CHAPTER 1

INTRODUCTION

1.1 Problem Description

Vehicle Parking management system is a tool that is used to maintain vehicle parking details. The system makes use of a single centralized database to maintain records of all the customers who park their vehicle. Authentication process is implemented to allow only the Admin. Vehicle-in data, Vehicle-out data, history (of Vehicle)credentials are all stored in the database. All the processes are carried out through an interactive interface on a web platform.

Having a centralized database for all vehicles of the same group helps in maintaining the records and makes administrator jobs much easier. It also helps in comparing different attributes of the group. Vehicle parking management system is an automatic system that delivers data processing at a very high speed in a systematic manner . New vehicle parking had challenges concerning its safety of data in the store since they currently use paper based systems , physical struggle for parking by drivers, wastage of time, congestion and collision.

Paper-based systems have been widely used in the past, they are gradually being replaced by digital solutions like Vehicle Parking Management Systems (VPMS). These digital systems offer automation, improved efficiency, enhanced data analysis capabilities, and greater convenience for both administrators and users. This system majorly solved the congestion, collision and saves time during parking activities .

A Vehicle Parking Management System (VPMS) serves as a comprehensive tool for organizing and maintaining vehicle parking details efficiently. By utilizing a single centralized database, it effectively records information pertaining to customers who utilize parking services.

Authentication protocols are integrated to grant access exclusively to the designated administrator, ensuring secure management of the system. Key data such as vehicle entry and

exit records, as well as payment credentials, are meticulously stored within this centralized database.

Overall, a Vehicle Parking Management System represents a sophisticated solution for efficiently managing parking facilities, enhancing convenience for users, and optimizing administrative processes through centralized data management.

1.2 Aim and Objectives of the Project

To create a simple web-based parking management system using the Flask framework. This application allows users to add vehicles, categorized as either Cars or Bikes, to a parking lot by providing details such as the vehicle type, plate number, arrival time, and date. It also enables the display of all currently parked vehicles, showing their respective details, and tracks the total count of vehicles, including separate counts for Cars and Bikes. Additionally, the system calculates the parking charges based on the duration of the vehicle's stay, with a fixed hourly rate, and facilitates the removal of vehicles from the lot, providing the total charge incurred. The user interface is accessible via a web browser, offering a straightforward way to interact with the system. The vehicle data is maintained in an in-memory list, which simplifies the management and retrieval of vehicle information. This basic structure can be expanded into a more comprehensive parking management solution, potentially including features like user authentication and database integration.

Vehicle Entry Management: Allow users to register a vehicle's entry into the parking lot, including its type (Car or Bike), plate number, and arrival time.

Display Current Vehicles: Provide a way to view all vehicles currently parked in the lot.

Count Vehicles: Keep track of the total number of cars and bikes currently in the parking lot.
Vehicle Exit Management: Allow users to record a vehicle's departure, calculate the parking fee based on the duration of stay, and remove the vehicle from the list.

Calculate Parking Charges: Implement logic to calculate the parking fee based on the duration of stay at a rate of Rs. 50 per hour.

User Interface: Use render_template to potentially serve a frontend interface (though not implemented in the provided code).

API Endpoints: Provide RESTful API endpoints for interacting with the system, including adding vehicles, showing all vehicles, counting total vehicles, and removing vehicles.

The system aims to offer a simple yet functional approach to managing a small parking lot, suitable for educational purposes or as a base for further development.

CHAPTER 2

BACKGROUND STUDY

2.1 Software Specification

The Vehicle Parking management System is a web application designed to allow users to manage vehicle parking efficiently. Following are the softwares used in this project

PYTHON (FLASK,JSON)

PYTHON:

- Python is used for server-side programming, handling backend logic, and managing routes. In this program, Python manages vehicle data, parking charges, and APIs.
- Python, with its simple syntax and readable structure, allows developers to quickly write and understand code. This is especially useful for building prototypes like the parking system.
- Python runs on multiple operating systems, so the application can be developed and tested on different platforms without major changes to the code.
- Python has libraries like datetime that help manage time-based operations, such as calculating the duration of parking in your system, without needing to implement complex logic from scratch.

FLASK,JSON:

- Flask is a lightweight web framework used for developing web applications in Python. It helps to define routes (/addVehicle, /showVehicles, etc.), handle HTTP requests, and render HTML templates.
- Flask, a lightweight and flexible web framework, makes it easy to create APIs, handle requests, and serve HTML templates. It's well-suited for projects that don’t require complex structures, which speeds up the development of web applications.
- Since Flask is a Python-based framework, it can run on any operating system that supports Python, such as Windows, macOS, or Linux.

- Data is exchanged between the frontend and backend in JSON format. For example, vehicle information is sent to the backend using a POST request as JSON, and the backend responds with JSON-formatted messages.

- flask.jsonify: Converts Python dictionaries into JSON responses, allowing the frontend to easily consume data.
- Python's built-in support for JSON allows easy handling of API requests and responses, which is essential when managing data like vehicle information.
- Python (via Flask) easily integrates with frontend technologies like HTML, CSS, and JavaScript, as shown in the example where the user interface communicates with the backend APIs.
- As the application grows in complexity, Python's modular approach (with libraries like SQLAlchemy for databases or Redis for caching) allows the system to scale efficiently.
- These features make Python an excellent choice for both building and extending this vehicle parking system, providing ease of development, flexibility.

