



MIT Art, Design and Technology University

MIT School of Computing, Pune

Department of Information Technology

LAB MANUAL

Practical - [Web Programming](#)

Class - S.Y. (SEM-IV), [DA](#)

Batch - DA-II

VIVEK KALYANI

[Mr./Ms.](#)

A.Y. 2024 – 2025 (SEM-IV)

Web Programming SEMESTER – IV			
Course Code:	23IT2008	Course Credits:	02
Teaching Hours / Week (L:T:P):	0:0:4	CA Marks:	25
Total Number of Teaching Hours:		END-SEM Marks:	25
Course Pre-requisites:			
Course Description: <p>This course provides a comprehensive introduction to web technology, designed to help students develop a strong foundation in building and managing websites and web applications. The curriculum covers key topics such as HTML, CSS, and JavaScript, PHP, MySQL, which are essential for creating interactive, well-designed web pages. Students will also explore the principles of responsive design, ensuring that web applications are optimized for different devices and screen sizes.</p> <p>The course dives deeper into server-side technologies, including HTTP, web servers, and databases, allowing students to understand how websites function behind the scenes. Emphasis is placed on practical learning, and students will gain hands-on experience by working on projects that showcase their ability to design, develop, and deploy websites.</p> <p>By the end of the course, students will be proficient in using modern web technologies to create web applications. They will understand how to handle client-server interactions, manage user data, and implement various web technologies to enhance the functionality of their applications.</p>			

Course Learning Objectives: This course will enable the students to:

1. Understand fundamental concepts of front-end web development.
2. Enable students to create basic web pages incorporating essential elements such as images, hyperlinks, lists, tables, and forms.
3. Teach students how to use CSS to manage fonts, lists, colors, text alignment, and background images for a cohesive and aesthetically pleasing web design.
4. Develop an understanding of JavaScript scopes to manage the visibility and lifetime of variables and functions effectively.
5. Equip students with the skills to implement and handle JavaScript events, enabling enhanced user interactions through event-driven programming.
6. Apply comprehensive knowledge of HTML, CSS, and JavaScript to develop a complete front-end application. Utilize project-based learning to showcase problem-solving skills and creativity in web development projects.
7. Configure server environments with Apache/TOMCAT.
8. Set up a PHP development environment and write basic PHP scripts.
9. Master PHP programming constructs for web development tasks.
10. Create and process HTML forms, and manage MySQL database operations.
11. Develop comprehensive back-end applications using PHP and MySQL.

Course Outcome: After taking this course, Students will be able to :

1. Apply knowledge of HTML to create the structure of the webpage and CSS to style and layout the elements, making the application visually appealing.
2. Apply comprehensive knowledge of HTML, CSS, and JavaScript to develop a complete front-end application and utilize project-based learning to showcase problem-solving skills and creativity in web development projects.
3. Set up and configure a server environment using tools like Apache or TOMCAT and set up a PHP development environment. Write & execute simple PHP scripts, understanding PHP syntax and basic features, create HTML forms to collect user data and integrate with PHP for processing.
4. Design and develop a back-end application using PHP and MySQL, implementing CRUD operations to manage data effectively.

UNIT – I	Introduction to HTML and Cascading Style Sheet	09 Hours
Module 1 - Markup Language (HTML): Introduction to HTML, Formatting and Fonts, Commenting Code, Anchors, Backgrounds, Images, Hyperlinks, Lists, Tables, Frames, HTML Forms Module 2 - CSS: Need for CSS, introduction to CSS, basic syntax and structure, Levels of style sheets, Style specification formats, BOX Model, Selector forms, Property value forms, Font properties, List properties, Color, Alignment of text, Background images		

Pedagogy	ICT Teaching / PowerPoint Presentation and Videos: Use tools like Visual Studio Code (free). Videos: https://www.coursera.org/learn/html-css-javascript-for-web-developers	
	Self-study / Do it yourself /: Practice creating basic HTML pages and enhancing them using CSS.	
	Experiential Learning Topics: Design a simple webpage for coffee shop website	
	PBL - Project Based Learning: Create a multi-page website (e.g., coffee shop website) using HTML and CSS.	
UNIT – II	Front-End Development	09 Hours
Module 3 - Overview of JavaScript, including JS in an HTML (Embedded, External), Basic JS syntax, basic interaction with HTML Module 4 - Core features of JavaScript: Data types, Control Structures, Arrays, Functions and Scopes		
Pedagogy	ICT Teaching / PowerPoint Presentation and Videos: Use tools like Visual Studio Code (free). Videos: https://www.coursera.org/learn/javascript-basics	
	Self-study / Do it yourself /: Solve exercises on JavaScript syntax, control structures, and functions	
	Experiential Learning Topics: Build a web page with interactive elements (e.g., a simple calculator).	
	PBL - Project Based Learning: Develop an interactive webpage that uses JavaScript to validate form inputs or perform basic calculations.	
UNIT – III	Advanced Front-End Development	09 Hours
Module 5 - DOM: DOM levels, DOM Objects and their properties and methods, Manipulating DOM Module 6 - JavaScript Events: JavaScript Events, Types of JavaScript Events, Objects in JS, Event Handling		

Pedagogy	ICT Teaching / PowerPoint Presentation and Videos: https://www.coursera.org/learn/building-interactive-web-pages-using-javascript Use tools like Visual Studio Code (free).	
	Self-study / Do it yourself /: Practice exercises on DOM traversal and event handling.	
	Experiential Learning Topics: Add dynamic behavior to a webpage using DOM and events (e.g., a to-do list app).	
	PBL - Project Based Learning: Develop a web page with dynamic content (e.g., a task manager or interactive quiz) using DOM manipulation and event handling.	
UNIT – IV	Server Side Scripting	09 Hours
Module 7 - Set up and configure a server environment using tools like Apache or TOMCAT, set up a PHP development environment. Module 8 -Introduction to PHP: : Introduction to PHP, Server side scripting Vs Client side scripting, Basic Development Concepts (Mixing PHP with HTML), Creating, Writing & Running First PHP Script, PHP syntax, conditions & Loops, Functions, String manipulation, Arrays & Functions, Module 9 - Form handling with HTML and PHP: Designing of Forms using HTML, Form Handling using GET and POST methods of Form		
Pedagogy	ICT Teaching / PowerPoint Presentation and Videos: https://www.coursera.org/learn/web-applications-php Use tools like Visual Studio Code (free), XAMPP/WAMP for PHP server setup, and MySQL Workbench for database management	
	Self-study / Do it yourself /: Practice exercises on form handling and server-side scripting with PHP.	
	Experiential Learning Topics: Create a basic form for data submission and handle it using PHP (e.g., feedback form).	
	PBL - Project Based Learning: Develop a small server-side application (e.g., a contact form with email validation and submission).	
UNIT – V	Working with Databases and Web Application Development	09 Hours

Module 10 - Working with databases using MySQL with PHP: MySQL database, create database, create table, primary key with AUTO_INCREMENT setting, Insert Data Into a Database Table, Select Data From a Database Table, Open or close a Connection to the MySQL Server.

Module 11 - Web Application Development (Project): Develop the web application to handle client-server interactions, manage user data, and implement various web technologies to enhance the functionality of their applications. Example: Website for a Coffee Shop

Pedagogy	ICT Teaching / PowerPoint Presentation and Videos: Use tools like Visual Studio Code (free), XAMPP/WAMP for PHP server setup, and MySQL Workbench for database management Videos: https://www.coursera.org/learn/web-app
	Self-study / Do it yourself /: Exercises on creating and manipulating databases using PHP and MySQL.
	Experiential Learning Topics: Create a database and design a webpage to display its data dynamically.
	PBL - Project Based Learning: Develop a fully functional web application (e.g., a Coffee Shop website or e-commerce platform) that integrates database functionality for data management.

Complete Project Theory: BookMyShow Web Application Clone

The BookMyShow Web Application Clone is a full-stack web development project that replicates the functionality of an online movie and event ticket booking platform. It combines front-end technologies such as HTML, CSS, and JavaScript with backend tools like PHP and MySQL to provide a seamless, dynamic, and interactive user experience. The objective is to emulate a real-world system where users can register, log in, browse movie listings, select seats, make bookings, and view their booking history—mirroring the core operations of a professional online ticketing system.

At the foundation of the project is semantic HTML5, which structures the web pages using meaningful tags like `<header>`, `<nav>`, `<main>`, and `<footer>`. This ensures that the content is accessible, SEO-friendly, and easy to maintain. The layout is divided logically to accommodate different components, including movie listings, user profiles, booking interfaces, and admin controls. CSS3 is employed extensively to enhance the visual aesthetics through consistent styling, responsive design with Flexbox and Grid, and well-defined color schemes and typography.

JavaScript is used on the client side to provide interactivity, especially for seat selection, dynamic content updates, form validations, and local storage management. For instance, users can click on available seats, and the system dynamically updates the total price. JavaScript also manages user login states and form inputs, improving the overall responsiveness and user experience.

The backend logic is developed in PHP, which handles form submissions, data validation, user authentication, and dynamic data rendering. The data is stored in a MySQL database, which contains tables for users, movies, bookings, and reviews. PHP scripts perform CRUD (Create, Read, Update, Delete) operations on the database, allowing data to persist across sessions and ensuring personalized content for logged-in users.

The admin panel is another key feature that enables content management for the platform. Administrators can log in through a separate portal to add, edit, or delete movie listings and monitor user activity. This ensures that the platform remains updated without requiring direct interaction with the database.

Additional advanced features include search and filter functionality, allowing users to find movies by title, genre, or rating using AJAX-powered PHP queries. The booking history module retrieves user-specific booking records using session-based authentication. The simulated payment gateway introduces the logic of secure transactions by validating dummy payment

details and confirming bookings. Though not connected to a real payment processor, it accurately mimics the flow of commercial checkout systems.

To improve user experience, responsive design is implemented through media queries, ensuring the platform adapts to different screen sizes such as desktops, tablets, and smartphones. Furthermore, a confirmation email feature is simulated using PHP's mail() function, reinforcing the real-world application aspect of the platform.

The review and rating system allows users to share feedback and rate movies on a 5-star scale, enhancing community engagement and social proof. JavaScript is used to handle the interactive star selection interface, while PHP stores and fetches the data for display.

Form validation is handled on both client and server sides to minimize errors, enhance security, and streamline user interaction. JavaScript performs real-time checks, while PHP ensures backend integrity.

Finally, the deployment process involves local hosting through XAMPP and optional online deployment via GitHub Pages, Netlify, Render, or Heroku. Configuration for routing, database connectivity, and environment management are addressed during deployment, rounding off the project with real-world applicability.

In conclusion, the BookMyShow Clone Web Application is a comprehensive full-stack development exercise that integrates multiple technologies and modules to build a functional, scalable, and user-centric application. It serves as a robust foundation for understanding professional web development workflows, from design and implementation to testing and deployment.

Experiment 1: Project Planning & Website Structure Design

Aim:

To create the foundational structure and visual planning document for the BookMyShow Clone website, including defining the homepage layout, header, content modules, and footer. The experiment also focuses on establishing the overall design strategy, audience definition, goals, and content planning for all modules.

Theory:

Building a successful full-stack web application begins with structured planning. This experiment focuses on conceptualizing and planning the BookMyShow Clone—an online movie ticket booking platform. The planning phase covers the project's intent, goals, modules, user interface structure, target users, and design system.

The purpose of the BookMyShow Clone is to simulate the features of a real-world ticket booking system. Users can register, log in, browse and filter movie listings, select available seats, book tickets, simulate payments, view past bookings, and provide ratings and reviews for movies. Administrators can manage movie listings and oversee user reviews.

Project Planning Includes the Following Steps:

- 1. Project Overview:**
BookMyShow Clone is designed to replicate the online ticket booking experience in a simplified and educational format. It includes all major modules found in production-grade platforms.
- 2. Goals & Deliverables:**
Deliverables include a responsive, mobile-friendly frontend; dynamic server-side pages; a secure backend; a functioning database; and features like authentication, booking history, and rating systems.
- 3. Module Breakdown:**
 - Homepage
 - Registration and Login (User/Admin)
 - Movie Listings with Filter/Search
 - Seat Booking UI
 - Payment Simulation
 - Booking History
 - Review and Rating Module
 - Admin Panel
- 4. Target Audience:**
This system is intended for regular users of online movie ticket platforms and cinema operators. It serves as a learning model for developers and a functional prototype for entertainment businesses.
- 5. Pain Points Addressed:**
Existing systems may be slow, confusing, or unresponsive. This project improves on them by ensuring a smoother, modern user experience, mobile accessibility, and intuitive navigation.

6. **Visual Design Direction:**

A dark theme with vibrant color highlights (e.g., red, gold) ensures visual consistency. Layouts follow a card/grid design with smooth transitions and hover effects. Fonts, icons, and spacing are carefully chosen for readability and appeal.

7. **Content Strategy:**

Every page is planned with clear sections:

- **Home:** Featured movies, login/register buttons
- **About:** Introduction and team mission
- **Movies:** Dynamically generated list
- **Cart/Booking:** Seat selection and summary
- **Contact:** Form-based communication
- **Admin Panel:** Add/remove movies
- **Reviews:** Ratings and comments

8. **Site Map/Structure:**

The project follows a modular structure where each component is connected logically. Navigation links enable smooth access to each section. Pages are designed to be scalable, enabling future updates like blog posts or FAQs.

9. **Design Elements:**

Consistent use of visual components like colors, fonts (Google Fonts), logo placements, movie posters, and UI icons ensures branding uniformity. Forms include tooltips and validation messages for better usability.

