Online Bus Ticket Booking Application

Introduction

This project is a **Bus Ticket Booking Application** designed to provide a seamless experience for users to register, search for buses, book tickets, and generate PDF confirmations. The application supports two roles:

1. **User:** Can search for buses, book tickets, and view their bookings.
2. **Admin:** Can manage bus details, including adding, updating, and deleting bus information.

Additionally, the application comes with an integrated frontend, the details of which can be accessed via the provided Drive link (ensure to read the **Mandatory Information**).

Table of Contents

Features

- User registration and secure login with Spring Security.
- Role-based access control for **Admin** and **User**.
- Search for buses by **origin**, **destination**, and **date**.
- Book bus tickets with the ability to input passenger details.
- View and manage individual booking tickets.
- Generate **PDF confirmations** for bookings.

- Admin-specific functionalities for managing buses:
    - Add, update, delete bus details.

Technologies Used

1. **Backend:**
    - Java
    - Spring Boot (REST API development)
    - Spring Security (authentication and authorization)
    - Jakarta Servlet API
    - iTextPDF (PDF generation)
    - Lombok (boilerplate code reduction)
2. **Frontend:**
    - Details available in the provided Drive link.
3. **Database:**
    - Relational Database (e.g., MySQL or PostgreSQL)

Running the Application

Prerequisites

- **Java**: Version 17 or higher
- **Maven**: Version 3.6.3 or higher
- **Database**: A relational database (e.g., MySQL or PostgreSQL)

Steps to Run the Application

1. **Clone the repository**: Download the project files.
2. **Configure the database**:
    - Update the application.properties or application.yml file with your database credentials:

properties

CopyEdit

spring.datasource.url=jdbc:mysql://localhost:3306/bus_ticket_booking

spring.datasource.username=<your-database-username>

spring.datasource.password=<your-database-password>

spring.jpa.hibernate.ddl-auto=update

3. **Build the application**:
    - Run mvn clean install to build the project.
4. **Run the application**:
    - Execute java -jar <application-jar-file> or run the application directly from an IDE.
5. **Access the application**:
    - Open a web browser and navigate to http://localhost:8080.

Usage

1. User Registration and Login

- **Registration**:
    - Access /registration to create a new user account.
    - Input details like username, email, and password.
- **Login**:
    - Navigate to /login to log in using registered credentials.

2. Admin Functionalities

- Log in as an admin to access the admin dashboard.
- **Manage buses**:
    - **Add a new bus**: Navigate to /addBus.
    - **View all buses**: Use /viewAllBuses.
    - **Find a bus by ID**: Access /findBusById.
    - **Update bus details**: Update details via /updateByBus.
    - **Delete a bus**: Delete via /delete/{serialNo}.

3. Booking Tickets

- **Search for buses**:
    - Use the search functionality on the user home page to find buses by origin, destination, and travel date.
- **Book a bus**:
    - Navigate to /bookBus/{busId} to book tickets for a specific bus.
- **View bookings**:

    - Access all your bookings at /bookings.
- **Generate PDF confirmation**:
    - Generate a booking confirmation PDF at /generatePdf/{bookingId}.

Mandatory Information

- Ensure the application runs on a port other than **8080** if it's already in use. You can modify the port in application.properties:

properties

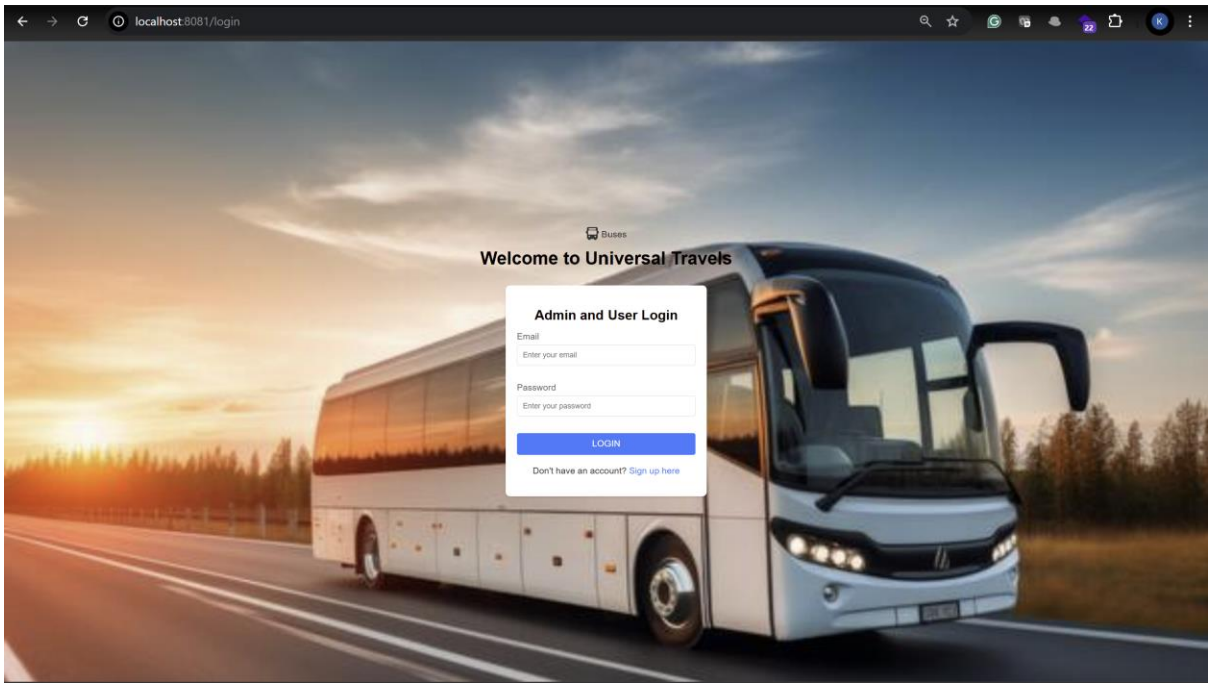
CopyEdit

server.port=<desired-port>

- Always verify that the database service is up and running before starting the application.

User Interface:

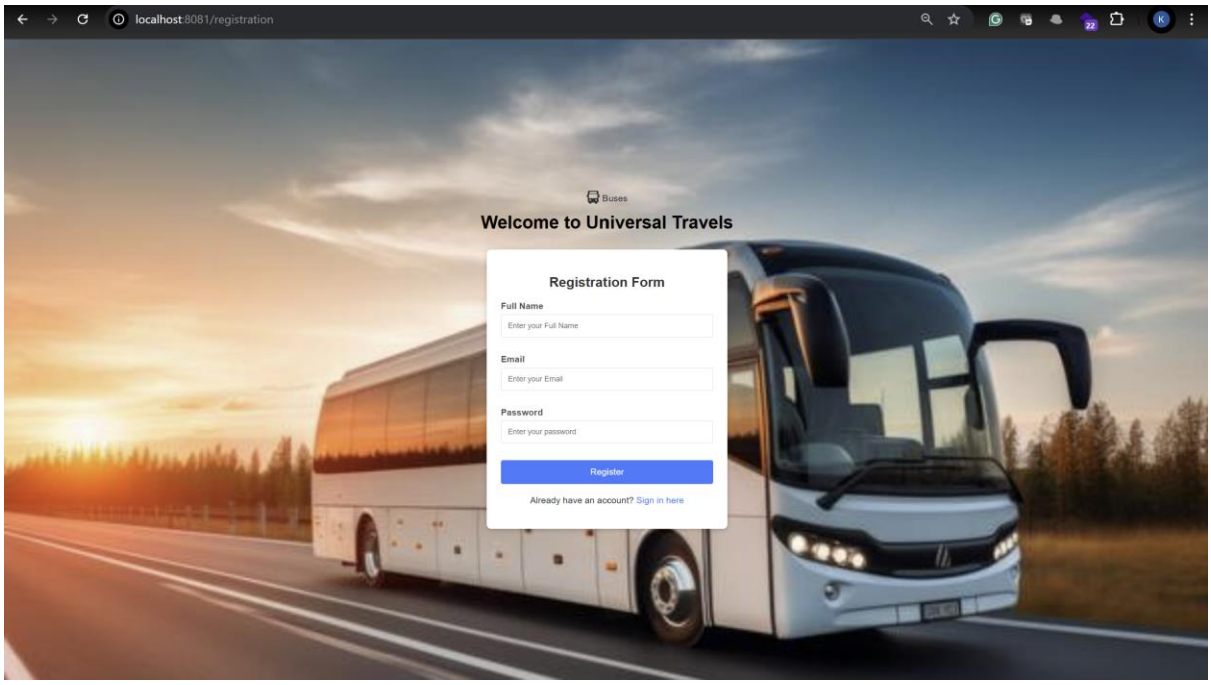
Home Page



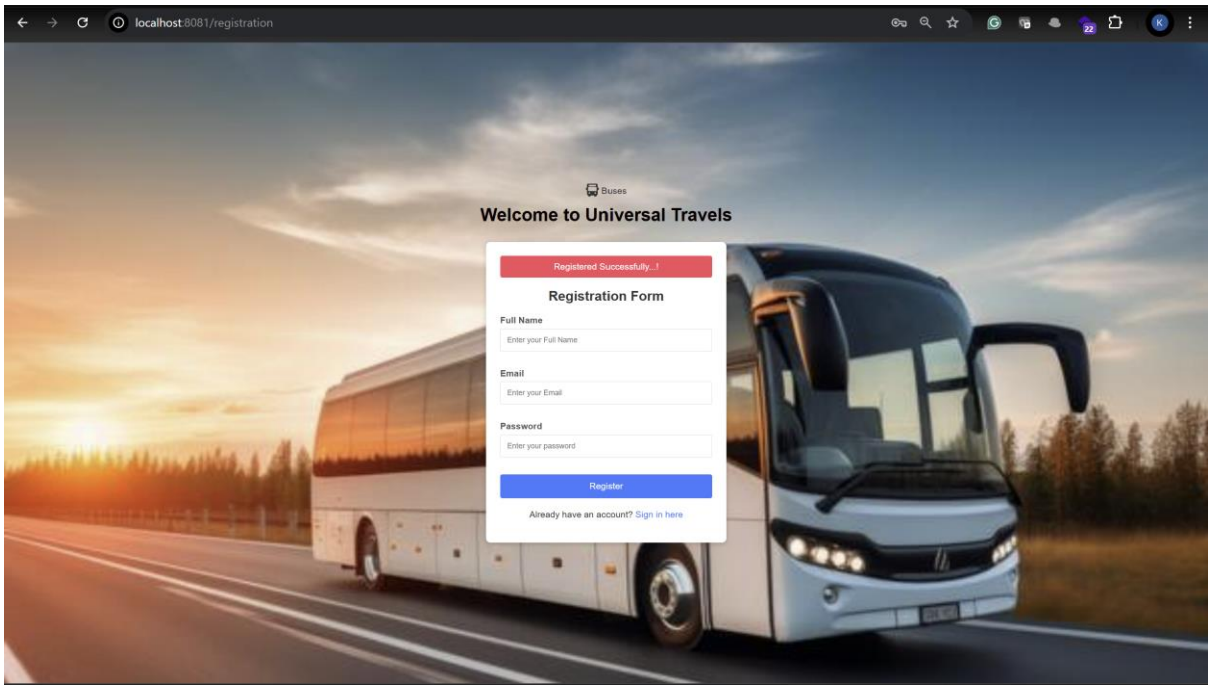
On the same page, we can log in with Admin and User Login.

