Intel® Unnati Industrial Training Program

Project Report

GPS Toll-based System Simulation using Python

Coimbatore Institute of Technology

Department of AI and DS

Reviewed By:

Dr. M. Prabhavathy (Mentor),

Associate Professor,

Dept. of AI & DS.

Team Members:

Anirudh K	- 71762208007
Edward Samuel L	- 71762208013
Gokul P	- 71762208015
Jaaswin S	- 71762208023

Table of Contents

PROBLEM STATEMENT

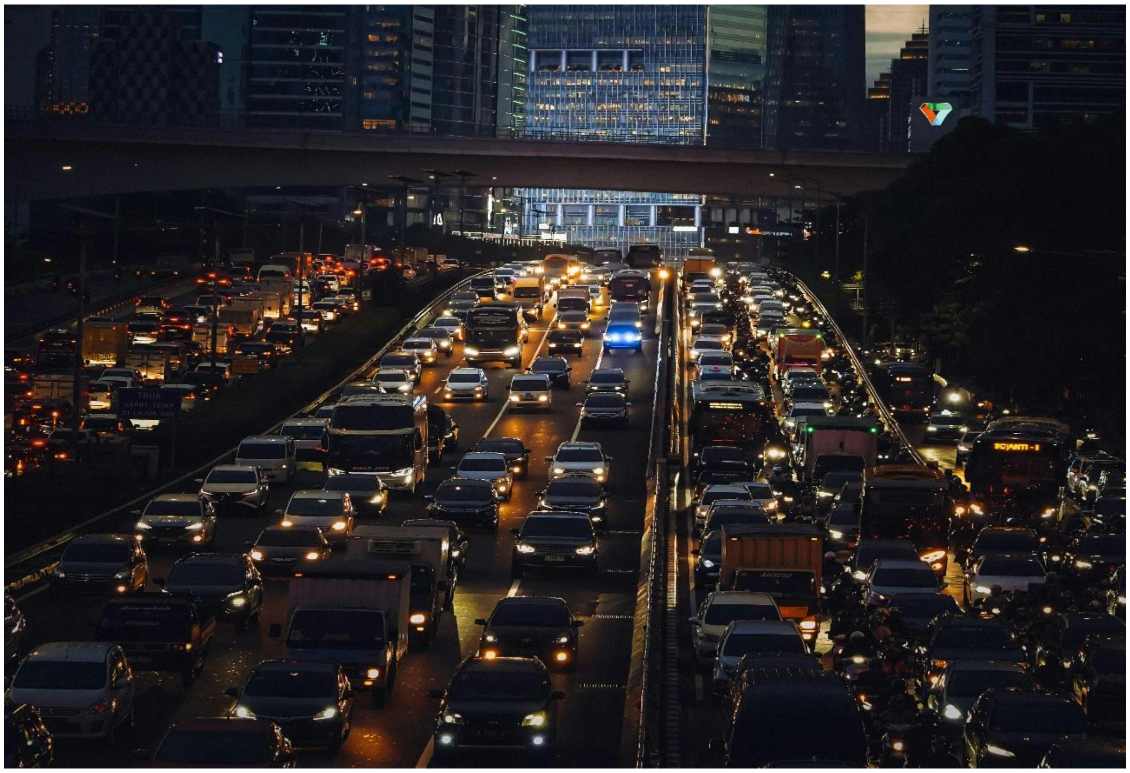
- With increasing population, industrialization, urbanization and pace of life, there is a need to increase speed and efficiency of our transportation system. Establishment of toll roads has dynamically increased in tandem with expansion in road transportation networks.
- Toll plazas play a crucial role in maintaining the speed and efficiency of road transportation. Toll fee collected is used for expansion, operational and maintenance purposes.
- Traditionally, toll fee is collected manually via cash or card at dedicated toll booths. However, processing time at such plazas is extremely high due to manual intervention. This results in traffic jams and huge concentration of cars causing inconvenience to travelers.

EXISTING TOLL SYSTEM:

- FASTag is an efficient Toll collection system used in India to reduce vehicle congestions and vehicles passing through need not stop and interact with a person to pay the toll. The toll is automatically deducted from their accounts.
- Separate electronic toll collection lanes are set up, equipped with scanners that continuously send out RF electronic pulses.
- FASTag is a device that employs Radio Frequency Identification (RFID) technology for making toll payments directly from the prepaid account linked to it.
- It is affixed on the windscreen of your vehicle and enables you to drive through toll plazas, without stopping for cash transactions. You will have to use the lanes demarcated for FASTag.
- The tag has a validity of 5 years and after purchasing, you only need to recharge/ top up the tag as per your requirement.

CHALLENGES :

1. **Manual Toll Collection Causing Congestion:** Manual toll booths often lead to traffic jams and delays as vehicles queue up to pay, significantly slowing down the flow of traffic, especially during peak hours.
2. **Inaccurate Toll Calculations:** Errors in manual toll calculation or malfunctioning electronic systems can result in overcharging or undercharging, leading to disputes and revenue losses.
3. **Privacy and Security Concerns:** The collection and storage of personal data for toll transactions raise privacy and security issues, with potential risks of data breaches and misuse.
4. **Integration Issues with Existing Systems:** New toll collection technologies often face compatibility challenges with existing infrastructure, requiring costly upgrades and complex implementation processes.
5. **GPS Signal Strength and Accuracy:** Reliance on GPS for toll calculations can be problematic in areas with weak signal strength or interference, leading to inaccurate toll assessments and billing discrepancies.



1. OVERVIEW & PROPOSED SOLUTION

OVERVIEW:

This project simulates a system to calculate toll charges based on vehicle locations using GPS data. It is designed for end users managing toll data, such as vehicle details, available balances, and last logged GPS data. The project uses Python and MySQL, with Flask for handling requests (GET and POST), Geopy for distance calculations, Folium for updating car balance details, and MySQL Connector for database interactions. An interactive web UI makes it easy for government employees to use.

PROPOSED SOLUTION:

To address the inefficiencies associated with traditional toll collection methods, we propose a GPS-based toll simulation system developed using Python. This system leverages modern technologies to automate toll calculation and collection, enhancing the overall efficiency of transportation systems.

The system features a user-friendly frontend interface built with HTML, CSS, and JavaScript, allowing users to register their vehicles, log in to their accounts, input GPS data, and view detailed route and toll information. The backend, developed using Flask, processes incoming requests, performs calculations, and interacts with the MySQL database, ensuring efficient and secure handling of data.

Folium is used for interactive mapping, allowing users to visualize vehicle routes in real-time. The system accepts GPS data from vehicles, processes it, and stores it in the database. Geofencing is implemented to define specific areas around entry and exit points, and the Geopy library calculates the total distance traveled within these geofenced areas. This distance is then used to determine the applicable toll fees, which are automatically deducted from the user's balance.

The system includes secure user authentication, session management, and password reset functionalities to protect user data and ensure authorized access. Detailed transaction records provide transparency and accountability, allowing users to view their toll deductions and balance updates.

This integrated solution provides a seamless experience for users by automating toll calculations, offering real-time route visualizations, and ensuring secure management of user data and transactions.

By implementing this GPS-based toll simulation system, we can significantly reduce processing time at toll plazas, eliminate long queues and traffic congestion, and provide a seamless experience for drivers. The system's accuracy, efficiency, and scalability make it a robust solution for modernizing toll collection processes and improving the efficiency of transportation networks.

2. COMPONENTS & FEATURES OFFERED

COMPONENTS:

The key components and functionalities of the proposed solution are outlined below:

1. FRONTEND INTERFACE:

- The system features a user-friendly frontend interface built with HTML, CSS, and JavaScript.
- This interface allows users to register their vehicles, log in to their accounts, input GPS data, and view detailed route and toll information.
- The intuitive design ensures a smooth user experience.

2. BACKEND PROCESSING:

- The backend is developed using Flask, a lightweight and powerful web framework for Python.
- Flask handles incoming requests, processes data, performs necessary calculations, and interacts with the database.
- It ensures efficient and secure processing of user data and GPS information.

3. DATABASE MANAGEMENT:

- MySQL is employed for storing and managing car information, GPS data, and user details.
- The database is designed to handle large volumes of data efficiently, providing quick access and ensuring data integrity.
- It plays a crucial role in maintaining the system’s reliability.

4. INTERACTIVE MAPPING:

- Folium, a powerful library for creating interactive maps, is used to visualize vehicle routes in real-time.
- Users can see the paths their vehicles have traveled, along with entry and exit points, on an interactive map.
- This visualization aids in understanding and verifying the calculated tolls.

5. GPS DATA HANDLING:

- The system accepts GPS data from vehicles, processes it, and stores it in the database.

- This data includes latitude, longitude, and timestamp, which are essential for plotting routes and calculating distances.
- The handling of GPS data is efficient, ensuring accurate tracking and recording.

6. GEOFENCING AND DISTANCE CALCULATION:

- Geofencing is implemented to define specific areas around entry and exit points.
- Using the Geopy library, the system calculates the total distance traveled by a vehicle within these geofenced areas.
- This distance is then used to determine the applicable toll fees, ensuring precision in toll calculation.

7. AUTOMATED TOLL CALCULATION:

- Based on the calculated distance, the system computes the toll amount automatically.
- A predefined rate per kilometer is applied to the distance traveled within the geofence.
- The calculated toll is then deducted from the user's balance, streamlining the toll collection process.

8. USER AUTHENTICATION AND SECURITY:

- Secure user authentication mechanisms are implemented to protect user data and ensure authorized access to the system.
- Features like session management and password reset functionalities enhance the security and usability of the system.

9. TRANSACTION MANAGEMENT:

- The system maintains a detailed record of all transactions, including toll deductions and balance updates.
- Users can view their transaction history, providing transparency and accountability.

FEATURES:

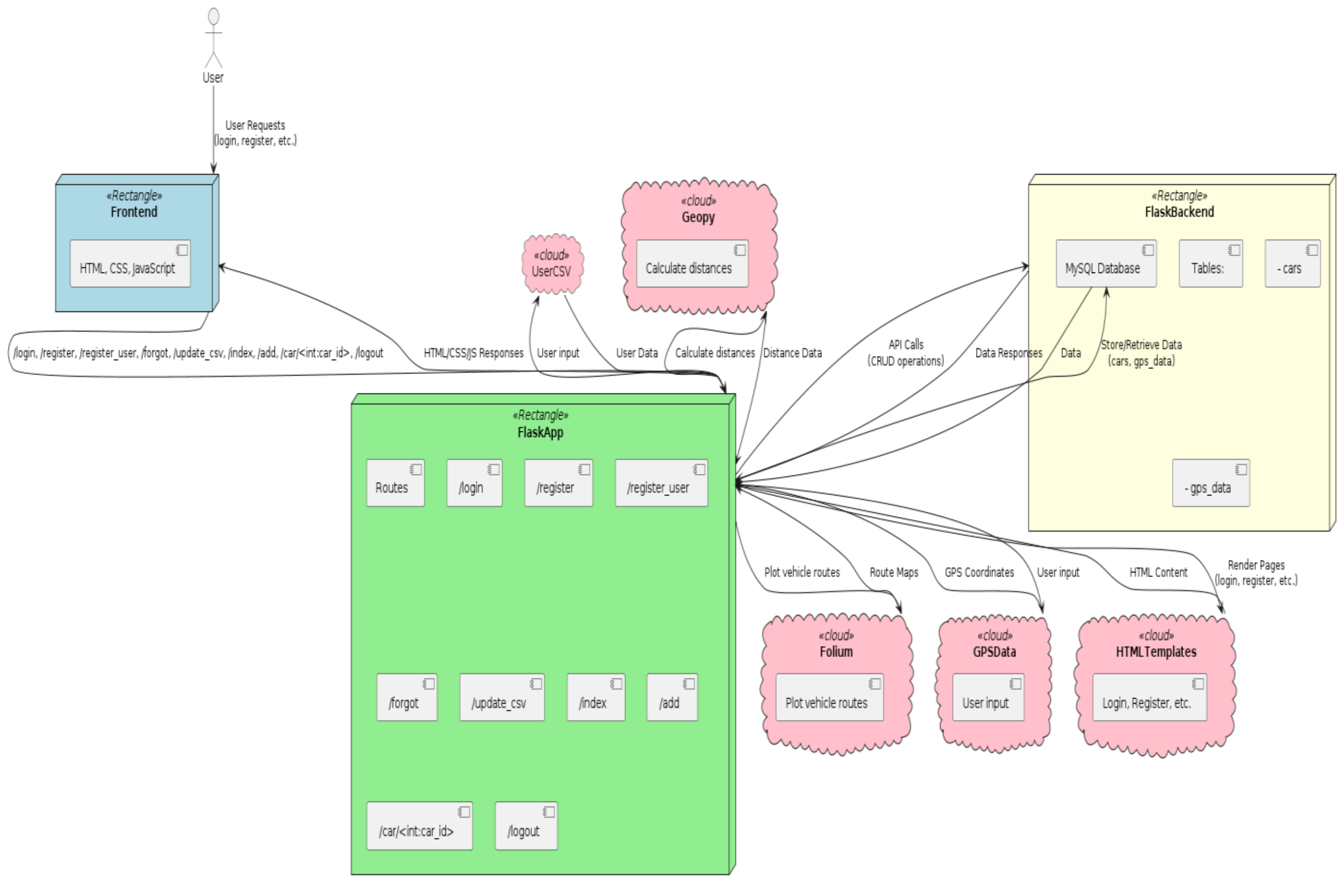
- **CAR REGISTRATION AND BALANCE MANAGEMENT:** Users can register their vehicles and manage their balance, which is essential for toll calculations.
- **GPS DATA HANDLING AND STORAGE:** The system accepts GPS data input, processes it, and stores it in the database for subsequent use in route plotting and toll calculations.
- **REAL-TIME ROUTE PLOTTING:** Folium maps provide real-time plotting of routes based on the GPS data, giving users a clear visual representation of their journey.
- **DISTANCE CALCULATION WITHIN GEOFENCED AREAS:** The system calculates the distance travelled within specified geofenced areas using the Geopy library, ensuring accurate toll charges.
- **AUTOMATED TOLL CALCULATION AND BALANCE DEDUCTION:** Based on the calculated distance, the system automatically computes the toll and deducts the appropriate amount from the user's balance.
- **USER AUTHENTICATION AND SESSION MANAGEMENT:** Secure login and session management functionalities protect user data and ensure only authorized access to the system.
- **PASSWORD RESET FUNCTIONALITY:** Users have the ability to reset their passwords, enhancing the security and user-friendliness of the system.

```
require File.expand_path( [illegible]
# Prevent database truncation [illegible]
abort("The Rails environment [illegible]
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = [illegible]
Category.delete_all; Category [illegible]
Shoulda::Matchers.configure [illegible]
  config.integrate do [illegible]
    with.test_framework [illegible]
    with.library :rails
  end
end

# Add additional requires [illegible]

# Requires supporting [illegible]
# spec/support/ and its [illegible]
# run as spec files [illegible]
# in _spec.rb will [illegible]
# run twice. It is [illegible]
# end with _spec.rb. [illegible]
```

