Virginia Tech

CS4624: Multimedia, Hypertext, and Information Access

Traffic Visualization Dashboard

Group Members:
Gabriel Worsley, Xi Chen, Brett Noneman, Matt Borghese, Xinchen Liao

Client:
Mohamed Farag

Date:
December 13, 2024

Blacksburg, VA

Contents

1 Executive Summary **3**
- 1.1 Project Overview . . . 3
- 1.2 Key Objectives . . . 3
- 1.3 System Overview . . . 3

2 Introduction **4**

3 Requirement **5**

4 Design **6**
- 4.1 Frontend . . . 6
- 4.2 Backend . . . 6
- 4.3 Program Objectives . . . 7
- 4.4 Frontend Functionality . . . 7
- 4.5 Deployment . . . 8

5 Implementation **10**
- 5.1 User Login and Token System . . . 10
- 5.2 Back End Endpoints to Database Server Storage . . . 11
- 5.3 Collection Uploads . . . 11
- 5.4 Collection Selection . . . 11
- 5.5 Visualization Selection . . . 11
- 5.6 Customizable Visualizations . . . 12

6 User Manual **13**
- 6.1 Logging In . . . 13
- 6.2 Uploading and Managing Collections . . . 14
- 6.3 Viewing and Selecting Files . . . 16
- 6.4 Logging Out . . . 17
- 6.5 Use Case: Visualization with Traffic Map and Statistic Graph . . . 17

7 Developer's Manual **18**
- 7.1 General Project Skeleton . . . 18
- 7.2 Inventory of all data files, program files . . . 18
- 7.3 Tutorials on Installing and Running Software . . . 21
  - 7.3.1 Installing the Project Files . . . 21
  - 7.3.2 Installing Node js . . . 21
  - 7.3.3 Installing Docker . . . 22
  - 7.3.4 Dockerize MySQL Server . . . 22
  - 7.3.5 Running the Project . . . 23
  - 7.3.6 Using Screen . . . 24
- 7.4 Server.js Explained . . . 25
  - 7.4.1 Login Requests: . . . 25
  - 7.4.2 File operations: . . . 25
  - 7.4.3 Simulation requests: . . . 26
- 7.5 Database design . . . 39

8 Lessons Learned **41**
8.1 Timeline 41
8.2 Problems 41
8.3 Solutions 41

9 Future Work **43**
9.1 General Future Work 43
9.2 Future Work: Visualizations Per File Type 45
9.2.1 Summary File 45
9.2.2 Simulation Details 45
9.2.3 Average Traffic Conditions 45
9.2.4 Minimum Path Trees 46
9.2.5 Trip Completion Probes 46
9.2.6 Road Probes 46

10 Acknowledgments **47**

11 References **48**

1 Executive Summary

1.1 Project Overview

The Traffic Visualization Dashboard project is an initiative by the Virginia Tech Transportation Institute (VTTI) to enhance the usability of the Integration Traffic Simulation Tool. This powerful tool simulates traffic flow, but the complexity of its raw data can make it difficult for users to interpret and derive valuable insights. The primary aim of this project is to transform this raw data into intuitive and meaningful visualizations through a comprehensive dashboard. The dashboard's key functionalities include user authentication and secure access, data collection upload, and dynamic visualization of traffic environments.

1.2 Key Objectives

User Authentication and Data Security: Implementing a secure login system to control access and ensure data integrity. Users will log in to access their personalized dashboard environments.

Data Collection Upload and Storage: Enabling users to upload simulation data collections directly to a centralized database, thus streamlining the data management process.

Interactive and Customizable Visualizations: Allowing users to choose from various visualization options to explore traffic metrics in detail. This includes features like heat maps, line charts, and node graphs, enhancing the user's ability to analyze specific metrics and trends.

Scalable Visualization Options: Designing the dashboard to support the addition of new visualizations as needed, ensuring future scalability and adaptability to evolving user requirements.

1.3 System Overview

Frontend: Developed using React for a responsive, interactive user experience. Visualization libraries such as AntV and Chart.js are utilized to create diverse and detailed visualizations.

Backend: Built using Node.js and Express.js, with MySQL for data storage. The backend will handle data parsing, token creation, user authentication, and retrieval of user-specific data.

Deployment: The application will be containerized using Docker to ensure consistent deployment across different environments.

User Manual The user manual provides step-by-step instructions on logging in, uploading and managing data collections, viewing and selecting visualizations, and logging out. It ensures users can efficiently utilize all features of the dashboard.

Developer Manual The developer manual provides further project details such file structure and file details. These details give the purpose and role of each javascript file. It gives a tutorial on the installation of the workable project

2 Introduction

The Virginia Tech Transportation Institute has developed a powerful tool for simulating traffic flow, known as the Integration Traffic Simulation Tool. This project aims to enhance the usability of the simulator by creating a comprehensive visualization dashboard that transforms complex raw data into intuitive, meaningful visualizations, enabling end users to gain valuable insights into traffic behavior and patterns.

Current challenges with the simulator include difficulties in interpreting raw data and a lack of accessible visual feedback, making it harder for users to make informed observations. The proposed dashboard will address these issues by providing a suite of visualizations to bring the simulator's output to life. Key features include a user login system for secure access, a data collection upload mechanism to populate the database, and a base visualization environment that loads automatically upon selecting a data collection.

This interactive dashboard will offer users multiple visualization options tailored to highlight various traffic metrics and trends. As a scalable solution, the dashboard can be expanded with additional visualizations to meet user requirements and evolve as new insights or needs emerge.

3 Requirement

The proposed dashboard will focus on the following core objectives:

User Authentication and Data Security: Implement a secure user login system to control access and maintain data integrity. Authorized users can securely log in to access their specific datasets and work within a customized, personalized dashboard environment.

Data Collection Upload and Storage: Users will be able to upload simulation data collections directly to a centralized database, streamlined the data management process. This feature will support seamless integration with the output of the Integration Traffic Simulation Tool, enabling consistent data storage, retrieval, and future analyses.

Interactive and Customization Visualizations: Users will have the ability to select from a variety of interactive charts and visualizations that display traffic metrics in more granular detail. For example, heat maps could represent congestion levels, line charts could show changes in traffic flow over time, and bar charts might display vehicle counts across different zones. Each visualization will be customized to help users focus on specific metrics, time frames, or locations within the simulated traffic environment.

Flexible, Scalable Visualization Options: Designed with future scalability in mind, the dashboard will support the addition of further visualizations as new data analysis needs arise. This flexibility ensures that the dashboard can continue to evolve, accommodating advanced visualization features or new metrics as they become relevant.

The Traffic Simulation Visualization Dashboard will transform complex traffic data into accessible insights, empowering end-users to interpret and analyze traffic patterns effectively. Through this dashboard, VTTI's Integration Traffic Simulation Tool will become more accessible, user-friendly, and impaction for diverse applications, from traffic research and policy-making to infrastructure planning and safety analysis.

4 Design

4.1 Frontend

The frontend of the Traffic Simulation Visualization Dashboard was implemented using React as the core framework. React was chosen for its component-based architecture, which allows for modular, reusable UI components that improve code maintainability and scalability. The use of React also enables a responsive and interactive user experience, essential for a data-driven dashboard with various visualizations.

To bring the simulation data to life, we will leverage powerful data visualization libraries.

1. AntV: AntV is a visualization library that offers a collection of ready-to-use and aesthetically pleasing charts. It complements D3.js by providing simpler and configurable visualizations that are useful for quickly building standard charts, such as line graphs, bar graphs, and pie graphs, enhancing the overall user experience.

2. Charts.js: For lightweight and straightforward charting needs, Chart.js provides an easy-to-use solution. It will be used for visualizations that require minimal configuration, such as displaying time-series data on traffic volume or average speeds. Chatr.js is particularly suitable for mobile-friendly charts due to its responsive design features.

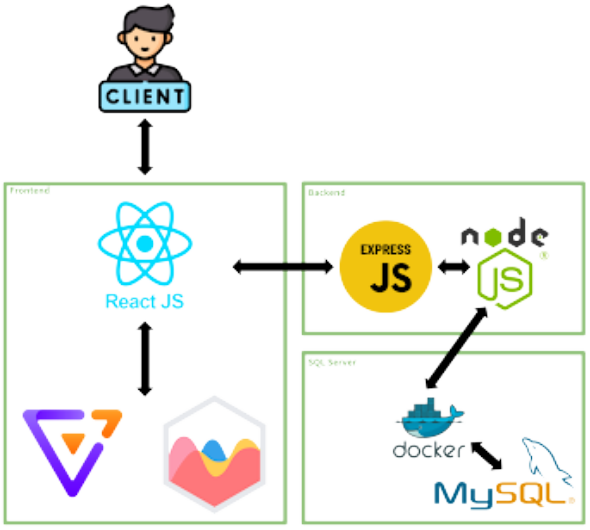

Figure 1: The Technology Stack

In summary, the front-end implementation combines React framework with visualization libraries, allowing us to create an interactive, scalable, and visually rich dashboard for end-users to explore traffic simulation data effectively. Each library will serve a unique purpose, from complex data handling to user-friendly charting that ensures a comprehensive and adaptable visualization solution.

4.2 Backend

The backend is developed using Node.js and Express.js, forming a robust server-side framework. Data related to files is stored securely in a MySQL server, which is contained

in a Docker container. The backend's responsibilities include parsing collection files to prepare them for frontend visualization, creating and verifying tokens, and retrieving user information, including collection data.

