"EVENT & MOVIE TICKET BOOKING SYSTEM USING SPRINGBOOT FRAMEWORK OF JAVA"

A MINOR PROJECT-I
Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING-DATA SCIENCE
SUBMITTED TO

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)

SUBMITTED BY

- **Deeksha Saini (0176CS211051)**
- **Esha Gandhi (0176CS211056)**
- **Kratika Kathal (0176CS211082)**

UNDER THE SUPERVISION OF
Prof. Ashish Chaturvedi
Department of Computer Science

Department of Computer Science Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)

December 2023

Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CANDIDATE'S DECLARATION

We, Deeksha Saini, Esha Gandhi, Kratika Kathal, Student of Bachelor of Technology, Computer Science, Lakshmi Narain College of Technology Excellence, Bhopal session 2023-24 hereby declare that the work presented in the Minor Project-I entitled "EVENT & MOVIE TICKET BOOKING SYSTEM USING SPRINGBOOT FRAMEWORK OF JAVA" is outcome of our own bonafide work, which is correct to the best of my/our knowledge and this work has been carried out taking care of Engineering Ethics. The work presented does not infringe any previous work and has not been submitted to any University for the award of any degree / diploma.

We also declare that "A check for plagiarism has been carried out on the Minor Project-I and is found within the acceptable limit and report of which is enclosed herewith".

Deeksha Saini (0176CS211051)
Esha Gandhi(0176CS211056)
Kratika Kathal(0176CS211082)

Lakshmi Narain College of Technology Excellence, Bhopal

Department of Computer Science & Engineering-Data Science

CERTIFICATE

This is to certify that the work embodies in this Minor Project-I entitled "Gym Management System (Gymrats)" being submitted by **Deeksha Saini (0176CS211051), Esha Gandhi (0176CS211056), Kratika Kathal(0176CS211082)** for partial fulfillment of the requirement for the award ofdegree of "**Bachelor of Technology** in **Computer Science**" discipline to "**RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL (M.P.)**" during the academic year 2023-24 is a record of real piece of work, carried out by him/her/them under my supervision and guidance in the **"Department of Computer Science", Lakshmi Narain College of Technology Excellence, Bhopal (M.P.)**

SUPERVISED BY

Prof. Ashish Chaturvedi

(Department of Cse)

LNCTE,Bhopal

FORWARDED BY

Prof. Nandita Tiwari

(Head, Department of CSE)

LNCTE,Bhopal

APPROVED BY

Dr. Anil Saxena
(Principal)
LNCTE,Bhopal

ACKNOWLEDGEMENT

At the outset, we would like to link to thank the Almighty who made all the things possible. Writing this project report would not have been possible without the support of several people whom we need to wholeheartedly thank. We express a deep sense of gratitude to my Supervisor " Prof. Ashish Chaturvedi", Dept. of CSE for the valuable and inspirational guidance from the initial to the final level that enabled me to develop an understanding of this Project work.

We would like to give my sincere thanks to Prof.(Dr.) Megha Kamble, Head, Dept. of CSE for their kind help, encouragement and co-operation throughout my Project period and We owe my special thanks to our Principal Dr. Anil Kumar Saxena for their guidance and suggestions during the Project work.

Lastly, We want to thank my parents, friends and to all those people who had contributed to my project directly or indirectly for their moral and psychological support.

Deeksha Saini (0176CS211051)
Esha Gandhi (0176CS211056)
Kratika Kathal (0176CS211082)

Table of Contents

ABSTRACT

The Event & Movie Ticket Booking System is an innovative web-based application developed using the Spring Boot framework in Java. This project addresses the challenges associated with conventional ticket booking methods by providing a centralized platform for users to seamlessly discover, select, and reserve tickets for a diverse array of events and movies. The system boasts a user-friendly interface, real-time seat availability updates, and secure payment processing, aiming to redefine and elevate the user experience in the realm of event and movie ticketing. By leveraging cutting-edge technologies, including Java, Spring Boot, this system ensures efficient and reliable functionality, ultimately delivering a streamlined and convenient ticket booking solution to meet the demands of modern digital consumers.

1. Introduction

1.1 Project Overview

In today's fast-paced digital age, the demand for efficient and user-friendly systems for event and movie ticket booking is on the rise. The Event & Movie Ticket Booking System, developed using the Spring Boot framework in Java, seeks to address the challenges inherent in traditional ticketing methods by providing a modern, centralized platform for users.

1.2 Objectives

The primary objectives of the project are as follows:

- **Simplify Ticket Booking:** Create a user-centric system that simplifies the process of discovering, selecting, and booking tickets for various events and movies.
- **Enhance User Experience:** Develop a user-friendly interface that ensures a seamless and enjoyable experience for users at every stage of the ticketing process.
- **Real-time Updates:** Implement real-time seat availability updates to provide users with accurate information during the ticket selection process.
- **Secure Transactions:** Integrate a secure payment gateway to facilitate safe and reliable financial transactions for ticket purchases.

1.3 Project Scope

The scope of this project includes the development of a robust and scalable ticket booking system that allows users to:

- Browse and search for events and movies.
- View details such as show timings, venue information, and pricing.
- Select seats and purchase tickets securely.
- Receive confirmation and electronic tickets via email.

- Provide feedback and ratings for events and movies.

1.4 Problem Statement

Traditional ticket booking methods often pose challenges such as long queues, limited accessibility, and a lack of real-time information for users. This project addresses these issues by leveraging modern technologies to create a centralized, efficient, and secure Event & Movie Ticket Booking System. The aim is to overcome the drawbacks of conventional ticketing approaches, offering users a convenient and streamlined solution for accessing and reserving tickets for a diverse range of events and movies. By implementing this system, we seek to revolutionize the ticket booking experience, making it more accessible, user-friendly, and responsive to the dynamic needs of both event organizers and attendees. The project aligns with the broader trend of digital transformation in the entertainment industry, providing a technologically advanced solution to meet the evolving expectations of users in the domain of event and movie ticketing.

2. Literary Survey

2.1 Previous Work

Various online ticket booking systems have been developed to enhance the convenience of users in accessing tickets for events and movies. Solutions such as Ticketmaster, BookMyShow, and Eventbrite have gained popularity for their comprehensive platforms. Examining existing systems provides insights into their strengths, such as user-friendly interfaces and widespread availability. However, limitations such as complex navigation and potential security concerns highlight areas for improvement in our proposed system.

2.2 Our Proposal

Exploring ticketing systems developed using Java provides an understanding of the advantages of utilizing Java-based frameworks, including Spring Boot. Analyzing successful implementations helps in shaping the architecture and functionalities of our project. Investigating projects leveraging Spring Boot reveals its role in enhancing development speed, simplifying configuration, and facilitating the creation of robust, scalable applications.

2.3 System Architecture

The system follows a three-tier architecture comprising a presentation layer, application layer (Spring Boot), and a database layer. The presentation layer handles the user interface, the application layer manages business logic, and the database layer stores and retrieves relevant information.

3. System Study

3.1 Existing System

3.1.1 Overview

The existing landscape of event and movie ticket booking systems encompasses platforms such as Ticketmaster, BookMyShow, and Eventbrite. These systems provide users with the ability to discover, select, and purchase tickets for a wide array of events and movies.

3.1.2 Limitations

- **Complex User Interfaces:** Some existing systems feature complex and non-intuitive user interfaces, potentially leading to user dissatisfaction and errors during the booking process.
- **Limited Real-time Updates:** The real-time availability of seats is not consistently provided, leading to potential discrepancies during the ticket selection process.
- **Security Concerns:** Certain systems exhibit vulnerabilities in financial transaction security, which can undermine user trust in the platform.

3.2 Proposed System

3.2.1 Overview

The proposed Event & Movie Ticket Booking System aims to address the limitations of existing systems by leveraging the Spring Boot framework in Java. This system intends to provide an intuitive, secure, and efficient platform for users to seamlessly navigate, select, and book tickets.

3.2.2 Intended Objectives

- **Simplified User Interface:** Develop an interface that is user-friendly and easy to navigate, enhancing the overall user experience.
- **Real-time Seat Availability Updates:** Implement real-time updates on seat availability to provide users with accurate and up-to-date information during the ticket selection process.
- **Secure Payment Processing:** Integrate a robust and secure payment gateway to ensure the safety and reliability of financial transactions.

3.3 Feasibility Study

3.3.1 Technical Feasibility

- **Spring Boot Framework:** Leveraging Spring Boot ensures rapid development, simplifies configuration, and provides a foundation for building a scalable and maintainable system.
- **Java Technology Stack:** The use of Java, a widely adopted and versatile programming language, ensures compatibility, ease of development, and community support.

3.3.2 Operational Feasibility

- **User Training:** The system is designed to be intuitive, reducing the need for extensive user training and making it accessible to users with varying levels of technical proficiency.
- **Real-time Updates:** Operational efficiency is enhanced through real-time seat availability updates, reducing the likelihood of user-related errors and improving overall system responsiveness.

