

BUS TICKET RESERVATION SYSTEM

A MINI PROJECT REPORT

Submitted by

ESPIN SHALO S P 220701072

GUHANRAJ P 220701078

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 - 24

BONAFIDE CERTIFICATE

Certified that this project report “**BUS TICKET RESERVATION SYSTEM**” is the bonafide work of “**ESPIN SHALO S P (220701072), GUHANRAJ P (220701078),**” who carried out the project work under my supervision.

Submitted for the Practical Examination held on ____

SIGNATURE

Dr.R.SABITHA
Professor and II Year Academic Head
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam,
Chennai - 602105

SIGNATURE

Ms.Dr.DHARANI DEVI
Associate Professor ,
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam,
Chennai - 602 105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The bus reservation system developed using Python and Flask serves as a comprehensive platform for booking intercity bus tickets online. Utilizing Flask's lightweight and efficient framework, the system integrates seamlessly with PostgreSQL to store and manage bus routes, schedules, and passenger bookings securely. The front-end interface, crafted with HTML, CSS, and JavaScript, offers users an intuitive experience for searching buses, selecting seats, and confirming bookings. User authentication ensures data security and personalized interactions throughout the booking process. Python's versatility powers backend functionalities, including data retrieval, validation, and processing, ensuring real-time updates and seamless user experience. The system supports dynamic seat selection and booking, reflecting contemporary web application standards. Its modular architecture allows for scalability and easy integration of additional features, ensuring adaptability to evolving user demands. This project showcases Python's robust capabilities in web development, providing a reliable solution for modern transportation needs.

TABLE OF CONTENTS

1.INRODUCTION

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

2.SURVEY OF TECHNOLOGIES

2.1SOFTWAREDESCRIPTION.

2.2LANGUAGES

2.3 ELEPHANT SQL

2.4 PYTHON

3.REQUIREMENTS AND ANALYSIS

3.1REQUIREMENT SPECIFICATION

3.2HARDWAREANDSOFTWARE REQUIREMENTS

3.3ARCHITECTURE DIAGRAM

3.4USER DIAGRAM

3.5NORMALIZATION

4. PROGRAM CODE

5. RESULTS AND DISCUSSION

6.CONCLUSION

7. REFERENCES

CHAPTER 1

1.1 INTRODUCTION

The bus reservation system is a Python-based web application designed to streamline and enhance the booking experience for bus travelers. Leveraging Flask and PostgreSQL, it provides a secure and efficient platform for users to search routes, check seat availability, and make reservations online. This project aims to address the complexities and inefficiencies in traditional ticketing systems by offering a modern, user-friendly interface and robust backend functionality. By integrating real-time data processing and secure authentication mechanisms, the system ensures reliability, data integrity, and seamless transactions. Future enhancements may include dynamic pricing, real-time tracking, and advanced reporting features, positioning it as a scalable solution for the transportation industry.

1.2 OBJECTIVES

Efficient Bus Search: Enable users to search for buses based on location and travel date, enhancing user convenience.

Real-Time Bus Availability: Display real-time bus availability, schedules, and seat information to users.

User Authentication: Implement secure user authentication and session management for personalized experiences.

Booking and Seat Selection: Facilitate seamless booking and seat selection processes for users.

Passenger Details Management: Capture and manage passenger details during booking for accurate record-keeping.

Confirmation and Payment: Provide a smooth booking confirmation process with integrated payment functionalities.

User Interaction: Enhance user interaction with dynamic seat selection interfaces and booking confirmations.

Error Handling and Alerts: Implement robust error handling and alerts to guide users through the booking process.

Responsive Design: Ensure a responsive design for optimal user experience across devices.

Scalability and Performance: Design the system to handle increasing user traffic and maintain performance.

Security Measures: Employ stringent security measures to protect user data and transactions.

Logging and Monitoring: Implement logging and monitoring functionalities to track system performance and user activities.

1.3 MODULES

User Authentication and Session Management

Handles user login, logout, and session management.
Personalizes user experience and manages access to booking features.

Bus Search and Display

Enables users to search for buses based on location and travel date.
Retrieves bus details from a database and displays search results.

Seat Selection and Booking

Allows users to view available seats and select seats for booking.
Integrates seat selection functionality using JavaScript.

Passenger Details Management

Collects passenger details after seat selection and validates form submission.
Stores passenger information and calculates total booking price.

Booking Confirmation and Payment Processing

Confirms bus booking with passenger and bus details.
Provides a summary of the booking and initiates payment processing.

Error Handling and Logging

Implements basic error handling for database connections and queries.
Logs errors to standard output for debugging purposes.

Responsive Frontend Design

Utilizes responsive design principles for user-friendly interface across devices.
Enhances accessibility and usability with structured HTML templates and CSS styles.

Security Measures

Protects sensitive data with session management and secure data transmission.
Ensures access control and implements Flask's security features.

2.1 SOFTWARE DESCRIPTION

The Bus Reservation System is a web-based application built using Python and Flask, designed to facilitate the online booking of bus tickets. The system allows users to search for available buses, select seats, enter passenger details, and confirm bookings with ease.

Key Features:

User Authentication:

Secure login and logout features with session management to ensure personalized and protected access.

Bus Search and Booking:

Users can search for buses by specifying departure and destination locations along with the travel date.

Displays search results with detailed information about each bus, including route, schedule, and available seats.

Interactive Seat Selection:

Users can view and select available seats through an interactive interface, making the booking process intuitive.

Passenger Details Management:

Collects and validates passenger information, including name, email, age, and gender, before confirming the booking.

Booking Confirmation:

Provides a summary of the booking details, allowing users to review and confirm their reservations.

Database Integration:

Utilizes PostgreSQL to manage and store bus schedules, user information, and booking details, ensuring efficient data handling and retrieval.

Responsive Design:

Ensures accessibility and usability across various devices, including desktops, tablets, and smartphones.

Security:

implements secure session management and data protection measures to safeguard user information.

The Bus Reservation System is a web-based application built using Python and Flask, designed to facilitate the online booking of bus tickets. The system allows users to search for available buses, select seats, enter passenger details, and confirm bookings with ease.

2.2 LANGUAGES

Python:

- Primary language for backend development.
- Handles server-side operations, database interactions, and session management using Flask.
- Powers functionalities like user authentication, bus search, and seat booking.

HTML:

- Structures the content and layout of the web pages.
- Provides the framework for the user interface, including forms, navigation menus, and display areas for search results and seat selection.

CSS:

styles the HTML elements to enhance the visual appeal and usability of the application.

Ensures a responsive and consistent design across different devices and screen sizes.

JavaScript:

- Adds interactivity to the web pages, allowing dynamic seat selection, form validation, and modal dialogues for login.
- Enhances the user experience by enabling real-time updates and feedback.

SQL:

- Utilized for managing the PostgreSQL database.
- Executes queries to store and retrieve data related to bus schedules, routes, user accounts, and bookings.

These languages and technologies work together to create a robust, efficient, and user-friendly bus reservation system, offering a seamless experience for users booking their bus tickets online.

2.3 ELEPHANT SQL

ElephantSQL is a cloud-based service providing fully managed PostgreSQL databases, simplifying database setup, maintenance, and operation.

Key Features:

Fully Managed Service: Automatic handling of backups, updates, and monitoring.

Scalability: Plans range from small development databases to large production environments.

High Availability: Automated failover and data replication for minimal downtime.
Automated Backups: Regular backups with point-in-time restore capabilities.

Security: Encryption at rest and in transit to protect data.

Ease of Use: User-friendly interface for easy database management.

Usage in Bus Reservation System:

In the Bus Reservation System project, ElephantSQL manages data related to bus schedules, routes, user accounts, and bookings, ensuring reliability, scalability, and high performance for handling large volumes of traffic and data.

2.4 PYTHON

Python is crucial in developing the Bus Reservation System, providing a robust platform for various functionalities. Here's how Python is used in the project:

Backend Development:

Flask Framework: Manages server-side operations, routing, form data processing, and database interactions.

Database Management:

psycopg2 Library: Facilitates interaction with the PostgreSQL database, handling queries and data storage efficiently.

Session Management:

Flask-Session: Manages user sessions, ensuring secure and personalized experiences for logged-in users.

Form Processing:

Data Handling: Processes and validates user input for searching buses, booking tickets, and entering passenger details.

