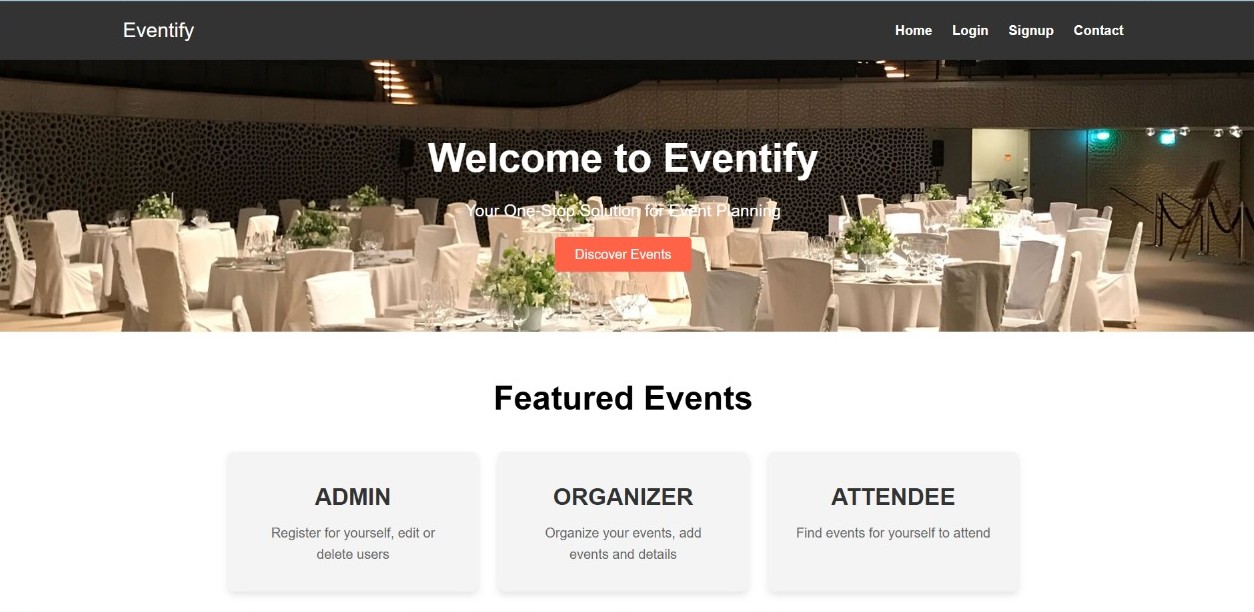
Event Management System - Project Documentation

**1. Introduction**

The Event Management System is a web-based application designed to streamline the process of event creation, management, and participation. It serves two primary roles: the **Organizer**, who creates and manages events, and the **Attendee**, who registers for and participates in events. This system leverages modern web technologies such as **Spring Boot**, **JSP**, **AJAX**, **JavaScript**, and **Hibernate** to create a seamless experience for both the organizers and attendees.