3.3.3 Economic Feasibility

- **Cost-Effective Development:** Utilizing open-source technologies, including the Spring Boot framework, contributes to cost-effectiveness in development.
- **Revenue Generation:** The system's potential to generate revenue through ticket sales makes it economically viable, justifying the initial investment.

4 System Analysis

4.1 Functional Requirement

The functional requirements of the Event & Movie Ticket Booking System outline the specific capabilities and features that the system must possess:

- **User Registration and Authentication:** Users should be able to register for an account. Authenticated users can log in to the system.
- **Event and Movie Listings:** Display a comprehensive list of upcoming events and movies. Provide search and filter options for users to find specific events or movies.
- **Ticket Booking:** Allow users to select seats, specify the number of tickets, and complete the booking process. Provide real-time updates on seat availability.
- **Payment Gateway Integration:** Integrate a secure payment gateway for processing transactions.
- **Booking Confirmation and Tickets:** Send confirmation emails to users with electronic tickets.
- **User Feedback and Ratings:** Allow users to provide feedback and ratings for attended events and movies.

4.2 Non-functional Requirement

Non-functional requirements highlight the qualities and characteristics the system must possess:

- **Performance:** The system should handle a concurrent user load without significant performance degradation.
- **Security:** Implement secure authentication and payment processing to protect user data and financial transactions.

- **Usability:** Ensure an intuitive and user-friendly interface for a positive user experience.
- **Scalability:** Design the system to handle a growing number of events, movies, and users.

4.3 Requirement Specification

4.3.1 Software Requirement Specification

The software requirements for the system include:

- VS Code for code implementation
- Java Development Kit (JDK)
- Spring Boot Framework
- Thymeleaf for server-side templating
- H2 Database for data storage
- HTML, CSS, and JavaScript for the front-end development

4.3.2 Implementation Tool & Language

The system is implemented using the following tools and languages:

- Development Framework: Spring Boot
- Programming Language: Java
- Database: H2
- Front-end: HTML, CSS, JavaScript
- Version Control: Git

4.4 System Flowchart

A system flowchart illustrates the high-level flow of processes and interactions within the Event & Movie Ticket Booking System. It provides a visual representation of the main stages from user registration to ticket booking and confirmation.

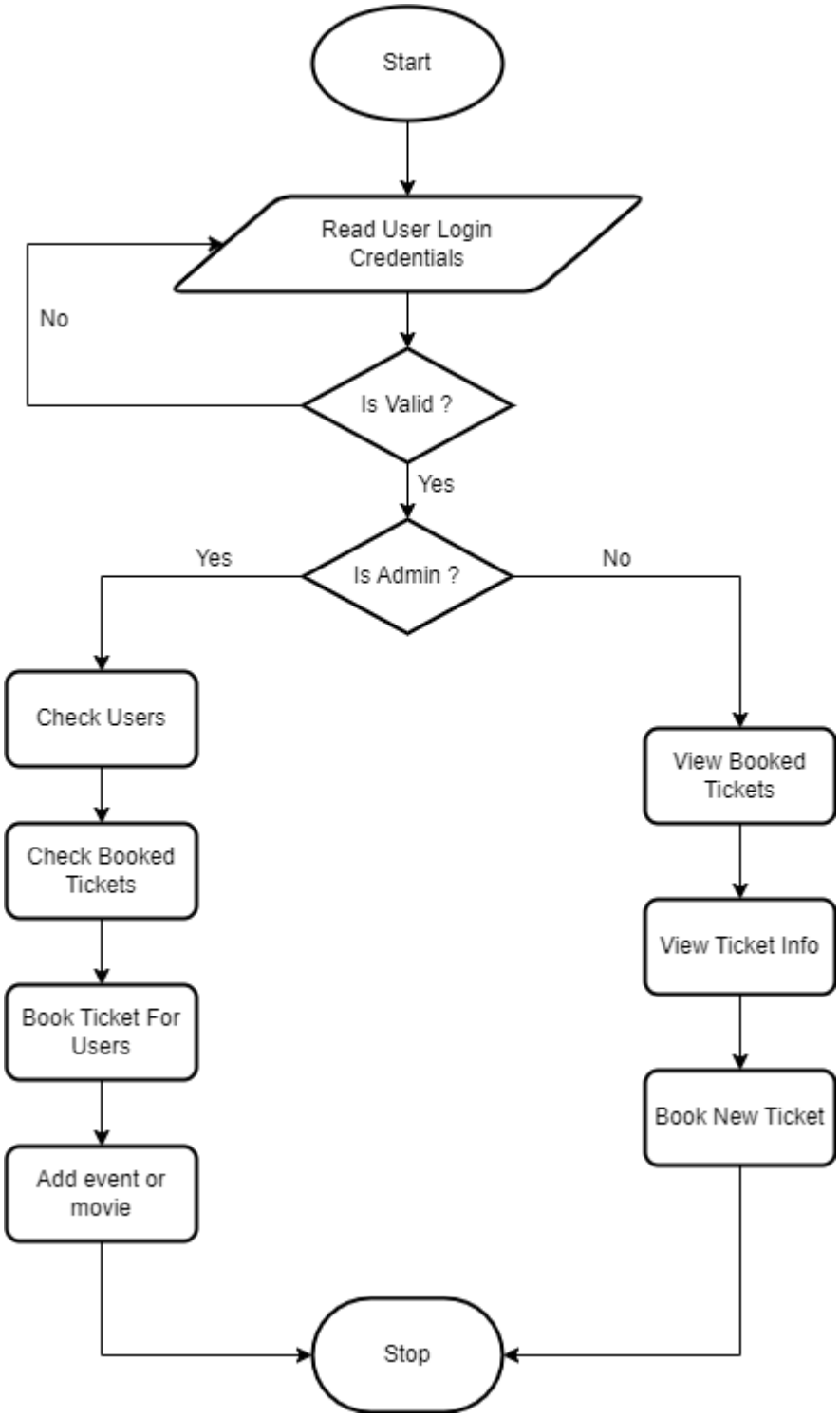


Fig. Flowchart

4.5 DFD (Data Flow Diagram)

4.5.1 Level 0 DFD

The Level 0 DFD provides an overview of the entire system, depicting major processes and data flows between them.

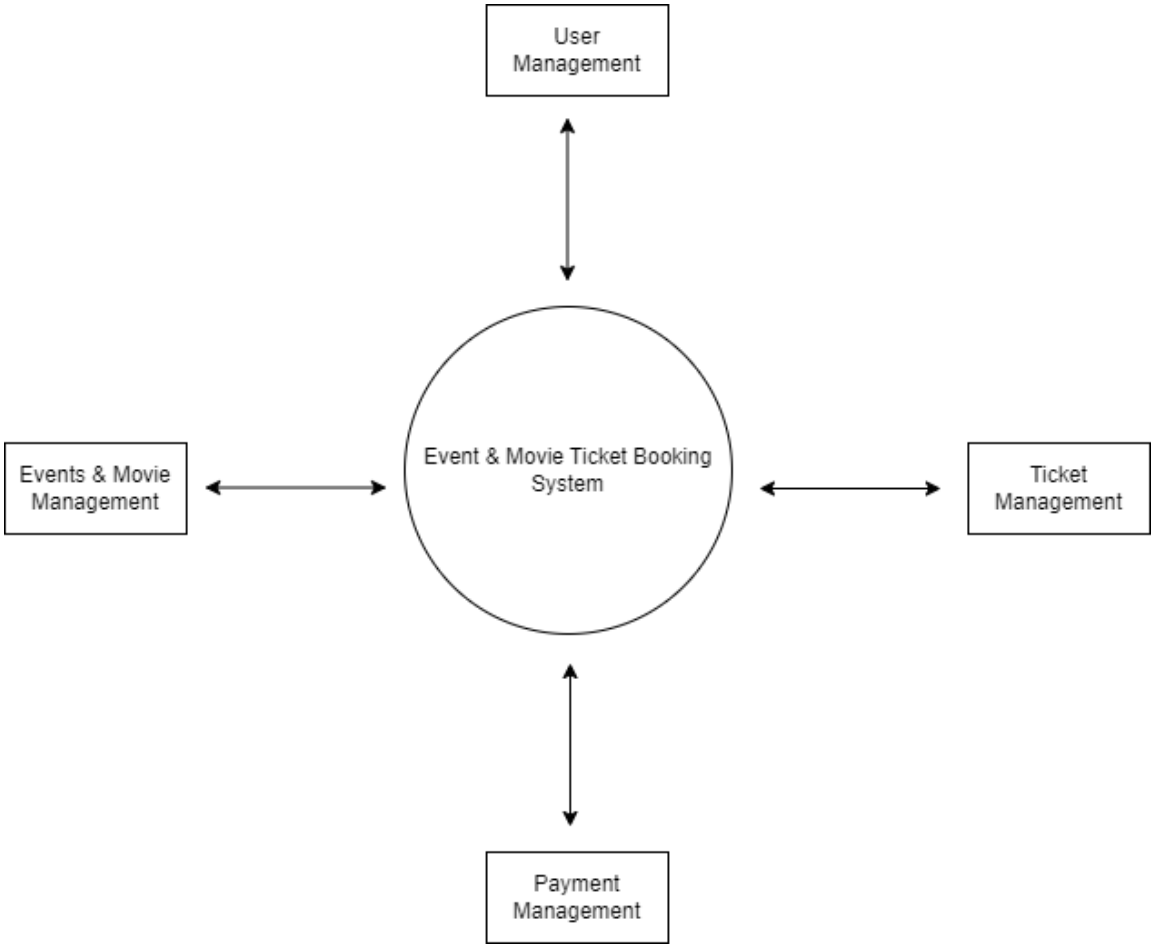


Fig. Level 0 DFD

4.5.2 Level 1 DFD

The Level 1 DFD delves into more detail, breaking down each major process into sub-processes and illustrating data flows between them.

