



# Building Web App for Automated Vehicles Energy and Emissions Estimations

Client: Mohamed Farag

Final Report  
**CS 4624**  
Virginia Tech, Blacksburg, VA 24061  
**Spring 2025**

Prepared by:  
Trevor White  
David Rankin  
Harsha Paladugu  
Katelyn Crumpacker

Group 10

May 6th, 2025

# Table of Contents

- Executive Summary/Abstract..... 3
- 1 Introduction..... 4
- 2 Requirements..... 5
- 3 Design..... 6
- 4 Implementation..... 10
- 5 Testing/Evaluation/Assessment..... 13
- 6 Users' Manual..... 14
- 7 Developer's Manual..... 26
- 8 Lessons Learned..... 29
- 9 Acknowledgements..... 31
- 10 References..... 32

# List of Figures

- Figure 1: System Architecture.....6
- Figure 2: Frontend Components..... 7
- Figure 3: Database Design.....8
- Figure 4: Particle Matter Calculations.....12
- Figure 5: Landing Page..... 14
- Figure 6: Sign Up Page..... 14
- Figure 7: Login Page..... 15
- Figure 8: Admin Dashboard.....15
- Figure 9: Collection Page.....16
- Figure 10: Collection Creation Form.....16
- Figure 11: Add New Metric Form..... 17
- Figure 12: Template File..... 18
- Figure 13: Vehicle Parameters.....19
- Figure 14: Update Parameters.....19
- Figure 15: Add New Vehicle Form.....20
- Figure 16: Edit Vehicle Form.....21
- Figure 17: Initial Collection Analytics Page.....21
- Figure 18: Sidebar.....22
- Figure 19: Key Metrics of a Collection.....22
- Figure 20: Efficiency Analysis of a Collection.....23
- Figure 21: Dropdown Menu.....23
- Figure 22: Collection Management Menu.....24

# List of Tables

- Table 1: Timeline.....29

## Executive Summary/Abstract

As transportation evolves towards sustainability, accurately understanding and comparing vehicle energy consumption and emissions becomes critical. In 2022, the transportation sector accounted for about 29% of total greenhouse gas emissions in the United States, making it the largest contributor among all sectors [1]. This shows the impact transportation has on our environment. Data collection in the transportation sector aids sustainability by enabling precise tracking of vehicle emissions, energy use, and traffic patterns, which helps identify inefficiencies and supports the development of cleaner, more efficient transportation systems. Without data, it is extremely difficult for decisions to be made to try and increase sustainability and decrease impact in transportation and other areas that contribute heavily to emissions. Our project addresses this need by creating an intuitive web application leveraging advanced energy consumption models developed by the Virginia Tech Transportation Institute (VTTI) for various vehicle types, including Internal Combustion Engine Vehicles (ICEV), Battery Electric Vehicles (BEV), Hybrid Electric Vehicles (HEV), and Hydrogen Fuel Cell Vehicles (HFCV). ICEVs are the typical gasoline burning vehicles that the majority of people today use [2]. BEVs solely use rechargeable batteries to operate [3]. HEVs use the same internal combustion engines that the ICEVs use, but they also have electric motors that run on battery power. These batteries are charged using regenerative braking or the engines, but not by plugging in like BEVs [4]. HFCVs generate electricity through a chemical reaction between hydrogen and oxygen in a fuel cell, powering an electric motor[5]. Users can input speed and acceleration profiles from these different types of vehicles to obtain immediate visual comparisons of estimated energy consumption and particle emissions. This allows users to compare and contrast between different vehicle types to compare them across different metrics. Whether the user is a researcher or just someone trying to figure out which car to purchase, this web app has uses. Building upon previous work, we significantly enhanced user experience through robust frontend changes for a better user interface, optimized performance for large datasets, improved data visualizations using Plotly.js, and streamlined user interactions. Our improvements also include intuitive file upload/download functionality, enhanced database management, a way for users to upload specific vehicles (make, model and year), support for particle matter analysis and flexible visualization options that enable users to select specific vehicles when displaying calculation data. By offering a dynamic platform for both general users and administrators, this web app democratizes complex analytics, empowering stakeholders and the public to make informed decisions regarding energy-efficient and environmentally responsible transportation.

# 1 Introduction

## 1.1 Problem and Motivation

Transportation is a significant contributor to energy consumption and emissions, impacting both environmental health and economic factors. While researchers at VTTI have developed sophisticated models for vehicle energy consumption, a gap remains in making these tools broadly accessible and user-friendly for the general public and researchers. Given the escalating concerns over vehicle emissions and energy usage, there is a clear need for accessible, comparative visualization tools that inform users' transportation choices. This project was built off of a previous team's effort and web app which can be read about in the link in the references section [6].

## 1.2 General Approach

Our approach involves converting a complex command-line interface (CLI) tool into a comprehensive web application that integrates VTTI's energy consumption models. We employ a robust technology stack consisting of React.js and Node.js for frontend development, Python Flask for backend API integration, SQLite for database management, and Plotly.js for advanced visualization leveraging GPU acceleration for enhanced performance on large datasets. Additionally, we will implement a dedicated file upload/download interface for administrators, allowing easy addition and management of new calculation models. Also, we will add a vehicle upload interface enabling the upload of new specific vehicles to analyze and compare. The frontend and backend of this system will be comprehensively tested to ensure system reliability and ease of use. User interface enhancements such as more comprehensive and descriptive instructions and improved data visualization methods further enhance the user experience.

## 2 Requirements

### 2.1 Key Functionalities:

- User-friendly graphical interface for speed and acceleration data input.
- Real-time graphical visualizations for energy consumption and particulate matter emissions across vehicle types (ICEV, BEV, HEV, HFCV).
- Automated computation using integrated VTTI models.
- Feature for administrators to intuitively upload and manage new computational models via the frontend.
- Combined visualization of different vehicle types with appropriate units.
- Intuitive controls for users to select specific vehicles and manage parameters.

### 2.2 Technical Requirements:

- Frontend: React.js, Node.js, Plotly.js (for efficient rendering of large datasets).
- Backend: Flask, Python, Werkzeug, REST API.
- Database: SQLite with integrated management and unit conversions.
- Deployment environment: Docker.

### 2.3 Team Skillset:

- Python programming
- Text processing and data management
- Web application development and UI/UX design
- Data visualization and analysis
- Automation and scripting
- Database management and optimization

This structured approach ensures delivery of a robust, user-centric web platform that significantly enhances the accessibility and usability of advanced vehicle energy and emissions analytics.

## 3 Design

### 3.1 System Architecture

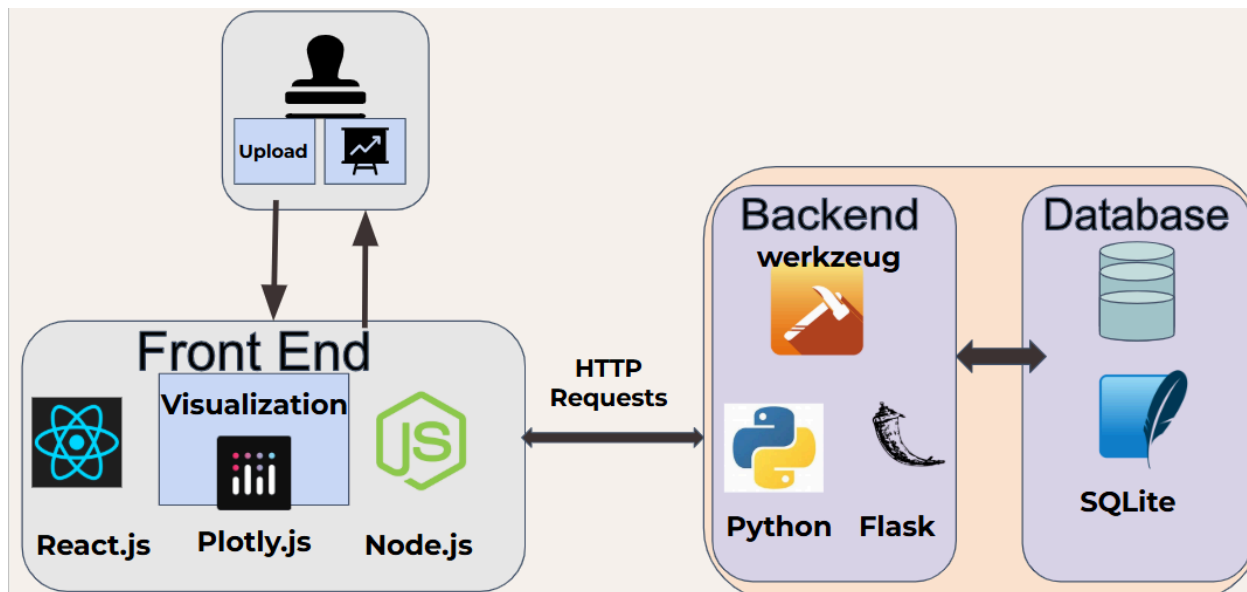


Figure 1: System Architecture

The system follows the client-server model, and the system architecture consists of ...

