Continuous Visualization of CyRide Through an Interactive Map

Team 22: Evan Schlarmann, Endi Odobasic, Braden Buckalew, Andrew McMahon

Advisor: Selim, Mohamed

Client: Soliman, Mohammed

Table of Contents

4 Design

4.1 Design Exploration

4.1.1 Design Decisions

TechStack: React-Django-CORS-SQL:

Using the React-Django-CORS-SQL tech stack is important for this project because it combines a powerful frontend framework (React) with a robust backend framework (Django), creating efficient development and an easy user experience. CORS (Cross-Origin Resource Sharing) improves security by controlling which domains can access resources. SQL databases provide reliability and scalability for managing data, which is essential for effectively handling real-time data transfer. This tech stack offers a comprehensive solution for building a secure, scalable, and efficient web application.

Websockets: Real-time UE Data:

WebSockets allow bi-directional communication between the server (Django backend) and the client (React frontend), enabling real-time data updates without constant polling. Unlike traditional HTTP requests, which require establishing a new connection for each request, WebSockets maintain a persistent connection, reducing overhead and latency. Websockets are well-suited for applications requiring high scalability and concurrent connections, making them ideal for simultaneously handling real-time data from multiple UE devices.

Machine Learning: Traffic Accuracy:

This is for displaying mock vehicle data if the vehicle is outside coverage of the towers, giving the users the illusion of the vehicle being on its correct route. This will be done with the data stored in the database as the bus travels across the intended routes throughout the year. In the initial stages, when first gathering data, the accuracy will not be too great as we will consider the time of year, and for each period, it will take a year to get to the next set of data to "learn" from. Because of this, it may take us a few years until it gets to the accuracy we hope for, but as we continue to gather enough data, the accuracy of our predictive algorithm continues to grow as much as it can.

4.2.2 Ideation

The identification of these technology stacks is based on common industry practices and the specific needs of web application development. Here's how and why each option was identified:

1. **MEAN Stack:**
    a. MongoDB: NoSQL database for storing data in JSON-like documents.

    b. Express.js: Web application framework for Node.js, providing a robust set of features for web applications.
    c. Angular: Frontend framework for building dynamic web applications.
    d. Node.js: JavaScript runtime environment for server-side development.
2. **MERN Stack:**
    a. MongoDB: NoSQL database for storing data in JSON-like documents.
    b. Express.js: Web application framework for Node.js, providing a robust set of features for web applications.
    c. React: Frontend library for building user interfaces.
    d. Node.js: JavaScript runtime environment for server-side development.
3. **LAMP Stack:**
    a. Linux: Operating system.
    b. Apache: Web server software.
    c. MySQL: Relational database management system.
    d. PHP: Server-side scripting language for web development.
4. **RDCS Stack:**
    a. React: Frontend library for building user interfaces.
    b. Django: Web framework for building web applications in Python, providing a clean and pragmatic design.
    c. MySQL: Relational database management system.
    d. CORS: enables secure communication between the React frontend and Django backend, even when served from different domains or ports.
5. **MEVN Stack:**
    a. MongoDB: NoSQL database for storing data in JSON-like documents.
    b. Express.js: Web application framework for Node.js, providing a robust set of features for web applications.
    c. Vue.js: Progressive JavaScript framework for building user interfaces.
    d. Node.js: JavaScript runtime environment for server-side development.

4.2.3 Decision-Making and Trade-Off

Weighted Decision Matrix

Criteria	**MEAN**	**MERN**	**LAMP**	**RDCS**	**MEVN**
Learning Curve/Difficulty	Low	Low	Moderate	Moderate	Low
Familiarity	High	High	Moderate	Moderate	Moderate
Flexibility	Moderate	Moderate	Moderate	Moderate	Low

Suitability	Moderate	Moderate	High	High	Moderate
Scalability	Moderate	Moderate	Moderate	Moderate	Low
Learning Opportunity	Low	Low	Moderate	Moderate	Low
Total	3/5	3/5	4/5	4/5	2/5

The Weighted Decision Matrix reflects the summarized ideation processes we had as a group when discussing the different tech stacks we could have used for our project. Most important to us was our familiarity with the languages/frameworks and their learning curves. We wanted to make sure we were using something manageable so we came to a great compromise that everyone was closely familiar with. Then again, we didn't want it to be too easy, so we decided to pick something we could still learn from and grow as programmers. That being said, the stack that gave us the most value was the RDCS stack.