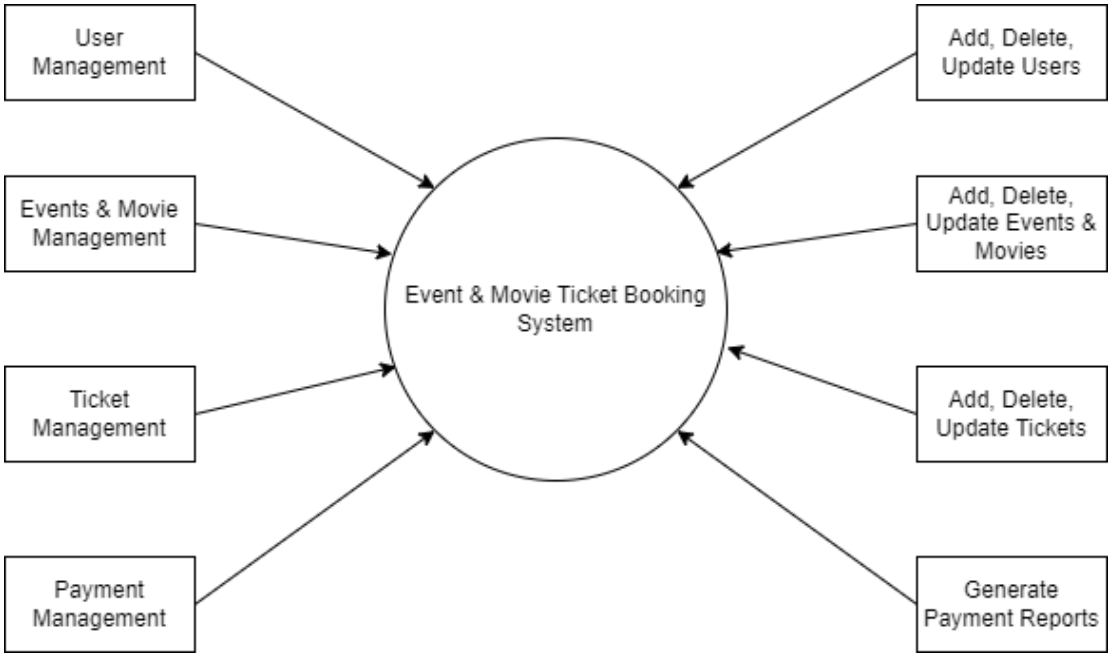


Fig. Level 1 DFD

4.6 System Design

The system design encompasses the overall architecture, including components, modules, and their interactions. It involves detailed technical specifications for each aspect of the system, from databases to user interfaces.

4.7 System Testing

System testing involves evaluating the system's functionality to ensure that it meets the specified requirements. This includes unit testing, integration testing, and system testing to identify and rectify any issues.

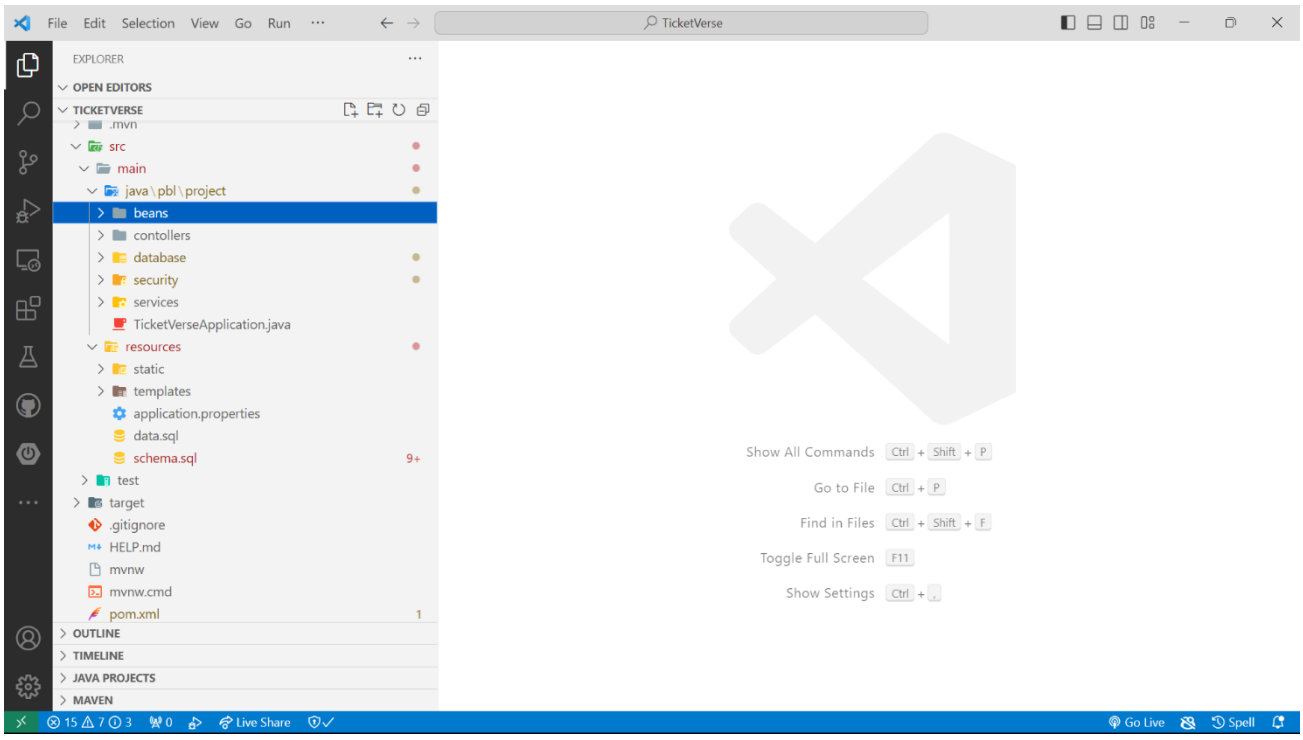
- **Unit Testing:** Utilized JUnit for testing individual components and methods.
- **Integration Testing:** Conducted integration tests to ensure the seamless collaboration of different modules.

- **Test Results:** All unit and integration tests passed successfully, ensuring the reliability and functionality of the system.

4.8 System Implementation

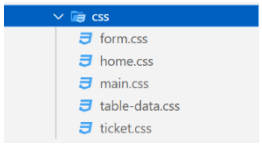
The system is implemented using the Spring Boot framework, following the Model-View-Controller (MVC) architectural pattern. The codebase is organized into modules, including controllers, services, repositories, and views. It involves translating the design into a working system. This phase includes coding, database creation, and integrating various components to create a functional Event & Movie Ticket Booking System.

4.8.1 Project Folder Structure:



4.8.2 HTML Files:

4.8.3 CSS Files:



4.8.4 Beans:

4.8.4.1 User:

User.java

src > main > java > pbl > project > beans > User.java > User > user_name

```java
package pbl.project.beans;

import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@NoArgsConstructor
@RequiredArgsConstructor
public class User {
    private int user_id;
    @NonNull
    private String user_name;
    @NonNull
    private String password;
    @NonNull
    private String user_email;
    @NonNull
    private Long user_contact;
    @NonNull
    private String user_role;

    public int getUser_id() {
        return user_id;
    }

    public String getUser_name() {
        return user_name;
    }

    public String getPassword() {
        return password;
    }

    public String getUser_email() {
        return user_email;
    }

    public Long getUser_contact() {
        return user_contact;
    }

    public String getUser_role() {
        return user_role;
    }

}
```

15 7 3 0 Live Share Ln 14, Col 30 Spaces: 4 UTF-8 CRLF Java Go Live Spell Prettier

