made. This way we can prevent any merge conflicts.

Challenges and Resolutions

Technical Challenges

One challenge was choosing what technologies we were going to use to complete our projects. This is a challenge that most teams face, but that doesn't make it any less of a challenge. There are a lot of different technologies that can accomplish the same thing, especially for databases. We chose our technologies based on what we already knew individually so that we didn't have to learn all the technology from scratch. There were those of us that had experience with web servers, database, and web pages, so we had them choose what technology would work best together.

Another challenge came from terminology for certain parts of the project. We were calling certain things one way when the sponsors were calling it another. This came to light when we were going over out database schema which caused us to redo some of our tables and fields. For example, when a vehicle goes from one point to another, it does so using an edge between the two points. We were calling that a path which confused our sponsors. They explained the difference between an edge and a path. Another example, was for what the different points that vehicle can go to. We assumed the only points were pick up stations, but in reality the points are called Nodes that could be a station. So now before we work on a new part of the project, we make sure we get the terminology right by asking our sponsors that we are properly calling something the right term. This way everyone doesn't get confused and cause delays by having to redo something.

A big challenge we've dealt with was documentation. Our sponsors have pointed out multiple times about how we write our documentation. Sometimes we write too much, not enough or we forget altogether. For example, our installation guide for our first iteration didn't have instructions for Linux which is what our sponsors use. None of us use Linux so we didn't even think about that. We also have trouble deciding where to put the documents. Currently we have documents in Google Drive and Github Wiki. Our sponsors suggested we add installation instructions to our readme instead of the Github Wiki, which was a good idea because it makes it easier to find. We've never had to do documentation like this so it's helpful for them to let us know how to do it properly. We also appreciate their understanding and not getting angry with us.

Resources Needed

Hunter Enoch, Hasham Mukhtar

The following are the technology we need to get our project complete.

Technology	Version	Description	Obtained/Not Obtained
VM Web server (requested from NC State IT)	Ubuntu 16.04.3	Used to host the webserver on. We will use Ubuntu as the operating system, since our sponsors use Ubuntu on their own system connecting with the vehicle. This way they will be able to easily deploy our web application beside their own code whenever they need to.	Obtained
NodeJS	8.9.4	Web Server software. This will run on the web server and will serve the web application and connect the web application to the database. It will contain a REST API. The REST API will allow the web application and other services to access the functions of our web application.	Obtained
MongoDB	3.6.3	The database we will be written using MongoDB. The web server will connect to it. This database will hold information about vehicles, edges, rides, users, nodes, and user authentication.	Obtained
Bootstrap	4.0.0	Using Bootstrap will gives us more freedom to customize the look and feel of our web application. It also makes it easier for us to make a	Obtained

		mobile friendly version of the web application.	
AngularJS	1.6.8	Will be used for the logic of the web application. It works very well for data binding which will be the bulk of our web application since we will be pulling all our data from the database.	Obtained

External Dependencies and Interfaces

The sponsors have data that they record from the vehicles that will be given to us for testing and simulations.

System Requirements

Jye Huang, Hasham Mukhtar

Overall View

The overall system consists of a database that stores all data, a web application that users of the system will interact with and a web server that connects the database and the web application. The database stores information about the vehicles in EcoPRT, account information, and information about the rides, edges, and nodes. There are two types of users for the system: an administrator user and a rider. These two users should be able to sign up and login to the system. Also, both users will see a map of the area with icons representing nodes, edges, and vehicles.

A rider is a user of the system that can only request, edit and cancel rides. They should be able to view a history of their rides on the web page. The administrator user can do much more. The administrator user will be able to add, edit, and delete nodes, vehicles, rides, and edges. The administrator will be able to see a list of all the information that is in the database on the web page. Also, the administrator will be able to turn vehicles on or off using the web page.

Formal Requirements

1. Functional Requirements
    1.1. Admin Functionality
        1.1.1. Admin users shall be able to view nodes on map
        1.1.2. Admin users shall be able to view edges on map
        1.1.3. Admin users shall be able to view vehicles on map
        1.1.4. Admin users shall be able to view rides on map
        1.1.6. Admin users shall be able to edit nodes
        1.1.7. Admin users shall be able to edit edges
        1.1.8. Admin users shall be able to edit vehicles
        1.1.9. Admin users shall be able to edit rides
        1.1.10. Admin users shall be able to delete nodes
        1.1.11. Admin users shall be able to delete edges
        1.1.12. Admin users shall be able to delete vehicles
        1.1.13. Admin users shall be able to delete rides
        1.1.14. Admin users shall be able to add vehicles
        1.1.15. Admin users shall be able to add nodes to map
        1.1.16. Admin users shall be able to add edges to map
        1.1.17. Admin users shall be able to add rides
        1.1.18. Admin users shall be able to disable/enable vehicles
    1.2. Rider Functionality
        1.2.1. Riders shall be able to view nodes on map
        1.2.2. Riders shall be able to request rides
        1.2.3. Riders shall be able to view a history of their rides
        1.2.4. Riders shall be able to edit their ride
        1.2.5. Riders shall be able to cancel their ride
    1.3. Signup

