Blue Bus Reservation application

Technologies used:

Front end-**Visual code and react**

Backend-**Intelliji-Springboot**

Database**- MySQL and MySQL workbench**

🡪Blue Bus reservation system is a software application or platform designed to facilitate the booking of bus ticket by passengers .It streamlines the process of ticket booking ,seat selection ,payment processing and managing reservations for bus travel.

1. Bus Routes and Schedules: It provide information about various bus routes, schedules, departure times, and arrival times for passengers to choose from.

2. User Interface: The system should have an easy-to-use interface for passengers to search for available buses, select departure and arrival locations, choose travel dates, and view seat availability.

3. Seat Availability: The system display seat availability for each bus trip, allowing passengers to select preferred seats from the available options.

4. Booking and Reservation: Passengers is able to make reservations and book tickets for their desired bus trips through the system. This includes providing passenger details, selecting seats, and confirming the booking.

5. Seat Selection: Passengers should be able to choose their preferred seats from an interactive seating layout displayed during the booking process.

6. User Accounts: Registration and login functionality for passengers to create accounts, manage their bookings, view booking history, and update personal information.

7. Admin Dashboard: An administrative interface for bus operators or system administrators to manage bus schedules, add or remove Bus, configure ticket prices. view bookings, generate reports, and perform other administrative tasks.

8. Security: Implement Jwt security measures to protect user data, secure payment transactions, and prevent unauthorized access to the system Accessibility: Investigate

To download a frontend project from GitHub using Visual Studio Code, you can follow these steps:

1. **Install Git**: Make sure you have Git installed on your system. You can download and install it from the official website.
2. **Open Visual Studio Code**: Launch Visual Studio Code on your computer.
3. **Open Terminal in Visual Studio Code**:
   * Press **Ctrl + ~** (or **View** > **Terminal**).
   * This will open a terminal window at the bottom of the Visual Studio Code interface.
4. **Navigate to the Desired Directory**: Use the **cd** command to navigate to the directory where you want to clone the project.th/to/your/directory
5. **Clone the Repository**: Use the **git clone** command followed by the URL of the GitHub repository you want to clone. For example:github.com/username/repository.git

Replace **username/repository** with the actual username and repository name of the GitHub project.

1. **Open the Project in Visual Studio Code**:
   * Once the cloning process is complete, you can open the project in Visual Studio Code by either selecting **File** > **Open Folder** and navigating to the cloned directory, or by using the command line:
2. **Install Dependencies (if any)**: If the project has dependencies listed in a **package.json** file, you'll need to install them. Open a new terminal in Visual Studio Code and run:

This will install all the required dependencies listed in the **package.json** file.

1. **Start the Development Server (if applicable)**: Depending on the project setup, you may need to start a development server to view the project in your browser.

Dependency :

**Jwt Token,Spring security:**

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>0.12.5</version>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>0.12.5</version>  
 <scope>runtime</scope>  
</dependency>  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>0.12.5</version>  
 <scope>runtime</scope>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

**To download a backend project in IntelliJ IDEA, you typically follow these steps:**

1. **Install IntelliJ IDEA**: If you haven't already, download and install IntelliJ IDEA from the official website: <https://www.jetbrains.com/idea/download/>
2. **Open IntelliJ IDEA**: After installation, open IntelliJ IDEA from your applications folder.
3. **Clone the Project**: If the project is hosted on a version control system like Git, you'll need to clone it. Here's how to do it:

a. Go to **VCS** menu at the top.

b. Click on **Get from Version Control**.

c. Choose the version control system your project is hosted on (e.g., Git, GitHub, Bitbucket, etc.).

d. Enter the repository URL and specify the directory where you want to clone the project.

e. Click on **Clone**.

1. **Open Existing Project**: If you already have the project downloaded, you can simply open it from IntelliJ IDEA:

a. Go to **File** menu.

b. Click on **Open**.

c. Navigate to the directory where your project is located.

d. Select the project folder and click **OK**.

1. **Wait for IntelliJ to Load the Project**: Once you've opened the project, IntelliJ IDEA will take some time to index and load all the necessary files and dependencies.
2. **Configure Project Settings (if necessary)**: Depending on the project setup, you might need to configure project SDK, language level, project structure, etc. You can do this by going to **File** > **Project Structure**.
3. **Install Dependencies**: If the project relies on external dependencies managed by build tools like Maven or Gradle, IntelliJ will typically recognize them and prompt you to download them automatically. Otherwise, you might need to manually configure dependencies.
4. **Build the Project**: After setting up the project, you may need to build it to ensure that all dependencies are resolved and the project is compiled successfully. You can do this by going to **Build** > **Build Project** or by pressing **Ctrl + F9** (Cmd + F9 on Mac).
5. <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
   </dependency>  
     