4.3 Program Objectives

The program aims to achieve the following primary objectives:

- User Login and Token System
- Back End Endpoints to Database Server Storage
- Collection Uploads
- Collection Selection
- Parsing of Uploaded Files
- Visualization Selection
- Customizable Visualizations

4.4 Frontend Functionality

The frontend provides users with the ability to log in to an existing account or create a new account. Logging in will pull the user through a protected route and loading screen to the main page. In the main page, users can upload a ZIP file as a collection (Naming it appropriately), which is associated with the logged-in user. The uploaded zip will be sent to the backend for parsing and the information relevant will be sent back to the frontend, which will be displayed as an uploaded collection. Upon selection of a collection, files within the collection that have been parsed will fill in below it as shown in figure 2. Noteably these features have error handling, like if you enter an invalid login, or enter a collection name that already exists.

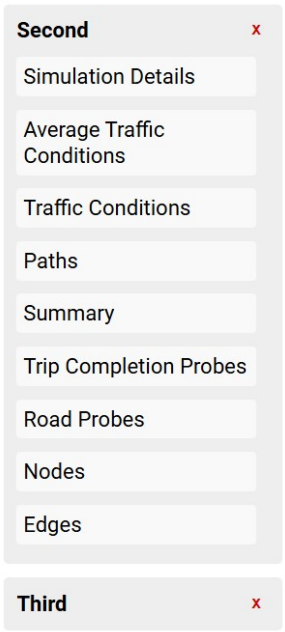

Figure 2: The Collection Selection Interface

The users can then select additional visualizations from a menu on the right half of the screen. The backend provides these visualization options based on the selected collection. When a new visualization is selected, it replaces the current one on the screen, with what is selected, generated by the files in the collection.

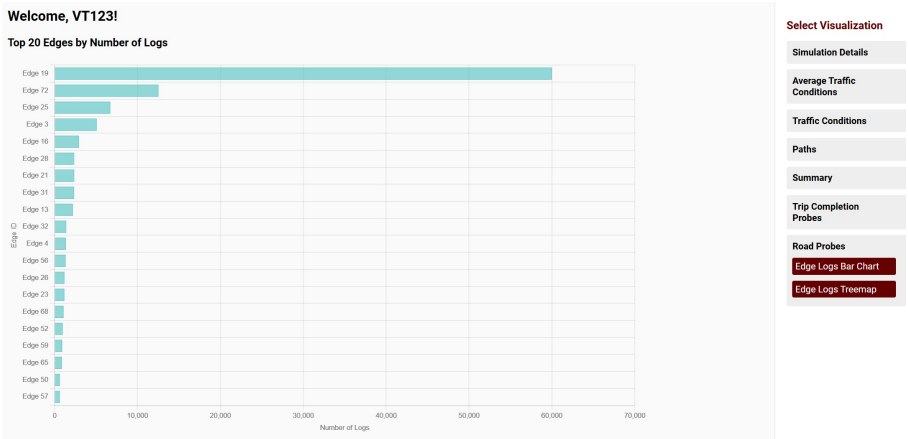

Figure 3: Visualization Example

4.5 Deployment

Deploying the application is an involved process on getting Node.js and Docker installed on whatever platform it is to be installed on. The frontend and backend run on Node.js at a core level and must be installed first. Then Docker must be installed. In the developer log, more detail on the product installation is given.

After these two prerequisites are met, it is a simple docker image creation, then start of a container for the MySQL Server. Next, run npm install in both backend and frontend folders. Then alter the .env file in both folders to match the conditions of the environment.

Lastly, run a node server.js command in the backend folder and an npm start command in the frontend folder.

5 Implementation

5.1 User Login and Token System

The implementation of the user authentication and login functionality involves:

- Setting up the user database schema in MySQL.

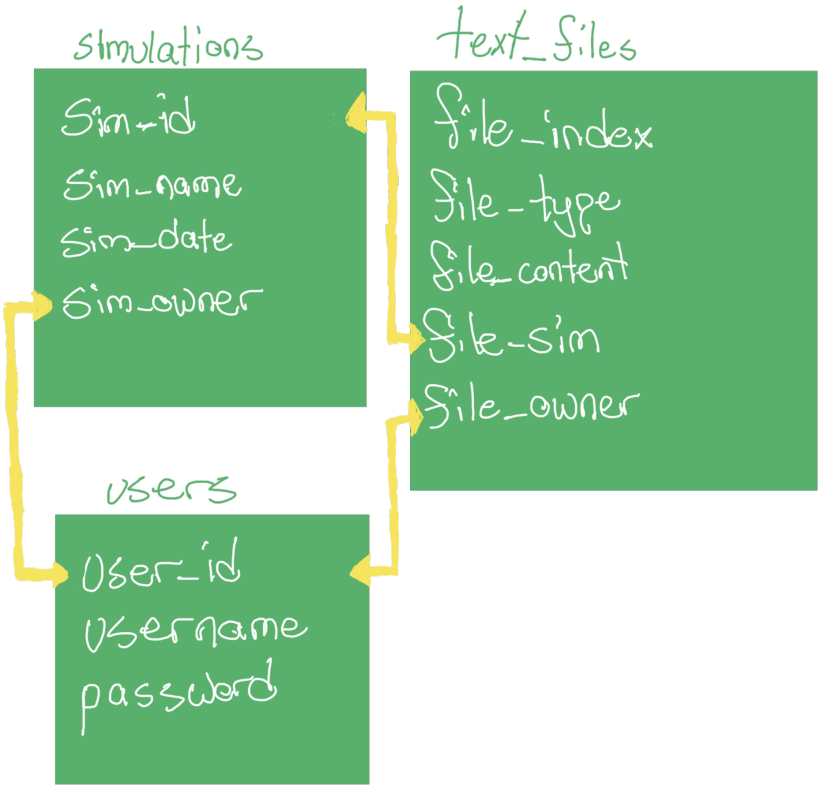

Figure 4: The Schema

- Using backend to handle user token registration, login verification, and token authentication.
- Implementing JWT (JSON Web Tokens) for secure token-based authentication.
- Implementing a login/sign up interface
- Error handling for bad inputs or invalid logins
- Creating routes and a ProtectedRoute to the main page
- Implementing a logout button on the main page.

5.2 Back End Endpoints to Database Server Storage

- Creating Backend Endpoints by utilizing express route commands
- Create routes for logging in, signing up, getting file x for user x, etc
- Create a connection pool for the MySQL server
- Implementing functions for handling all Server interactions and their corresponding error handling

5.3 Collection Uploads

To enable users to upload collections to the database:

- Designing the frontend component in React to handle collection uploads and names.
- Creating API endpoints in the backend to accept and store the uploaded files in the MySQL database.
- Parsing file data in the backend to relevant datatypes and json format.
- Implementing error handling and validation for file uploads.
- Creating API endpoint to enable getting collections associated with a user.
- Creating API endpoint to get file types sent to be stored in the collection.
- Implementing a frontend interface to see and select a collection, which dropdown to display the file types associated.

5.4 Collection Selection

- Create API endpoint to get all collections owned by a user
- Creating a user interface in React to see and select a collection
- Create an endpoint to get all parsed filenames in a collection

5.5 Visualization Selection

To allow users to select and view additional visualizations:

- Implementing a menu in React to list available visualizations.
- Creating backend logic to provide visualization options based on the selected collection.
- Ensuring seamless switching between visualizations on the frontend.
- Backend API to get specific parsed file information per user, per collection.

5.6 Customizable Visualizations

This section varied more in each designer's approach to creating their corresponding visualizations. There were common themes in thought and design.

- Each graph must have some kind of potential user input for customization.
- Information shown must be easily understand to those unfamiliar with the system.
- For time's sake, information from a graph must be tied to one output file and any number of input files.

6 User Manual

To begin using the **Traffic Visualizer**, follow these steps:

6.1 Logging In

- Navigate to our website. (http://tml.cs.vt.edu:4624)

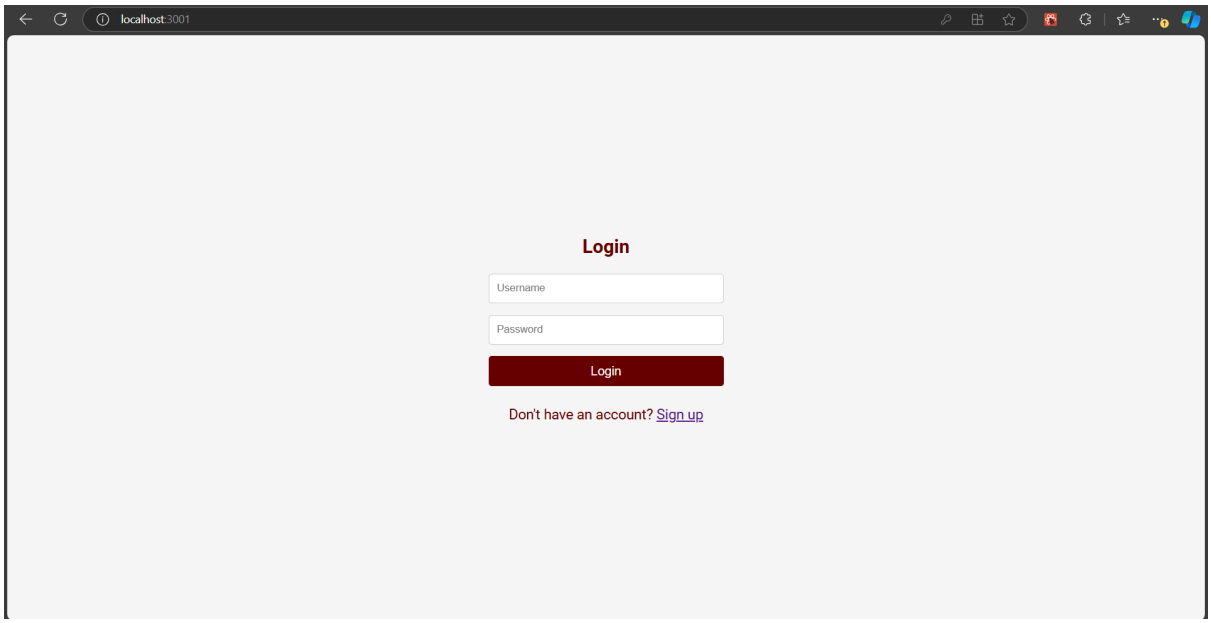

Figure 5: The Login Page

- If you have an account, log in. The public login is Username: 'root' Password: 'rootpass'