- User Interface (UI): includes the ability to upload files and displays the data to the user
- Frontend: component system using React-based JS that specifically uses plotly.js to display data to the user quickly in a visually appealing manner
- Backend: Python using Flask, which leverages Werkzeug for request handling and routing, and exposes functionality through REST API.
- Database: stores the data for the system using SQLite

Changes to this architecture vary in difficulty based on the area the changes are in. Changing the visualization package is relatively simple. It involves changing package imports, installing new packages, and replacing code where the previous visualization package is used. Changing the database could also be feasible; however, all of the endpoints use SQL code to add to the SQLite database, so all of those endpoints would have to be modified, and the other database would have to be connected. However, changing from using Flask and React would be very difficult and not recommended.

### 3.2 Frontend Component Changes

The previous team working on this project included a lot of components, routes, and pages needed for this project. While we updated and added to a lot of the routes and components, we only added two, which are highlighted in red in the image below.

Key Components:

- Admin: Makes sure certain components are able to be accessed by admins
- Routes: Handles the navigation to different components
- Welcome Page: Landing page for users
- Collections: Allows users to upload speed profiles and see visualizations of them
- Vehicle Page: Allows admins to view, add, update, and delete vehicle params and types
- Calculation Page: Allows admins to view and add metrics

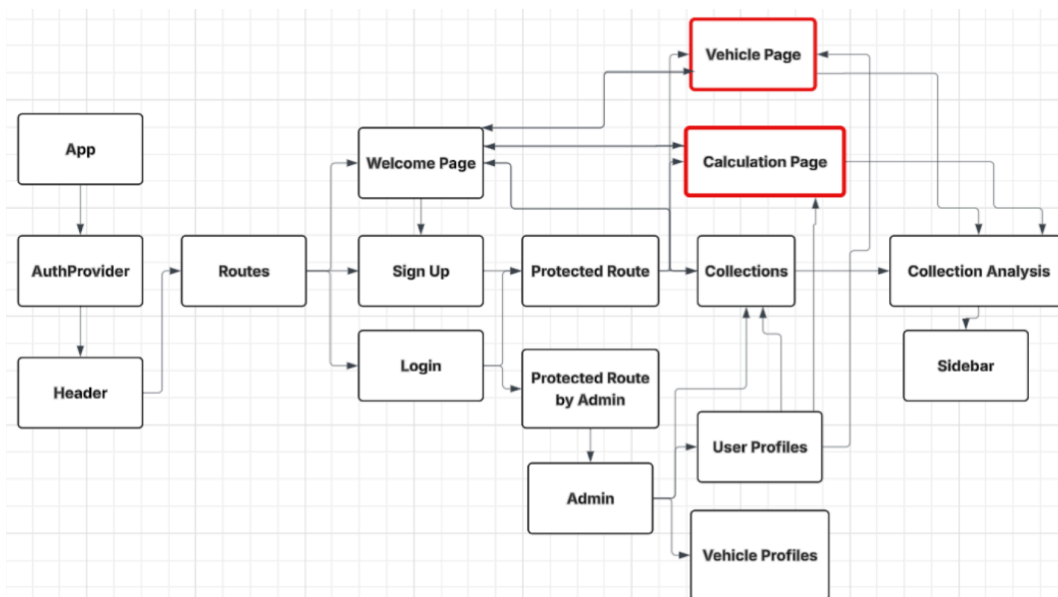


Figure 2: Frontend Components

### 3.3 Overview of Features Added

- Support uploading and managing vehicle types, as well as a metric uploading system for admins.
- Enhanced GUI with sidebar, displaying vehicle name and engine types, as well as different collections for the user to make the experience more intuitive.
- Created an option for users to see the data for all of the different vehicle types on one combined graph as well as separate
- Along with the ability to upload new metrics, we added the ability for users to see data displayed on particulate matter

### 3.4 Database Design Changes

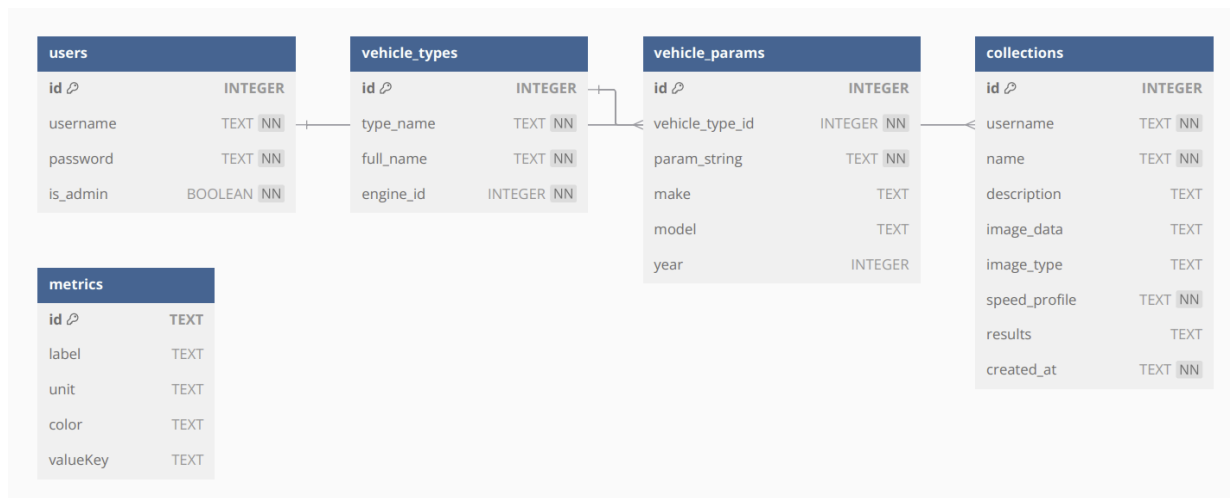


Figure 3: Database Design

- The previous team's database design already included the users, vehicle\_types, vehicle\_params and collections tables
- Our team used the vehicle types and vehicle params tables to implement the vehicle type upload feature
- Our team added the metrics table to allow admins to upload new metrics that are then integrated into the visualizations

### 3.5 Designing the Template

To enable customizable and extendable energy/emissions analysis, we created a Python-based template file structure that administrators can download, edit, and then re-upload in order to define new vehicle metrics. The template serves as a tool that administrators could use to insert new metric-specific logic while maintaining the standard data flow required by the system. Internally, this makes the logic used in the template easily integrated with the back end Flask API and the front end Plotly component for visualization. When administrators submit the template, the system automatically loads and executes the file to deliver updated results, allowing admins to add functionality without having to change any actual source code

### 3.6 Other Design Decisions

Another essential design decision was moving from Recharts to Plotly.js for graph and data visualization. The key motive behind this move was to enhance analysis methods and user experience, particularly in the case of rendering large datasets. Plotly.js has many features like GPU rendering, zooming, and tooltips, allowing for a more flexible display, especially with intricate time-series data in a range of vehicles. Making this change allowed us to show both the combined view and the separate graph view, enabling the user to have multiple different

visualization techniques on vehicles. Another design decision was choosing to have an option for the user to display the data about each vehicle type for the chosen metric all on one graph. The previous team allowed the user to view different graphs for each vehicle type and select metrics for the separate graphs. While this is helpful information, our team thought it would be beneficial to add an option for users to compare different vehicle types data for the different metrics.

# 4 Implementation

## 4.1 Upload Features

### 4.1.1 Vehicle Type Upload and Display Design

#### 4.1.1.1 Frontend

The previous team had a vehicles page, but we wanted to update it to allow admins to manage vehicle types. This includes creating a button that the user can click to add a new vehicle type. This button opens a form where the user inputs the required information about the vehicle type, which is then sent to the backend via a POST request so it can be retrieved when viewing graphs or a list of the vehicle types. The vehicle types are displayed in the table, which has buttons next to each vehicle type to edit and delete. Editing brings up the form, but it's populated, and allows the user to change the information, which is propagated to the backend with a PUT request. Delete asks if the user is sure, and then deletes the vehicle type with a DELETE request sent to the backend.