Code: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>BookMyShow Clone - Desktop</title>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    }
    body {
      background-image:
url('https://images.unsplash.com/photo-1598899134739-24c46f58fa9c?auto=format&fit=crop&w=1920&q=80');
      background-size: cover;
      background-position: center;
```

```
    color: #fff;
}
.container {
    max-width: 1200px;
    margin: auto;
    padding: 20px;
    background-color: rgba(0, 0, 0, 0.75);
    animation: fadeIn 1.5s ease-in-out;
}
@keyframes fadeIn {
    from { opacity: 0; transform: scale(0.95); }
    to { opacity: 1; transform: scale(1); }
}
header {
    text-align: center;
    padding: 30px 0;
}
header h1 {
    font-size: 3rem;
    color: #ff4757;
}
nav {
    margin-top: 20px;
    background-color: rgba(255, 255, 255, 0.1);
    padding: 15px 0;
}
nav ul {
    list-style: none;
    display: flex;
    justify-content: center;
    gap: 40px;
}
nav ul li a {
    text-decoration: none;
    color: #fff;
    font-size: 1.2rem;
    padding: 8px 16px;
    transition: background 0.3s ease, color 0.3s ease;
}
nav ul li a:hover {
```

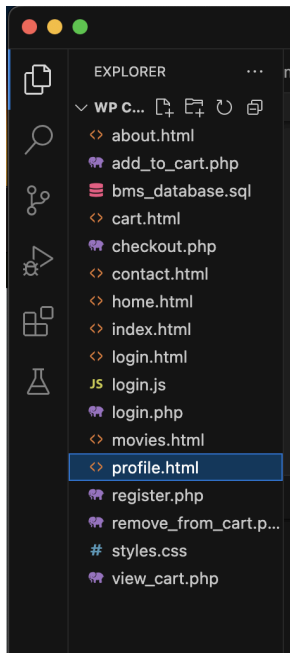
```

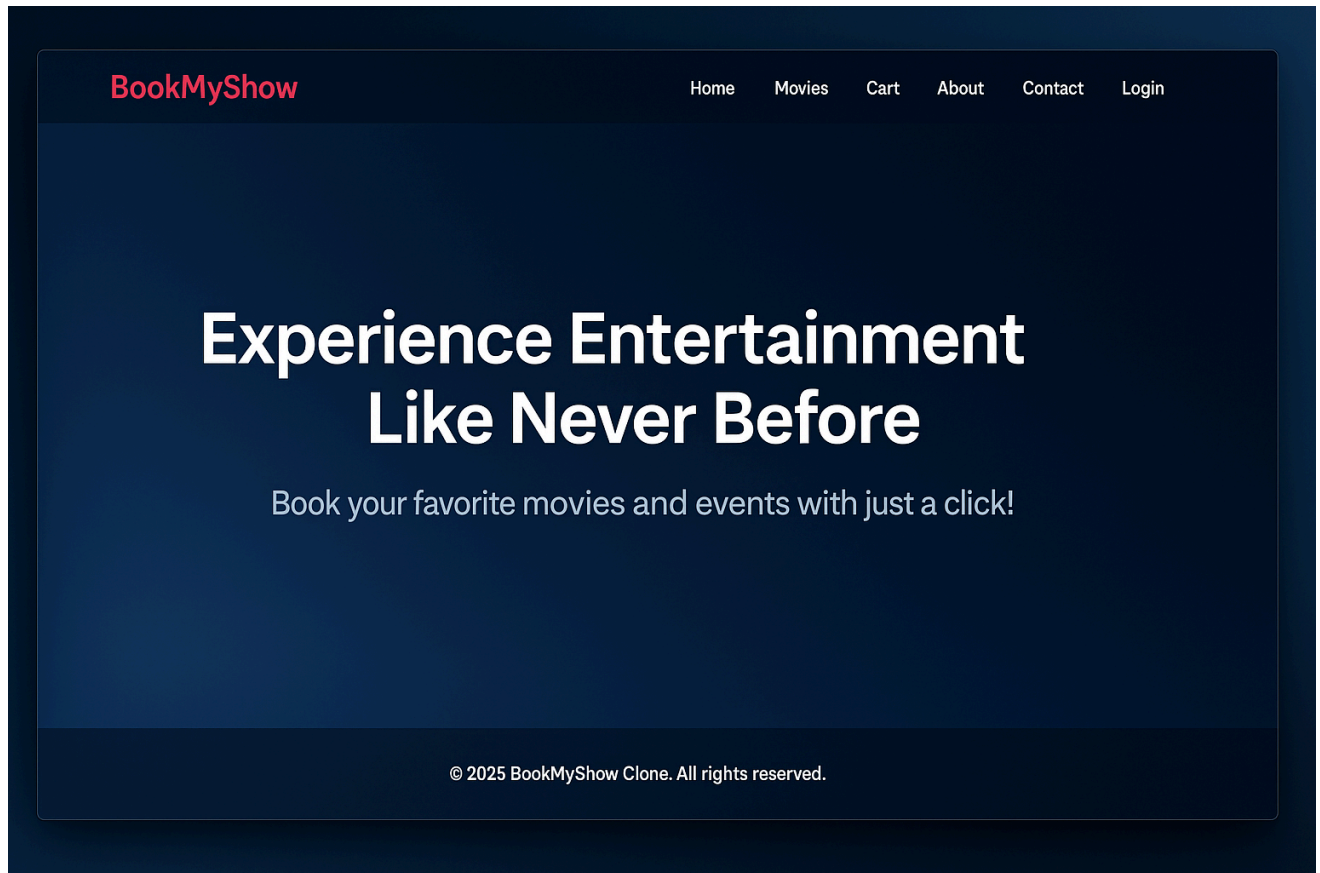
    background-color: #ff4757;
    border-radius: 5px;
    color: #fff;
  }
  section {
    text-align: center;
    margin: 80px 0;
  }
  section h2 {
    font-size: 2.5rem;
    color: #f1c40f;
    margin-bottom: 20px;
    animation: slideUp 1.2s ease-in-out;
  }
  section p {
    font-size: 1.3rem;
    line-height: 1.6;
  }
  @keyframes slideUp {
    from { transform: translateY(50px); opacity: 0; }
    to { transform: translateY(0); opacity: 1; }
  }
  footer {
    text-align: center;
    padding: 20px;
    background-color: rgba(255, 255, 255, 0.1);
    font-size: 1rem;
    margin-top: 100px;
  }
</style>
</head>
<body>
<div class="container">
  <header>
    <h1>BookMyShow</h1>
    <nav>
      <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#movies">Movies</a></li>
        <li><a href="#events">Events</a></li>

```

```
<li><a href="#login">Login</a></li>
</ul>
</nav>
</header>
<section>
  <h2>Welcome to BookMyShow Clone</h2>
  <p>Select and book your favorite movies and events with ease.<br/>
  Enjoy a seamless booking experience just like the real thing!</p>
</section>
<footer>
  <p>&copy; 2025 BookMyShow Clone. All rights reserved.</p>
</footer>
</div>
</body>
</html>
```

Output:





Conclusion:

This experiment lays the groundwork for the entire BookMyShow Clone project. By thoroughly defining the structure, design direction, audience, content, and goals, we ensure that the development process is efficient and goal-oriented. With a clearly mapped site layout and visual guide, future experiments (HTML, CSS, JavaScript, PHP, MySQL) can be implemented with minimal friction. This step is essential in professional web development to avoid design inconsistencies, usability issues, and development delays. Effective planning in Experiment 1 results in a coherent and user-centered web application that closely replicates a real-world cinema booking platform.

Experiment 2

Aim:

HTML

- A. Create a detailed home page for the BookMyShow website.
- B. Create a detailed movie listing page that displays available movies categorized appropriately.
- C. Create a cart page that allows customers to review and manage selected seats before checkout.
- D. Create an about us page that provides information about the platform's mission and team.
- E. Create a contact page that allows users to get in touch with the platform through a form.
- F. Design and implement admin/user registration form for the BookMyShow website.
- G. Design and implement admin/user login form for the BookMyShow website.

Theory:

HTML (HyperText Markup Language) is the foundation of web development. It serves as the **skeleton of all web pages**, defining and organizing content into a readable and structured format for browsers. In this experiment, HTML was used to build the **static layout** and **core structure** of a movie ticket booking platform—**BookMyShow Clone**—by creating all essential pages and components of the website. These include the **homepage**, **movie listing page**, **cart**, **about**, **contact**, **registration**, and **login** pages.

The experiment lays the groundwork for front-end design and backend integration by constructing each webpage with **semantic HTML elements**, preparing it for further enhancement using **CSS**, **JavaScript**, **PHP**, and **MySQL**.

1. Homepage Layout

The **homepage** acts as the **first impression** of the platform and is designed to be both welcoming and informative. It includes the following structural components:

- **Header** with branding/logo and a **navigation bar** linking to other pages such as Movies, Login, and Contact.
- **Main banner section** that features promotional visuals (e.g., upcoming movie releases or offers).
- **Content sections** for trending movies or featured events.
- **Footer** with copyright, social media icons, and quick links.

Semantic HTML tags used include:

`<header>`, `<nav>`, `<main>`, `<section>`, `<footer>`

These enhance **SEO**, **readability**, and **accessibility**, making the site usable for screen readers and search engines.

2. Movie Listing Page

The **movie listing page** is built using a **card-based layout** within a `<section>` or `<main>` tag. Each movie is presented inside a `<div>` representing a **movie card**, which includes:

- Movie **title**
- **Genre**
- **Ticket price**
- A **poster image**
- A “**Book Now**” button with a link to the booking/cart page

The content structure is created statically using HTML but later designed to support **dynamic population via PHP and MySQL**.

Here’s a simplified structure:

```
<div class="movie-card">

    <h3>Movie Title</h3>

    <p>Genre: Action</p>

    <p>Price: ₹200</p>

    <a href="cart.html" class="book-btn">Book Now</a>

</div>
```

3. Cart Page

The **cart page** simulates the **shopping cart experience**, similar to an e-commerce site. Users can:

- View selected movies/seats
- See the ticket breakdown and total amount
- Adjust quantities or remove items (simulated for now)
- Proceed to checkout (to be implemented in future backend integration)

This page sets up the **data structure and layout** needed for future dynamic handling using JavaScript and PHP.

4. About Page

The **about page** is designed to:

- Introduce the purpose of the project
- Describe the team or developer
- Share the mission and vision of the platform

HTML tags such as `<article>`, `<section>`, and `<p>` are used to organize the content into readable blocks.

This page enhances the **credibility** of the platform by providing transparency and background information.

5. Contact Page

The **contact page** contains a simple **HTML form** that enables users to reach out. The form includes:

- Name (`<input type="text">`)
- Email (`<input type="email">`)
- Message (`<textarea>`)
- Submit button (`<button>`)

Basic validation attributes such as `required` and `placeholder` are included to ensure completeness and improve user experience.

Here is a basic form structure:

```
<form action="#" method="post">

  <input type="text" name="name" placeholder="Your Name" required>

  <input type="email" name="email" placeholder="Your Email" required>

  <textarea name="message" placeholder="Your Message"
required></textarea>

  <button type="submit">Send</button>

</form>
```

This form is currently static but serves as the foundation for future server-side processing using PHP.

6. Registration and Login Pages

These pages form the core of **user authentication**. Each contains an HTML `<form>` that collects:

- **Registration form:** Full name, email, password, and password confirmation
- **Login form:** Email and password

Each form is designed with semantic HTML tags and structured into fieldsets or sections for better clarity. Attributes like `required`, `type="email"`, and `type="password"` enforce basic client-side validation.

Example registration form snippet:

```
<form action="register.php" method="post">

  <input type="text" name="fullname" placeholder="Full Name" required>

  <input type="email" name="email" placeholder="Email" required>

  <input type="password" name="password" placeholder="Password"
required>

  <input type="password" name="confirm_password" placeholder="Confirm
Password" required>

  <button type="submit">Register</button>

</form>
```

These pages are styled consistently and are ready to be integrated with JavaScript for validations and PHP for backend processing.

7. Semantic and Structural Best Practices

All HTML pages in the project:

- Use semantic tags like `<header>`, `<nav>`, `<main>`, `<section>`, and `<footer>` for better readability, maintainability, and SEO.
- Follow a **consistent layout** across the website.
- Are linked to a central `style.css` file using `<link rel="stylesheet" href="style.css">` for consistent styling.
- Include `<meta>` tags and proper `<title>` and `<lang>` attributes in the `<head>` section for accessibility and search engine compatibility.

8. Preparation for Future Enhancements

Each page is carefully structured to support future integration of:

- **CSS** for styling and layout refinement
- **JavaScript** for form validation and UI interactivity
- **PHP & MySQL** for backend processing and dynamic data rendering

This modular and scalable design approach ensures that the project can evolve from a static prototype into a fully functional dynamic web application.

Code: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>BookMyShow Clone - Desktop</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
  <div class="container">
    <header>
      <h1>BookMyShow</h1>
      <nav>
        <ul>
          <li><a href="movies.html">Now Showing</a></li>
          <li><a href="login.html">Login</a></li>
          <li><a href="register.html">Register</a></li>
          <li><a href="cart.html">Cart</a></li>
        </ul>
      </nav>
    </header>

    <section>
      <h2>Welcome to BookMyShow Clone</h2>
      <p>
        Select and book your favorite movies and events with ease.<br />
        Enjoy a seamless booking experience just like the real thing!
      </p>
    </section>
  </div>
</body>
</html>
```

```
<footer>
  <p>&copy; 2025 BookMyShow Clone. All rights reserved.</p>
</footer>
</div>
</body>
</html>
```