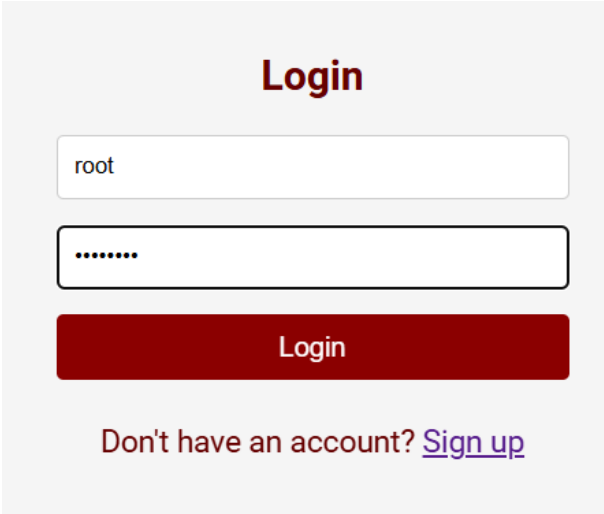

Figure 6: Entered root Login

- If not, select "Sign Up" to create an account, then log in.

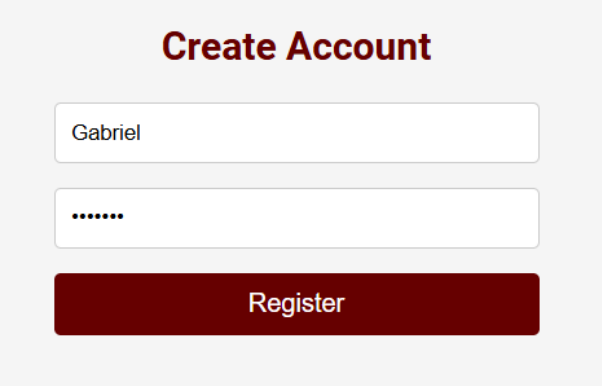

Figure 7: Sign Up Page

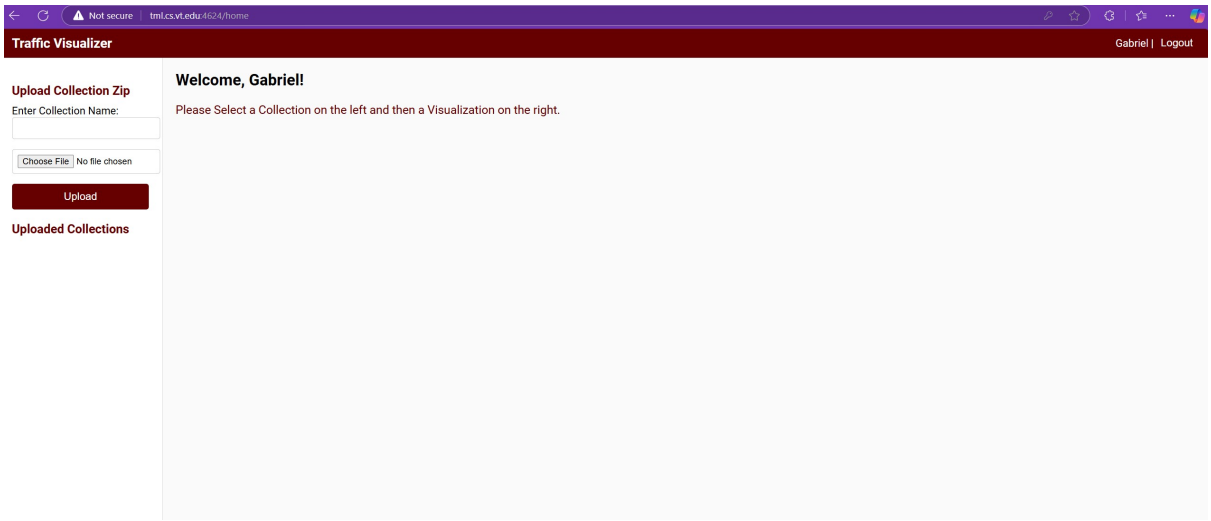

Figure 8: Default Empty Main Page

6.2 Uploading and Managing Collections

- After logging in, you can either:
  - Upload a new collection by entering a collection name that doesn't already exist and isn't empty, and attach the collection ZIP file. Finally, hit the upload button.

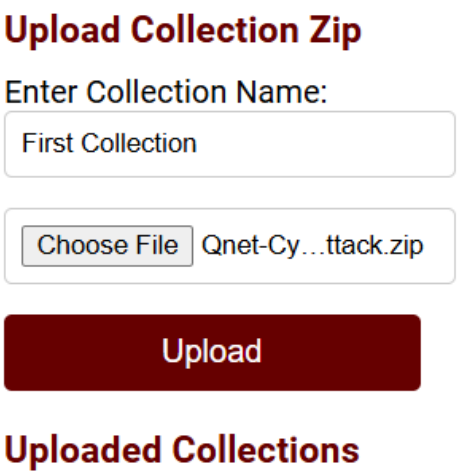

Figure 9: Collection Upload Interface

– On hitting upload, a loading animation will occur until the collection is successfully uploaded. The collection should appear below the upload interface as shown in figure 10.

– When a collection is open/clicked, not only are the file types within show, but a node graph visualization of the traffic simulation will show in the center of the screen and other visualization options will appear in a menu on the right side of the screen.

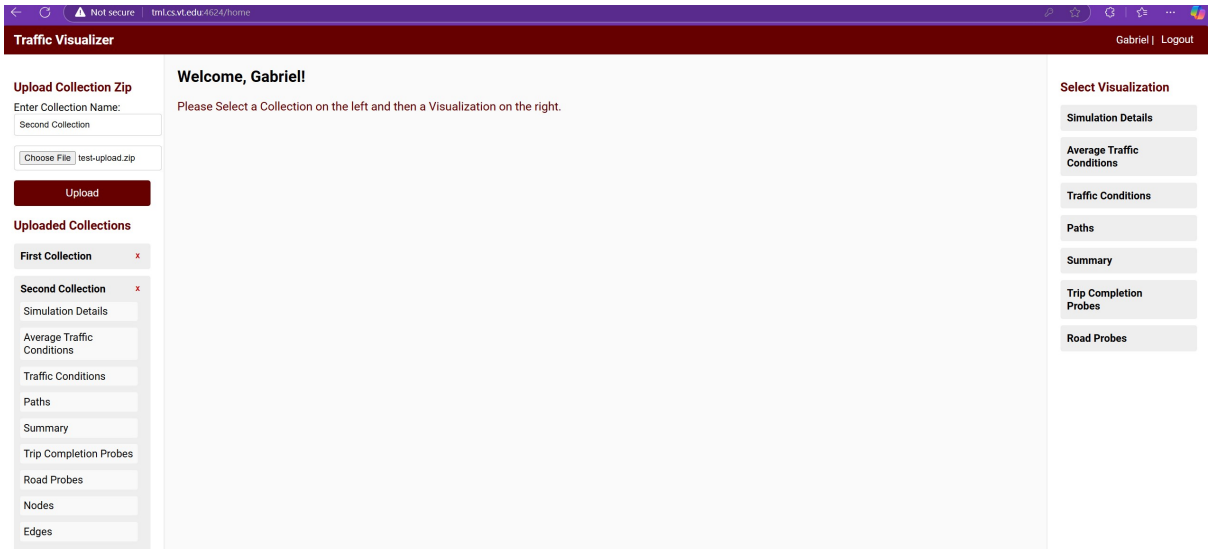

Figure 10: First and Second Collections Uploaded

6.3 Viewing and Selecting Files

- Within each collection, there are multiple file types as determined by what was uploaded.

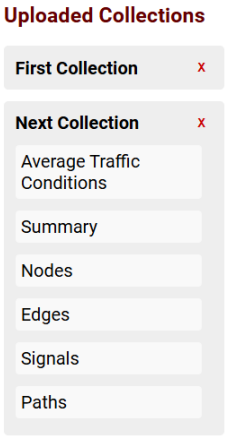

Figure 11: Collection Selection Interface

- Select a collection to display its visualization options and file types. The above shows files 11 (Average Traffic Conditions), Summary, inputs 1 and 2(Nodes and Edges), and more were uploaded and ready for visualization use.
- You can delete collections from the "Uploaded Collections" section by selecting the red 'X' next to the collection name.
- Select another visualization option, on the right hand menu, to switch the shown graph on screen. In figure 12, a Traffic Map was chosen from the first collection.