#### 4.1.2 Backend

For each of the functionalities described above, the backend has a corresponding endpoint to perform the task and store the data in the database. These will be further expanded on in the endpoints section.

## 4.2 Metric Upload and Display Design

### 4.2.1 Frontend

For the metric upload, we created a new page that includes an input form for the user to put in details about the metric, a download button for the user to get the template file, and a submission button and box for users to submit new metrics.

### 4.2.2 Backend

For users to upload a metric file, a POST endpoint receives the uploaded file and metric metadata, stores them in the SQLite database, and registers the metric so it can be dynamically called during energy/emissions calculations. The uploaded template is saved to the server and is imported at runtime using Python's dynamic import system. Error handling mechanisms are in place to validate the file structure and function definitions to ensure the uploaded metric adheres to expected standards.

## 4.3 API Endpoints

- Authentication
  - POST /api/signup – Register.
  - POST /api/login – Log in for users and admins.

- Vehicle Types
  - GET /api/vehicle-types – Retrieve all vehicle types.
  - POST /api/admin/vehicle-types – Add a new vehicle type for admins.
  - DELETE /api/admin/vehicle-types/<type\_id> – Delete vehicle type by ID for admins.
- Vehicle Parameters
  - GET /api/vehicle-params – Get all vehicle parameter entries.
  - POST /api/admin/vehicle-params – Add new vehicle parameters for admins.
  - PUT /api/admin/vehicle-params/<param\_id> – Update a vehicle parameter for admins.
  - DELETE /api/admin/vehicle-params/<param\_id> – Delete a vehicle parameter by ID for admins.
  - POST /api/admin/clear-vehicle-params – Clear vehicle parameters for admins.
- Collections
  - POST /api/collections – Create a collection.
  - GET /api/collections/<username> – Get collections for a specific user.
  - GET /api/collections/<collection\_id> – Get a collection by ID.
  - PUT /api/collections/<collection\_id> – Update collection parameters.
  - DELETE /api/collections/<collection\_id> – Delete a collection with the username parameter.
  - GET /api/collections/<collection\_id>/download – Gets a ZIP with simulation results.
- Energy Estimation
  - POST /api/estimate – Get energy and emissions estimation with uploaded speed profile and vehicle ID.
- Metrics
  - GET /api/metrics – Get all metric definitions.

## 4.4 Energy Calculations

Methods in which we calculate energy emissions are a core functionality for our application in visualizing accurate data. For each of the vehicle types - BEV, HEV, HFCV, ICEV - there are different calculation files - `calculate_total_fuel_VEHICLETYPE.py` - to estimate metrics such as fuel consumption and efficiency. Each of these calculations factors in different methods such as quadratic fuel rate, cubic regression, auxiliary power, and net power draw. Adding on to these key metrics, we added a standard ParticleMatter calculation function as well, which estimates PM2.5 emissions using different vehicle parameters: battery capacity, power data, speed, etc. Particulate matter (PM) refers to a mixture of tiny solid particles and liquid droplets suspended in the air, including dust, soot, and metal particles, many of which are emitted by vehicle exhaust. Measuring PM is crucial for assessing the sustainability of different vehicle types because fine particles (especially PM2.5) can penetrate deep into the lungs and bloodstream, posing serious health risks and contributing to environmental pollution [7].

The process of building the model involved first understanding the original Fortran implementation, then translating it into Python while incorporating our specific vehicle parameters. The particle matter, which focuses on emission-generated braking, also mainly focuses on ICEV and BEV vehicle types because they rely heavily on mechanical and regenerative braking. It is calculated by applying a brake emission function on the vehicle deceleration cubed, while also adjusting values based on power draw.

```
adjusted_dec = dec
if dec < 0.0:
    adjusted_dec = max(-6.0, dec)
if engineType == "ICEV": # ICEV
    pm25_brake = 0.052 * (0.548782 + 0.000305 * mass) * abs(adjusted_dec) ** 3.195
elif engineType == "BEV": # BEV
    if power < power_min:
        print("in here")
        adjusted_dec = (power - power_min) / power * adjusted_dec if power != 0 else 0
    pm25_brake = 0.052 * (0.548782 + 0.000305 * mass) * abs(adjusted_dec) ** 3.195
```

Figure 4: Particle Matter Calculations

The tire emissions are then modeled using the decay function -  $pm25\_tire = 2.97213e-6 * spd * mass * \text{math.exp}(-0.009375 * spd)$  - which when added to the PM2.5 emissions give use to total emission value. Ultimately, our particle matter calculation function returns both the PM2.5 rates and total emission value in milligrams.

## 5 Testing/Evaluation/Assessment

Testing for the Vehicle Energy Estimation Web Application focused on ensuring the accuracy of backend functionality, user authentication, and system operations. Currently we are just doing manual tests.

### 5.1 Frontend

Most frontend features are tested through direct use and user interaction. However, with the implementation of Plotly for time and efficiency, a test was needed to ensure it was faster than the previous version.

#### 5.1.1 Plotly versus Recharts Time Test

Running both versions of the application each three times, one version with Recharts and another version with Plotly. The Recharts implementation yielded an average of 3.01 seconds load time of four 130 point graphs. Whereas the Plotly implementation yielded an average of 2.71 seconds load time. Which was significant enough to allow for Plotly implementation over Recharts.

### 5.2 Backend

#### 5.2.1 Available Tests

There are 3 scripts that can be run to make sure the backend is working as it should. The scripts are made up with a series of curl requests that hit the backend endpoints and the person running the tests can verify the responses are correct.

The tests are ...

- test\_login.sh
  - Tests that the user is able to sign up, login, and then it gets the list of users to make sure the user is added to the database.
- test\_vehicle.sh
  - Tests that the user is able to add vehicle types and params, delete vehicle types and params, and update vehicle types and params. Then it gets the list of vehicle types and params to check that the database has been updated accordingly.
- test\_metric.sh
  - Tests that the user is able to add metrics and then gets the list of metrics to check that the database has been updated accordingly.

The scripts prompt the user to run `python init_db.py` after the tests so the database is returned to its pre-test state.

### 5.3 Database

While there are no explicit tests, the backend tests and working features on the website have proven that the database is being updated correctly after different actions like logging in, adding new metrics, etc.

# 6 Users' Manual

**Note:** We only added section 6.3 Adding New Metric and 6.4 Adding New Vehicle Type, the rest of the sections were created by the previous team.

This manual acts as a guide to the Automated Vehicles and Energy and Emissions. The user will be shown how to login or sign up, create and manage collections, create and manage new metrics, create and manage vehicle types, and logging out.

## 6.1 Getting Started

### 6.1.1 Welcome Page

1. Running the project locally or online.
2. Click "Sign Up" if you do not already have an account
3. If you already have an account click "Login"