4.8.4.2 Ticket:

Ticket.java

src > main > java > pbl > project > beans > Ticket.java > Ticket > booking_reference

```java
package pbl.project.beans;

public class Ticket {
    private int ticket_id;

    private String event_name;
    private String event_date;
    private String venue_name;
    private String ticket_holder_name;
    private Long ticket_holder_contact;
    private String ticket_type;
    private String booking_reference;

    private double ticket_price;

    public int getTicket_id() {
        return ticket_id;
    }

    public void setTicket_id(int ticket_id) {
        this.ticket_id = ticket_id;
    }

    public String getEvent_name() {
        return event_name;
    }

    public void setEvent_name(String event_name) {
        this.event_name = event_name;
    }

    public String getEvent_date() {
        return event_date;
    }

    public void setEvent_date(String event_date) {
        this.event_date = event_date;
    }

    public String getVenue_name() {
        return venue_name;
    }

    public void setVenue_name(String venue_name) {
        this.venue_name = venue_name;
    }

    public String getTicket_holder_name() {
        return ticket_holder_name;
    }

    public void setTicket_holder_name(String ticket_holder_name) {
        this.ticket_holder_name = ticket_holder_name;
    }

    public Long getTicket_holder_contact() {
        return ticket_holder_contact;
    }

    public void setTicket_holder_contact(Long ticket_holder_contact) {
        this.ticket_holder_contact = ticket_holder_contact;
    }

    public String getTicket_type() {
        return ticket_type;
    }

    public void setTicket_type(String ticket_type) {
        this.ticket_type = ticket_type;
    }

    public String getBooking_reference() {
        return booking_reference;
    }

    public void setBooking_reference(String booking_reference) {
        this.booking_reference = booking_reference;
    }

    public double getTicket_price() {
        return ticket_price;
    }

    public void setTicket_price(double ticket_price) {
        this.ticket_price = ticket_price;
    }

}
```

4.8.5 Controllers:

MainController.java

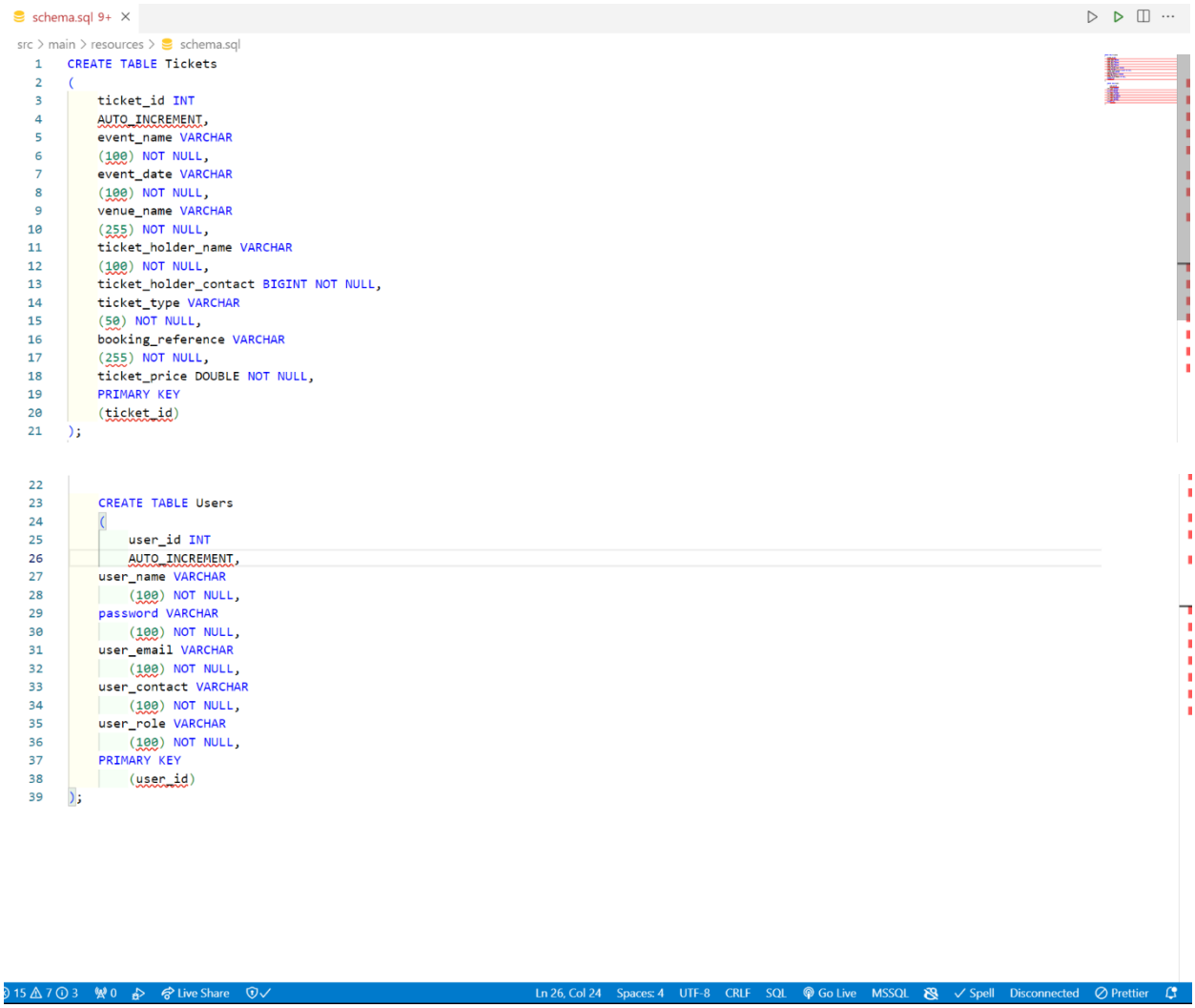
src > main > java > pbl > project > contollers > MainController.java > ...

```java
package pbl.project.contollers;

import java.security.Principal;
import java.text.DecimalFormat;
import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import pbl.project.beans.Ticket;
import pbl.project.beans.User;
import pbl.project.database.DatabaseOperation;

@Controller
public class MainController {
    @Autowired
    @Lazy
    private DatabaseOperation databaseOperation;

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/login")
    public String login() {
        if (SecurityContextHolder.getContext().getAuthentication().getPrincipal() instanceof UserDetails) {
            return "redirect:/";
        }
        return "login";
    }

    @GetMapping("/registration")
    public String registration() {
        if (SecurityContextHolder.getContext().getAuthentication().getPrincipal() instanceof UserDetails) {
            return "redirect:/";
        }
        return "registration";
    }

    @PostMapping("/registration")
    public String register_user(@RequestParam String username, @RequestParam String password,
            @RequestParam String user_email, @RequestParam Long user_contact) {
        User user = databaseOperation.get_user_info(username);
        if (user == null) {
            databaseOperation.add_user(username, password, user_email, user_contact);
            return "redirect:/login?registered";
        }
        return "redirect:/registration?error";
    }

    @GetMapping("/display_ticket/{ticket_id}")
    public String display_ticket(@PathVariable int ticket_id, Model model) {
        model.addAttribute(attributeName:"Ticket", databaseOperation.get_ticket_by_id(ticket_id));
        return "display_ticket";
    }

    @GetMapping("/create_ticket")
    public String create_ticket(Ticket ticket, Model model) {
        model.addAttribute(attributeName:"ticket", ticket);
        ArrayList<User> user_list = databaseOperation.get_all_users();
        model.addAttribute(attributeName:"user_list", user_list);
        return "create_ticket";
    }

    @PostMapping("/add_ticket")
    public String add_ticket(@ModelAttribute Ticket ticket, BindingResult bindingResult) {
        Ticket ticket_to_add = databaseOperation.get_ticket_by_event_name(ticket.getEvent_name());
        if (ticket_to_add == null) {
            databaseOperation.add_ticket(ticket);
            return "redirect:/ticket_list?message";
        }
        return "redirect:/add_ticket?error";
    }

}
```

```java
    @GetMapping("/ticket_list")
    public String ticket_list(Model model, Principal principal) {
        String username = principal.getName();
        User user = databaseOperation.get_user_info(username);
        ArrayList<Ticket> ticket_list = databaseOperation.get_all_tickets(user);

        DecimalFormat df = new DecimalFormat(pattern:"0.0");

        double sub_total = ticket_list.stream().mapToDouble(Ticket::getTicket_price).sum();

        double tax = sub_total * (13.00 / 100.00);

        double total = sub_total + tax;

        sub_total = Double.parseDouble(df.format(sub_total));
        tax = Double.parseDouble(df.format(tax));
        total = Double.parseDouble(df.format(total));

        model.addAttribute(attributeName:"tickets_list", ticket_list);
        model.addAttribute(attributeName:"sub_total", sub_total);
        model.addAttribute(attributeName:"tax", tax);
        model.addAttribute(attributeName:"total", total);

        return "ticket_list";
    }

    @GetMapping("/user_list")
    public String user_list(Model model) {
        ArrayList<User> user_list = databaseOperation.get_all_users();
        model.addAttribute(attributeName:"user_list", user_list);
        return "user_list";
    }

    @GetMapping("/config_ticket/{ticket_id}")
    public String config_ticket(@PathVariable int ticket_id, Model model) {
        model.addAttribute(attributeName:"Ticket", databaseOperation.get_ticket_by_id(ticket_id));
        return "config_ticket";
    }

    @PostMapping("/update_ticket")
    public String update_ticket(@ModelAttribute Ticket ticket, RedirectAttributes redirectAttributes) {
        databaseOperation.update_ticket(ticket);
        redirectAttributes.addFlashAttribute(attributeName:"message", attributeValue:"Successfully Updated !!");
        return "redirect:/ticket_list";
    }

    @GetMapping("/delete_ticket/{ticket_id}")
    public String delete_ticket(@PathVariable int ticket_id, RedirectAttributes redirectAttributes) {
        databaseOperation.delete_ticket(ticket_id);
        redirectAttributes.addFlashAttribute(attributeName:"message", attributeValue:"Successfully Deleted !!");
        return "redirect:/ticket_list";
    }
}
```

