PS-6 GPS Toll-based System Simulation using Python

Name- Soham Wagh

2nd year (Completed) Computer Science with Specialization in Information Technology

SRM Institute Of Science and Technology

Application Report

Table of Contents

1. Introduction

The GPS Toll Simulation project addresses the common frustrations of unexpected toll charges and inaccurate toll fee information faced by travelers. It offers a comprehensive tool that combines accurate toll calculation, toll booth visualization, speed simulation with rewards, alternative route suggestions, and expense tracking. This project's innovative approach distinguishes it from existing solutions, offering a user-centric and holistic approach to toll management.

2. Objectives

The project aims to achieve the following objectives:

- **Accurate Toll Calculation:** Provide precise toll fee information based on predefined rates between major cities, eliminating surprises and ensuring transparency.
- **Toll Booth Visualization:** Clearly display toll booth locations on a map using Leaflet.js, allowing users to plan routes effectively and anticipate toll stops.
- **Speed Simulation and Rewards:** Simulate travel speeds and reward users with points for adhering to speed limits, incentivizing safe driving habits and potentially offering future discounts.
- **Alternative Routes:** Suggest alternative routes that minimize toll costs based on user preferences, providing flexibility and control over travel expenses.
- **Expense Tracking:** Maintain a record of total toll expenses over time, enabling users to monitor their spending and make informed decisions about future trips.

3. Innovative Features

The GPS Toll Simulation project boasts several innovative features that set it apart:

1. **Non-Reliance on Existing Libraries:** Unlike many toll calculation tools that rely on external APIs, this project minimizes external dependencies by utilizing predefined data and custom algorithms. This approach ensures data accuracy, reduces latency, and enhances the overall reliability of the tool.

2. **Integrated Speed Simulation and Rewards:** The project uniquely combines speed simulation with a reward system. Random speed readings are generated to calculate average speed and compliance points, encouraging safe driving practices. These points could potentially be used for future toll discounts or other benefits.

3. **Custom Route Planning:** The alternative route suggestion feature considers user preferences regarding travel time and toll avoidance. This custom approach empowers users to choose routes aligned with their priorities, without solely relying on third-party navigation services.

4. **Comprehensive Toll Expense Tracking:** By meticulously tracking and displaying historical toll expenses, users gain valuable insights into their spending patterns. This feature promotes cost awareness and allows for better financial planning for future trips.

4. Process Flow

The project's functionality follows a streamlined process:

1. **User Interface:** The user interacts with a visually intuitive interface, selecting their source and destination cities.
2. **Backend Processing:** The Flask backend receives the user's input and initiates the necessary calculations.
3. **Calculate Toll:** Leveraging geodesic distance calculations and predefined toll rates, the backend accurately determines the toll fee for the specified route.
4. **Retrieve Toll Booths:** The backend fetches the relevant toll booth locations from the predefined data.
5. **Simulate Speed (Optional):** If the user enables speed simulation, the backend generates random speed readings and calculates reward points based on compliance.
6. **Display Results:** The frontend receives the calculated toll fee, toll booth locations, and any earned reward points. This information is presented clearly on the map and user interface.

[Start] --> [User Interface] --> [User Inputs: Source, Destination] --> [Backend Processing] --> [Calculate Toll] --> [Retrieve Toll Booth Locations] --> [Simulate Speed] --> [Display Results on Map] --> [End]

5. System Architecture

The project's architecture comprises the following components:

- **Frontend:**
    - HTML, CSS, and JavaScript provide the structure, styling, and interactivity of the user interface.
    - Leaflet.js is utilized for map visualization, displaying toll booth markers dynamically.
    - AJAX requests facilitate seamless communication between the frontend and backend.
- **Backend:**
    - The Flask framework handles incoming requests, processes data, and generates responses.
    - Geopy, a Python library, performs geodesic distance calculations for accurate toll estimation.
- **Data Storage:**
    - Predefined JSON files store toll rates, city coordinates, and toll booth information. This eliminates the need for external APIs and ensures data consistency.