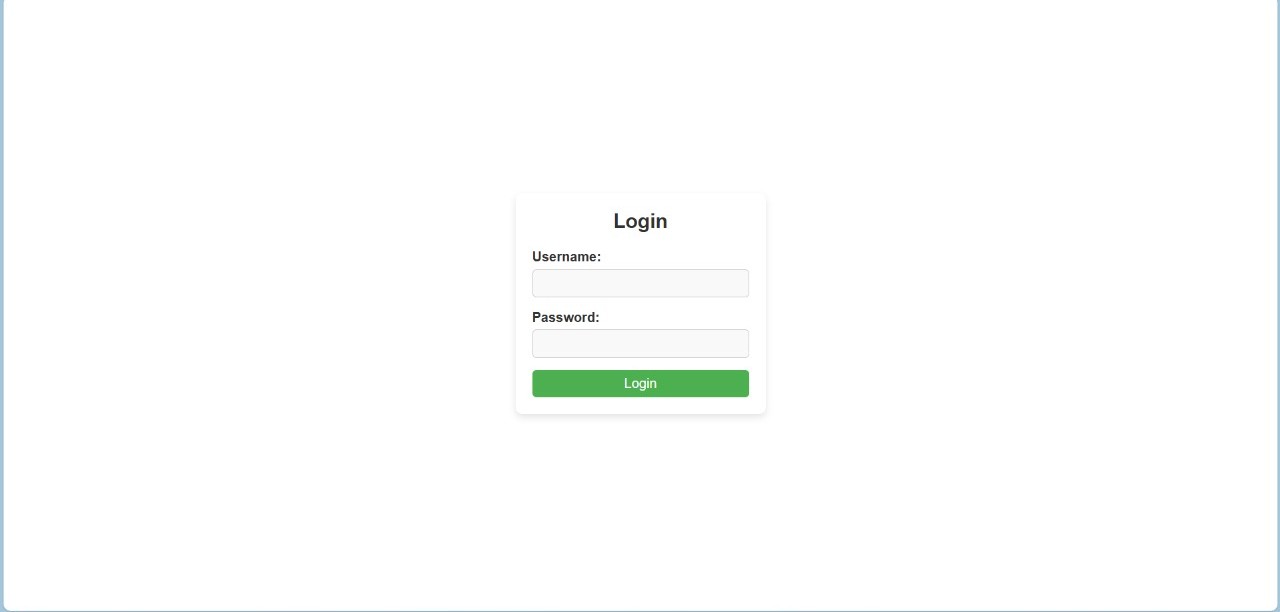
**2. System Requirements**

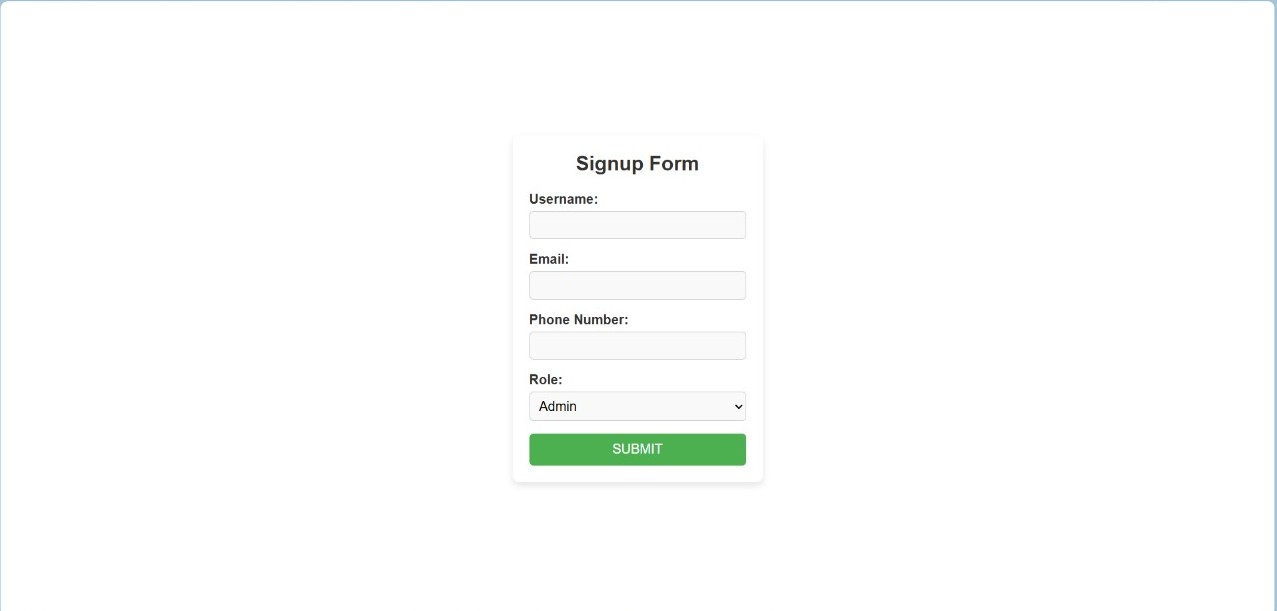
* **Frontend**: HTML, CSS, JSP (JavaServer Pages), JavaScript (AJAX), Bootstrap (for responsive design)
* **Backend**: Java, Spring Boot, Hibernate, Spring Security
* **Database**: MySQL (or any relational database)
* **Tools**: Maven, IntelliJ IDEA (or Eclipse), Tomcat for deployment, JUnit for testing