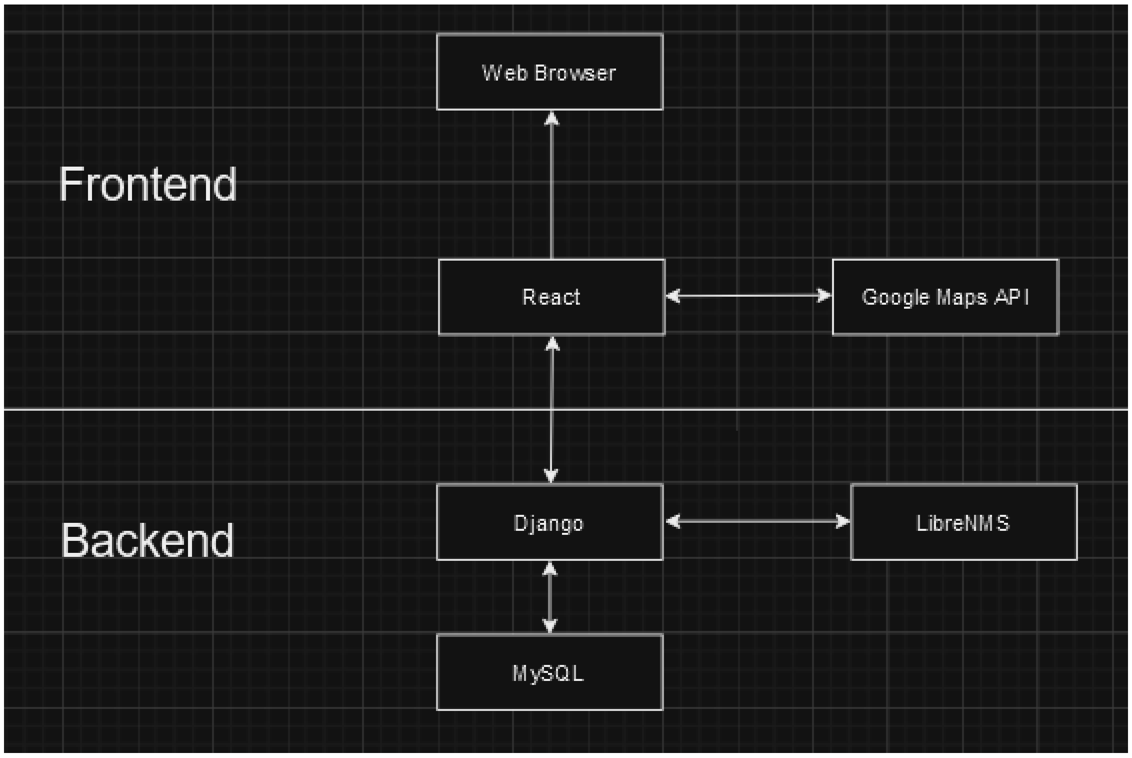
4.2 Proposed Design

4.2.1 Overview

CyRide visualization provides a web service for tracking bus routes and displaying user equipment connectivity using React. This service uses Google Maps API to display an interactive view of Ames, Iowa. Django provides all the data needed to be displayed on Google Maps, which uses multiple methods to provide accurate locations and predictions. LibreNMS and machine learning algorithms retrieve this data. Bus location data is stored in the MySQL database to be used in machine learning to deliver a seamless tracking experience when user equipment is out of range to transmit data.

To track bus locations, UE (User Equipment) is put onto buses and acts as a receiver while signals from data towers are transmitted around campus. This allows the application to gather longitude and latitude pairs to get the exact location of the bus by calling the host service LibreNMS. When the UE is not in range of the signals, it goes into a dead state, which is where a predictive algorithm takes over to estimate where exactly the bus could be at a given time.