3. SIMULATION WORKFLOW

1. **FRONTEND**:
    - **HTML, CSS, JavaScript**: The frontend interacts with the user and sends data to the Flask application.
2. **FLASK APP**:
    - The Flask application handles various routes such as /login, /register, /index, etc.
    - **Routes**: Defined to manage user interactions, register vehicles, add GPS data, and more.
3. **FLASK BACKEND**:
    - **MySQL Database**: The backend interacts with the MySQL database to store and retrieve car and GPS data.
        - **Tables**:
            - **cars**: Stores car details including car number, balance, and transaction history.
            - **gps_data**: Stores GPS data for each car including timestamps, latitude, and longitude.
4. **EXTERNAL LIBRARIES**:
    - **Geopy**: Used to calculate distances between GPS coordinates.
    - **Folium**: Used to create interactive maps and plot vehicle routes.
5. **STATIC FILES**:
    - **GPS Data**: Retrieved and stored in the gps_data table.
    - **User CSV**: The login.csv file stores user details for login and registration.
    - **HTML Templates**: Templates for pages like login, register, forgot password, index, and car details.

SYSTEM ARCHITECTURE:



FLOW:

1. **USER INTERACTION**: Users interact with the frontend (HTML forms) to register, login, add GPS data, and view car details.
2. **REQUEST HANDLING**: The Flask application processes these requests and interacts with the MySQL database to store and retrieve data.
3. **DATA PROCESSING**:
    - **Geopy** calculates distances based on GPS data.
    - **Folium** plots vehicle routes and displays them in the browser.
4. **RESPONSE**: The application renders HTML templates to display the requested information, such as car details and maps.

4. IMPLEMENTATION (WORKING CODE)

FRONT END:

INDEX PAGE.html:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>GPS Toll System</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      background-color: #f5f5dc;
      color: #333;
      background: url('https://i.imghippo.com/files/rtJv71720352316.jpg');
      background-size: cover;
      background-repeat: no-repeat;
    }

    header {
      width: 100%;
      max-width: 900px;
      margin: auto;
      display: flex;
      justify-content: space-between;
      align-items: center;
      background-color: #A0522D;
      padding: 10px 20px;
      border-radius: 8px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    header h1 {
      margin: 0;
      color: #fff;
    }
```

```
header a {
    color: #fff;
    text-decoration: none;
    background-color: #8B4513;
    padding: 10px 20px;
    border-radius: 5px;
}

.container {
    width: 100%;
    max-width: 900px;
    margin: auto;
    margin-top: 20px;
    display: flex;
    flex-direction: column;
    align-items: center;
}

.welcome-text {
    margin-top: 10px;
    font-size: 1.2em;
    color: #A0522D;
}

.form-container {
    width: 100%;
    display: flex;
    justify-content: space-between;
    margin-bottom: 20px;
}

form {
    flex: 1;
    background-color: #c1e3f0;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    margin-right: 10px;
}

form:last-child {
    margin-right: 0;
}

form h2 {
    margin-top: 0;
}

form label {
    display: block;
    margin-bottom: 5px;
}

form input {
    margin-bottom: 10px;
    padding: 8px;
    width: calc(100% - 18px);
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

form button {
    padding: 10px 20px;
    background-color: #8B4513;
    color: white;
    border: none;
    cursor: pointer;
    border-radius: 4px;
    transition: background-color 0.3s;
}

form button:hover {
    background-color: #5C3317;
}

.result-container {
    width: 100%;
    max-height: 300px;
    overflow-y: auto;
    background-color: #addaea;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 8px;
```

```
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }

        .result-container h2 {
            margin-top: 0;
        }

        .car-container {
            border: 1px solid #ddd;
            padding: 10px;
            margin-bottom: 10px;
            border-radius: 8px;
            background-color: #fafafa;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            transition: background-color 0.3s;
        }

        .car-container:hover {
            background-color: #f0f0f0;
        }

        .car-container h2 {
            margin: 0;
            font-size: 1.2em;
            color: #A0522D;
        }

        .car-container p {
            margin: 5px 0;
            color: #555;
        }

        #map {
            height: 400px;
            margin-top: 20px;
            width: 100%;
        }
    </style>
</head>

<body>
    <header>
        <h1>GPS Toll System</h1>
    <a href="/logout">Logout</a>
  </header>

  <div class="container">
    <div class="welcome-text">Welcome {{ name }}</div>
    <div class="form-container">
      <form method="post" action="/add">
        <h2>Add Car Information</h2>
        <label for="car_id">Car ID:</label>
        <input type="text" id="car_id" name="car_id" required>
        <label for="gps_data">GPS Data (lat,lon;lat,lon;...):</label>
        <input type="text" id="gps_data" name="gps_data" required>
        <button type="submit">Submit</button>
      </form>
      <form method="post" action="/register">
        <h2>Register Car Information</h2>
        <label for="car_number">Car Number:</label>
        <input type="text" id="car_number" name="car_number" required>
        <label for="balance">Balance:</label>
        <input type="text" id="balance" name="balance" required>
        <button type="submit">Register</button>
      </form>
    </div>

    <div class="result-container" id="result-container">
      <h2>Transaction History</h2>
      {% for car in cars %}
      <div class="car-container">
        <h2><a href="{{ url_for('car_detail', car_id=car[0]) }}">Car ID: {{ car[0] }}</a></h2>
        <p>Car Number: {{ car[1] }}</p>
        <p>Balance: {{ car[2] }} currency units</p>
      </div>
      {% endfor %}
    </div>
  </div>

  <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
  <script>
    function scrollToBottom() {
      var container = document.getElementById('result-container');
      container.scrollTop = container.scrollHeight;
    }
```

```
        // Scroll to bottom when the page loads
        window.onload = function() {
            scrollToBottom();
        };

        // Optional: Scroll to bottom when new content is added
        // This assumes you're using some client-side update mechanism
        // You might need to adapt this to your specific case
        // Example:
        // document.getElementById('result-container').addEventListener('DOMNodeInserted', scrollToBottom);
    </script>
</body>

</html>
```

LOGIN PAGE.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login - GPS Based Toll System</title>
    <style>
        @import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

body {
    font-family: 'Roboto', sans-serif;
    background: url('https://www.intel.com/content/dam/www/central-libraries/us/en/images/2022-10/404-error-page-
background-image-thumb.jpg.rendition.intel.web.864.486.jpg') no-repeat center center fixed;
    background-size: cover;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    color: #333;
}

.container {
    max-width: 400px;
    padding: 40px 50px;
    background-color: rgba(255, 255, 255, 0.95);
    border-radius: 15px;
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.3);
    text-align: center;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}

.container:hover {
    transform: translateY(-5px);
    box-shadow: 0 15px 30px rgba(0, 0, 0, 0.4);
}

.container h1 {
    color: #0071c5;
    margin-bottom: 20px;
    font-weight: 700;
}

.container img {
    width: 100px;
    margin-bottom: 20px;
}

.input-group {
    position: relative;
    margin: 20px 0;
}

.input-group input {
    width: 100%;
    padding: 15px;
    padding-left: 40px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 10px;
    box-sizing: border-box;
    font-size: 16px;
    transition: border 0.3s ease;
}

.input-group input:focus {
    border-color: #0071c5;
    outline: none;
}

.input-group .icon {
    position: absolute;
    top: 50%;
    left: 10px;
    transform: translateY(-50%);
    width: 20px;
    height: 20px;
    fill: #999;
    transition: fill 0.3s ease;
}

.input-group input:focus + .icon {
    fill: #0071c5;
}

input[type="submit"] {
    width: 100%;
    padding: 15px;
    background-color: #0071c5;
    color: white;
    border: none;
    border-radius: 10px;
    cursor: pointer;
    font-size: 18px;
    transition: background-color 0.3s ease, transform 0.3s ease, box-shadow 0.3s ease;
}

input[type="submit"]:hover {
    background-color: #005b9e;
    transform: translateY(-2px);
    box-shadow: 0 5px 10px rgba(0, 0, 0, 0.2);
}

.footer {
    margin-top: 20px;
    font-size: 14px;
    color: #666;
}

.button-group {
    display: flex;
    justify-content: space-between;
    margin-top: 20px;
}

.button-group a {
    display: inline-block;
    width: 48%;
    padding: 10px;
    text-align: center;
    background-color: #31c5dc;
    color: white;
    border-radius: 10px;
    text-decoration: none;
    font-size: 16px;
    transition: background-color 0.3s ease, transform 0.3s ease, box-shadow 0.3s ease;
}

.button-group a:hover {
    background-color: #dd31dd;
    transform: translateY(-2px);
    box-shadow: 0 5px 10px rgba(0, 0, 0, 0.2);
}

.button-group a.forgot-password {
    background-color: #f44336; /* Red color for "Forgot Password" button */
    padding: 12px; /* Increased padding for "Forgot Password" button */
}

.button-group a.new-registration {
    background-color: #4caf50; /* Green color for "New User Registration" button */
    padding: 12px; /* Increased padding for "New User Registration" button */
}

.button-group a.forgot-password:hover {
    background-color: #c62828; /* Darker red color on hover for "Forgot Password" button */
}

.button-group a.new-registration:hover {
    background-color: #388e3c; /* Darker green color on hover for "New User Registration" button */
}
    </style>
</head>
```

```
<body>
  <div class="container">

    <h1>Login</h1>
    <form id="loginForm" onsubmit="return validateForm()">
      <div class="input-group">
        <input type="text" id="username" name="username" placeholder="Enter username">
        <svg class="icon" viewBox="0 0 24 24">
          <path d="M12 12c2.21 0 4-1.79 4-4s-1.79-4-4-4-4 1.79-4 4 1.79 4 4 4zm0 2c-2.67 0-8 1.34-8 4v2h16v-
2c0-2.66-5.33-4-8-4z"></path>
        </svg>
      </div>
      <div class="input-group">
        <input type="password" id="password" name="password" placeholder="Enter password">
        <svg class="icon" viewBox="0 0 24 24">
          <path d="M12 2C9.79 2 8 3.79 8 6v2H6v12h12V8h-2V6c0-2.21-1.79-4-4-4zm0 2c1.1 0 2 .9 2 2v2h-
4V6c0-1.1.9-2 2-2zm-6 8v8h12v-8H6z"></path>
        </svg>
      </div>
      <input type="submit" value="Login">
    </form>
    <div class="button-group">
      <a href='/forgot'>Forgot Password?</a>
      <a href='/register'>New User Registration</a>
    </div>
    <div class="footer">
      &copy; 2024 GPS Based Toll System. All rights reserved.
    </div>
  </div>

  <script>
    document.addEventListener("DOMContentLoaded", function() {
      document.getElementById("loginForm").addEventListener("submit", function(event) {
        event.preventDefault(); // Prevent form submission
        validateForm();
      });
    });

    function validateForm() {
      var username = document.getElementById("username").value;
      var password = document.getElementById("password").value;

      if (username === "" || password === "") {
            alert("Please enter both username and password.");
            return;
        }

        fetch("/static/login.csv")
        .then(response => response.text())
        .then(data => {
            // Parse the CSV data
            const rows = data.trim().split('\n');
            const users = rows.map(row => row.split(','));

            // Validate the username and password
            validateUserCredentials(username, password, users);
        })
        .catch(error => {
            console.error('Error reading CSV file:', error);
        });
    }

    function validateUserCredentials(username, password, users) {
        const isValidUser = users.some(user => {
    const [storedUsername, storedPassword] = user;
    return storedUsername.trim() === username.trim() && storedPassword.trim() === password.trim();
  });
        if (isValidUser) {
            window.location.href = "/index?name=" + encodeURIComponent(username);
        } else {
            alert("Incorrect username or password.");
        }
    }
  </script>
</body>
</html>
```