**3. Functionalities**

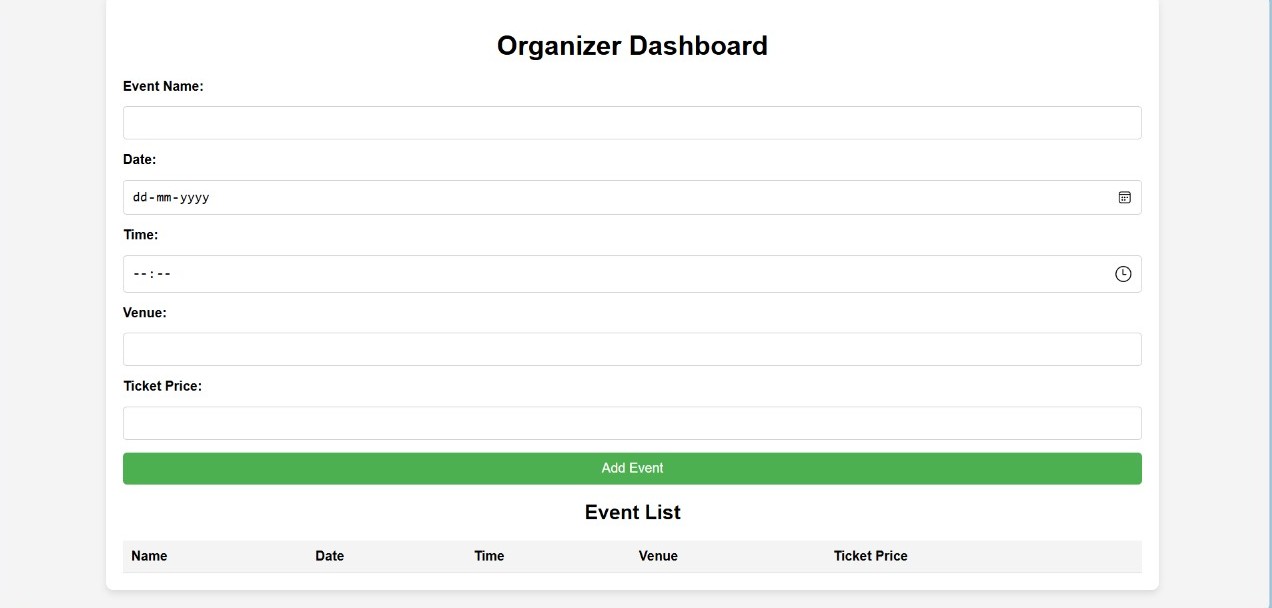
The **Event Management System** is designed to support the following key functionalities:

1. **User Registration & Login**
   * Users can sign up with their personal details (name, email, password, phone number).
   * Users can log in and access the system as either an **Organizer** or an **Attendee** based on their credentials.





1. **Event Creation (Organizer)**
   * Organizers can create events with details such as:
     + Event Name
     + Date, Time, and Venue
     + Ticket Price
   * The system validates event creation and ensures that each event has complete and valid details.



1. **Event Listing (Attendee)**
   * Attendees can view a list of all upcoming events available to them.
   * Events are displayed with details such as event name, date, time, venue, and ticket price.
   * Attendees can register for events (optional functionality can be added based on the system's requirements).

A screenshot of a website

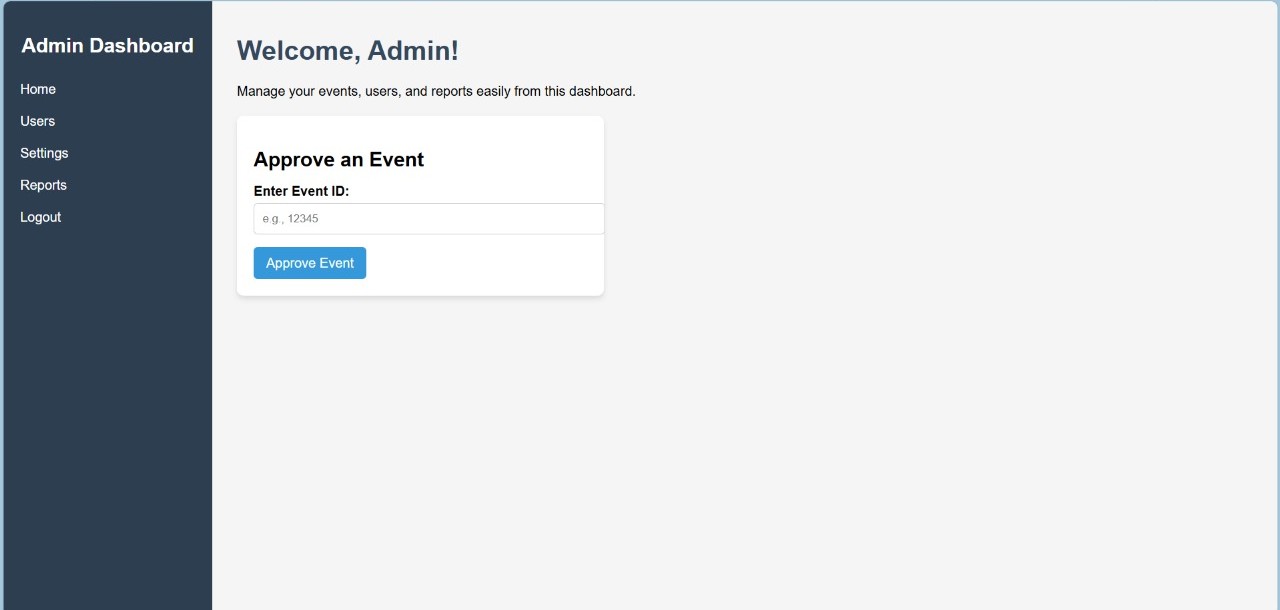
Description automatically generated

1. **Event Registration**
   * Organizers can register and manage events they have created.
   * Attendees can search for and view events and register for them.
   * Events are listed based on the role of the user (Organizers see their events, and Attendees see all events).