Dynamic Content Rendering:

Jinja2 Templating: Renders dynamic HTML content, displaying real-time information like search results and seat layouts.

Error Handling:

Exception Management: Ensures stability and user-friendly error messages through robust exception handling.

Scalability and Maintenance:

3. REQUIREMENTS AND ANALYSIS

Functional Requirements:

User Registration and Login:

Users should be able to register, log in, and log out.

Bus Search:

Users can search for buses based on from location, to location, and travel date.

Bus Details Display:

Display detailed information about available buses, including routes, schedules, and prices.

Seat Selection:

Users can view and select available seats on a bus.

Passenger Information:

Collect passenger details such as name, email, age, and gender during booking.

Booking Confirmation:

Confirm the booking and display a summary of the selected seats and total price.

User Session Management:

Maintain user sessions to provide personalized experiences.

Non-Functional Requirements:

Performance:

The system should respond quickly to user inputs and database queries.

Scalability:

The system should handle an increasing number of users and transactions.

Security:

Protect user data and sessions through secure authentication and data encryption.

Usability:

Provide an intuitive and user-friendly interface.

Reliability:

Ensure the system is consistently available and operational.

Analysis:

User Analysis:

Admin Users:

Manage bus schedules, routes, and booking records.

Regular Users:

Search for buses, book seats, and view their bookings.

System Analysis:**Database Schema:**

Tables for users, buses, routes, seats, and bookings.

Data Flow:

User inputs are processed by the Flask server, which interacts with the PostgreSQL database to retrieve and store data.

Security Mechanisms:

Use Flask-Session for secure session management.

Encrypt sensitive data and ensure secure communication between client and server.

Technical Feasibility:**Language and Frameworks:**

Python with Flask for backend development.

psycopg2 for PostgreSQL database interactions.

Hosting:

Deploy the application on a suitable web server or cloud platform.

Risk Analysis:**Data Breach:**

Implement strong security measures to protect user data.

System Downtime:

Ensure high availability through robust server infrastructure.

User Error:

Provide clear instructions and error messages to guide users.

By conducting thorough requirements and analysis, the project aims to develop a reliable, efficient, and user-friendly bus reservation system that meets both user expectations and technical standards.

3.1 REQUIREMENT SPECIFICATION

Functional Requirements:

User Registration and Authentication:

Users can create accounts and log in/out securely.

Bus Search and Listing:

Users can search for buses by locations and date.

Search results display bus details including name, type, route, times, duration, and fare.

Seat Selection:

Users can view and select available seats on a bus.

Seats may have gender-specific reservations.

Passenger Information and Booking

Users must provide passenger details for each seat.

Users can confirm and review bookings before submission.

Booking Management

Users can view and cancel their bookings.

Non-Functional Requirements

Performance

The system must handle multiple requests with minimal response time.

Security

Secure authentication and data encryption are essential.

Usability

The interface should be intuitive and user-friendly.

Scalability

The system should support growing user numbers without performance issues.

Reliability

The system should be available 24/7 with regular backups.

System Requirements

Hardware

Server: Minimum 4 GB RAM, 100 GB HDD, multi-core processor.

Client: Any device with internet and a web browser.

Software

Server: Linux/Windows OS, PostgreSQL database, compatible web server.

Programming: Python, Flask framework, psycopg2, Flask-Session libraries.

Database Requirements

User Table, Bus Table, Seat Table, Booking Table, Passenger Table,

Interface Requirements

User Interface: Responsive design for various devices, easy navigation.

Admin Interface: Manage buses, routes, schedules, and bookings.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

Server:

Processor: Quad-core CPU or higher.

Memory: Minimum 8 GB RAM.

Storage: Minimum 250 GB SSD.

Network: High-speed internet connection.

Backup: Regular backup system for data redundancy.

Client:

Device: Desktop or laptop computer, smartphone, or tablet.

Processor: Dual-core CPU or higher.

Memory: Minimum 4 GB RAM.

Storage: Minimum 20 GB available space.

Display: Minimum resolution of 1024x768 pixels.

Network: Reliable internet connection.

Software Requirements:

Server-side:

Operating System: Linux (Ubuntu/CentOS), Windows Server, or macOS.

Web Server: Apache HTTP Server or Nginx.

Database Management System: PostgreSQL.

Programming Language: Python.

Framework: Flask (Python web framework).

Libraries/Packages:

psycopg2 for PostgreSQL database connectivity.

Flask-Session for session management.

Version Control: Git.

Client-side:

Operating System: Windows, macOS, Linux, Android, or iOS.

Web Browser: Latest versions of Chrome, Firefox, Safari, or Edge.

Text Editor/IDE:

Visual Studio Code, PyCharm, Sublime Text, or any preferred code editor.

Browser Developer Tools: For debugging and testing front-end code.

Development Tools:

Integrated Development Environment (IDE):

Visual Studio Code, PyCharm, or any preferred Python IDE.

Version Control System:

Git with a GitHub, GitLab, or Bitbucket repository for code collaboration and versioning.

Postman: For API testing and debugging.

Docker: Optional, for containerization and easier deployment.

Additional Software:

Python Packages:

Flask: For building the web application.

psycopg2: For PostgreSQL database operations.

Jinja2: For templating in Flask.

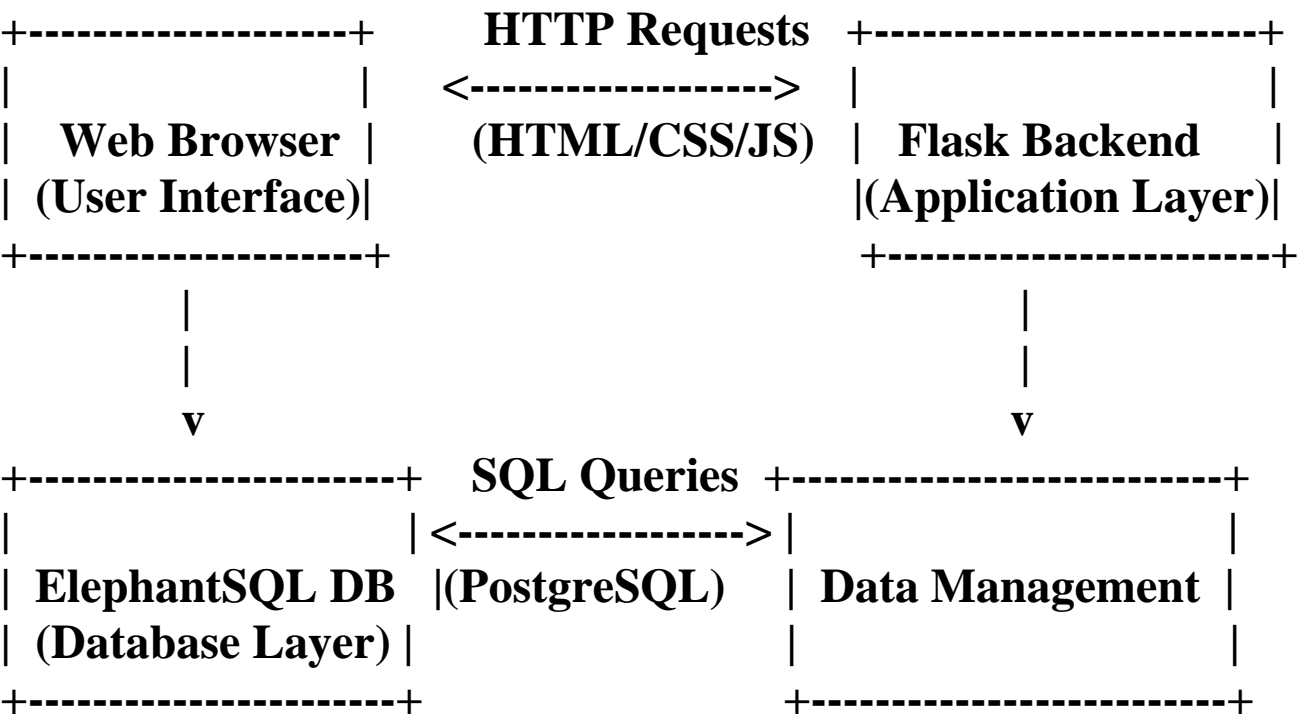
Browser Plugins:

JSON Viewer: For viewing JSON data in the browser.