HTML,CSS,JAVASCRIPT:

HTML:

- HTML provides the foundational structure of web pages. It allows developers to define elements like headings, paragraphs, forms, buttons, and other content elements.
- HTML has a simple, tag-based syntax that is easy to understand and write. This allows for quick development of basic web pages.
- HTML5 introduces semantic elements like <header>, <footer>, <article>, and <section>, making the content more accessible and improving SEO (Search Engine Optimization).
- HTML is supported by all modern browsers, ensuring that web pages are rendered properly across different platforms

CSS:

- CSS is responsible for the look and feel of the web pages. It controls the colors, fonts, spacing, layout, and overall presentation, making the application visually appealing.

- CSS separates the content (HTML) from the design, making it easier to maintain and update the design without affecting the structure of the page.

- CSS enables responsive web design through media queries, allowing web pages to adapt to different screen sizes and devices (mobile, tablet, desktop).
- With CSS, developers can ensure consistent styling across multiple web pages by defining global styles, which are applied throughout the application.
- CSS allows for advanced customization with features like transitions, animations, and grid/flexbox layouts to create dynamic and interactive designs.

JAVASCRIPT:

- JavaScript allows the creation of interactive elements on the web page, such as buttons, form validation, dynamic content updates, and handling events (e.g., clicks, scrolls).
- JavaScript can manipulate the DOM (Document Object Model), which allows the content of the page to change dynamically without reloading the entire page (e.g., adding vehicles to the list or showing total vehicles in your system).With features like fetch and XMLHttpRequest, JavaScript enables asynchronous communication with the server (AJAX), allowing for real-time updates (such as adding or removing vehicles without refreshing the page).
- JavaScript has a rich ecosystem of libraries (like jQuery) and frameworks (like React, Vue, and Angular) that simplify complex tasks and speed up development.
- Modern JavaScript runs seamlessly on all major browsers, making it an ideal language for client-side scripting.
- JavaScript can be used to handle real-time interactions, such as enabling auto-updating vehicle counts or handling parking charges without delays.

2.2 Key Features of the Project

Uses of Python in the Code

Web Framework (Flask): Python is used to build the web application with the Flask framework. Flask is lightweight and easy to use, making it suitable for small to medium-sized applications. It provides routing, request handling, and template rendering capabilities.

Data Management: Python is used to manage the data related to vehicles, such as storing the vehicle type, plate number, arrival time, and date. The data is stored in a list called vehicles.

Charge Calculation: Python functions are used to calculate parking charges based on the time duration a vehicle stays in the parking lot. The calculate_charge function determines the duration of the stay and calculates the charge accordingly.

API Endpoints: Python is used to define various API endpoints (e.g., /addVehicle, /showVehicles, /totalVehicles, /deleteVehicle) that allow users to interact with the system by adding vehicles, viewing all vehicles, getting the total count of vehicles, and removing vehicles.

Advantages of Using Python

Simplicity and Readability: Python's syntax is clear and concise, making it easier for developers to write and maintain the code. This improves productivity and reduces the likelihood of errors.

Extensive Libraries and Frameworks: Python has a rich ecosystem of libraries and frameworks, such as Flask for web development. These libraries provide pre-built functionalities, saving time and effort.

Rapid Development: Python's simplicity and the availability of libraries allow for rapid development and prototyping. This makes it an excellent choice for quickly building and deploying applications.

Cross-Platform: Python is a cross-platform language, meaning the same code can run on different operating systems without modification.

Community Support: Python has a large and active community, providing extensive support, documentation, and third-party packages. This makes it easier to find solutions to common problems and integrate new features.

Versatility: Python is a versatile language that can be used for various applications, including web development, data analysis, machine learning, and more. This versatility allows developers to use a single language for different parts of a project.

HTML creates the structure and presentation of the web page, allowing users to interact with the system through a graphical user interface. Here are the uses and advantages of HTML in this code:

Uses of HTML in the Code

Structure and Layout: HTML provides the basic structure of the web page. Elements like , <h1>, <label>, <input>, <button>, and <p> are used to organize the content, such as the form for adding vehicles, displaying vehicle information, and showing total counts.

Forms and User Input: HTML form elements like <input> and <select> allow users to input data, such as the vehicle type, number, arrival time, and date. These inputs are then sent to the server for processing.

Styling and Aesthetics: The <style> block defines the CSS rules for the page's appearance, including the layout, colors, fonts, and other visual elements. This makes the application more user-friendly and visually appealing.

Dynamic Content Display: The element serves as a container to dynamically display the output, such as the list of vehicles or messages about vehicle departures, using JavaScript.

Interactivity: HTML integrates with JavaScript to handle user interactions, such as clicking buttons to add vehicles, show vehicles, or calculate total counts. This interactivity is crucial for a responsive and user-friendly web application.

Advantages of Using HTML

Ease of Use and Learning: HTML is relatively simple and easy to learn, making it accessible for developers of all skill levels. It provides a straightforward way to define the structure of web content.

Wide Browser Support: HTML is universally supported by all web browsers, ensuring that web pages can be viewed and interacted with across different platforms and devices.

Separation of Content and Presentation: HTML provides the structure and content of a web page, while CSS handles the presentation. This separation allows for easier maintenance and updates, as changes to styling don't require modifications to the HTML structure.