1.3.1 Users shall be able to sign up by entering their first name, last name, email, username, and password

1.4. Login

1.4.1. User shall be able to authenticate with their username and password chosen at sign up

2. Non Functional Requirements

2.1. Database should be able to scale to fit larger areas in future iterations

3. Constraints

3.1. Must use Google Maps API to show the map that the Admin and Riders will see

3.2. Components of system must be easily added for future features

3.3. Database must be used to store all information needed

3.4. Create REST APIs for user functionality

Design

Hasham Mukhtar

Architecture

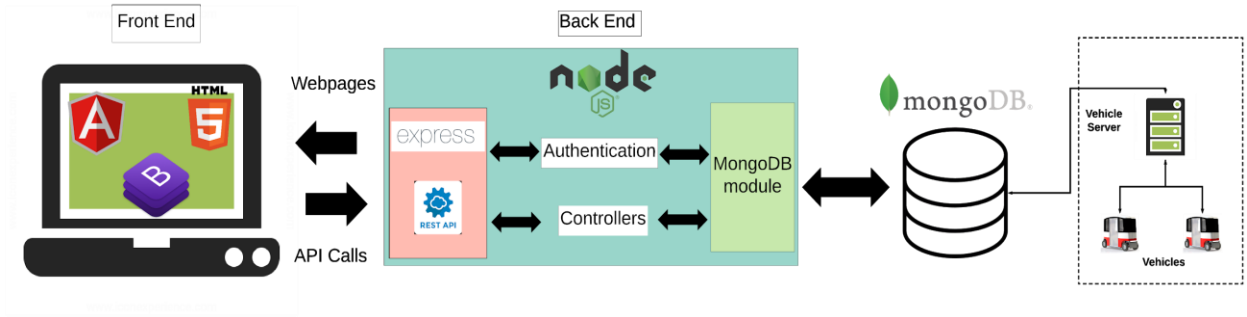

Figure 1 - Architecture Diagram

Figure 1 represents our architecture diagram. The diagram shows the database at the top which is made using MongoDB. We chose MongoDB because it works well with NodeJS, which is the language our server is using. The database stores information about vehicles, nodes, edges, rides, users, and user authentication. Our backend is using NodeJS which communicates with the database vice versa. Inside the NodeJS server, we have Express which loads and sends the web pages to the front end that the users will interact with. Express also creates REST API endpoints. The REST API endpoints call functions that retrieve and add data to the MongoDB database through the MongoDB npm module. One of the main reasons we are using MongoDB is because of NodeJS. MongoDB has seamless integration with NodeJS. In addition, during our research, we learned of MongoDB's ability to scale (MongoDB, 2018) and handle large loads of requests (Performance, 2018).

The front end is made using HTML 5, AngularJS, and Boostrap. Bootstrap allows us to customize the look and feel of our front end to how we want. It also makes it easier to create a mobile friendly version should the sponsors want one in the future. HTML 5 is the current version of HTML that we have to use to make web pages. Angular JS is used for the logic of the web application. It works very well for data binding. There is a single html page that the user stays on at all times. That html page has an angular UI-View DOM element in it. The UI-View loads a snippet of html into it depending on what the URL is. The mapping of URL to html snippets is an angular routes config. Each page also is mapped to a angular controller. The controller contains all the logic of that page. The controller retrieves and adds data through API calls that communicates with API endpoints on the NodeJS server. Then those endpoints will call a function, communicate with the MongoDB module which will retrieve or add data and then send the response back to the front end for the user to see.

The last areas of the EcoPRT system are the vehicle server and vehicles themselves. We are not in charge of these areas for our project, but we have it represented in the figure inside a dotted box. The vehicles send data about themselves to the vehicle server which will communicate with our MongoDB database to store data. Then our NodeJS server reads and sends that data to our front end for a user to view. The server will also send commands from the front end to be stored on the database that will be read by the vehicle server to send to the vehicles.