Movies.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Now Showing</title>
  <link rel="stylesheet" href="style.css">
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      background-color: #f4f4f4;
    }
    header {
      background-color: crimson;
      color: white;
      padding: 20px;
      text-align: center;
    }
    nav ul {
      list-style: none;
      padding: 0;
      margin: 10px 0 0;
      display: flex;
      justify-content: center;
      gap: 20px;
    }
    nav ul li a {
      color: white;
      text-decoration: none;
      font-weight: bold;
```

```
}  
.movie-grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));  
  gap: 20px;  
  padding: 20px;  
}  
.movie-card {  
  background-color: #fff;  
  border-radius: 10px;  
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);  
  overflow: hidden;  
  transition: transform 0.3s ease;  
  text-align: center;  
}  
.movie-card:hover {  
  transform: scale(1.05);  
}  
.movie-card img {  
  width: 100%;  
  height: 350px;  
  object-fit: cover;  
}  
.movie-card h3 {  
  margin: 10px 0;  
}  
.movie-card p {  
  margin: 0 10px 10px;  
  font-size: 14px;  
  color: #555;  
}  
.movie-card a button {  
  margin: 10px;  
  padding: 10px 15px;  
  background-color: crimson;  
  color: white;  
  border: none;  
  border-radius: 5px;  
  cursor: pointer;  
}
```

```

    .movie-card a button:hover {
        background-color: darkred;
    }
</style>
</head>
<body>
    <header>
        <h1>BookMyShow - Movies</h1>
        <nav>
            <ul>
                <li><a href="index.html">Home</a></li>
                <li><a href="cart.html">Cart</a></li>
                <li><a href="login.html">Login</a></li>
            </ul>
        </nav>
    </header>
    <main class="movie-grid">
        <div class="movie-card">
            
            <h3>Interstellar</h3>
            <p>Adventure, Drama, Sci-Fi</p>
            <a href="booking.html?movie=Interstellar"><button>Book Now</button></a>
        </div>
        <div class="movie-card">
            
            <h3>Joker</h3>
            <p>Crime, Drama, Thriller</p>
            <a href="booking.html?movie=Joker"><button>Book Now</button></a>
        </div>
        <div class="movie-card">
            
            <h3>Oppenheimer</h3>
            <p>Biography, Drama, History</p>
            <a href="booking.html?movie=Oppenheimer"><button>Book Now</button></a>
        </div>
        <div class="movie-card">
            
            <h3>Avatar: The Way of Water</h3>
            <p>Action, Adventure, Fantasy</p>
            <a href="booking.html?movie=Avatar+2"><button>Book Now</button></a>
        </div>
    </main>
</body>
</html>

```

```

</div>
<div class="movie-card">
  
  <h3>KGF Chapter 2</h3>
  <p>Action, Drama, Thriller</p>
  <a href="booking.html?movie=KGF+2"><button>Book Now</button></a>
</div>
<div class="movie-card">
  
  <h3>Avengers: Endgame</h3>
  <p>Action, Sci-Fi, Superhero</p>
  <a href="booking.html?movie_id=<?= $movie['id'] ?>">Book Now</a>
</div>
<?php include 'php/fetch-movies.php'; ?>
</main>
</body>
</html>

```

Cart.html

```

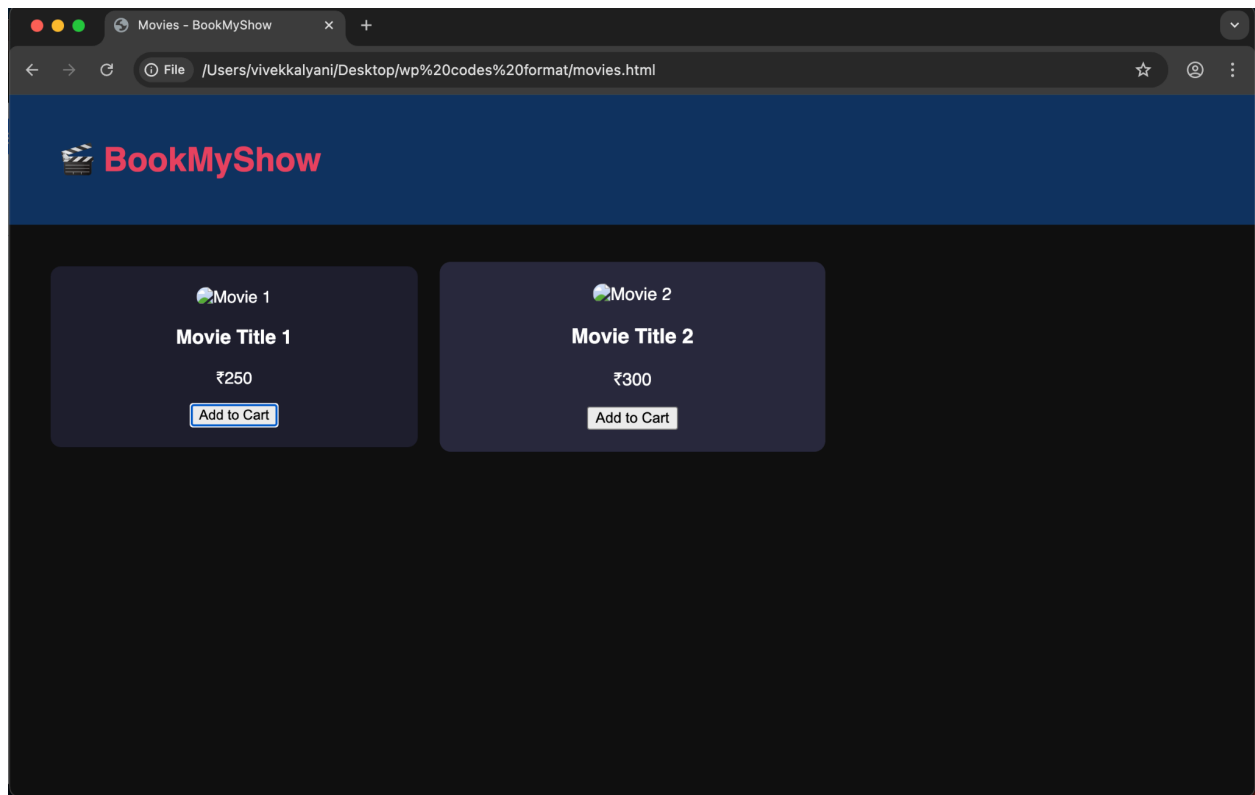
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Cart</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h2>Your Seat Selection</h2>
  <div id="screen">SCREEN</div>
  <div id="seats"></div>
  <p>Seats Selected: <span id="count">0</span></p>
  <p>Total Price: ₹<span id="total">0</span></p>
  <button onclick="checkout()">Proceed to Payment</button>
  <script src="cart.js"></script>
</body>
</html>

```

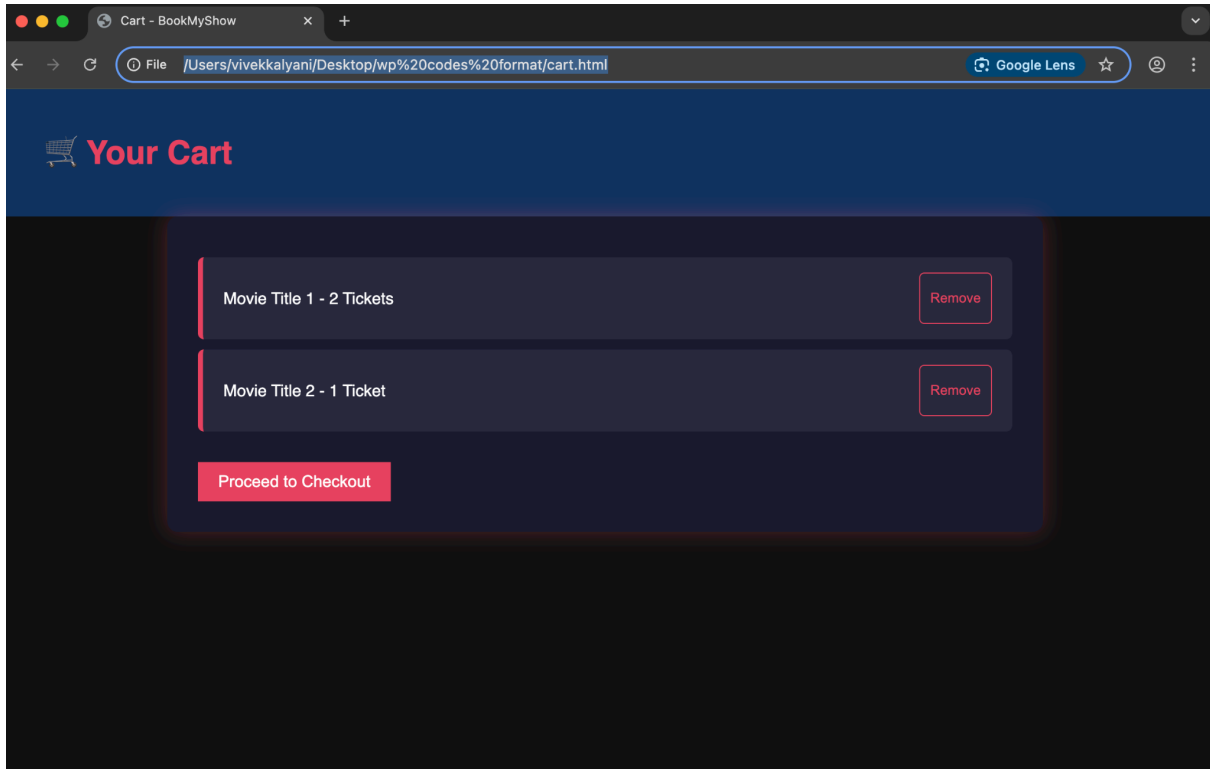
About.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <title>About Us</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <header>
    <h1>About BookMyShow Clone</h1>
  </header>
  <main>
    <section>
      <p>This project is a clone of the BookMyShow platform made for educational purposes. It
allows users to register, browse movies, select seats, and simulate payments.</p>
      <p>Created by: Vivek Kalyani</p>
    </section>
  </main>
  <footer>
    <p>© 2025 BookMyShow Clone</p>
  </footer>
</body>
</html>
```

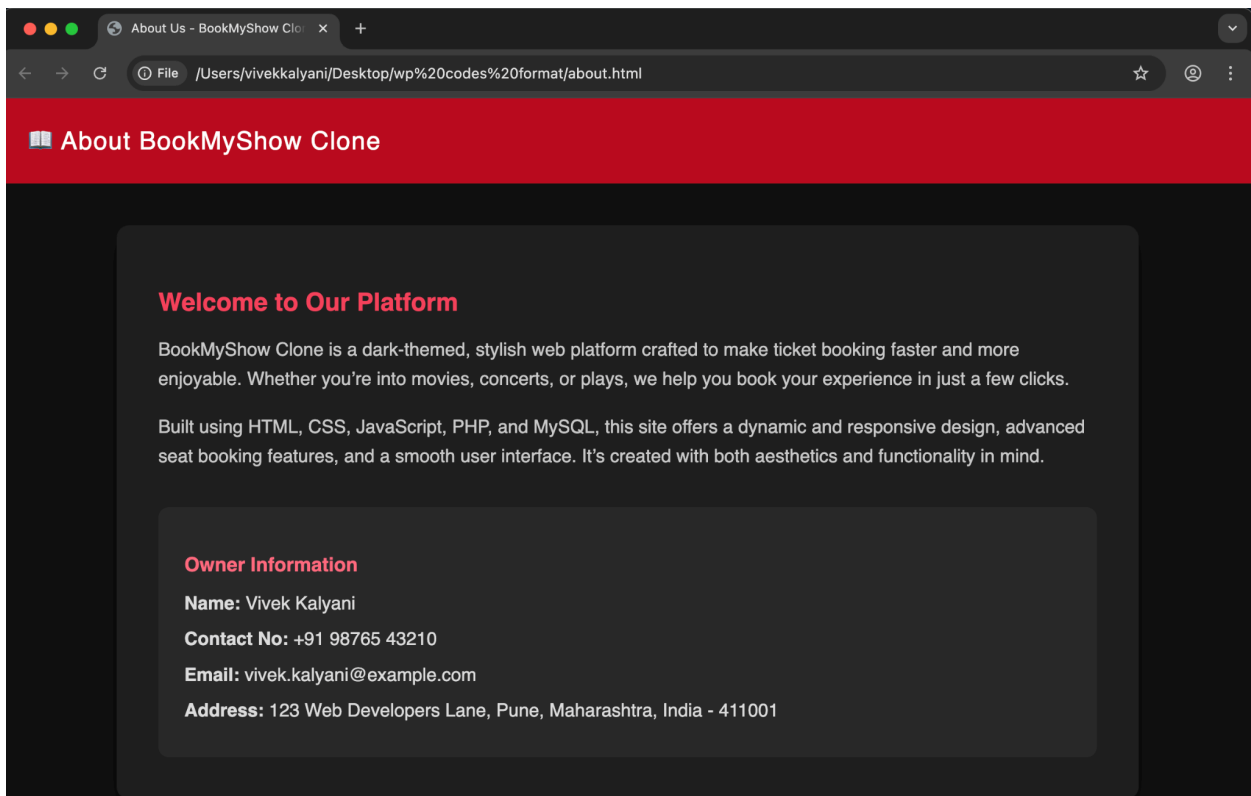
Output:**Movies page**



Cart page



About page



Conclusion:

This experiment successfully lays down the structural foundation of the BookMyShow web application using HTML. By creating the core static pages—home, movie listings, cart, about, contact, registration, and login—the user interface becomes navigable, informative, and ready for styling and dynamic integration.