A screenshot of a calendar

Description automatically generated

1. **Admin Authentication & Authorization (Spring Security)**
   * Use of **Spring Security** to implement authentication and authorization.
   * Restrict event creation and modification to organizers only, and event viewing to both attendees and organizers.



**4. Technologies Used**

* **Spring Boot**: Backend framework for building REST APIs and managing business logic.
* **Spring Security**: Provides authentication and authorization to ensure that only authorized users can create or manage events.
* **JPA/Hibernate**: Data persistence layer to interact with MySQL or other relational databases.
* **JSP (JavaServer Pages)**: Server-side rendering technology to generate dynamic web pages.
* **AJAX/JavaScript**: Used for asynchronous data fetching, especially for listing events and submitting forms without reloading the page.
* **MySQL**: Relational database to store user and event data.

**5. System Architecture**

The system follows a **Layered Architecture**:

1. **Presentation Layer** (Web Layer):
   * Consists of **JSP** pages that interact with the backend.
   * The frontend communicates with the backend using **AJAX** for dynamic content loading and form submissions.
2. **Service Layer** (Business Logic):
   * This layer contains the core business logic related to event creation, registration, and user authentication.
   * Service classes like EventService, UserService manage the flow of data between the controller and the database.
3. **Data Layer** (Persistence Layer):
   * The **EventRepository** and **UserRepository** interfaces are used to interact with the database using **JPA/Hibernate** for persistence.
4. **Security Layer**:
   * Configures **Spring Security** to protect routes and ensure users are authenticated and authorized for specific actions.

**6. Database Schema**

The system uses two primary entities: **User** and **Event**.

* **User Table**:
  + id (Primary Key)
  + username
  + email
  + password (hashed using BCrypt)
  + phoneNumber
  + role (either "Organizer" or "Attendee")
* **Event Table**:
  + id (Primary Key)
  + name
  + date
  + time
  + venue
  + ticketPrice
  + organizerId (Foreign Key referencing User.id)

**7. User Roles**

* **Organizer**:
  + Can create, update, and delete events.
  + Can view events they have created.
* **Attendee**:
  + Can view all upcoming events.
  + Can register for events (optional functionality to add).

A screenshot of a computer

Description automatically generated

**8. Flow of Operations**

1. **User Registration & Authentication**:
   * When a new user signs up, their information is stored in the User table.
   * The password is securely hashed using **BCrypt**.
   * The login page authenticates the user based on their username and password.
2. **Event Creation (Organizer)**:
   * After logging in as an organizer, users can create new events.
   * Event details (name, date, time, venue, ticket price) are saved in the Event table, and the event is linked to the organizer.
3. **Viewing Events (Attendee and Organizer)**:
   * Both organizers and attendees can view events.
   * The event data is fetched from the database and displayed dynamically using AJAX.
4. **Event Registration**:
   * Attendees can register for events (optional feature). This can be done by storing attendee registration information in a separate table.

**9. Security Measures**

* **Password Hashing**: Passwords are hashed using **BCrypt** to ensure secure authentication.
* **Authorization**: Organizers are granted permissions to create and manage events, while attendees can only view events.
* **Session Management**: Spring Security manages user sessions and protects endpoints from unauthorized access.

**10. Testing**

* **JUnit** tests are written for:
  + Testing service methods like signup, login, and event management operations.
  + Testing repository methods using **@DataJpaTest**.
  + **MockMvc** is used for testing controller layer APIs and ensuring proper HTTP responses.