Database

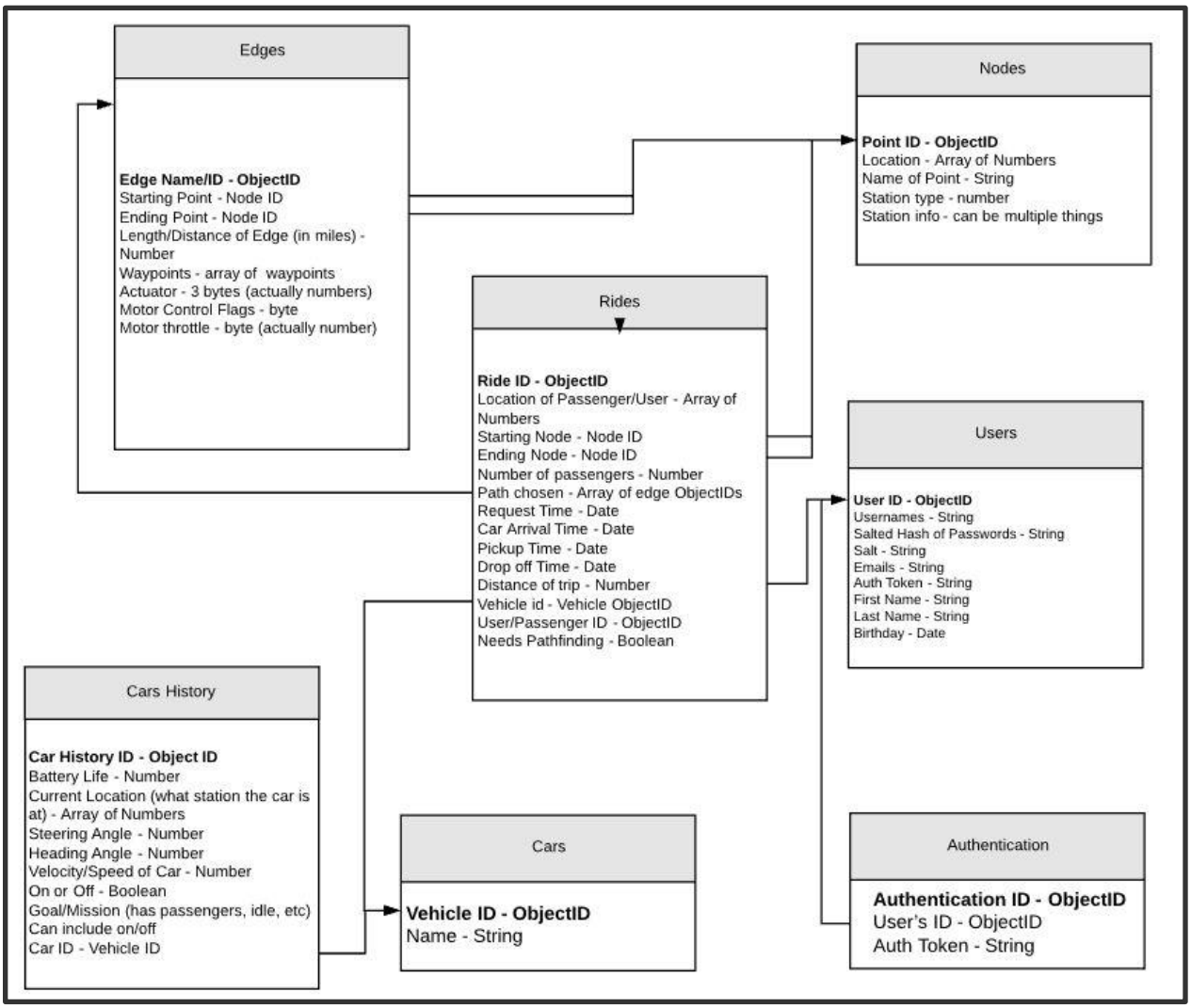

Figure 2 - Database Entity Relationship Diagram

Figure 2 represents our database entity relationship diagram. Each box represents a collection that the database will have. We have collections for vehicles, nodes, edges, rides, users, vehicle history, and authentication. The arrows from a box to another represents that those collections have a relationship. Edges are connected to the Nodes collection because there is a starting and ending node for an edge. Rides also have nodes which are where someone is picked up and dropped off so they will also need to be connected to the Nodes collection. Cars History is the history about a certain vehicle so it will need to be connected to the Cars collections. Rides also have a vehicle that completed the ride and that's why it needs the Cars collections. It also has a user that requested the ride, so it need to be connected to the Users collection as well. Authentication is connected to Users because each user has an authentication token that is used when a user is logging in.