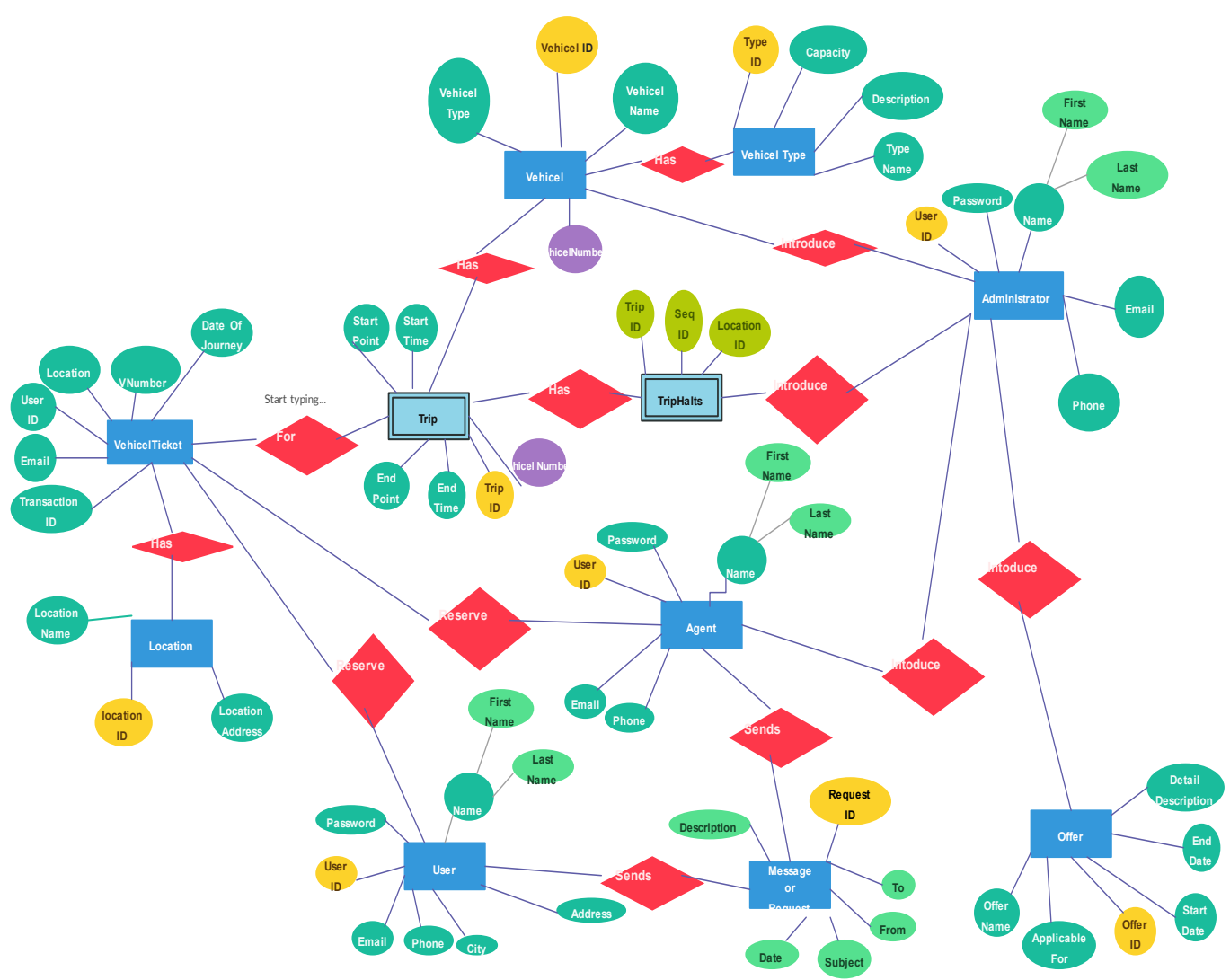
Postman Interceptor: For capturing browser requests.

These hardware and software requirements ensure the efficient development, deployment, and operation of the bus reservation system, providing a seamless experience for both users and administrators.

3.3 ARCHITECTURE DIAGRAM



3.4 ER DIAGRAM



3.5 NORMALIZATION

Step 1: Identify Entities and Attributes

1. User: user_id, username, password, email
2. Bus: bus_id, bus_number, capacity, type
3. Route: route_id, origin, destination, distance
4. Trip: trip_id, bus_id, route_id, departure_time, arrival_time, date
5. Booking: booking_id, user_id, trip_id, seat_number, booking_date
6. ActivityLog: log_id, user_id, ip_address, action, timestamp

Step 2: Apply Normalization Forms

1NF (First Normal Form)

Ensure that each table has a primary key and that all columns contain atomic values.

- User: user_id, username, password, email
- Bus: bus_id, bus_number, capacity, type
- Route: route_id, origin, destination, distance
- Trip: trip_id, bus_id, route_id, departure_time, arrival_time, date
- Booking: booking_id, user_id, trip_id, seat_number, booking_date
- ActivityLog: log_id, user_id, ip_address, action, timestamp

2NF (Second Normal Form)

Ensure that all non-key attributes are fully functional dependent on the primary key.

- User: user_id, username, password, email
- Bus: bus_id, bus_number, capacity, type
- Route: route_id, origin, destination, distance
- Trip: trip_id, bus_id, route_id, departure_time, arrival_time, date
- Booking: booking_id, user_id, trip_id, seat_number, booking_date
- ActivityLog: log_id, user_id, ip_address, action, timestamp

3NF (Third Normal Form)

Ensure that all the attributes are dependent only on the primary key and not on any other non-key attributes.

- User: user_id, username, password, email
- Bus: bus_id, bus_number, capacity, type
- Route: route_id, origin, destination, distance
- Trip: trip_id, bus_id, route_id, departure_time, arrival_time, date
- Booking: booking_id, user_id, trip_id, seat_number, booking_date
- ActivityLog: log_id, user_id, ip_address, action, timestamp

Final Schema

1. User

- user_id (Primary Key)
- username
- password
- email

2. Bus

- bus_id (Primary Key)
- bus_number
- capacity
- type

3. Route

- route_id (Primary Key)
- origin
- destination
- distance

4. Trip

- trip_id (Primary Key)
- bus_id (Foreign Key)
- route_id (Foreign Key)
- departure_time
- arrival_time
- date
-

5. Booking

- booking_id (Primary Key)
- user_id (Foreign Key)
- trip_id (Foreign Key)
- seat_number
- booking_date

6. ActivityLog

- log_id (Primary Key)
- user_id (Foreign Key)
- ip_address
- action
- timestamp

4. PROGRAM CODE

Index.Html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="icon" href="{ { url_for('static', filename='img/logi.png') } }" type="image/x-
icon">
  <link rel="shortcut icon" href="{ { url_for('static', filename='img/logi.png') } }"
type="image/x-icon">
  <title>Confirm Booking</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    .container {
      max-width: 600px;
      margin: auto;
      padding: 20px;
      border: 1px solid #ccc;
      border-radius: 10px;
    }
    .header {
      text-align: center;
      margin-bottom: 20px;
    }
    .details {
      margin-bottom: 20px;
    }
    .details h2 {
      margin: 0 0 10px 0;
    }
    .details p {
      margin: 5px 0;
    }
    .total {
      text-align: right;
      font-size: 1.2em;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="header">
      <h2>Confirm Booking</h2>
    </div>
    <div class="details">
      <h3>Details</h3>
      <p>...</p>
    </div>
    <div class="total">
      <p>...</p>
    </div>
  </div>
</body>
</html>
```