NEW USER REGISTER PAGE.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Register</title>
  <style>
```

```
    body {
      font-family: Arial, sans-serif;
      background: #f5f5f5;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }
    .container {
      background: white;
      padding: 20px;
      border-radius: 15px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      text-align: center;
      max-width: 400px;
      width: 100%;
    }
    h2 {
      margin-bottom: 20px;
    }
    button {
      padding: 10px 20px;
      background-color: #007bff;
      color: white;
      border: none;
      border-radius: 8px;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
  </style>
  <script>
    async function registerUser(event) {
      event.preventDefault();

      const formData = new FormData(event.target);
      const formDataObject = Object.fromEntries(formData.entries());

      const response = await fetch('/register_user', {
        method: 'POST',
        headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(formDataObject)
        });

        if (response.ok) {
            alert('Registration successful!');
            window.location.href = '/login.html';
        } else {
            alert('Registration failed!');
        }
    }
  </script>
</head>
<body>
  <div class="container">
  <h2>Register</h2>
  <form onsubmit="registerUser(event)">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <label for="security_question">Security Question:</label>
    <input type="text" id="security_question" name="security_question" required>
    <br>
    <label for="security_answer">Security Answer:</label>
    <input type="text" id="security_answer" name="security_answer" required>
    <br>
    <button type="submit">Register</button>
  </form>
  </div>
</body>
</html>
```

FORGOT PASSWORD PAGE.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Forgot Password</title>
<style>
  body {
    font-family: Arial, sans-serif;
    background: #f5f5f5;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
  }
  .container {
    background: white;
    padding: 20px;
    border-radius: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
    max-width: 400px;
    width: 100%;
  }
  h1 {
    margin-bottom: 20px;
  }
  .input-group {
    margin-bottom: 15px;
  }
  .input-group label {
    display: block;
    text-align: left;
    margin-bottom: 5px;
  }
  .input-group input {
    width: calc(100% - 20px);
    padding: 10px;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 4px;
  }
  .input-group button {
    padding: 10px 20px;
    background-color: #007bff;
```

```
            color: white;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
        .input-group button:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Forgot Password</h1>
        <form id="forgotPasswordForm" onsubmit="return validateForm()">
            <div id="usernameDiv" class="input-group">
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" required>
                <button type="button" onclick="getSecurityQuestion()">Get Security Question</button>
            </div>
            <div id="securityQuestionDiv" class="input-group" style="display: none;">
                <label for="securityQuestion">Security Question:</label>
                <input type="text" id="securityQuestion" name="securityQuestion" readonly>
                <label for="securityAnswer">Answer:</label>
                <input type="text" id="securityAnswer" name="securityAnswer" required>
                <button type="button" onclick="validateSecurityAnswer()">Validate Answer</button>
            </div>
            <div id="newPasswordDiv" class="input-group" style="display: none;">
                <label for="newPassword">New Password:</label>
                <input type="password" id="newPassword" name="newPassword">
                <button type="submit">Reset Password</button>
            </div>
        </form>
    </div>

    <script>
        function getSecurityQuestion() {
            var username = document.getElementById("username").value;
            fetch("/static/login.csv")
            .then(response => response.text())
            .then(data => {
                const rows = data.trim().split('\n').slice(1); // Remove header row
                const users = rows.map(row => row.split(','));
        const user = users.find(user => user[0] === username);
        if (user) {
          document.getElementById("securityQuestion").value = user[2];
          document.getElementById("usernameDiv").style.display = "none";
          document.getElementById("securityQuestionDiv").style.display = "block";
        } else {
          alert("Username not found.");
        }
      })
      .catch(error => {
        console.error('Error reading CSV file:', error);
      });
    }

    function validateSecurityAnswer() {
      var username = document.getElementById("username").value;
      var securityAnswer = document.getElementById("securityAnswer").value;

      fetch("/static/login.csv")
      .then(response => response.text())
      .then(data => {
        const rows = data.trim().split('\n').slice(1); // Remove header row
        const users = rows.map(row => row.split(','));

        const user = users.find(user => user[0] === username);
        if (user && user[3].trim() === securityAnswer.trim()) {
          document.getElementById("securityQuestionDiv").style.display = "none";
          document.getElementById("newPasswordDiv").style.display = "block";
          document.getElementById("newPassword").required = true;
        } else {
          alert("Incorrect answer to the security question.");
        }
      })
      .catch(error => {
        console.error('Error reading CSV file:', error);
      });
    }

    function validateForm() {
      var newPasswordDiv = document.getElementById("newPasswordDiv");
      if (newPasswordDiv.style.display !== "none") {
        document.getElementById("newPassword").required = true;
      }
    return true;
}

document.getElementById("forgotPasswordForm").addEventListener("submit", function(event) {
    event.preventDefault();

    var username = document.getElementById("username").value;
    var newPassword = document.getElementById("newPassword").value;

    fetch("/static/login.csv")
    .then(response => response.text())
    .then(data => {
        let rows = data.trim().split('\n');
        let headers = rows[0].split(',');
        let users = rows.slice(1).map(row => row.split(','));

        users = users.map(user => {
            if (user[0] === username) {
                user[1] = newPassword; // Update password
            }
            return user;
        });

        const updatedCsv = [headers.join(','), ...users.map(user => user.join(','))].join('\n');

        fetch('/update_csv', {
            method: 'POST',
            headers: {
                'Content-Type': 'text/csv',
            },
            body: updatedCsv,
        })
        .then(response => {
            if (response.ok) {
                alert('Password updated successfully.');
                window.location.href = '/login.html';
            } else {
                alert('Error updating password.');
            }
        })
        .catch(error => {
            console.error('Error updating CSV file:', error);
            });
        })
        .catch(error => {
            console.error('Error reading CSV file:', error);
        });
    });
  </script>
</body>
</html>
```

CAR DETAILS DISPLAY.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Car Details</title>
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
  <style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
      background-color: #f7f7f7;
      display: flex;
      flex-direction: column;
      align-items: center;
    }
    h1 {
      color: #333;
      text-align: center;
    }
    .car-details {
      margin-bottom: 20px;
      padding: 20px;
      border: 2px solid #007BFF;
      border-radius: 10px;
      background-color: #fff;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      width: 100%;
      max-width: 400px;
    }
```

```
        .car-details p {
            margin: 5px 0;
            font-weight: bold;
        }
        #map-container {
            width: 100%;
            max-width: 800px;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <h1>Car Details</h1>
    <div class="car-details">
        <p>Car ID: {{ car[0] }}</p>
        <p>Car Number: {{ car[1] }}</p>
        <p>Balance: {{ car[2] }} currency units</p>
    </div>
    {% if map_html %}
    <div id="map-container">
        <h3>Last stopped region</h3>
        {{ map_html|safe }}
    </div>
    {% else %}
    <p>No GPS data available for this car.</p>
    {% endif %}
    <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
</body>
</html>
```

BACK END:

FLASK DEPLOYMENT:

```
from flask import Flask, render_template, request,redirect, url_for,jsonify
from datetime import datetime, timedelta
import mysql.connector
from geopy.distance import geodesic
import folium
import math
import csv
app = Flask(__name__)
```

```
# MySQL database configuration
db_config = {
    'host': 'Anik005.mysql.pythonanywhere-services.com',
    'user': 'Anik005',
    'password': 'ani12345',
    'database': 'Anik005$intern'
}

# Geofence parameters (entry and exit points, and radius in kilometers)
entry_point = (37.7749, -122.4194)  # Example entry point (latitude, longitude)
exit_point = (37.7849, -122.4094)   # Example exit point (latitude, longitude)
geofence_radius_km = 2.0            # Radius in kilometers beyond entry/exit points

# Function to connect to the MySQL database
def connect_db():
    return mysql.connector.connect(**db_config)

@app.route('/', methods=['POST','GET'])
def login():
     return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        car_number = request.form['car_number']
        balance = float(request.form['balance'])
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("INSERT INTO cars (car_number, balance) VALUES (%s, %s)", (car_number, balance))
        conn.commit()
        cursor.execute("SELECT car_id, car_number, balance, transaction FROM cars")
        cars = cursor.fetchall()
        conn.commit()
        cursor.close()
        conn.close()
        return render_template('index.html', cars=cars)
    else:
        return render_template('register.html')
@app.route('/register_user', methods=['POST'])
def register_user():
    if request.method == 'POST':
        data = request.get_json()
        username = data.get('username')
        password = data.get('password')
        security_question = data.get('security_question')
        security_answer = data.get('security_answer')

        # Append the new user's data to the CSV file
        with open('static/login.csv', 'a', newline='') as csvfile:
            csvwriter = csv.writer(csvfile)
            csvwriter.writerow([username, password, security_question, security_answer])

        return jsonify({'message': 'Registration successful'}), 200
@app.route('/forgot', methods=['GET', 'POST'])
def forgot_password():
    if request.method == 'POST':
        username = request.form['username']
        # Implement the logic to reset the password
        # This could involve querying the database, sending an email, etc.
        return 'Password reset instructions sent to your email'
    return render_template('forgot.html')
@app.route('/update_csv', methods=['POST'])
def update_csv():
    csv_data = request.get_data(as_text=True)
    with open('static/login.csv', 'w', newline='') as csvfile:
        csvfile.write(csv_data)
    return 'CSV file updated successfully', 200
@app.route('/index',methods=['POST','GET'])
def root():
    try:
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("SELECT car_id, car_number, balance,transaction FROM cars")
        cars = cursor.fetchall()
    except mysql.connector.Error as e:
        cars = []
        print(f"Database error: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()

    return render_template('index.html',  cars=cars)
```

```
# Flask route for handling requests
@app.route('/add', methods=['GET', 'POST'])

def index():
    car_results = []
    if request.method == 'POST':
        car_id = request.form.get('car_id')
        gps_data_str = request.form.get('gps_data')

        if car_id and gps_data_str:
            gps_data_list = gps_data_str.split(';')
            gps_data = []
            for i, coord in enumerate(gps_data_list):
                if coord.strip():  # Check if coordinate string is not empty or whitespace
                    try:
                        lat, lon = map(float, coord.split(','))
                        timestamp = datetime.now() + timedelta(minutes=i)
                        gps_data.append((car_id, timestamp, lat, lon))
                    except ValueError:
                        return "Invalid coordinate format", 400

            try:
                conn = connect_db()
                cursor = conn.cursor()

                # Insert GPS data into gps_data table
                insert_query = "INSERT INTO gps_data (car_id, timestamp, latitude, longitude) VALUES (%s, %s, %s,
%s)"
                cursor.executemany(insert_query, gps_data)
                conn.commit()

                # Plotting vehicle route using Folium
                plot_html = plot_route(gps_data, entry_point, exit_point)

                # Calculating total distance traveled with geofencing
                total_distance, total_distance_within_geofence = calculate_distance_traveled(gps_data, entry_point,
exit_point, geofence_radius_km)
                rate_per_km = 0.10  # Example rate per kilometer
                total_toll = total_distance * rate_per_km

                # Retrieve car balance and update
                select_query = "SELECT car_id, car_number, balance, transaction FROM cars WHERE car_id = %s"
                cursor.execute(select_query, (car_id,))
                result = cursor.fetchone()

                if result:
                    car_results.append({
                        'car_id': result[0],
                        'car_number': result[1],
                        'balance': result[2],
                        'transaction': result[3],
                        'total_distance': total_distance_within_geofence,  # Displaying distance within geofence
                        'total_toll': total_toll,
                        'new_balance': result[2] - total_toll,
                        'plot_html': plot_html
                    })

                    # Update balance after deducting toll
                    update_query = "UPDATE cars SET balance = %s WHERE car_id = %s"
                    cursor.execute(update_query, (result[2] - total_toll, car_id))
                    conn.commit()

                else:
                    return "Car ID not found", 404

            except mysql.connector.Error as e:
                conn.rollback()
                return f"Database error: {str(e)}", 500
            finally:
                if 'cursor' in locals():
                    cursor.close()
                if 'conn' in locals():
                    conn.close()

        else:
            return "Invalid input", 400

    # Fetch all cars for transaction history
    try:
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute("SELECT car_id, car_number, balance, transaction FROM cars")
        cars = cursor.fetchall()
    except mysql.connector.Error as e:
        cars = []
        print(f"Database error: {str(e)}")
    finally:
        if 'cursor' in locals():
            cursor.close()
        if 'conn' in locals():
            conn.close()
    name = request.args.get('name')
    return render_template('index.html',name=name, car_results=car_results, cars=cars)