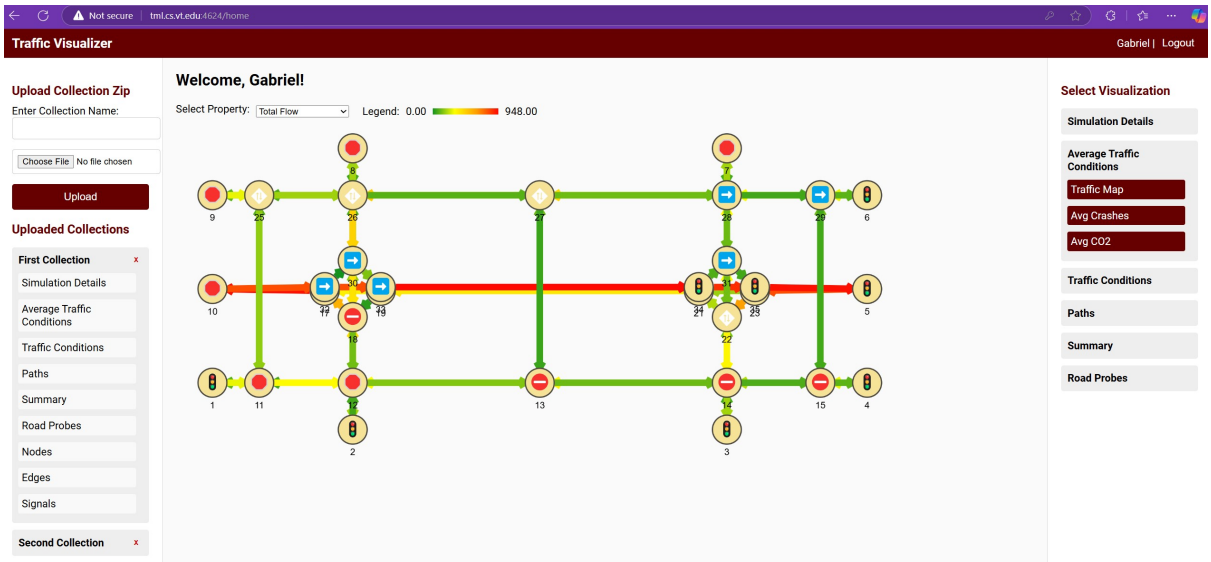

Figure 12: File 11 Traffic Map

- The node graph will be displayed by default.

- Use the selection interface on the right side of the screen to switch to different visualizations as desired.

6.4 Logging Out

- Once finished, log out and close the webpage.

Figure 13: Logout Button

6.5 Use Case: Visualization with Traffic Map and Statistic Graph

Preconditions:

- The user is logged in and on the main visualization page.
- The user has a collection of files ready to upload to the collection or has previously uploaded collections, which contain the first two input files and file 11.

Basic Flow:

1. The user accesses the main page and views a left sidebar with options to upload files and view a history collection of previously uploaded files.
2. The user uploads a new collection, ZIP file, which is added to the history section.
3. The user selects a collection from the history collection, prompting the system to display a visualization option on the right-hand menu.
4. The user has the option to choose a visualization type from available graph options.
5. After selecting the Average Traffic Conditions type(File 11), the chosen visualization appears as the Traffic Map, displaying traffic data based on the selected file and input files 1 and 2 (Nodes and Edges).
    - *Alternative Flow 1*: If a specific chart type is unavailable for the selected data, the option will not appear for the user to select.
6. The user can interact with the visualization, such as zooming and adjusting data filters.

7 Developer's Manual

7.1 General Project Skeleton

This website is split into 3 servers, one for handling the webpages, and another for handling backend processes like parsing and tokening, last is the MySQL server storing the data. The server for the webpages uses "React" and is located in the ./frontEnd directory. This is where anything related to the visuals can be found. The server for the backend functionality is located in ./backEnd. This part is made with "express" and is where things relating to functionality (like login and files uploads) are handled.

7.2 Inventory of all data files, program files

1. ./**backEnd** This directory stores all of the script files related to the non-visual/non-webpage components of the website. These files are used by a nodejs server.

2. ./backEnd/**server.js** This script stores the callback functions for non-webpage HTTP request and is the one backend responsible file. Server.js has a lot to unpack so it will be further discussed in section 7.4. This is the only unique/modified script in ./backEnd.

3. ./**frontEnd** This directory stores the scripts related to the server that distributes the webpages.

4. ./frontEnd/**src** This directory stores the files specific to this project. In this directory, there should be the files: "App.js", "index.css", "index.js", "nodeGraph.js", "reportWebVitals.js", "setupTests.js".

5. ./frontEnd/src/**App.js** This file defines the main "app" and the directories of the webpages though it.

6. ./frontEnd/src/**index.js** This file is the start of the webpage scripts, and is just used to call "./frontEnd/src/App.js".

7. ./frontEnd/src/**reportWebVitals.js** This script contains a single function used for getting information/statistics about the webpage server.

8. ./frontEnd/src/**components/** This directory contains the actual webpages for the website. Each script contains a corresponding ".css", except for "ProtectedRoute.js".

9. ./frontEnd/src/components/**AveTrafficConds.js** This script defines file 11 Average Traffic Condition visualizations, graphs, and all.

10. ./frontEnd/src/components/**CustomSummaryChart.js** This script defines the Custom Summary Chart visualization as part of the Summary file visualization.

11. ./frontEnd/src/components/**Charts.js** This script contains logic for choosing which chart component will be displayed based on state variables controlled in RightSidebar.js.

    Follow patterns seens in these two files to be able to construct a new visualization. It involves getting the parsed file type name from the backend, adding the name in

Charts.js as a file type case, in the case add the component you want to control the visualization of that file type. Or you you could set the case as seen in Road Probes, which first takes the state of the selected visualization(selectedGraph) in the file type and passes the component accordingly. Lastly, go to right side bar to add the file type to visualizations, with their individual visualizations in it's corresponding array. Also add the file type to the outputFileTypes array.

Individual visualization components are implemented differently so no solid theme is established other than the state variable selectedGraph to determine when to render the visualization. expandedCollection represents the file type selected.

```
case "Simulation Details":
        ret = (
            <Signals
                dimensions={dimensions}
                selectedGraph={props.selectedGraph}
                expandedCollection={props.expandedCollection}
            />
        );
        break;
case 'Road Probes':
    if (props.selectedGraph === 'Edge Logs Bar Chart') {
        ret = (
        <EdgeLogsBarChart
            dimensions={dimensions}
            selectedGraph={props.selectedGraph}
            expandedCollection={props.expandedCollection}
        />
        );
    } else {
        ret = (
        <EdgeLogsTreemap
            dimensions={dimensions}
            selectedGraph={props.selectedGraph}
            expandedCollection={props.expandedCollection}
        />
        );
    }
    break;
```

Figure 14: Charts.js screenshot

```
const visualizations = ({
    'Road Probes': ['Edge Logs Bar Chart', 'Edge Logs Treemap'],
    'Summary': ['Traffic Map', 'Custom Summary Chart'],
    'Average Traffic Conditions': ['Traffic Map','Avg Crashes', 'Avg CO2'],
    'Traffic Conditions': ['Traffic in Series'],
    'Paths' : ['Minimum Path Trees'],
    'Simulation Details': ['Time Optimizations'],
    'Trip Completion Probes' : ["Car Information Filter"],
});
const outputFileTypes = ['Summary', 'Average Traffic Conditions', 'Traffic Conditions', 'Paths',
```

Figure 15: RightSidebar.js screenshot

12. ./frontEnd/src/components/**EdgeLogsBarChart.js** This script defines the Edge Logs Bar chart as part of file 16 visualizations.

13. ./frontEnd/src/components/**EdgeLogsTreemap.js** This script defines the Edge Logs Tree map as part of file 16 visualizations.

14. ./frontEnd/src/components/**LoadingSpinner.js** The loading spinner component used in the loading screen and upload animations.

15. ./frontEnd/src/components/**Login.js** The main login page for the website. This is the first page the client is redirected to.

16. ./frontEnd/src/components/**Main.js** The main page component for the website after having been logged in. This webpage makes use of "Navbar.js", "Charts.js". "RightSidebar.js", and "Sidebar.js". This is the central hub for logic flow and thus contains most state definitions relevant to the entire page. It also contains handlers for token verification/storage as well as makes most calls to the backend for collection updates.

17. ./frontEnd/src/components/**MinPathTree.js** This script defines file 13 visualization or Minimum Path Trees visualizations.

18. ./frontEnd/src/components/**Navbar.js** The top bar of the main page/menu. Contains the logout button.

19. ./frontEnd/src/components/**Popup.js** This component is used by other visualization components to enable an on screen popup.

20. ./frontEnd/src/components/**ProtectedRoute.js** A sort of "barrier" page that is used as a mid-point to handle when the loading spinner thing should be used, or when the client should be rediected back to the login page. Also contains the loading screen between home and login screens.

21. ./frontEnd/src/components/**Register.js** The page for creating a new account.