```

.confirm-button {
    display: block;
    width: 100%;
    padding: 10px;
    background-color: #007BFF;
    color: white;
    text-align: center;
    text-decoration: none;
    border-radius: 5px;
    margin-top: 20px;
}
.confirm-button:hover {
    background-color: #0056b3;
}
</style>
</head>
<body>
<div class="container">
<div class="header">
<h1>Booking Confirmation</h1>
</div>
<div class="details">
<h2>Bus Details</h2>
<p><strong>Bus Name:</strong> {{ bus_details.bus_name }}</p>
<p><strong>From:</strong> {{ bus_details.from_location }}</p>
<p><strong>To:</strong> {{ bus_details.to_location }}</p>
<p><strong>Travel Date:</strong> {{ bus_details.date }}</p>
</div>
<div class="details">
<h2>Passenger Details</h2>
{% for seat, passenger in passengers.items() %}
<p><strong>Seat {{ seat }}:</strong> {{ passenger.name }}, {{ passenger.age
}}, {{ passenger.gender }}</p>
{% endfor %}
</div>
<div class="total">
<p>Total Price: ₹{{ total_price }}</p>
</div>
<a href="/payment" class="confirm-button">Proceed to Payment</a>
</div>
</body>
</html>

```

Add_passenger.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Add Passenger</title>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
    .passenger-form {
      margin: 20px;
    }
    .passenger-form div {
      margin-bottom: 10px;
    }
    .passenger-form label {
      display: inline-block;
      width: 100px;
    }
    .passenger-form input,
    .passenger-form select {
      padding: 5px;
      width: 200px;
    }
    .passenger-form button {
      padding: 10px 20px;
      background-color: #007BFF;
      color: white;
      border: none;
      cursor: pointer;
    }
    .passenger-details {
      margin-bottom: 20px;
    }
    .seat-number {
      font-weight: bold;
    }
  </style>
</head>
<body>
  <h1>Add Passenger Details</h1>
  <form action="/confirm-booking" method="post" class="passenger-form">
    {% for seat_id in seat_ids %}
```

```

<div class="passenger-details">
  <h2>Seat <span class="seat-number">{{ seat_id }}</span></h2>
  <div>
    <label for="name_{{ seat_id }}">Name:</label>
    <input type="text" id="name_{{ seat_id }}" name="name_{{ seat_id }}"
required>
  </div>
  <div>
    <label for="email_{{ seat_id }}">Email:</label>
    <input type="email" id="email_{{ seat_id }}" name="email_{{ seat_id }}"
required>
  </div>
  <div>
    <label for="age_{{ seat_id }}">Age:</label>
    <input type="number" id="age_{{ seat_id }}" name="age_{{ seat_id }}"
required>
  </div>
  <div>
    <label for="gender_{{ seat_id }}">Gender:</label>
    <select id="gender_{{ seat_id }}" name="gender_{{ seat_id }}" required>
      <option value="male">Male</option>
      <option value="female">Female</option>
      <option value="other">Other</option>
    </select>
  </div>
</div>
{% endfor %}
<button type="submit">Confirm Booking</button>
</form>
</body>
</html>

```

Search_result.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{ from_location }} to {{ to_location }} Bus Online Tickets Booking |
ClicBus</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
  <link rel="icon" href="{{ url_for('static', filename='img/logi.png') }}" type="image/x-

```


icon">

<link rel="shortcut icon" href="{ { url_for('static', filename='img/logi.png') } }"
type="image/x-icon">

<style>

/* Add necessary styles for the bus result card */

.bus-card {
border: 1px solid #ddd;
padding: 20px;
margin: 10px 0;
border-radius: 5px;
display: flex;
justify-content: space-between;
align-items: center;
}

.bus-info {
display: flex;
flex-direction: column;
}

.bus-info div {
margin-bottom: 5px;
}

.bus-info .bus-name {
font-size: 1.5em;
font-weight: bold;
}

.bus-info .bus-type {
color: #888;
}

.bus-schedule {
text-align: center;
}

.bus-schedule div {
margin-bottom: 5px;
}

.bus-price {
text-align: right;
}

.bus-price .price {
font-size: 1.5em;
color: #E53935;
}

.view-seats {
background-color: #E53935;
color: white;
padding: 10px 20px;

```

        border-radius: 5px;
        text-decoration: none;
    }
    .navbar-right {
        float: right;
    }
    .navbar-right span {
        margin-right: 10px;
    }
</style>
</head>
<body>
<header>
    
    <nav id="navBar">
        <a href="/">Home</a>
        <a href="#">Book Tickets</a>
        <a href="#">My Bookings</a>
        <a href="#">Help</a>
        <div class="navbar-right">
            { % if username % }
            <span>Welcome, { { username } }!</span>
            <a href="{ { url_for('logout') } }">Logout</a>
            { % else % }
            <button class="login-button" id="loginBtn">Login</button>
            { % endif % }
        </div>
    </nav>
</header>
<div class="container">
    <h1>Bus Search Results</h1>
    <div class="location-info">
        <h3>{ { from_location } } &#x27F6; { { to_location } }</h3>
    </div>
    { % if buses % }
    { % for bus in buses % }
    <div class="bus-card">
        <div class="bus-info">
            <div class="bus-name">{ { bus[1] } }</div>
            <div class="bus-type">{ { bus[4] } }</div>
            <div>Route: { { bus[2] } }</div>
        </div>
        <div class="bus-schedule">
            <div>Departure: 19:00</div>
            <div>Duration: 12h 00m</div>
        </div>
    </div>
    { % endfor % }
    { % endif % }
</div>

```

```

        <div>Arrival: 07:00</div>
        <div>Date: { { bus[6].strftime("%d-%b") } }</div>
    </div>
    <div class="bus-price">
        <div class="price">INR 1800</div>
        <div>Seats available: 7</div>
        <div>Window: 3</div>
    </div>
    <div>
        <a href="{ { url_for('view_seats', bus_id=bus[0]) } }" class="view-
seats">VIEW SEATS</a>
    </div>
</div>
{ % endfor % }
{ % else % }
    <p>No buses found for your search criteria.</p>
{ % endif % }
    <a href="/">Go back</a>
</div>
</body>
</html>

```

Confirm_booking.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="{ { url_for('static', filename='img/logi.png') } }" type="image/x-
icon">
    <link rel="shortcut icon" href="{ { url_for('static', filename='img/logi.png') } }"
type="image/x-icon">
    <title>Confirm Booking</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        .container {
            max-width: 600px;
            margin: auto;

```

```
padding: 20px;
border: 1px solid #ccc;
border-radius: 10px;
}
.header {
  text-align: center;
  margin-bottom: 20px;
}
.details {
  margin-bottom: 20px;
}
.details h2 {
  margin: 0 0 10px 0;
}
.details p {
  margin: 5px 0;
}
.total {
  text-align: right;
  font-size: 1.2em;
  font-weight: bold;
}
.confirm-button {
  display: block;
  width: 100%;
  padding: 10px;
  background-color: #007BFF;
  color: white;
  text-align: center;
  text-decoration: none;
  border-radius: 5px;
  margin-top: 20px;
}
.confirm-button:hover {
  background-color: #0056b3;
}
</style>
</head>
<body>
<div class="container">
  <div class="header">
    <h1>Booking Confirmation</h1>
  </div>
  <div class="details">
    <h2>Bus Details</h2>
```

```

        <p><strong>Bus Name:</strong> {{ bus_details.bus_name }}</p>
        <p><strong>From:</strong> {{ bus_details.from_location }}</p>
        <p><strong>To:</strong> {{ bus_details.to_location }}</p>
        <p><strong>Travel Date:</strong> {{ bus_details.date }}</p>
    </div>
    <div class="details">
        <h2>Passenger Details</h2>
        {% for seat, passenger in passengers.items() %}
            <p><strong>Seat {{ seat }}:</strong> {{ passenger.name }}, {{ passenger.age
        }}, {{ passenger.gender }}</p>
        {% endfor %}
    </div>
    <div class="total">
        <p>Total Price: ₹{{ total_price }}</p>
    </div>
    <a href="/payment" class="confirm-button">Proceed to Payment</a>
</div>
</body>
</html>

```

View_seats.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="icon" href="{{ url_for('static', filename='img/logi.png') }}" type="image/x-
icon">
    <link rel="shortcut icon" href="{{ url_for('static', filename='img/logi.png') }}"
type="image/x-icon">
    <title>View Seats</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        .logo {
            width: 100px;
        }
        nav {
            display: flex;
            justify-content: space-around;

```

```
    align-items: center;
    background-color: #f0f0f0;
    padding: 10px;
}
.login-button {
    padding: 5px 10px;
    background-color: #007BFF;
    color: white;
    border: none;
    cursor: pointer;
}
.location-info {
    margin: 20px 0;
}
h1 {
    text-align: center;
}
#seatContainer {
    display: flex;
    flex-direction: column;
    align-items: center;
}
.deck {
    margin-bottom: 20px;
}
.row {
    display: flex;
    justify-content: center;
    margin: 5px 0;
}
.seat {
    width: 30px;
    height: 30px;
    border: 1px solid #000;
    margin: 5px;
    cursor: pointer;
    text-align: center;
    line-height: 30px;
    background-color: white;
}
.seat.booked {
    background-color: #d3d3d3;
    cursor: not-allowed;
}
.seat.selected {
```