   <dependency>  
    <groupId>com.mysql</groupId>  
    <artifactId>mysql-connector-j</artifactId>  
    <scope>runtime</scope>  
   </dependency>  
   <dependency>  
    <groupId>org.projectlombok</groupId>  
    <artifactId>lombok</artifactId>  
    <optional>true</optional>  
   </dependency>  
   <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-test</artifactId>  
    <scope>test</scope>  
   </dependency>

**Front end:**

|  |
| --- |
| **Note**:  If my application throws AXIOS ERROR-403,you can directly go to Browser🡪Inspect-🡪Application🡪clear Local storage and session storage.So that,AXIOS ERROR will be resolved. |

**Terminals:**

Node 1:

C:\Users\Kumaresa\capstone-react> cd demo

PS C:\Users\Kumaresa\capstone-react\demo> **npm start**

Node 2:

C:\Users\Kumaresa\capstone-react> cd demo

PS C:\Users\Kumaresa\capstone-react\demo> **npm install react-router-dom --save**

Node 3:

C:\Users\Kumaresa\capstone-react> cd demo

PS C:\Users\Kumaresa\capstone-react\demo> **npm install axios –save**

Node 4:

C:\Users\Kumaresa\capstone-react> cd demo

PS C:\Users\Kumaresa\capstone-react\demo> **npm install bootstrap --save**

**URL:**

**Front Page** : <http://localhost:3000/front>

**Sign up**: <http://localhost:3000/reg>

New User can sign up.

All registered user can be viewed by admin.(<http://localhost:3000/hello>)

Individual User can view his Profile,by clicking **MyProfile**.

|  |
| --- |
| **Login**: <http://localhost:3000/login>    **ADMIN:**  **Username**: admin  Password: admin  **User:**  Registered User(by Sign Up) |

**ADMIN:**

**Front page (admin):** <http://localhost:3000/admin>

**Assign Routes**: <http://localhost:3000/suc>

Bus details given by admin.

**Bus list**: <http://localhost:3000/list>

To display different Buses.

**a.update**: [http://localhost:3000/update-bus/{id}](http://localhost:3000/update-bus/%7bid%7d)

To update the bus details

**b.delete**:

Deleted by refresh

**Booking History**: <http://localhost:3000/booking>

To view all bookings by admin

**Users Profile**: <http://localhost:3000/hello>

To view all users by Admin.

|  |
| --- |
| **Note**:Once entered as admin,please log out and enter as user. |

**User:**

**FROM,TO Bus routes**: <http://localhost:3000/bus>

He can select any routes and is displayed in table.

**Add Passenger**:

To add passenger detail.

If we add 2nd user,we will get counted by counter.

ll’y, we add 3rd,4th,5th ,we get Passenger{count}.

Based on no.of Passenger added,total Fare will be calculated.

a.**Update Passenger**:

We can update Passenger details ,by clicking update.

b.**Back**:

when we click,we navigate to passenger list.

**Payment**: <http://localhost:3000/success>

It navigate to success page,where we can download e-tickets.

**Header Component:**

a.**Download e-ticket**: <http://localhost:3000/tic>

By giving ,emailId of Passenger,we can download ticket.

b.**MyProfile**: <http://localhost:3000/label>

-Give your registered email of userlogin ,to view Users profile.

1.By Navigate to: [http://localhost:3000/user123/{email}](http://localhost:3000/user123/%7bemail%7d),

User can view Users profile data,given during login register.

2.on clicking update,it navigate [http://localhost:3000/change-password/{username}](http://localhost:3000/change-password/%7busername%7d),Where we can submit,new password.

.

c.**Booking History**: <http://localhost:3000/my>

-Give your email Id of Passenger,we can view Booking history of us.

d.**Logout**:

After logout,you can go to login to go to Admin Component.

Because,once loggedout,both local & session storage will be cleared.

|  |
| --- |
| **Note**:  If my application throws AXIOS ERROR-403,you can directly go to Browser🡪Inspect-🡪Application🡪clear Local storage and session storage.So that,AXIOS ERROR will be resolved. |

**Validation:**

 try {

      if (name === "") {

        setError(true);

      } else if (username === "") {

        setError(true);

      } else if (email === "") {

        setError(true);

      } else if (password === "") {

        setError(true);

      } else {

If the given input is empty ,it wont get registered.

If accepts only unique email id of registered user.

If username is invalid ,it wont login the page.