4.8.6 Database:

4.8.6.1 Schema:



4.8.6.2 Data:

data.sql

src > main > resources > data.sql

```sql
INSERT INTO Tickets
    (event_name, event_date, venue_name, ticket_holder_name, ticket_holder_contact, ticket_type, booking_reference, ticket_price)
VALUES
    ('Concert in the Park', '2023-08-15', 'Central Park', 'Tanish', 9358174158, 'GENERAL', 'https://dummy.com/concert', 50.00),
    ('Movie Night', '2023-08-20', 'Cinema City', 'Utkarsh', 19876543211, 'STANDARD', 'https://dummy.com/movies', 12.50),
    ('Tech Conference', '2023-09-05', 'Convention Center', 'Tanish', 19876543212, 'VIP', 'https://dummy.com/conference', 250.00),
    ('Art Exhibition', '2023-09-10', 'Gallery Hall', 'Utkarsh', 19876543213, 'GENERAL', 'https://dummy.com/art', 20.00),
    ('Live Comedy Show', '2023-09-18', 'Laugh Factory', 'Yogesh', 13335557777, 'PREMIUM', 'https://dummy.com/comedy', 35.00),
    ('Sports Game', '2023-09-25', 'Stadium Arena', 'Tanish', 19876543210, 'VIP', 'https://dummy.com/sports', 75.00),
    ('Fashion Runway', '2023-10-05', 'Fashion Pavilion', 'Ujjwal', 19876543211, 'GENERAL', 'https://dummy.com/fashion', 30.00),
    ('Music Festival', '2023-10-12', 'Festival Grounds', 'Yogesh', 17894561230, 'STANDARD', 'https://dummy.com/music', 65.00),
    ('Science Expo', '2023-11-03', 'Science Center', 'Utkarsh', 19876543213, 'GENERAL', 'https://dummy.com/science', 15.00),
    ('Dance Performance', '2023-11-15', 'City Theater', 'Ujjwal', 13335557777, 'PREMIUM', 'https://dummy.com/dance', 40.00);

INSERT INTO Users
    (user_name, password, user_email, user_contact, user_role)
VALUES
    ('Tanish', '$2a$10$RNTZ8u0y.MvZHPASac/9rOPZX4nqDNBakruhyGICXzxPjnZ2HjDs2', 'gtanish544@gmail.com', 9358174158, 'ROLE_VENDER');
INSERT INTO Users
    (user_name, password, user_email, user_contact, user_role)
VALUES
    ('Utkarsh', '$2a$10$RNTZ8u0y.MvZHPASac/9rOPZX4nqDNBakruhyGICXzxPjnZ2HjDs2', 'utkarshsirohi13@gmail.com', 19876543210, 'ROLE_GUEST');
INSERT INTO Users
    (user_name, password, user_email, user_contact, user_role)
VALUES
    ('Ujjwal', '$2a$10$RNTZ8u0y.MvZHPASac/9rOPZX4nqDNBakruhyGICXzxPjnZ2HjDs2', 'ujjwal189@gmail.com', 19876543211, 'ROLE_GUEST');
INSERT INTO Users
    (user_name, password, user_email, user_contact, user_role)
VALUES
    ('Yogesh', '$2a$10$RNTZ8u0y.MvZHPASac/9rOPZX4nqDNBakruhyGICXzxPjnZ2HjDs2', 'ysr2002@gmail.com', 19876543212, 'ROLE_GUEST');
```

4.8.6.3 Database Setup:

DatabaseConfig.java 1

src > main > java > pbl > project > database > DatabaseConfig.java > Language Support for Java(TM) by Red Hat > DatabaseConfig > namedParameterJdbcTemplate(DataSource)

```java
package pbl.project.database;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;

public class DatabaseConfig {
    @Bean
    public NamedParameterJdbcTemplate namedParameterJdbcTemplate(DataSource data_source) {
        return new NamedParameterJdbcTemplate(data_source);
    }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource data_source = new DriverManagerDataSource();
        data_source.setDriverClassName(driverClassName:"org.h2.Driver");
        data_source.setUrl(url:"jdbc:h2:mem:testdb");
        data_source.setUsername(username:"sa");
        data_source.setPassword(password:"");
        return data_source;
    }

    @Bean
    public DataSource loadSchema() {
        return new EmbeddedDatabaseBuilder().setType(EmbeddedDatabaseType.H2)
                .addScript(script:"classpath:data.sql")
                .addScript(script:"classpath:schema.sql")
                .build();
    }

}
```

4.8.6.4 Database Access:

DatabaseOperation.java

src > main > java > pbl > project > database > DatabaseOperation.java > ...

```java
package pbl.project.database;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Repository;

import pbl.project.beans.Ticket;
import pbl.project.beans.User;

@Repository
public class DatabaseOperation {
    @Autowired
    protected NamedParameterJdbcTemplate jdbc_connector;

    @Autowired
    private BCryptPasswordEncoder password_encoder;


    public void add_user(String username, String password, String email, Long contact) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "INSERT INTO Users (user_name, password, user_email, user_contact, user_role) VALUES (:user_name, :password,
        :user_email, :user_contact, :user_role)";
        parameters.addValue(paramName:"user_name", username);
        parameters.addValue(paramName:"password", password_encoder.encode(password));
        parameters.addValue(paramName:"user_email", email);
        parameters.addValue(paramName:"user_contact", contact);
        parameters.addValue(paramName:"user_role", value:"ROLE_GUEST");
        jdbc_connector.update(sql_query, parameters);
    }

    public User get_user_info(String username) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "SELECT * FROM Users WHERE user_name = :user_name";
        parameters.addValue(paramName:"user_name", username);
        ArrayList<User> users = (ArrayList<User>) jdbc_connector.query(sql_query, parameters,
                new BeanPropertyRowMapper<User>(mappedClass:User.class));
        if (users.size() > 0) {
            return users.get(index:0);
        }
        return null;
    }

    public ArrayList<User> get_all_users() {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "SELECT * FROM Users";
        return (ArrayList<User>) jdbc_connector.query(sql_query, parameters,
                new BeanPropertyRowMapper<User>(mappedClass:User.class));
    }


    public void add_ticket(Ticket ticket) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "INSERT INTO Tickets (event_name, event_date, venue_name, ticket_holder_name, ticket_holder_contact,
        ticket_type, booking_reference, ticket_price) VALUES (:event_name, :event_date, :venue_name, :ticket_holder_name,
        :ticket_holder_contact, :ticket_type, :booking_reference, :ticket_price)";
        parameters.addValue(paramName:"event_name", ticket.getEvent_name());
        parameters.addValue(paramName:"event_date", ticket.getEvent_date());
        parameters.addValue(paramName:"venue_name", ticket.getVenue_name());
        parameters.addValue(paramName:"ticket_holder_name", ticket.getTicket_holder_name());
        parameters.addValue(paramName:"ticket_holder_contact", ticket.getTicket_holder_contact());
        parameters.addValue(paramName:"ticket_type", ticket.getTicket_type());
        parameters.addValue(paramName:"booking_reference", ticket.getBooking_reference());
        parameters.addValue(paramName:"ticket_price", ticket.getTicket_price());
        jdbc_connector.update(sql_query, parameters);
    }

    public void update_ticket(Ticket ticket) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "UPDATE Tickets SET event_name = :event_name, event_date = :event_date, venue_name = :venue_name,
        ticket_holder_contact = :ticket_holder_contact, ticket_type = :ticket_type, ticket_price = :ticket_price WHERE ticket_id =
        :ticket_id";
        parameters.addValue(paramName:"event_name", ticket.getEvent_name());
        parameters.addValue(paramName:"event_date", ticket.getEvent_date());
        parameters.addValue(paramName:"venue_name", ticket.getVenue_name());
        parameters.addValue(paramName:"ticket_holder_contact", ticket.getTicket_holder_contact());
        parameters.addValue(paramName:"ticket_type", ticket.getTicket_type());
        parameters.addValue(paramName:"ticket_price", ticket.getTicket_price());
        parameters.addValue(paramName:"ticket_id", ticket.getTicket_id());
        jdbc_connector.update(sql_query, parameters);
    }