Accessibility: HTML includes features that improve accessibility, such as semantic tags and attributes. This helps create web pages that are accessible to users with disabilities, such as those using screen readers.

SEO Optimization: Proper use of HTML elements, like headings (<h1>, <h2>, etc.), and semantic tags improves the page's search engine optimization (SEO), making it easier for search engines to index and rank the content.

Responsive Design: HTML, in combination with CSS and JavaScript, allows for responsive design. This means the web page can adapt to different screen sizes and devices, providing a consistent user experience.

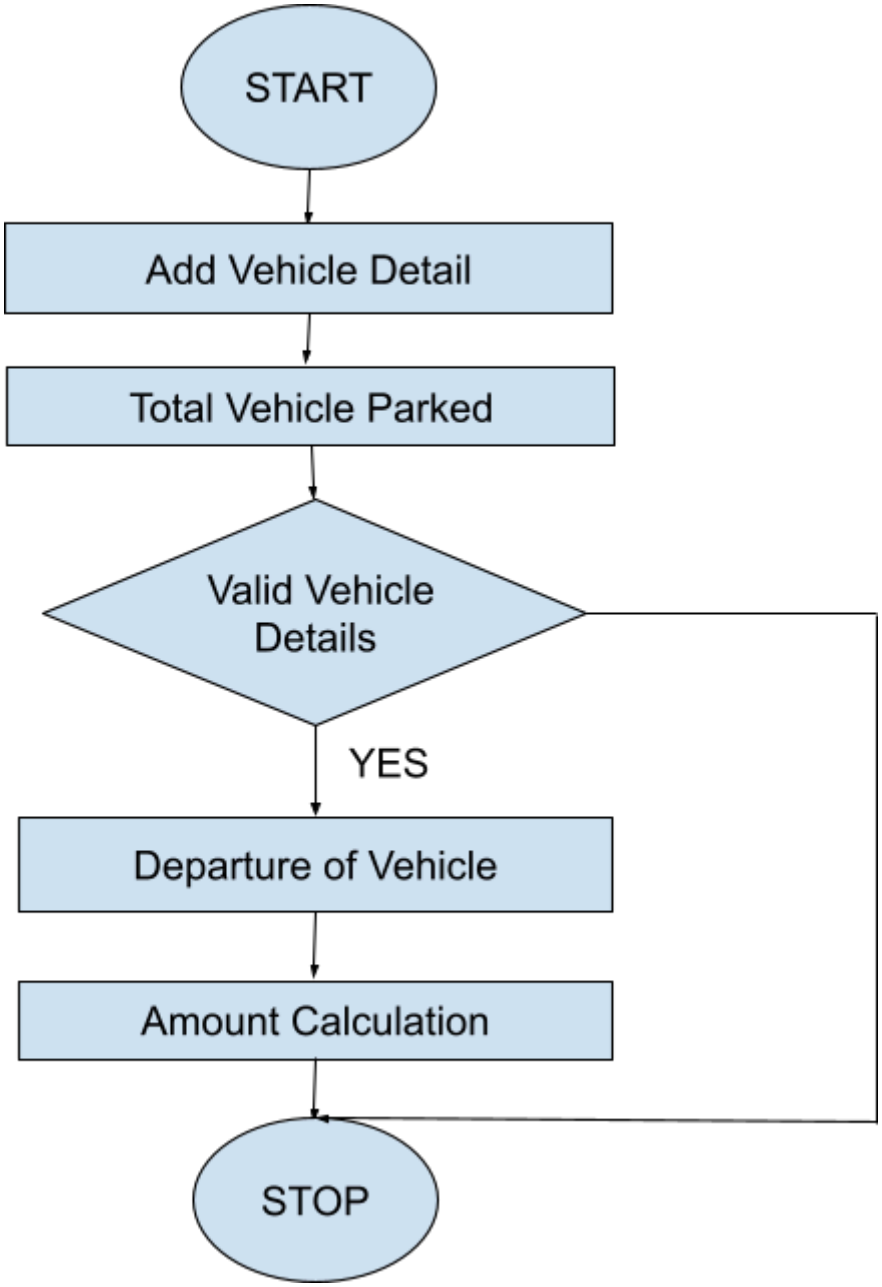
Integration with Other Technologies: HTML easily integrates with other web technologies like CSS for styling, JavaScript for interactivity, and frameworks like Bootstrap for responsive design.

Overall, HTML is a fundamental technology for web development, providing the structure and basic functionality of web pages. It allows for the creation of interactive, accessible, and visually appealing applications like the Vehicle Parking Reservation System

CHAPTER 3

SYSTEM DESIGN

3.1 System Flow Digram



CHAPTER 4

MODULES AND DESCRIPTION

Vehicle parking system project is been divided into 5 different modules:

1. Module 1- Vehicle in (Add vehicle)
2. Module 2- Total Number of vehicle parked
3. Module 3- Details of the Vehicle
4. Module 4- Vehicle out (Departure of the Vehicle)
5. Module 5- Amount Calculation

Vehicle in Module:

The module handles the process of registration of vehicles entering the parking lot . It collects the necessary information like vehicle type (car , bike) , time of the vehicle entry , vehicle number , date of vehicle entry.