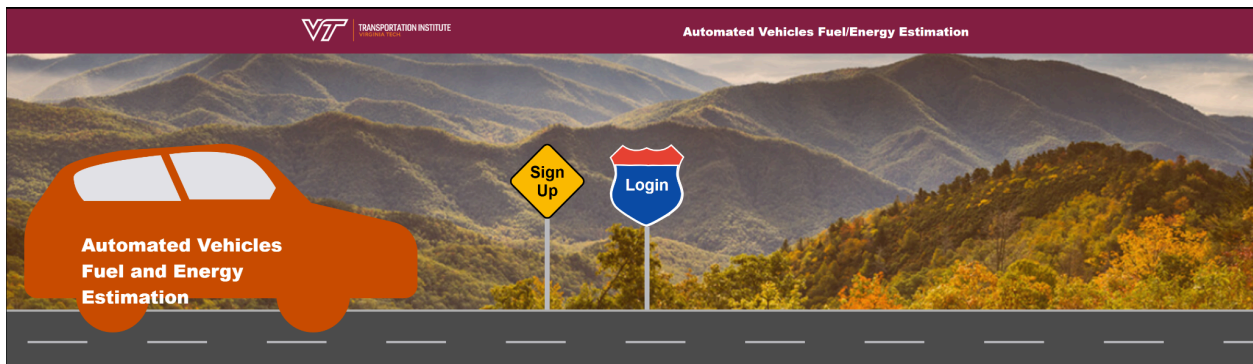


Figure 5: Landing Page

### 6.1.3 Sign Up

1. Enter in a username and password.
2. Click the orange "Sign-Up" button and a login will be created.

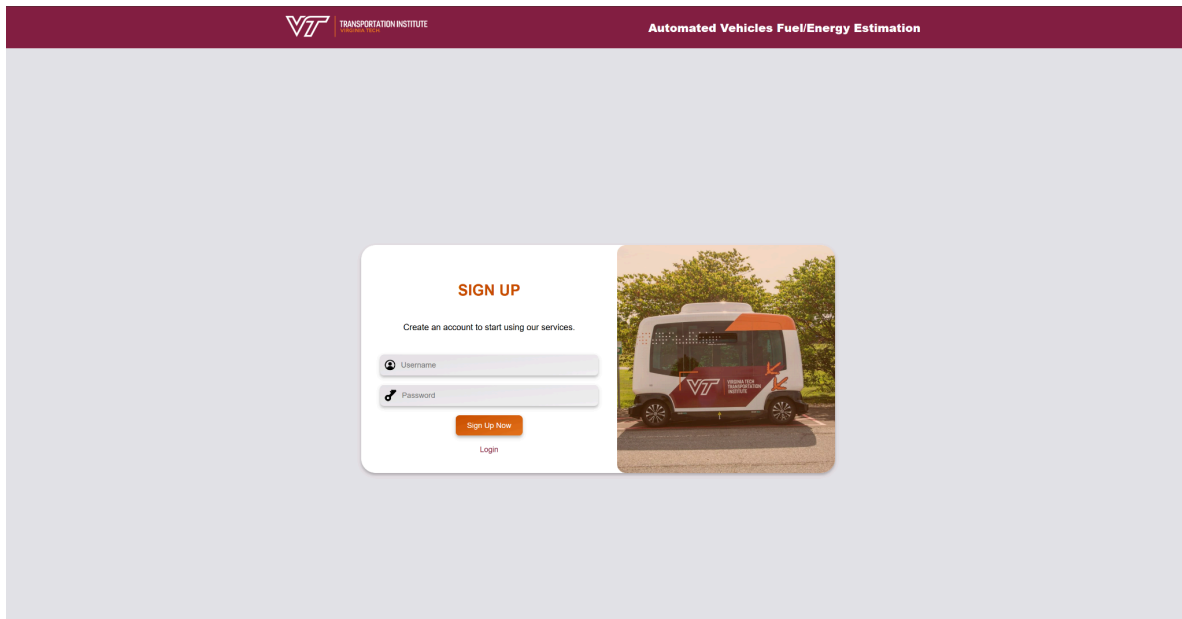


Figure 6: Sign Up Page

### 6.1.3 Login

1. If you already have an account, enter in your username and password.
2. Click the maroon "Log In" button to continue.

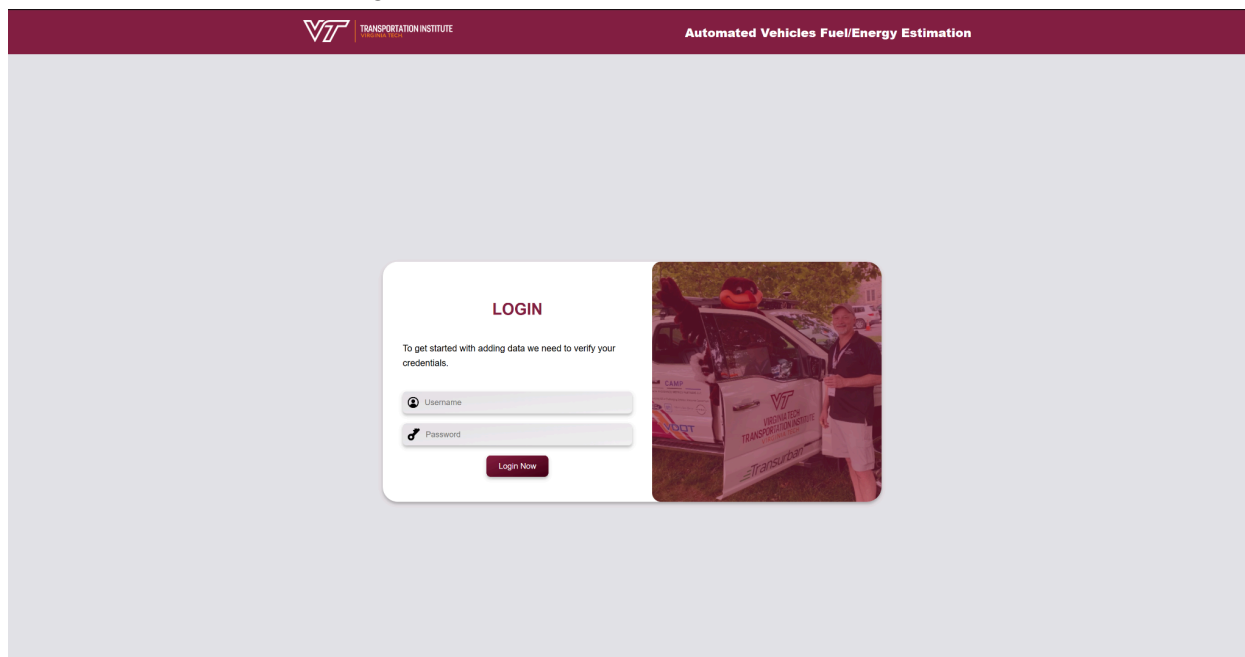


Figure 7: Login Page

### 6.2 Collection Dashboard

After logging in, user's will see their collections unless they are an admin then they will see the main dashboard, which houses a "User Collections" button. When clicked on the admin will be presented with collections. These collections provide a way to store data in one central area. New user's will not start out with any data.

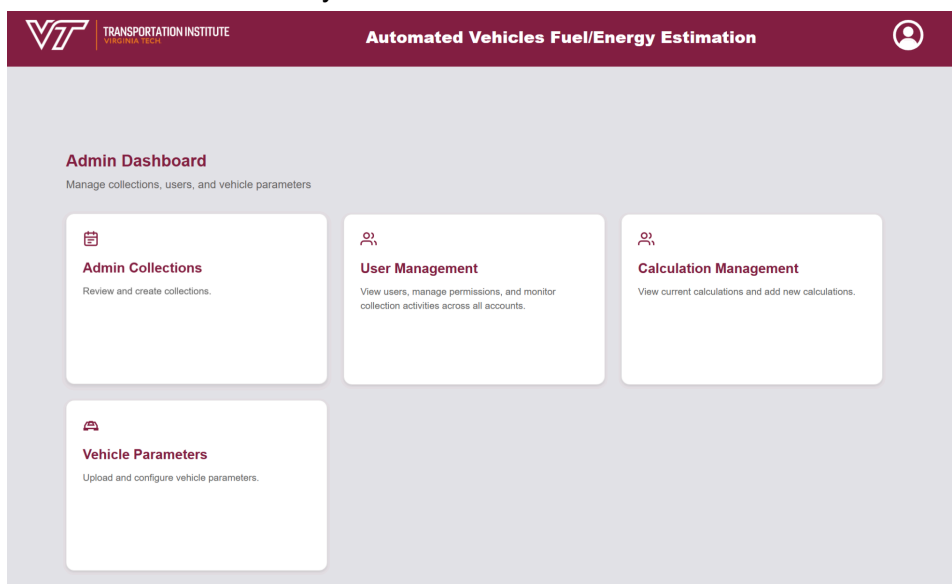


Figure 8: Admin Dashboard

## 6.2.1 Create a Collection

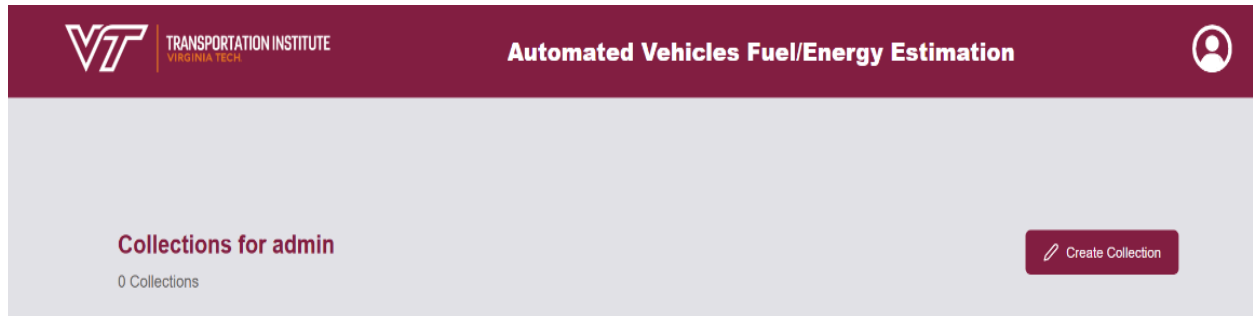


Figure 9: Collection Page

1. Click the maroon *"Create Collection"* button to get started.
2. A form will appear where you can define your collection.
  - **Required Fields:**
    - **Collection Name:** Give your collection a descriptive name.
    - **Speed Profile:** Specify the speed profile for your collection.
  - **Optional Fields**
    - **Description:** Add details to describe your collection.
    - **Image:** Upload an image to visually represent your collection.