# Function to fetch GPS data for a specific car from the database
def fetch_gps_data(car_id):
    try:
        conn = connect_db()
        cursor = conn.cursor()

        select_query = "SELECT timestamp, latitude, longitude FROM gps_data WHERE car_id = %s"
        cursor.execute(select_query, (car_id,))
        gps_data = cursor.fetchall()

        cursor.close()
        conn.close()

        return gps_data

    except mysql.connector.Error as e:
        print(f"Error fetching GPS data: {e}")
        return []

# Flask route for displaying car details and Folium map
@app.route('/car/<int:car_id>')
def car_detail(car_id):
    conn = mysql.connector.connect(**db_config)
    cursor = conn.cursor()

    cursor.execute("SELECT car_id, car_number, balance FROM cars WHERE car_id = %s", (car_id,))
    car = cursor.fetchone()

    cursor.execute("SELECT latitude, longitude FROM gps_data WHERE car_id = %s ORDER BY id DESC LIMIT
1", (car_id,))
    gps_data = cursor.fetchone()

    cursor.close()
    conn.close()
    if gps_data:
        map_html = create_map(gps_data[0], gps_data[1])
    else:
        map_html = None

    return render_template('car_detail.html', car=car, map_html=map_html)

def create_map(lat, lon):
    m = folium.Map(location=[lat, lon], zoom_start=15)
    folium.Marker([lat, lon]).add_to(m)
    return m._repr_html_()

# Function to plot the vehicle route using Folium
def plot_route(gps_data, entry_point, exit_point):
    try:
        map_center = [(entry_point[0] + exit_point[0]) / 2, (entry_point[1] + exit_point[1]) / 2]
        m = folium.Map(location=map_center, zoom_start=13)

        # Adding entry and exit points to the map
        folium.Marker(entry_point, popup='Entry Point', icon=folium.Icon(color='red')).add_to(m)
        folium.Marker(exit_point, popup='Exit Point', icon=folium.Icon(color='red')).add_to(m)

        # Adding vehicle path to the map if gps_data is not empty
        if gps_data:
            coords = [(point[2], point[3]) for point in gps_data if len(point) >= 4]  # Check tuple length
            folium.PolyLine(coords, color='blue', weight=2.5, opacity=1).add_to(m)

        # Adding geofence boundary to the map
        geofence_coords = get_geofence_boundary(entry_point, exit_point, geofence_radius_km)
        folium.Polygon(locations=geofence_coords, color='green', fill=True, fill_opacity=0.2).add_to(m)

        # Save map to HTML and return the HTML string
        map_html = m._repr_html_()
        return map_html

    except Exception as e:
        print(f"Error in plot_route: {str(e)}")
        return None


def plot_route1(gps_data, entry_point, exit_point):
    try:
        map_center = [(entry_point[0] + exit_point[0]) / 2, (entry_point[1] + exit_point[1]) / 2]
        m = folium.Map(location=map_center, zoom_start=13)

        # Adding entry and exit points to the map
        folium.Marker(entry_point, popup='Entry Point', icon=folium.Icon(color='red')).add_to(m)
        folium.Marker(exit_point, popup='Exit Point', icon=folium.Icon(color='red')).add_to(m)

        # Adding vehicle path to the map if gps_data is not empty
        if gps_data:
            coords = [(point[1], point[2]) for point in gps_data]  # Extract latitude and longitude
            folium.PolyLine(coords, color='blue', weight=2.5, opacity=1).add_to(m)

        # Adding geofence boundary to the map
        geofence_coords = get_geofence_boundary1(entry_point, exit_point, geofence_radius_km)
        folium.Polygon(locations=geofence_coords, color='green', fill=True, fill_opacity=0.2).add_to(m)

        # Save map to HTML and return the HTML string
        map_html = m._repr_html_()
        return map_html

    except Exception as e:
        print(f"Error in plot_route: {str(e)}")
        return None


# Function to calculate the total distance traveled with geofencing
def calculate_distance_traveled(gps_data, entry_point, exit_point, geofence_radius_km):
    total_distance = 0.0
    total_distance_within_geofence = 0.0
    for i in range(1, len(gps_data)):
        point1 = (gps_data[i-1][2], gps_data[i-1][3])  # latitude, longitude of previous point
        point2 = (gps_data[i][2], gps_data[i][3])      # latitude, longitude of current point
        distance = geodesic(point1, point2).kilometers

        # Check if both points are within the geofence boundary
        if is_within_geofence(point1, entry_point, exit_point, geofence_radius_km) and is_within_geofence(point2,
entry_point, exit_point, geofence_radius_km):
            total_distance_within_geofence += distance
        elif is_within_geofence(point1, entry_point, exit_point, geofence_radius_km) or is_within_geofence(point2,
entry_point, exit_point, geofence_radius_km):
            # Calculate distance within geofence for segments crossing the geofence boundary
            total_distance_within_geofence += distance * 0.5

        total_distance += distance
    return total_distance, total_distance_within_geofence

# Function to check if a point is within the geofence boundary
def is_within_geofence(point, entry_point, exit_point, radius_km):
    # Assuming a rectangular geofence for simplicity
    min_lat = min(entry_point[0], exit_point[0]) - (radius_km / 111)  # 1 degree of latitude ~ 111 km
    max_lat = max(entry_point[0], exit_point[0]) + (radius_km / 111)
    min_lon = min(entry_point[1], exit_point[1]) - (radius_km / (111 * math.cos(math.radians(point[0]))))
    max_lon = max(entry_point[1], exit_point[1]) + (radius_km / (111 * math.cos(math.radians(point[0]))))

    return min_lat <= point[0] <= max_lat and min_lon <= point[1] <= max_lon

# Function to get the geofence boundary coordinates based on entry and exit points
def get_geofence_boundary(entry_point, exit_point, radius_km):
    # Calculate midpoint between entry and exit points
    mid_lat = (entry_point[0] + exit_point[0]) / 2
    mid_lon = (entry_point[1] + exit_point[1]) / 2

    # Calculate bearing from midpoint to entry point
    bearing = math.atan2(entry_point[1] - mid_lon, entry_point[0] - mid_lat)

    # Calculate geofence boundary coordinates
    geofence_coords = []
    for angle in range(0, 360, 30):  # Adjust angle increment based on desired resolution
        lat = mid_lat + (radius_km / 111) * math.cos(math.radians(angle))
        lon = mid_lon + (radius_km / (111 * math.cos(math.radians(mid_lat)))) * math.sin(math.radians(angle))
        geofence_coords.append((lat, lon))

    # Close the geofence polygon
    geofence_coords.append(geofence_coords[0])

    return geofence_coords


def calculate_distance_traveled1(gps_data, entry_point, exit_point, geofence_radius_km):
    total_distance = 0.0
    total_distance_within_geofence = 0.0
    for i in range(1, len(gps_data)):
        point1 = (gps_data[i-1][1], gps_data[i-1][2])  # latitude, longitude of previous point
        point2 = (gps_data[i][1], gps_data[i][2])      # latitude, longitude of current point
        distance = geodesic(point1, point2).kilometers
        # Check if both points are within the geofence boundary
        if is_within_geofence1(point1, entry_point, exit_point, geofence_radius_km) and is_within_geofence1(point2,
entry_point, exit_point, geofence_radius_km):
            total_distance_within_geofence += distance
        elif is_within_geofence1(point1, entry_point, exit_point, geofence_radius_km) or is_within_geofence1(point2,
entry_point, exit_point, geofence_radius_km):
            # Calculate distance within geofence for segments crossing the geofence boundary
            total_distance_within_geofence += distance * 0.5

        total_distance += distance

    return total_distance, total_distance_within_geofence

# Function to check if a point is within the geofence boundary
def is_within_geofence1(point, entry_point, exit_point, radius_km):
    # Assuming a rectangular geofence for simplicity
    min_lat = min(entry_point[0], exit_point[0]) - (radius_km / 111)  # 1 degree of latitude ~ 111 km
    max_lat = max(entry_point[0], exit_point[0]) + (radius_km / 111)
    min_lon = min(entry_point[1], exit_point[1]) - (radius_km / (111 * math.cos(math.radians(point[0]))))
    max_lon = max(entry_point[1], exit_point[1]) + (radius_km / (111 * math.cos(math.radians(point[0]))))

    return min_lat <= point[0] <= max_lat and min_lon <= point[1] <= max_lon

# Function to get the geofence boundary coordinates based on entry and exit points
def get_geofence_boundary1(entry_point, exit_point, radius_km):
    # Calculate midpoint between entry and exit points
    mid_lat = (entry_point[0] + exit_point[0]) / 2
    mid_lon = (entry_point[1] + exit_point[1]) / 2

    # Calculate bearing from midpoint to entry point
    bearing = math.atan2(entry_point[1] - mid_lon, entry_point[0] - mid_lat)

    # Calculate geofence boundary coordinates
    geofence_coords = []
    for angle in range(0, 360, 30):  # Adjust angle increment based on desired resolution
        lat = mid_lat + (radius_km / 111) * math.cos(math.radians(angle))
        lon = mid_lon + (radius_km / (111 * math.cos(math.radians(mid_lat)))) * math.sin(math.radians(angle))
        geofence_coords.append((lat, lon))

    # Close the geofence polygon
    geofence_coords.append(geofence_coords[0])

    return geofence_coords
```

```
@app.route('/logout')
def lo():
    return redirect(url_for('login'))
# Error handler for internal server errors
@app.errorhandler(500)
def internal_server_error(error):
   return "Internal Server Error. Please try again later.", 500

if __name__ == '__main__':
   app.run(debug=False, host='0.0.0.0', port=5007)
```

DATABASE SCHEMA:

```
CREATE TABLE cars (
    car_id INT AUTO_INCREMENT PRIMARY KEY,
    car_number VARCHAR(20) NOT NULL,
    balance FLOAT NOT NULL
);
CREATE TABLE gps_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    car_id INT,
    timestamp DATETIME,
    latitude FLOAT,
    longitude FLOAT,
    FOREIGN KEY (car_id) REFERENCES cars(car_id));
```

MODULARITY:

Here's a list of the different modules and aspects of the provided code:

ROUTE HANDLING

- **login**: Manages the login page rendering.
- **register**: Handles car registration.
- **register_user**: Manages user registration.

- **forgot_password , update_csv**: : Handles password reset functionality, Updates the CSV file with user data.
- **root**: Displays the main page with car details.
- **index**: Manages the addition of GPS data and displays results.
- **car_detail**: Displays detailed information and map for a specific car.
- **logout**: Handles user logout.

DATABASE OPERATIONS

- **connect_db**: Establishes a connection to the MySQL database.

DATA HANDLING AND PROCESSING

- **fetch_gps_data**: Fetches GPS data for a specific car.
- **plot_route**: Plots the vehicle route on a map using Folium.
- **calculate_distance_traveled**: Calculates the total distance traveled, including geofenced areas.

UTILITY FUNCTIONS

- **is_within_geofence**: Checks if a point is within the geofence boundary.
- **get_geofence_boundary**: Generates the geofence boundary coordinates.
- **create_map**: Creates a Folium map centered on a given location.

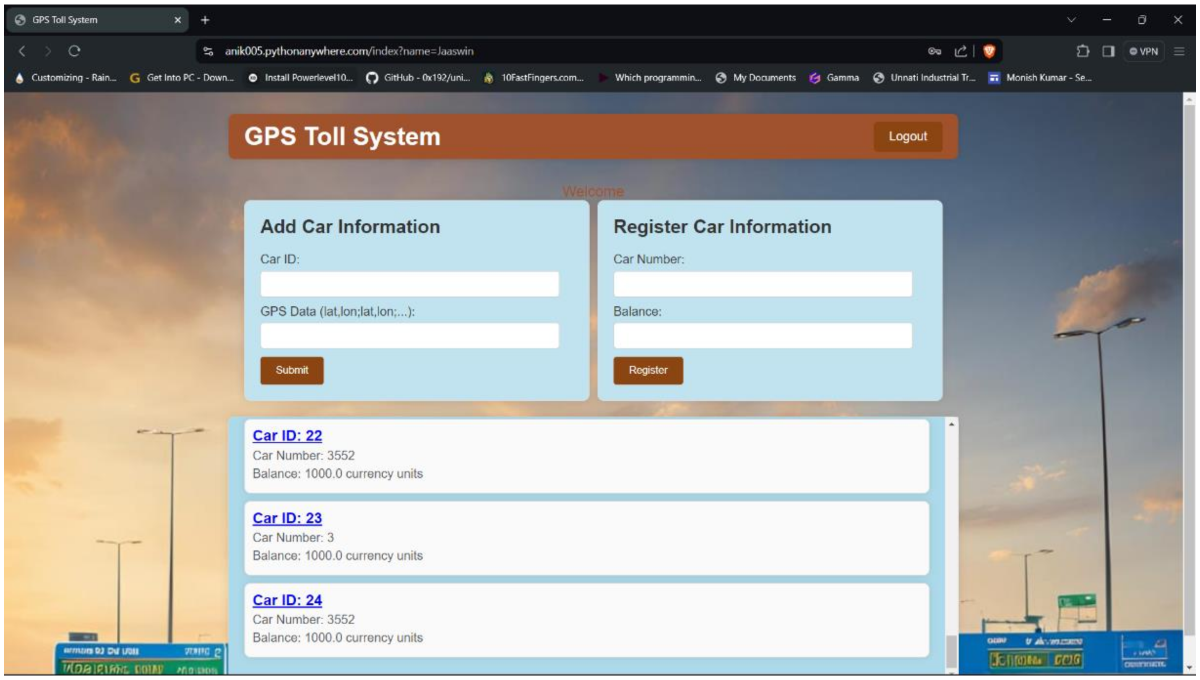
TEMPLATES AND STATIC FILES

- **Templates**: HTML files for login, register, index, forgot password, and car details pages.
- **Static Files**: login.csv for storing user details.

5. RESULTS & BENEFITS

RESULTS:

INDEX PAGE:

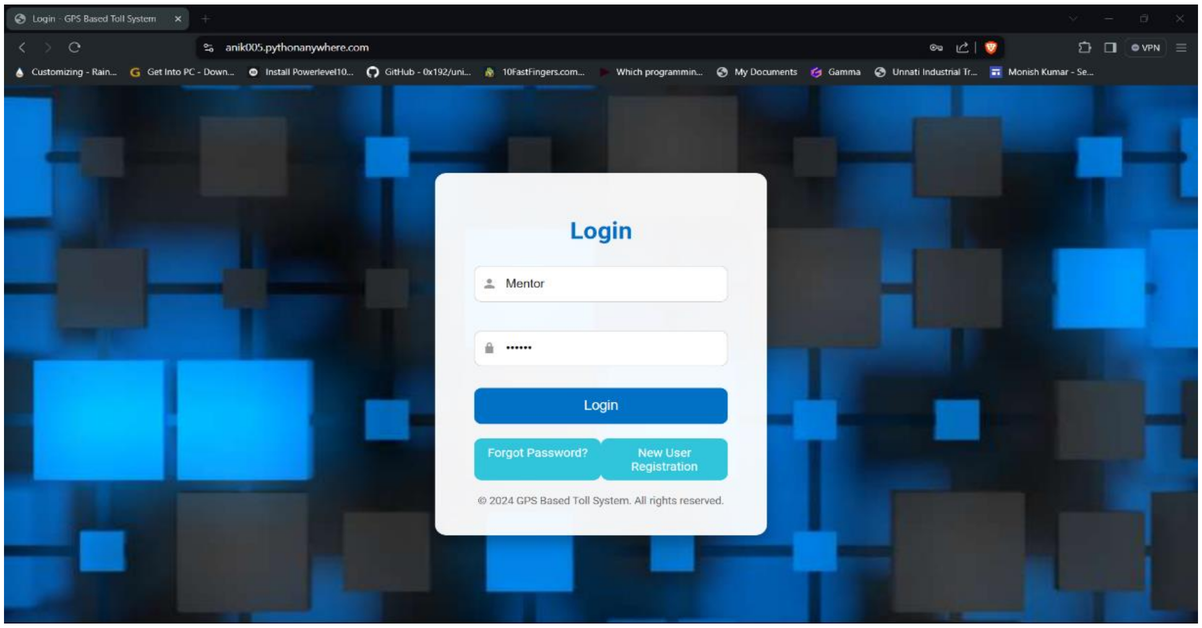


Car ID: 22
Car Number: 3552
Balance: 1000.0 currency units

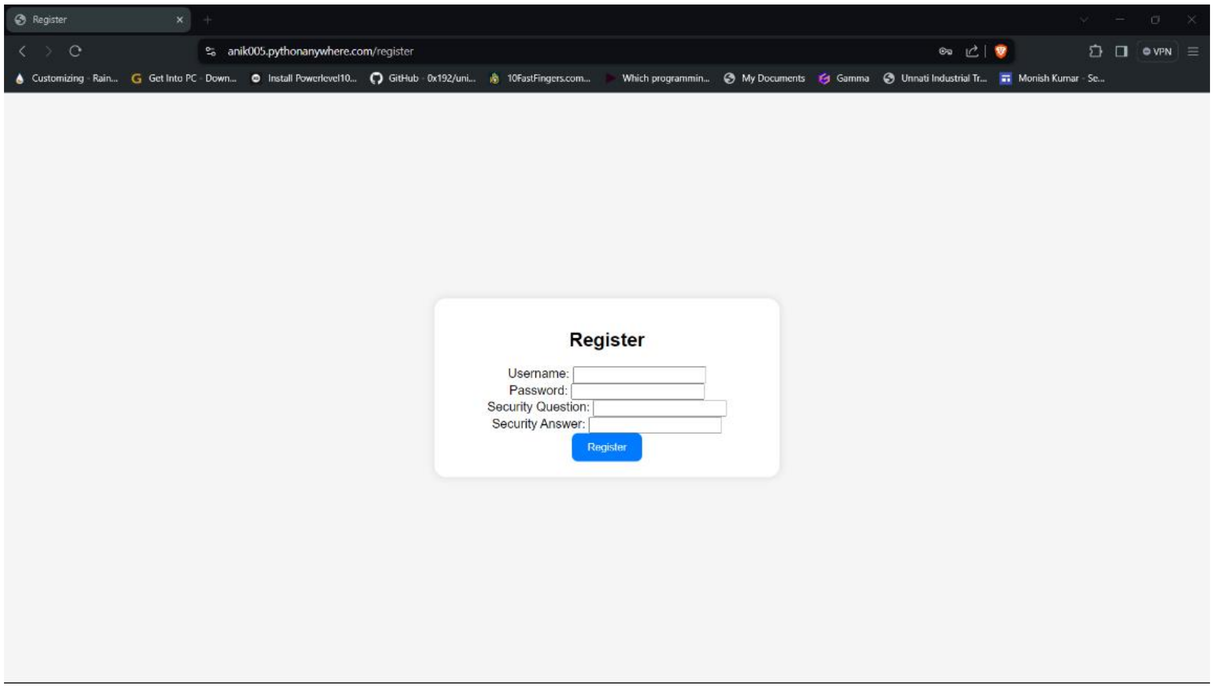
Car ID: 23
Car Number: 3
Balance: 1000.0 currency units

Car ID: 24
Car Number: 3552
Balance: 1000.0 currency units

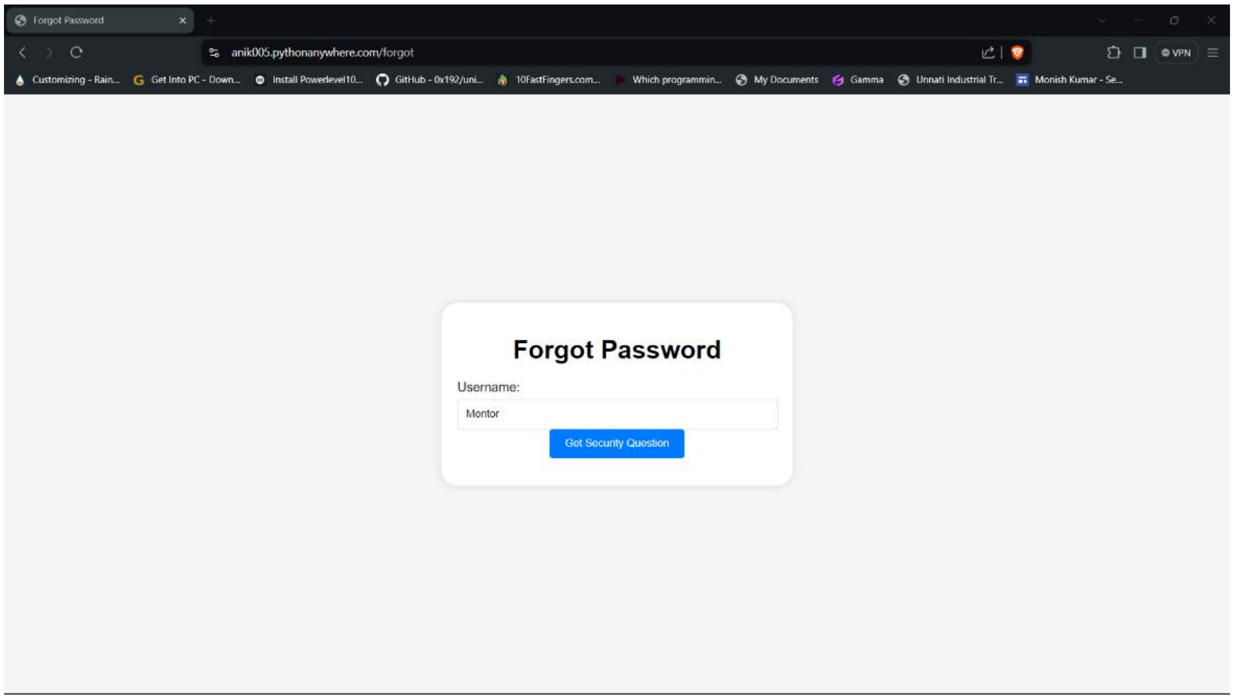
LOGIN PAGE:


Login - GPS Based Toll System
anik005.pythonanywhere.com
Login
Mentor
••••••
Login
Forgot Password?
New User Registration
© 2024 GPS Based Toll System. All rights reserved.

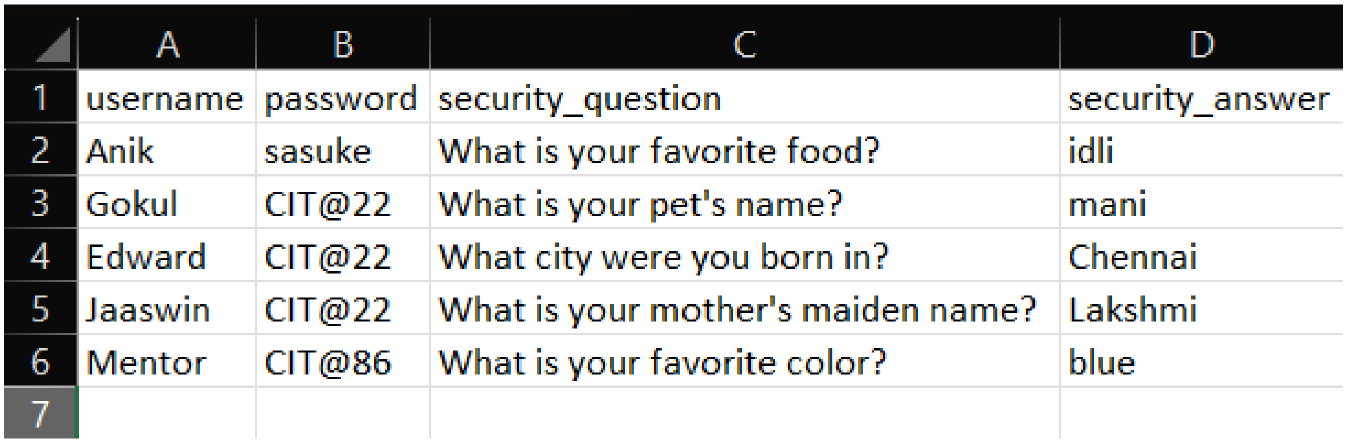
NEW USER REGISTRATION PAGE:



FORGOT PASSWORD PAGE:

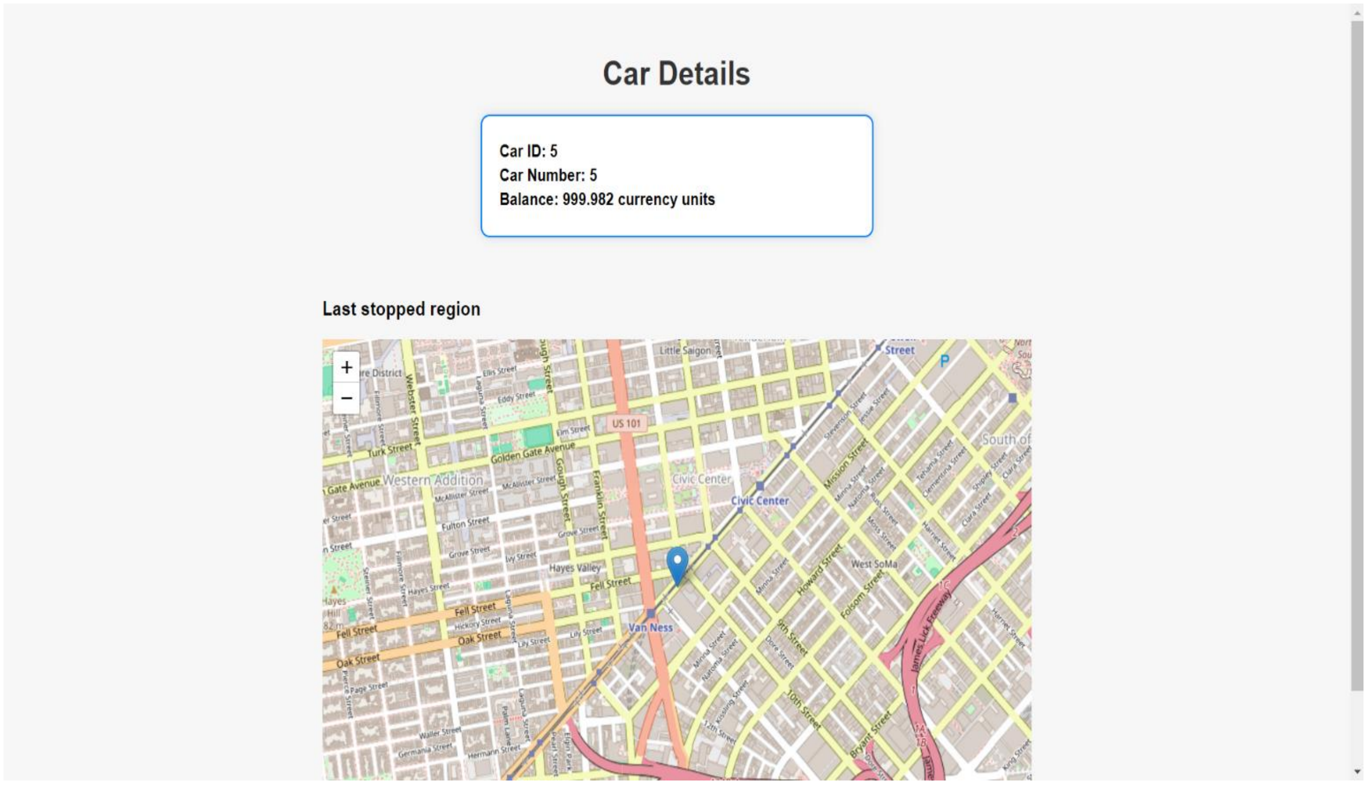


LOGIN CREDENTIALS FILE:



	A	B	C	D
1	username	password	security_question	security_answer
2	Anik	sasuke	What is your favorite food?	idli
3	Gokul	CIT@22	What is your pet's name?	mani
4	Edward	CIT@22	What city were you born in?	Chennai
5	Jaaswin	CIT@22	What is your mother's maiden name?	Lakshmi
6	Mentor	CIT@86	What is your favorite color?	blue
7				

CAR DETAILS DISPLAY:



Car Details

Car ID: 5
Car Number: 5
Balance: 999.982 currency units

Last stopped region

BENEFITS:

 Efficiency: Automated toll calculation and collection significantly reduce processing time at toll plazas, eliminating long queues and traffic congestion.

 Accuracy: GPS data ensures precise tracking of vehicle movements and accurate toll calculations based on actual distances travelled.

 User Convenience: The user-friendly interface and automated processes offer a seamless experience for drivers, reducing the inconvenience associated with manual toll payments.

 Scalability: The system is designed to handle large volumes of data and can be easily scaled to accommodate an increasing number of users and vehicles.

 Security: Robust authentication and session management mechanisms ensure the security of user data and transactions.

6. CHALLENGES & FUTURE ENHANCEMENTS

KEY CHALLENGES:

- **Accuracy of GPS Data:** Ensuring precise location data to calculate tolls correctly.
- **Data Security:** Safeguarding sensitive vehicle and personal information.
- **Scalability:** Managing increased data load as the number of vehicles and transactions grow.
- **Real-time Processing:** Maintaining efficient real-time processing for immediate toll calculations.
- **User Adoption:** Ensuring the system is user-friendly for government employees.
- **Integration with Existing Systems:** Seamlessly integrating with current toll and vehicle registration systems.

OVERCOMING THEM:

- **Improved GPS Algorithms:** Utilizing advanced algorithms and error-correction techniques to enhance GPS accuracy.
- **Robust Security Measures:** Implementing encryption, secure authentication, and regular audits to protect data.
- **Scalable Infrastructure:** Using cloud-based solutions and scalable databases to handle growing data volumes.
- **Optimized Code and Hardware:** Employing efficient coding practices and powerful hardware to ensure real-time processing.
- **User Training and Support:** Providing comprehensive training and ongoing support to facilitate user adoption.
- **API Integration:** Developing APIs for smooth integration with existing systems.

FUTURE DEVELOPMENTS:

- **Enhanced Analytics:** Incorporating advanced data analytics to provide insights into toll patterns and vehicle movements.
- **Mobile Application:** Developing a mobile app for users to check balances, transaction history, and manage accounts on the go.

- **Automated Alerts:** Setting up automated alerts for low balances, toll deductions, and other important notifications.
- **AI and Machine Learning:** Leveraging AI and ML to predict toll patterns, optimize routes, and detect fraudulent activities.
- **Expanded Coverage:** Extending the system to cover more geographic areas and additional types of vehicles.
- **Interoperability:** Ensuring the system can work seamlessly with other transportation and logistics platforms.