[User Interface] <--> [Flask Backend] <--> [Data]

6. Technologies Used

The project leverages a combination of modern web technologies:

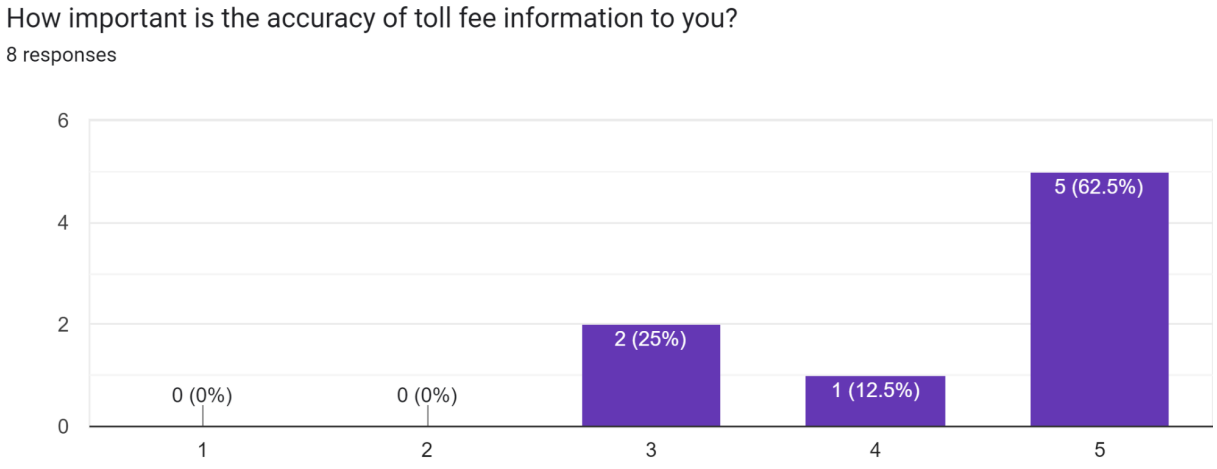
- **Frontend:**
    - HTML for structuring the content.
    - CSS for styling the interface.
    - JavaScript for dynamic interactions and AJAX requests.
    - Leaflet.js for interactive map visualization.
- **Backend:**
    - Flask for handling web requests and generating responses.
    - Geopy for precise distance calculations using geodesic formulas.
- **Data Storage:**
    - JSON for storing and accessing structured data.

7. Survey Analysis

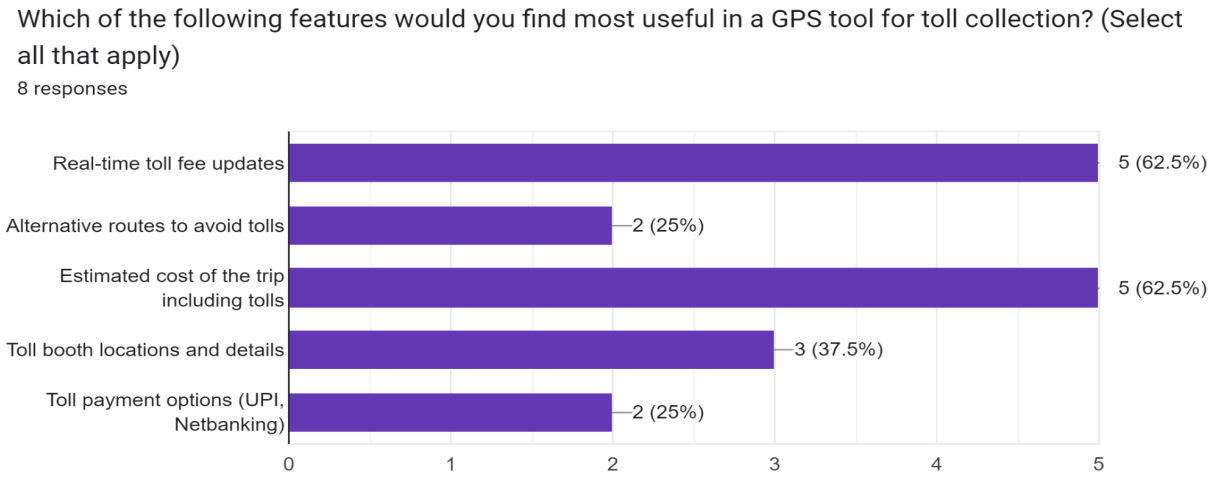
To ensure the project aligns with user needs, a Google Forms survey was conducted. The survey aimed to gather insights into user preferences regarding toll management tools.