```

```java
    public void delete_ticket(int ticket_id) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "DELETE FROM Tickets WHERE ticket_id = :ticket_id";
        parameters.addValue(paramName:"ticket_id", ticket_id);
        jdbc_connector.update(sql_query, parameters);
    }

    public Ticket get_ticket_by_id(int ticket_id) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "SELECT * FROM Tickets WHERE ticket_id = :ticket_id";
        parameters.addValue(paramName:"ticket_id", ticket_id);
        ArrayList<Ticket> tickets = (ArrayList<Ticket>) jdbc_connector.query(sql_query, parameters,
                new BeanPropertyRowMapper<Ticket>(mappedClass:Ticket.class));
        if (tickets.size() > 0) {
            return tickets.get(index:0);
        }
        return null;
    }

    public Ticket get_ticket_by_event_name(String event_name) {
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        String sql_query = "SELECT * FROM Tickets WHERE event_name = :event_name";
        parameters.addValue(paramName:"event_name", event_name);
        ArrayList<Ticket> ticket_info = (ArrayList<Ticket>) jdbc_connector.query(sql_query, parameters,
                new BeanPropertyRowMapper<Ticket>(mappedClass:Ticket.class));
        if (ticket_info.size() > 0) {
            return ticket_info.get(index:0);
        }
        return null;
    }

    public ArrayList<Ticket> get_all_tickets(User user) {
        String sql_query = "SELECT * FROM Tickets";
        if (user.getUser_role().equals(anObject:"ROLE_VENDER")) {
            return (ArrayList<Ticket>) jdbc_connector.query(sql_query,
                    new BeanPropertyRowMapper<Ticket>(mappedClass:Ticket.class));
        } else if (user.getUser_role().equals(anObject:"ROLE_GUEST")) {
            sql_query = "SELECT * FROM Tickets WHERE ticket_holder_name = :ticket_holder_name";
            MapSqlParameterSource parameters = new MapSqlParameterSource();
            parameters.addValue(paramName:"ticket_holder_name", new Object[] { user.getUser_name() });
            return (ArrayList<Ticket>) jdbc_connector.query(sql_query, parameters,
                    new BeanPropertyRowMapper<Ticket>(mappedClass:Ticket.class));
        }
        return new ArrayList<Ticket>();
    }
}
```

4.8.7 Security:

LoggingAccessDeniedHandler.java

src > main > java > pbl > project > security > LoggingAccessDeniedHandler.java > ...

```java
package pbl.project.security;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;

@Component
public class LoggingAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
            AccessDeniedException accessDeniedException) throws IOException, ServletException {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

        if (authentication != null) {
            System.out.println("Unauthorized access by " + authentication.getName());
        }
    }
}

```

15 7 6 0 Live Share Ln 14, Col 1 Tab Size: 4 UTF-8 CRLF {} Java Go Live 3 Spell Prettier

SecurityConfig.java 5

src > main > java > pbl > project > security > SecurityConfig.java > ...

```java
package pbl.project.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private LoggingAccessDeniedHandler access_denied_handler;

    @Autowired
    UserDetailsService user_details_service;

```

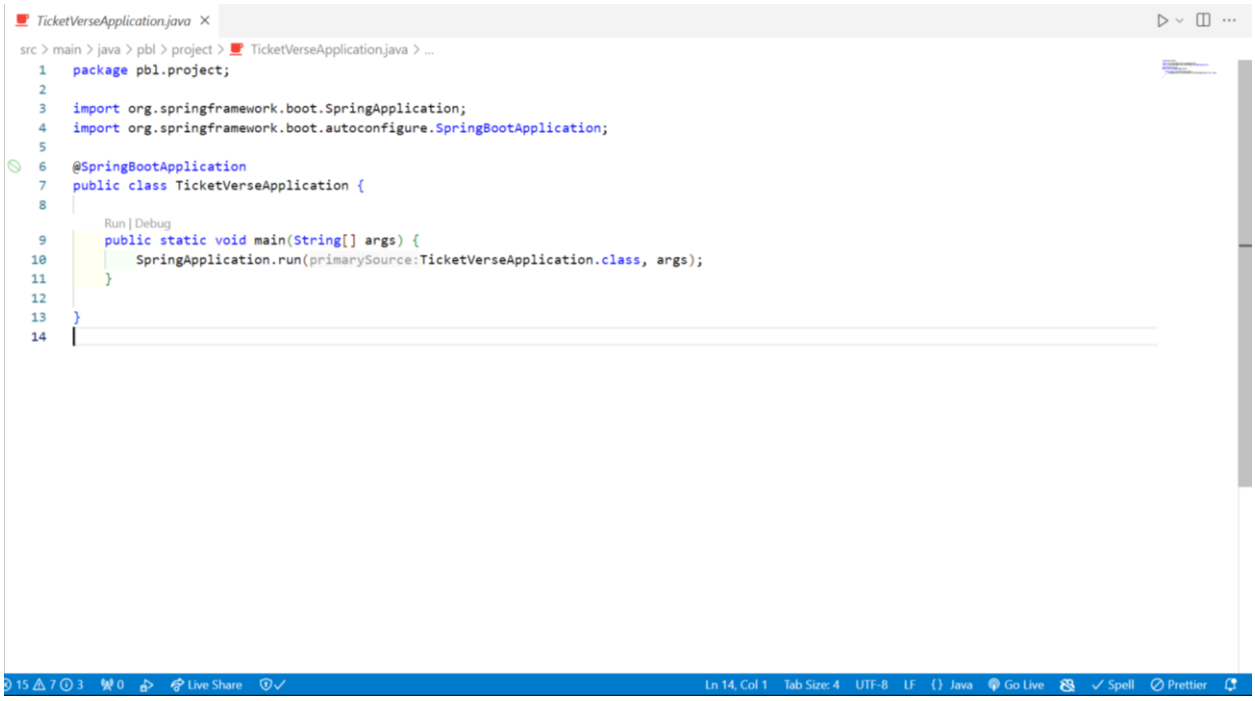
```
    public void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http.headers().frameOptions().disable();
        http.authorizeHttpRequests()
                .antMatchers(...antPatterns:"/add_ticket/**", "/create_ticket/**", "/config_ticket/**", "/update_ticket/**",
                        "/delete_ticket/**", "/user_list/**")
                .hasAnyRole(...roles:"VENDER")
                .antMatchers(...antPatterns:"/ticket_list", "/display_ticket/**").hasAnyRole(...roles:"GUEST", "VENDER")
                .antMatchers(...antPatterns:"/", "/registration", "/css/**", "/images/**", "/login", "/**").permitAll()
                .antMatchers(...antPatterns:"/h2/**").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin().loginPage(loginPage:"/login").defaultSuccessUrl(defaultSuccessUrl:"/ticket_list", alwaysUse:true).permitAll
                ()
                .and()
                .logout().invalidateHttpSession(invalidateHttpSession:true).clearAuthentication(clearAuthentication:true)
                .logoutRequestMatcher(new AntPathRequestMatcher(pattern:"/logout"))
                .logoutSuccessUrl(logoutSuccessUrl:"/login?logout").permitAll()
                .and()
                .exceptionHandling().accessDeniedHandler(access_denied_handler);
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    public void configure(AuthenticationManagerBuilder authentication) throws Exception {
        authentication.userDetailsService(user_details_service).passwordEncoder(passwordEncoder());
    }

}
```


UserDetailsServiceImpl.java

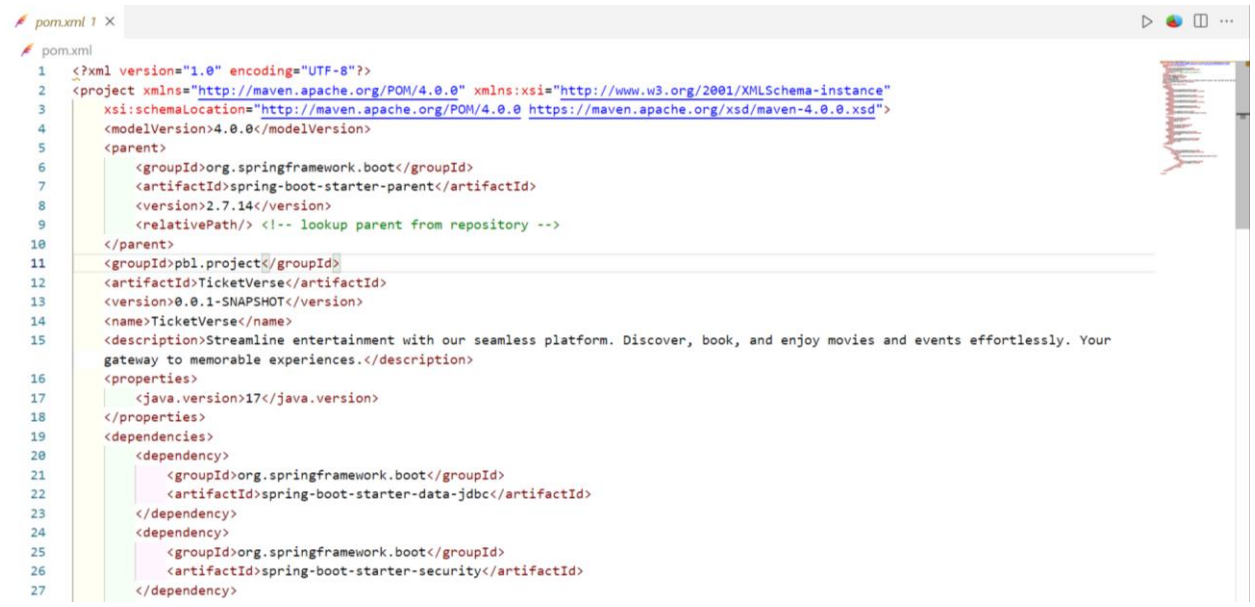
src > main > java > pbl > project > services > UserDetailsServiceImpl.java > ...

```
package pbl.project.services;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;

import pbl.project.database.DatabaseOperation;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    @Lazy
    private DatabaseOperation databaseOperation;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        pbl.project.beans.User user_info = databaseOperation.get_user_info(username);
        if (user_info == null) {
            throw new UsernameNotFoundException("username with " + username + "not found.");
        }

        ArrayList<String> role_list = new ArrayList<String>();
        role_list.add((String) user_info.getUser_role());

        List<GrantedAuthority> grant_list = new ArrayList<GrantedAuthority>();

        if (role_list != null) {
            for (String role : role_list) {
                grant_list.add(new SimpleGrantedAuthority(role));
            }
        }

        UserDetails login_user_info = (UserDetails) (new User(user_info.getUser_name(), user_info.getPassword(),
                grant_list));

        return login_user_info;
    }

}
```