### Create New Collection

Upload a speed profile to analyze vehicle energy consumption

Collection Name \*

Collection Image

Click to add image

Description

Speed Profile \*

↑

Choose file or drag here

.txt or .csv, max 5MB

Cancel

Create Collection

Figure 10: Collection Creation Form

- When you have finished filling out the form click the maroon *"Create Collection"* button

## 6.3 Adding New Metric

Adding a new metric is an admin only feature. Returning to the admin dashboard, navigate to the *"Calculation Managements"* button to add a new metric to be displayed on the graphs.

### 6.3.1 Filling out the form

After clicking on the badge, the user will be prompted with the following window:

The screenshot shows a web interface for adding a new metric. At the top, there is a maroon header with the VT Transportation Institute logo on the left, the title 'Automated Vehicles Fuel/Energy Estimation' in the center, and a user profile icon on the right. Below the header, the main content area is light gray and contains the following elements:

- Title:** 'Add a New Metric' in maroon text.
- Instructions:** A block of text in bold: 'Input the name and units for the metric. Download the Python template file to add a new calculation. Upload the completed template to then be able to see your metric in the collections visualization dashboard. Click "Submit Metric" Button.'
- Input Fields:** Three white text input fields with maroon borders. The first is labeled 'Enter a name for the new metric', the second 'Enter the units for the new metric', and the third 'Enter a unique model name that matches the one in the template file'.
- Download Button:** A white button with a maroon border and text 'Download Template File' located below the first input field.
- Upload Button:** A white button with a maroon border and text 'Upload File' located below the third input field. Above it is a yellow folder icon and the text 'Click to upload your completed template file .py file only'.
- Submit Button:** A white button with a maroon border and text 'Submit Metric' located at the bottom center of the form.

Figure 11: Add New Metric Form

The user should start by downloading the python template file, then entering in the new metric name, units and a unique identifier/model name. Once this is done the user can go on to edit the template file to allow for a new metric to be returned, after this the user can submit the metric and view it with the others.

### 6.3.2 Updating the Template

After downloading the template, open it in your python editor of choice. Change what you want, and adapt the *return* field to return the new metric you want to be displayed.

```
def fuction_name(
    v,          # Array of vehicle speeds (m/s)
    power_kw,  # Array of power values (kW)
    engine_type, # This will be the current engine the function is being called on
    parameters # Dictionary of required parameters
):
    """
    Template function for calculating total fuel/energy consumption. Modify this function to suit different vehicle types or models.
    Parameters:
    v (array): Speed data in meters per second.
    power_kw (array): Power consumption data in kilowatts.
    parameters (dict): A dictionary containing necessary coefficients and vehicle parameters.
    Returns:
    dict: A dictionary containing calculated metric(s)
    """
    # Extract parameters (modify or add new parameters as needed)
    param1 = parameters.get('param_name1', 0)
    param2 = parameters.get('param_name2', 0)
    param3 = parameters.get('param_name3', 0)
    # model that is used to calculate the metric value for the given speed modify for your metric
    model_name = np.where(power_kw < 0, param1, param1 + param2 * power_kw + param3 * power_kw**2)
    allowed_engine_types = ['Engine1', 'Engine2']
    if engine_type not in allowed_engine_types:
        return {
            'model_name': model,
            'units': 0
        }
    # Compute value(s) needed to calculate the total value of your metric over the speed profile
    total_metric = param1 + param2
    # Add any conversion values necessary
    return {
        'model_name_that_matches_input': model.tolist(),
        'units_that_match_input': total_metric
    }
}
```

Figure 12: Template File

## 6.4 Updating Vehicle Parameters

Adding new vehicle parameters is an admin only feature. Returning to the admin dashboard, navigate to the “*Vehicle Parameters*” button to add a new metric to be displayed on the graphs.

### 6.4.1 Viewing Vehicle Parameters

After clicking on the button, the user will be shown the current list of Vehicle Parameters as well a section for Update Parameters and Format Requirements. In the list of the Current Vehicle Parameters, the user can view a car’s make, model, year, mass, length and more.

**Vehicle Parameters**

Current vehicle configurations and their parameters.

ID	Type	Make	Model	Year	Mass (kg)	Length (m)	Mass Proportion	Friction Coef.	Engine Power (kW)	Battery Power (kW)	Trans. Efficiency	Drag Coef.	Front Area
18	BEV	BEV	Generic	2024	4.75	0.6	0.5	317	317	0.92	0.23	2.652	1.75
20	HFCV	HFCV	Generic	2024	1,858	4.89	0.6	0.5	113	113	0.92	0.28	2.372
22	HEV	Ford	Fusion	2019	1,600	4.3	0.62	0.77	140	160	0.87	0.3	2.4
26	HEV	HEV	Generic	2024	1,600	4.3	0.62	0.77	140	160	0.87	0.3	2.4
27	BEV	Tesla	Model3	2022	1,800	4.2	0.6	0.7	200	220	0.9	0.28	2.3
28	HFCV	Hyundai	Nexo	2023	1,700	4.6	0.64	0.74	130	145	0.86	0.31	2.6
29	ICEV	Generic	ICEV	2024	1,500	4.5	0.6	0.015	100	100	0.92	0.3	2.2
30	ICEV	Toyota	Corolla	2024	1,375	4.63	0.6	0.015	97	97	0.92	0.29	2.2

Figure 13: Vehicle Parameters

## 6.4.2 Adding, Deleting and Updating Vehicle Parameters

**Update Parameters**

Manage vehicle types and upload parameter configurations.

**Available Vehicles** Add New Vehicle

**Internal Combustion Engine Vehicle**

Make	Model	Year	Actions
Generic	ICEV	2024	<a href="#">Edit</a> <a href="#">Delete</a>
Toyota	Corolla	2024	<a href="#">Edit</a> <a href="#">Delete</a>

**Battery Electric Vehicle**

Make	Model	Year	Actions
BEV	Generic	2024	<a href="#">Edit</a> <a href="#">Delete</a>
Tesla	Model3	2022	<a href="#">Edit</a> <a href="#">Delete</a>

**Hybrid Electric Vehicle**

Make	Model	Year	Actions
Ford	Fusion	2019	<a href="#">Edit</a> <a href="#">Delete</a>
HEV	Generic	2024	<a href="#">Edit</a> <a href="#">Delete</a>

**Hydrogen Fuel Cell Vehicle**

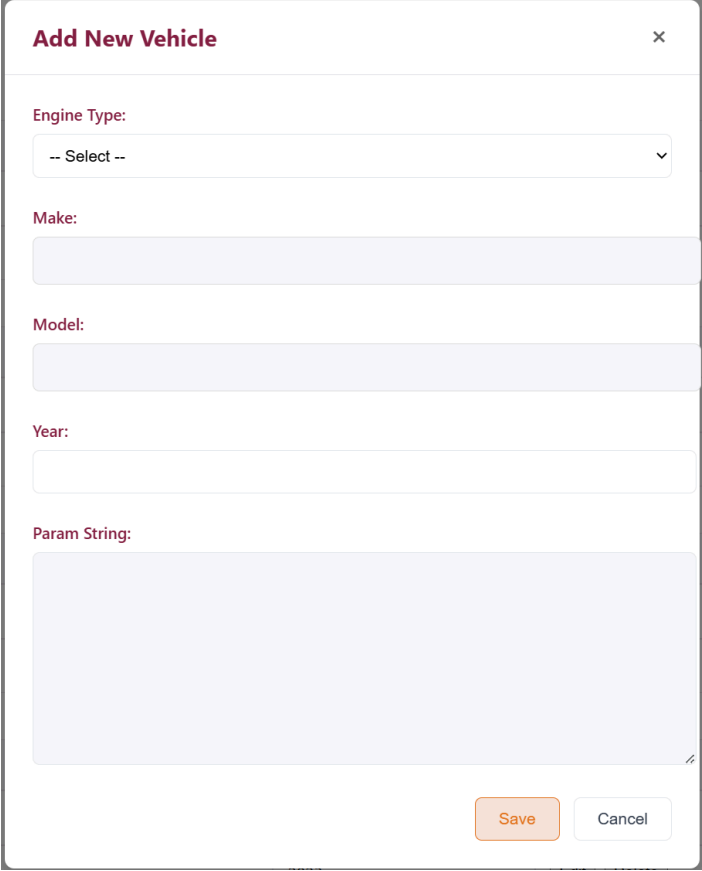
Make	Model	Year	Actions
HFCV	Generic	2024	<a href="#">Edit</a> <a href="#">Delete</a>
Hyundai	Nexo	2023	<a href="#">Edit</a> <a href="#">Delete</a>

Figure 14: Update Parameters

Under Vehicle Parameters, there is Update Parameters. Here an admin can delete, edit and add new Vehicles:

- Add:** To add a new vehicle click the maroon “Add New Vehicle” button at the top left of the Update Vehicle section. Here the user will be prompted to fill out the form below.
 