Each page plays a critical role in the application's flow. The homepage serves as the brand gateway and connects users to other sections. The movie listing page functions as the central hub for browsing available shows and initiating bookings. The cart page ensures that users can review their selections before payment, mimicking real-world checkout processes.

The about and contact pages add professionalism and credibility to the platform. They demonstrate transparency and provide communication channels, enhancing user trust. The user registration and login forms are the starting point of user engagement, essential for personalized services like saving booking history and managing profiles.

The consistent use of semantic HTML tags improves accessibility, SEO, and maintainability. Tags such as `<section>`, `<article>`, `<input type="email">`, and `<form>` ensure compatibility with screen readers and assistive technologies. Forms were built with clean layouts and appropriate input types to facilitate future JavaScript validations and PHP processing.

Additionally, the layout structure was designed with responsiveness in mind, ready to be styled using CSS Grid and Flexbox. Navigation links were aligned with the planned user flow, ensuring a smooth experience for both authenticated and guest users.

In conclusion, this experiment forms the backbone of the BookMyShow project, transforming the initial blueprint into tangible HTML pages. It sets the stage for the next experiments where interactivity (via CSS and JavaScript) and dynamic content handling (via PHP and MySQL) will bring life to the application.

Experiment 3

Aim:

CSS

- A. Enhance the layout of the BookMyShow website using **CSS Grid** for the home page.
- B. Use **CSS Grid** to layout the movie listing page in a structured format and style the movie categories with appropriate headings, spacing, separators, images, descriptions, and ticket prices.

Theory:

Cascading Style Sheets (CSS) is the cornerstone of web design, responsible for defining the look, structure, and responsiveness of a web page. While traditional layout techniques like floats and inline blocks were used in early web design, **modern CSS layout systems** like **CSS Grid** offer a more intuitive and powerful approach for crafting sophisticated two-dimensional layouts.

In this experiment, CSS Grid is leveraged to **structure and style the homepage and movie listing page** of the **BookMyShow Clone** project, resulting in a clean, organized, and responsive design. CSS Grid helps manage both rows and columns simultaneously, enabling complex layouts to be built with minimal code and maximum flexibility.

1. Overview of CSS Grid

CSS Grid is a **two-dimensional layout system** introduced in CSS3. Unlike Flexbox, which is primarily one-dimensional (row *or* column), Grid allows developers to create layouts in **both directions** simultaneously, making it ideal for web pages with multiple content blocks.

Key CSS Grid properties used include:

- **display: grid;** – Activates grid layout on a container.
- **grid-template-columns** and **grid-template-rows** – Define the number and size of columns and rows.
- **grid-template-areas** – Assign named layout areas for more readable and semantic CSS.
- **gap** or **grid-gap** – Controls spacing between grid items.
- **justify-content** and **align-content** – Manage content alignment inside the grid container.

2. Homepage Layout Using CSS Grid

The homepage of the BookMyShow Clone is divided into the following structured sections:

- **Header:** Contains the logo, navigation bar, and login/register buttons.
- **Main Banner:** Highlights promotional offers or new releases with large visuals.
- **Trending Movies/Promotional Sections:** Showcases new or popular titles in horizontal grids.

- **Footer:** Includes links to terms, social media, and contact info.

Using CSS Grid, the layout was created with clarity and hierarchy. For example:

```
.grid-container {  
  display: grid;  
  grid-template-areas:  
    "header"  
    "banner"  
    "trending"  
    "footer";  
  grid-gap: 20px;  
}
```

Each child section was given an area name:

```
.header {  
  grid-area: header;  
}  
  
.banner {  
  grid-area: banner;  
}  
  
.trending {  
  grid-area: trending;  
}  
  
.footer {
```

```

    grid-area: footer;

}

```

This modular layout allows the page to be **easily modified or rearranged** while maintaining responsiveness and consistency.

3. Movie Listing Page with Grid-Based Cards

The movie listing page uses a **card-based grid layout** to present movie information attractively. Each card includes:

- A poster image
- Title
- Genre
- Price
- A "Book Now" button

To maintain uniformity and responsiveness, CSS Grid was used as follows:

```

.movie-grid {

    display: grid;

    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));

    gap: 20px;

    padding: 20px;

}

```

Here:

- `repeat(auto-fit, minmax(250px, 1fr))` automatically fits as many columns as possible within the container, with a **minimum width of 250px per card**.
- This makes the layout **responsive**, as it adapts to the available screen width without needing explicit media queries.

Each card is styled with:

- **Box shadows** and **rounded corners** for depth
- **Hover effects** to create interactivity and highlight clickable areas
- **Transitions** to animate changes smoothly

```
.movie-card:hover {  
  
    transform: scale(1.05);  
  
    box-shadow: 0 8px 16px rgba(0,0,0,0.2);  
  
}
```

4. Responsive Design with Media Queries

Although CSS Grid is inherently responsive, media queries provide **extra control** for extreme screen sizes. For instance, you might adjust font sizes or padding on mobile:

```
@media (max-width: 768px) {  
  
    .banner-text {  
  
        font-size: 1.5rem;  
  
        text-align: center;  
  
    }  
  
    .movie-grid {  
  
        grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));  
  
    }  
  
}
```

This ensures optimal display across:

- Mobile phones
- Tablets
- Laptops
- Desktop monitors

5. Global Styling and Reusability

All CSS styles were defined in a **central `style.css` file**, which was linked across all pages using:

```
<link rel="stylesheet" href="style.css">
```

This ensures:

- **Uniform typography, colors, and button styles** across the entire project
- **Easy maintenance and scalability**, as changes in `style.css` reflect on all pages

Google Fonts were used for professional typography:

```
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&dis
play=swap');
```

Custom color palette:

- **Dark background** (e.g., `#121212`) for a cinema-themed ambiance
- **Accent colors** like **red** (`#E50914`) for action buttons and **yellow** (`#FFD700`) for ratings or highlights

6. Advantages of Using CSS Grid in This Project

- **Visual Consistency:** Cards, banners, and forms are neatly aligned and spaced.
- **Responsiveness:** Works smoothly across screen sizes without requiring extensive restructuring.
- **Maintainability:** Clear structure allows developers to make layout changes without affecting content.
- **Performance:** Grid layout uses fewer HTML elements and less JavaScript, improving load speed and performance.
- **Modern UX:** Layouts feel clean, modern, and professional—aligned with user expectations from a movie booking platform.

Code: Style.css

```
/* Reset & base */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
/* Desktop background */
body {
```



```

    background-image:
url('https://images.unsplash.com/photo-1598899134739-24c46f58fa9c?auto=format&fit=crop&
w=1920&q=80');
    background-size: cover;
    background-position: center;
    background-attachment: fixed;
    color: #fff;
    animation: fadeInBody 1.5s ease;
    overflow-x: hidden;
}
@keyframes fadeInBody {
    from { opacity: 0; }
    to { opacity: 1; }
}
/* Main container - fixed desktop width */
.container {
    max-width: 1200px;
    margin: 40px auto;
    padding: 40px;
    background-color: rgba(0, 0, 0, 0.85);
    border-radius: 15px;
    animation: fadeIn 1.5s ease-in-out;
}
@keyframes fadeIn {
    from { opacity: 0; transform: scale(0.95); }
    to { opacity: 1; transform: scale(1); }
}
/* Header styles */
header {
    text-align: center;
    padding-bottom: 20px;
}
header h1 {
    font-size: 3.5rem;
    color: #ff4757;
    animation: slideDown 1s ease;
}
@keyframes slideDown {
    from { transform: translateY(-30px); opacity: 0; }
    to { transform: translateY(0); opacity: 1; }
}

```

```
}
/* Navigation styles */
nav {
  margin-top: 20px;
  background-color: rgba(255, 255, 255, 0.1);
  padding: 20px 0;
  border-radius: 8px;
}
nav ul {
  list-style: none;
  display: flex;
  justify-content: center;
  gap: 50px;
}
nav ul li a {
  text-decoration: none;
  color: #fff;
  font-size: 1.3rem;
  font-weight: 500;
  padding: 10px 20px;
  border-radius: 6px;
  transition: background-color 0.3s ease, color 0.3s ease;
}
nav ul li a:hover {
  background-color: #ff4757;
  color: #fff;
}
/* Section styles */
section {
  text-align: center;
  margin: 100px 0;
  animation: slideUp 1.2s ease-in-out;
}
section h2 {
  font-size: 2.8rem;
  color: #f1c40f;
  margin-bottom: 30px;
}
section p {
  font-size: 1.5rem;
```

```

    line-height: 1.8;
    max-width: 800px;
    margin: auto;
    color: #f5f5f5;
}
@keyframes slideUp {
  from { transform: translateY(50px); opacity: 0; }
  to { transform: translateY(0); opacity: 1; }
}
/* Footer */
footer {
  text-align: center;
  padding: 25px;
  background-color: rgba(255, 255, 255, 0.1);
  font-size: 1.1rem;
  margin-top: 100px;
  border-radius: 0 0 10px 10px;
}
/* Seat Layout */
#screen {
  margin: 30px auto;
  width: 120px;
  height: 20px;
  background: #000;
  color: #fff;
  text-align: center;
  line-height: 20px;
  border-radius: 5px;
  font-weight: bold;
  box-shadow: 0 0 5px #fff;
}
.seat-layout {
  display: flex;
  flex-wrap: wrap;
  max-width: 250px;
  margin: 20px auto;
  justify-content: center;
}
.seat {
  background-color: #ccc;

```

```
height: 25px;
width: 25px;
margin: 5px;
border-radius: 5px;
cursor: pointer;
transition: transform 0.2s ease;
}
.seat:hover {
  transform: scale(1.1);
}
.seat.selected {
  background-color: #6feaf6;
}
.seat.occupied {
  background-color: #444;
  cursor: not-allowed;
}
/* Base styles */
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #121212;
  color: #eee;
}
/* Header and Footer */
header, footer {
  background-color: #20232a;
  color: white;
  text-align: center;
  padding: 1em;
}
/* Navigation */
nav ul {
  list-style: none;
  display: flex;
  justify-content: center;
  gap: 20px;
  padding: 0;
  margin: 0;
```

```
}
nav ul li a {
  color: white;
  text-decoration: none;
  font-weight: 600;
  font-size: 1.1rem;
  padding: 8px 12px;
  border-radius: 6px;
  transition: background-color 0.3s ease;
}
nav ul li a:hover,
nav ul li a:focus {
  background-color: #ff4757;
  outline: none;
}
/* Container for page content */
.container {
  max-width: 1200px;
  margin: auto;
  padding: 20px;
}
/* Responsive Form Styling */
form {
  max-width: 400px;
  margin: 2em auto;
  padding: 1.5em;
  background: #222;
  border-radius: 12px;
  box-shadow: 0 0 20px rgba(255, 71, 87, 0.7);
  color: #eee;
}
input, button {
  width: 100%;
  padding: 12px;
  margin-top: 12px;
  border-radius: 8px;
  border: 1px solid #444;
  font-size: 1rem;
  background-color: #333;
  color: #eee;
```

```
    transition: box-shadow 0.3s ease, background-color 0.3s ease;
}
input:focus {
    outline: none;
    box-shadow: 0 0 12px #ff4757;
    background-color: #444;
}
button {
    border: none;
    background-color: #ff4757;
    font-weight: 700;
    cursor: pointer;
    color: white;
    transition: background-color 0.3s ease;
}
button:hover,
button:focus {
    background-color: #e03e4f;
    outline: none;
}
/* Table styling */
table {
    width: 100%;
    border-collapse: collapse;
    margin: 2em 0;
    background-color: #222;
    color: #eee;
    border-radius: 8px;
    overflow: hidden;
}
th, td {
    padding: 14px 18px;
    text-align: center;
    border-bottom: 1px solid #444;
}
th {
    background-color: #ff4757;
    font-weight: 700;
}
tr:nth-child(even) {
```

```

    background-color: #2c2c2c;
}
/* Responsive styles */
@media (max-width: 768px) {
    nav ul {
        flex-direction: column;
        gap: 12px;
    }
    form {
        width: 90%;
        padding: 1em;
    }
    table {
        font-size: 0.9rem;
    }
}
@media (max-width: 480px) {
    body {
        font-size: 14px;
    }
    nav ul li a {
        font-size: 16px;
        padding: 10px;
    }
    form {
        padding: 1em;
        width: 95%;
    }
}

```

Form-style.css

```

/* Base Reset */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}
body {