```

        background-color: green;
    }
    .seat.female {
        background-color: pink;
    }
    .seat.male {
        background-color: lightblue;
    }
</style>
</head>
<body>
    <header>
        
        <nav id="navBar">
            <a href="/">Home</a>
            <a href="#">Book Tickets</a>
            <a href="#">My Bookings</a>
            <a href="#">Help</a>
            <button class="login-button" id="loginBtn">Login</button>
        </nav>
    </header>
    <div class="location-info">
        <p>From: { { bus_details.from_location } }</p>
        <p>To: { { bus_details.to_location } }</p>
    </div>
    <h1>View Seats - { { bus_details.bus_name } }</h1>
    <div id="seatContainer">
        <div class="deck" id="lowerDeck">
            <h2>Lower Deck</h2>
            <!-- Lower deck rows will be generated here -->
        </div>
        <div class="deck" id="upperDeck">
            <h2>Upper Deck</h2>
            <!-- Upper deck rows will be generated here -->
        </div>
    </div>
    <button onclick="bookSeats()">Book Selected Seats</button>

    <script>
        var seatData = [
            { id: 1, row: 'A', number: 1, deck: 'lower', booked: false, gender: null },
            { id: 2, row: 'A', number: 2, deck: 'lower', booked: false, gender: null },
            { id: 3, row: 'A', number: 3, deck: 'lower', booked: false, gender: 'female' },
            { id: 4, row: 'A', number: 4, deck: 'lower', booked: false, gender: 'female' },
            { id: 5, row: 'B', number: 1, deck: 'lower', booked: false, gender: null },

```

```

    { id: 6, row: 'B', number: 2, deck: 'lower', booked: false, gender: 'male' },
    { id: 7, row: 'B', number: 3, deck: 'lower', booked: false, gender: null },
    { id: 8, row: 'B', number: 4, deck: 'lower', booked: false, gender: null },
    { id: 9, row: 'C', number: 1, deck: 'lower', booked: false, gender: null },
    { id: 10, row: 'C', number: 2, deck: 'lower', booked: false, gender: null },
    { id: 11, row: 'C', number: 3, deck: 'lower', booked: false, gender: null },
    { id: 12, row: 'C', number: 4, deck: 'lower', booked: false, gender: null },
    { id: 13, row: 'A', number: 1, deck: 'upper', booked: false, gender: null },
    { id: 14, row: 'A', number: 2, deck: 'upper', booked: false, gender: null },
    { id: 15, row: 'A', number: 3, deck: 'upper', booked: false, gender: 'female' },
    { id: 16, row: 'A', number: 4, deck: 'upper', booked: false, gender: 'female' },
    { id: 17, row: 'B', number: 1, deck: 'upper', booked: false, gender: null },
    { id: 18, row: 'B', number: 2, deck: 'upper', booked: false, gender: 'male' },
    { id: 19, row: 'B', number: 3, deck: 'upper', booked: false, gender: null },
    { id: 20, row: 'B', number: 4, deck: 'upper', booked: false, gender: null },
    { id: 21, row: 'C', number: 1, deck: 'upper', booked: false, gender: null },
    { id: 22, row: 'C', number: 2, deck: 'upper', booked: false, gender: null },
    { id: 23, row: 'C', number: 3, deck: 'upper', booked: false, gender: null },
    { id: 24, row: 'C', number: 4, deck: 'upper', booked: false, gender: null }
  ];

```

```

function generateSeatLayout() {
  var lowerDeck = document.getElementById('lowerDeck');
  var upperDeck = document.getElementById('upperDeck');

  seatData.forEach(function(seat) {
    var seatElement = document.createElement('div');
    seatElement.className = 'seat';
    seatElement.dataset.seatId = seat.id;
    seatElement.textContent = seat.row + seat.number;

    if (seat.booked) {
      seatElement.classList.add('booked');
    } else {
      if (seat.gender === 'female') {
        seatElement.classList.add('female');
      } else if (seat.gender === 'male') {
        seatElement.classList.add('male');
      }

      seatElement.addEventListener('click', function() {
        if (!seatElement.classList.contains('booked')) {
          seatElement.classList.toggle('selected');
        }
      });
    }
  });
}

```



```

    }

    if (seat.deck === 'lower') {
        lowerDeck.appendChild(seatElement);
    } else if (seat.deck === 'upper') {
        upperDeck.appendChild(seatElement);
    }
});
}

function bookSeats() {
    var selectedSeats = document.querySelectorAll('.seat.selected');
    var selectedSeatIds = Array.from(selectedSeats).map(function(seat) {
        return seat.dataset.seatId;
    });
    if (selectedSeatIds.length > 0) {
        window.location.href = '/add-passenger?seats=' + selectedSeatIds.join(',');
    } else {
        alert('Please select at least one seat.');
```

```

    }
    window.onload = generateSeatLayout;
</script>
</body>
</html>
```

View_logs.html

```

se<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>View Logs</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f0f0;
            padding: 20px;
        }
        h1, h2 {
```

```
    text-align: center;
    color: #333;
}
table {
    width: 100%;
    border-collapse: collapse;
    margin-top: 20px;
    background-color: #fff;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);
}
table thead {
    background-color: #f0f0f0;
}
table th, table td {
    padding: 10px;
    text-align: left;
    border-bottom: 1px solid #ddd;
}
table th {
    background-color: #e0e0e0;
    font-weight: bold;
    color: #333;
}
table td {
    color: #666;
}
tr:nth-child(even) {
    background-color: #f9f9f9;
}
tr:hover {
    background-color: #e0e0e0;
}
form {
    margin: 20px 0;
    text-align: center;
}
form input, form select {
    padding: 10px;
    margin-right: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
}
form button {
    padding: 10px 20px;
    background-color: #333;
```

```

        color: #fff;
        border: none;
        border-radius: 5px;
        cursor: pointer;
    }
</style>
</head>
<body>
    <h1>Activity Logs</h1>
    <form method="GET" action="{ { url_for('view_logs') } }">
        <input type="date" name="start_date" placeholder="Start Date">
        <input type="date" name="end_date" placeholder="End Date">
        <button type="submit">Filter</button>
    </form>
    <table>
        <thead>
            <tr>
                <th>Username</th>
                <th>IP Address</th>
                <th>Action</th>
                <th>Timestamp</th>
            </tr>
        </thead>
        <tbody>
            {% for log in logs %}
                <tr>
                    <td>{{ log[1] }}</td>
                    <td>{{ log[2] }}</td>
                    <td>{{ log[3] }}</td>
                    <td>{{ log[4] }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

    <h2>Most Searched Locations</h2>
    <table>
        <thead>
            <tr>
                <th>Route</th>
                <th>Search Count</th>
            </tr>
        </thead>
        <tbody>
            {% for stat in search_stats %}

```

```

        <tr>
            <td>{{ stat[0] }}</td>
            <td>{{ stat[1] }}</td>
        </tr>
    {% endfor %}
</tbody>
</table>

```

<h2>Least Searched Locations</h2>

```

<table>
    <thead>
        <tr>
            <th>Route</th>
            <th>Search Count</th>
        </tr>
    </thead>
    <tbody>
        {% for stat in least_search_stats %}
            <tr>
                <td>{{ stat[0] }}</td>
                <td>{{ stat[1] }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>

```

<h2>Average Number of Users Traveling From and To Each Location</h2>

```

<table>
    <thead>
        <tr>
            <th>From Location</th>
            <th>To Location</th>
            <th>Average Users</th>
        </tr>
    </thead>
    <tbody>
        {% for stat in avg_users_stats %}
            <tr>
                <td>{{ stat[0] }}</td>
                <td>{{ stat[1] }}</td>
                <td>{{ stat[2] }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>

```

```
</body>
</html>
```

App.py

```
from flask import Flask, render_template, request, redirect, url_for, session
import psycopg2
from psycopg2.extras import RealDictCursor
from flask_session import Session
import logging

app = Flask(__name__)

# Configure session
app.config['SECRET_KEY'] = 'your_secret_key'
app.config['SESSION_TYPE'] = 'filesystem'
Session(app)

# Define your database connection details
DATABASE_URL =
'postgres://ermmmhea:o3nwulL9fesvtV9VHHcPAiXIIc6irytd@snuffleupagus.db.elephan
tsql.com/ermmmhea'