As per the security concern, we have disabled the sign-up for admin users. In real time, the admin team needs to sign up with a norma user later by calling customer care; they can then change their role. Beginning, everyone comes under the USER role.

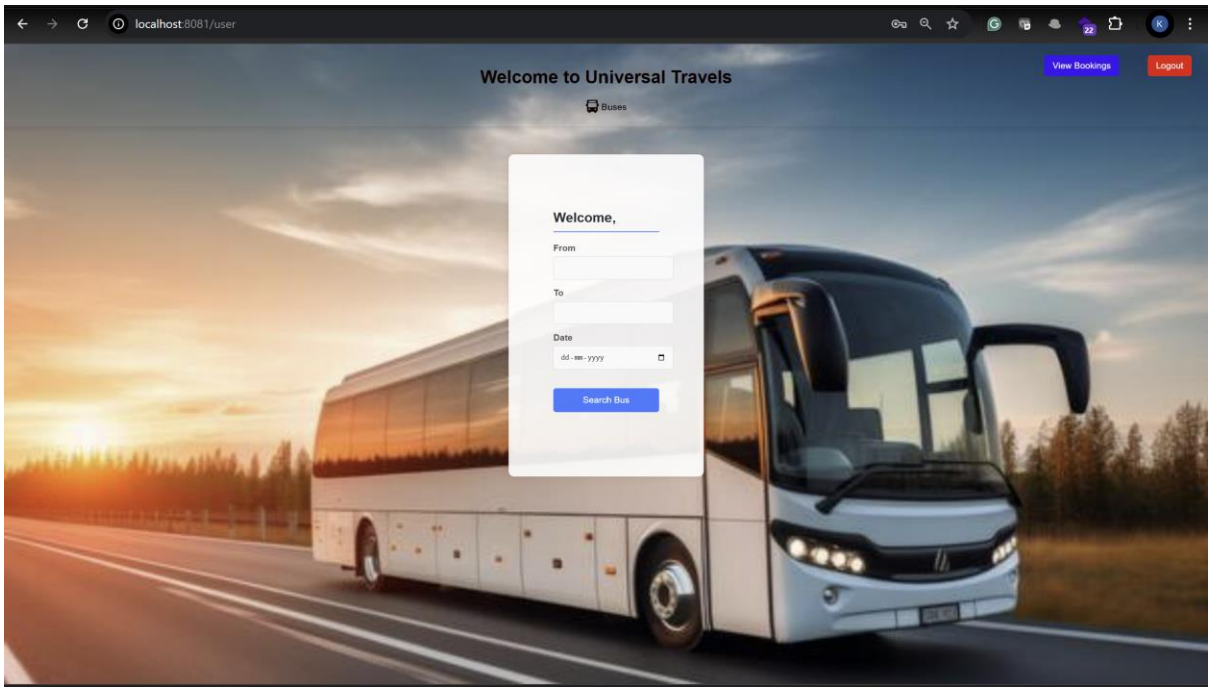
Sign-up page



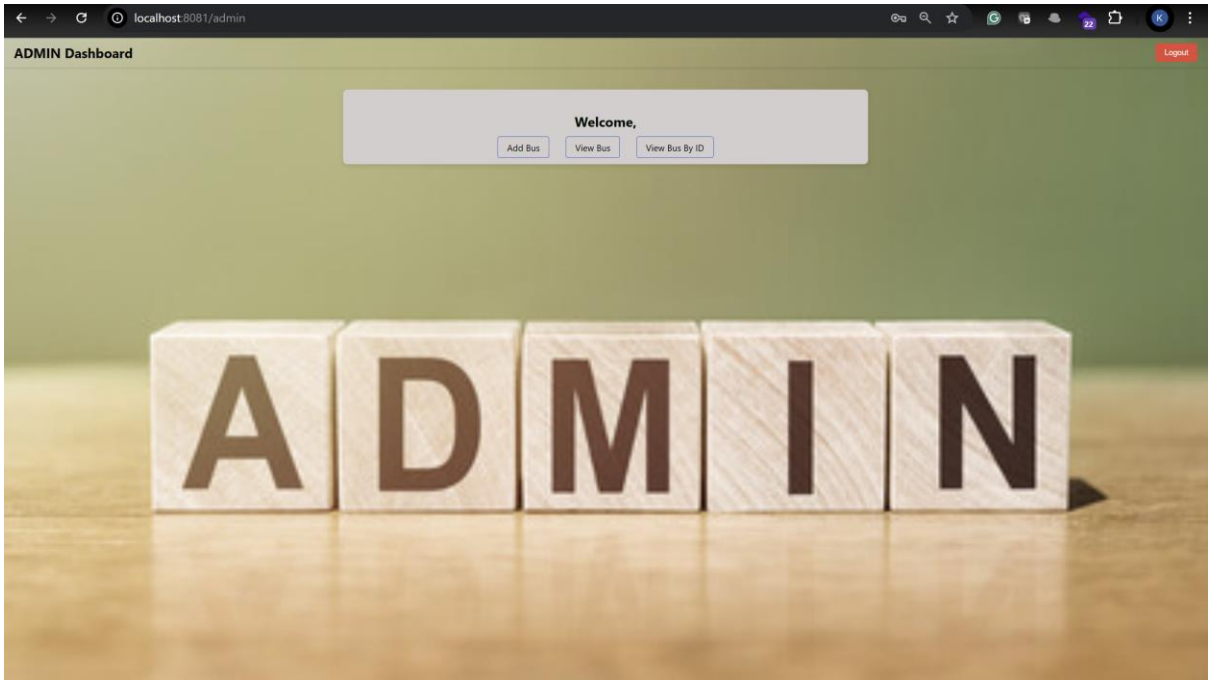
After registration you will get pop-up like below then click on sign in to log in



As I mentioned initially everyone has USER role so you will land on below page



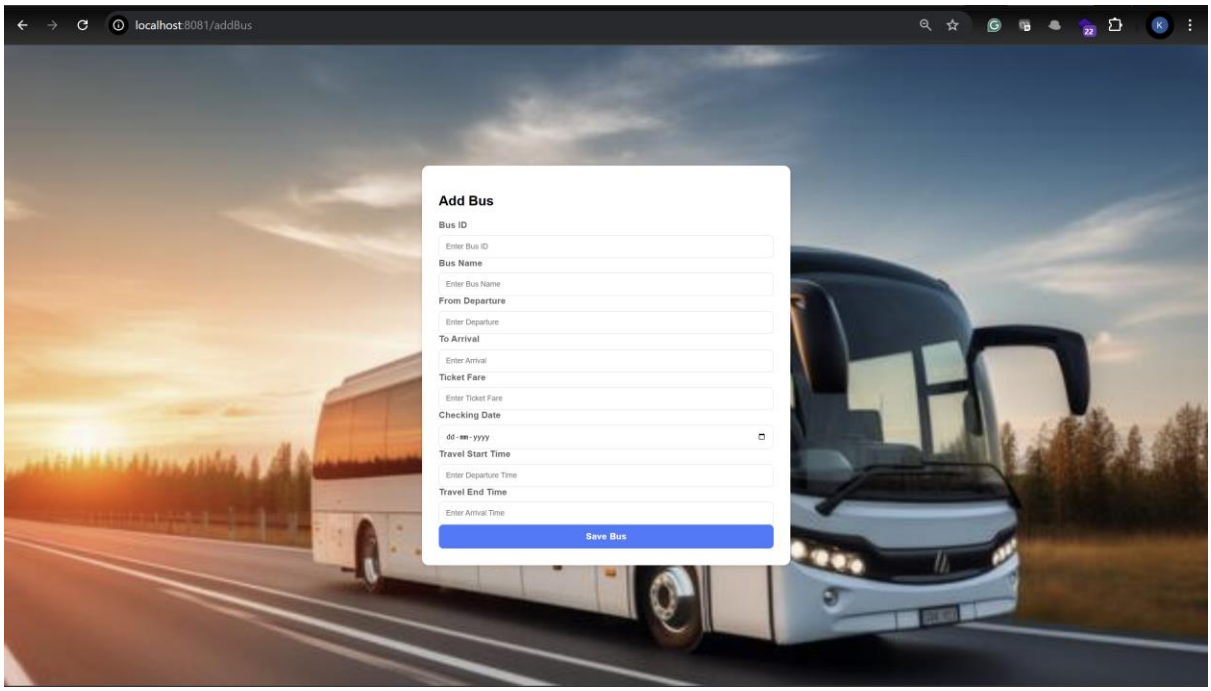
After Changing the role to ADMIN from Database Admin will land on below page

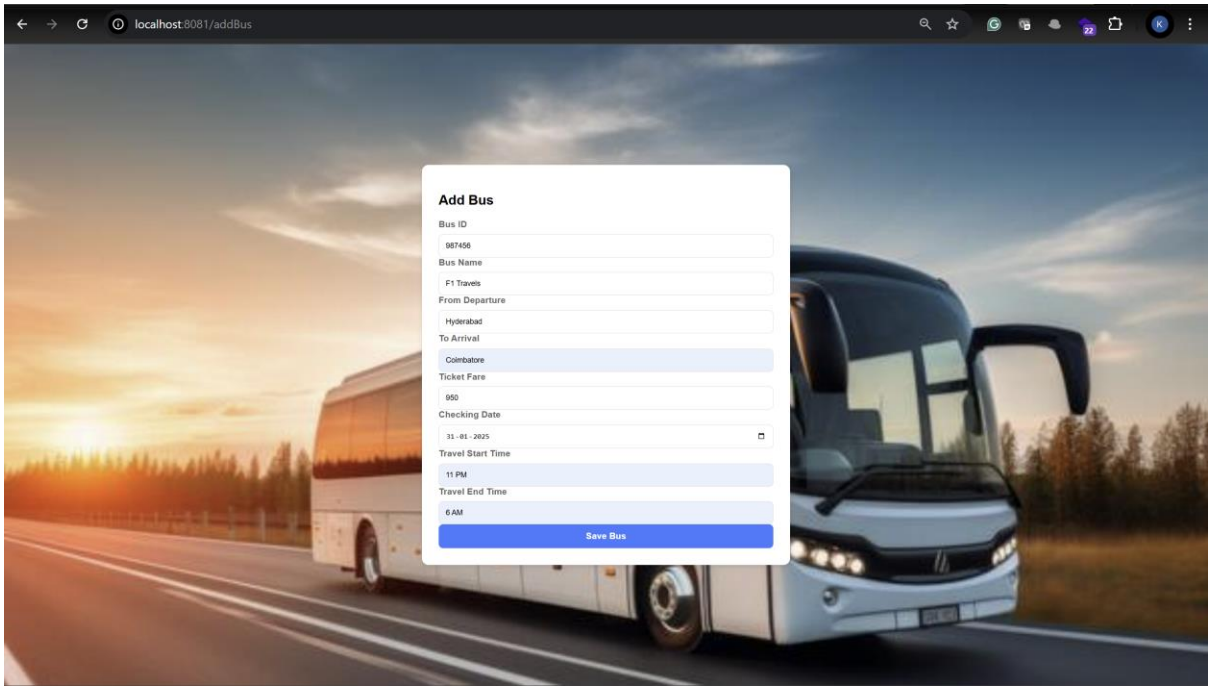


Here Admin will get option to Add View and Search the Bus.