```

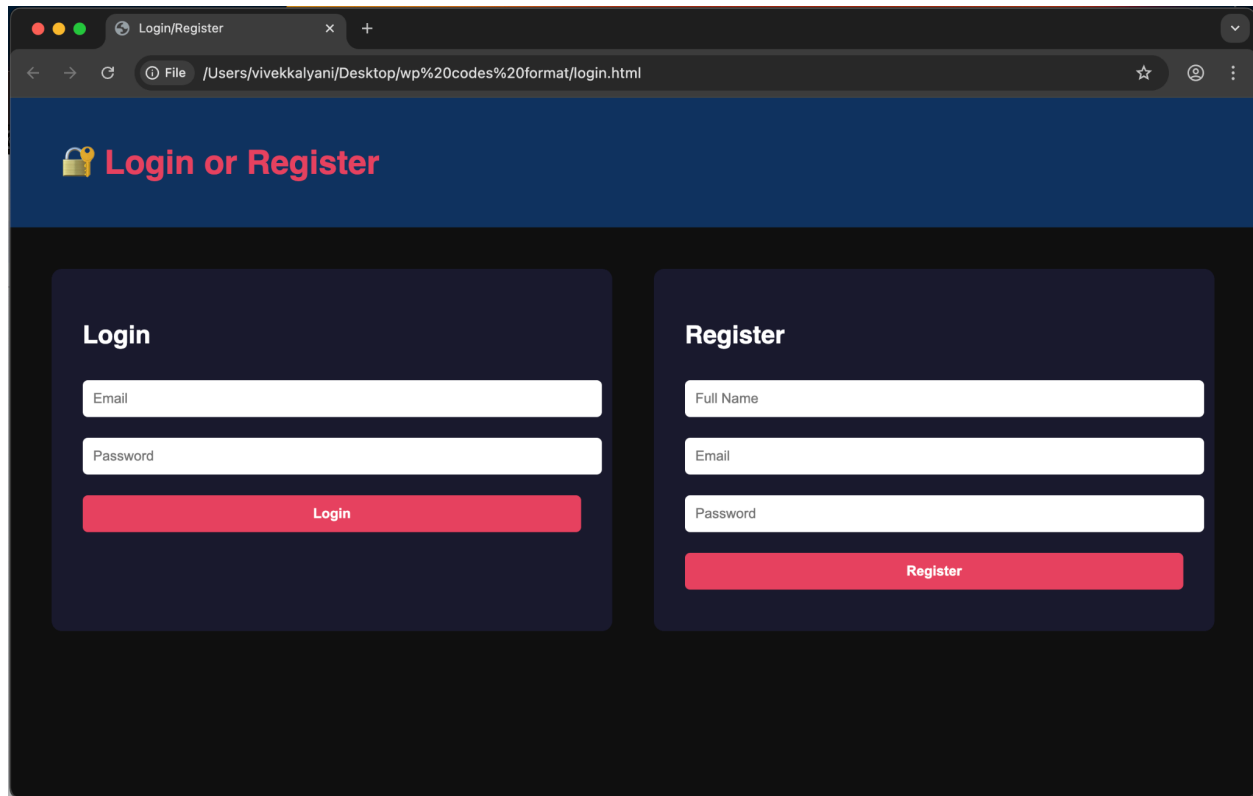
```
background-image:
url('https://images.unsplash.com/photo-1524985069026-dd778a71c7b4?auto=format&fit=crop&
w=1920&q=80');
background-size: cover;
background-position: center;
background-attachment: fixed;
color: #fff;
animation: fadeInBody 1.2s ease;
}
@keyframes fadeInBody {
  from { opacity: 0; }
  to { opacity: 1; }
}
.container {
  max-width: 800px;
  margin: 50px auto;
  padding: 40px;
  background-color: rgba(0, 0, 0, 0.85);
  border-radius: 15px;
  animation: fadeIn 1.2s ease-in-out;
}
@keyframes fadeIn {
  from { opacity: 0; transform: translateY(20px); }
  to { opacity: 1; transform: translateY(0); }
}
header {
  text-align: center;
}
header h1 {
  font-size: 3rem;
  color: #ff4757;
  margin-bottom: 10px;
}
nav ul {
  list-style: none;
  display: flex;
  justify-content: center;
  gap: 30px;
  margin-top: 10px;
}
```



```
nav ul li a {
  color: white;
  text-decoration: none;
  font-size: 1.1rem;
  padding: 6px 12px;
  border-radius: 5px;
  transition: background 0.3s ease;
}
nav ul li a:hover {
  background-color: #ff4757;
}
section {
  margin-top: 40px;
  text-align: center;
}
section h2 {
  font-size: 2.2rem;
  margin-bottom: 30px;
  color: #f1c40f;
}
form {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 20px;
}
input[type="text"],
input[type="email"],
input[type="password"] {
  width: 300px;
  padding: 12px;
  border-radius: 6px;
  border: none;
  font-size: 1rem;
}
input:focus {
  outline: none;
  box-shadow: 0 0 8px #ff4757;
}
button {
```

```
width: 320px;
padding: 12px;
border: none;
border-radius: 6px;
background-color: #ff4757;
color: white;
font-size: 1.1rem;
font-weight: bold;
cursor: pointer;
transition: background-color 0.3s ease;
}
button:hover {
  background-color: #e63a45;
}
footer {
  margin-top: 60px;
  text-align: center;
  background-color: rgba(255, 255, 255, 0.1);
  padding: 15px;
  font-size: 0.95rem;
  border-radius: 0 0 10px 10px;
}
```

Output: form-style.css



Conclusion:

The successful application of **CSS Grid** in this experiment significantly improved the structure, responsiveness, and visual appeal of the homepage and movie listing page in the BookMyShow Clone project.

On the homepage, the grid layout helped in aligning content blocks such as the navigation bar, promotional banners, movie categories, and the footer. This modular design ensures that each component fits into its place regardless of the screen size. It also makes future edits to layout easier, as CSS Grid is highly maintainable and scalable.

The movie listing page, which is central to the functionality of the web app, benefited immensely from Grid. Movies were presented in uniform cards, making it easier for users to browse, compare, and select. On smaller screens, the cards stack vertically while maintaining spacing and readability, ensuring a consistent user experience across devices. This responsiveness is vital as users today access booking sites from a wide range of devices, from phones to large monitors.

This experiment also demonstrated how CSS Grid can coexist with Flexbox and traditional layouts when used appropriately. For example, Flexbox was used in the navigation bar for horizontal alignment, while Grid handled the main content sections.

The consistent use of a stylesheet (`style.css`) also promoted clean code and centralized control over the website's design. Using variables and reusable classes, the style file remains organized and easy to update.

In conclusion, CSS Grid proved to be a powerful tool for creating structured, modern, and responsive layouts. This experiment enhanced the user interface of the BookMyShow Clone while keeping the layout flexible for future enhancements like filters, pagination, and dynamic data insertion. The visual impact created through this design elevates the overall professionalism of the project and aligns closely with real-world commercial booking platforms.

Experiment 4: Enhancing User Interface Using CSS - Styling Booking Cart and User Pages

Aim:

The aim of this experiment is to enhance the visual appeal and usability of the BookMyShow Clone website by applying advanced CSS styling techniques. The focus will be on styling the booking cart page, user profile pages, and the registration/login forms to provide a seamless and user-friendly experience.

Theory:

Cascading Style Sheets (CSS) is a fundamental technology in web development used to control the presentation and layout of HTML elements. It transforms raw HTML content into a visually appealing, user-friendly interface, significantly impacting user experience (UX).

For a booking platform like the BookMyShow Clone, a well-designed UI ensures users can easily navigate through movie selections, manage their cart, register, and complete transactions without confusion or frustration.

1. Importance of CSS in Web UI Design

While HTML provides the structure and content of a webpage, CSS defines how it looks — including colors, fonts, spacing, and positioning. The effectiveness of CSS styling directly affects:

- Usability: How intuitive and easy the site is to use.
- Accessibility: How well users with disabilities can interact with the site.
- Branding: Consistent use of colors and styles reflects the project's identity.
- Engagement: Attractive UI encourages users to stay longer and complete actions like bookings.

2. Styling the Booking Cart Page

The cart page is a crucial touchpoint where users review their selected tickets and proceed to payment. The page should provide:

- Clear Item Listings: Display movie names, showtimes, seat numbers, and prices in a tabular or card format.
- Quantity Controls: Buttons or input fields to increase/decrease ticket quantities dynamically.
- Price Calculations: Prominently show subtotal, taxes, and total price.
- Call-to-Action Buttons: Bold and clearly visible “Proceed to Checkout” or “Remove Item” buttons.

CSS Techniques for Cart Styling:

- Box Model (Margins, Padding, Borders): Proper spacing around elements improves readability and prevents clutter.
- Typography: Use readable fonts, font sizes, and weights to differentiate headings, labels, and details.

- Button Styles: Use colors, rounded corners, shadows, and hover effects to indicate interactivity and responsiveness.
- Layout Models: CSS Grid or Flexbox arrange the items in neat rows and columns, adapting gracefully on different devices.

3. Styling User Interaction Pages (Registration, Login, Profile)

Forms on these pages require:

- Clear Input Fields: Adequate size, padding, and borders so users can easily identify where to input information.
- Labels and Placeholders: Properly aligned labels and helpful placeholders guide users on expected data.
- Validation Messages: Use CSS to style error messages (usually red text) and success confirmations (green text) that appear dynamically based on user input.
- Focus States: Highlight input fields with subtle shadows or border color changes when active, enhancing accessibility.

Advanced CSS Effects:

- Transitions: Smooth changes for hover effects, focus highlights, or button presses improve the tactile feel.
- Box Shadows and Rounded Corners: Provide a modern, soft look that reduces visual harshness.
- Consistent Color Schemes: Color psychology helps build trust; green can signal success, red signals errors, blue conveys calmness and reliability.

4. Responsive Web Design

Users access websites on a range of devices — phones, tablets, laptops, desktops — with varying screen sizes and resolutions. Responsive design ensures the interface adapts fluidly without losing functionality or aesthetics.

CSS Tools for Responsiveness:

- Media Queries: Conditional CSS rules that apply styles based on screen width, orientation, resolution, etc.
- Fluid Grids and Flexible Images: Use relative units (% , em, rem) instead of fixed pixels to allow elements to resize proportionally.
- Breakpoints: Define multiple viewport widths where the layout changes (e.g., stacking columns vertically on small screens).

5. Consistency and Branding

A well-styled website uses a coherent style guide for:

- Colors
- Fonts
- Button styles
- Margins and padding
- Icons and images

Consistency reduces cognitive load on users and gives the BookMyShow Clone a professional and trustworthy appearance.

6. Integration with Dynamic Features

Good CSS styling is also important when combined with:

- JavaScript Validations: CSS highlights input errors immediately.
- Backend Integration: CSS ensures forms and dynamic content load smoothly and maintain usability.

7. Accessibility Considerations

Enhancing UI with CSS should not sacrifice accessibility:

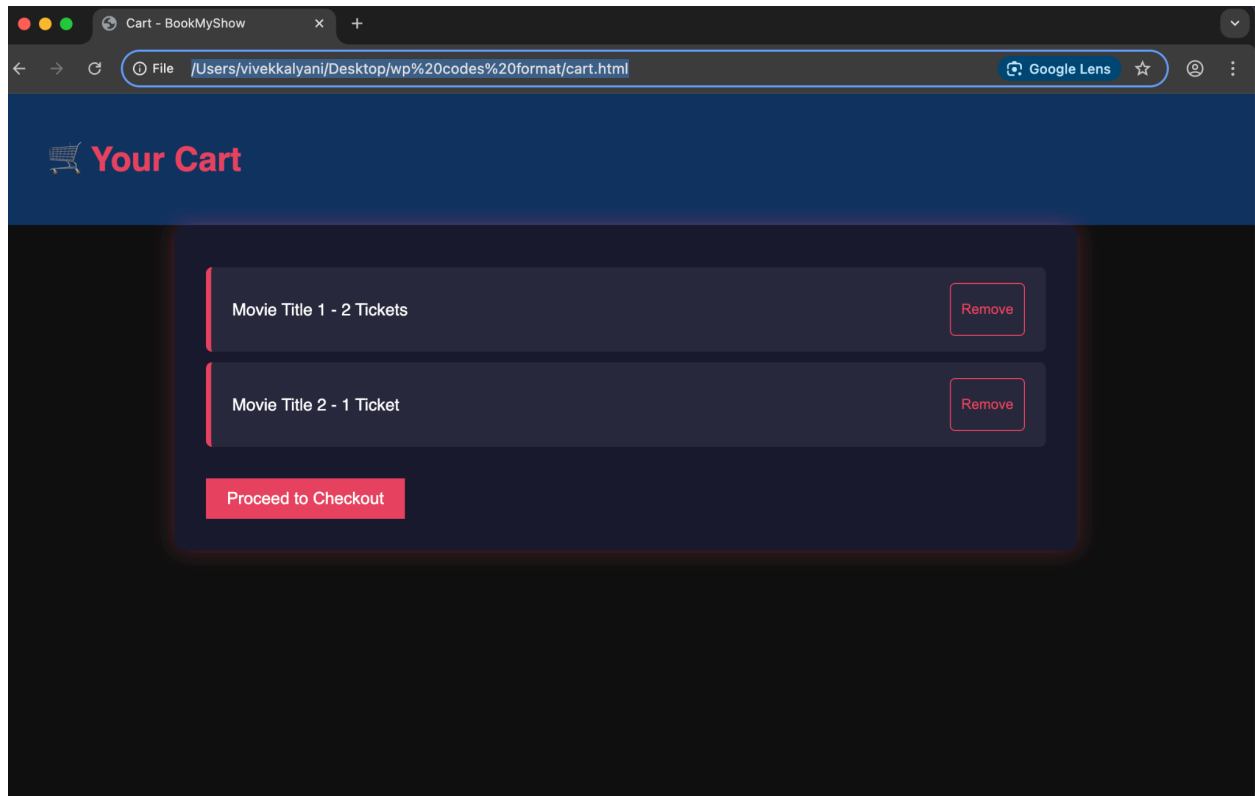
- Use sufficient color contrast for text readability.
- Provide visible focus indicators for keyboard navigation.
- Avoid relying solely on color to convey information (e.g., use icons or text too).