22. ./frontEnd/src/components/**RightSideBar.js** This component is the right hand visualization selection interface. It is here where, potential visualization must be declared with name of the file type (which must match the backend name of the file) and self chosen visualization names that must be consistent with how they are named in their respective file visualization component and Charts.js

23. ./frontEnd/src/components/**Sidebar.js** The left sidebar for the main page of the website. This contains the form for uploading new simulations/collections along with a log of all of the currently stored simulations for the currently signed in account.

24. ./frontEnd/src/components/**Signals.js** This script defines file 10 visualizations or Simulation Details. At the time there is only an unfinished visualization anmed Time Optimizations.

25. ./frontEnd/src/components/**Summary.js** This script defines the Traffic Map visualization of the Summary visualizations.

26. ./frontEnd/src/components/**TrafficConds.js** This script defines file 12 Traffic Conditions Visualizations.

27. ./frontEnd/src/components/**TripProbe.js** This script defines the file 15 Trip Completion Probes Visualization.

28. ./**Dockerfile** The file is used to dockerize the MySQL server. It contains steps to grab the correct MySQL server version, set the root user default password, expose port 3306 (port used for us to hook it to 4626), and use sim_tables.sql as a default schema by running its contents on initializing the server.

29. ./**sim_tables.sql** This sql script is used for initializing the schema and root login/permissions of the MySQL server.

The rest is generic React/Node.js files and package information.

7.3 Tutorials on Installing and Running Software

Installing the project will change depending on your OS. This tutorial was crafted for the express purpose of CentOS 7 on Linux. Some steps changed based on your OS like for example cloning the repository project files, may differ since you have a user interface and could use something such as Github Desktop.

7.3.1 Installing the Project Files

1. Have a collaborator git account. If not, email yelsrow@vt.edu to be invited as a collaborator.
2. To clone the repository:

   ```
   git clone https://github.com/Yelsrow/Traffic_Visualization.git
   ```

3. Enter collaborator username and PAT (Personal Access Token) for cloning access
4. Generate PAT on Github if you have not already and verify it has repo permissions for cloning.

7.3.2 Installing Node js

Once again changes per system, this is for CentOS 7. On a GUI just go to the official Node.js website and follow installation instructions.

1. Get the repository :

   ```
   curl -sL https://rpm.nodesource.com/setup_16.x | sudo bash
   ```

2. Install Node with the command given in the output of the curl, Step one prints a command to finish installation.
3. Verify installation with :

   ```
   npm --version
   ```

Note version 16 was chosen since a newer version of Node on CentOS 7 requires a more updated version glibc than what was available on the tml.cs.vt.edu server. Naturally breaking that would ruin the whole server, thus to not mess with that an older version on Node.js was chosen despite a deprecation warning.

7.3.3 Installing Docker

Already installed on CentOS 7 Server - check with :

```
docker --version
```

Since this was already installed completion of these steps couldn't be verified, but once again with GUI installation is a simple process of following on screen instructions from the Docker website.

1. Install docker repository :

   ```
   sudo yum-config-manager --add-repo https://download.docker.com/linux
      /centos/docker-ce.repo
   ```

2. Install community docker :

   ```
   sudo yum install -y docker-ce docker-ce-cli containerd.io
   ```

7.3.4 Dockerize MySQL Server

In the deployed version of the project it is recommended to use docker for MySQL Server. However, if this project is to be in a development environment, it may prove best to install MySQL Server and Dashboard. Just update the .env variables in /backend for the correct ports and information. Note - I had to add group 7 permission to dockerize on the tml server with :

```
sudo usermod -aG docker grp7_f24
```

1. Create Docker Image

   (a) Navigate to **Traffic_Visualization** folder and Dockerize a mysql-server:

   ```
   cd Traffic_Visualization/
   docker build -t traffic_visualization/mysqlserver:1.0 .
   ```

2. Run the MySQL container :

   ```
   docker run -d --name traffic_visual-mysqlserver
      traffic_visualization/mysqlserver:1.0
   ```

7.3.5 Running the Project

1. Update the .env in both front/back ends and install dependencies

   (a) Navigate to the frontend by running:

   ```
   cd Traffic_Visualization/frontEnd
   ```

   (b) Install frontend dependencies with :

   ```
   npm install
   ```

   (c) Swap all localhost for tml.cs.vt.edu :

   ```
   Start with a : nano .env :

   PORT=4624
   REACT_APP_API_URL=http://tml.cs.vt.edu
   REACT_APP_API_BACKEND_PORT=4625
   GENERATE_SOURCEMAP=false
   HOST=tml.cs.vt.edu
   ```

   (d) Navigate to the backend:

   ```
   cd ../backEnd
   ```

   (e) Install backend dependencies :

   ```
   npm install
   ```

   (f) Swap DATABASE_PORT=4626 (Developing on MySQL Server the default is port 3306) and swap localhost for tml.cs.vt.edu :

   ```
   Start with : nano .env :

   SECRET_KEY=***
   PORT=4625
   DATABASE_PORT=4626
   SQL_HOST='tml.cs.vt.edu'
   ```

2. Run both frontEnd and backEnd - Run either on same terminal (using & in the command) or different ones, or use Screen as its use is given 7.3.6

   (a) In Traffic_Visualization/frontEnd run :

   ```
   npm start
   ```

   Optionally you can run this next command, however it's not tested, but its advertised to be better since its not for development:

```
npm run build
```

(b) In Traffic_Visualization/backEnd run :

```
node server.js
```

7.3.6 Using Screen

Using 'screen' is similar to tmux and allows persistent sessions. On the tml server the Traffic Visualizer is running on a screen session named traffic_visual-session.

1. Install screen (already installed on server, so potentially incomplete commands):

   ```
   sudo yum install -y screen
   ```

2. Start a screen session:

   ```
   screen -S my-session
   ```

3. Run the application
4. To detach from the session: Press Ctrl+A, then press D.
5. To reattach to the session later:

   (a) Run the command:

   ```
   screen -r my-session
   ```

   (b) For the Traffic Visualizer run :

   ```
   screen -r traffic_visual-session
   ```

7.4 Server.js Explained

The **./backEnd/server.js** file contains the nodejs script for the "back end" server. This server handles request related to the non-visual portion of the website. This includes the login system, file uploads, and uploaded simulation details. All request are either POST or GET and are handled through hooks. A MySQL database is used for storing all information.

7.4.1 Login Requests:

- **GET: /api/verify-token** Checks the login JWT sent through the "authorization" header. Returns status 200 on succuss, 403 if no token/cookie was provided, or 401 if the token is invalid or expired. Other requests that require a login can return these same 3 error responses if they fail due to the same conditions.

- **POST: /login** Takes a JSON file with 2 strings named "username" and "password". "username" is the username attempting to login with, and "password" is the password attempting to login with corresponding to the provided username. On success, this will return a JSON file with 1 value named "token". This value is the login JWT to be sent by the "authorization" header for each request that requires. If the username or password is an invalid string, then a response of 403 is returned. If no matching username, password pair is found or some other error occurs, then a response of 401 is returned.

- **POST: /register** Takes a JSON file with 2 strings named "username" and "password". "username" is the username to register for and "password" is the password corresponding to the username. If the username or password is an invalid string, then a response of 403 is returned. If there already exists an account with the desired username, then a response of 409 is returned. If any other error occurs, a response of 500 is returned. On success, a response of 201 is returned.

- **NOTE: A valid string for a username or password is one that is a string and contains only numbers, uppercase letters, lowercase letters, and spaces.**

7.4.2 File operations:

- **GET: /api/get-collections** Returns a JSON file containing an array of all the names of all the simulations of the currently logged in user. If some non-login related error occurs, a response of 500 is returned. This request should not be used if you plan to access the user's uploaded simulation. Use **/api/get-directory** instead.

- **GET: /api/get-directory** Returns a JSON file containing an array of objects. There is an object for each of the logged in user's uploaded simulation where each object contains information about the individual simulation. "sim_name" is a string containing the simulation's name. "sim_date" is a string representing the simulation's date. "sim_id" is a unique number to access the corresponding simulation by.

- **GET: /api/select-uploads** Returns a JSON file containing an array of strings representing which files have been recognized and interpreted for a simulation. The query "collection_name" determines which simulation of the currently logged in user is to be retrieved by the simulation's name. If no simulation could be found, a response of 400 is returned. If any other error occurs, a response of 500 is returned. This request only exists for backwards compatibility and has no real use. Here are what the file titles correspond to:
  - **Simulation Details**: Output File 10 (Labeled Output Statistics)
  - **Average Traffic Conditions**: Output File 11
  - **Traffic Conditions**: Output File 12
  - **Paths**: Output File 13
  - **Trip Completion Probes**: Output File 14
  - **Road Probes**: Output File 15
  - **Summary**: Output File 28 (Summary Output File/ SUMMARY.OUT)
  - **Nodes**: Input File 1
  - **Edges**: Input File 2
  - **Signals**: Input File 3
- **POST: /api/delete-collection** Takes a JSON file with the string named "fileName" and deletes simulation from the logged in user using this value as the targeted simulation's name. If some other error occurs, a response of 500 is returned.
- **POST: /api/delete-upload** Takes a JSON file with the string named "fileName" and attempts to delete an individual file uploaded from some simulation. If some other error occurs, a response of 500 is returned. This is old and unused and should not be used as it neither does anything useful nor works.
- **POST: /api/upload** Takes a "single file" for upload (a zip file containing all files) and attempts to process it. The "body" value "collectionName" determines the name of the simulation". If no uploaded file was found, or if the file was not specified to be a zip file, or there already exists a simulation with the provided name, then a response of 400 is returned. If the simulation's name is invalid, a response of 403 is returned. If any other error occurs, an error of 500 is returned. The uploaded files must have specific names: Each file must contain it's file number in it's name. All input files must end in ".dat". All output files must end in ".out". These rules apply to all files except for file 28, which must have a name containing "summary" and ending in ".out", and file 10, which can have any name.
- **NOTE: All of these requests require login, and, therefore, can return all the same errors as /api/verify-token.**

7.4.3 Simulation requests:

All of these request return a JSON file contain information corresponding only to a single input or output uploaded within the ZIP file for the simulation. All requests require login. All requests require the query "sim" with the "sim_id" of the simulation to access. This

value is obtained from "/api/get-directory". If any error occurs, including no simulation have been found, a response of 500 is returned.
Each of these requests return a JSON file based on the files they originate from. The structures are shown for each request. For more information on the meaning of each parameter, look at the original documentation. Structure and order of values is mostly preserved with names of parameters shortened. Arrays and other variable length structures are shown with "..." within. The types of each parameter are shown and additional information, such as array length, are sometimes specified.