Total Number of Vehicles Parked:

The module handles the process of the total number of vehicles parked in the parking lot . It contains the information like total number of vehicles parked , total number of car parked , total number of bike parked in the parking lot

Details of the Vehicles:

In this module it contains the details of all vehicles parked in the parking lot . It contains information like vehicle number , date and the time of the vehicle parked in the parking lot.

Vehicle Out Modules :

It manages the process of vehicles leaving the parking lot . It records the details like vehicle number to find the details of the vehicle , the time of exit.

Amount Calculation:

Calculate the parking fees according to the entry and exit time . As per the hours the vehicles have been parked the amount will be calculated and shown during the departure of the vehicle. It will charge 50 Rs per hour.

CONCLUSION

The code provides a basic but functional vehicle parking management system. It includes functionalities for adding vehicles, displaying them, and calculating charges upon departure. The use of Flask as a backend and a simple HTML/JavaScript front-end allows for a straightforward user interface. However, the current implementation lacks data persistence, as the vehicles are stored in a Python list that is reset every time the server restarts. For a real-world application, a database should be used to store vehicle information persistently. Additionally, features like user authentication, advanced charge calculations, and a more sophisticated UI could enhance the system.

APPENDICES

a. References

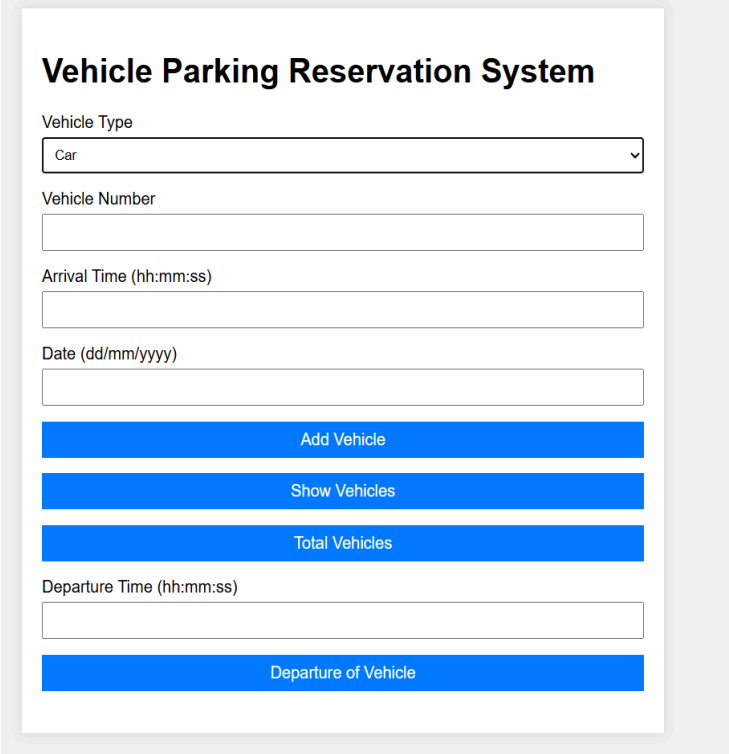
- Software engineering: McGraw-Hill Education, 2 Penn Plaza, New York. Copyright © 2020
- Python: Mark Lutz , O'Reilly Media, February 2014.
- Html: Elizabeth Robson and Eric Freeman.Head First HTML and CSS was published in 2012