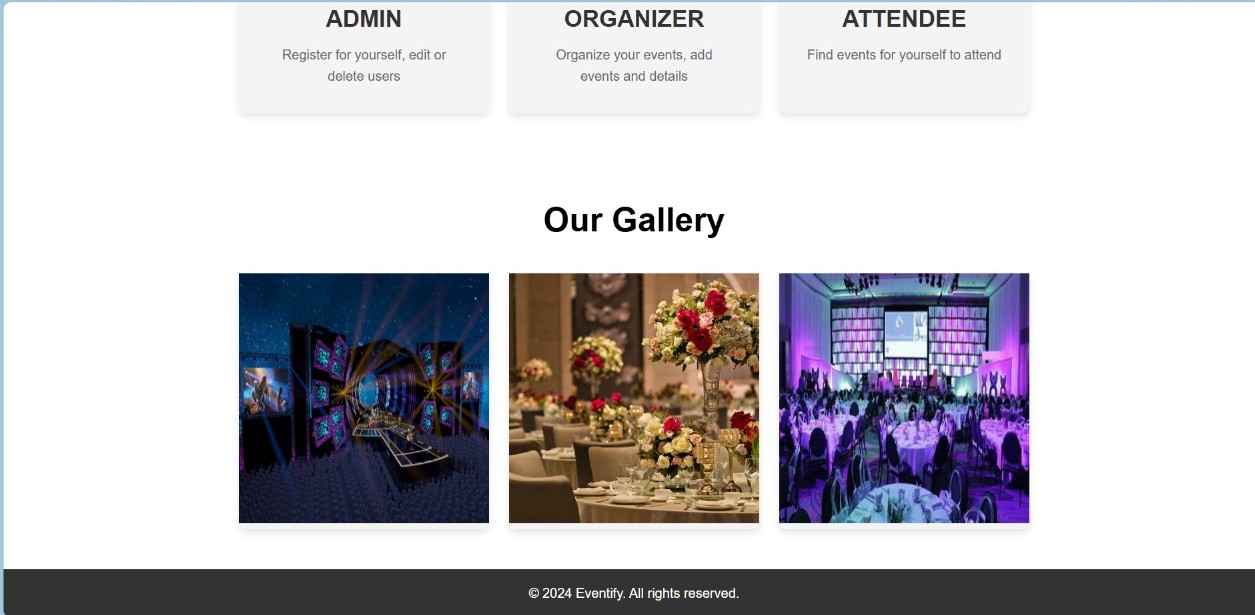
**11. Deployment**

* The project can be deployed on any servlet container such as **Apache Tomcat**.
* The **WAR** file is generated via **Maven**, and the event-management.war file is placed in the webapps directory of Tomcat.

**12. Future Enhancements**

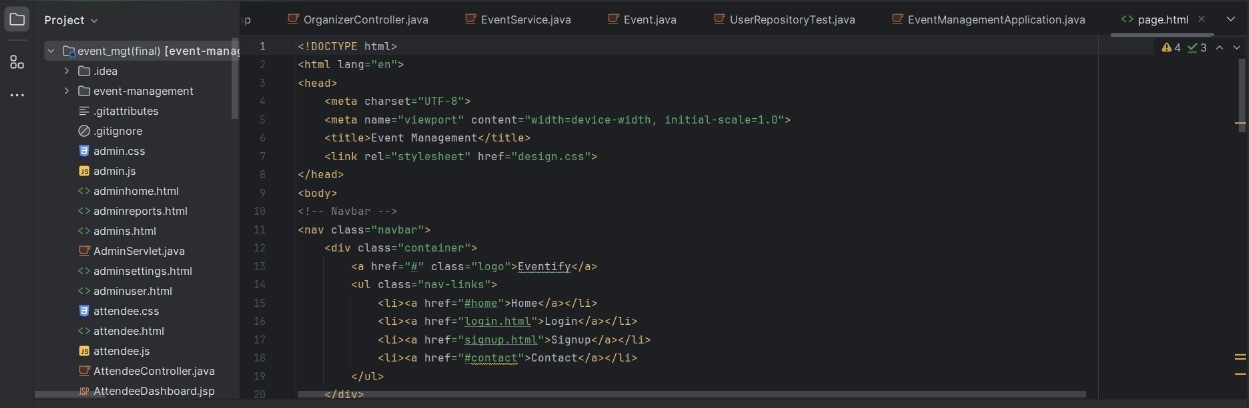
1. **Payment Integration**: Integrating payment gateways for event registration fees.
2. **Event Notifications**: Sending email/SMS notifications to users about event updates and registrations.
3. **Search and Filter**: Adding advanced search and filter options for events (by date, location, price range, etc.).