- **GET: /api/file-summary** Retrieves information from the "SUMMARY" file (File 28).

```
{
  total =
  {
  ...
  <name of stat> = <float[6], stat per vehicle class + average>
  ...
  }

  average =
  {
  ...
  <name of stat> = <float[6], stat per vehicle class + average>
  ...
  }
}
```

- **GET: /api/file-overview** Retrieves information from the "Labeled Output Statistics" file (File 10).

```
{
  signals =
  [
  ...
  { <signal info at time period>
    time = <int>
    signal = <int>
    a = [
    ...
    {
      pj = <int>
      ln = <int>
      link = <int>
      lane = <int>
      Arri Flow (vph) = <int>
      Saturation Flow = <int>
      Y-critical lane (%) = <float>
```

```
    Y-critical appr (%) = <float>
    Y-critical sum (%) = <float>
    Offset Time (sec) = <float>
    Cycle Time (sec) = <float>
    Lost Time (sec) = <float>
    Green Time (sec) = <float>
    Green Phase (sec) = <float>
    Inter Green (sec) = <float>
    Phase Start (sec) = <float>
    Phase End (sec) = <float>
  }
  ...
  ]

  b = [
  ...
  {
    ph = <float>
    Offset Time (sec) = <float>
    Cycle Time (sec) = <float>
    Lost Time (sec) = <float>
    Green Time (sec) = <float>
    Green Phase (sec) = <float>
    Inter Green (sec) = <float>
    Phase Start = <float>
    Phase End (sec) = <float>
  }
  ...
  ]
}
...
]

linkFlow = [
...
{
  time = <int>
  links = [
  ...
  {
    name = <string>
    link = <int>
    start = <int>
    end = <int>
    Speed = <int>
    Saturation = <int>
    Lanes = <int>
    Length = <float>
```

```
    "Link Flow (Vehs)" = <int>
    "Grn Time (%)" = <int>
    "V/C Rat (%)" = <int>
    "Total Travel Time (min)" = <int>
    "Free Travel Time (min)" = <float>
    "Avg Travel Time (min)" = <float>
    "Avg Speed (kph)" = <float>
    "Avg Stops" = <int>
    "Max Veh Pos" = <int>
    "Max Veh Obs" = <int>
    "Cur Veh Obs" = <int>
    "Total Travel Time (veh-min)" = <float>
    "Total Travel Time (veh-hrs)" = <float>
    "Total Network Travel (veh-km)" = <float>
    "Total Metwork Length (km)" = <float>
    "Average Network Speed (km/h)" = <float>
    "Average Trip Time/Veh (min)" = <float>
    "avarage trip Length/Veh (km)" = <float>
    "Num Invisible Vehicles" = <float>
    "Total Network Stops" = <float>
    "Average Network Stops" = <float>
  }
  ...
  ]
}
...
]

avgOD1 =
{
  stats = [
  ...
  {
    "Vehicle Type" = <int>
    "Origin Zone" = <int>
    "Destination Zone" = <int>
    "Number Departed" = <int>
    "Number Arrived" = <int>
    "Number Entered" = <int>
    "First Departure (min)" = <float>
    "Last Departure (min)" = <float>
    "First Arrival (min)" = <float>
    "Last Arrival (min)" = <float>
    "Total Trip Time (Veh-Min)" = <float>
    "Min Trip Time (min)" = <float>
    "Avg Trip Time (min)" = <float>
    "Max Trip Time (min)" = <float>
    "Trip Time SD (min)"  = <float>
```

```
    "Total Distance (Veh-Km)" = <float>
  }
  ...
  ]

  totals = [
  ...
  {
    class = <int>
    "Total Veh-Km" = <float; class != -1>
    "Total Veh-Hrs" = <float; class != -1>
  }
  ...
  ]
}

avgOD2 =
{
  stats = [
  ...
  {
    "Origin Zone" = <int>
    "Destination Zone" = <int>
    "Number Departed" = <int>
    "Number Arrived" = <int>
    "Number Entered" = <int>
    "Avg Trip Time (min)" = <float>
    "Trip Time SD (min)" = <float>
    "Total Trip Time (min)" = <float>
    "Max Pre-Trip Parked Vehicles" = <int>
    "Longest Pre-Trip Park Time" = <int>
    "Total Dist (Veh-Km)" = <float>
  }
  ...
  ]
  "All Vehicle Classes Total Veh-Km" = <float>
  "All Vehicle Classes Total Veh-Hrs" = <float>
}

"Sum of the total trip time (veh-mins)" = <float>
"Sum of the total trip time (veh-hrs)" = <float>
"Average trip time (mins)" = <float>
"Average trip time (secs)" = <float>
"Total demand to enter network" = <int>
"Vehicles eligible to enter" = <int>
"Vehicles in their driveways" = <int>
"Vehicles left on network" = <int>
"Vehicles that completed trip = <int>
```

```
  incd = [
  ...
  {
    "Link" = <int>
    "Start node" = <int>
    "End node" = <int>
    "Start time" = <float>
    "End time" = <float>
    "Duration" = <float>
    "Lane Losses" = <string>
  }
  ...
  ]
}
```

- **GET: /api/file-avgconds** Retrieves information from the "Average Traffic Conditions" file (File 11).

```
{
  time = <int>
  conditions = <int>
  edgeCount = <int>
  flow =
  [
    ...
    { <condtion at edge>
      edgeID = <int>
      length = <float>
      baseCapacity = <int>
      totalFlow = <int>
      flow = <int[5], flow by vehicle class>
      freeSpeedTime = <float>
      totalAverageTime = <float>
      averageTime = <float[5], average time by vehicle class>
      averageToll = <float[5], average toll by vehicle class>
      averageVehicles = <float>
      averageQueue = <float>
      averageStops = <float>
      fuel, HC, CO, NO, CO2, PM = <float>
      expectedCrashes = <float>
      expectedTopInjurt = <float>
      fatelCrashes = <float>
      crashLowDamage = <float>
      crashMedDamage = <float>
      crashHighDamage = <float>
    }
```

```
    ...
  ]
}
```

- **GET: /api/file-conds** Retrieves information from the "Traffic Conditions" file (File 12).

```
{
  periodCount = <int>
  time = <int>
  edgeCount = <int>
  edgeMaxID = <int>
  periods =
  [ <len = periodCount>
    ...
    { <condition at period>
      time = <int>
      index = <int>
      edges =
      [ <index in array == edge ID; len = edgeMaxID>
        ...
        { <edge condition>
          length = <float>
          baseCapacity = <int>
          totalFlow = <int>
          flow = <int[5], flow by vehicle class>
          freeSpeedTime = <float>
          totalAverageTime = <float>
          averageTime = <float[5], average time by vehicle class>
          averageToll = <float[5], average toll by vehicle class>
          averageVehicles = <float>
          averageQueue = <float>
          averageStops = <float>
          modelParameters = <float[8], "model parameters" (?)>
          fuel, HC, CO, NO, CO2, PM = <float>
          totalEnergy = <float>
          expectedCrashes = <float>
          expectedTopInjurt = <float>
          fatelCrashes = <float>
          crashLowDamage = <float>
          crashMedDamage = <float>
          crashHighDamage = <float>
        }
        ...
      ]
    }
    ...
```

```
  ]

}
```

- **GET: /api/file-paths** Retrieves information from the "Paths" file (File 13).

```
{
  maxOriginID = <int, the highest orgin zone ID>
  originCount = <int, the actual number of origin zones>
  maxDestID = <int, the highest destination zone ID>
  edgeCount = <int, the number of edges>
  maxEdgeID = <int, the highest edge ID>
  periods =
  [ <trees at specific time period; len = periodCount>
  ...
  {
    index = <int, index of time period>
    treeCount = <int, number of trees in period>
    paths =
    [ <tree; len = treeCount>
      ...
      {
        treeVal1 = <int, unused?>
        proportion = <float>
        index = <int>
        origins =
        [ <starting origins; len = maxOriginID>
          ...
          <int[]; len = maxDestID>
          [...<int, edgeID>...]
          ...
        ]
        edges =
        [ <starting edge; len = maxEdgeID>
          ...
          <int[]; len = maxDestID>
          [...<int, next edgeID>...]
          ...
        ]
      }
      ...
    ]
  }
  ...
  ]
}
```