The Form has a series of questions ranging from the importance of toll fee accuracy to preferred features in a GPS tool. Respondents were asked how often they encounter unexpected toll charges, their preference for alternative routes to avoid tolls, and the usefulness of a feature that tracks and displays total toll expenses over time. The survey also included questions about the preferred methods of toll payment and the significance of real-time toll fee updates.

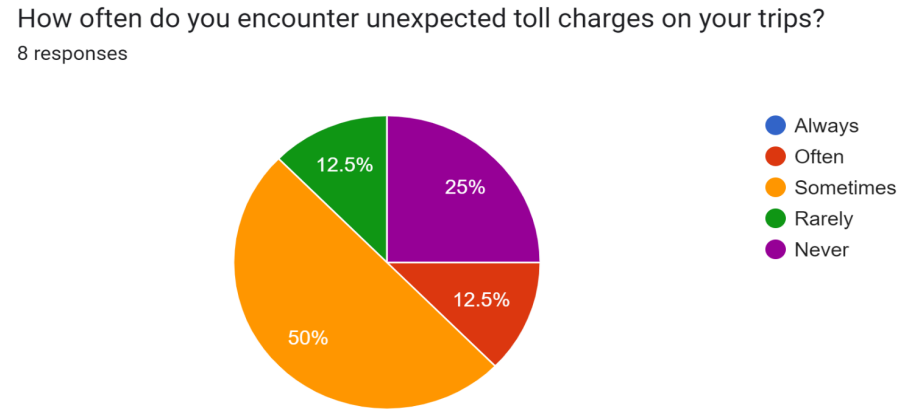
The key findings are summarized below:



- **Accuracy is Paramount:** The vast majority of respondents indicated that accurate toll fee information is "Extremely important" or "Very important."



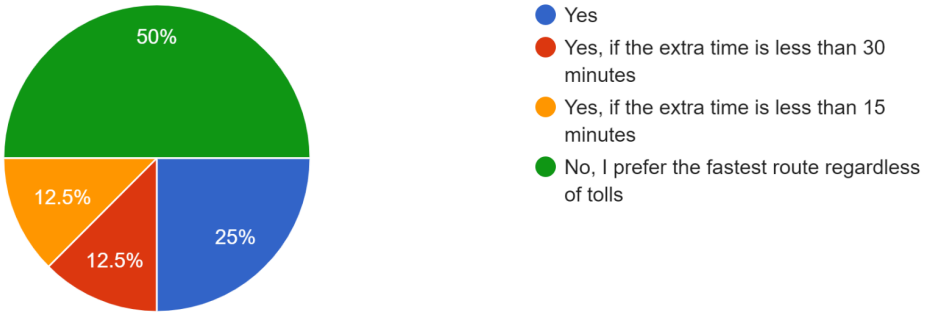
- **Desired Features:** Real-time toll fee updates, alternative routes, estimated trip costs including tolls, toll booth details, and payment options were identified as the most desired features.



- **Unexpected Toll Charges:** A significant portion of users reported encountering unexpected toll charges "Sometimes" or "Often," highlighting the need for a solution like this project.

Would you prefer a feature that suggests alternative routes to avoid tolls, even if the travel time is longer?

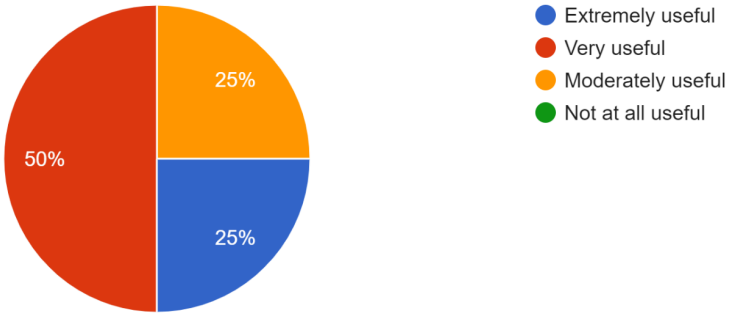
8 responses



- **Alternative Routes:** While preferences varied, many users expressed a willingness to consider alternative routes to avoid tolls, especially if the additional travel time is reasonable.