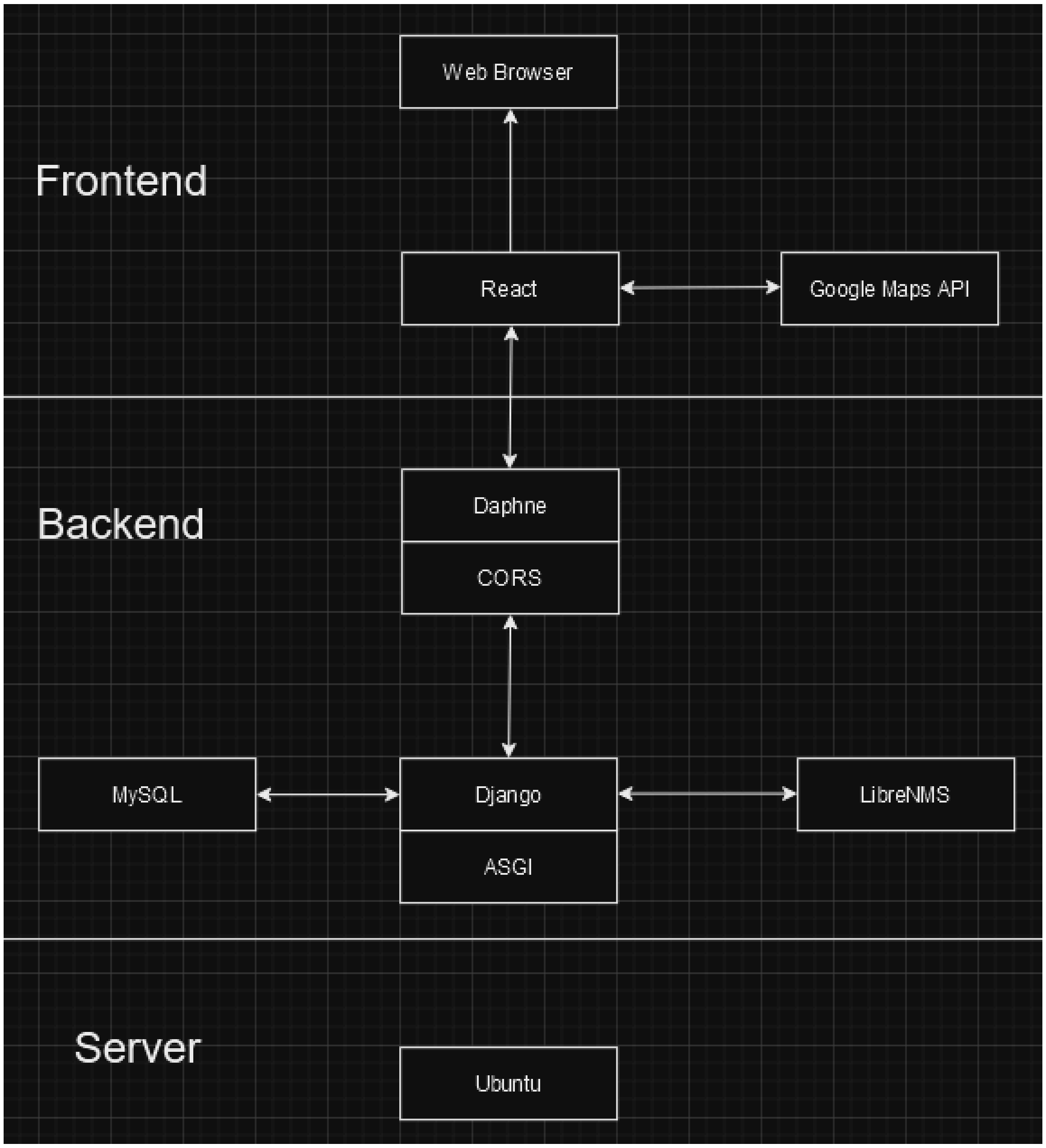
The estimation of bus locations will be done using past longitude and latitude pairs stored within the database. By using this data, with the power of basic machine learning, we can use that past data to approximate exactly where the bus might be at a given time. This allows the application to predict when UE devices will be connected to base stations. It also gives a seamless flow for the application so that buses will still be tracked even if UE connections are lost.