GUI/Wireframes

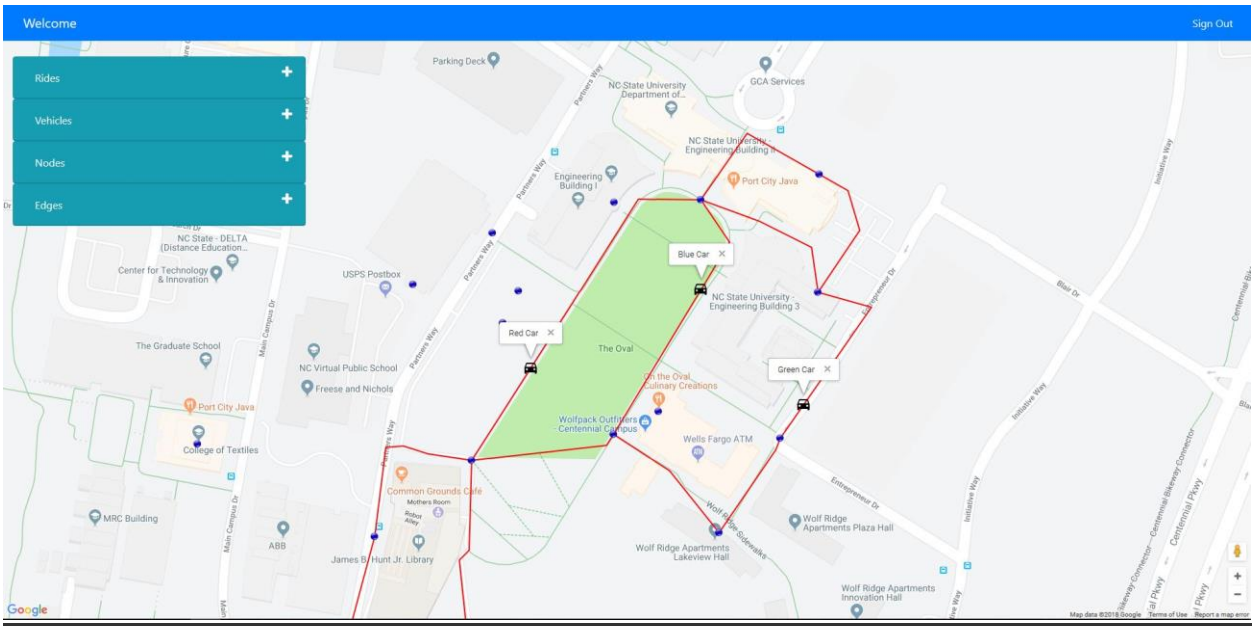

Figure 3 - Admin Page

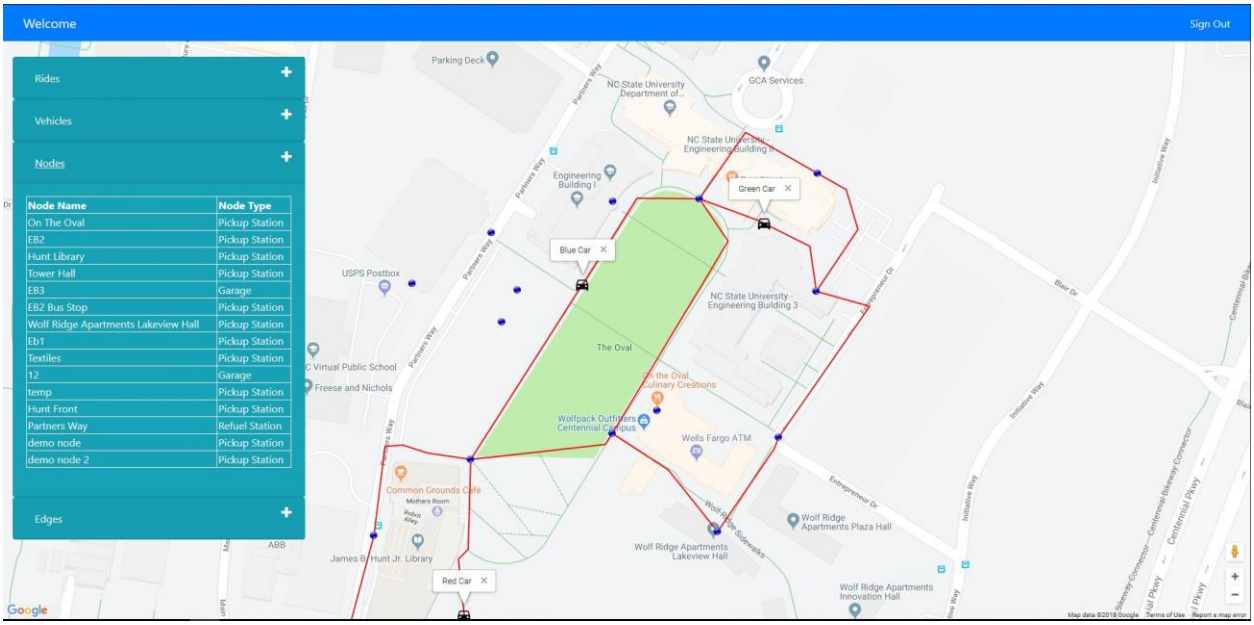

Figure 4 - Admin Page, Node dropdown opened

Figure 5 - Edge Clicked on with Info Bar

Figures 3 and 4 show what our Admin page looks like. There is a Google Map showing the Oval that an Admin will see when they log in. The map has blue dots that represents nodes, red lines that represent edges between those nodes, and black car icons that represent vehicles that are currently in a ride. In Figure 3, you can see dropdowns on the left side which has more information about rides, vehicles, nodes, and edges. Figure 4 shows what the expanded Nodes dropdown looks like. It shows a table of all the nodes in the database and on the map. The other dropdowns look similar to this one. When an icon is clicked on the map, it will change color and open an info bar at the bottom of the screen as shown in Figure 5.