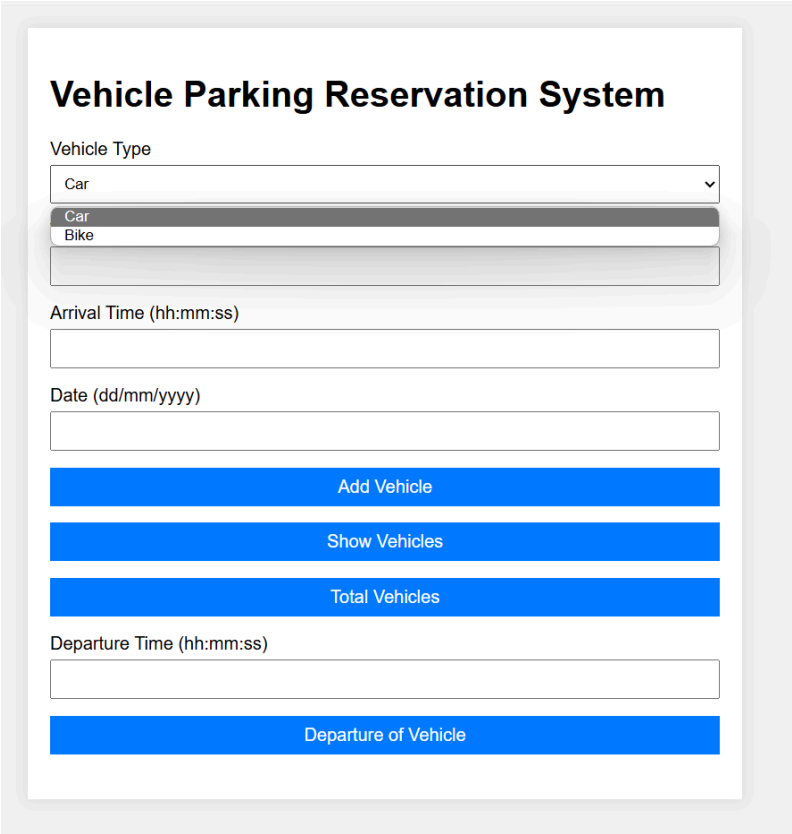
mheducation.com/highered

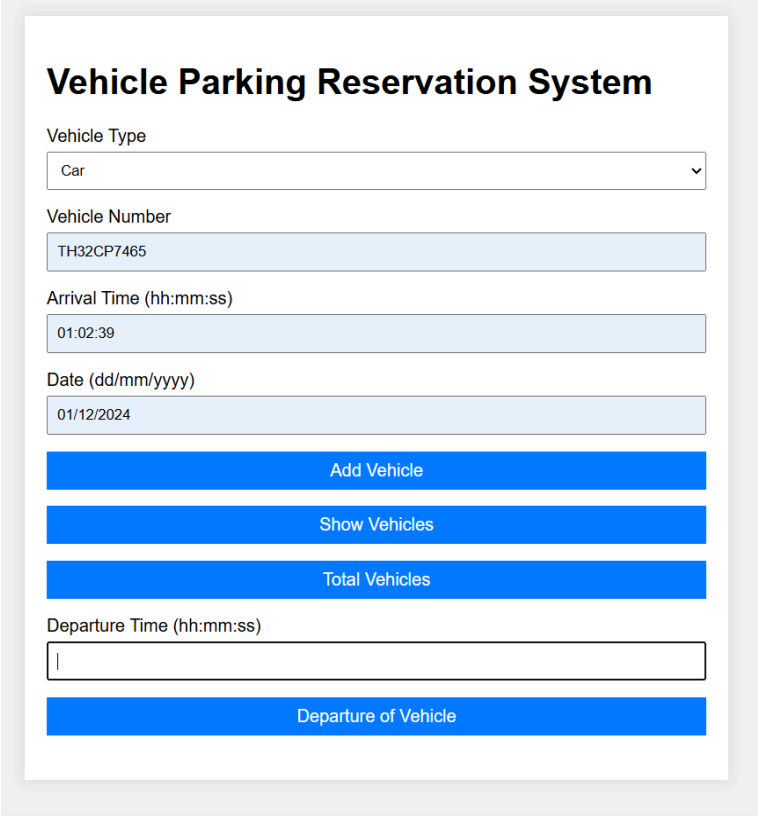
https://www.geeksforgeeks.org/python-programming-language-tutorial/

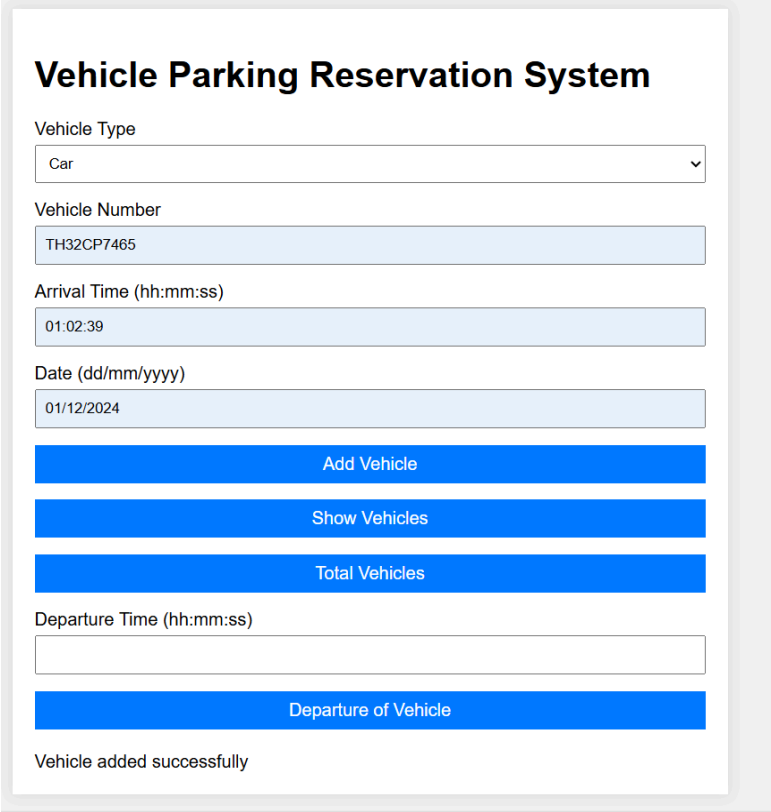
https://www.geeksforgeeks.org/html-tutorial/