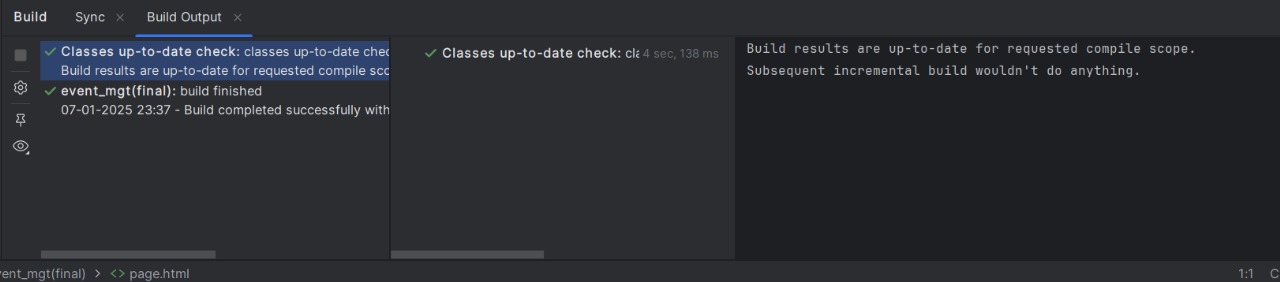
**13. Conclusion**

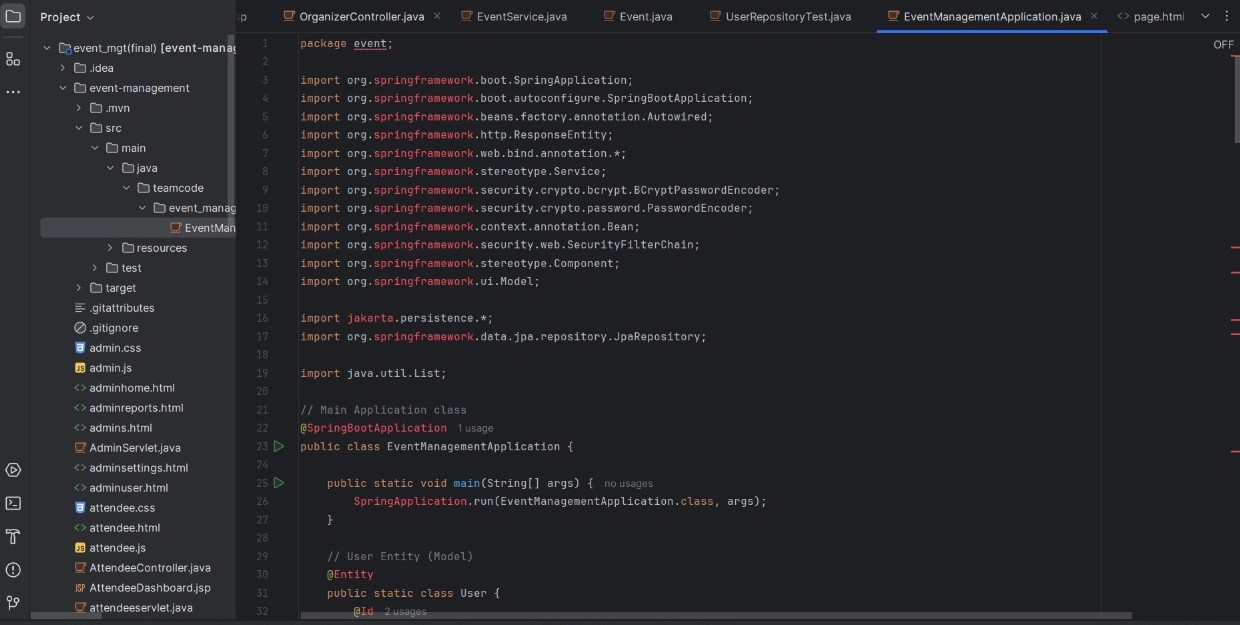
****

The Event Management System is a powerful tool that automates event management processes, streamlining event creation, registration, and participation. The system is modular, scalable, and easily extendable with features like payment integration and enhanced user interaction.

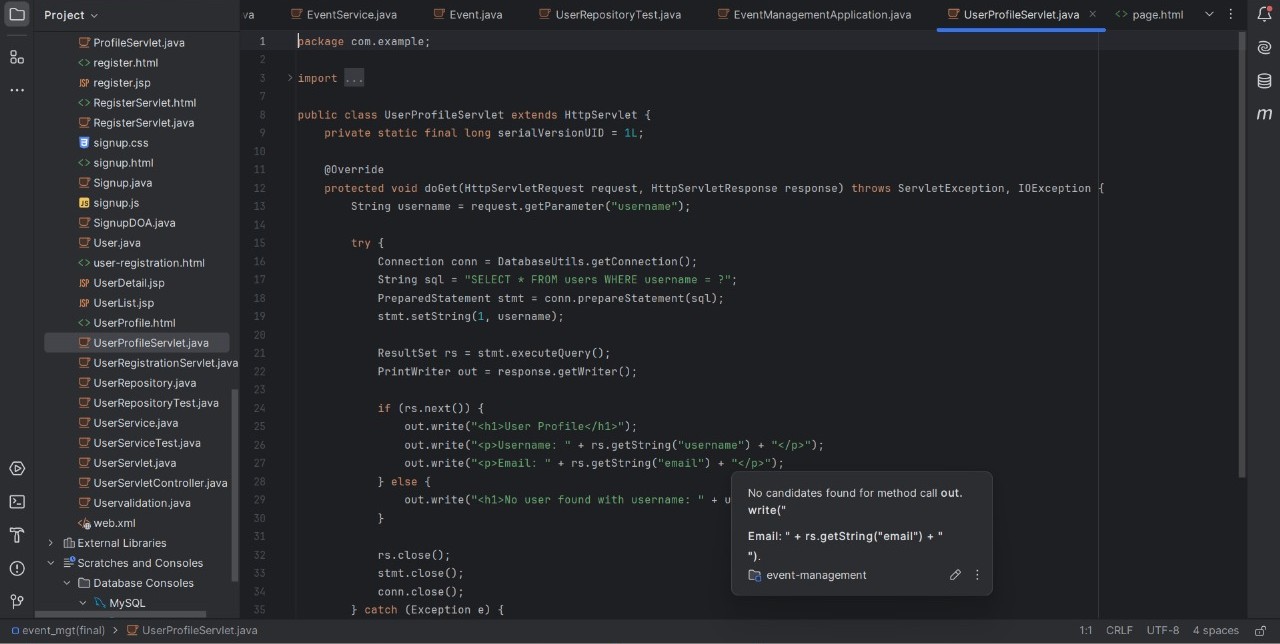
This system utilizes **Spring Boot** and **JSP** technologies to create a robust, dynamic, and secure platform for event organizers and attendees.