Add Bus:

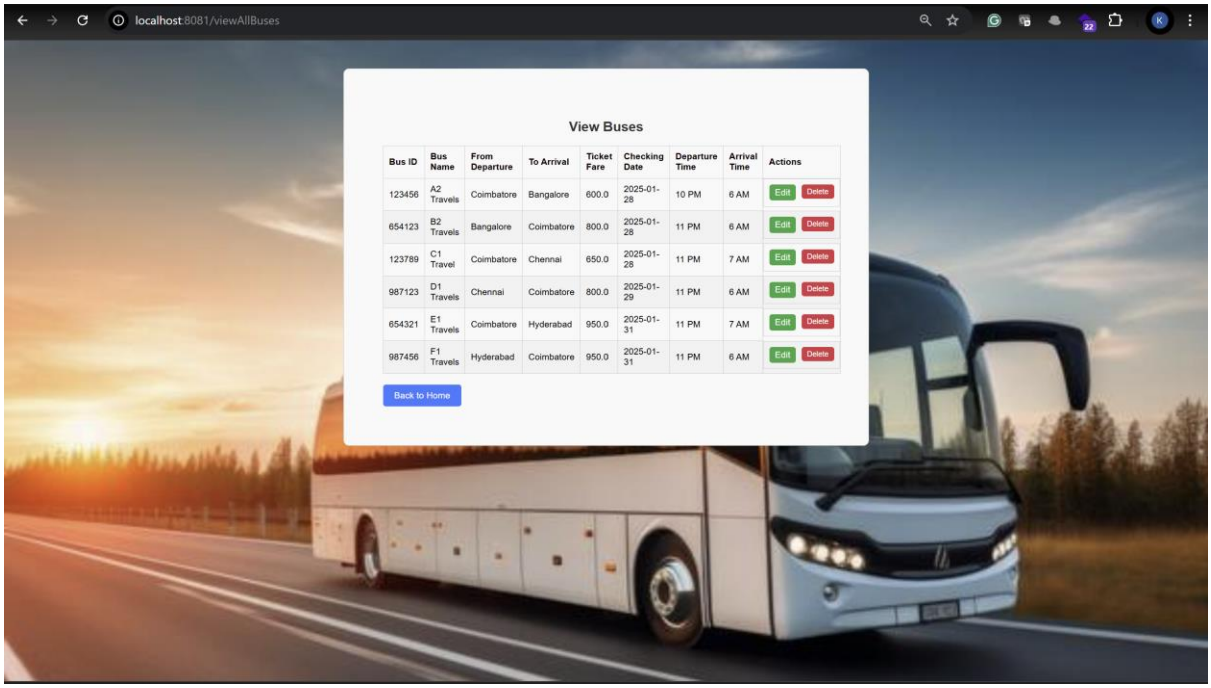
User Interface to add bus





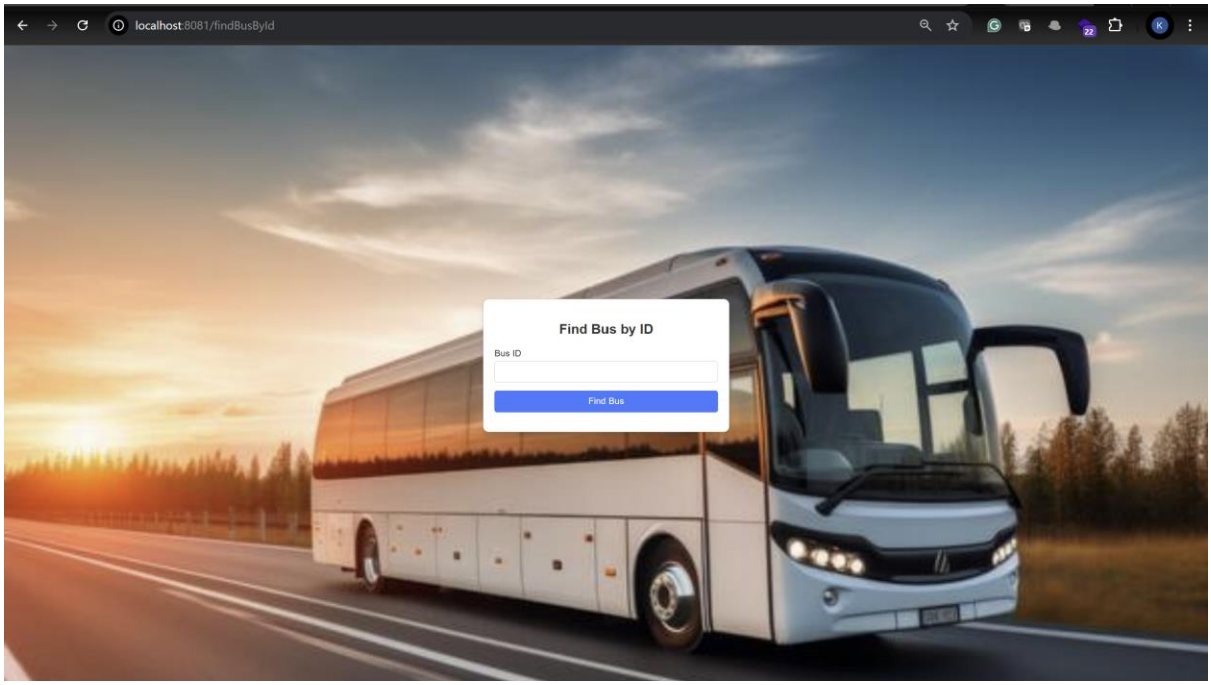
Bus Details will be added to the db by clicking the save bus button.

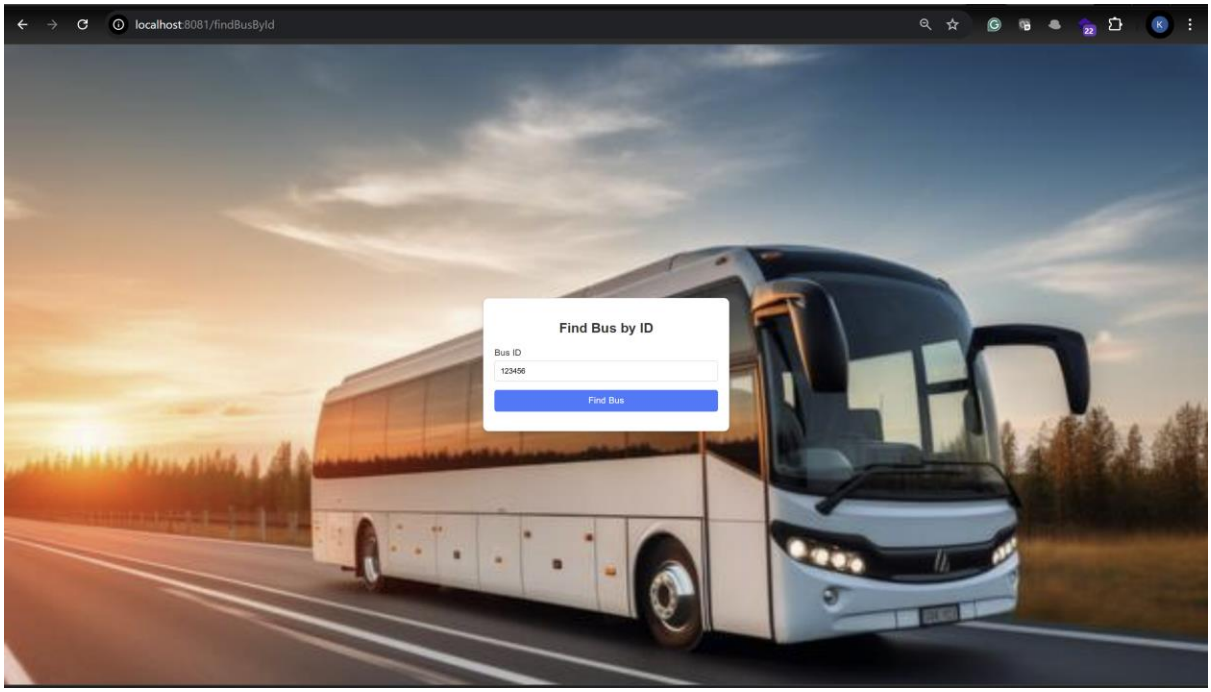
By Clicking view bus Admin can access the list of buses



Bus ID	Bus Name	From Departure	To Arrival	Ticket Fare	Checking Date	Departure Time	Arrival Time	Actions
123456	A2 Travels	Coimbatore	Bangalore	600.0	2025-01-28	10 PM	6 AM	Edit Delete
654123	B2 Travels	Bangalore	Coimbatore	800.0	2025-01-28	11 PM	6 AM	Edit Delete
123789	C1 Travel	Coimbatore	Chennai	650.0	2025-01-28	11 PM	7 AM	Edit Delete
987123	D1 Travels	Chennai	Coimbatore	800.0	2025-01-29	11 PM	6 AM	Edit Delete
654321	E1 Travels	Coimbatore	Hyderabad	950.0	2025-01-31	11 PM	7 AM	Edit Delete
987456	F1 Travels	Hyderabad	Coimbatore	950.0	2025-01-31	11 PM	6 AM	Edit Delete

