Movie Booking Application Documentation

1. Project Overview

The **Movie Booking Application** is a full-stack web application that allows users to register, log in, browse available movies, and book tickets with specific categories (Normal/Premium).

- **Frontend:** HTML5, CSS3, JavaScript (Vanilla).
- **Backend:** Java 17, Spring Boot 3.1.0.
- **Database:** MySQL.
- **ORM:** Spring Data JPA / Hibernate.

2. System Architecture

The application follows a **Client-Server Architecture**:

1. **Client (Frontend):** Browser-based UI that communicates with the backend via RESTful APIs using `fetch`.
2. **Server (Backend):** Spring Boot application handling business logic, authentication, and data persistence.
3. **Database:** Persistent storage for Users, Movies, and Booking records.

3. Backend Documentation (Spring Boot)

3.1 Project Structure

- `entity/`: Contains JPA entities that map to database tables.
- `userRepository/`: Interfaces for data access (extends `JpaRepository`).
- `service/`: Business logic layer (validation, calculations).
- `controller/`: REST controllers defining the API endpoints.

3.2 Data Models (Entities)

1. **User:** Stores `id`, `name`, `email` (Unique), and `password`.
2. **Movie:** Stores `id`, `movieName`, `showTime`, and `price`.
3. **Booking:** Stores `id`, `userId`, `movieId`, `seats`, and `totalAmount`.

3.3 API Endpoints

Method	Endpoint	Description
POST	`/api/register`	Registers a new user. Validates if email already exists.
POST	`/api/login`	Authenticates user. Returns user details on success.
GET	`/api/movies`	Fetches the list of all available movies.
POST	`/api/book-ticket`	Creates a booking. Calculates final price on server-side.

3.4 Key Logic (Exception Handling)

The `UserController` is designed to catch exceptions (like "User not found") and return a structured JSON response:

```
// Example from UserController.java
Map<String, String> response = new HashMap<>();
response.put("message", e.getMessage());
return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
```

4. Frontend Documentation (Web)

4.1 Project Files

- `indx.html` : The landing page with Login/Registration forms.
- `booking.html` : The seat selection and payment simulation page.
- `confirmation.html` : Final success page showing booking details.
- `script.js` : The heart of the frontend logic.

4.2 Core Logic (`script.js`)

1. **Session Management:** Uses `localStorage` to keep the user logged in across pages.
    - `localStorage.setItem('currentUser', JSON.stringify(user))`
2. **Dynamic Category Logic:**
    - Supports **Normal** (Base Price) and **Premium** (Base Price + ₹150 for Popcorn/Soda).
3. **Real-time Calculation:** An event listener on the `seats` input updates the `total` amount instantly.

4. **Fallback Mechanism:** If the backend database is empty, the script provides "Demo Movies" so the UI remains functional for testing.

5. User Journey (Workflow)

1. **Registration:** User signs up. Data is sent to `/api/register`.
2. **Login:** User authenticates. User object is saved to `localStorage`.
3. **Movie Selection:** User views the list of movies. Clicking "Book Now" saves the movie object to `localStorage` and redirects to `booking.html`.
4. **Customization:** User selects the number of seats and chooses between Normal or Premium categories.
5. **Payment/Booking:** Clicking "Pay" sends a booking request to `/api/book-ticket`.
6. **Confirmation:** On success, the user is redirected to a summary page showing a "Booking Confirmed" message.

6. Database Configuration (`application.properties`)

```
spring.datasource.url=jdbc:mysql://localhost:3306/movie_booking
spring.jpa.hibernate.ddl-auto=update
```

- **ddl-auto=update:** Ensures that the database structure is updated without deleting existing user data every time the server restarts.

7. How to Setup

1. **Database:** Create a schema named `movie_booking` in MySQL.
2. **Backend:** Open the project in IntelliJ/Eclipse and run `MovieBookingApplication.java`.
3. **Frontend:** Open `indx.html` in any modern web browser.
4. **Interaction:** Ensure the `BASE_URL` in `script.js` matches your Spring Boot server (Default: `http://localhost:8080/api`).