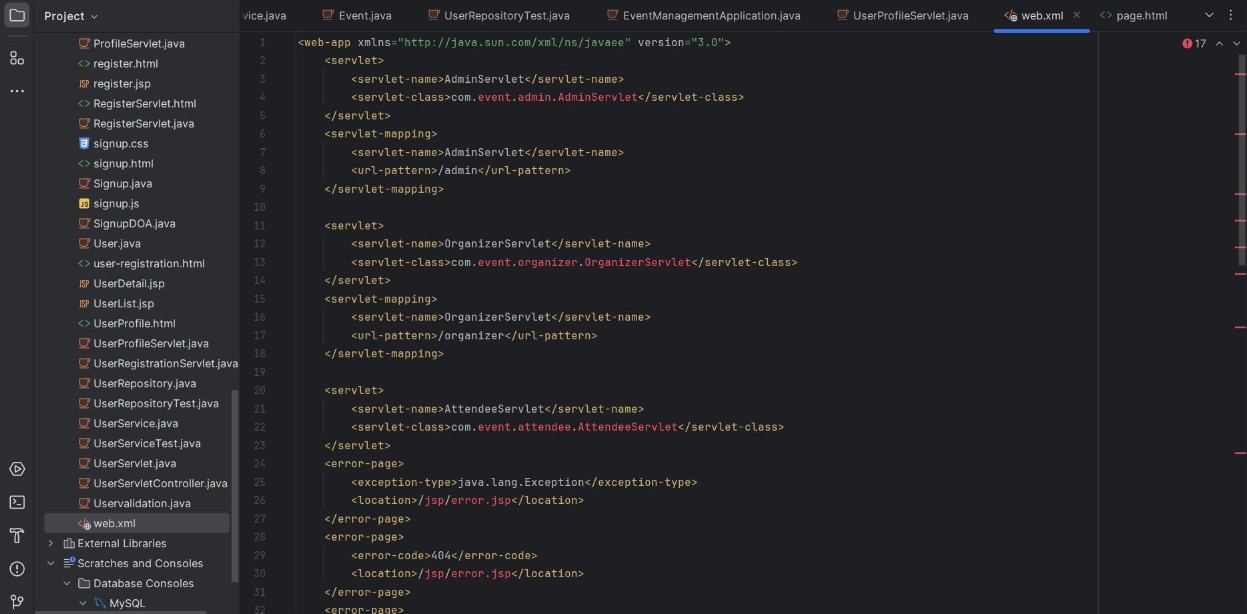


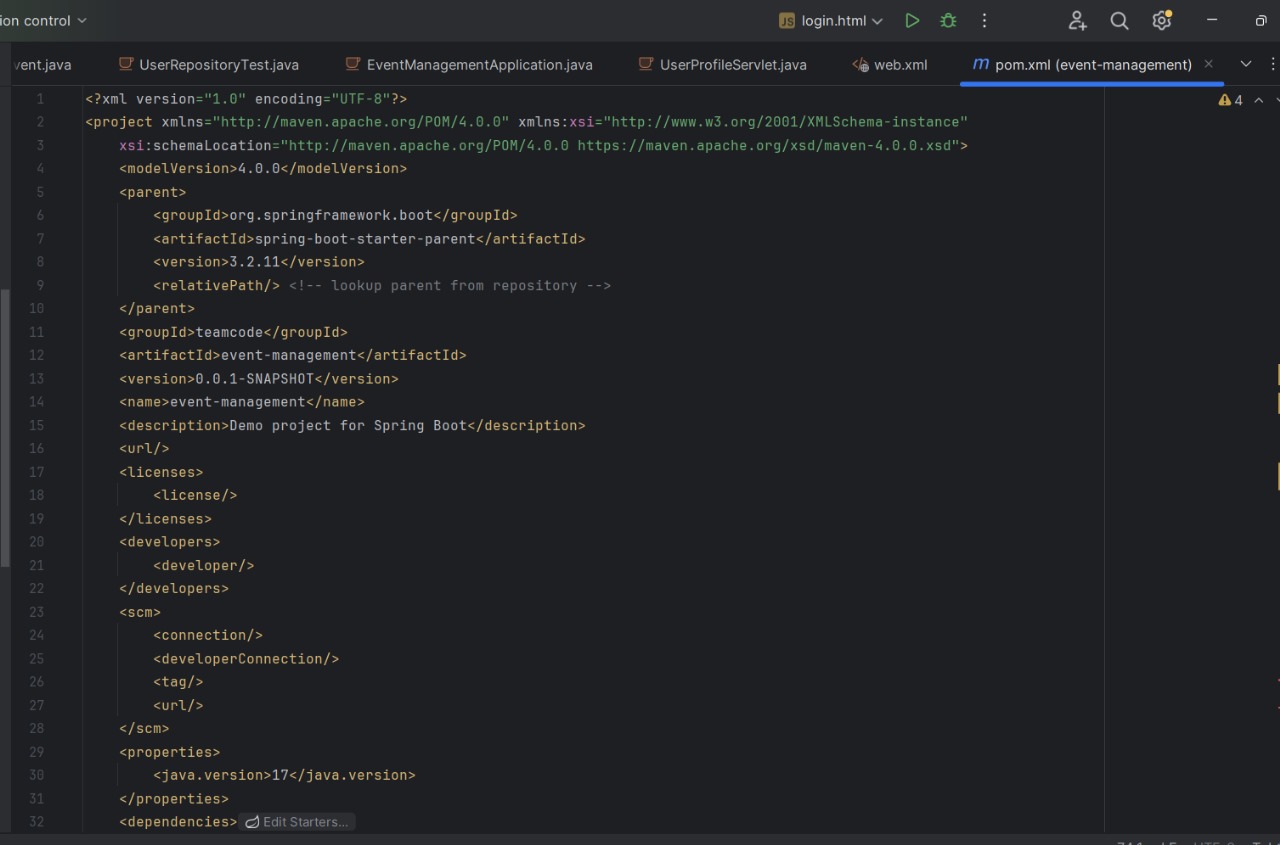