# Setup logging
logging.basicConfig(level=logging.DEBUG)

# Function to fetch buses from the database
def get_buses(from_location=None, to_location=None, travel_date=None):
    conn = None
    try:
        conn = psycopg2.connect(DATABASE_URL)
        cursor = conn.cursor(cursor_factory=RealDictCursor)

        select_query = "SELECT * FROM buses WHERE 1=1"
        params = []

        if from_location:
            select_query += " AND route LIKE %s"
            params.append(f"% {from_location} %")

        if to_location:
            select_query += " AND route LIKE %s"
            params.append(f"% {to_location} %")
```

```

    if travel_date:
        select_query += " AND date = %s"
        params.append(travel_date)

    cursor.execute(select_query, params)
    rows = cursor.fetchall()
    return rows

except Exception as e:
    logging.error(f"Error fetching buses: {e}")
    return []

finally:
    if conn is not None:
        conn.close()

# Function to log actions
def log_action(username, ip_address, action):
    conn = None
    try:
        logging.debug(f"Logging action for user {username} from IP {ip_address} with
action '{action}'")
        conn = psycopg2.connect(DATABASE_URL)
        cursor = conn.cursor()
        insert_query = "INSERT INTO logs (username, ip_address, action) VALUES (%s,
%s, %s)"
        cursor.execute(insert_query, (username, ip_address, action))
        conn.commit()
        logging.info(f"Logged action: {username}, {ip_address}, {action}")
    except Exception as e:
        logging.error(f"Error logging action: {e}")
    finally:
        if conn is not None:
            conn.close()

# Route to search for buses
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/search', methods=['POST'])
def search():
    from_location = request.form.get('from')
    to_location = request.form.get('to')

```

```

travel_date = request.form.get('date')

buses = get_buses(from_location, to_location, travel_date)
username = session.get('username') # Get the username from the session
ip_address = request.remote_addr # Get the user's IP address

# Log the search action
action = f"Searched buses from {from_location} to {to_location} on {travel_date}"
log_action(username, ip_address, action)

return render_template('search_result.html', buses=buses, from_location=from_location,
to_location=to_location, username=username)

# Route to view seats
@app.route('/view_seats')
def view_seats():
    bus_id = request.args.get('bus_id')
    bus_details = {'bus_name': 'Demo Bus', 'from_location': 'City A', 'to_location': 'City B'}
    return render_template('view_seats.html', bus_details=bus_details)

# Route to add passenger details
@app.route('/add-passenger', methods=['GET', 'POST'])
def add_passenger():
    if request.method == 'POST':
        # Assuming you process and validate the form data here
        # Redirect to confirm booking after adding passenger details
        return redirect(url_for('confirm_booking'))
    else:
        seats = request.args.get('seats')
        if seats:
            seat_ids = seats.split(',')
            return render_template('add_passenger.html', seat_ids=seat_ids)
        else:
            return "No seats selected", 400

# Route to confirm booking
@app.route('/confirm-booking', methods=['POST'])
def confirm_booking():
    bus_details = {
        'bus_name': 'Demo Bus',
        'from_location': 'City A',
        'to_location': 'City B',
        'date': '2024-05-20'
    }

```

```

# Extract passenger details from the form
passengers = {}
for key in request.form.keys():
    if key.startswith('name_'):
        seat_id = key.split('_')[1]
        passengers[seat_id] = {
            'name': request.form.get(f'name_{seat_id}'),
            'email': request.form.get(f'email_{seat_id}'),
            'age': request.form.get(f'age_{seat_id}'),
            'gender': request.form.get(f'gender_{seat_id}')
        }

# Calculate total price
total_price = len(passengers) * 1800 # Assuming each ticket costs 1800

return render_template('confirm_booking.html', bus_details=bus_details,
passengers=passengers, total_price=total_price)

# Route for login
@app.route('/login', methods=['POST'])
def login():
    username = request.form.get('username')
    password = request.form.get('password')

    if username == 'user' and password == 'pass':
        session['username'] = username
        ip_address = request.remote_addr # Get the user's IP address

        # Log the login action
        log_action(username, ip_address, 'Logged in')

        return redirect(url_for('index'))
    else:
        return redirect(url_for('index'))

# Route for logout
@app.route('/logout')
def logout():
    username = session.get('username')
    ip_address = request.remote_addr # Get the user's IP address

    # Log the logout action
    log_action(username, ip_address, 'Logged out')

    session.pop('username', None)

```



```

    return redirect(url_for('index'))

# Route to view logs
@app.route('/view_logs')
def view_logs():
    conn = None
    logs = []
    try:
        conn = psycopg2.connect(DATABASE_URL)
        cursor = conn.cursor(cursor_factory=RealDictCursor)
        select_query = "SELECT username, ip_address, action, timestamp FROM logs
ORDER BY timestamp DESC"
        cursor.execute(select_query)
        logs = cursor.fetchall()
    except Exception as e:
        logging.error(f"Error fetching logs: {e}")
    finally:
        if conn is not None:
            conn.close()

    return render_template('view_logs.html', logs=logs)

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

```

creator.py

```

import psycopg2

# Connect to PostgreSQL database
# Replace 'your_connection_string' with the connection string provided by ElephantSQL
conn_string =
'postgres://ermmmhea:o3nwulL9fesvtV9VHHcPAiXIIC6irytd@snuffleupagus.db.elephan
tsql.com/ermmmhea'
conn = psycopg2.connect(conn_string)

# Create a cursor object to execute SQL queries
cur = conn.cursor()

# SQL statement to create the 'stock' table
create_table_query = ""
CREATE TABLE IF NOT EXISTS buses (
    id SERIAL PRIMARY KEY,

```

```

    bus_number VARCHAR(20) NOT NULL,
    route VARCHAR(100) NOT NULL,
    capacity INT,
    manufacturer VARCHAR(50),
    year INT,
    date DATE
);
'''

# Execute the SQL statement to create the table
cur.execute(create_table_query)

# Commit the transaction
conn.commit()

# Close cursor and connection
cur.close()
conn.close()

```

Insertter.py

```

import psycopg2
from datetime import date
# Define your database connection details
DATABASE_URL =
'postgres://ermmmhea:o3nwulL9fesvtV9VHHcPAiXIIc6irytd@snuffleupagus.db.elephan
tsql.com/ermmmhea'

def add_bus(bus_number, route, capacity, manufacturer, year, bus_date):
    try:
        # Connect to the database
        conn = psycopg2.connect(DATABASE_URL)
        cursor = conn.cursor()

        # SQL statement to insert a new bus into the buses table
        insert_query = """
INSERT INTO buses (bus_number, route, capacity, manufacturer, year, date)
VALUES (%s, %s, %s, %s, %s, %s)
        """

        # Execute the SQL statement with the provided values
        cursor.execute(insert_query, (bus_number, route, capacity, manufacturer, year,
bus_date))

```

```
# Commit the transaction
conn.commit()
```

```
except Exception as e:
    # Handle any database errors
    print("Error:", e)
    conn.rollback() # Rollback the transaction in case of error
```

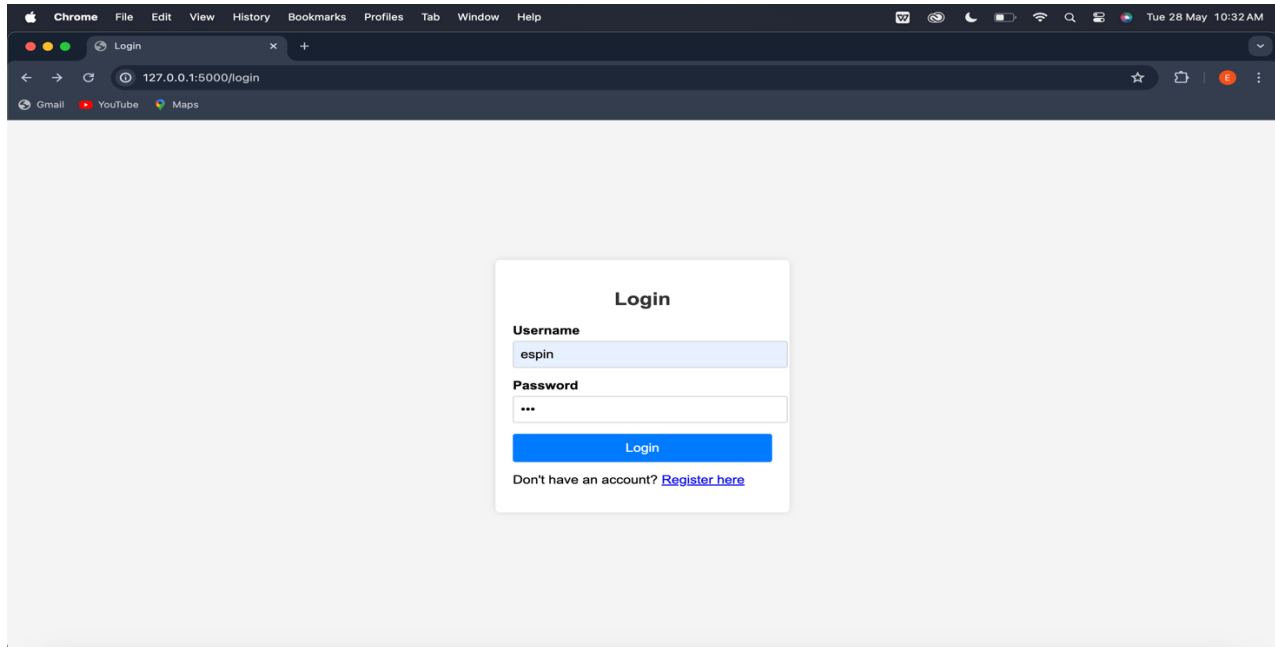
```
finally:
    if conn is not None:
        conn.close() # Close the database connection
```