Code: cart-style.css

```
body {  
  font-family: Arial, sans-serif;  
  text-align: center;  
  background: #f0f0f0;  
}  
#screen {  
  margin: 20px auto;  
  width: 200px;  
  background: #222;  
  color: white;  
  padding: 10px;  
  border-radius: 5px;  
}  
#seats {  
  display: flex;
```

```
flex-wrap: wrap;
justify-content: center;
max-width: 300px;
margin: 0 auto;
}
.seat {
background-color: #ccc;
height: 25px;
width: 25px;
margin: 5px;
border-radius: 5px;
cursor: pointer;
}
.selected {
background-color: #6feaf6;
}
.occupied {
background-color: #444;
cursor: not-allowed;
}
button {
margin-top: 20px;
padding: 10px 20px;
background: #ff4b2b;
color: white;
border: none;
border-radius: 5px;
cursor: pointer;
}
```

Output:

**Conclusion:**

This experiment successfully demonstrates how CSS can transform a basic web layout into an engaging, easy-to-use interface. The booking cart page is made more organized and accessible, helping users clearly see their selections and interact with the cart seamlessly. The registration and login forms are enhanced with user-friendly designs that improve clarity and reduce errors. Through proper use of spacing, typography, color schemes, and responsiveness, the website now offers a polished look aligned with modern UI/UX standards. These improvements are critical in retaining users and encouraging completion of ticket purchases, making the BookMyShow Clone a more professional and practical application. Overall, the experiment highlights the importance of CSS in delivering a smooth user experience and preparing the site for further JavaScript and backend enhancements.

Experiment 5: Implementing User Registration and Login Using JavaScript

Aim:

The aim of this experiment is to implement user registration and login functionality on the BookMyShow Clone website using JavaScript. This includes creating forms to capture user data, validating inputs on the client side, and storing authentication state to allow users to access personalized features such as booking history and profile management.

Theory:

User authentication is the core mechanism through which a web application identifies and verifies users. It enables the system to provide personalized content, enforce access control, and maintain secure sessions. In the context of the BookMyShow Clone, user authentication allows customers to register, log in, view their bookings, and interact with the platform in a personalized manner.

JavaScript plays a vital role in the client-side aspect of this process by handling:

- Input validation
- UI updates based on login state
- Session persistence using Web Storage APIs

This experiment focuses on building a smooth, secure, and user-friendly frontend authentication system before backend integration.

1. Registration Form and Validation (Client-Side)

When a new user visits the site, they are presented with a registration form to create an account. This form typically includes:

- Full Name
- Email Address
- Password
- Confirm Password

JavaScript performs real-time validation before the data is submitted to the server:

 Validation Tasks Include:

Field	Validation Rule	JavaScript Function
Name	Not empty	<code>name.trim() !== ""</code>

Email	Valid format	<code>/^\S+@\S+\.\S+\$/</code>
Password	Min 8 chars, contains number and symbol	Regex check
Confirm Password	Matches Password	<code>password === confirmPassword</code>

Validation ensures:

- Data is clean and secure
- Server receives only valid requests
- Users get instant feedback (e.g., “Invalid email” or “Passwords don’t match”)

Benefits:

- Reduces server processing
- Minimizes data integrity issues
- Enhances user experience by avoiding full page reloads

2. Login Form and Verification

For returning users, a login form accepts their email and password. JavaScript checks:

- Non-empty fields
- Proper email format

Once validated, this data is typically passed to a PHP backend (or simulated during frontend development) for authentication. If login is successful, JavaScript stores the user's authenticated state in `localStorage` or `sessionStorage` for session persistence.

3. Authentication Persistence Using Web Storage

The Web Storage API in JavaScript provides a way to store small amounts of data in the browser. This helps maintain the user's login session:

Storage Type	Description	Use Case
--------------	-------------	----------

`localStorage` Persistent, survives browser close “Remember Me” functionality

`sessionStorage` Temporary, cleared on tab close Short-term sessions

4. Logout and Session Termination

To properly end a session:

Once logged out:

- The session is cleared
- UI returns to guest view
- User is redirected to homepage/login

5. UI Personalization and Redirection


After login, JavaScript is used to:

- Show user-specific UI elements (e.g., dashboard, booking history)
- Hide login/register buttons
- Redirect the user to their personalized page (e.g., `dashboard.html`)

6. Password Security (Client-side Best Practices)

Although sensitive authentication logic should occur on the backend (e.g., PHP + MySQL), the client-side still plays a role in promoting secure practices:

- Password complexity enforcement (e.g., at least 8 characters, includes symbols/numbers)
- Confirmation fields to reduce typos
- Obfuscation of input fields (`type="password"`)

 Important: Passwords are never stored or validated entirely on the frontend in real systems. JavaScript is only for initial validation and UI enhancement.

7. Preparing for Backend Integration

This experiment sets up the frontend structure for smooth backend integration:

- JavaScript can make AJAX requests to PHP scripts for login and registration.
- Validated data from the forms can be securely submitted via **POST**.
- On successful server response, JavaScript handles session persistence and redirection.

Security Considerations

Concern	Explanation	Solution
XSS Attacks	Malicious script reads localStorage	Sanitize all inputs, use CSP headers
Weak Passwords	Easy to guess or reuse	Enforce password complexity via JavaScript
Manual localStorage Tampering	User modifies login status	Always verify session on backend before showing protected content (in full implementation)

Code: login.html (using JAVASCRIPT)

```
// login.js
// Simulate storing login info
function registerUser() {
  const name = document.getElementById("reg-name").value;
  const email = document.getElementById("reg-email").value;
  const pass = document.getElementById("reg-pass").value;
  if (name && email && pass.length >= 6) {
    localStorage.setItem("user", JSON.stringify({ name, email, pass }));
    alert("Registration Successful!");
  } else {
    alert("Please fill all fields correctly.");
  }
}

function loginUser() {
  const email = document.getElementById("login-email").value;
  const pass = document.getElementById("login-pass").value;
```

```

const user = JSON.parse(localStorage.getItem("user"));
if (user && user.email === email && user.pass === pass) {
  sessionStorage.setItem("loggedIn", "true");
  alert("Login successful!");
  window.location.href = "profile.html";
} else {
  alert("Invalid credentials.");
}
}

<script src="login.js"></script>
<script>
document.querySelector(".auth-form:nth-child(1)").addEventListener("submit", e => {
  e.preventDefault();
  loginUser();
});
document.querySelector(".auth-form:nth-child(2)").addEventListener("submit", e => {
  e.preventDefault();
  registerUser();
});
</script>

```

Register.html (using JAVASCRIPT)

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Register - BookMyShow</title>
  <link rel="stylesheet" href="style.css" />
  <link rel="stylesheet" href="form-style.css" />
  <script>
    function validateRegisterForm(event) {
      event.preventDefault(); // Prevent form from submitting normally
      const username = document.getElementById("username").value.trim();
      const email = document.getElementById("email").value.trim();
      const password = document.getElementById("password").value;
      const emailPattern = /^[^\s@]+@[^\s@]+\.[a-z]{2,3}$/i;
      if (!username || !email || !password) {

```

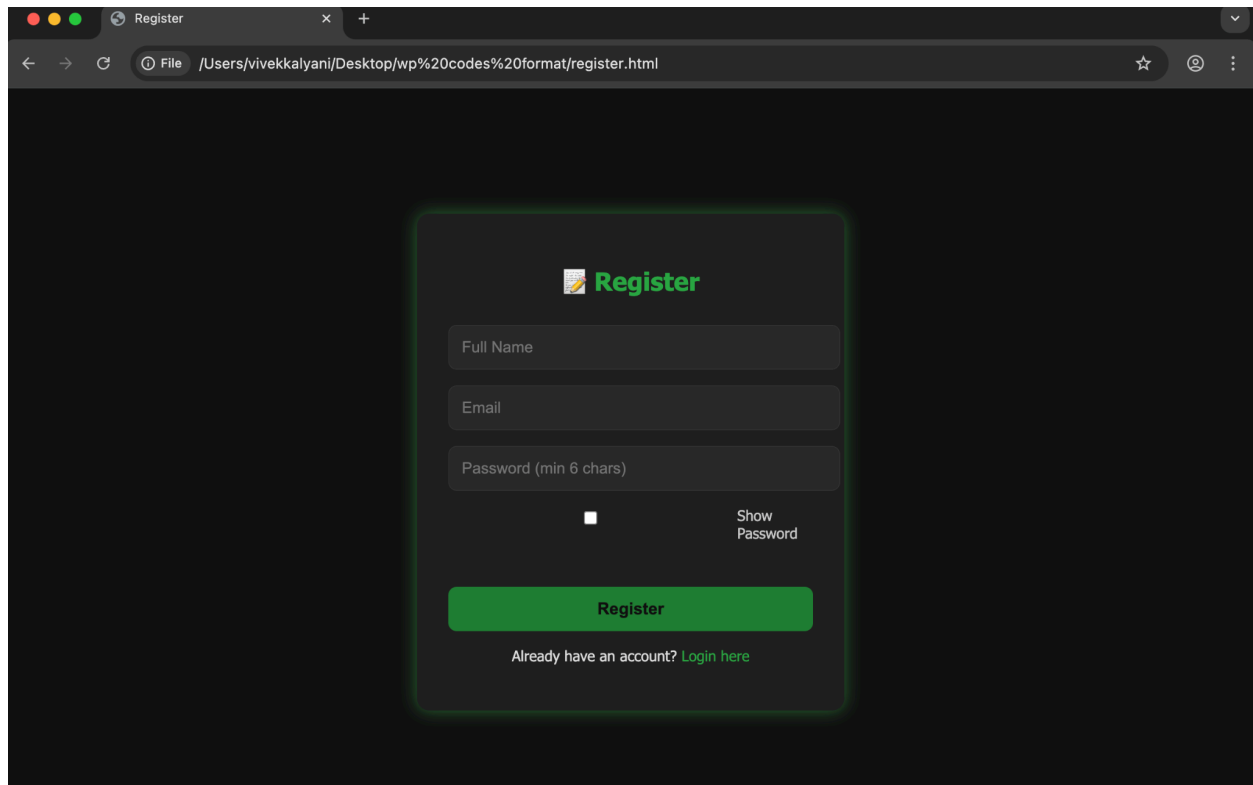
```

        alert("All fields are required.");
        return false;
    }
    if (!emailPattern.test(email)) {
        alert("Invalid email format.");
        return false;
    }
    if (password.length < 6) {
        alert("Password must be at least 6 characters long.");
        return false;
    }
    // Simulate successful registration
    alert("Registration successful! Redirecting to login page...");
    window.location.href = "login.html";
    return true;
}
</script>
</head>
<body>
<div class="container">
<header>
<h1>BookMyShow</h1>
<nav>
<ul>
<li><a href="index.html">Home</a></li>
<li><a href="register.html">Register</a></li>
<li><a href="login.html">Login</a></li>
</ul>
</nav>
</header>
<section>
<form onsubmit="validateRegisterForm(event)">
<input type="text" id="username" placeholder="Username" required />
<input type="email" id="email" placeholder="Email" required />
<input type="password" id="password" placeholder="Password" required minlength="6" />
<button type="submit">Register</button>
</form>
</section>
<footer>
<p>&copy; 2025 BookMyShow Clone. All rights reserved.</p>

```

```
</footer>
</div>
</body>
</html>
```

Output:



Conclusion:

The experiment successfully implements a functional and user-friendly registration and login system using JavaScript for client-side operations. The forms effectively capture and validate user inputs, providing instant feedback that improves the registration and login experience. By leveraging localStorage/sessionStorage, the application maintains user login states, allowing seamless navigation without frequent re-authentication.

This enhances user engagement by personalizing the BookMyShow Clone website and enabling features like booking history and saved preferences. The approach also reduces backend server load by catching invalid inputs early.

Overall, this experiment demonstrates the importance of JavaScript in building interactive, secure, and user-centered web applications. The implemented authentication system is a critical step towards a fully functional movie booking platform, ensuring that users can manage their accounts safely and conveniently.

Experiment 6: Enhancing User Login with Authentication Persistence Using JavaScript

Aim:

The aim of this experiment is to enhance the user login functionality by implementing persistent authentication using JavaScript. This includes managing user session data via `localStorage` or `sessionStorage`, handling login redirection, and maintaining login status across page reloads or revisits to the BookMyShow Clone website.

Theory:

Authentication persistence is a fundamental usability and security feature in any web application that involves user login. It ensures that once users have logged in, their session or login state is remembered—either for the current session or across future visits—without requiring them to re-enter their credentials on every page load or visit. In this experiment, we use JavaScript's Web Storage API—specifically `localStorage` and `sessionStorage`—to implement client-side authentication persistence in the BookMyShow Clone project.

Why is Authentication Persistence Important?

Without authentication persistence:

- Users must log in repeatedly whenever they reload a page or revisit the site.
- The application will lose the authenticated state after each navigation or tab close.
- It leads to frustration, decreased usability, and poor user retention.

With authentication persistence:

- The login session is retained across tabs or visits.
- The interface can dynamically adapt based on logged-in status.
- Personalized content (e.g., user dashboard, booking history) is shown immediately.

Web Storage API Overview

The Web Storage API in JavaScript provides two key tools:

Storage Type	Lifetime	Accessible In
<code>localStorage</code>	Persistent (remains even after browser is closed)	All tabs/windows of the same origin

<code>sessionStorage</code>	Temporary (cleared when the tab/window is closed)	Only in the current tab/session
-----------------------------	---	---------------------------------

These are key-value pair storage systems and are much more efficient and secure (compared to cookies) for non-sensitive client-side data like login flags or UI states.

How It Works – Step-by-Step Authentication Persistence

◆ 1. After Successful Login (Back-end)

Once the backend (e.g., a PHP login system) confirms valid credentials:

- A success message is returned.
- JavaScript on the frontend captures this success and stores the login state in `localStorage`.

◆ 2. Session Check on Page Load

Whenever the user visits the site, a script checks whether the user is already logged in by querying `localStorage`.

◆ 3. Persistent Redirection

To enhance user experience:

- If a logged-in user visits `login.html`, they are redirected to `dashboard.html`.

◆ 4. Logout Functionality

For proper session management, a logout button clears the login data from `localStorage`:

Benefits in BookMyShow Clone

- Seamless User Experience: No need to log in repeatedly.
- Fast Page Loads: UI updates instantly based on login status.
- Cross-Page State Management: Allows shared access to login state across all pages using JavaScript.

Security Considerations

While `localStorage` is useful, it has limitations:

Limitation	Description
Client-Side Only	Can be manually edited by users via browser dev tools.
No Encryption	Sensitive data like passwords or tokens should never be stored here.
Susceptible to XSS	If your site is vulnerable to Cross-Site Scripting (XSS), localStorage can be compromised.

Solutions:

- Only store non-sensitive data (e.g., email, isLoggedIn).
- Combine with server-side session verification in production.
- Implement XSS protection, content security policies, and input sanitization.

Beyond Web Storage: A Path to Token-Based Authentication

While Web Storage works well for simple clones or prototypes, real-world systems use:


- JWT (JSON Web Tokens) for authentication.
- Access tokens with expiration time.
- Secure HTTP-only cookies for better protection.

This experiment lays the foundation for future enhancements involving token-based security models.

Code: profile.html