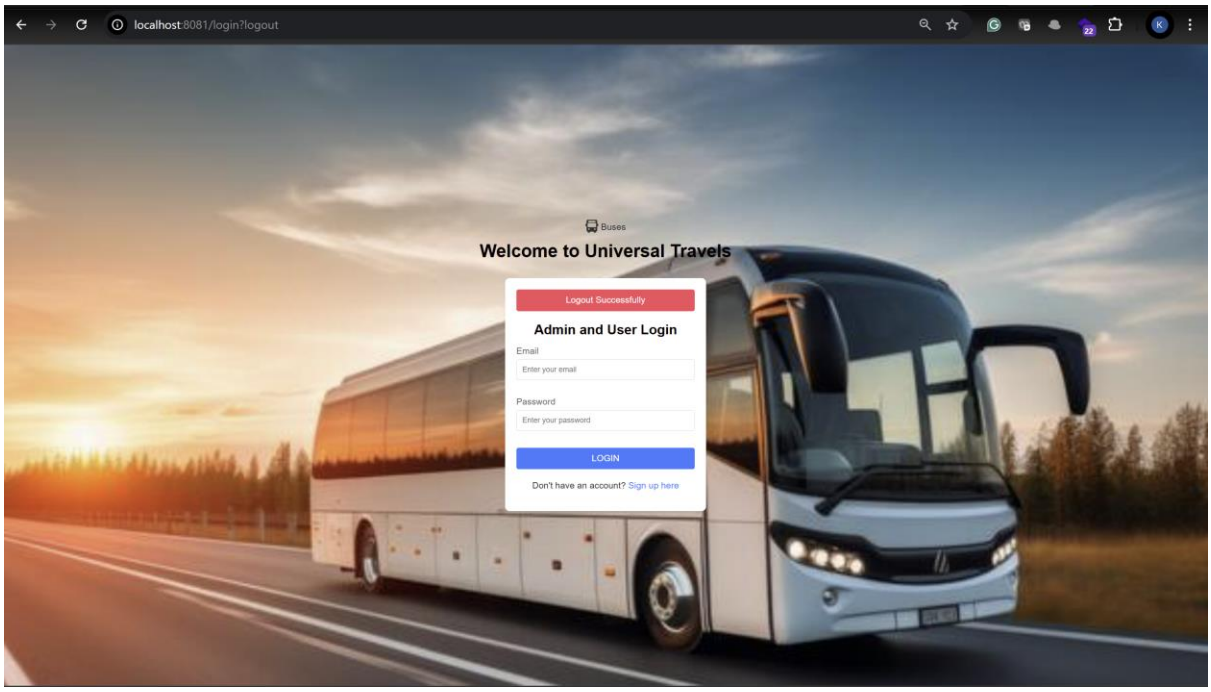
Search the bus by id





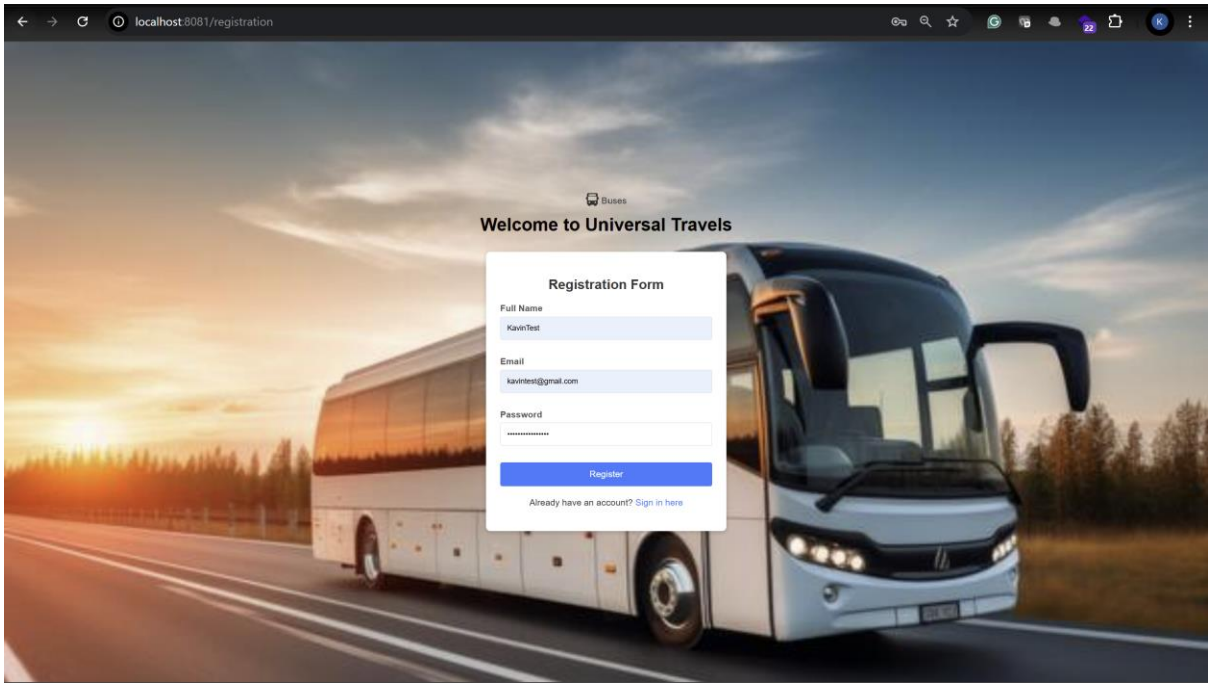
By clicking find the bus you will see the bus details, where the admin can edit or delete the bus

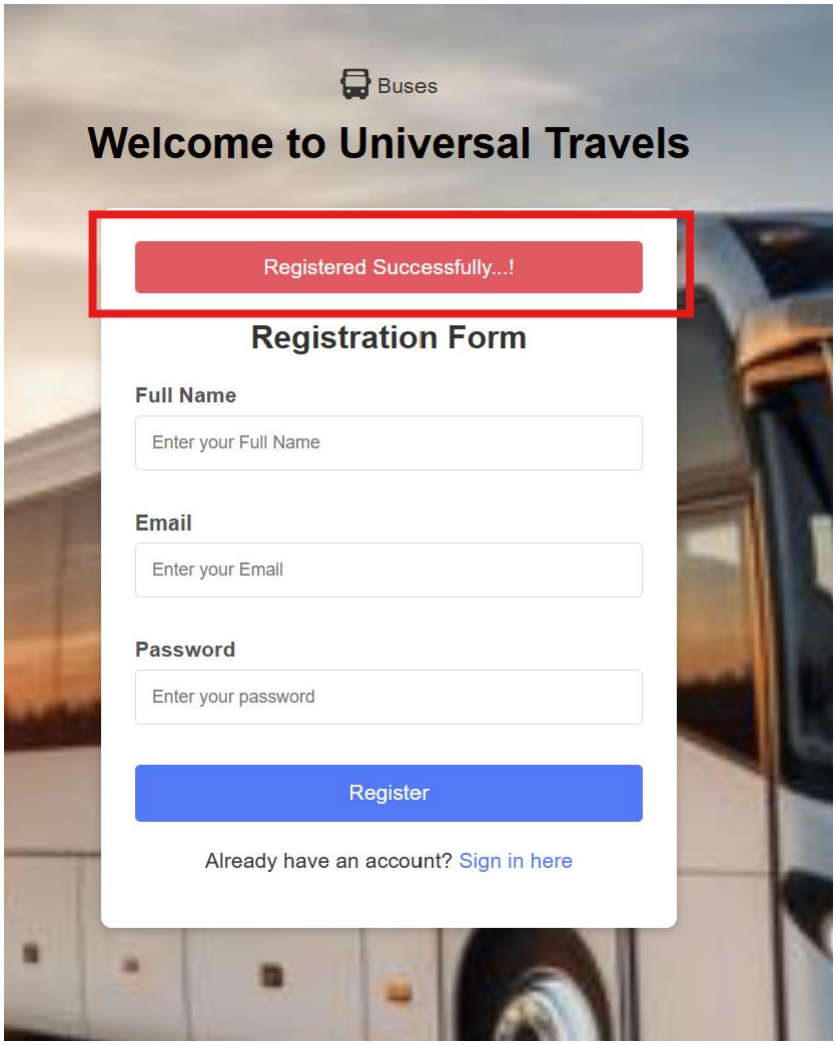
Once the admin click log-out from the top right page, they can logout successfully and redirected to below page