*Note:* For requirements on how to fill out the form check *Parameter Format Requirements* just below the Update Parameters section.



**Add New Vehicle** ×

Engine Type:  
-- Select --

Make:

Model:

Year:

Param String:

Save Cancel

Figure 15: Add New Vehicle Form

- **Delete:** To delete a vehicle just press the grey “Delete” button located in the Action row for the vehicle you want to delete.
- **Edit:** To edit a vehicle press the grey “Edit” button, located in the Action row for the vehicle you want to edit. Pressing the button will open up the form below. Where the user can change every aspect of a vehicle.

**Edit Vehicle** ×

**Engine Type:**  
Internal Combustion Engine Vehicle ▼

**Make:**  
Generic

**Model:**  
ICEV

**Year:**  
2024

**Param String:**

```
0 1 1500 4.50 0.60 0.015 100 100 0.92 0.30 2.20 0.015 4 0 0 3.50 0 0 0
0 0 0 0 0 0 0.01 0.02 0.03 0.00
```

**Save** **Cancel**

Figure 16: Edit Vehicle Form

## 6.5 Analyze Collections

Upon creating a new collection you will be redirected to the Data Analytics Page. It will appear blank at first. Follow the directions below to populate the page.

**VT** TRANSPORTATION INSTITUTE **Automated Vehicles Fuel/Energy Estimation** 👤

**Select Engine Types**

**Key Metrics**

Efficiency Analysis

Select All

ICEV

BEV

HEV

HFCV

**Analyze Selected Engines**

**Main Street**

Figure 17: Initial Collection Analytics Page

### 6.5.1 Sidebar Use

To run the models on your uploaded speed profile, follow these steps:

1. Check the engine types you would like to analyze in the sidebar.
2. Click the maroon *“Analyze Selected Engines”* button.

- If the button is grey and not maroon, you need to select at least one engine type before proceeding.

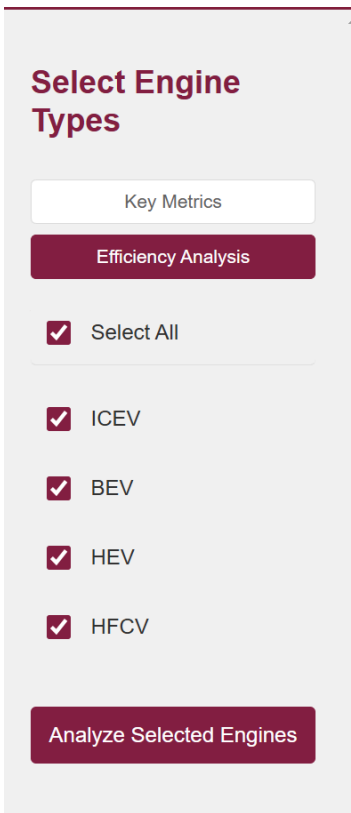


Figure 18: Sidebar

3. After running the analysis, key metrics for the selected engines will be displayed.

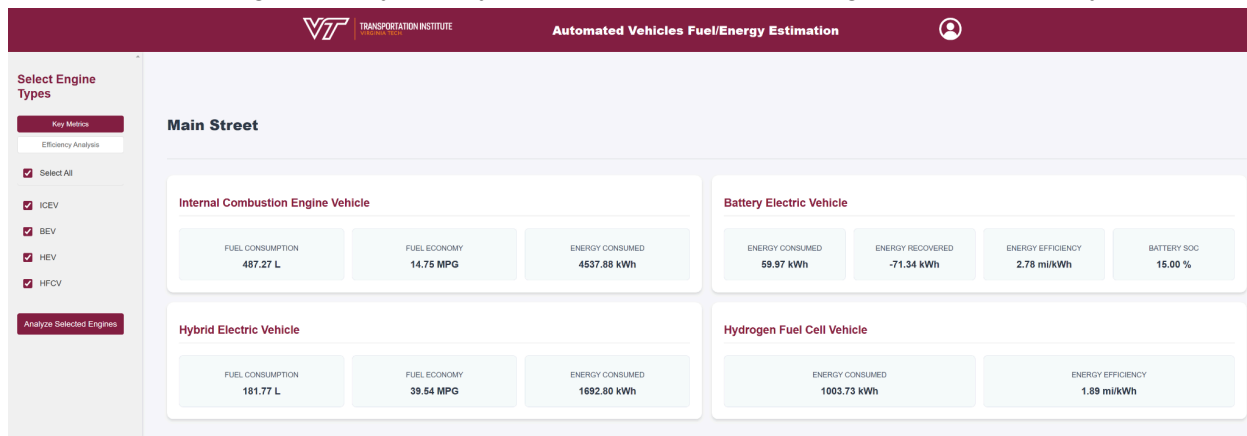


Figure 19: Key Metrics of a Collection

4. To view graphical representations, return to the sidebar and select “Efficiency Analysis”.

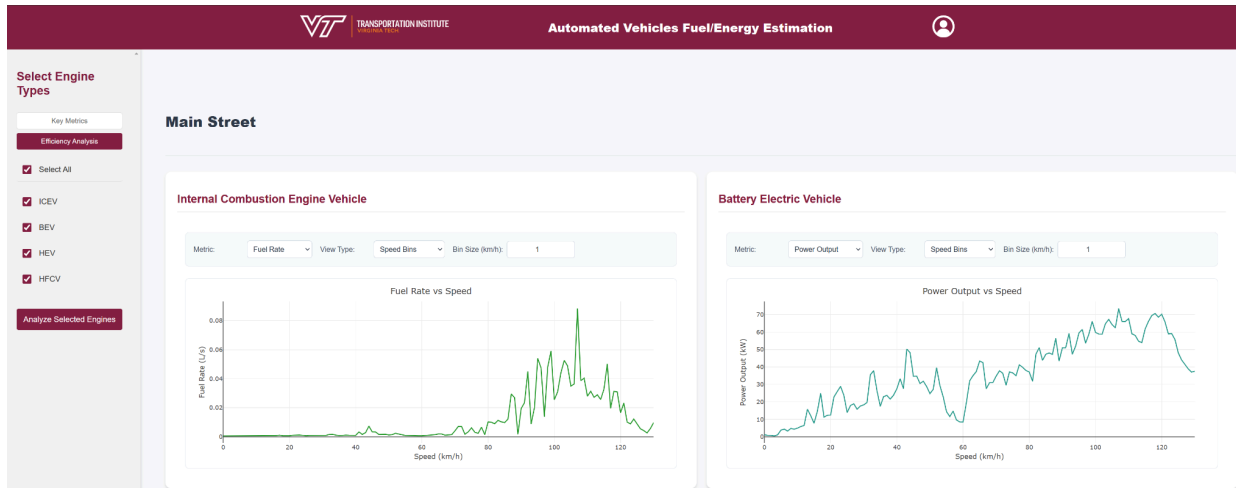


Figure 20: Efficiency Analysis of a Collection

## 6.6 Return to Collections Dashboard

To return to your Collections Dashboard, follow these steps:

1. Hover your cursor over the profile icon located at the top-right corner of the screen.
2. From the dropdown menu that appears, select the “Collections” option to access your dashboard.

Additional options available in the dropdown menu; revisit the landing or welcome page or log out to end your session.