4.8.8 Application Initialization:



4.8.9 Project Configuration:

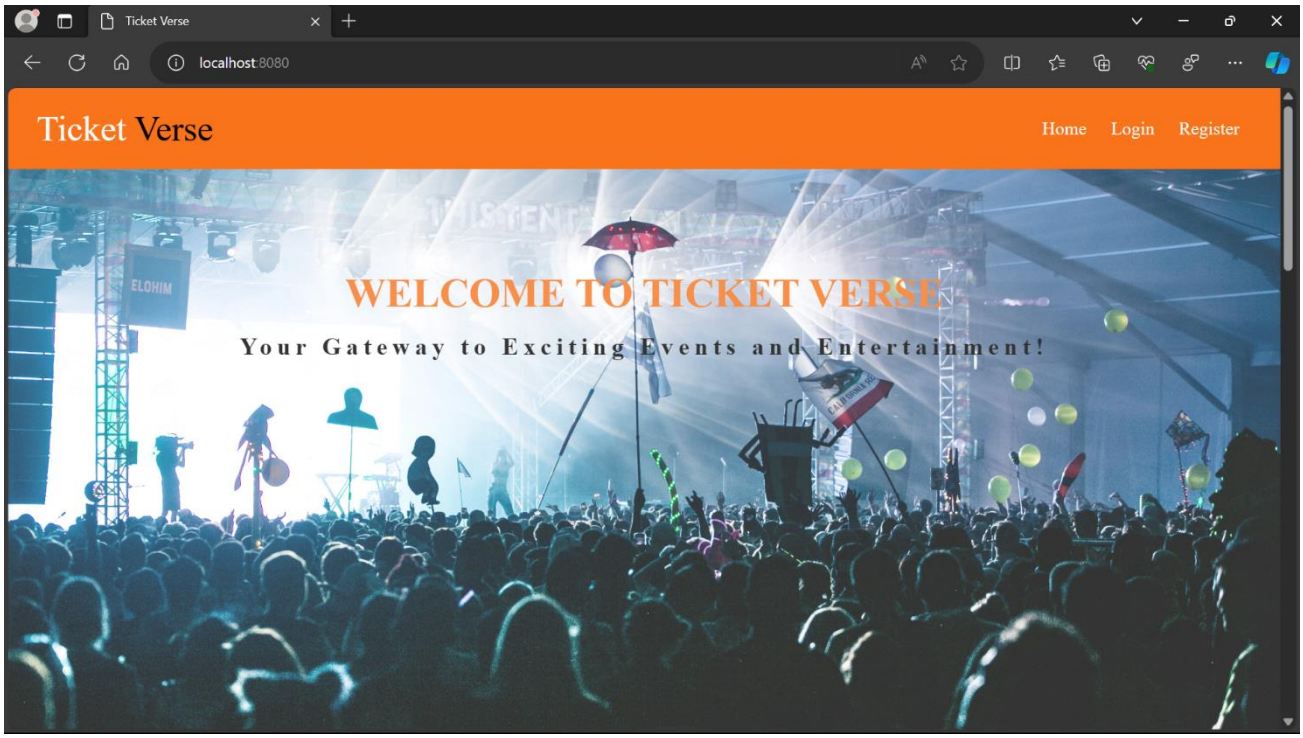


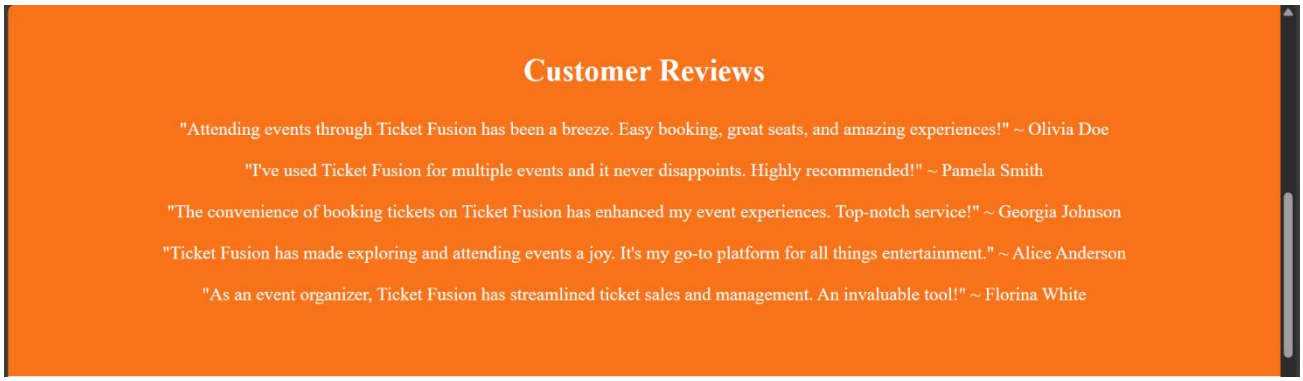
```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.thymeleaf.extras</groupId>
            <artifactId>thymeleaf-extras-springsecurity5</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <image>
                        <builder>paketobuildpacks/builder-jammy-base:latest</builder>
                    </image>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>