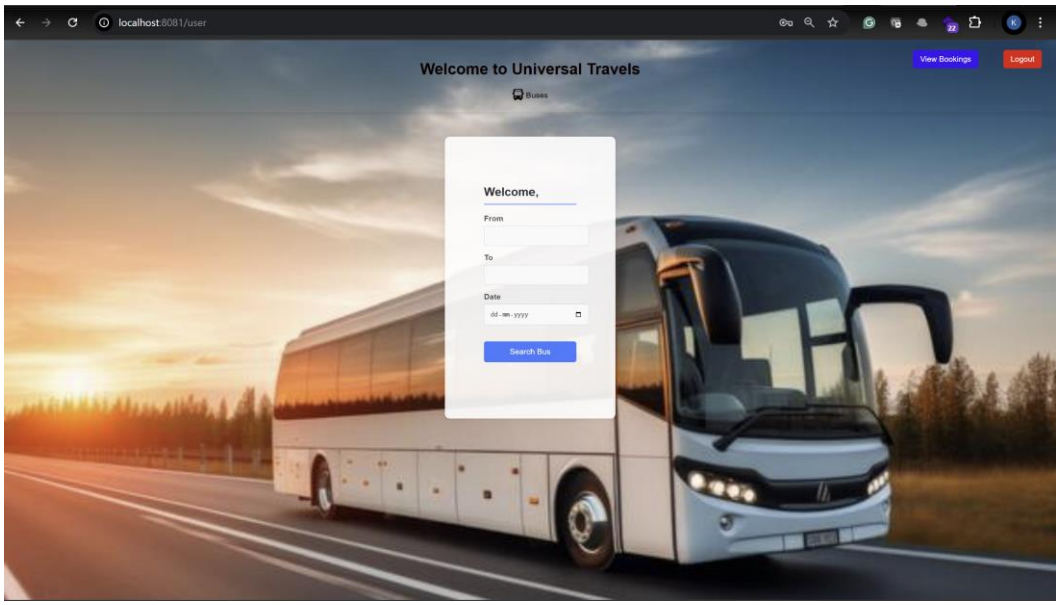
User sign-up and login

Click on signup from home page

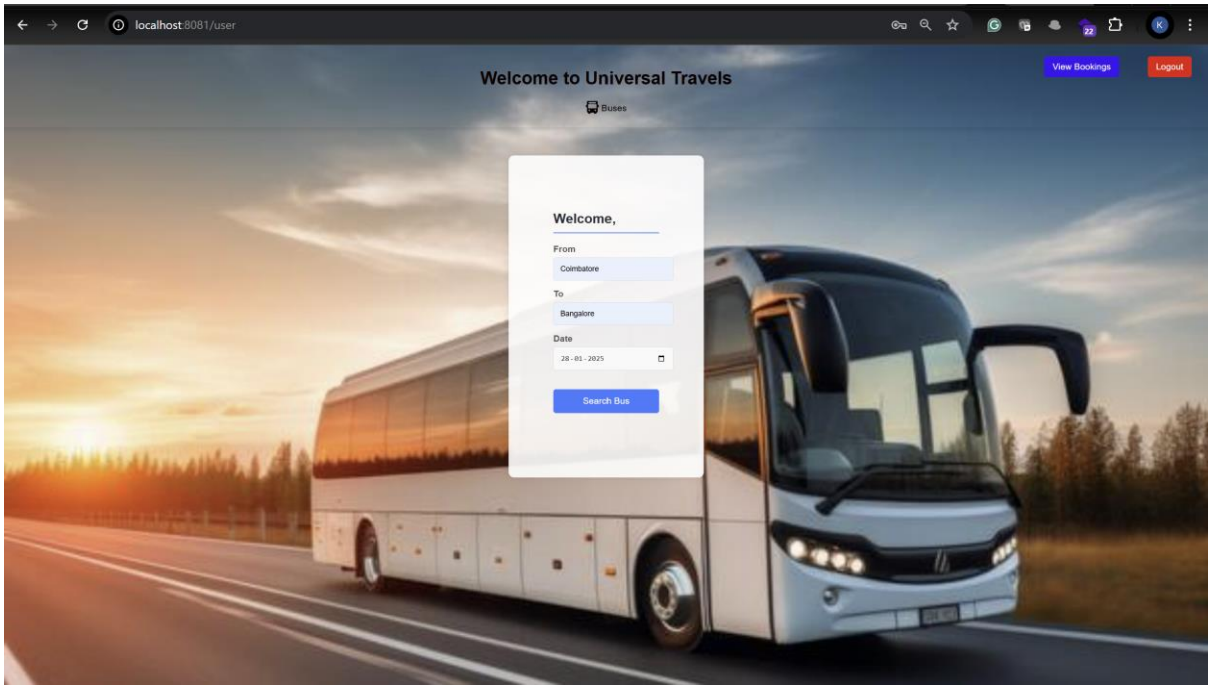




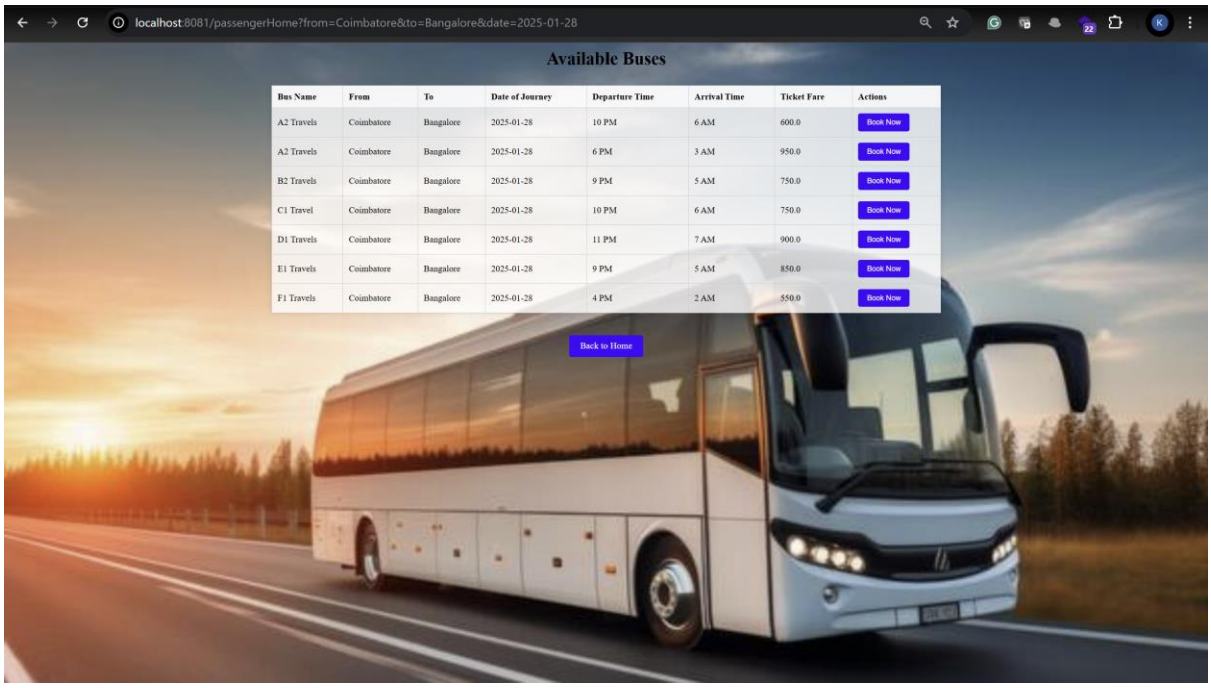
User Login page



By selecting the required details user can view the list of bus



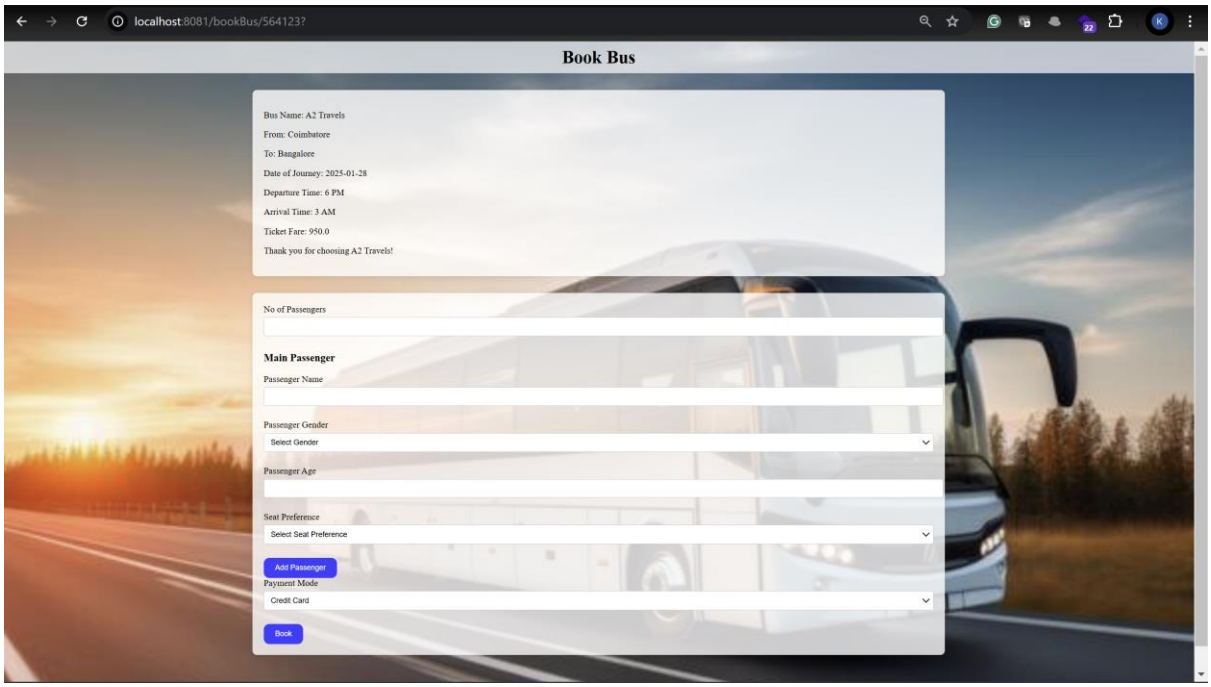
Result page



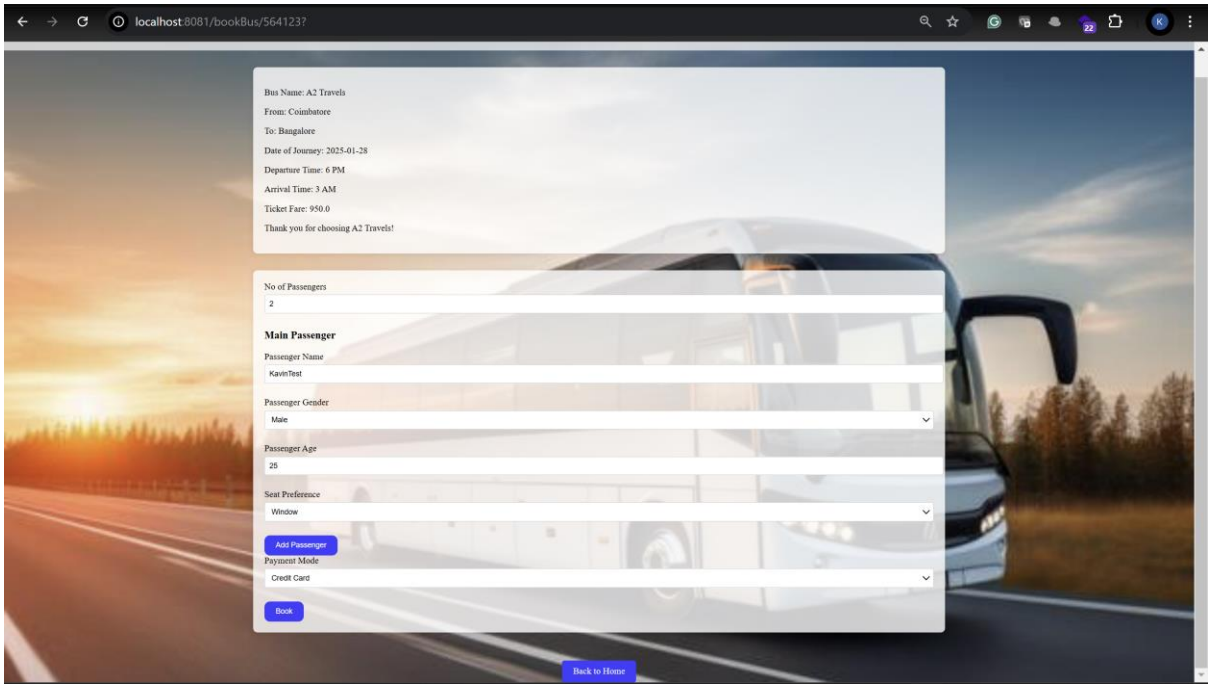
Bus Name	From	To	Date of Journey	Departure Time	Arrival Time	Ticket Fare	Actions
A2 Travels	Coimbatore	Bangalore	2025-01-28	10 PM	6 AM	600.0	Book Now
A2 Travels	Coimbatore	Bangalore	2025-01-28	6 PM	3 AM	950.0	Book Now
B2 Travels	Coimbatore	Bangalore	2025-01-28	9 PM	5 AM	750.0	Book Now
C1 Travel	Coimbatore	Bangalore	2025-01-28	10 PM	6 AM	750.0	Book Now
D1 Travels	Coimbatore	Bangalore	2025-01-28	11 PM	7 AM	900.0	Book Now
E1 Travels	Coimbatore	Bangalore	2025-01-28	9 PM	5 AM	850.0	Book Now
F1 Travels	Coimbatore	Bangalore	2025-01-28	4 PM	2 AM	550.0	Book Now

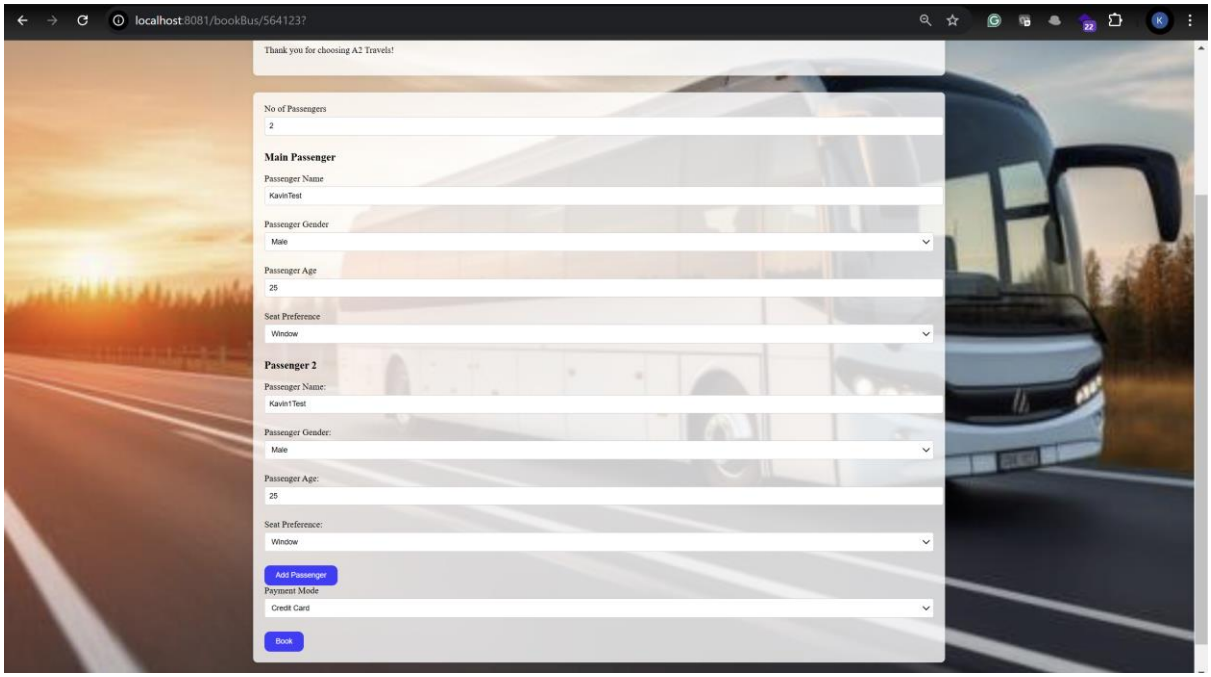
Here customer can choose a bus and book the ticket