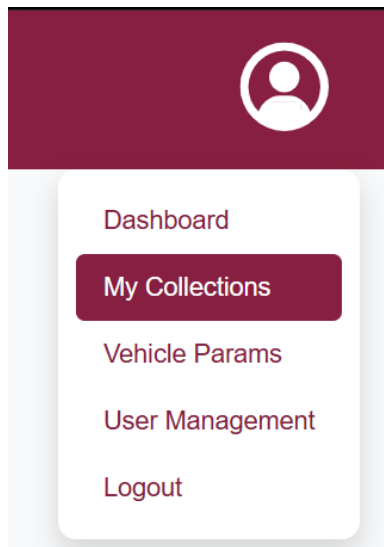


Figure 21: Dropdown Menu

## 6.7 Read, Update, and Delete Collections

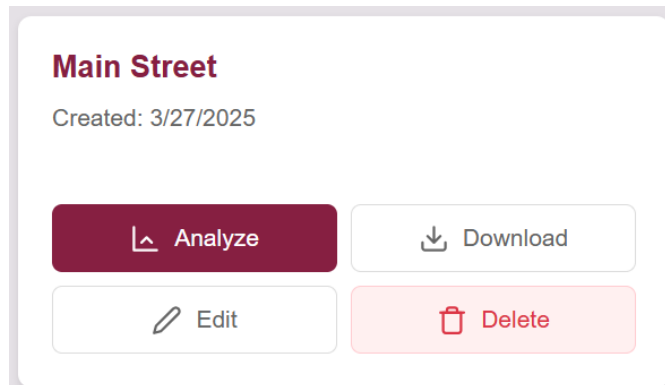


Figure 22: Collection Management Menu

Each collection comes with the following core functionality to help you manage and interact with it:

- **Read:** Allows you to download the data associated with the collection.
- **Write:** Enables you to make changes to the collection's name, description, image, or speed profile.
- **Update:** Permanently removes the collection and its data.

### 6.7.1 Read

To download the data from a collection, follow these steps:

1. Navigate to the collection you want to download and click the "Download" button on the collection management menu.
2. A .zip file containing the collection's data will be sent to your downloads folder.
3. Once the .zip file is downloaded, extract its contents. The extracted folder will include all data files related to the collection.
4. **Contents of the Downloaded Folder:**
  - a. **Top-level Files:]**
    - speed\_profile.csv: Contains the speed profile data for analysis
    - vehicle\_comparison.csv: Summary comparison of vehicles.
    - README.txt: A text file with additional information about the contents and usage.
  - b. **Results Directory:**
    - **ICEV (Internal Combustion Engine Vehicle):**
    - **BEV (Battery Electric Vehicle):**
    - **HEV (Hybrid Electric Vehicle):**
    - **HFCV (Hydrogen Fuel Cell Vehicle):**
    - Each vehicle type has the three following files
      - analysis\_results.json: Contains analysis results for vehicle type.
      - time\_series.csv: Time series data for vehicle type.
      - parameters.txt: Parameters used for vehicle type analysis.

### 6.7.2 Update

To update a collection, click the “*Edit*” button on the collection card (see Figure 18). This will take you back to the same form as the “Create Collection” page, pre-filled with the current details of the collection. You can update the following:

- **Collection Name:** Modify the name of your collection.
- **Description:** Edit the description of your collection.
- **Collection Image:** Change or upload a new image to represent the collection.
- **Speed Profile:** Upload a new speed profile file (.csv or .txt format).

Once you have made the necessary changes, click the maroon “*Update Collection*” button at the bottom of the Collection Management Menu.

### 6.7.3 Delete

To delete a collection, click the red “*Delete*” button on the Collection Management Menu. A confirmation prompt will appear to ensure that you want to proceed with the deletion.

**Note:** *Deleting a collection is a permanent action and cannot be undone. Ensure that you have downloaded any necessary data before confirming the deletion.*

## 6.8 Log Out

After managing your collections, don’t forget to log out to ensure your account remains secure.

To log out:

1. Click on the profile icon located at the top-right corner of the screen.
2. From the Dropdown Menu, select “*Logout*”.

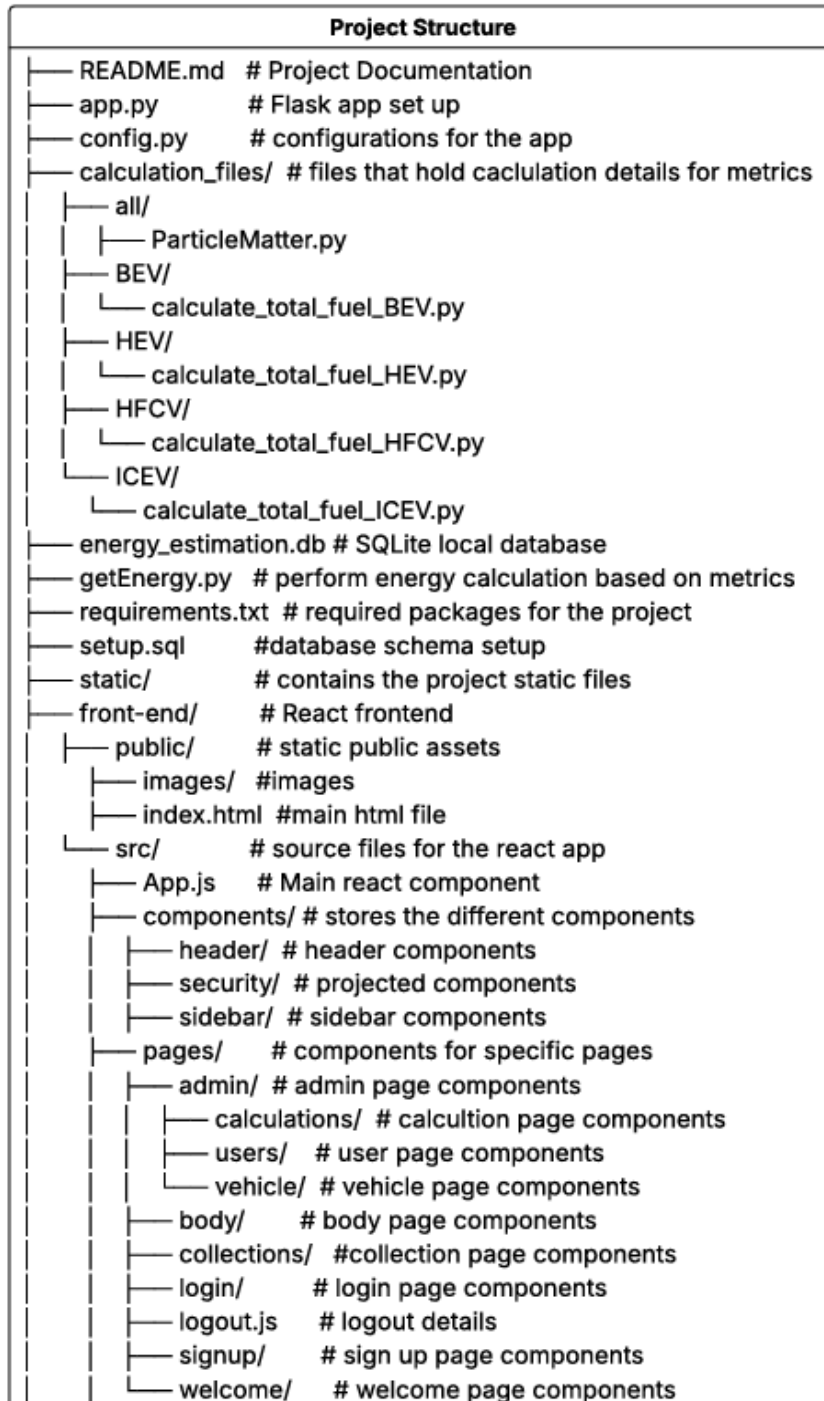
**Note:** *Logging out is especially important if you are using a shared or public computer to protect your data.*

# 7 Developer's Manual

Project Repository: [loopisturf/CS4624-Project10](https://github.com/loopisturf/CS4624-Project10)

This is the manual that will provide all the information on how to maintain and continue development of the Vehicle Energy Estimation Web App. It provides instructions on how to set up the backend, frontend, database, docker containers, and their connections.

## 7.1 Project Structure



## 7.2 Development Environment Setup

### 7.2.1 Running the System Locally

Enter the following commands in your terminal...