Figure 6 - Rider User page

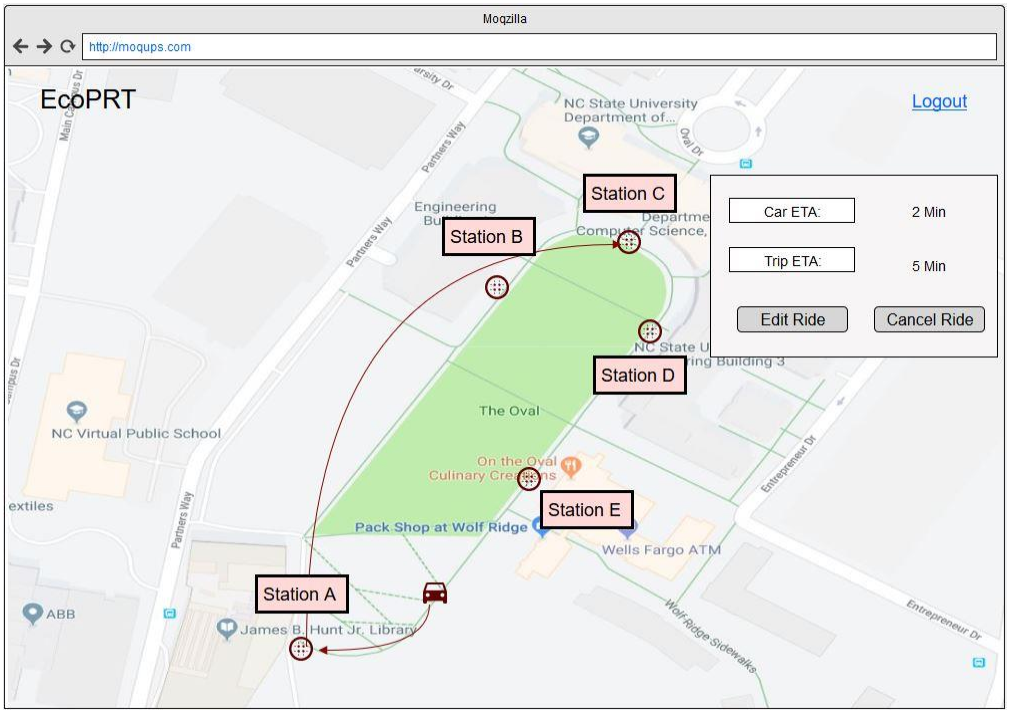

Figure 7 - Rider User Page, Ride in Progress

The wireframes are based on our interpretation of the requirements for riders. The rider will see a Google Map, similar to the Admin page, with all the nodes that a rider can get picked up from and dropped off at, as shown in Figure 5. The rider can click a request ride button that will bring up a form for rider to select their starting and ending node. When they request a ride, they will get a box showing the Estimated Time of Arrival for the vehicle to pick them up. The rider will also see the Estimated Time of Arrival for when they will arrive at their destination. Rides will also have the ability to cancel or edit the ride as shown in Figure 6. When the vehicle is moving, it will be shown on the map for the rider to view.

Implementation

Hunter Enoch, Jye Huang

Our current code implementation has a MongoDB database and Node JS server running. The root of the project folder contains the Node JS script and a package.json file. The package.json file contains information about the dependencies required for the project. There is a public folder in root that contains the pages displayed to the user. The user cannot see anything outside of this folder. It contains the html, js, css, and images used. for the web pages. The root also contains a mongo folder with a script to load the database with test data.

The root folder of the project also contains a folder called “test” and a file called “index_noSim.js”. The file “index_noSim.js” is the file that we perform the unit tests on. It contains the same exact code as our server, “index_v2”, but with the simulation code removed. The folder called “test” contains a file called “indexTest.js”, a file called “waypoint.txt” and a folder called “casperTest”. The file “indexTest.js” contains all the code for the unit tests using the tools Mocha and Chai-HTTP. The file called “waypoint.txt” is an example waypoints file that is used by the unit tests. Inside the folder called “casperTest” is a folder called “img” and two files called “adminTest.js” and “userNavigationTest.js”. The file “adminTest.js” contains code for automated testing of admin functionality and the file “userNavigationTest.js” contains code for automated testing of user navigation on our EcoPRT web application. Both of these files use CasperJS and take screenshots after each step taken in the tests. The screenshots are saved in the folder called “img”.

Code coverage for the project is implemented with a tool called Istanbul. The output from the tool is stored in a folder called “coverage” in the root directory. Inside the folder “coverage” is a file called “index.html”. When this file is viewed in a web browser, the current coverage can be seen in a user friendly format. Figure 9 in the testing section shows a screenshot of the user friendly format. View the testing section in the Development Guide for details on how to add more test cases and how to run the tests. Figure 8 shows our project directory.

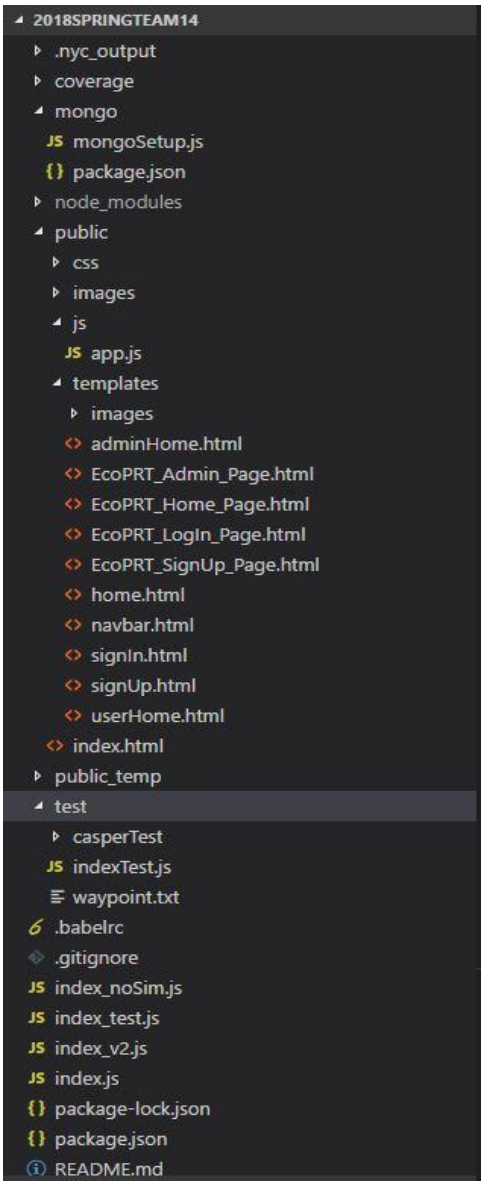

Figure 8 - Project Directory

We broke our work into iterations and they are outlined below.