By clicking the Book Now Button user will land on the below page where they can enter their details

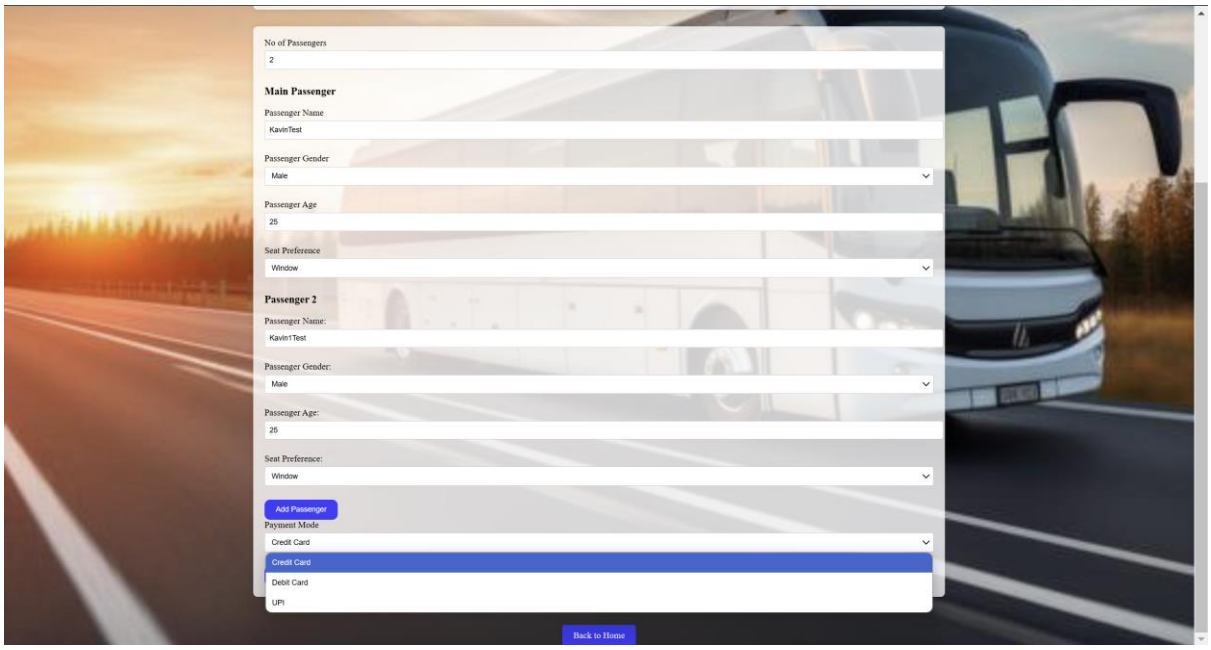


Once you enter the person 1 details you will be an option to add a passenger button to add as many person details as the user wants



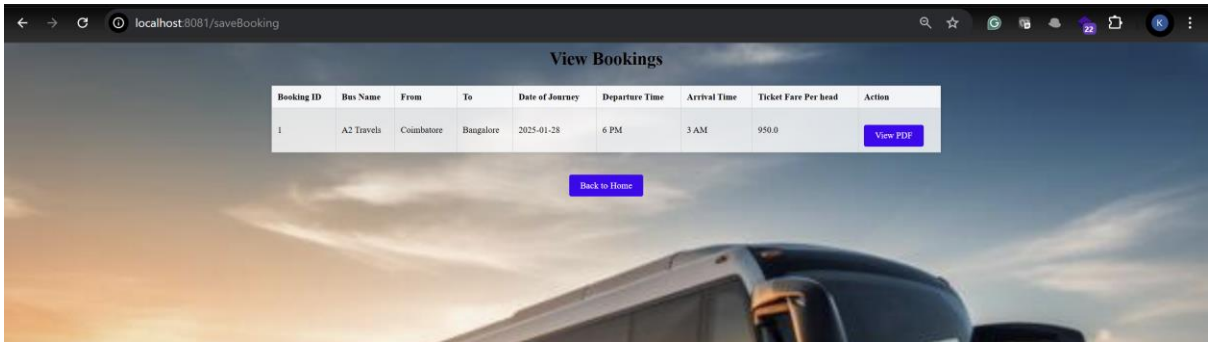


Next, you have to choose the payment method below

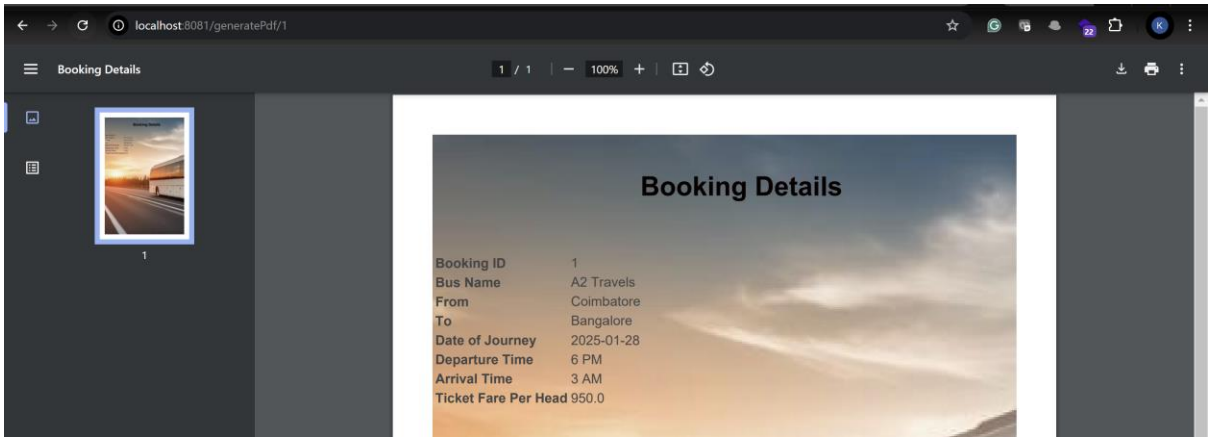


I choose credit card then click on book

Once you book the ticket you will see the confirmation and there is an option to view the ticket in pdf format



PDF will be downloaded and you can see the ticket as below



Swagger API Documentation for BookingController

1. Find Respective Buses

- **Endpoint**: GET /passengerHome
- **Description**: Finds buses based on origin, destination, and travel date.
- **Parameters**:
  - from (required, query parameter): The origin of the journey (e.g., "New York").
  - to (required, query parameter): The destination of the journey (e.g., "Boston").
  - date (required, query parameter): The date of travel (in ISO format: YYYY-MM-DD).
- **Responses**:
  - 200 OK: Returns the list of buses matching the search criteria.
  - 404 Not Found: No buses found for the given parameters.

2. Book Bus by Bus ID

- **Endpoint**: GET /bookBus/{busId}
- **Description**: Allows a user to select a bus for booking by its bus ID.
- **Parameters**:
  - busId (required, path parameter): The ID of the bus to be booked.
- **Responses**:
  - 200 OK: Displays the booking page with the selected bus details.

3. Save Booking

- **Endpoint**: POST /saveBooking
- **Description**: Saves a booking with passenger details.
- **Parameters** (all required):
  - busId (form parameter): The ID of the bus to be booked.
  - passengerId (form parameter): The ID of the passenger booking the bus.
  - noOfPassengers (form parameter): The number of passengers for this booking.
  - travelPassengerNames (form parameter, list): The names of the passengers.
  - travelPassengerGenders (form parameter, list): The genders of the passengers.
  - travelPassengerAges (form parameter, list): The ages of the passengers.
  - seatPreferences (form parameter, list): The seat preferences for the passengers.
  - paymentMode (form parameter): The payment mode (e.g., "Credit Card").
- **Responses**:
  - 200 OK: Booking saved successfully and redirects to the booking view page.

4. Get Bookings for Logged-in Passenger

- **Endpoint**: GET /bookings
- **Description**: Displays the bookings for the currently logged-in passenger.
- **Responses**:
  - 200 OK: Returns the list of bookings for the logged-in user.
  - 404 Not Found: No bookings found for the logged-in passenger.

5. Generate PDF Booking Confirmation

- **Endpoint**: GET /generatePdf/{bookingId}
- **Description**: Generates a PDF for the booking confirmation.
- **Parameters**:
  - bookingId (required, path parameter): The ID of the booking for which the PDF is generated.
- **Responses**:
  - 200 OK: Returns the booking confirmation PDF.
  - 404 Not Found: Booking not found with the provided ID.