7. SCALABILITY & SECURITY FEATURES

SCALABILITY:

 Modular Design: The application is structured into distinct routes and functions, making it easier to maintain, extend, and scale as the project grows.

 Database Integration: Utilizes MySQL for efficient data storage and retrieval, supporting a large number of entries and high user requests with optimization capabilities for performance.

 Responsive Design: HTML and CSS ensure the application is accessible and functional on various devices, enhancing user experience and supporting a wide user base.

 Efficient Use of External Libraries: External libraries like folim for map rendering handle large datasets efficiently, improving performance.

 Asynchronous Operations: Asynchronous operations are implemented where applicable to improve performance and responsiveness, allowing the application to handle multiple requests concurrently.

SECURITY FEATURES:

 Session Management: Flask's session management handles user sessions securely, storing sensitive information server-side and session identifiers in cookies to maintain user state.

 Password Hashing: (Not explicitly shown but critical) Passwords should be hashed using algorithms like bcrypt to store them securely, protecting against data breaches.

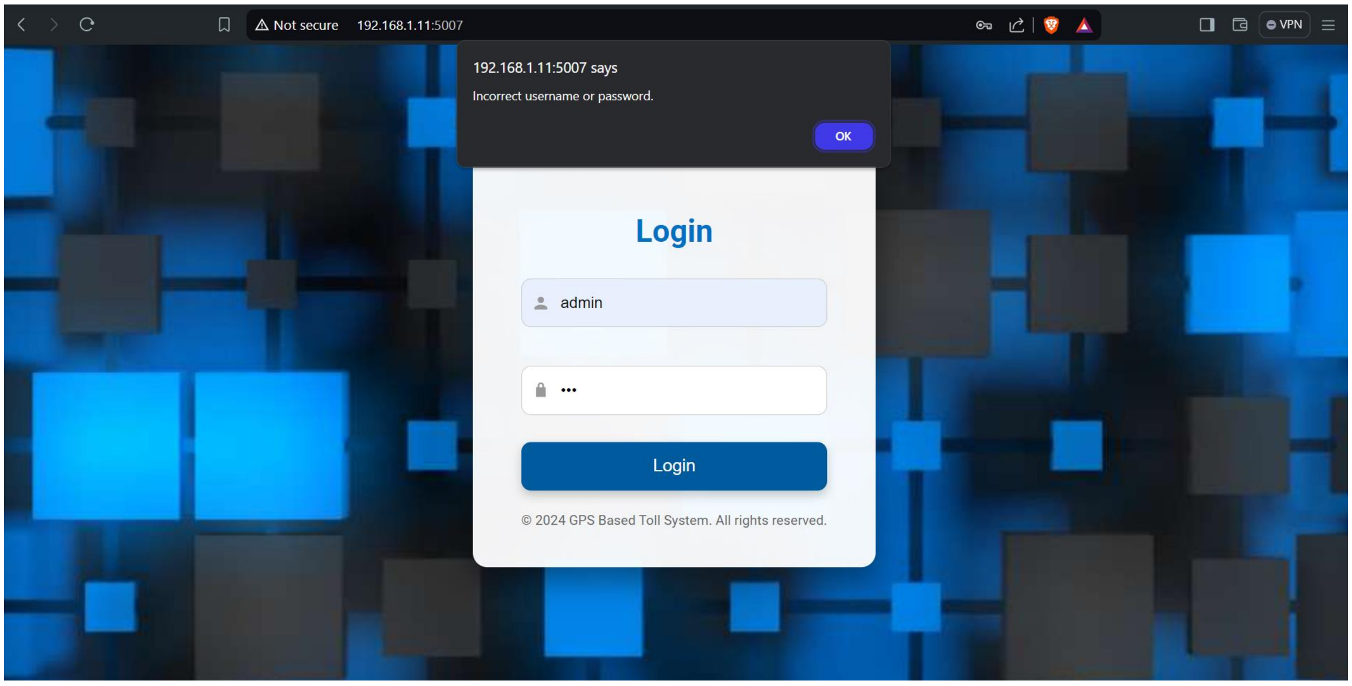
 Input Validation and Sanitization: Validates and sanitizes user inputs to prevent SQL injection and cross-site scripting (XSS) attacks, ensuring data integrity and application security.

 CSRF Protection: Implements Cross-Site Request Forgery (CSRF) protection using libraries like Flask-WTF to secure form submissions and prevent unauthorized requests.

 HTTPS Implementation: Ensures the application is served over HTTPS to encrypt data transmitted between client and server, protecting against man-in-the-middle attacks.

 Content Security Policy (CSP): Implements CSP headers to mitigate the risk of XSS attacks by controlling the sources from which content can be loaded.

 Form Validation: HTML forms include required attributes and JavaScript functions for input validation before submission, ensuring data accuracy and preventing invalid inputs.



192.168.1.11:5007 says

Incorrect username or password.

© 2024 GPS Based Toll System. All rights reserved.

GLOSSARY

1. GPS (Global Positioning System): A satellite-based navigation system that provides location and time information anywhere on Earth.

2. Toll Charges: Fees collected for using a road, bridge, or tunnel, typically used for maintenance and infrastructure improvements.

3. Flask: A lightweight web framework in Python used for developing web applications and handling HTTP requests.

4. Geopy: A Python library used for geocoding (finding latitude and longitude from an address) and calculating geographical distances.

5. Folium: A Python library used for creating interactive maps and visualizing geographical data.

6. MySQL: A widely-used open-source relational database management system.

7. MySQL Connector: A Python library used for connecting to a MySQL database server and executing SQL queries.

8. GET Request: An HTTP method used to request data from a specified resource.

9. POST Request: An HTTP method used to send data to a server to create or update a resource.

10. Interactive Web UI: A user-friendly web interface that allows users to interact with the application easily.

11. Scalability: The ability of a system to handle a growing amount of work or its potential to accommodate growth.

12. Real-time Processing: Immediate processing of data to provide instant outputs and updates.

13. Encryption: The process of converting information or data into a code to prevent unauthorized access.

14. API (Application Programming Interface): A set of protocols and tools for building software and applications, allowing different systems to communicate.

15. AI (Artificial Intelligence): The simulation of human intelligence processes by machines, especially computer systems.

16. Machine Learning (ML): A branch of AI that involves the use of algorithms and statistical models to enable computers to improve their performance on tasks through experience.

OPERATING ENVIRONMENT & GUIDELINESS

HARDWARE:

Processor (CPU)

Intel Core i5 (8th Gen or newer) or AMD Ryzen 5 (3rd Gen or newer)

Quad-core or higher

Memory (RAM)

16 GB (minimum)

32 GB (recommended) for handling larger datasets and more intensive computations

Storage

SSD (Solid State Drive) for faster read/write speeds

256 GB (minimum)

512 GB to 1 TB (recommended) depending on the size of your data and projects

Graphics Card (GPU)

Not mandatory unless you are performing GPU-accelerated tasks (e.g., Machine Learning)

NVIDIA GTX 1650 or higher for GPU tasks

Operating System

Windows 10/11, macOS, or Linux (Ubuntu 20.04 LTS or newer)

Additional Requirements

Network Connectivity: Reliable internet connection for downloading dependencies and database connectivity

Display: Full HD (1920x1080) or higher resolution

Battery: For laptops, consider a model with a good battery life if you need portability

SOFTWARE:

1. Python: Version 3.8 or newer
2. Database Client: Depending on the database you are connecting to (e.g., MySQL, PostgreSQL, SQLite)
3. IDE/Editor: PyCharm, VS Code, or Jupyter Notebook
4. Virtual Environment: Use venv or conda for managing project dependencies
5. Dependencies: List of dependencies in requirements.txt (for pip) or environment.yml (for conda)

HOW TO USE:

1. **Clone the Project Repository**:
   - Open your terminal (Command Prompt, Git Bash, etc.).
   - Navigate to the directory where you want to clone the project.
   - Use the following command to clone the repository:

```
git clone https://github.com/JASWINCKS/GPS-Toll-based-System-Simulation.git
```

2. **Install Dependencies**:
   - Ensure you have Python and pip installed on your system.
   - Navigate to the cloned project directory:

```
cd GPS-Toll-based-System-Simulation
```

   - Install the dependencies listed in requirements.txt using pip:

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

3. **Database Configuration**:
   - The Data base schema is provide as db schema.txt.
   - Implement the database.
   - Update the database details in the app(2).py file.
4.
```
db_config = {
'host': '<host>',
'user': '<username>',
'password': '<password>',
'database': '<database Name>'}
```

5. **Start the Application**:
   - Use the following command to start the application:

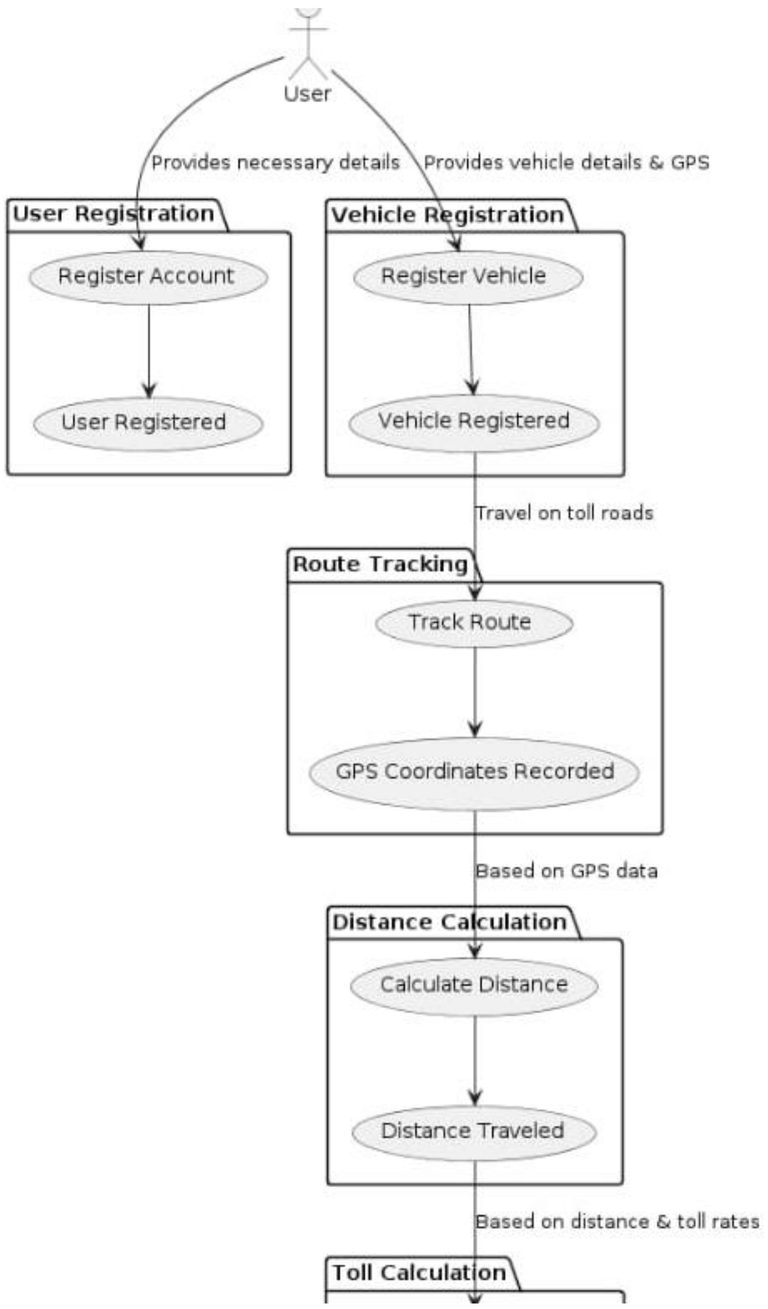
```
python app(2).py
```

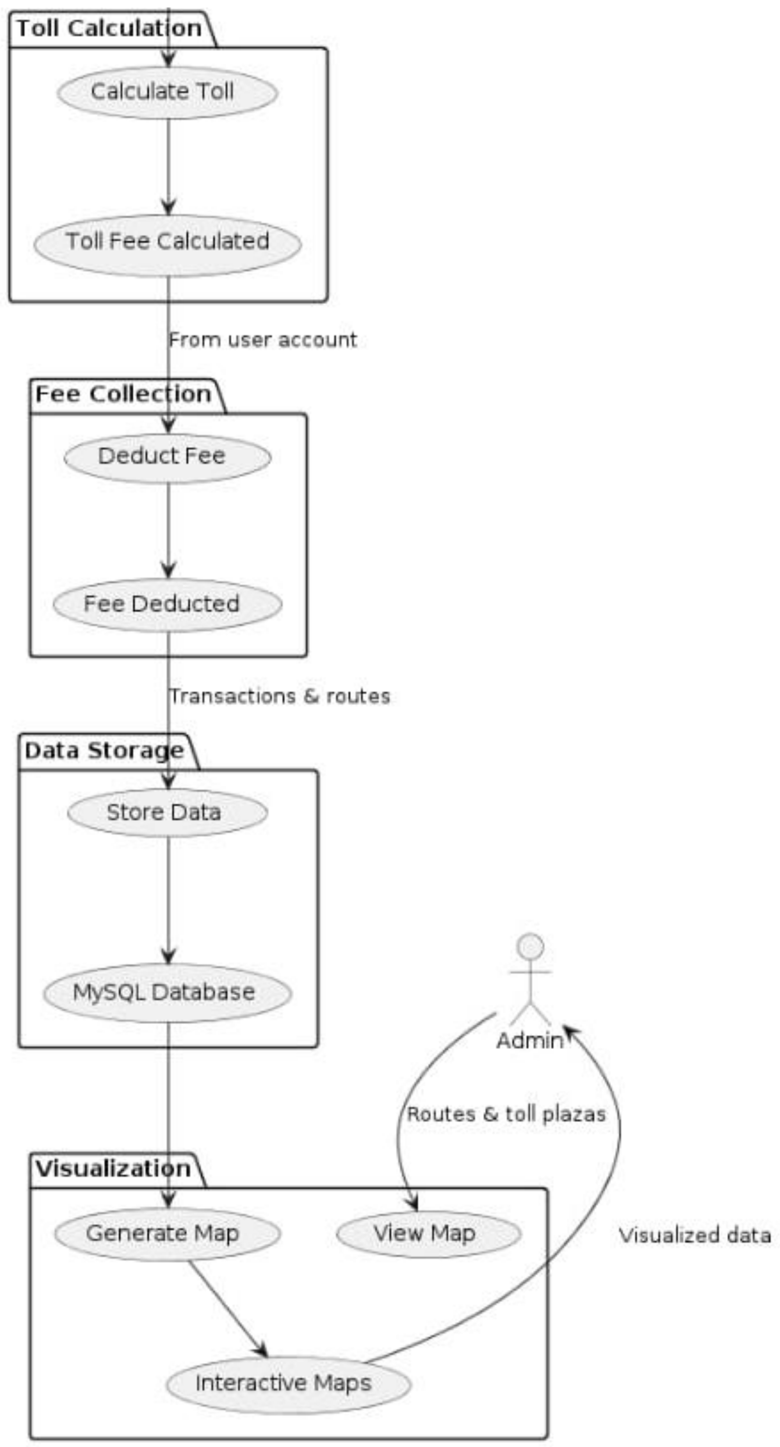
6. **Access the Application**:
   - Open your web browser and go to http://127.0.0.1:5000/.
   - Log in using your username and password. If you are a new user, proceed with the registration process and follow the on-screen instructions to register and then log in with your username and password.
   - After logging in, you will be redirected to the home page of the application. Enjoy using the application!

APPENDIX A: CONTEXT MODEL

INTERNAL CONTEXT DIAGRAM:

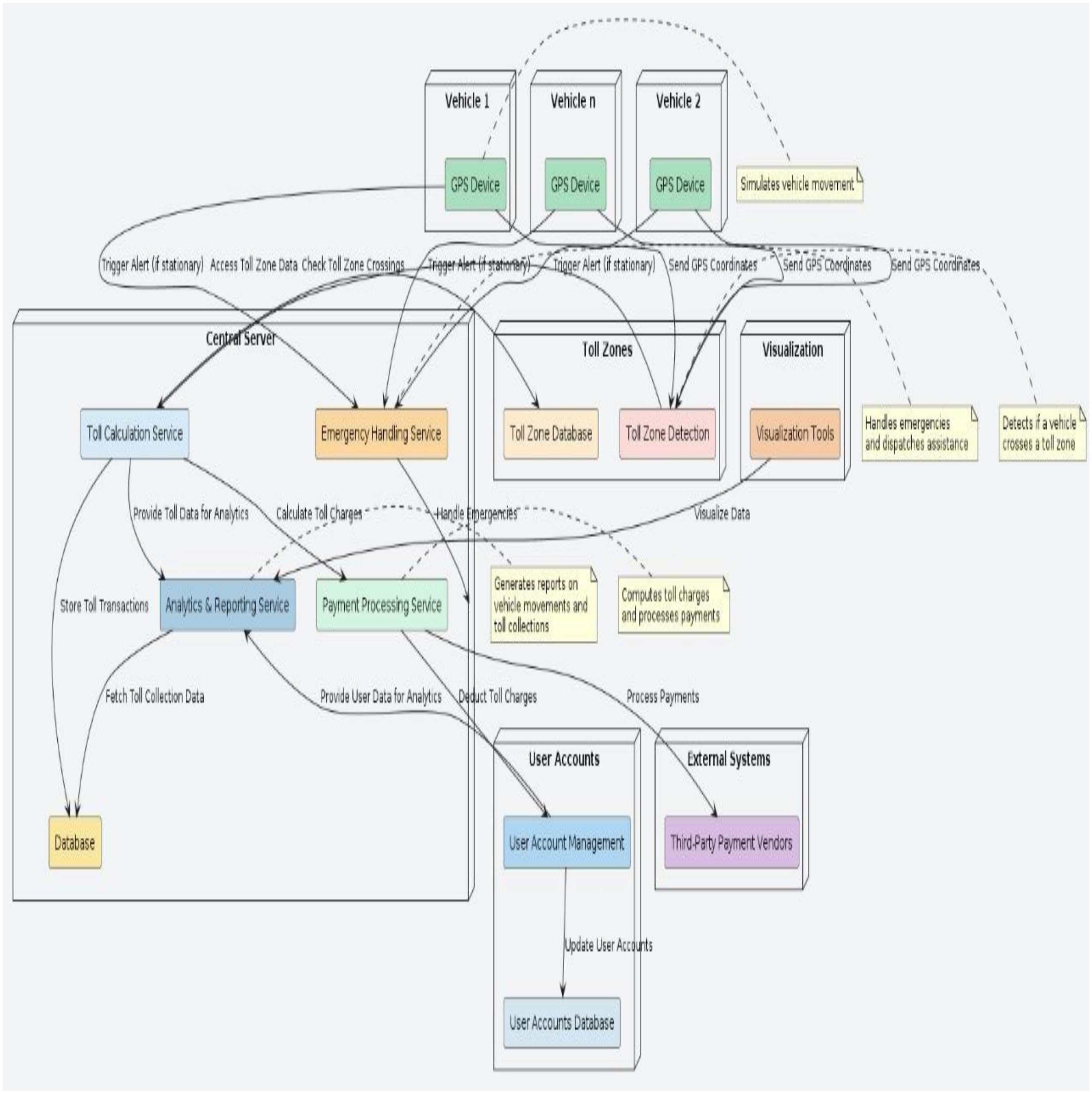
FLOWCHART:




Toll Calculation
Calculate Toll
Toll Fee Calculated
From user account
Fee Collection
Deduct Fee
Fee Deducted
Transactions & routes
Data Storage
Store Data
MySQL Database
Admin
Routes & toll plazas
Visualization
Generate Map
View Map
Visualized data
Interactive Maps

EXTERNAL CONTEXT DIAGRAM:

ARCHITECTURE DIAGRAM:


Vehicle 1
Vehicle n
Vehicle 2
GPS Device
GPS Device
GPS Device
Simulates vehicle movement
Trigger Alert (if stationary)
Access Toll Zone Data
Check Toll Zone Crossings
Trigger Alert (if stationary)
Trigger Alert (if stationary)
Send GPS Coordinates
Send GPS Coordinates
Send GPS Coordinates
Central Server
Toll Zones
Visualization
Toll Calculation Service
Emergency Handling Service
Toll Zone Database
Toll Zone Detection
Visualization Tools
Handles emergencies and dispatches assistance
Detects if a vehicle crosses a toll zone
Provide Toll Data for Analytics
Calculate Toll Charges
Handle Emergencies
Visualize Data
Store Toll Transactions
Analytics & Reporting Service
Payment Processing Service
Generates reports on vehicle movements and toll collections
Computes toll charges and processes payments
Fetch Toll Collection Data
Provide User Data for Analytics
Deduct Toll Charges
Process Payments
User Accounts
External Systems
Database
User Account Management
Third-Party Payment Vendors
Update User Accounts
User Accounts Database

USES OF CONTEXT MODELS:

 Visualization of Processes:

- Flowcharts visualize the sequence of operations, aiding in understanding data flow and logic.
- Architecture diagrams offer a high-level overview of system structure and component interactions.

 Improved Communication:

- Both tools help bridge the gap between technical and non-technical stakeholders.
- Useful in meetings, presentations, and documentation to ensure common understanding.

 Enhanced Design and Planning:

- Flowcharts assist in mapping out logic before coding, identifying issues early.
- Architecture diagrams help design modular, scalable, and maintainable systems.

 Debugging and Maintenance:

- Flowcharts make it easier to trace and debug issues by providing a step-by-step process view.
- Architecture diagrams show component connections, aiding in maintenance and updates.

 Training and Onboarding:

- New team members can quickly understand the system through these visual aids, reducing the learning curve.

 Documentation:

- Including these diagrams in documentation ensures a permanent record of the system's design and processes for future reference.

APPENDIX B: USE CASE ANALYSIS

USE CASE 1: REGISTER VEHICLE

Description: Users can register their vehicle in the toll system.

- Actors: User, System
- Preconditions: User must have valid vehicle information.
- Main Flow:
  1. User accesses the registration page.
  2. User inputs vehicle details (e.g., license plate number, vehicle type).
  3. System validates the input data.
  4. System registers the vehicle and stores details in the database.
  5. System confirms the registration to the user.
- Postconditions: Vehicle is registered in the system.

USE CASE 2: CALCULATE TOLL CHARGES

Description: System calculates toll charges based on vehicle location data.

- Actors: User, System
- Preconditions: Vehicle must be registered in the system, and GPS data must be available.
- Main Flow:
  1. System receives GPS data of the vehicle.
  2. System calculates the distance traveled using Geopy.
  3. System applies toll rate based on distance and vehicle type.
  4. System updates the toll charges for the vehicle in the database.
  5. System notifies the user of the updated toll charges.
- Postconditions: Toll charges are calculated and updated in the system.

USE CASE 3: UPDATE VEHICLE BALANCE

Description: Users can update the balance associated with their vehicle.

- Actors: User, System
- Preconditions: Vehicle must be registered and have a balance account.
- Main Flow:
  1. User accesses the balance update page.
  2. User inputs the vehicle ID and amount to add.
  3. System validates the input data.
  4. System updates the balance in the database.
  5. System confirms the update to the user.
- Postconditions: Vehicle balance is updated in the system.

USE CASE 4: VIEW VEHICLE DETAILS

Description: Users can view details of their registered vehicles.

- Actors: User, System
- Preconditions: User must have registered vehicles in the system.
- Main Flow:
  1. User accesses the vehicle details page.
  2. User inputs the vehicle ID or selects from a list.
  3. System retrieves the vehicle details from the database.
  4. System displays the details to the user.
- Postconditions: Vehicle details are displayed to the user.

Use Case 5: Manage Toll Booths

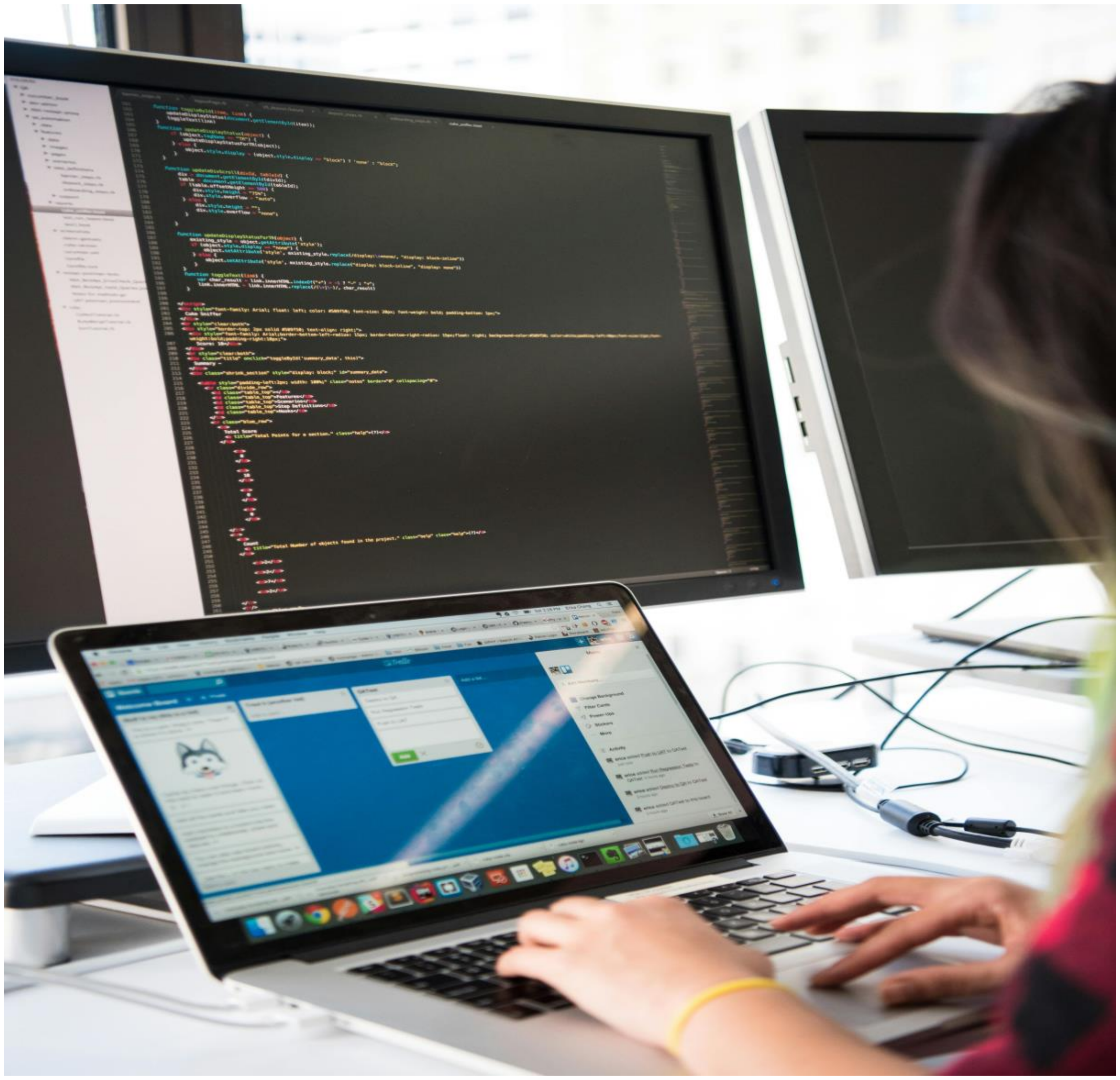
Description: Administrators can manage toll booths in the system.

- Actors: Administrator, System
- Preconditions: Administrator must have appropriate access rights.
- Main Flow:

1. Administrator accesses the toll booth management page.
2. Administrator inputs or updates toll booth information (e.g., location, rates).
3. System validates the input data.
4. System updates the toll booth information in the database.
5. System confirms the update to the administrator.

- Postconditions: Toll booth information is updated in the system.

Each of these use cases helps define the functionality and interaction of the system, ensuring that all requirements are met and providing a clear roadmap for development and testing.



APPENDIX C: DEVELOPMENT PROCESS

1. Planning

The planning phase involves defining the scope and objectives of the project. Key activities include:

- Requirement Gathering: Collecting requirements from stakeholders, such as toll operators and government officials.

- Feasibility Study: Analyzing the feasibility of the project in terms of technical, economic, and operational aspects.

- Project Plan: Creating a detailed project plan outlining timelines, resources, and milestones.

2. System Design