Iteration 1: Basic Admin Functionality Part 1 (Add/View) with Basic HTML Forms

Start Date: February 7, 2018

Due Date: February, 21, 2018

- Admin users can view Edges, Vehicles, Rides and Nodes on a web page (Requirements 1.1.1 - 1.1.4)

- Admin can add Edges, Vehicles, Rides, and Nodes to database (Requirements 1.1.14 - 1.1.17)
- Show data from database as a list
- Have basic HTML forms to add information
- Have REST APIs for getting data from and adding data to the database

Iteration 2: Visually appealing and interactive Admin Web Page

Start Date: February 21, 2018

Due Date: April 4, 2018

- Have a Google Maps that shows Oval (Requirement 3.1)
- The map shows nodes, vehicles, and edges that the Admin user can interact
- Have icons for nodes and vehicles
- Have lines drawn on the map between nodes to show edges
- Have dropdown menus to get more detailed information about vehicles, nodes, edges, and rides.
- Added add icon to dropdown so Admins can add vehicles, nodes, edges, and rides
- Had system ready to integrate with sponsors code and all necessary documentation
- Users can sign up (Requirement 1.3.1)
- User can login as an admin (Requirement 1.4.1)
- Did some testing of the front end UI

Iteration 3: Basic Admin Functionality Part 2 (Edit/Delete) with Interactive Web Page

Start Date: April 4, 2018

Due Date: April 25, 2018

- Fixed any bugs from integration with sponsor's code
- Admin can delete vehicles, nodes, edges (Requirements 1.1.10 - 1.1.13)
    - Added icon to dropdown to do this
    - Added an alert box when deleting items
- Admin can edit vehicles, nodes, edges using forms (Requirements 1.1.6 - 1.1.9)
    - Added icon to dropdown to do this
    - Added edit forms for the items
- Icons change color to show that they are clicked on

- There is a new pop up box at the bottom of the screen to show information about the selected item
- Tables in Dropdowns are scrollable
- Vehicle icon rotate based on the heading angle
- Admin can enable/disable vehicles using the dropdown table for vehicles (Requirement 1.1.18)
- Finished front end UI and backend JS testing

Test Plan and Results

Jye Huang, Hunter Enoch, Hasham Mukhtar

Acceptance Test Plan

There are two portions of acceptance testing. For the project, we have created web pages for our front-end. Additionally, we have built a database and made a REST API to connect this database to our front end. Because of this, we have two acceptance test suites: one for the REST API and another for the front-end web pages.

How to Set Up Testing

In order to run the tests, a terminal window must be open to the project directory and the latests version of MongoDB is installed. The command “sudo npm install” must be run in that terminal window. This will install all the dependencies that are necessary for testing the REST API. In the same terminal window, running the command “mongod” will start running the database that is needed for the tests. There is no test data that needs to be loaded. All necessary data for testing is loaded when the tests are run.

Front-End Testing

We use a tool called CasperJS to test our EcoPRT web application. This tool allows us to automate user interactions with our web application. With this tool, we can insure that the correct web pages are being loaded and the links on our EcoPRT web application go to the correct pages. In addition, CasperJS allows us to make sure different elements on our webpages are labeled properly. If the items are labeled incorrectly, CasperJS will be unable to locate and automatically click on the desired web element. CasperJS also allows us to automate some of the black box test cases that are listed in the table below, such as as black box test ID adminLogin.

REST API Testing

We use two tools to test our REST API: Mocha and Chai-HTTP. Mocha is a JavaScript framework for Node.js that allows for Asynchronous testing while Chai-HTTP is an assertion library that runs on top of Mocha. Because all the functions used for setting up and interacting with our MongoDB database are called through the REST API Calls, these functions are tested through the REST API calls.

Black Box Test

Preconditions

For all black box tests, we are assuming that the web server is running and the database is loaded with information for the administrator account with username “admin@example.com” and password “admin”. The EcoPRT web application must also be opened to the homepage.