Endpoint	Method	Response Code	Description
`/passengerHome`	`GET`	200	List of buses based on search criteria
`/bookBus/{busId}`	`GET`	200	Booking page for the selected bus
`/saveBooking`	`POST`	200	Booking saved successfully
`/bookings`	`GET`	200	List of bookings for logged-in passenger
`/generatePdf/{bookingId}`	`GET`	200	Booking confirmation PDF

```yaml
openapi: 3.0.0
info:
  title: Bus Ticket Booking API
  description: API documentation for the Bus Ticket Booking Application.
  version: 1.0.0
paths:
  /passengerHome:
    get:
      summary: "Find Respective Buses"
      parameters:
        - name: from
          in: query
          required: true
          description: "Origin of the journey."
          schema:
            type: string
        - name: to
          in: query
          required: true
          description: "Destination of the journey."
          schema:
            type: string
        - name: date
          in: query
          required: true
          description: "Date of travel (format: YYYY-MM-DD)."
          schema:
```

```yaml
        schema:
          type: string
          format: date
    responses:
      '200':
        description: "List of buses matching the criteria."
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Bus'
      '404':
        description: "No buses found for the given search."

  /bookBus/{busId}:
    get:
      summary: "Book a bus by Bus ID"
      parameters:
        - name: busId
          in: path
          required: true
          description: "The ID of the bus to be booked."
          schema:
            type: integer
            format: int32
      responses:
        '200':
          description: "Bus booking page with bus details."
          content:
            application/json:
              schema:
```

```yaml
            schema:
              $ref: '#/components/schemas/Bus'

/saveBooking:
  post:
    summary: "Save Bus Booking"
    parameters:
      - name: busId
        in: formData
        required: true
        description: "The ID of the bus being booked."
        schema:
          type: integer
      - name: passengerId
        in: formData
        required: true
        description: "The ID of the passenger making the booking."
        schema:
          type: integer
      - name: noOfPassengers
        in: formData
        required: true
        description: "Number of passengers."
        schema:
          type: integer
      - name: travelPassengerNames
        in: formData
        required: true
        description: "Names of passengers."
```

```yaml
        schema:
          type: array
          items:
            type: string
      - name: travelPassengerGenders
        in: formData
        required: true
        description: "Genders of passengers."
        schema:
          type: array
          items:
            type: string
      - name: travelPassengerAges
        in: formData
        required: true
        description: "Ages of passengers."
        schema:
          type: array
          items:
            type: integer
      - name: seatPreferences
        in: formData
        required: true
        description: "Seat preferences of passengers."
        schema:
          type: array
          items:
            type: string
      - name: paymentMode
        in: formData
```

```yaml
        - name: paymentMode
          in: formData
          required: true
          description: "Mode of payment (e.g., Credit Card)."
          schema:
            type: string
      responses:
        '200':
          description: "Booking saved successfully."

  /bookings:
    get:
      summary: "Get Bookings for Logged-in Passenger"
      responses:
        '200':
          description: "List of bookings for the logged-in passenger."
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Booking'
        '404':
          description: "No bookings found for the logged-in passenger."
```

```yaml
/generatePdf/{bookingId}:
  get:
    summary: "Generate PDF for Booking Confirmation"
    parameters:
      - name: bookingId
        in: path
        required: true
        description: "The ID of the booking."
        schema:
          type: integer
          format: int32
    responses:
      '200':
        description: "Booking confirmation PDF."
        content:
          application/pdf:
            schema:
              type: string
              format: binary
```

```yaml
components:
  schemas:
    Bus:
      type: object
      properties:
        id:
          type: integer
          description: "Bus ID"
        name:
          type: string
          description: "Name of the bus."
        origin:
          type: string
          description: "Origin location of the bus."
        destination:
          type: string
          description: "Destination location of the bus."
        travelDate:
          type: string
          format: date
          description: "Travel date of the bus."
        availableSeats:
          type: integer
          description: "Number of available seats."
```

```yaml
Booking:
  type: object
  properties:
    id:
      type: integer
      description: "Booking ID"
    passengerId:
      type: integer
      description: "Passenger ID"
    busId:
      type: integer
      description: "Bus ID"
    bookingDate:
      type: string
      format: date
      description: "Date when the booking was made."
    noOfPassengers:
      type: integer
      description: "Number of passengers booked."
    paymentMode:
      type: string
      description: "Payment mode used for the booking."
```

```yaml
PassengerDetail:
  type: object
  properties:
    id:
      type: integer
      description: "Passenger detail ID"
    name:
      type: string
      description: "Name of the passenger."
    gender:
      type: string
      description: "Gender of the passenger."
    age:
      type: integer
      description: "Age of the passenger."
    seatPreference:
      type: string
      description: "Seat preference for the passenger."
```

Swagger API Documentation for BusController

1. Get Admin Page

- **Endpoint**: GET /admin
- **Description**: Displays the admin page, which is accessible only to users with admin privileges.
- **Responses**:
  - 200 OK: Returns the admin page for the logged-in admin.

2. Show Bus Registration Form

- **Endpoint**: GET /addBus
- **Description**: Displays the form to add a new bus to the system.
- **Responses**:
  - 200 OK: Returns the page with the bus registration form.

3. Save Bus

- **Endpoint**: POST /saveBus
- **Description**: Saves a new bus to the database.
- **Parameters**:
  - bus (required, body): A Bus object containing bus details (e.g., bus number, origin, destination).
- **Responses**:
  - 200 OK: Successfully saves the bus and redirects to the admin page.
  - 400 Bad Request: Invalid bus details.

4. View All Buses

- **Endpoint**: GET /viewAllBuses
- **Description**: Displays all the buses in the system.
- **Responses**:
  - 200 OK: Returns a list of all buses.
  - 404 Not Found: No buses found in the system.

5. Show Search Form for Bus by Bus ID

- **Endpoint**: GET /findBusById
- **Description**: Displays the form to search for a bus by its ID.
- **Responses**:
  - 200 OK: Returns the page with the search form for bus ID.

6. Find Bus by ID

- **Endpoint**: GET /findBusId
- **Description**: Searches for a bus by its ID.
- **Parameters**:
  - busId (required, query parameter): The ID of the bus to be searched.
- **Responses**:
  - 200 OK: Returns the bus details if found.
  - 404 Not Found: Returns an error message if the bus is not found.

7. Show Bus Update Form

- **Endpoint**: GET /updateByBus
- **Description**: Displays the form to update bus details.
- **Parameters**:
  - busId (required, query parameter): The ID of the bus to be updated.
- **Responses**:
  - 200 OK: Displays the update form with existing bus details.
  - 404 Not Found: Returns an error message if the bus with the given ID is not found.

8. Update Bus Details

- **Endpoint**: PUT /saveUpdateBus
- **Description**: Updates the details of an existing bus.
- **Parameters**:
  - bus (required, body): The updated Bus object with new details.
- **Responses**:

- 200 OK: Successfully updates the bus and redirects to the view buses page.
- 400 Bad Request: Invalid bus details.
- 404 Not Found: Returns an error if the bus with the given ID is not found.

9. Delete Bus

- **Endpoint**: DELETE /delete/{serialNo}
- **Description**: Deletes a bus from the system based on the provided bus serial number.
- **Parameters**:
  - serialNo (required, path parameter): The serial number of the bus to be deleted.
- **Responses**:
  - 200 OK: Successfully deletes the bus and shows a success message.
  - 404 Not Found: Returns an error message if the bus with the given serial number is not found.

```yaml
openapi: 3.0.1
info:
  title: Bus Ticket Booking API
  description: API for managing bus ticket bookings, buses, and related functionalities f
  version: 1.0.0
servers:
  - url: http://localhost:8080
    description: Local server

paths:
  /admin:
    get:
      summary: "Get Admin Page"
      description: "Displays the admin page, accessible only by admins."
      responses:
        '200':
          description: "Admin page displayed successfully"
          content:
            text/html:
              schema:
                type: string
                example: "<html>Admin Page</html>"
```

```yaml
/addBus:
  get:
    summary: "Show Bus Registration Form"
    description: "Displays the form to add a new bus."
    responses:
      '200':
        description: "Bus registration form displayed successfully"
        content:
          text/html:
            schema:
              type: string
              example: "<html>Bus Registration Form</html>"
```

```yaml
/saveBus:
  post:
    summary: "Save Bus"
    description: "Saves a new bus to the system."
    operationId: "saveBus"
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Bus'
    responses:
      '200':
        description: "Bus saved successfully"
        content:
          text/html:
            schema:
              type: string
              example: "<html>Bus successfully added</html>"
      '400':
        description: "Invalid bus details"
```

```yaml
/viewAllBuses:
  get:
    summary: "View All Buses"
    description: "Displays all buses available in the system."
    responses:
      '200':
        description: "List of buses displayed successfully"
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/Bus'
      '404':
        description: "No buses found"
```

```yaml
/findBusById:
  get:
    summary: "Show Bus Search Form"
    description: "Displays the form to search for a bus by ID."
    responses:
      '200':
        description: "Search form for bus ID displayed successfully"
        content:
          text/html:
            schema:
              type: string
              example: "<html>Find Bus by ID Form</html>"
```

```yaml
/findBusId:
  get:
    summary: "Find Bus by ID"
    description: "Searches for a bus by its ID."
    parameters:
      - name: "busId"
        in: query
        description: "The ID of the bus to search for"
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: "Bus found successfully"
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Bus'
      '404':
        description: "Bus not found"
```

```yaml
/updateByBus:
  get:
    summary: "Show Bus Update Form"
    description: "Displays the form to update an existing bus."
    parameters:
      - name: "busId"
        in: query
        description: "The ID of the bus to update"
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: "Update form displayed successfully"
        content:
          text/html:
            schema:
              type: string
              example: "<html>Update Bus Form</html>"
      '404':
        description: "Bus not found"
```

```yaml
/saveUpdateBus:
  put:
    summary: "Update Bus Details"
    description: "Updates the details of an existing bus."
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Bus'
    responses:
      '200':
        description: "Bus updated successfully"
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Bus'
      '400':
        description: "Invalid bus details"
      '404':
        description: "Bus not found"
```

```yaml
/delete/{serialNo}:
  delete:
    summary: "Delete Bus"
    description: "Deletes a bus based on its serial number."
    parameters:
      - name: "serialNo"
        in: path
        description: "The serial number of the bus to delete"
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: "Bus deleted successfully"
      '404':
        description: "Bus not found"
```

```yaml
components:
  schemas:
    Bus:
      type: object
      properties:
        busId:
          type: integer
          description: "The ID of the bus."
        busNumber:
          type: string
          description: "The number of the bus."
        origin:
          type: string
          description: "The origin city."
        destination:
          type: string
          description: "The destination city."
        capacity:
          type: integer
          description: "The seating capacity of the bus."
        travelDate:
          type: string
          format: date
          description: "The date of travel."
```

```yaml
    Passenger:
      type: object
      properties:
        passengerId:
          type: integer
          description: "The unique ID of the passenger."
        firstName:
          type: string
          description: "First name of the passenger."
        lastName:
          type: string
          description: "Last name of the passenger."
        gender:
          type: string
          description: "Gender of the passenger."
        age:
          type: integer
          description: "Age of the passenger."
```

Swagger API Documentation for PassengerController

1. Show Registration Form

- **Endpoint**: GET /registration
- **Description**: Displays the user registration form.
- **Parameters**: None
- **Responses**:
  - 200 OK: Returns the registration page with the form.

2. Save Passenger (Register New User)

- **Endpoint**: POST /registration
- **Description**: Saves the passenger registration details.
- **Parameters**:
  - userDto (required, form parameter): The user registration details including:
    - username: The desired username for the user.
    - password: The password for the new user.
    - email: The email address of the user.
    - role: The role of the user (always set as "USER").
- **Responses**:
  - 200 OK: Registration successful. A success message is displayed.
  - 400 Bad Request: Invalid or missing data for user registration.

3. Login Page

- **Endpoint**: GET /login
- **Description**: Displays the login page.
- **Parameters**: None
- **Responses**:
  - 200 OK: Returns the login page.