The system design phase focuses on designing the architecture of the toll system. Key activities include:

- High-Level Design (HLD): Defining the overall system architecture, including components like the database, web server, and GPS integration.

- Low-Level Design (LLD): Detailing the design of individual components, such as database schema, API endpoints, and user interface layout.

3. Development

The development phase involves coding and implementing the system. Key activities include:

- Backend Development: Implementing the server-side logic using Flask, including APIs for vehicle registration, toll calculation, and balance updates.

- Frontend Development: Creating the user interface using HTML, CSS, and JavaScript to interact with the backend.

- Database Integration: Setting up the MySQL database and implementing data models to store vehicle and toll data.

- Distance Calculation: Using the Geopy library to calculate distances based on GPS data.

- Mapping: Implementing Folium for visualizing vehicle locations and toll booth positions on interactive maps.

4. Testing

The testing phase ensures the system works as expected and meets the requirements. Key activities include:

- Unit Testing: Testing individual components and functions to ensure they work correctly.

- Integration Testing: Testing the integration of different components to ensure they work together seamlessly.

- System Testing: Conducting end-to-end testing of the entire system to identify and fix any issues.

- User Acceptance Testing (UAT): Allowing end users to test the system and provide feedback for further improvements.

5. Deployment

The deployment phase involves making the system available for use. Key activities include:

- Server Setup: Setting up the production server and configuring it to run the Flask application.

- Database Deployment: Deploying the MySQL database and migrating data from the development environment.

- Application Deployment: Deploying the backend and frontend code to the production server.

- Monitoring: Setting up monitoring tools to track system performance and detect any issues.

6. Maintenance

The maintenance phase involves ongoing support and updates to the system. Key activities include:

- Bug Fixes: Identifying and fixing any bugs or issues that arise after deployment.

- Performance Optimization: Continuously optimizing the system for better performance and scalability.

- Feature Enhancements: Adding new features and improvements based on user feedback and changing requirements.

- Documentation: Keeping the system documentation up-to-date with any changes or new features..

This development process ensures a structured approach to building a reliable and efficient toll system, from initial planning to ongoing maintenance.

APPENDIX D: TECHNOLOGIES USED

Programming Languages

- Python: The primary language used for backend development, including server-side logic, API creation, and integration with various libraries.

- HTML/CSS: Used for designing and styling the user interface.

Frameworks and Libraries

- Flask: A lightweight web framework for developing the backend and handling HTTP requests.

- Geopy: A Python library for calculating distances between geographic coordinates.

- Folium: A Python library for creating interactive maps to visualize GPS data.

- MySQL Connector: A library for connecting and interacting with the MySQL database from Python.

Database

- MySQL: A relational database management system used to store and manage vehicle and toll data.

Web Technologies

- HTML5: The latest version of the HTML standard, used for structuring the web pages.

- CSS3: The latest version of the CSS standard, used for styling the web pages.

Tools and Platforms

- Git: A version control system used for tracking changes and collaborating on the codebase.

- GitHub: A web-based platform for hosting and managing the Git repositories.

Development Environment

- Visual Studio Code (VS Code): A source code editor used for writing and debugging the code.

- VirtualBox: A virtualization tool used for creating virtual machines to simulate different environments.

- Ubuntu: A Linux-based operating system used for development and testing.

Deployment and Monitoring

- Heroku: A cloud platform used for deploying the web application and ensuring it is accessible online.

- New Relic: A monitoring tool used to track the performance and health of the deployed application.

Data Visualization

- Folium: Used for generating maps to visualize vehicle movements and toll booth locations.

- Matplotlib: A Python plotting library used for creating static, animated, and interactive visualizations.

APIs and Services

- Google Maps API: Used for geolocation and mapping services, providing accurate location data for vehicles and toll booths.

- Flask-RESTful: An extension for Flask that simplifies the creation of REST APIs.

This appendix provides a comprehensive overview of the technologies used in the development and deployment of the GPS toll system simulation project.

CONTRIBUTIONS

- **MEMBER 1: ANIRUDH.K (BACKEND DEVELOPER)**
  1. **Flask Implementation:** Configured Flask for seamless backend operations, facilitating frontend and database interactions.
  2. **Car Registration:** Developed features for registering cars and managing balances in MySQL.
  3. **GPS Data Handling:** Implemented mechanisms for receiving, storing, and processing GPS data, including distance calculations and geofencing logic.
- **MEMBER 2: JAASWIN.S (FRONTEND DEVELOPER)**
  1. **User Interface Development:** Designed intuitive UI using HTML, CSS, and JavaScript for optimal user experience.
  2. **Map Integration:** Integrated Folium maps for real-time route visualization and GPS data representation.
  3. **Authentication & Forms:** Implemented user authentication, session management, and interactive forms for registration and data input.
- **MEMBER 3: EDWARD SAMUEL.L (DATA & GEOSPATIAL ANALYST)**
  1. **GPS Data Analysis:** Analyzed GPS data to determine vehicle routes and distances traveled.
  2. **Geospatial Calculations:** Used Geopy for geospatial calculations, including distance measurement and geofencing.
  3. **Toll Calculation:** Developed algorithms for accurate toll fee computation based on distance travelled within geofenced areas.
- **MEMBER 4: GOKUL.P (SIMULATION & REPORT DEVELOPER)**
  1. **Simulation:** Conducted comprehensive simulations to validate toll calculation scenarios and system reliability.
  2. **Documentation & Presentation:** Compiled comprehensive documentation covering system architecture, user guides, and technical reports, alongside preparing and delivering presentations to effectively communicate project goals, methodologies, and outcomes to stakeholders.
  3. **Password and Security:** Integrated Flask-Login for secure user authentication and session management, and implemented password recovery techniques to enhance overall system security.

MODIFICATION HISTORY

Date	**Modifications**	**Reason**	**Version**
30/05/2024	Created a draft version of the document.	Adapted from various IEEE standards	1.0d
15/06/2024	Added a flowchart	To understand the rough workflow of the project	1.1d
30/06/2024	Minor amendments	As recommended by reviewer	1.2d
05/07/2024	Modify document content for the GPS Toll-based System Simulation using Python	Adapted for Project Reoprt	2.0d

Table 1.0 Modification History