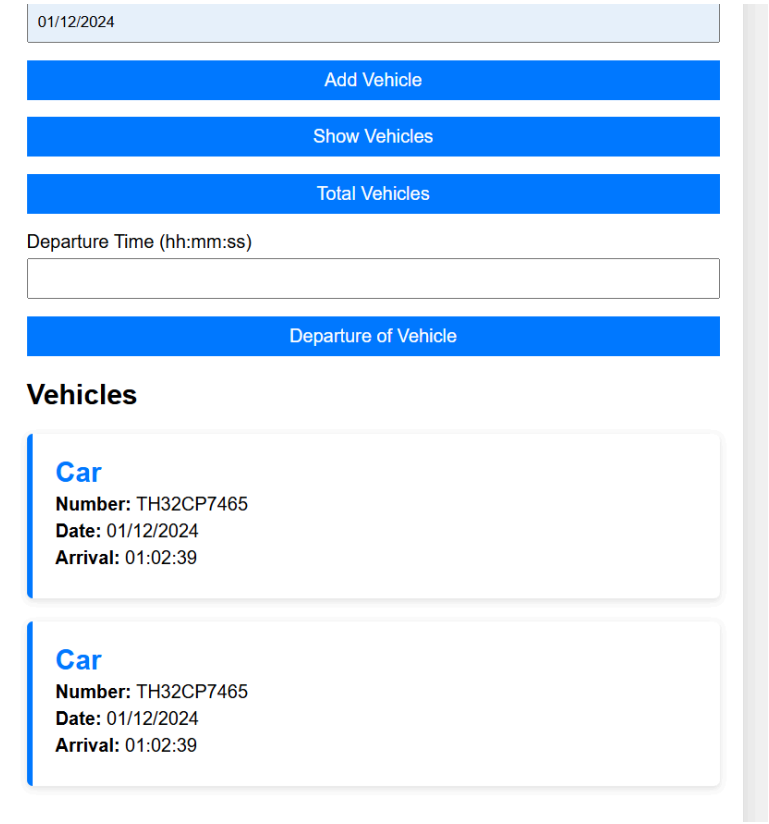
b. Screenshots

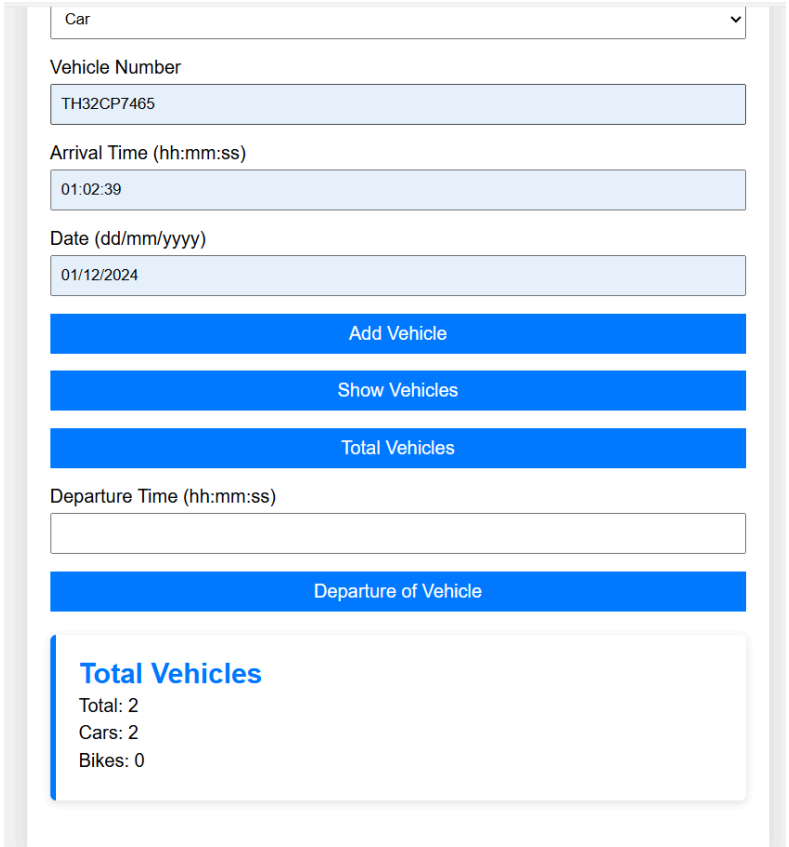


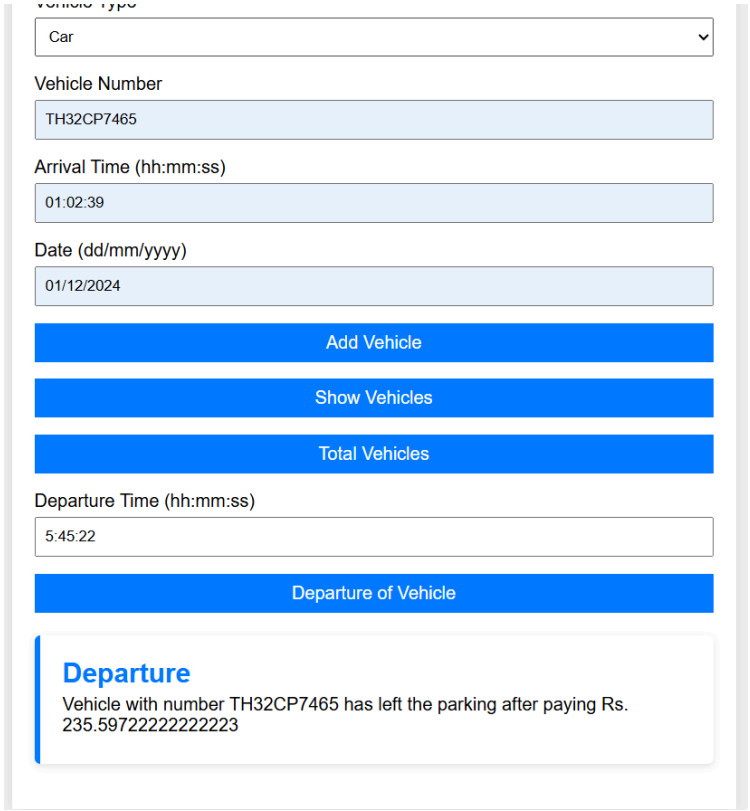


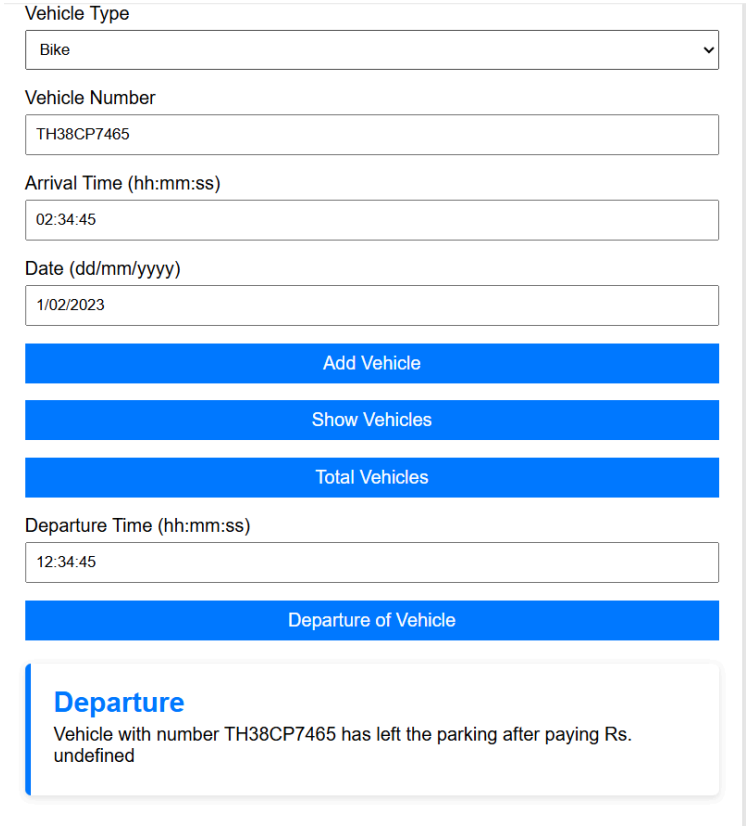

Vehicle Parking Reservation System
Vehicle Type
Car
Vehicle Number
TH32CP7465
Arrival Time (hh:mm:ss)
01:02:39
Date (dd/mm/yyyy)
01/12/2024
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
Departure of Vehicle


Vehicle Parking Reservation System
Vehicle Type
Car
Vehicle Number
TH32CP7465
Arrival Time (hh:mm:ss)
01:02:39
Date (dd/mm/yyyy)
01/12/2024
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
Departure of Vehicle
Vehicle added successfully


01/12/2024
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
Departure of Vehicle
Vehicles
Car
Number: TH32CP7465
Date: 01/12/2024
Arrival: 01:02:39
Car
Number: TH32CP7465
Date: 01/12/2024
Arrival: 01:02:39


Car
Vehicle Number
TH32CP7465
Arrival Time (hh:mm:ss)
01:02:39
Date (dd/mm/yyyy)
01/12/2024
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
Departure of Vehicle
Total Vehicles
Total: 2
Cars: 2
Bikes: 0


Car
Vehicle Number
TH32CP7465
Arrival Time (hh:mm:ss)
01:02:39
Date (dd/mm/yyyy)
01/12/2024
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
5:45:22
Departure of Vehicle
Departure
Vehicle with number TH32CP7465 has left the parking after paying Rs. 235.59722222222223


Vehicle Type
Bike
Vehicle Number
TH38CP7465
Arrival Time (hh:mm:ss)
02:34:45
Date (dd/mm/yyyy)
1/02/2023
Add Vehicle
Show Vehicles
Total Vehicles
Departure Time (hh:mm:ss)
12:34:45
Departure of Vehicle
Departure
Vehicle with number TH38CP7465 has left the parking after paying Rs. undefined

c. Sample Code

Python (Backend code)

```
from flask import Flask, request, jsonify, render_template
from datetime import datetime

app = Flask(_name_)