- **GET: /api/file-tripprobes** Retrieves information from the "Trip Probes" file (File 15).
  - **Queries:**
    - **"origin"** The origin zone to filter for. -1 allows for all edges. Defaults to -1.
    - **"dest"** The destination zone to filter for. -1 allows for all edges. Defaults to -1.
    - **"skip"** The number of entries to skip. Only entries that would have been chosen otherwise are counted towards the number skipped.
    - **"max"** The max number of entries to find. Defaults to 500.
    - **"stride"** The progression of the number of entries. 1/n entries will be found compared to when stride = 1 and max = 99999999999.
    - **"time0"** The minimum time for entries to filter by. Inclusive. Defaults to 0.
    - **"time1"** The maximum time for entries to filter by. Exclusive. Defaults to 999999999.
  - **Response** "*origin*" $= -1$&"*dest*" $= -1$

```
{
  time0 = <min entry time>
  time1 = <max entry time>
  total = <total number of entries>
  pairs =
  [ <stats for each orgin/dest pair that has entries>
  ...
  { <stat for a pair>
    beg = <int, the origin zone>
    end = <int, the destination zone>
    quant = <the number of logs pertaining to this pair>
    time0 = <min time of entries of this pair>
    time1 = <max time of entries of this pair>
  }
  ...
  ]
}
```

  - **Response**

```
[ <array of entries>
...
{ <entry>
"Time simulation produced record" = <float>
"Vehicle ID number" = <int>
"Vehicle class" = <int>
"Vehicle last link" = <int>
"Origin node" = <int>
```

```
"s_Destination node" = <int>
"Scheduled departure time",
"Actual departure time",
"Trip duration",
"Total delay",
"Stopped delay",
"Number of stops",
"Distance covered",
"Average speed",
"Fuel used (L)",
"Hydrocarbon produced",
"Carbon monoxide produced",
"Nitrous oxide produced",
"CO2 produced",
"PM produced",
"hydrogen consumption (kg)",
"Number of expected crashes",
"Where injury was highest level",
"Where expected a fatal crash",
"Where maximum damage was low",
"Where maximum damage was moderate",
"Where maximum damage was high",
"Total toll paid",
"Total acceleration noise" = <float>
}
...
]
```

- **GET: /api/file-edgeprobes** Retrieves information from the "Edge Probes" file (File 16).
  - **Queries:**
    - **"edge"** The edge to filter for. -1 allows for all edges, -2 for no edges, but information about quantities of entries instead. Defaults to -2.
    - **"skip"** The number of entries to skip. Only entries that would have been chosen otherwise are counted towards the number skipped.
    - **"max"** The max number of entries to find. Defaults to 500.
    - **"stride"** The progression of the number of entries. 1/n entries will be found compared to when stride = 1 and max = 99999999999.
    - **"time0"** The minimum time for entries to filter by. Inclusive. Defaults to 0.
    - **"time1"** The maximum time for entries to filter by. Exclusive. Defaults to 999999999.
  - **Response ("edge" = -2)**

```
{
  time0 = <min entry time>
```

```
  time1 = <max entry time>
  total = <total number of entries>
  edges =
  [ <stats for each edge that has entries>
  ...
  { <stat for an edge>
    edgeID = <the edge's ID>
    numOfLogs = <the number of logs pertaining to this edge>
    time0 = <min time of log on this edge>
    time1 = <max time of log on this edge>
  }
  ...
  ]
}
```

– **Response**

```
[ <array of entries>
...
{ <entry>
  type = <int, format type: 11 or 21>
  time = <float, time of entry>
  vehicleID = <int, vehicle ID>
  vehicleClass = <int>
  vehicleType = <int; type = 11>
  edge = <int>
  lane = <int>
  nextEdge = <int>
  nextLane = <int>
  origin = <int, origin zone>
  dest = <int, destination zone>
  schedDepart = <float, schedualed departure time>
  departTime = <float, actual departure time>
  edgeTime = <float>
  delay = <float>
  stopDelay = <float>
  stops = <float>
  dist = <float>
  avgSpeed = <float; type = 11>
  finalSpeed = <float; type = 11>
  space = <float; type = 21>
  speed = <float; type = 21>
  accel = <float; type = 21>
  fuel = <float>
  energyRate = <float; type = 21>
  HC, CO, NO, CO2, PM = <float>
  energy = <float; type = 11>
  expectCrash = <float, number of expected crashes E-6>
```

```
    expectHighInjury = <float, number of expected crashes with high injury E-6
    expectFatal = <float, expected number of fatal crashes E-6>
    crashLow = <float, number of low damage crashes E-6>
    crashMed = <float, number of medium damage crashes E-6>
    crashHigh = <float, number of high damage crashes E-6>
    toll = <float>
    noise = <float>
  }
  ...
  ]
```

- **GET: /api/file-nodes** Retrieves information from the "Node" file (File 1).

```
{
  count = <int>
  xScale = <float>
  yScale = <float>
  nodes =
  [ <len = count>
  ...
  {
    id = <int>
    x = <float>
    y = <float>
    type = <int>
    zone = <int>
    info = <float>
    tag = <string>
  }
  ...
  ]
}
```

- **GET: /api/file-edges** Retrieves information from the "Edge" file (File 2).

```
{
  count = <int>
  lengthScale = <float>
  freeSpeedScale = <float>
  flowRateScale = <float>
  capSpeedScale = <float>
  jamScale = <float>
  edges =
  [ <len = count>
  ...
  { <node>
    id = <int>
```

```
    start = <int>
    end = <int>
    length = <float>
    freeSpeed = <float>
    satFlowRate = <float>
    numOfLanes = <float>
    speedVar = <float>
    capSpeed = <float>
    jamDensity = <float>
    prohIndc  = <int>
    enableTime = <int>
    disableTime = <int>
    oppose1 = <int>
    oppose2 = <int>
    signal = <int>
    phase1 = <int>
    phase2 = <int>
    vehProhIndc = <int>
    survLevel = <int>
    tag = <string>
  }
  ...
  ]
}
```

- **GET: /api/file-signals** Retrieves information from the "Signal" file (File 3).

```
{
  planCount = <int>
  planTime = <int>
  planNumber = <int>
  signals = [
  ...
  {
    signalNum = <float>
    baseTime = <float>
    minTime = <float>
    maxTime = <float>
    signalOff = <int>
    splitFreq = <int>
    phases = [...<int[2]>...]
  }
  ...
  ]
}
```

7.5 Database design

The database design has gone through a couple of iterations. The final and working version is shown in figure 16. For the simulations table the primary key is sim_id. The primary key of users is user_id. This information and more is revealed in sim_tables.sql. Figure 17 is a screenshot showing the creation of the tables and their relations and variable types.

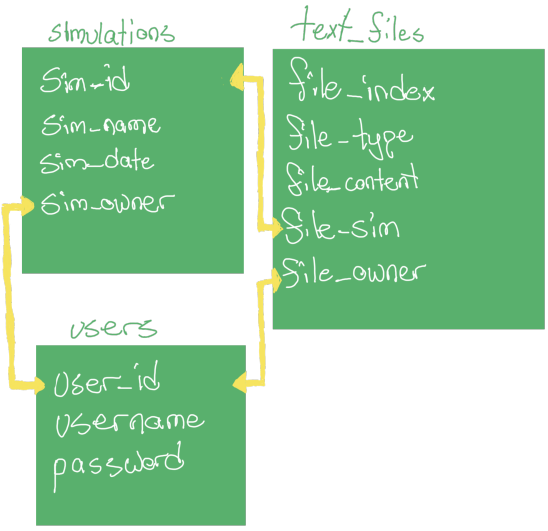

Figure 16: The Schema

```sql
USE traffic_visual;

-- Create the users table
CREATE TABLE IF NOT EXISTS users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS simulations (
    sim_id INT AUTO_INCREMENT PRIMARY KEY,
    sim_name varchar(32),
    sim_date varchar(32),
    sim_owner INT,
    FOREIGN KEY (sim_owner) REFERENCES users(user_id),
    UNIQUE (sim_name, sim_owner)
);

CREATE TABLE IF NOT EXISTS text_files (
    file_index INT AUTO_INCREMENT UNIQUE,
    file_type INT,
    file_content LONGBLOB, -- only takes up 2 more bytes
    file_owner INT,
    file_sim INT,
    FOREIGN KEY (file_sim) REFERENCES simulations(sim_id),
    FOREIGN KEY (file_owner) REFERENCES users(user_id),
    UNIQUE (file_sim, file_type)
);
```

Figure 17: The Schema Set Up

8 Lessons Learned

8.1 Timeline

Task	September	October	November	December
User Login Tokening System	•	•		
Upload Collections		•		
First Traffic Visualization		•		
Select Other Visualizations		•	•	
Visualization Per Output File			•	•
Draft 1 of Presentation		•		
Draft 2 of Presentation			•	
Final Presentation				•

Table 1: Project Timeline

8.2 Problems

Throughout the development of our website, we have encountered various problems in many diverse places. One of the main places where we have had trouble was with the **input files and documentation** to go off of. Multiple of the provided files have been updated past documentation or not given to the group making the documentation given for these files fail to match as well.

Another mainline problem had been **using AntV.** Upon importing the AntV library there are several version inconsistencies from what is shown from online documentation and what generative AI could grab. This led to excess time wasted trying to find and put together information on how to create the new graph type the Traffic Map, which also delayed creation of other visualizations and frontend innovations such as some interface ideas.