Test ID	Description	Expected Results	Actual Results

adminLogin	Steps: 1. Click Log In on navbar 2. Type In “admin@example.com” in the username field 3. Type In “admin” in password field 4. Click “Sign In”	● Admin is Authenticated ● Admin is sent to Admin Page	● Admin is Authenticated ● User is sent to Admin Page ● User sees google map
createAnAccount	Steps: 1. Click Sign Up on navbar 2. Type “Username” in field labeled username 3. Type “First” in field labeled first name 4. Type “Last” in field labeled last name 5. Type “06/09/1969” in field labeled birthday 6. Type “email@example.com” in field labeled email 7. Type “Password” in field labeled password 8. Type “Password” in	● Account should be created ● User should be redirected to sign in page ● Rider Log in works with the new username	● Account created ● User is redirected to sign in page ● Rider Log in works with the new username

	field labeled confirm password 9. Click “Sign Up”		
addNode	Preconditions: ● The user is signed in as “admin@example.com” Steps: 1. Click the + sign beside the node drop down 2. Type “node” in field labeled node name 3. Type “35.77” in the field labeled Degrees of Latitude 4. Type “-78.67” in the field labeled Degrees of Longitude 5. Click Add Node	● Node should be created at coordinates 35.77 latitude and -78.67 longitude ● Node should have name and type filled in ● Node will appear in the drop down when created	● Node is added at coordinates 35.77 latitude and -78.67 longitude ● The node is added to the drop down menu on the left hand menu. ● It shows the node name “node” and node type “Pickup Station”
riderRequestRide	Preconditions: ● Rider account “rider” exists in the database. ● Rider functionality is completed Steps: 1. Rider Log in 2. Click Request a ride 3. Choose	● Ride with beginning and ending node selected should be selected ● Will show vehicle’s path to the starting node to pick up user with an ETA	● Rider functionality has not been implemented

	Starting Node 4. Choose Ending Node 5. Click Request	● When user is picked up, it will show the vehicle's path to ending node with an ETA	
addVehicle	Preconditions: ● The user is signed in as "admin@example.com" Steps: 1. Click the + sign beside Vehicles dropdown 2. Type "vehicle" in field labeled vehicle name 3. Click Add Vehicle	● A vehicle should be added to the list of vehicles in the dropdown	● Vehicle is added to the the table of vehicles in the dropdown menu on the left side of the screen
removeNode	Preconditions: ● The user is signed in as "admin@example.com" ● Steps for Black box test addNode are performed Steps: 1. Click on "Nodes" in the left menu 2. Click the "-" symbol next to node labeled "node" 3. Click on	● The node labeled "node" is removed from the table and the database	● The node labeled "node" is removed from the table and the database.

	“OK” in the pop up box		
removeVehicle	Preconditions: ● The user is signed in as “admin@example.com” ● Steps for Black box test addVehicle are performed Steps: 1. Click on “Vehicles” in left menu 2. Click on the “-” next to vehicle labeled “vehicle” 3. Click on “OK” in the pop up box	● The vehicle labeled “vehicle” is removed from the table and the database	● The vehicle labeled “vehicle” is removed from the table and the database
sendCommandAsAdmin	Preconditions: ● The user is signed in as “admin@example.com” Steps: 1. Click “Vehicle” in left menu 2. Select Command from command dropdown next to the vehicle name 3. Click on send command	● Vehicle should receive command and act accordingly on the map and with a status update in the table	● This functionality has not been implemented
addEdge	Preconditions: ● The user is	● Edge should be added with	● The edge is added to the

	signed in as "admin@example.com" ● Nodes "On The Oval" and "EB2" exist Steps: 1. Click the "+" beside "Edge" 2. Choose starting node "On The Oval" 3. Choose ending node "EB2" 4. Type "42" in field labeled length 5. Add a waypoint file "waypoints.txt" with the file picker 6. Click "Add Edge"	starting node at "On The Oval" and ending node at "EB2"	table in the drop down on the left side. ● The table shows the starting node of "On The Oval", ending node "EB2" and the length "42". ● The edge is drawn on the map between the two nodes chosen. The line is colored red
editEdge	Preconditions: ● The user is signed in as "admin@example.com" ● Steps for Black box test addEdge are performed Steps: 1. Click "Edges" in the left menu 2. Click the pencil icon next to the edge with starting node "On The	● The length in the edges table in the left menu is changed to 22	● The value for length next to the edge with starting node "On The Oval" and ending node "EB2" is 22.

	Oval" and ending node "EB2" 3. Type "22" in the field labeled length 4. Click "Edit Edge"		
riderEditRide	Preconditions: ● Rider account "rider" exists in the database. ● Rider functionality is completed Steps: 1. User logs in as "rider" 2. User requests a ride 3. They choose starting and ending node 4. They click request 5. They click edit Ride 6. They choose a new starting and ending node 7. They click request	● The vehicle will show up on the map and have a different destination path than the first request. Will still have an ETA for both	● Rider functionality is not implemented

riderCancelRide	Preconditions: ● Rider account "rider" exists in the database. ● Rider functionality is completed Steps: 1. Rider logs in 2. They click request ride button 3. They choose starting and ending node 4. They click request button 5. They click the cancel ride button	● The vehicle and the paths on the map will disappear and the map will reset to how it was before the user requested a ride. Will show the nodes and the request a ride button for the user to click on.	● Rider functionality is not implemented

Unit Test Plan

Similar to the REST API Testing, the same two tools: Mocha and Chai-HTTP, are used for unit testing. When each branch of the REST API is called, a specific HTTP status is returned. The unit tests check if the HTTP status that was returned matches the expected HTTP status. This allows us to know if our code is functioning as we expect. If a code of 400 or "Bad Request" is returned when a code of 200 or "OK" was expected, we know where to look in the code to fix the problem.