vehicles = []

# Helper function to calculate parking charges
def calculate_charge(arrival_time, departure_time):
    time_format = "%H:%M:%S"
    arrival = datetime.strptime(arrival_time, time_format)
    departure = datetime.strptime(departure_time, time_format)
    duration = (departure - arrival).seconds / 3600  # Convert to hours
    return 50 * duration  # Assuming Rs. 50 per hour

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

@app.route('/addVehicle', methods=['POST'])
def add_vehicle():
    data = request.json
    vehicle = {
        "type": "Car" if data['type'] == "1" else "Bike",
        "pltno": data['pltno'],
        "arrive": data['arrive'],
```

```
        "date": data['date']
    }
    vehicles.append(vehicle)
    return jsonify({"message": "Vehicle added successfully"}), 201

@app.route('/showVehicles', methods=['GET'])
def show_vehicles():
    return jsonify(vehicles), 200

@app.route('/totalVehicles', methods=['GET'])
def total_vehicles():
    total_cars = len([v for v in vehicles if v['type'] == 'Car'])
    total_bikes = len([v for v in vehicles if v['type'] == 'Bike'])
    return jsonify({"total": len(vehicles), "cars": total_cars, "bikes": total_bikes}), 200



@app.route('/deleteVehicle', methods=['POST'])
def delete_vehicle():
    data = request.json
    for vehicle in vehicles:
        if vehicle['pltno'] == data['pltno'] and vehicle['type'] == ("Car" if data['type'] == "1" else
"Bike"):
            charge = calculate_charge(vehicle['arrive'], data['depart'])
            vehicles.remove(vehicle)
              return jsonify({"message": f"Vehicle with number {data['pltno']} has left the parking",
"charge": charge}), 200
    return jsonify({"message": "Vehicle not found"}), 404

if _name_ == '_main_':
    app.run(debug=True)
```