4.2.2 Detailed Design and Visual(s)

All data pertaining to buses is retrieved/calculated by Django. Django utilizes API calls to retrieve updated data from LibreNMS for all UE's and inserts that data into the MySQL database. This relational database has a direct SQL connection with Django, where the data can later be retrieved and used in machine learning algorithms. The database will also store other information about buses that UE's collect including their direction and connectivity.

Django and React send data using Rest API calls and WebSocket connections. Django utilizes Daphne and Channels to create WebSockets connections. Daphne is a server deployment that handles HTTP and WebSocket protocol connections for an ASGI application. ASGI provides asynchronous support for programming Python on the web server. WebSocket connections are created for all necessary live updates on bus locations and their UE data. This ensures the Google Maps interface is constantly updated with the most bus data. Rest API calls are used to retrieve all other bus data that only need to be retrieved once per user request. To ensure that HTTP calls are secure, Django utilizes CORS, which receives incoming requests and checks if the origin can connect with Django.



Below are wireframes for the web browser that users would interact with to get bus location data. They show how the pathing of the bus would be displayed along a given route with insight on connectivity. A user can choose a specific route that a UE is on to receive data pertaining to those buses. The user can also select a specific bus on that route to get information about the bus location and UE connection.

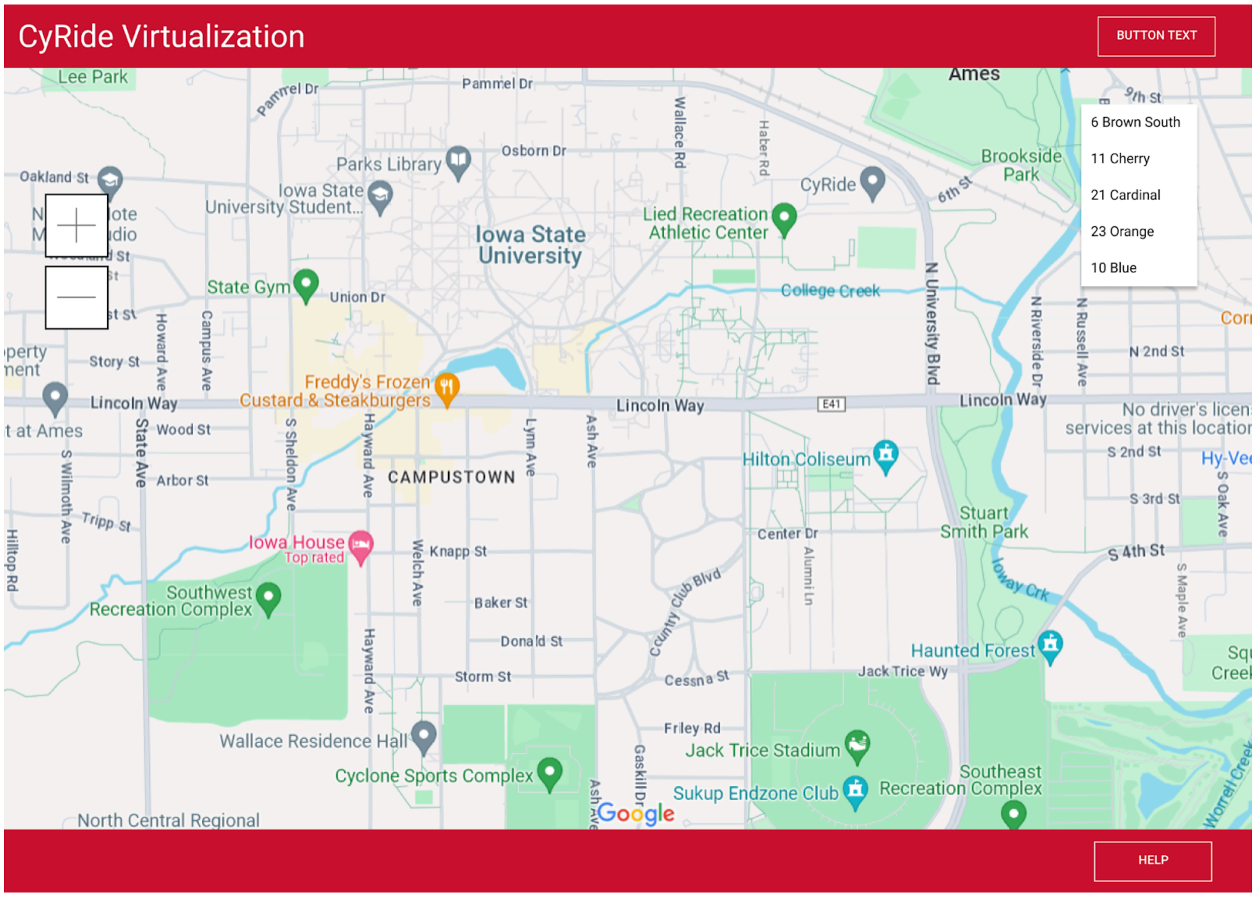


Figure 1. Default View

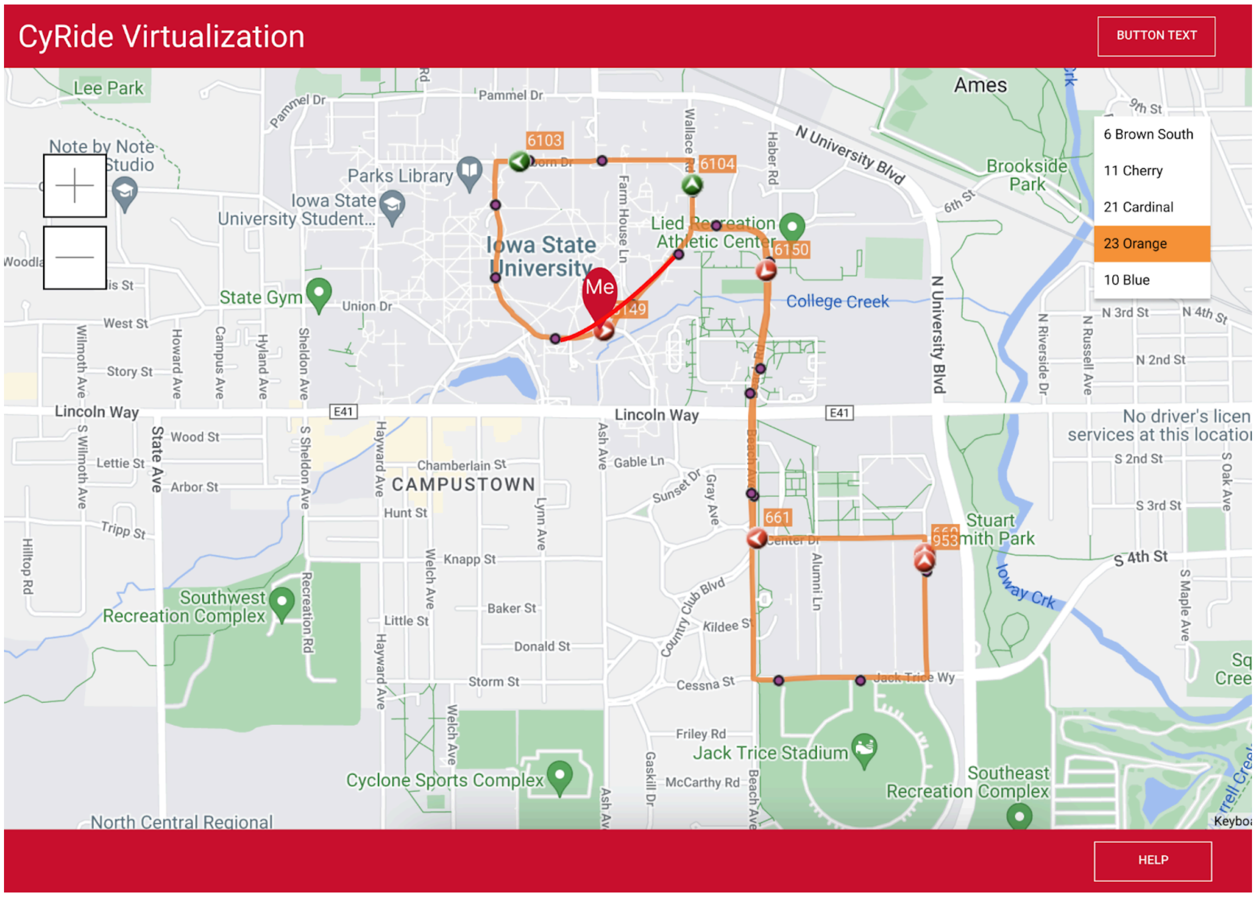
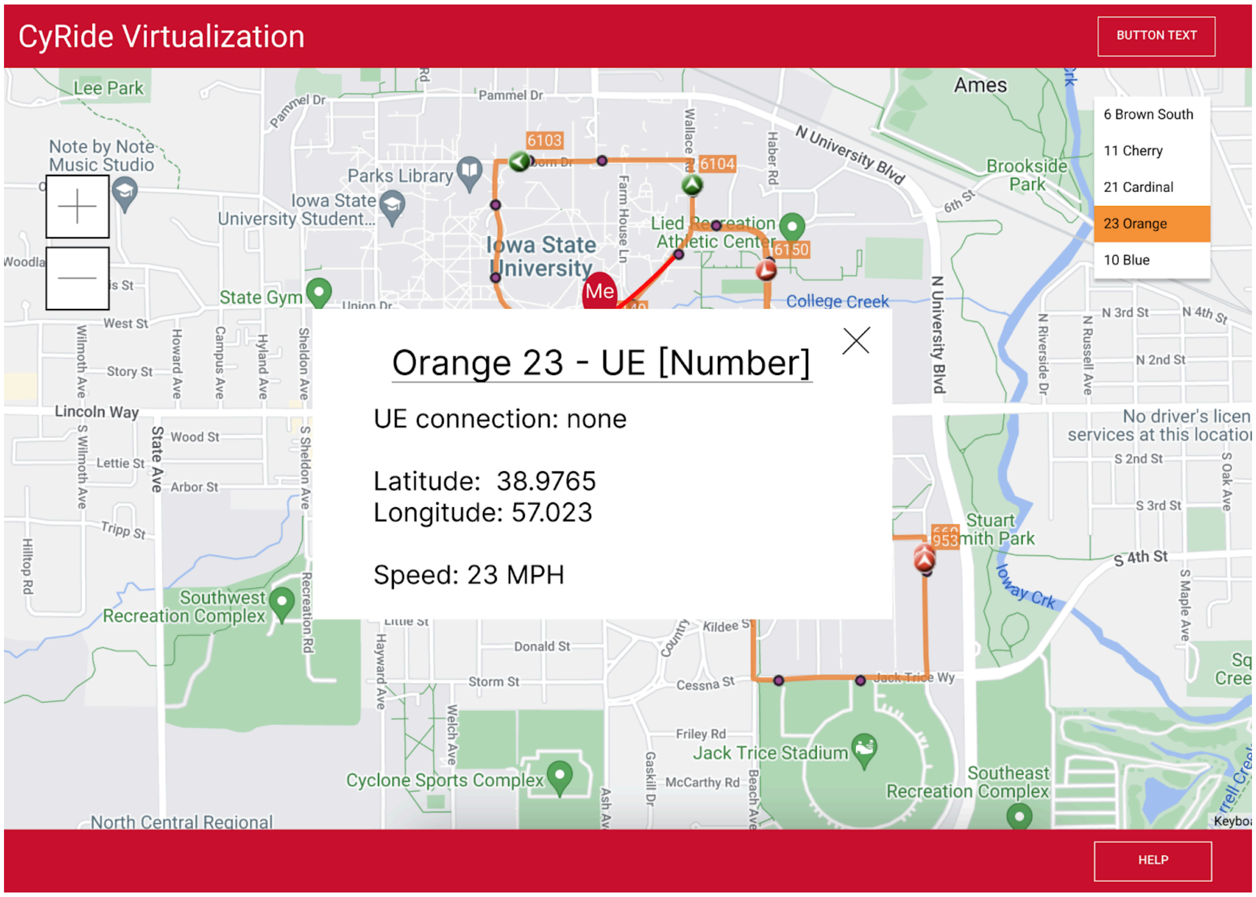