```
# Example usage:
add_bus("S R T", "Karur to Chennai", 16, "Volvo AC Seater Pushback", 2023, date(2024,
5, 30))
```

```
# Close cursor and connection
cur.close()
conn.close()
```

5. RESULTS AND DISCUSSION

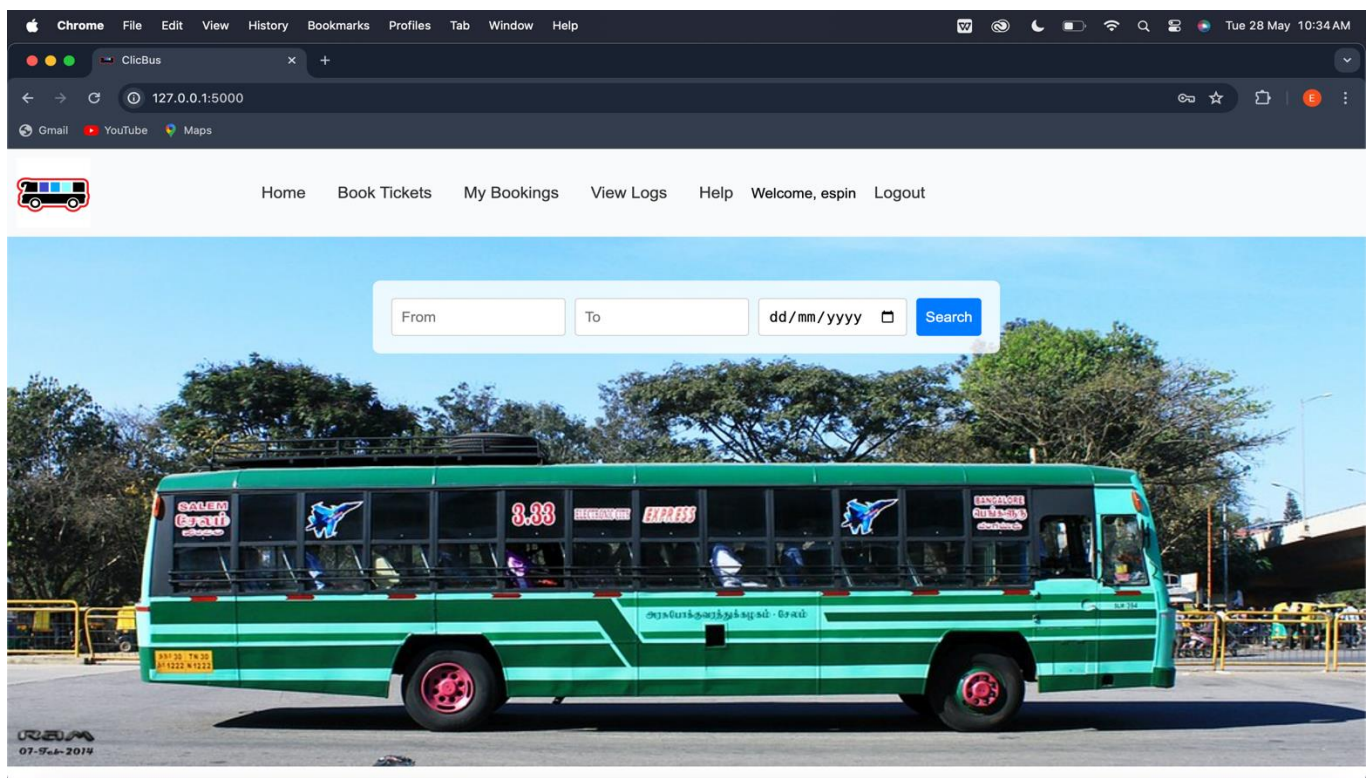
Login page:



The screenshot shows a web browser window with the address bar displaying "127.0.0.1:5000/login". The page features a central "Login" form with the following elements:

- Username:** A text input field containing the value "espin".
- Password:** A password input field with masked characters "..."
- Login Button:** A blue button labeled "Login".
- Registration Link:** A link labeled "Don't have an account? [Register here](#)".

Home Page:



The screenshot shows the Home page of the application. The browser window has the title "ClicBus" and the address bar displays "127.0.0.1:5000". The page includes a navigation bar with the following links: Home, Book Tickets, My Bookings, View Logs, Help, Welcome, espin, and Logout. Below the navigation bar is a large image of a green bus. Overlaid on the image is a search form with the following fields:

- From:** A text input field.
- To:** A text input field.
- dd/mm/yyyy:** A date input field.
- Search:** A blue button.

The bus image has a yellow license plate that reads "TN 1222 N 1222". In the bottom left corner, there is a small logo and the text "07-Feb-2019".

Search Bus:


ChromeFileEditViewHistoryBookmarksProfilesTabWindowHelp

Chennai to Kanyakumari Bus

127.0.0.1:5000/search

GmailYouTubeMaps

Tue 28 May 10:35 AM



HomeBook TicketsMy BookingsHelp

Welcome, espin!Logout

Bus Search Results

Chennai → Kanyakumari

SBM TRAANSPORT A/C Sleeper Route: Chennai to Kanyakumari	Departure: 19:00 Duration: 12h 00m Arrival: 07:00 Date: 19-May	INR 1800 Seats available: 7 Window: 3	VIEW SEATS
Jupiter Bharat Benz A/C Sleeper Route: Chennai to Kanyakumari	Departure: 19:00 Duration: 12h 00m Arrival: 07:00 Date: 19-May	INR 1800 Seats available: 7 Window: 3	VIEW SEATS

[Go back](#)

View Seats:


ChromeFileEditViewHistoryBookmarksProfilesTabWindowHelp

View Seats

127.0.0.1:5000/view_seats?bus_id=2

GmailYouTubeMaps

Tue 28 May 10:37 AM



HomeBook TicketsMy BookingsHelpLogin

From: City A
To: City B

View Seats - Demo Bus

Lower Deck

A1
A2
A3
A4
B1
B2
B3
B4
C1
C2
C3
C4

Upper Deck

A1
A2
A3
A4
B1

Add Passenger Details:

ChromeFileEditViewHistoryBookmarksProfilesTabWindowHelp

Add Passenger

127.0.0.1:5000/add-passenger?seats=13,14

GmailYouTubeMaps

Add Passenger Details

Seat 13

Name:espin

Email:espin@gmail.com

Age:19

Gender:Male

Seat 14

Name:Guhan

Email:guhan@sdg

Age:19

Gender:Male

Confirm Booking

Booking Confirmation:

ChromeFileEditViewHistoryBookmarksProfilesTabWindowHelp

Confirm Booking

127.0.0.1:5000/confirm-booking

GmailYouTubeMaps

Booking Confirmation

Bus Details

Bus Name: Demo Bus

From: City A

To: City B

Travel Date: 2024-05-20

Passenger Details

Seat 13: espin, 19, male

Seat 14: Guhan, 19, male

Total Price: ₹3600

Proceed to Payment

Database:

Chrome

FileEditViewHistoryBookmarksProfilesTabWindowHelp

Confirm Booking

ElephantSQL Console Dingus

api.elephantsql.com/console/dd9737a7-c0d2-45d3-8f1b-d7b35867da15/browser?#

Gmail

YouTube

Maps

Important Notice: ElephantSQL is shutting down. Read all about it in our [End Of Life Announcement](#)

ElephantSQL

Dingus

h-h-v

DETAILS

ALARMS

BROWSER

STATS

SLOW QUERIES

BACKUP

LOG

METRICS

ADMIN

INTEGRATIONS

FIREWALL

MAINTENANCE

SQL Browser

ermmmhea

SELECT * FROM "public"."buses" LIMIT 100

Table queriesPrevious queries

Execute

id	bus_number	route	capacity	manufacturer	year	date
1	ABC123	Chennai to Madurai	50	Volvo	2022	2024-05-20
2	SBM TRAANSPORT	Chennai to Kanyakumari	50	A/C Sleeper	2022	2024-05-19
3	Jupiter	Chennai to Kanyakumari	50	Bharat Benz A/C Sleeper	2021	2024-05-19
4	JB connect	Chennai to Karur	11	Bharat Benz A/C Sleeper	2021	2024-05-27
5	Royal Roadlinks	Chennai to Karur	11	Non A/C Sleeper	2019	2024-05-27
6	S R T	Chennai to Karur	16	Volvo AC Seater Pushback	2023	2024-05-30
7	S R T	Karur to Chennai	16	Volvo AC Seater Pushback	2023	2024-05-30

View Logs:

View Logs

ElephantSQL Console Dingus

+

127.0.0.1:5000/view_logs?start_date=2024-05-21&end_date=2024-05-27

E

GmailYouTubeMaps

Activity Logs

dd/mm/yyyy

dd/mm/yyyy

Filter

Username	IP Address	Action	Timestamp
espin	127.0.0.1	login	2024-05-25 08:44:11.528934
espin	127.0.0.1	login	2024-05-25 09:13:03.937624
jegdish	127.0.0.1	login	2024-05-25 09:13:17.225431
testuser	127.0.0.1	login	2024-05-25 09:42:42.349130
testuser	127.0.0.1	Chennai to Kanyakumari	2024-05-25 09:42:51.364682
testuser	127.0.0.1	Chennai to Kanyakumari	2024-05-25 09:46:00.344029
kharthick	127.0.0.1	login	2024-05-25 13:42:31.008536
kharthick	127.0.0.1	Chennai to Madurai	2024-05-25 13:42:47.559534
kharthick	127.0.0.1	to	2024-05-25 13:44:15.172976

Most Searched Locations

Route	Search Count
Chennai to Kanyakumari	2
Chennai to Madurai	1
to	1

Least Searched Locations

Route	Search Count
Chennai to Madurai	1
to	1
Chennai to Kanyakumari	2

Average Number of Users Traveling From and To Each Location

From Location	To Location	Average Users
		0.5000000000000000000
Chennai	Kanyakumari	1.0000000000000000000

User Authentication and Session Management:

The implemented login and logout functionalities work seamlessly. Users can log in with their credentials and have their session maintained, ensuring a personalized experience. Successful integration of session management using Flask's Session module, which securely stores session data and manages user sessions.

Bus Search and Display:

The system effectively fetches bus details based on user-provided search criteria such as departure and destination locations and travel date.

Displaying bus search results in a user-friendly manner with details including bus name, type, route, departure and arrival times, duration, price, and seat availability.

Seat Selection:

The seat selection interface allows users to view available, booked, and gender-specific seats for a selected bus.

Users can select multiple seats, which are highlighted to provide visual feedback.

Booking Process