HTML (frontend code)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vehicle Parking Reservation System</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            margin: 0;
            padding: 20px;
        }
        .container {
            max-width: 600px;
            margin: auto;
            padding: 20px;
            background-color: white;
            box-shadow: 0px 0px 10px rgba(0,0,0,0.1);
        }
        .form-group {
            margin-bottom: 15px;
        }
        .form-group label {
            display: block;
            margin-bottom: 5px;
```

```
}
.form-group input, .form-group select, .form-group button {
    width: 100%;
    padding: 8px;
    box-sizing: border-box;
}
.form-group button {
    background-color: #007BFF;
    border: none;
    color: white;
    font-size: 16px;
    cursor: pointer;
}
.form-group button:hover {
    background-color: #0056b3;
}
.output {
    margin-top: 20px;
}
.card {
    background-color: #fff;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
    margin-bottom: 20px;
    padding: 20px;
    border-left: 5px solid #007BFF;
}
.card h3 {
    margin: 0;
    font-size: 1.5em;
```

```
            color: #007BFF;
        }
        .card p {
            margin: 5px 0;
        }
    </style>
</head>
<body>

<div class="container">
    <h1>Vehicle Parking Reservation System</h1>
    <div class="form-group">
        <label for="type">Vehicle Type</label>
        <select id="type">
            <option value="1">Car</option>
            <option value="2">Bike</option>
        </select>
    </div>
    <div class="form-group">
        <label for="pltno">Vehicle Number</label>
        <input type="text" id="pltno">
    </div>
    <div class="form-group">
        <label for="arrive">Arrival Time (hh:mm:ss)</label>
        <input type="text" id="arrive">
    </div>
    <div class="form-group">
        <label for="date">Date (dd/mm/yyyy)</label>
        <input type="text" id="date">
    </div>
```

```
    <div class="form-group">
        <button onclick="addVehicle()">Add Vehicle</button>
    </div>
    <div class="form-group">
        <button onclick="showVehicles()">Show Vehicles</button>
    </div>
    <div class="form-group">
        <button onclick="totalVehicles()">Total Vehicles</button>
    </div>
    <div class="form-group">
        <label for="depart">Departure Time (hh:mm:ss)</label>
        <input type="text" id="depart">
    </div>
    <div class="form-group">
        <button onclick="deleteVehicle()">Departure of Vehicle</button>
    </div>
    <div class="output" id="output"></div>
</div>

<script>
    function addVehicle() {
        const type = document.getElementById('type').value;
        const pltno = document.getElementById('pltno').value;
        const arrive = document.getElementById('arrive').value;
        const date = document.getElementById('date').value;

        fetch('/addVehicle', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
```

```
        },
        body: JSON.stringify({ type, pltno, arrive, date })
    })
    .then(response => response.json())
    .then(data => {
        document.getElementById('output').innerText = 'Vehicle added successfully';
    })
    .catch(error => console.error('Error:', error));
}

function showVehicles() {
    fetch('/showVehicles')
    .then(response => response.json())
    .then(data => {
        let output = '<h2>Vehicles</h2>';
        data.forEach(vehicle => {
            output += `
            <div class="card">
                <h3>${vehicle.type}</h3>
                <p><strong>Number:</strong> ${vehicle.pltno}</p>
                <p><strong>Date:</strong> ${vehicle.date}</p>
                <p><strong>Arrival:</strong> ${vehicle.arrive}</p>
            </div>`;
        });
        document.getElementById('output').innerHTML = output;
    })
    .catch(error => console.error('Error:', error));
}

function totalVehicles() {
```

```
    fetch('/totalVehicles')
    .then(response => response.json())
    .then(data => {
        document.getElementById('output').innerHTML = `
            <div class="card">
                <h3>Total Vehicles</h3>
                <p>Total: ${data.total}</p>
                <p>Cars: ${data.cars}</p>
                <p>Bikes: ${data.bikes}</p>
            </div>`;
    })
    .catch(error => console.error('Error:', error));
}



function deleteVehicle() {
    const type = document.getElementById('type').value;
    const pltno = document.getElementById('pltno').value;
    const depart = document.getElementById('depart').value;

    fetch('/deleteVehicle', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({ type, pltno, depart })
    })
    .then(response => response.json())
    .then(data => {
```

```
        document.getElementById('output').innerHTML = `
            <div class="card">
                <h3>Departure</h3>
                        <p>Vehicle with number ${pltno} has left the parking after paying Rs. ${data.charge}</p>
            </div>`;
        })
        .catch(error => console.error('Error:', error));
    }
</script>

</body>
</html>
```