Coverage

The goal is to keep line, statement, and branch coverage at 70% or higher for all back-end code. We use a tool called Istanbul to obtain our coverage reports for the back-end code. Istanbul is a coverage report tool that works well with most JavaScript testing frameworks such as Mocha.

Test Result and Discussion

The result of running all the tests should always have all tests passing in the master branch. The rider functionality was not implemented or fully designed, resulting in the inability to perform their corresponding Black Box tests and have accurate preconditions and steps for performing the corresponding Black Box tests. We have implemented eighteen acceptance tests with CasperJS and forty-five unit tests with Mocha and Chai-HTTP resulting in at least 70% statement, branch, and line coverage according to the Istanbul output in Figure 9. The actual coverage may be higher due to the unused code that is left in the code we tested. The unused

code is code that we have implemented for future features of the EcoPRT system but are currently unused by us. All code for features that we have implemented are tested.

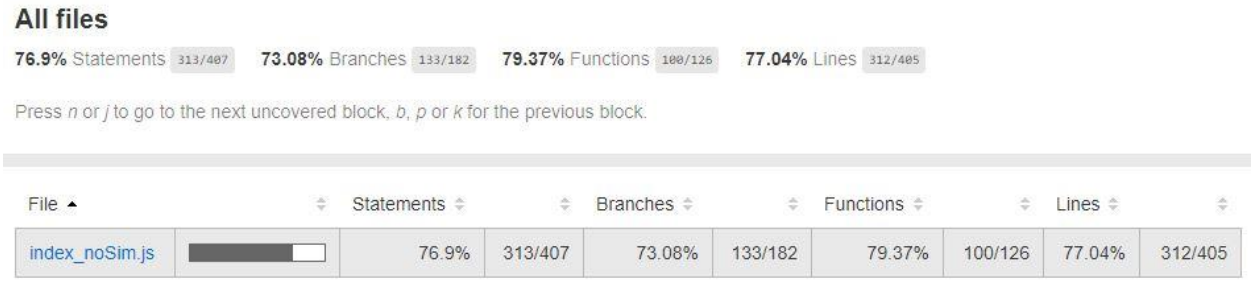

Figure 9 - Coverage from Unit Tests

Suggestion for Future Teams

Hasham Mukhtar, Jye Huang

Documentation

For documentation, discuss with you sponsors before you start developing, where the best place to store it would be. This way it will be in one central place that everyone has access to. It can be Google Drive, Github Wiki, etc as long as everyone agrees. This is one area we struggled with as we never had to do documentation for installing and running software on multiple systems and versions. There are times where we forget where certain documents are because they are in multiple places. We will put our documentation in a good central place for a new team to read and learn about what we've done.

Learning Technologies & Testing

We would recommend learning the technologies we've used as early as possible. A good way to practice Bootstrap would be to make a simple web page and test the different elements it has to offer. Same goes for the server technologies. Make simple applications to test them. You can also practice using the testing technologies on these applications. Using them on simple use cases will help you learn the technologies easier. Also, we would recommend testing as new features or iterations for the project are completed and not wait until the final few weeks of the semester. If you wait to test, you may forget or make poor quality test cases just to get it done.

Rider Functionality

We didn't get to the Rider functionality during the semester, so that is something you will have to do. The Requirements section of this document will tell you what functionality the Rider should have. In terms of UX, you can look at the Admin page and use a good amount of the same features for the Rider page. Coding wise, you can reuse a lot the code from the Admin page to make the webpage for a Rider.

Admin Functionality

There are some Admin features that the sponsors wanted that we didn't get to because of time. They are listed below:

Feature	Description
Vehicle draws edge	The vehicle will draw the path as it moves on the map
Rides Dropdown	Have more useful information in the rides dropdown. Maybe have the vehicle name with its pickup and dropoff nodes
Info Bar	Have more information appear in the info bar at the bottom of the screen. The information shown will be dependent on what is clicked on. Maybe and edge will show the pickup and

	dropoff node and a vehicle will show battery life and heading angle.
Hovering Node Name	When the move hovers over a node, a box appears with the name of that node. Will help Admins when looking for a certain node. That way they don't have to click on every node and look at the info bar.

Other Features

Having a mobile app or mobile version of the web application would be a nice thing to have for the future. We are using Bootstrap which will makes it easier to make a mobile friendly version. Re-designing the web application to fit phones or tablets will be something that the sponsors would want eventually. We thought about doing it for this iteration of the project, but it was out of scope for the semester.

Another feature that would be nice would be some sort of notification system. Maybe have an alert when a vehicle is damaged or if certain edges on the map are blocked. This would help Riders when they are requesting rides. It would also help Admins when they are monitoring the system as they will be directed to certain events. It would also help the Admin and Riders if there were blocked or out of commission edges, that they would be grayed out. This way you can't click on it and it would visually show that it's unavailable.

References

Jye Huang

"MongoDB at Scale." MongoDB, www.mongodb.com/mongodb-scale. Accessed 21 January 2018.

"Performance Best Practices For MongoDB." MongoDB, Nov. 2017, https://webassets.mongodb.com/_com_assets/collateral/MongoDB-Performance-Best-Practices.pdf?_ga=2.224448794.1077654519.1520006540-1904699590.1518192664. Accessed 21 January