4. User Home Page

- **Endpoint**: GET /user
- **Description**: Displays the user home page for the logged-in passenger, including their details.
- **Parameters**:
  - principal: The currently logged-in user (automatically provided by Spring Security).
- **Responses**:
  - 200 OK: Returns the user's home page with the passenger's details.

```yaml
openapi: 3.0.0
info:
  title: Bus Ticket Booking API
  description: API documentation for the Bus Ticket Booking Application.
  version: 1.0.0
servers:
  - url: http://localhost:8080
    description: Local Development Server

paths:
  /registration:
    get:
      summary: "Display Registration Form"
      description: "Displays the registration form for new users."
      operationId: getRegistrationForm
      responses:
        '200':
          description: "Registration form displayed successfully"
          content:
            text/html:
              schema:
                type: string
                example: "registration form HTML content"
```

```yaml
  post:
    summary: "Save User Registration"
    description: "Saves the new user registration details."
    operationId: savePassenger
    requestBody:
      required: true
      content:
        application/x-www-form-urlencoded:
          schema:
            type: object
            properties:
              username:
                type: string
                example: "john_doe"
              password:
                type: string
                example: "password123"
              email:
                type: string
                example: "johndoe@example.com"
              role:
                type: string
                example: "USER"
```

```yaml
      responses:
        '200':
          description: "User registered successfully"
          content:
            text/html:
              schema:
                type: string
                example: "Registration successful, user redirected to register page."
        '400':
          description: "Invalid input"
          content:
            text/html:
              schema:
                type: string
                example: "Registration failed due to invalid data."
```

```yaml
/login:
  get:
    summary: "Display Login Page"
    description: "Displays the login page for users."
    operationId: login
    responses:
      '200':
        description: "Login page displayed successfully"
        content:
          text/html:
            schema:
              type: string
              example: "login form HTML content"
```

```yaml
/user:
  get:
    summary: "Display User Home Page"
    description: "Displays the home page for the logged-in user with their details."
    operationId: usersPage
    parameters:
      - name: principal
        in: header
        description: "Principal (Logged-in User)"
        required: true
        schema:
          type: string
          example: "john_doe"
    responses:
      '200':
        description: "User home page displayed successfully"
        content:
          text/html:
            schema:
              type: string
              example: "User home page content with details."
```

```yaml
components:
  schemas:
    UserDto:
      type: object
      properties:
        username:
          type: string
          description: "The username of the user."
          example: "john_doe"
        password:
          type: string
          description: "The password of the user."
          example: "password123"
        email:
          type: string
          description: "The email address of the user."
          example: "johndoe@example.com"
        role:
          type: string
          description: "The role assigned to the user."
          example: "USER"
```

```yaml
responses:
  UserRegistered:
    description: "Successfully registered user."
    content:
      application/json:
        schema:
          type: object
          properties:
            message:
              type: string
              example: "Registered Successfully...!"
```

Schema Structure

Table: **Booking_Details**

Column Name	Data Type	Constraints	Description
`Booking_Id`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the booking.
`bus_Id`	INTEGER	FOREIGN KEY (Bus.Bus_Id)	Foreign key linking to the `Bus` entity.
`Passenger_Id`	INTEGER	FOREIGN KEY (Passengers.Passenger_Id)	Foreign key linking to the `Passengers` entity.
`Booking_Date`	DATE	NOT NULL	Date the booking was made.
`Payment_Mode`	VARCHAR(50)		The payment mode for the booking (e.g., "Credit Card", "Cash").
`no_Of_Passengers`	INTEGER		The number of passengers for the booking.

Foreign Key Relationships:

- **Bus Table**:
  - bus_Id is a foreign key referencing Bus(bus_Id).
- **Passengers Table**:
  - Passenger_Id is a foreign key referencing Passengers(Passenger_Id).

Bus Table Schema

Column Name	Data Type	Constraints	Description
`Bus_Id`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each bus.
`Bus_Name`	VARCHAR(255)	NOT NULL	Name of the bus.
`Bus_Type`	VARCHAR(100)		Type of bus (e.g., sleeper, deluxe).
`Origin`	VARCHAR(100)	NOT NULL	The origin location for the bus.
`Destination`	VARCHAR(100)	NOT NULL	The destination location for the bus.
`Seats_Available`	INTEGER	NOT NULL	Number of available seats on the bus.

Passengers Table Schema

Column Name	Data Type	Constraints	Description
`Passenger_Id`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each passenger.
`Name`	VARCHAR(255)	NOT NULL	Name of the passenger.
`Email`	VARCHAR(255)	NOT NULL, UNIQUE	Email address of the passenger.
`Phone_Number`	VARCHAR(15)	NOT NULL	Phone number of the passenger.
`Password`	VARCHAR(255)	NOT NULL	Password for user authentication.
`Role`	VARCHAR(20)	NOT NULL	Role of the user (e.g., USER, ADMIN).

PassengerDetail Table Schema

Column Name	Data Type	Constraints	Description
`Passenger_Detail_Id`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each passenger detail.
`Booking_Id`	INTEGER	FOREIGN KEY (Booking_Details.Booking_Id)	Foreign key referencing the `Booking` table.
`Passenger_Name`	VARCHAR(255)	NOT NULL	Name of the passenger for this detail.
`Passenger_Gender`	VARCHAR(10)	NOT NULL	Gender of the passenger (e.g., Male, Female).
`Passenger_Age`	INTEGER	NOT NULL	Age of the passenger.
`Seat_Preference`	VARCHAR(50)		The seat preference (e.g., Window, Aisle).

ERD (Entity Relationship Diagram)

Here's the relationship breakdown between the Booking model and its associated tables:

- **Booking ↔ Bus**: Many-to-One relationship. A booking is related to a single bus.
- **Booking ↔ Passengers**: Many-to-One relationship. A booking is related to a single passenger.
- **Booking ↔ PassengerDetail**: One-to-Many relationship. A booking can have multiple passenger details.

ERD Overview:

- **Booking** has foreign keys to both **Bus** and **Passengers**.
- **Booking** has a one-to-many relationship with **PassengerDetail**.

```yaml
Booking:
  type: object
  properties:
    bookingId:
      type: integer
      description: The unique identifier for the booking.
      example: 1
    busEntity:
      $ref: '#/components/schemas/Bus'
    passengers:
      $ref: '#/components/schemas/Passengers'
    passengerDetails:
      type: array
      items:
        $ref: '#/components/schemas/PassengerDetail'
    bookingDate:
      type: string
      format: date
      description: The date when the booking was made.
      example: '2025-01-27'
    paymentMode:
      type: string
      description: The mode of payment for the booking (e.g., credit card, cash, etc.)
      example: 'Credit Card'
    noOfPassengers:
      type: integer
      description: The number of passengers associated with the booking.
      example: 3
  required:
    - bookingId
    - busEntity
    - passengers
    - bookingDate
    - noOfPassengers
```

Table: **BusDetails**

Column Name	Data Type	Constraints	Description
`serial_No`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the bus (used as the primary key).
`bus_Id`	INTEGER	UNIQUE, NOT NULL	Unique identifier for the bus.
`bus_Name`	VARCHAR(255)		Name of the bus (e.g., "Express", "Deluxe").
`departureLocation`	VARCHAR(255)		The location from where the bus departs.
`arrivalLocation`	VARCHAR(255)		The location where the bus arrives.
`ticketFar`	DECIMAL(10, 2)		The price of the bus ticket.
`checkingDate`	DATE		The date when the bus is checked for availability.
`departureTime`	VARCHAR(50)		The time the bus departs.
`arrivalTime`	VARCHAR(50)		The time the bus arrives.

ERD (Entity Relationship Diagram)

- **Bus** is an entity that represents bus details.
- It is related to **Booking** through the bus_Id as a foreign key in the Booking_Details table.
- Each **Bus** can have many bookings associated with it, but a **Booking** belongs to only one bus.

Booking Table Schema

- **Bus Table**:
    - bus_Id is a foreign key in Booking_Details, referencing Bus(bus_Id).

```yaml
Bus:
  type: object
  properties:
    serialNo:
      type: integer
      description: The unique serial number for the bus (Primary Key).
      example: 101
    busId:
      type: integer
      description: The unique identifier for the bus.
      example: 2025
    busName:
      type: string
      description: The name of the bus (e.g., "Express", "Deluxe").
      example: "Deluxe Express"
    from:
      type: string
      description: The location from which the bus departs.
      example: "New York"
    to:
      type: string
      description: The location where the bus arrives.
      example: "Washington DC"
    ticketFar:
      type: number
      format: float
      description: The price of the bus ticket.
      example: 50.75
    checkingDate:
      type: string
      format: date
      description: The date when the bus is available for booking.
      example: "2025-05-15"
    departureTime:
      type: string
      description: The time the bus departs.
      example: "08:00 AM"
    arrivalTime:
      type: string
      description: The time the bus arrives.
      example: "02:00 PM"
  required:
    - serialNo
    - busId
    - busName
    - from
    - to
    - ticketFar
    - checkingDate
    - departureTime
    - arrivalTime
```

Table: **Passenger_Details**

Column Name	Data Type	Constraints	Description
`detailId`	INTEGER	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the passenger detail record (Primary Key).
`Booking_Id`	INTEGER	FOREIGN KEY (Booking)	Foreign key referencing the `Booking` table (booking associated with this detail).
`travel_PassengerName`	VARCHAR(255)		Name of the passenger traveling on the bus.
`Gender`	VARCHAR(10)		Gender of the passenger (e.g., "Male", "Female").
`Age`	INTEGER	NOT NULL	Age of the passenger.
`seat_Preference`	VARCHAR(50)		The seat preference of the passenger (e.g., "Window", "Aisle").

ERD (Entity Relationship Diagram)

- **PassengerDetail** is an entity that represents detailed information about individual passengers in a booking.
- It has a **Many-to-One** relationship with the **Booking** model, where multiple passenger details can belong to one booking.

Relationships:

- **PassengerDetail ↔ Booking**
    - Each **Booking** can have multiple **PassengerDetails**, while each **PassengerDetail** belongs to one **Booking**.

Booking Table Schema

- **Booking Table**:
    - Booking_Id is referenced as a foreign key in the Passenger_Details table.

```yaml
PassengerDetail:
  type: object
  properties:
    detailId:
      type: integer
      description: The unique identifier for the passenger detail record (Primary Key).
      example: 1
    booking:
      type: object
      description: The booking to which the passenger detail belongs.
      properties:
        bookingId:
          type: integer
          description: The ID of the booking.
          example: 101
    travelPassengerName:
      type: string
      description: The name of the traveling passenger.
      example: "John Doe"
    travelPassengerGender:
      type: string
      description: The gender of the passenger.
      example: "Male"
    travelPassengerAge:
      type: integer
      description: The age of the passenger.
      example: 30
    seatPreference:
      type: string
      description: The seat preference of the passenger.
      example: "Window"
  required:
    - detailId
    - booking
    - travelPassengerName
    - travelPassengerGender
    - travelPassengerAge
    - seatPreference
```

Table: Passengers

Column Name	Data Type	Constraints	Description
`Passenger_Id`	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for the passenger (Primary Key).
`full_name`	VARCHAR(255)	NOT NULL	Full name of the passenger.
`email`	VARCHAR(255)	UNIQUE, NOT NULL	Email of the passenger (unique constraint).
`password`	VARCHAR(255)	NOT NULL	Password for the passenger (hashed).
`role`	VARCHAR(50)	NOT NULL	Role of the passenger (e.g., "USER", "ADMIN").

ERD (Entity Relationship Diagram)

- **Passengers** is an entity that stores the details of the passengers.
- It has a **One-to-Many** relationship with the **Booking** model (each passenger can have multiple bookings).

Relationships:

- **Passengers ↔ Booking**
    - A passenger can have multiple bookings. Each booking is linked to a passenger.

```yaml
Passengers:
  type: object
  properties:
    passengerId:
      type: integer
      description: The unique identifier for the passenger.
      example: 1
    fullName:
      type: string
      description: The full name of the passenger.
      example: "John Doe"
    email:
      type: string
      description: The email address of the passenger (unique).
      example: "johndoe@example.com"
    password:
      type: string
      description: The password for the passenger (hashed).
      example: "password123"
    role:
      type: string
      description: The role of the passenger (e.g., "USER", "ADMIN").
      example: "USER"
  required:
    - fullName
    - email
    - password
    - role
```