An additional problem we faced was ensuring that the **GUI was both intuitive and easy to understand** while also being **functional** and **informative** for the user. We found that achieving a balance between simplicity and feature-richness was challenging, as some design elements that were clear to us as developers were not as obvious to users.

Another time consuming issue that had arisen were **commit merges that deleted progress**. When using Github sometimes, when people were making changes some had to merge files and on occasion not enough caution was taken and some features were inadvertently removed, without any kind of redflag to the group. It was typically only in testing when it was discovered some of these features were removed, and while reinstating them didn't take as long as writing the features initially, it added some paranoia to pushing/pulling updates.

8.3 Solutions

To get around the inconsistent files and incorrect documentation, the files uploaded by the user are read so that error can be tolerated in the formatting. For example, the numbers in the files are padded with spaces so that they always take up the same amount of room, but these chunks themselves have no space between them, meaning that the individual values cannot be found by looking for spaces, which is the

typical solution. The way to get around this is by splitting the file into chunks based on the know sizes of each parameter, guaranteeing that all values are found. The bad documentation was worked around by looking at additional example files to try and guess the formatting as well as asking the client when the answer was not inherent.

The AntV solution was to install an older version of G6. In this case the project is running 4.3.11, which is much more documented. This has allowed generating graph much simpler and time saving.

To address the GUI problem, we conducted user testing sessions to gather feedback on the interface's intuitiveness. Based on this feedback, we simplified navigation, improved the layout, and added tooltips to make features clearer. Additionally, we implemented a more logical grouping of features and ensured that the most commonly used functions were easily accessible. This iterative design process helped us create a GUI that is both informative and user-friendly.

Solving merge deletions took someone to look at a push before pulling and ensure that nothing created had been accidently removed by version inconsistencies.

9 Future Work

9.1 General Future Work

The most critical work is completed at the time of writing. Here are some future works yet to be done:

1. Make visualization options in the selection interface dependent on their necessary input files. Right now a visualization will populate if its necessary output file is present. This is not the case for necessary input files such as files 1 and 2 for visualization utilizing the Traffic Map and is an oversight at the time of writing. This will doubtless lead to a bug in a collection without input files 1 and 2 were uploaded.

2. Make frontend file type names more flexible. In short, to add a visualization to the selection interface it must be named according to the backend file type naming scheme. Also, in RightSidebar.js the listing of each visualization and then file type again seems redundant and may be optimized for better flexibility. The reason of it not currently being done is due to time constraint and lack of immediate importance. At the time, the importance was placed on developing a method to allow all team members to add visualization for different file types at the same time.

3. Parsing the remaining file types. For starters, there is file 14, which has not been parsed (mostly due to lack of time and scope). There are also more input files which could be used to generate more useful graphs like how input files 1 and 2 where used to make the Traffic Map.

4. Break up server.js into multiple logical files. As shown in the developer manual server.js has many lines of code within and not for just one express purpose. It is hard to work on a file that big and it is easily cut into smaller logical files to handle tasks separately.

5. Handle token expiration better. At the current time, tokens have a one hour expiration timer. At the end of that time, the user is no longer able to access backend functionality, but instead of being asked to log back in, some on page errors show up(If a user takes action, else nothing of note happens). Like when trying to open a collection, it will say there is nothing in the collection. Right now, the solution is to refresh the page and it will make the user log back in. In the future either implement auto token refreshing or send the user to the login page. The point is to not leave the user in a situation where the application is not intuitive or intended.

6. Implement Database password encryption. At the current time since no one can access the MySQL Server, much less login to the server, there is no issue with not encrypting passwords. If this were a big application and this was confidential information, passwords should not be visible even to the Software Engineers who made the application.

7. Make the upload interface minimize-able to focus attention on collections

8. A confirmation menu on deleting a collection

9. Making the words "Collections Uploaded" appear only when there is at least one uploaded

10. There was a mention from the client of having a visualization history, check with the client to see if this is still desired.

11. Generate more visualization and their associated functions. This one is more vague and is up to future developers to design and think through.

9.2 Future Work: Visualizations Per File Type

In writing the final presentation, teammates were asked to put in future work for their file type visualization, since many did not finish and had a vision of where they wanted it to go. I am writing what I see them put in the slides with any minor insights I have alongside it(I being the team leader). Understanding how to create a visualization per file type takes contextual knowledge gained from the simulator documentation of the corresponding file type(which wasn't always up to date).

9.2.1 Summary File

Aside from basic improvements that make the summary visualizations more intuitive, there is a strong idea of allowing comparison of summary graphs across collections. At the time being a user can just hop to a different collection to see the difference, but there are no cross collections comparisons and it would be intuitive instead of making a user traverse and remember the past collection graph. The author of the custom summary chart was Brett Noneman.

9.2.2 Simulation Details

In the final presentation and demo Simulation Details' Time Optimization visualization was incomplete. From what I remember it was to span a fourth of the scope of file 10. The author of file 10 visualization was Xi Chen. The devoper noted to:

1. Add a bar chart to compare the Arrival Flow to the Saturation Flow
2. Understand the level of traffic congestion by comparing actual flow to maximum capacity
3. Add a pie chart and a bar chart to show the percentages of Y-critical lane(%), Y-critical approach(%), and Y-critical sum(%)
4. Highlight how critical lanes contribute to overall intersection efficiency.
5. Add a line chart to illustrate the progression of phase over time, such as Phase Start (sec) to Phase End (sec), and highlight Green Phase (sec) and Intern Green (sec)
6. Show how signal phases are sequenced and the duration of each phase

9.2.3 Average Traffic Conditions

The Average Traffic Conditions was initially authored by Xinchen Liao. She received assistance with the Traffic map by Brett Noneman.

1. Add two way road icon on the edge. Right now an edge is one way and two way roads are edges stacked on each other.
2. Add cars on the Traffic Map, using a number of cars to show traffic flow
3. Give edge popup information page more meaningful mini charts, which can show values in this edge in more insightful ways.

9.2.4 Minimum Path Trees

Minimum Path Trees was authored by Xi Chen.

1. It was a goal to make the tooltip and popup data more meaningful for users analyzing the minimum path tree
2. Replace or supplement current edge information with the number of crashes per edge to aid in path selection
3. Highlight crash statistics that help users prioritize safer routes while finding the minimum path tree
4. Provides actionable insights for informed decision-making

9.2.5 Trip Completion Probes

Trip Completion Probes was authored by Gabriel Worsley(Me the team leader!). File 15 is similar to summary file, but with control on individual vehicle statistics for vehicles that have completed their trip.

1. The Car Information Filter visualization will ideally be able to show the id of every vehicle ID, used. This is to grab vehicle ID information so it could be used in the second planned graph.
2. The second graph would be for comparisons on a smaller scope of cars showing their entire routes and perhaps contrasting to another car.
3. In order to show car information by car ID. It may require a backend update to query for vehicle ID. Since file 15 API endpoint doesn't have a vehicle id parameter, and searching in the frontend will create massive overhead.

9.2.6 Road Probes

Road Probes was authored by Brett Noneman. He notes that he wants to add additional, more detailed visualizations to provide deeper insight into:

1. Fuel use
2. Greenhouse gas emissions
3. Crash rates, and injury rates/fatalities for those crashes
4. Speed

He also wanted to create more visualizations that involved speed heatmaps, safety maps, speed maps, emission maps, etc.

10 Acknowledgments

Figure 18: Mohamed Farag

Dr. Farag is a research associate in the Center for Sustainable Mobility (CSM). His research interests include intelligent transportation systems, connected/automated vehicles, C-V2X, machine learning, large-scale data analysis, large-scale system analysis and design, big data, and information retrieval. Dr. Farag received Ph.D. in computer science from Virginia Tech in 2016, and his M.Sc. degree in computer science from the Arab Academy for Science, Technology, and Maritime Transport (AAST) in Alexandria, Egypt, in 2010. Thank you to our Client Mohamed Farag for the opportunity to work on this project.

11 References

References

[1] AntV G6 Documentation. Available at: https://g6-v3-2.antv.vision/en/docs/manual/introduction. Accessed on: September 9, 2024.

[2] D3.js Documentation. Available at: https://devdocs.io/d3/. Accessed on: September 13, 2024.

[3] Chart.js Documentation. Available at: https://www.chartjs.org/docs/latest/. Accessed on: September 17, 2024.

[4] Sam Meech-Ward YouTube Channel. Available at: https://www.youtube.com/@SamMeechWard. Accessed on: September 21, 2024.

[5] Fireship YouTube Channel. Available at: https://www.youtube.com/@Fireship. Accessed on: September 25, 2024.

[6] Simplilearn YouTube Channel. Available at: https://www.youtube.com/@SimplilearnOfficial. Accessed on: September 29, 2024.

[7] Web Dev Simplified YouTube Channel. Available at: https://www.youtube.com/@WebDevSimplified. Accessed on: October 3, 2024.

[8] The Code Bootcamp YouTube Channel. Available at: https://www.youtube.com/@TheCodeBootcamp. Accessed on: October 6, 2024.

[9] Node.js API Documentation. Available at: https://nodejs.org/docs/latest/api/. Accessed on: September 11, 2024.

[10] React Documentation. Available at: https://devdocs.io/react/. Accessed on: October 5, 2024.