Figure 2. Selected UE Shows Bus in and out of coverage



Orange 23 - UE [Number]

UE connection: none

Latitude: 38.9765
Longitude: 57.023

Speed: 23 MPH

Figure 3. Shows specific metadata of bus

4.2.3 Functionality

When the system is available to users, they will be able to see all mobile UEs that are currently in use. They will be on buses tracking the routes while also giving insight into the connectivity and location prediction. Users can select a route they would like to view, which would isolate only those buses and their data. The user can go even further to select a single bus that would give information about that specific UE and its location. The route of a bus will change its color appearance for areas where the UE has no connection, and machine learning will provide location predictions. This gives users insight into when a UE will be connected again while also tracking the bus. This interaction with the user is displayed in the above Figma designs.

4.2.4 Areas of Concern and Development

The current design provides all the functionalities needed to satisfy design requirements and meet user needs. React provides a robust framework for developing user interfaces that can be translated into mobile development. Django's environment simplifies application development and allows for easier integration of machine learning algorithms. A primary concern with this design is developing an efficient and accurate machine-learning algorithm that provides bus location data. To address this, we adapted Django, which has access to Python's vast library, to simplify development and integration.

4.3 Technology Considerations

TechStack: React-Django-CORS-SQL:

React (Frontend): Powerful Typescript library for building user interfaces. Allows developers to create reusable UI components that update efficiently when data changes. This is important for project success as it enables the development of a responsive and dynamic user interface, which is crucial for a modern web application. The downside of React is that there is a lack of documentation that doesn't cover the details of the framework. Angular is a similar componentizing framework that is harder to learn and understand but has detailed documentation.

Django (Backend): High-level Python web framework that's easy to deploy and scale. This is important for project success as Django provides a robust backend framework that simplifies the development of complex web applications. Python's versatility allows for easy integration with other technologies, such as Machine Learning. A weakness of Django is that it can be slow if the application is not properly optimized. A lack of convention in Python leads us to a configuration boilerplate that could slow down the development process. Ruby on Rails is another option with a better standard convention that is easier to develop.

CORS (Cross-Origin Resource Sharing): Used for same-origin communication between Django and React. Used to avoid Web Scraping and creating secure connections. This is important for project success as it ensures that the frontend and backend can communicate securely and helps prevent security vulnerabilities such as cross-site scripting (XSS) attacks. If CORS is configured incorrectly an attacker can look for known vulnerabilities in the CORS implementation and gain access to restricted resources. CORS is still a better option than another resource-sharing service like JSON-P, which detects errors after the data has been passed.

SQL (Structure Query Language): Django can only connect to Relational Databases, and SQL is familiar to the development team. This is important for project success as it ensures that the backend can efficiently store and retrieve data, and SQL's familiarity with the team will make development easier and more efficient. Another alternative to the project would be PostgreSQL.

4.4 Design Analysis

The current design progress is a fully functional tech stack deployed onto a remote server. This design utilizes communication between React and Django using WebSockets and the Rest API framework. Django also has a SQL connection to MySQL that automatically updates the database schema when object models are built or changed. Lastly, it has a functional map utilizing the Google Maps API, which displays location data. Currently, the deployment shows success for the implementation and will fulfill all requirements for the application. This design simplifies many aspects of development, making it easier to fulfill all user needs.

In the future an external connection will be made with LibreNMS to collect real data from UE's and display the locations on the UI. A machine learning model will be implemented to predicate bus locations even when UEs have no connection. This estimates when UEs will be connected and gives a seamless tracking experience. The bus locations will have to be updated smoothly in the user interface, and provide interfaces to view UE/location data of a given bus. We will also have to provide security updates by providing HTTPS support and utilizing CORS headers.