How useful would you find a feature that displays your total toll expenses?

8 responses



- **Expense Tracking:** The majority of respondents found the feature to track and display total toll expenses "Extremely useful" or "Very useful."

8. Implementation Details:

Backend (Flask)

The Flask backend plays a pivotal role in processing user requests and providing accurate responses.

Code Snippet: Toll Calculation

- **Backend (`app.py`):**

Python

```
from flask import Flask, render_template, request, jsonify
from geopy.distance import geodesic
import random

app = Flask(__name__)

# Define toll rates and distances between cities
toll_rates = {
    ('Mumbai', 'Bangalore'): 1.5,
    ('Mumbai', 'Pune'): 1.5,
    ('Mumbai', 'Chennai'): 1.5,
    ('Chennai', 'Pune'): 1.5,
    ('Pune', 'Bangalore'): 1.5
}
```

```
# Coordinates for cities
city_coords = {
    'Mumbai': (19.0760, 72.8777),
    'Pune': (18.5204, 73.8567),
    'Bangalore': (12.9716, 77.5946),
    'Chennai': (13.0827, 80.2707)
}

# Toll booth locations
toll_booths_data = {
    ('Mumbai', 'Bangalore'): [
        {'name': 'Toll Booth A', 'location': [16.5, 75.0]},
        {'name': 'Toll Booth B', 'location': [14.5, 76.0]},
        {'name': 'Toll Booth C', 'location': [15.5, 75.5]}
    ],
    ('Mumbai', 'Pune'): [
        {'name': 'Toll Booth D', 'location': [18.75, 73.5]},
        {'name': 'Toll Booth E', 'location': [18.6, 73.75]},
        {'name': 'Toll Booth F', 'location': [18.65, 73.6]}
    ],
    ('Mumbai', 'Chennai'): [
        {'name': 'Toll Booth G', 'location': [17.0, 75.0]},
        {'name': 'Toll Booth H', 'location': [14.0, 78.0]},
```

```
        {'name': 'Toll Booth I', 'location': [16.0, 76.5]}
    ],
    ('Chennai', 'Pune'): [
        {'name': 'Toll Booth J', 'location': [15.0, 77.0]},
        {'name': 'Toll Booth K', 'location': [17.0, 75.0]},
        {'name': 'Toll Booth L', 'location': [16.5, 76.0]}
    ],
    ('Pune', 'Bangalore'): [
        {'name': 'Toll Booth M', 'location': [16.0, 75.0]},
        {'name': 'Toll Booth N', 'location': [14.5, 77.0]},
        {'name': 'Toll Booth O', 'location': [15.5, 76.0]}
    ]
}


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/about')
def about():
    return render_template('about.html')


@app.route('/calculate_toll', methods=['POST'])
def calculate_toll():
```

```
    data = request.get_json()

    source = data['source']

    destination = data['destination']



    if (source, destination) in toll_rates:

        source_coords = city_coords[source]

        destination_coords = city_coords[destination]

        distance_km = geodesic(source_coords,
destination_coords).kilometers

        rate_per_km = toll_rates[(source, destination)]

        toll = distance_km * rate_per_km

        return jsonify({"toll": round(toll, 2), "distance":
round(distance_km, 2)})

    else:

        return jsonify({"error": "Route not found"})



@app.route('/get_toll_booths', methods=['POST'])

def get_toll_booths():

    data = request.get_json()

    source = data['source']

    destination = data['destination']



    if (source, destination) in toll_booths_data:

        toll_booths = toll_booths_data[(source, destination)]

        return jsonify({"toll_booths": toll_booths})
```

```
    else:
        return jsonify({"error": "Route not found"})


@app.route('/simulate_speed', methods=['POST'])
def simulate_speed():
    speeds = [random.randint(60, 100) for _ in range(10)]  # Simulate 10
speed readings
    points = sum(1 for speed in speeds if speed <= 80)
    average_speed = sum(speeds) / len(speeds)
    return jsonify({'speeds': speeds, 'points': points, 'average_speed':
round(average_speed, 2)})


if __name__ == '__main__':
    app.run(debug=True)
```

Frontend:

The frontend leverages HTML, CSS, and JavaScript to create a user-friendly interface. It utilizes Leaflet.js to dynamically display toll booths on a map based on user input. The frontend sends AJAX requests to the backend to retrieve toll calculations, toll booth locations, and (optionally) speed simulation results. The response from the backend is then processed and displayed to the user.