**JWT TOKEN:**

To login using a JWT (JSON Web Token), these are steps:

1. **Authentication:** Authenticate the user using their credentials (username/email and password).
2. **Generate Token:** Upon successful authentication, generate a JWT containing relevant user information and any additional data necessary (e.g., user roles, permissions, etc.).
3. **Token Sending:** Send the JWT to the client (usually in the response body or headers).
4. **Client Storage:** Store the JWT securely on the client-side, typically in local storage or a cookie.
5. **Token Sending with Subsequent Requests:** Include the JWT in the headers of subsequent requests to the server for protected resources.
6. **Token Validation:** On the server-side, validate the JWT for each incoming request to ensure that it has not been tampered with and that it is still valid.
7. **Access Control:** Use the information within the JWT to determine whether the user has permission to access the requested resource.

Back End:

**Setting up a backend for a bus booking application using Spring Boot involves several steps:**

A. **Create a new Spring Boot project:**

* Use Spring Initializr (<https://start.spring.io/>) to create a new Spring Boot project.
* Choose the required dependencies :
* Spring Web,
* Spring Data JPA (for database interaction),
* LOMBOK
* MYSQL Driver
* B. **Define entities and data models:**
* Create entity classes for Bus, Booking, Passenger, and any other relevant entities.
* Use annotations such as **@Entity**, **@Id**, **@GeneratedValue**
* -**Passenger**
* **-Bus**
* **-Booking**
* ., to define the data models and relationships between entities.
* Define relationships such as one-to-many (e.g., a Bus has multiple Bookings) using annotations like **@OneToMany**, **@ManyToOne**, etc.
* **-IN our project**:
* **@ManyToOne mapping between Booking-Passenger & Booking-Bus**

C. **Implement RESTful APIs using Spring Boot's controllers:**

* Create Spring Boot controller classes to handle various endpoints for bus booking requests, user authentication, and data retrieval.
* Use annotations such as **@RestController**, **@RequestMapping**, **@GetMapping**, **@PostMapping**, etc., to define endpoints and HTTP methods.
* Implement methods within the controllers to perform actions like creating bookings, retrieving bus schedules, authenticating users, etc.

D. **Set up a database:**

* Choose a database (e.g., MySQL) to store bus schedules, booking information, and user data.
* Configure database connection properties in the **application.properties** file:

spring.datasource.url=jdbc:mysql://localhost:3306/obra  
spring.datasource.username=root  
spring.datasource.password=root3306  
spring.jpa.show-sql: true  
spring.jpa.hibernate.ddl-auto=update  
server.port=8083  
  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

Use Spring Data JPA repositories or MongoDB repositories to interact with the database.

* Define database tables/collections corresponding to your entities, and let Spring Boot handle the schema creation/update using Hibernate or MongoDB's schema-less nature.

Insert MySQL comment for many-many mappings 🡪 user- roles table

|  |
| --- |
| create database obra;  use obra;  insert into users(email,name,password,username)values("admin@gmail.com","admin","$2a$10$F.NlXYddsdivrBWoC.6L4uoF9XB89s5ZWLJcqZozR6IFbEQkR9awW","admin");  insert into users(email,name,password,username)values("john@gmail.com","user","$2a$10$d.8g5aedjLEwkjyM8G6TG.FcPrw3/vNSlSUa1ZbVaQx9kFaj2yn8q","john");  insert into roles(name)values("ROLE\_ADMIN");  insert into roles(name)values("ROLE\_USER");  insert into users\_roles values(1,1);  insert into users\_roles values(1,2);  insert into users\_roles values(2,2); |

**API ENDPOINTS:**

**POST url**:

**Passenger**: http://localhost:8083/api/passengers/savePassenger

**Bus**: http://localhost:8083/api/buses/saveBus

**Booking**: http://localhost:8083/api/bookings/saveBooking

**Login**: http://localhost:8083/api/auth/login

**Register**: http://localhost:8083/api/auth/register

**GET:**

Bus :<http://localhost:8083/api/buses>

Passenger

GET ALL :<http://localhost:8083/api/passengers>

GET BY ID: http://localhost:8083/api/passengers/{id}

GET BY EMAIL:<http://localhost:8083/api/passengers/email/{email}>

Booking:

GET BY EMAIL: [http://localhost:8083/api/bookings/email/{email}](http://localhost:8083/api/bookings/email/%7bemail%7d)

GET BY ID: [http://localhost:8083/api/bookings/{id}](http://localhost:8083/api/bookings/%7bid%7d) (giving booking id )

GET BY PASSENGER ID: http://localhost:8083/api/bookings/id/{id} (giving passenger id)

**Register(user):**

GET ALL: <http://localhost:8083/api/auth>

GET BY EMAIL: [http://localhost:8083/api/auth/email/{email}](http://localhost:8083/api/auth/email/%7bemail%7d)

Get by id: [http://localhost:8083/api/auth/id/{id}](http://localhost:8083/api/auth/id/%7bid%7d)

**Get by username**: http://localhost:8083/api/auth/{username}

**Unit test cases:**

**Run the test case for REPOSITORY,SERVICES (by mocking the repository)**

**Front end project images:**