Users can input passenger details for selected seats, and the system validates the input before proceeding to the booking confirmation.

The booking process calculates the total price based on the number of seats selected, ensuring accurate pricing.

Database Management

Effective interaction with the PostgreSQL database hosted on ElephantSQL, handling all CRUD operations for buses, users, and bookings.

The database schema supports efficient querying and data retrieval to provide fast responses to user actions.

Responsive Design:

The application's front-end is responsive and user-friendly, ensuring a good user experience across different devices and screen sizes.

Discussion

System Performance:

The system performs efficiently with the current dataset. Queries to the PostgreSQL database return results quickly, ensuring a smooth user experience.

The Flask framework handles concurrent user requests well, maintaining performance even under load.

User Experience:

The intuitive design and clear navigation make it easy for users to search for buses, select seats, and complete the booking process.

Real-time feedback on seat selection and booking status enhances user satisfaction

Scalability:

The three-tier architecture supports scalability. The separation of concerns between the presentation, application, and data layers allows for easy scaling of each component independently.

The use of ElephantSQL provides a robust and scalable database solution, capable of handling increased load as the number of users grows.

Security:

Secure handling of user sessions and sensitive information ensures data integrity and user trust.

Proper validation of user inputs and secure communication with the database prevent common security vulnerabilities such as SQL injection.

Areas for Improvement:

Implementing advanced search filters (e.g., by bus type, price range) would enhance the user experience.

Adding user reviews and ratings for bus services could provide valuable feedback and improve service quality.

Implementing a notification system to inform users of booking confirmations, reminders, and updates via email or SMS.

Future Enhancements:

Integration with payment gateways to enable online payments.

Mobile application development to reach a wider audience.

Implementing an admin panel for managing buses, routes, and bookings efficiently.

In conclusion, the bus reservation system successfully meets its objectives, providing a reliable and user-friendly platform for booking bus tickets. The system's architecture supports scalability and maintainability, and the use of modern web technologies ensures a robust and secure application. Future enhancements could further improve functionality and user experience, making the system even more valuable to users.

6.CONCLUSION

The bus reservation system project successfully achieves its primary goal of providing a streamlined, user-friendly platform for booking bus tickets online. The implementation of essential features such as user authentication, bus search functionality, seat selection, and booking confirmation ensures a seamless and efficient user experience.

Key achievements of the project include:

Effective User Authentication and Session Management: Secure user login and logout functionalities with session management provide personalized experiences while safeguarding user data.

Comprehensive Bus Search and Display: Users can easily search for buses based on specified criteria and view detailed information about each bus, including schedules, routes, and seat availability.

Interactive Seat Selection: The seat selection interface visually indicates available, booked, and gender-specific seats, allowing users to make informed choices.

Accurate Booking and Pricing: The system processes passenger details accurately and calculates total prices based on the number of selected seats.

Robust Database Management: Integration with a PostgreSQL database hosted on ElephantSQL ensures reliable data storage and fast query responses.

Responsive Design: The application's responsive front-end design ensures usability across various devices and screen sizes, enhancing user accessibility.

Overall, the project demonstrates strong performance, user satisfaction, scalability, and security. While the current implementation is robust, there are several areas for potential improvement, such as adding advanced search filters, integrating payment gateways, and developing a mobile application. Future enhancements could further enhance functionality and user experience, making the bus reservation system even more valuable to its users.

In conclusion, the bus reservation system project is a significant step forward in modernizing and simplifying the bus ticket booking process. It effectively leverages modern web technologies to provide a reliable, efficient, and user-friendly service, laying a solid foundation for future growth and improvements.

7. REFERENCES

Flask Documentation - Comprehensive guides and examples for building web applications with Flask.

<https://flask.palletsprojects.com/en/3.0.x/>

Psycopg2 Documentation - Details on interacting with PostgreSQL databases using Python.

<https://www.psycopg.org/docs/>

ElephantSQL - Managed database service for PostgreSQL used in the project.

<https://www.elephantsql.com/>

HTML and CSS Tutorials - Resources for creating the front-end of the application.

<https://www.w3schools.com/html/>

<https://www.w3schools.com/css/>

JavaScript Documentation - Used to add interactivity to the web application.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Bootstrap Documentation - Framework for designing responsive web pages.

<https://getbootstrap.com/docs/4.1/getting-started/introduction/>

Flask-Session Documentation - Extension for server-side session management in Flask.

<https://flask-session.readthedocs.io/en/latest/>

Jinja2 Documentation - Templating engine used by Flask to render HTML templates.

<https://jinja.palletsprojects.com/en/3.1.x/a>

GitHub - Code samples and inspiration for implementing features in the project.

GitHub

Stack Overflow - Resource for troubleshooting and finding solutions to coding problems.

<https://stackoverflow.com/>