#### Backend set up

**Clone the project and go into the newly created project directory**

```
git clone https://github.com/loopisturf/CS4624-Project10.git
cd CS4624-Project10/
```

**Create a virtual environment for the project and activate it**

```
python -m venv venv
Linux: source venv/bin/activate
Windows: venv\Scripts\activate
```

**Install the required packages**

```
pip install -r requirements.txt
```

**Run the backend**

```
python app.py
```

#### Frontend set up

**Create a new terminal and go into the frontend folder**

```
cd CS4624-Project10/front-end/
```

**Install the required packages and run the frontend**

```
npm install react-scripts@latest or
npm start
```

#### Running the website during development

Once you have done these steps, the process to run the project simplifies

**Activate your virtual environment**

```
Linux: source venv/bin/activate
Windows: venv\Scripts\activate
```

**Run the backend**

```
python app.py
```

**Create a new terminal and go into the frontend folder**

```
cd CS4624-Project10/front-end/
```

**Run the frontend**

```
npm start **
```

\*\* if you have added new packages you will need to run npm install

**The frontend should run on <http://localhost:3000>**

**The backend should run on <http://localhost:5000>**

### 7.2.2 Running the system with Docker

Running with docker needs to be done on a place docker compose can be accessed (common ones are command prompt, powershell, and git bash), it will not work on vscode

**Open Docker desktop on your computer and open your terminal of choice**

- Link to install docker desktop for windows: [Windows | Docker Docs](#)

**Clone the project and go to the newly created project directory**

```
git clone https://github.com/loopisturf/CS4624-Project10.git
```

`cd CS4624-Project10/`

### Checkout the branch with files for docker

`git checkout` deploying

### Build and run the project with docker-compose

`docker-compose up` —build

Click on the <http://127.0.0.1:5000> url to access the website

\*\* if you don't make any change you can just run docker-compose up and not have the build every time

## 7.3 How to Contribute

### 7.3.1 Git Workflow

#### Create a branch for your contribution/feature

`git checkout -b` <your\_branch\_name>

#### Make any changes and add those changes to the remote repository

`git add` <file\_name\_you\_changed>

\*\* repeat git add until all files you changed are added or use...

`git add` —all

`git commit -m` “<your commit message>”

#### Push to the remote repository

If it is the first time pushing to your branch ...

`git push` —set-upstream origin <your branch name>

If it is not the first time pushing to the branch...

`git push`

### 7.3.2 Tips

- If you are working on a branch with someone, or checking out someones branch, make sure to do a `git pull` so you can get all of the changes from the remote repository
- Make sure to keep your branch up to date with what is all main, so your development doesn't get behind

## 7.4 Troubleshooting

- If the website is taking a while to load (it will load eventually) if you restart the backend and then the frontend it might take less time
- If you are having issues with data showing up on the website run `python init_db`

## 8 Lessons Learned

### 8.1 Timeline

Schedule			
Date	Status	Phase	Task
February 15th	Completed ▾	Planning	Project page and client meeting
February 27th	Completed ▾	Planning	Complete project design and execution plan
March 15th	Completed ▾	Prototyping	Code base and data base set up
March 30th	Completed ▾	Implementation	Switch to plotly, file upload feature for calculations, have data displayed on 1 graph
April 15th	Completed ▾	Implementation	File upload for vehicle parameters, admin page includes all updated admin abilities, hashing passwords, final frontend changes
May 6th	Completed ▾	Deployment	Website deployed, final report and presentation completed

Table 1: Timeline

### 8.2 Problems

One of the main struggles this semester was understanding legacy code. Since this project is done with a limited time from, the code written is not always commented well as a result. This made the code a little difficult to follow and it took a lot of effort to follow the flow of the user inputted data through the code.

There was also a lot of discussion about how to report to the user that the metric they have selected does not work for a certain vehicle or engine type. We didn't know whether to display an "error message" to the user, not have the metric available in the drop down for some graphs, or a completely different solution. We also had to make it work for the separate graphs and for the combined graphs, which needed different approaches. I think we struggled over-complicating this issue a lot before we landed on a simpler solution.

Another issue we had to tackle was converting the fortran code about particle matter to python and having it follow the template of our system while keeping the fortran code functionality in

tact. There were not any real problems in this process, fortran was just a language none of us had encountered before, so we had a lot to learn about it.

### **8.3 Solutions**

In response to the code legacy issues, during development the team added a lot of print statements and console.log statements to the code. This allowed us to follow the flow of the data from the user input or the database, through the backend endpoints, calculation files, and frontend components. These print and console.log statements allowed us to solve a lot of problems where we would see the message: "No data for visualization available". If we had had more foresight, I think mapping out the flow of the data from the database to being displayed to the user before doing any coding would have been beneficial.

For reporting to the user which metrics were allowed for which vehicle and engine types, we tried to simplify the task and so when the user clicks on a metric that doesn't work for a specific vehicle or engine type in the separate graphs, the message: "No data available for visualization message appear". However, in the combined graph, only lines that corresponded to an allowed vehicle or engine type would appear on the combined graph, so there would just be less lines than for a selected metric that works for all vehicle and engine types.

The solution for the fortran conversion was just doing research on methods in fortran and searching for the python equivalent.

### **8.4 Future Work**

This semester we met the primary objectives of providing an option for visualizing the data all in one graph, allowing admins to upload new metrics and vehicles that are reflected in the visualization graphs, and including a new particle matter metric. The current implementation and design includes the ability of users to upload a collection for a specific road and upload the speed profile on that road so calculations can be made for each car and compared. There are also default collections about Main Street in Blacksburg and I-81. Data on these roads is not as interesting to users who are not from Virginia, however. As a result, an interesting addition for future semesters would be to allow the user to input their state and see public data about the different calculations for different vehicles that are based on speed profiles of a popular road in their state. Currently users without speed profile data, especially users not from Virginia would not get much out of this web app. This would just be for general educational purposes about the environmental impacts of different types of cars that users can see without having to have their own data. Another future application building off these public speed profile data visualizations, is to allow users to crowdsourcing to these public visualizations if they see anything that needs adding or changed.

## 9 Acknowledgements

We would like to thank our client Dr. Mohamed Farag (mmadgy@vt.edu) for providing the motivation behind this project. He guided the implementation of this project through his ideas for features and technical advice. This ultimately allowed this project to come to fruition.

We would also like to thank the Virginia Tech Transportation Institute (VTTI) for providing us with the calculations so the project could show the environmental impact of different cars using different measurements.

Another thank you goes out to the team members on this project.

- Katelyn Crumpacker - backend developer, group leader
- Trevor White - backend and database developer
- David Rankin - authentication and frontend developer
- Harsha Paladugu - frontend and backend developer

This project was made possible through the teamwork and collaboration of this team. We hope this web app will aid in research related to the energy emissions of different vehicle and engine types.

## 10 References

- [1] United States Environmental Protection Agency, "Inventory of U.S. Greenhouse Gas Emissions and Sinks," *US EPA*, Jan. 15, 2025.  
<https://www.epa.gov/ghgemissions/inventory-us-greenhouse-gas-emissions-and-sinks>
- [2] Vehicle Technologies Office, "Internal Combustion Engine Basics," *Energy.gov*, Nov. 22, 2013. <https://www.energy.gov/eere/vehicles/articles/internal-combustion-engine-basics>
- [3] U.S. Department of Energy, "Alternative Fuels Data Center: How Do Hybrid Electric Cars Work?," *U.S. Department of Energy*, 2019.  
<https://afdc.energy.gov/vehicles/how-do-hybrid-electric-cars-work>
- [4] "Battery Electric Vehicles | NIST," *NIST*, Nov. 05, 2024.  
<https://www.nist.gov/el/applied-economics-office/manufacturing/circular-economy/battery-electric-vehicles>
- [5] U.S. Department of Energy, "Alternative Fuels Data Center: How Do Fuel Cell Electric Vehicles Work Using Hydrogen?," *Energy.gov*, 2019.  
<https://afdc.energy.gov/vehicles/how-do-fuel-cell-electric-cars-work>
- [6] C. Quinn, L. Scott, C. Chao, E. Tapia, and R. Batra, "Building Web App for Automated Vehicles Fuel/Energy Estimation," Dec. 2024, Accessed: Mar. 27, 2025. [Online]. Available: <https://vtechworks.lib.vt.edu/server/api/core/bitstreams/c0077b75-5021-451f-b62a-c93506057585/content>
- [7] EPA, "Particulate Matter (PM) Basics," *United States Environmental Protection Agency*, Jul. 11, 2023. <https://www.epa.gov/pm-pollution/particulate-matter-pm-basics>