```
<!-- profile.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Profile</title>
  <link rel="stylesheet" href="styles.css" />
</head>
<body>
```

```

<div class="profile-container">
  <h2>  User Profile</h2>
  <p id="username"></p>
  <button onclick="logout()">Logout</button>
</div>
<script>
  const isLoggedIn = sessionStorage.getItem("loggedIn");
  const user = JSON.parse(localStorage.getItem("user"));
  if (!isLoggedIn || !user) {
    window.location.href = "login.html";
  } else {
    document.getElementById("username").innerText = `Welcome, ${user.name}`;
  }
  function logout() {
    sessionStorage.removeItem("loggedIn");
    window.location.href = "login.html";
  }
</script>
</body>
</html>

```

Conclusion:

The experiment successfully implements persistent user authentication using JavaScript's **localStorage**. Users remain logged in even after page reloads or closing and reopening the browser, significantly enhancing usability and convenience.

The login form validation ensures that only users with valid credentials gain access, while the redirection after login provides a seamless transition to the personalized interface. The dynamic UI updates based on authentication state reinforce the personalized experience, increasing user engagement.

This client-side authentication persistence serves as an important bridge toward a fully integrated backend authentication system with PHP and MySQL, allowing smoother user management and a professional-grade BookMyShow Clone website.

Overall, this experiment highlights the effectiveness of JavaScript storage APIs in managing user sessions and improving web application interactivity and user retention.

Experiment 7: User Registration Handling Using PHP

Aim:

The aim of this experiment is to develop a PHP script that handles user registration on the BookMyShow Clone website. The script will accept user inputs such as name, email address, password, and other required details; validate them; store the data securely in a database; and provide appropriate feedback for successful or failed registrations.

Theory:

User registration is the gateway through which new users gain access to the features of any secure web application. In the BookMyShow Clone, user registration is essential for enabling personalized services like movie ticket booking, managing seat preferences, viewing booking history, writing reviews, and editing profile information. This experiment involves building a secure, functional, and scalable user registration system using PHP (server-side scripting) and MySQL (relational database).

1. Purpose of the Registration System

The objective of the registration module is to:

- Allow users to create unique accounts.
- Securely store user credentials and personal data.
- Lay the foundation for authenticated and personalized access.
- Prevent duplicate or fraudulent accounts.

2. Frontend Registration Form

The registration form typically includes fields such as:

- Full Name
- Email Address
- Password
- (Optional) Confirm Password, Phone Number, etc.

3. Backend Processing with PHP

Once the form is submitted, the `register.php` script handles the logic behind:

- Input validation
- Duplicate account checks
- Secure password hashing
- Data insertion

- User feedback

4. Input Validation

Proper validation ensures that users submit meaningful and correct data.

5. Email Duplication Check

Before inserting the new user into the database, the script checks whether the email already exists to prevent duplicate accounts:

6. Password Hashing

Rather than storing passwords as plain text (which is highly insecure), PHP's `password_hash()` function is used to encrypt the password before saving it to the database:

This ensures:

- Even if the database is breached, attackers cannot easily retrieve user passwords.
- Complies with modern security practices (salting and hashing automatically handled).

7. Inserting the User into MySQL

Once all validations are passed and the password is hashed, the new user record is inserted into the database:

8. Error Handling

The script provides clear and user-friendly error messages to handle:

- Database connection issues
- SQL execution errors
- Form input mistakes
- Already registered email addresses

These messages help guide the user to complete the process successfully.

9. Security Best Practices

Security Feature

Implementation

Input Sanitization	<code>trim()</code> , <code>htmlspecialchars()</code>
SQL Injection Protection	Prepared statements with <code>mysqli</code>
Password Security	<code>password_hash()</code> and <code>password_verify()</code>
Error Concealment	Avoid exposing detailed DB errors to users
HTTPS Use	Recommended to encrypt data in transit

10. Additional Enhancements (Optional)

- Email Verification: Send confirmation link before activating the account.
- Password Strength Meter: Frontend JS to show strength.
- CAPTCHA: Prevent bot registrations.
- Phone Number Verification: OTP-based verification.

These additions make the system even more robust and production-ready.

11. Integration in BookMyShow Clone

The registration system links directly to:

- Login Module (users must register before logging in).
- Checkout System (only registered/logged-in users can book).
- User Dashboard (booking history, preferences, etc.).
- Reviews & Ratings Module (to associate feedback with real users).

User information like `user_id`, `email`, and `fullname` are often stored in the session post-login for personalization and tracking.

Code: register.php

```

<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "bookmyshow";
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$name = $_POST['name'];
$email = $_POST['email'];
$password = password_hash($_POST['password'], PASSWORD_DEFAULT);
$sql = "INSERT INTO users (name, email, password) VALUES ('$name', '$email', '$password')";
if ($conn->query($sql) === TRUE) {
    echo "Registration successful. <a href='login.html'>Login Now</a>";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>

```

Conclusion:

This experiment demonstrates the effective use of PHP to handle user registration for a BookMyShow Clone website. By validating inputs and securely hashing passwords, the system ensures that user data is stored safely and correctly.

The error handling mechanisms improve user experience by giving immediate feedback on issues like missing fields or duplicate emails. Successful registrations trigger confirmations and allow users to move forward to login and access personalized features.

This PHP-based registration process is essential for building a robust user management system, laying the foundation for further functionalities like login authentication, session management, and order history tracking.

Overall, this experiment highlights the importance of secure backend scripting in creating reliable and trustworthy web applications, making the BookMyShow Clone both functional and user-friendly.

Experiment 8: User Login Handling Using PHP

Aim:

The aim of this experiment is to develop a PHP script that handles user login for the BookMyShow Clone website. The script will accept user credentials (email and password), validate them against stored user data, manage authentication, and provide appropriate feedback or redirection based on the login success or failure.

Theory:

The user login system is one of the most fundamental and security-sensitive features in any web-based application that deals with personalized services like ticket booking, viewing history, and profile management. In the BookMyShow Clone, this system is responsible for securely authenticating users and providing them access to their personalized dashboard.

This experiment involves implementing a login mechanism using PHP for backend scripting and MySQL for database operations, with a strong emphasis on input validation, password hashing, session handling, and error feedback to ensure a smooth and secure user experience.

1. Objective of the Login System

The goal of the login module is to:

- Authenticate registered users by verifying credentials.
- Initiate a session for persistent user access.
- Secure user data by enforcing proper password handling.
- Prevent unauthorized access to protected areas like bookings or profiles.

2. Frontend Login Form Design

3. Backend Processing (PHP Script)

When the form is submitted, the data is sent to a PHP script (`login.php`) via POST. The following steps are executed:

4. Input Validation

Before querying the database, the script performs validation:

5. Querying the Database

Using a parameterized SQL query (to prevent SQL injection), the script checks whether the user exists:

6. Password Verification

PHP's `password_verify()` is used to compare the entered password with the hashed password stored in the database:

7. Session Management

Upon successful login, a session is initiated to track the user's authenticated state:

8. Error Handling and Feedback

Robust error messages guide the user in case of:

- Wrong password.
- Invalid email.
- Empty fields.

These are usually shown without refreshing the page (using AJAX or inline PHP), enhancing user experience:

9. Security Best Practices Implemented

Security Aspect	Implementation
Password hashing	<code>password_hash()</code> and <code>password_verify()</code>
SQL Injection	Prepared statements using <code>mysqli_prepare()</code>
Session Hijacking Prevention	Use of <code>session_regenerate_id(true)</code> after login
Session Timeout	Inactivity logout through session time checks
XSS Protection	Sanitizing user input using <code>htmlspecialchars()</code>

10. Optional Enhancements

To further improve the login system:

- Remember Me checkbox to set login cookies securely.
- Two-Factor Authentication (2FA) using OTP or email verification.
- Captcha Integration to prevent bot logins.
- Account Lockout Mechanism after multiple failed attempts.

11. Integration with Overall System

The login system is linked to:

- User Registration (where passwords are hashed and stored).
- Cart and Checkout (only logged-in users can book tickets).
- Profile Section (display/update user details).
- Order History Page (shows past bookings based on `user_id`).

All these components rely on the session variables created after login to fetch and personalize data.

Code: login.php

```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "bookmyshow";

$conn = new mysqli($servername, $username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$email = $_POST['email'];
$pass = $_POST['password'];

$sql = "SELECT * FROM users WHERE email = '$email'";
$result = $conn->query($sql);

if ($result->num_rows === 1) {
    $row = $result->fetch_assoc();
    if (password_verify($pass, $row['password'])) {
```

```

session_start();
$_SESSION['user'] = $row['name'];
header("Location: profile.php");
} else {
    echo "Invalid password.";
}
} else {
    echo "No account found with that email.";
}