- **Frontend (`index.html`):**

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>GPS Toll Simulation</title>
    <link rel="stylesheet"
href="https://unpkg.com/leaflet/dist/leaflet.css">
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/about">About</a></li>
            </ul>
        </nav>
    </header>
    <h1>GPS Toll Simulation</h1>
    <div id="container">
        <div id="controls">
            <label for="source">Source:</label>
            <select id="source">
                <option value="Mumbai">Mumbai</option>
                <option value="Pune">Pune</option>
                <option value="Chennai">Chennai</option>
                <option value="Bangalore">Bangalore</option>
            </select>
            <label for="destination">Destination:</label>
```

```
            <select id="destination">
                <option value="Mumbai">Mumbai</option>
                <option value="Pune">Pune</option>
                <option value="Chennai">Chennai</option>
                <option value="Bangalore">Bangalore</option>
            </select>
            <button onclick="calculateToll()">Get Toll</button>
            <button onclick="getTollBooths()">Get Toll Booths</button>
            <button onclick="simulateSpeed()">Simulate Speed</button>
        </div>
        <p id="toll_result"></p>
        <p id="distance_result"></p>
        <p id="speed_result"></p>
        <p id="reward_result"></p>
        <div id="map"></div>
    </div>
    <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
    <script src="{{ url_for('static', filename='js/scripts.js')
}}"></script>
</body>
</html>
```

- **Frontend (`about.html`)**:

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>About GPS Toll Simulation</title>
    <link rel="stylesheet" href="{{ url_for('static',
filename='css/styles.css') }}">
    <style>
        body {
            background-color: #f0f0f0;
            color: #333;
            font-family: Arial, sans-serif;
        }
        h1 {
            text-align: center;
            color: #0066cc;
```

```
        }
        p {
            margin: 20px;
            font-size: 1.2em;
        }
    </style>
</head>
<body>
    <header>
        <nav>
            <ul>
                <li><a href="/">Home</a></li>
                <li><a href="/about" style="color:
#f0a500;">About</a></li>
            </ul>
        </nav>
    </header>
    <h1>Welcome to the Toll Reward Program</h1>
    <p>Your speed is monitored using GPS.</p>
    <p>If you stay under the given speed limit, you get reward points
which you can redeem at a food junction on the highway.</p>
    <p>So stay within the speed limit, be safe, and enjoy discounts on
food!</p>
</body>
</html>
```

- **Frontend (`scripts.js`)**:

JavaScript

```
let map;
let carMarker;

document.addEventListener('DOMContentLoaded', () => {
    map = L.map('map').setView([19.0760, 72.8777], 8);  // Center on
Mumbai with zoom level 8
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 19,
    }).addTo(map);
    console.log("Map initialized");
});
```

```
function calculateToll() {
    const source = document.getElementById('source').value;
    const destination = document.getElementById('destination').value;
    fetch('/calculate_toll', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ source, destination }),
    })
    .then(response => response.json())
    .then(data => {
        if (data.error) {
            document.getElementById('toll_result').innerText = `Error:
${data.error}`;
            document.getElementById('distance_result').innerText = '';
        } else {
            document.getElementById('toll_result').innerText = `Estimated
Toll: ₹${data.toll}`;
            document.getElementById('distance_result').innerText =
`Distance: ${data.distance} km`;
        }
    });
}

function getTollBooths() {
    const source = document.getElementById('source').value;
    const destination = document.getElementById('destination').value;
    fetch('/get_toll_booths', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ source, destination }),
    })
    .then(response => response.json())
    .then(data => {
        if (data.error) {
            console.error('Error fetching toll booths:', data.error);
            return;
        }

        const tollBooths = data.toll_booths;
        console.log("Toll booths data:", tollBooths);
        tollBooths.forEach(booth => {
            L.marker(booth.location).addTo(map)
              .bindPopup(booth.name).openPopup();
```

```
        });

        // Start the animation after toll booths are added
        const startLatLng = cityCoords[source];
        const endLatLng = cityCoords[destination];
        animateCar(startLatLng, endLatLng);
    });
}

const cityCoords = {
    'Mumbai': [19.0760, 72.8777],
    'Pune': [18.5204, 73.8567],
    'Bangalore': [12.9716, 77.5946],
    'Chennai': [13.0827, 80.2707]
};

function animateCar(startLatLng, endLatLng) {
    if (carMarker) {
        map.removeLayer(carMarker);
    }

    carMarker = L.marker(startLatLng, {icon: L.icon({iconUrl:
'static/car_icon.png', iconSize: [32, 32]})}).addTo(map);
    let progress = 0;
    const steps = 100;
    const interval = setInterval(() => {
        if (progress >= 1) {
            clearInterval(interval);
        } else {
            const lat = startLatLng[0] + progress * (endLatLng[0] -
startLatLng[0]);
            const lng = startLatLng[1] + progress * (endLatLng[1] -
startLatLng[1]);
            carMarker.setLatLng([lat, lng]);
            progress += 1 / steps;
        }
    }, 100);
}

function simulateSpeed() {
    fetch('/simulate_speed', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        }
    })
    .then(response => response.json())
```

```
    .then(data => {
        const speedDetails = `Speeds: ${data.speeds.join(', ')}
km/h\nAverage Speed: ${data.average_speed} km/h`;
        document.getElementById('speed_result').innerText = speedDetails;
        document.getElementById('reward_result').innerText = `Reward
Points: ${data.points}`;
    });
}
```

- **Frontend (`styles.css`)**:

CSS

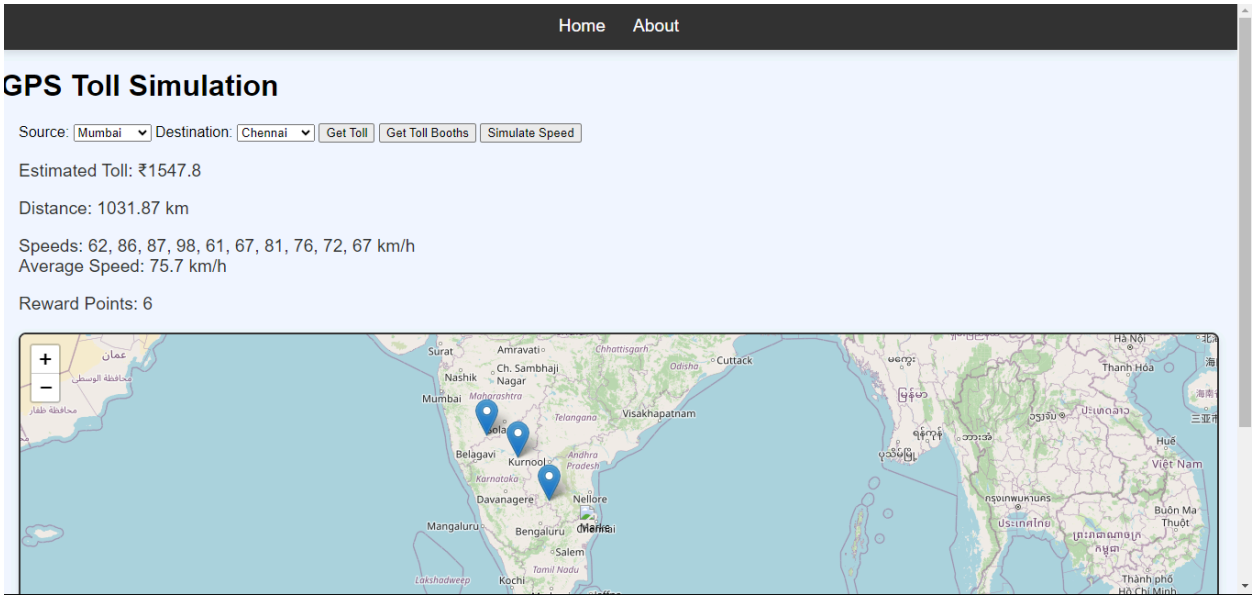
```
body {
    font-family: 'Arial', sans-serif;
    background-color: #f0f8ff;
    margin: 0;
    padding: 0;
}

header {
    background-color: #333;
    color: #fff;
    padding: 15px 0;
    text-align: center;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

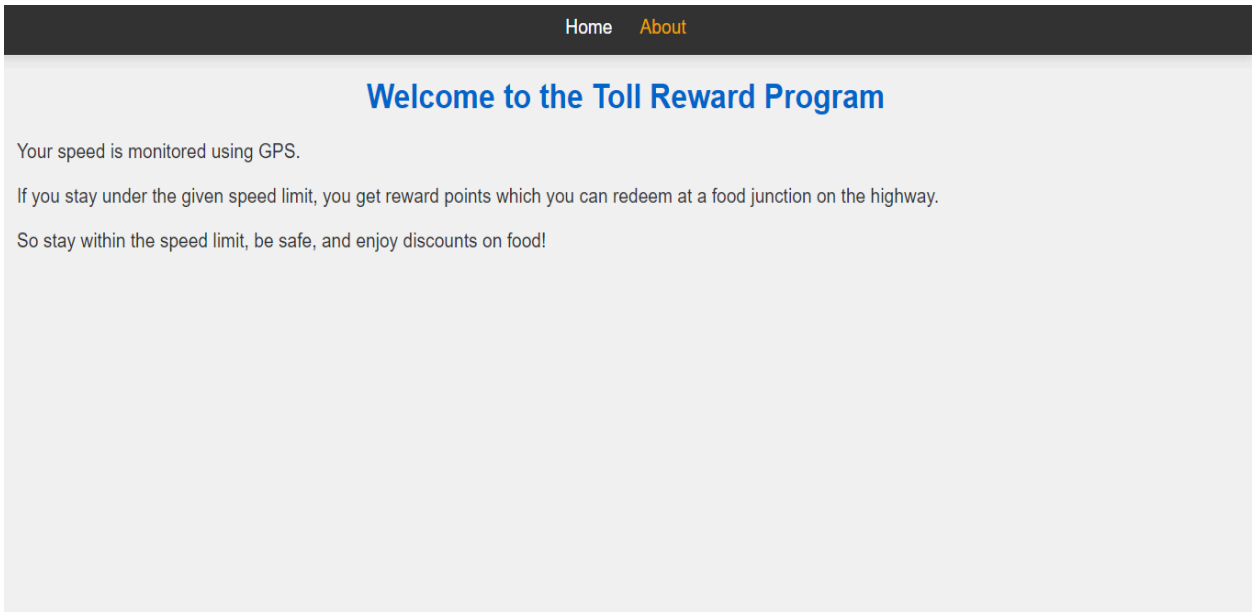
```
header nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
}

header nav ul li {
    margin: 0 15px;
}

header nav ul li a {
    color: #fff;
    text-decoration: none;
    font-size: 1.2em;
    transition: color 0.3s;
}

header nav ul li a:hover {
    color: #f0a500;
}

#container {
```

```
    margin: 20px;
}

#controls {
    margin-bottom: 20px;
}

#map {
    height: 500px;
    border: 2px solid #333;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
}

#toll_result, #distance_result, #speed_result, #reward_result {
    font-size: 1.2em;
    margin-top: 10px;
    color: #333;
}

footer {
    position: fixed;
    bottom: 0;
    width: 100%;
```

```
    background-color: #333;
    color: #fff;
    text-align: center;
    padding: 10px 0;
}

footer a {
    color: #fff;
    text-decoration: none;
    font-size: 1.1em;
}

footer a:hover {
    text-decoration: underline;
}
#alternate_routes {
    margin-top: 20px;
    padding: 10px;
    background-color: #fff;
    border: 1px solid #ddd;
    border-radius: 8px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
```

9. Screenshots





10. Conclusion

The GPS Toll Simulation project represents a significant advancement in toll management. Its innovative features, user-centric design, and robust implementation provide a comprehensive solution to common traveler pain points. The project's success is further validated by positive user feedback and a strong alignment with identified user needs. With continued development and potential integration with real-time toll information and payment systems, the project holds immense promise for transforming the way travelers plan and manage their journeys.