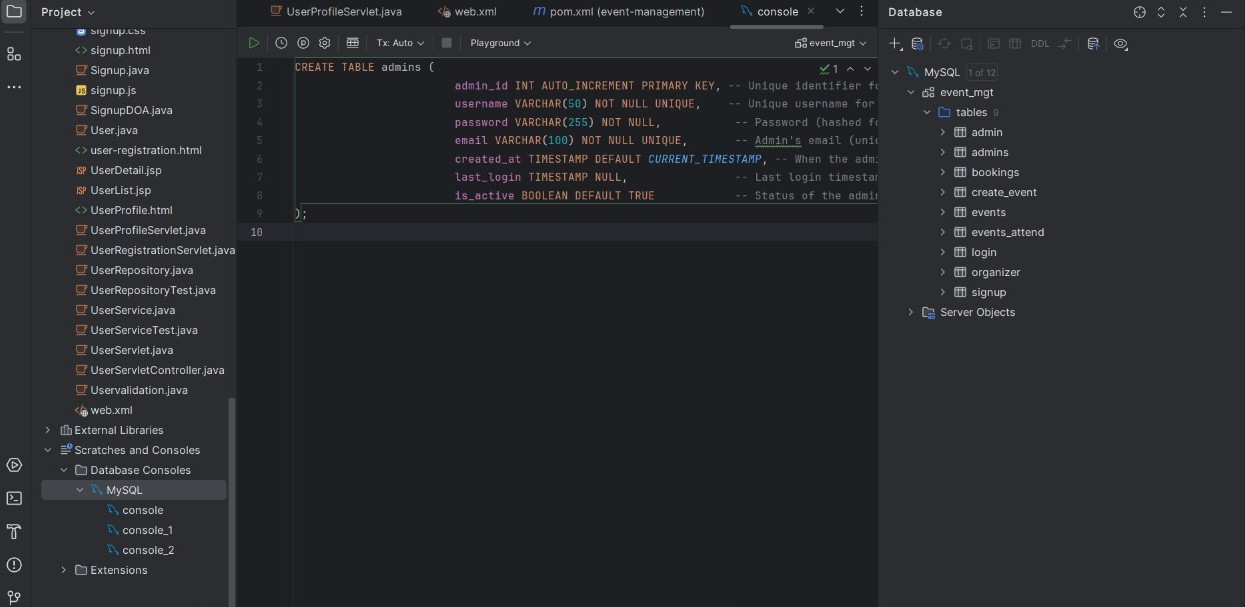












Bottom of Form