$conn->close();
?>

```

Output:

The screenshot displays a MySQL database management interface. At the top, a navigation bar includes tabs for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Tracking, and Triggers. Below this, a status bar indicates "Showing rows 0 - 3 (4 total, Query took 0.0001 seconds.)". The SQL query entered is `SELECT * FROM `users``. Below the query, there are options for Profiling, Edit inline, Edit, Explain SQL, Create PHP code, and Refresh. A filter section shows "Show all", "Number of rows: 25", "Filter rows: Search this table", and "Sort by key: None". An "Extra options" button is also present. The main area displays a table with 4 rows of user data:

	user_id	full_name	email	password	user_type	created_at
<input type="checkbox"/>	1	Aarav Mehta	aarav.mehta@example.com	password123	student	2025-05-15 13:35:22
<input type="checkbox"/>	2	Sanya Sharma	sanya.sharma@example.com	password456	student	2025-05-15 13:35:22
<input type="checkbox"/>	3	Ravi Verma	ravi.verma@example.com	teachpass789	instructor	2025-05-15 13:35:22
<input type="checkbox"/>	4	Kritika Iyer	kritika.iyer@example.com	adminpass321	admin	2025-05-15 13:35:22

Below the table, there are options to "Check all", "With selected: Edit, Copy, Delete, Export". Another filter section is present with "Show all", "Number of rows: 25", "Filter rows: Search this table", and "Sort by key: None". The "Query results operations" section includes buttons for Print, Copy to clipboard, Export, Display chart, and Create view. The "Bookmark this SQL query" section has a label input field, a checkbox for "Let every user access this bookmark", and a "Bookmark this SQL query" button.

Conclusion:

This experiment successfully implements a secure user login system using PHP and MySQL for the BookMyShow Clone website. By validating user inputs and verifying credentials against hashed passwords, the system maintains high security standards critical for protecting user data.

Error handling improves the user experience by guiding users when incorrect credentials are entered, while session management allows smooth navigation and access to personalized features post-login.

The login process forms a cornerstone of the overall authentication and authorization system, enabling features such as booking management, profile editing, and secure access control.

Through this experiment, one understands the importance of encryption, validation, and session handling in developing reliable, secure web applications, which is vital for any real-world online ticket booking system like BookMyShow.

Experiment 9: Shopping Cart Management Using PHP and MySQL

Aim:

The aim of this experiment is to develop a PHP script integrated with a MySQL database that allows users of the BookMyShow Clone website to manage their shopping cart. This includes functionalities to add movie tickets or event bookings to the cart, view current cart contents, update quantities, and remove items, ensuring persistent cart data across user sessions.

Theory:

The shopping cart is a vital intermediary system in any digital transaction workflow, particularly for e-commerce and booking platforms like BookMyShow. It acts as a dynamic and interactive interface where users can temporarily store and manage their selected items—in this case, movie tickets—prior to completing the booking through the checkout process.

In the BookMyShow Clone Web Application, the cart system is developed using PHP as the server-side scripting language and MySQL for persistent data storage. This architecture ensures that user selections are not only maintained during a session but also persist across different logins and devices for a consistent and reliable user experience.

1. Purpose and Functionality of the Cart

The shopping cart performs the following critical roles:

- Temporarily holds selected movie tickets: Including movie ID, show time, seat numbers, and quantities.
- Allows updates or deletions: Users can change quantities or remove tickets from the cart.
- Acts as a staging area before checkout: Data from the cart is used during final booking.
- Enables persistent cart state: Through database-backed storage, selections remain intact even if the session ends or the user logs in from a different device.

This makes the cart system an essential module for user convenience, data consistency, and seamless navigation within the application.

2. Database Design and Tables

The backend uses a `cart` table in the MySQL database. A typical table structure may include:

Column Name	Data Type	Description
<code>cart_id</code>	<code>INT (AUTO_INCREMENT)</code>	Unique identifier for the cart entry

user_id	INT	References the logged-in user
movie_id	INT	References the movie being booked
show_id	INT	References a specific show timing
seat_number	VARCHAR	Seat ID or range (A1, A2, etc.)
quantity	INT	Number of tickets
price_per_ticket	DECIMAL	Price per seat
total_price	DECIMAL	Computed price (quantity \times price per ticket)
added_at	TIMESTAMP	When the cart item was added or updated

This structure ensures modular, scalable data handling for cart items.

3. Core Operations and PHP Logic

Several PHP scripts manage cart functionality, such as:

a. Adding Items to Cart:

When a user selects a movie, show, and seats, a PHP script:

- Validates the seat availability (checks `seat_availability` table).
- Inserts a new row into the `cart` table if the item is new.
- Updates the row if the item already exists (e.g., changing quantity).

b. Viewing Cart:

Another PHP script fetches all rows from the cart table for the logged-in user and displays the content dynamically:

- Movie and show details are fetched using JOINS.
- Seat numbers, prices, and quantities are displayed in a styled table or card format.

c. Updating/Removing Items:

- The user can adjust the number of tickets or remove an item.
- AJAX or form submissions trigger PHP scripts that either:
 - **UPDATE** the quantity in the cart table, or
 - **DELETE** the row entirely from the cart.

4. Cart Persistence and Session Management

One major strength of this implementation is that the cart is stored in the database, not in browser memory (like **localStorage** or cookies). This provides several advantages:

- Users can retrieve their cart from different devices after logging in.
- The cart data is maintained even after a browser crash or logout.
- Server-side control over data integrity and validation is possible.

The **user_id** in the cart table ensures that the cart is always associated with the correct user session. PHP **\$_SESSION** variables help maintain the user's login state securely throughout the interaction.

5. Validation and Error Handling

To ensure the cart operates reliably:

- Seat availability is rechecked before every addition.
- Seat duplication is prevented by disallowing multiple users from adding the same seat (reserved in real-time or at checkout).
- Invalid quantities (e.g., zero or negative values) are blocked by input validation.
- Graceful error messages are shown to the user using **<div class="error">** tags or JavaScript alerts.

6. AJAX for Seamless UX

For a smoother experience, AJAX can be used to add, update, or delete cart items without full-page reloads. This improves responsiveness and feels more modern:

7. Integration with Checkout Module

The cart is directly linked to the checkout module. When the user proceeds to checkout:

- The cart items are read and verified.
- They are transformed into a finalized order.
- Corresponding seats are marked as booked.
- The cart is then cleared for that user to avoid duplication.

The **shopping cart** is a vital intermediary system in any digital transaction workflow, particularly for e-commerce and booking platforms like BookMyShow. It acts as a dynamic and interactive interface where users can temporarily store and manage their selected items—in this case, movie tickets—prior to completing the booking through the checkout process.

In the **BookMyShow Clone Web Application**, the cart system is developed using **PHP** as the server-side scripting language and **MySQL** for persistent data storage. This architecture ensures that user selections are not only maintained during a session but also persist across different logins and devices for a consistent and reliable user experience.

1. Purpose and Functionality of the Cart

The shopping cart performs the following critical roles:

- **Temporarily holds selected movie tickets:** Including movie ID, show time, seat numbers, and quantities.
- **Allows updates or deletions:** Users can change quantities or remove tickets from the cart.
- **Acts as a staging area before checkout:** Data from the cart is used during final booking.
- **Enables persistent cart state:** Through database-backed storage, selections remain intact even if the session ends or the user logs in from a different device.

This makes the cart system an essential module for **user convenience**, **data consistency**, and **seamless navigation** within the application.

2. Database Design and Tables

The backend uses a **cart** table in the MySQL database. A typical table structure may include:

Column Name	Data Type	Description
cart_id	INT (AUTO_INCREMENT)	Unique identifier for the cart entry
user_id	INT	References the logged-in user

movie_id	INT	References the movie being booked
show_id	INT	References a specific show timing
seat_number	VARCHAR	Seat ID or range (A1, A2, etc.)
quantity	INT	Number of tickets
price_per_ticket	DECIMAL	Price per seat
total_price	DECIMAL	Computed price (quantity × price per ticket)
added_at	TIMESTAMP	When the cart item was added or updated

This structure ensures modular, scalable data handling for cart items.

3. Core Operations and PHP Logic

Several PHP scripts manage cart functionality, such as:

a. Adding Items to Cart:

When a user selects a movie, show, and seats, a PHP script:

- Validates the seat availability (checks `seat_availability` table).
- Inserts a new row into the `cart` table if the item is new.
- Updates the row if the item already exists (e.g., changing quantity)

b. Viewing Cart:

Another PHP script fetches all rows from the cart table for the logged-in user and displays the content dynamically:

- Movie and show details are fetched using JOINS.
- Seat numbers, prices, and quantities are displayed in a styled table or card format.

c. Updating/Removing Items:

- The user can adjust the number of tickets or remove an item.
- AJAX or form submissions trigger PHP scripts that either:
 - **UPDATE** the quantity in the cart table, or
 - **DELETE** the row entirely from the cart.

4. Cart Persistence and Session Management

One major strength of this implementation is that the **cart is stored in the database**, not in browser memory (like **localStorage** or cookies). This provides several advantages:

- Users can retrieve their cart from different devices after logging in.
- The cart data is maintained even after a browser crash or logout.
- Server-side control over data integrity and validation is possible.

The **user_id** in the cart table ensures that the cart is always associated with the correct user session. PHP **\$_SESSION** variables help maintain the user's login state securely throughout the interaction.

5. Validation and Error Handling

To ensure the cart operates reliably:

- **Seat availability is rechecked** before every addition.
- **Seat duplication is prevented** by disallowing multiple users from adding the same seat (reserved in real-time or at checkout).
- Invalid quantities (e.g., zero or negative values) are blocked by input validation.
- Graceful error messages are shown to the user using `<div class="error">` tags or JavaScript alerts.

6. AJAX for Seamless UX

For a smoother experience, **AJAX** can be used to add, update, or delete cart items without full-page reloads. This improves responsiveness and feels more modern:

javascript

7. Integration with Checkout Module

The cart is directly linked to the **checkout module**. When the user proceeds to checkout:



- The cart items are read and verified.
- They are transformed into a finalized order.
- Corresponding seats are marked as booked.

- The cart is then cleared for that user to avoid duplication.

Code: remove_from_cart.php

```
<?php
session_start();
$id = $_GET['id'];
unset($_SESSION['cart'][$id]);
header("Location: view_cart.php");
?>
```

View_cart.php

```
<?php
session_start();
echo "<h2> Your Cart</h2>";
if (isset($_SESSION['cart'])) {
    echo "<ul>";
    foreach ($_SESSION['cart'] as $id => $item) {
        echo "<li>{$item['name']} - Qty: {$item['quantity']}
        <a href='remove_from_cart.php?id=$id'> Remove</a></li>";
    }
    echo "</ul><a href='checkout.php'>Proceed to Checkout</a>";
} else {
    echo "Cart is empty!";
}
?>
```

Add_to_cart.php

```
<?php
session_start();
$id = $_POST['movie_id'];
$name = $_POST['movie_name'];
$qty = $_POST['quantity'];
if (!isset($_SESSION['cart'])) {
    $_SESSION['cart'] = [];
}
$_SESSION['cart'][$id] = [
```

```
'name' => $name,
'quantity' => $qty
];
echo "Added to cart!";
?>
```

Output:

The screenshot displays the phpMyAdmin interface for a database table. The top navigation bar includes links for Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, and Tracking. The 'Table structure' tab is active, showing a table with the following columns:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	cart_id	int(10)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	user_id	int(10)		UNSIGNED	No	None			Change Drop More
3	course_id	int(10)		UNSIGNED	No	None			Change Drop More
4	added_at	timestamp			No	current_timestamp()			Change Drop More

Below the table structure, there are options to 'Check all', 'With selected:', 'Browse', 'Change', 'Drop', 'Primary', 'Unique', 'Index', 'Spatial', and 'Find'. There are also links for 'Print', 'Propose table structure', 'Track table', 'Move columns', and 'Normalize'. A form to 'Add' a new column is visible, with '1' in the input field and 'after added_at' in the dropdown menu. The 'Indexes' section shows the following indexes:

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Rename Drop	PRIMARY	BTREE	Yes	No	cart_id	0	A	No	
Edit Rename Drop	user_id	BTREE	No	No	user_id	0	A	No	
Edit Rename Drop	course_id	BTREE	No	No	course_id	0	A	No	

At the bottom, there is a form to 'Create an index on' with '1' in the input field and a 'Go' button.

Conclusion:

This experiment successfully creates a robust shopping cart system using PHP and MySQL for the BookMyShow Clone website. The integration with MySQL allows cart data persistence, providing a seamless and consistent user experience.

Users can efficiently add, update, and remove booking items from their cart, with all changes immediately reflected in the database. This persistence ensures that user selections are saved even if sessions expire or the user logs out, which is essential for a real-world ticket booking application.

The experiment highlights key programming concepts such as CRUD operations, session management, database interactions, and error handling in PHP, all of which are vital for e-commerce and booking systems.

Experiment 10: Checkout Process Handling Using PHP and MySQL

Aim:

The aim of this experiment is to develop a comprehensive PHP script integrated with MySQL to handle the checkout process for the BookMyShow Clone website. This includes processing the user's cart data, validating input, managing order details, updating seat availability, and providing appropriate feedback to the user upon successful or failed transactions.

Theory:

The checkout system is a crucial and final step in the user journey on any ticket booking or e-commerce platform. It is where the transaction is completed, and the system transitions from cart status to a confirmed order. In the context of the BookMyShow Clone Web Application, the checkout module is responsible for validating selected tickets, simulating or processing payment, storing order details, and updating seat availability in the database. This ensures the accuracy, reliability, and integrity of the booking system.

1. Purpose and Functionality

The checkout system allows users to finalize their ticket bookings after they have added desired movie shows and seats to their cart. It performs several key tasks:

- Verifies user session and login status.
- Fetches the latest cart data for the user.
- Validates all form inputs like name, contact number, and payment details.
- Confirms seat availability to avoid double bookings.
- Stores complete order information in the database.
- Updates seat availability to reflect the confirmed booking.
- Displays a confirmation message and order summary to the user.

This entire process is managed using PHP scripts that communicate with the MySQL database, ensuring both server-side validation and secure data handling.

2. Database Interaction and Workflow

The checkout process typically involves the following database interactions:

1. Cart Verification:

The selected seats and showtime are fetched from the `cart` table using the user's session ID or user ID. The system checks whether the selected seats are still available in the `seat_availability` or `show_seats` table.

2. Input Validation:

Before processing the checkout, user inputs like full name, email, and payment information are validated on both client and server sides. This prevents malicious data and accidental omissions.

3. Order Insertion:

If validation succeeds, an entry is made into the `orders` table which typically contains:

- User ID
 - Show/Movie ID
 - Seat numbers
 - Quantity
 - Total price
 - Timestamp of transaction
 - Booking status
4. Seat Update:
After inserting the order, the script updates the corresponding seat status in the `seat_availability` table to 'booked' or removes those seats from the available pool.
5. Transaction Management:
To ensure that no double booking or inconsistency occurs, the script may use SQL transactions. If any part of the process fails (e.g., seat is already booked), the entire transaction is rolled back, and the user is prompted with an error.

3. Payment Simulation or Integration

In a real-world scenario, a payment gateway like Razorpay, Stripe, or PayPal would be integrated. However, in most academic or simulated versions:

- The payment is mocked or simulated using a form where the user enters dummy card information.
- Upon "success," the system treats the booking as confirmed and proceeds to save order details.

This simulated approach allows developers to demonstrate logic flow without handling real financial data.

4. User Feedback and Confirmation

Upon successful booking:

- The user is shown a success message.
- An order summary is displayed which includes movie/show details, seat numbers, price breakdown, and a booking reference number.
- Optionally, a confirmation email or SMS can be triggered.

If errors occur during any phase (validation failure, database write error, seat unavailability), the system catches and displays the appropriate error message using PHP error handling mechanisms and redirects the user to retry.

5. Backend Concepts Demonstrated

This module touches on several core backend and database concepts:

- Server-side validation to ensure all submitted data is valid and secure.
- Transactional operations to maintain database integrity.
- CRUD operations in MySQL to insert order records and update seat status.

- Session management to associate bookings with the logged-in user.
- Error handling to manage failed operations gracefully.
- Security practices such as input sanitization and session verification.

Code:

checkout.php

```
<?php
session_start();
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "bookmyshow";
$conn = new mysqli($servername, $username, $password, $dbname);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
if (!isset($_SESSION['cart']) || empty($_SESSION['cart'])) {
    die("Cart is empty. <a href='index.html'>Go back</a>");
}
foreach ($_SESSION['cart'] as $id => $item) {
    $name = $item['name'];
    $qty = $item['quantity'];
    $sql = "INSERT INTO orders (movie_id, movie_name, quantity) VALUES ('$id', '$name', '$qty')";
    $conn->query($sql);
}
echo "<h2>✔ Checkout Successful!</h2><p>Thank you for your booking.</p>";
unset($_SESSION['cart']);
$conn->close();
?>
```

Conclusion:

The checkout handling experiment successfully implements a critical function of the BookMyShow Clone by enabling users to complete their ticket bookings with confidence.

By integrating PHP with MySQL, the system ensures data integrity and prevents overbooking by managing seat availability in real-time during checkout. The validation processes reduce errors and improve user experience by guiding users through correct data entry and clear feedback.

This experiment reinforces key development practices such as secure server-side validation, transactional updates, and user-friendly feedback mechanisms — all essential for robust online booking applications.

Overall, the completed checkout process not only finalizes purchases efficiently but also ensures that users receive clear confirmations, contributing to their trust and satisfaction with the platform.

Through this exercise, a practical understanding of combining database-driven backend processing with user-centric design principles is achieved, equipping developers to build scalable and reliable booking systems.