```

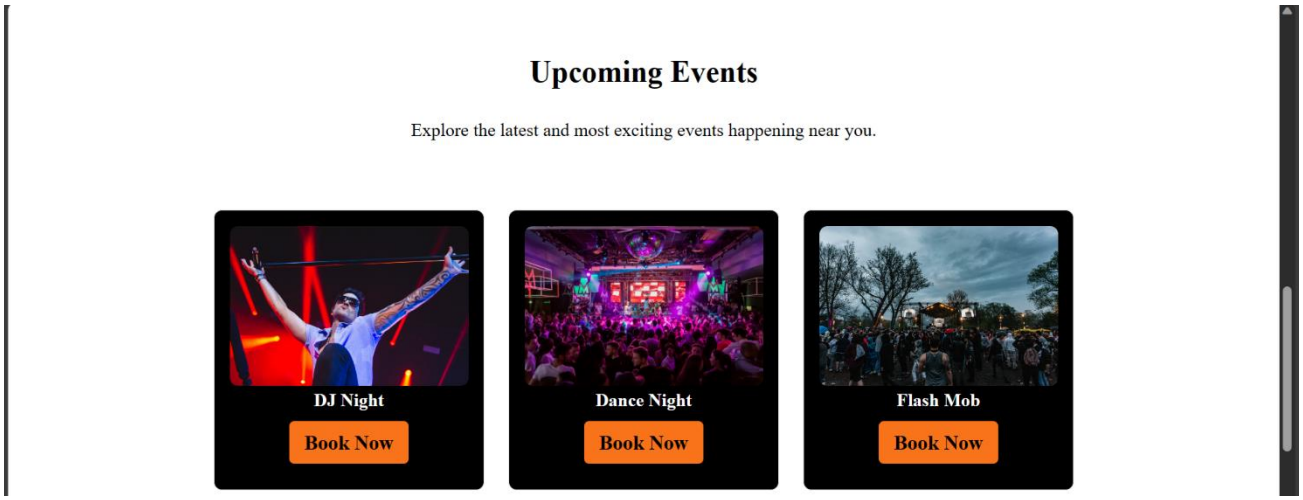
4.9Screenshots of the Project

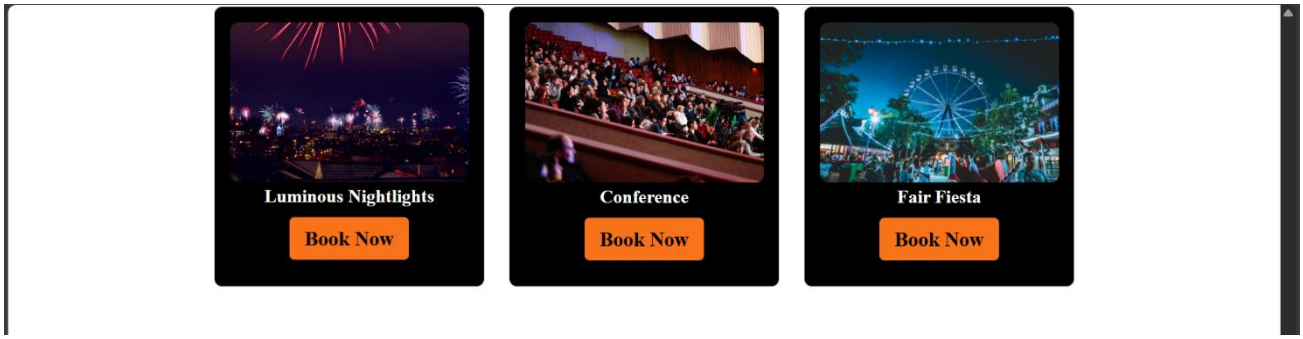
4.9.1 Web UI Interface

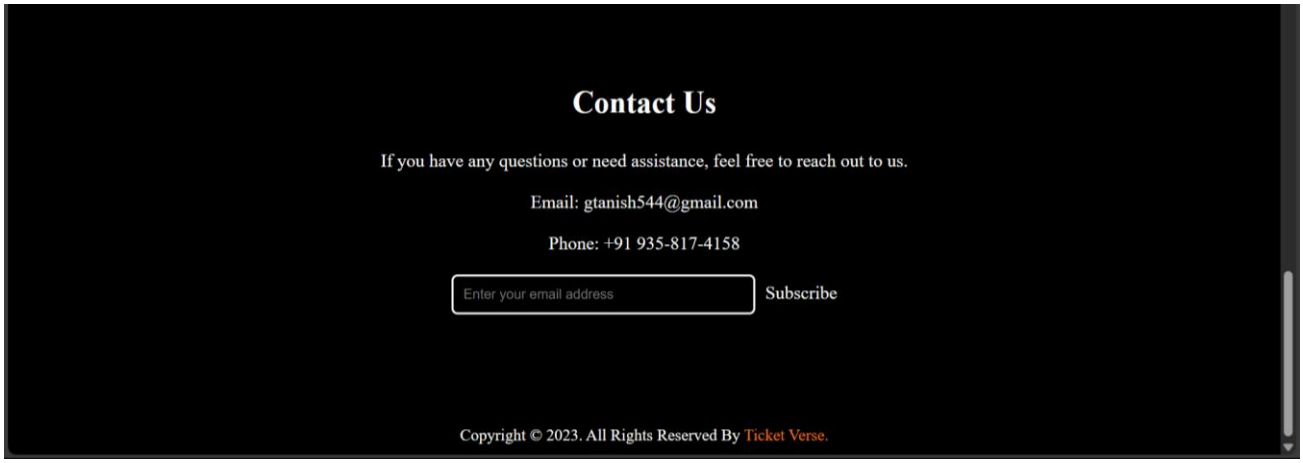
- **Home Page:**



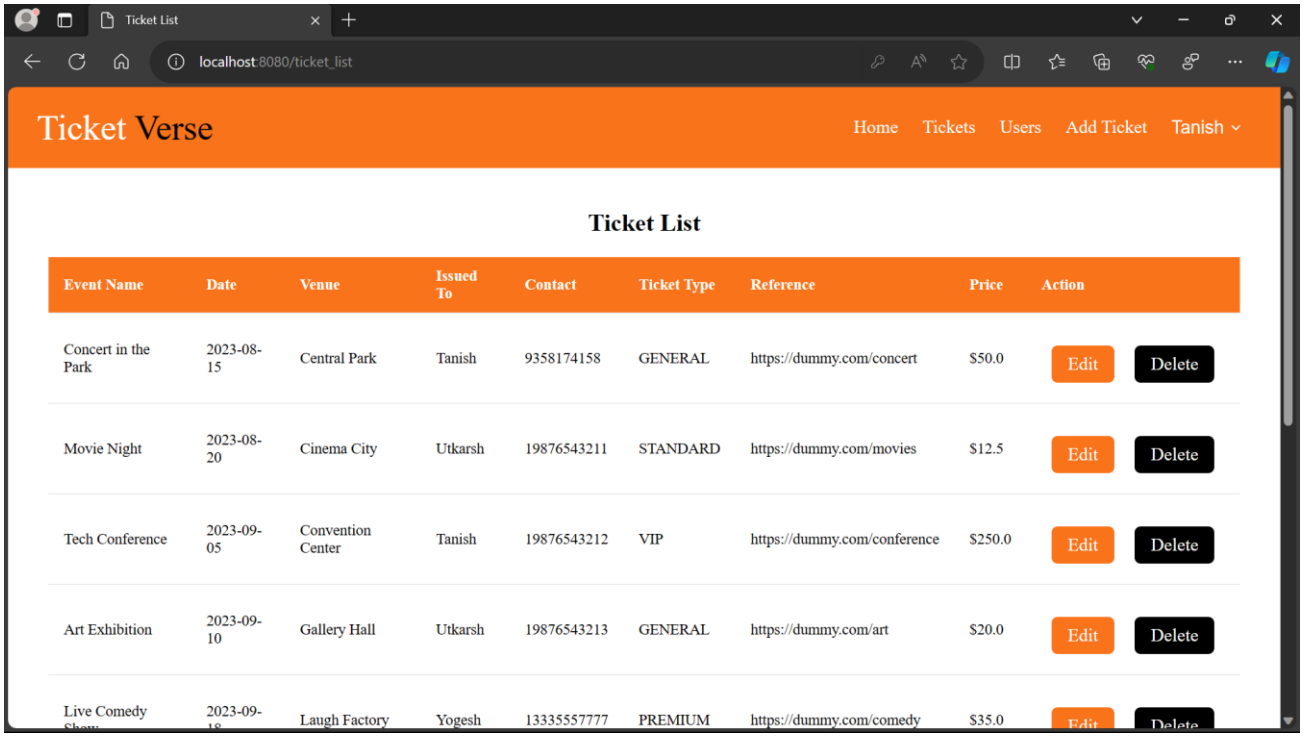




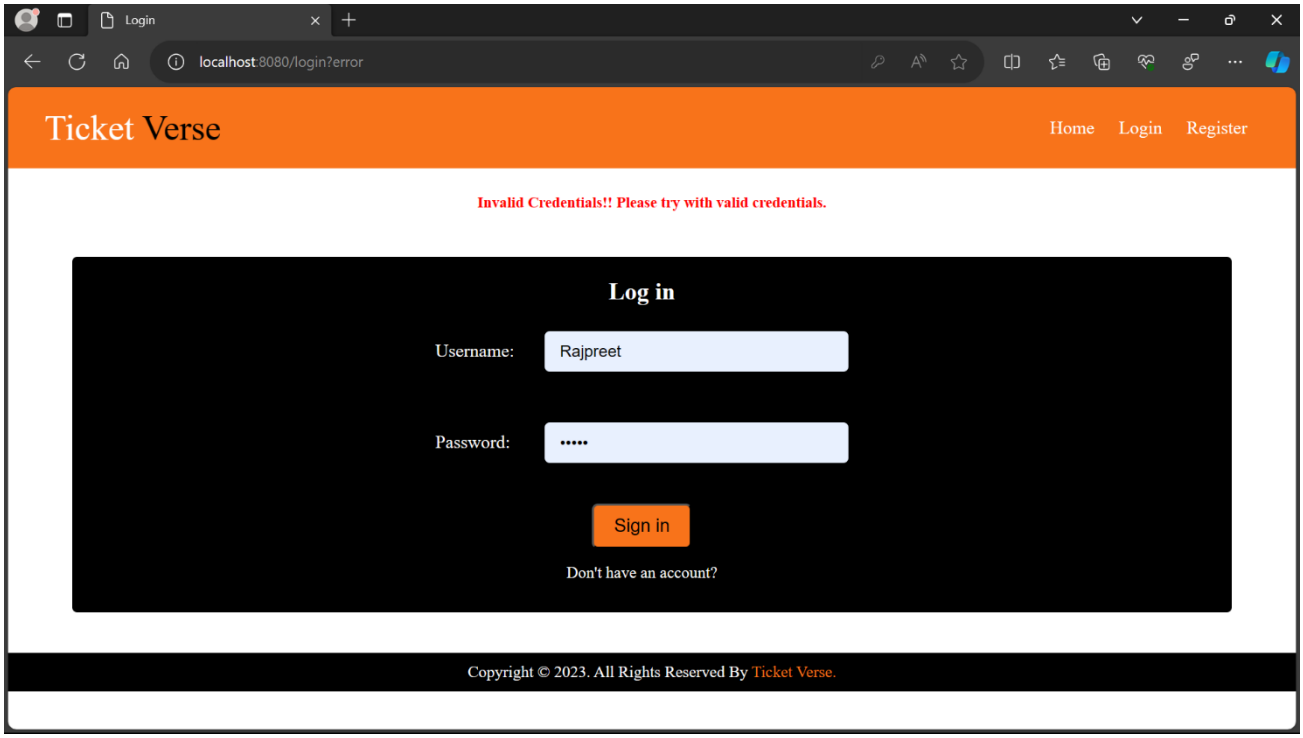




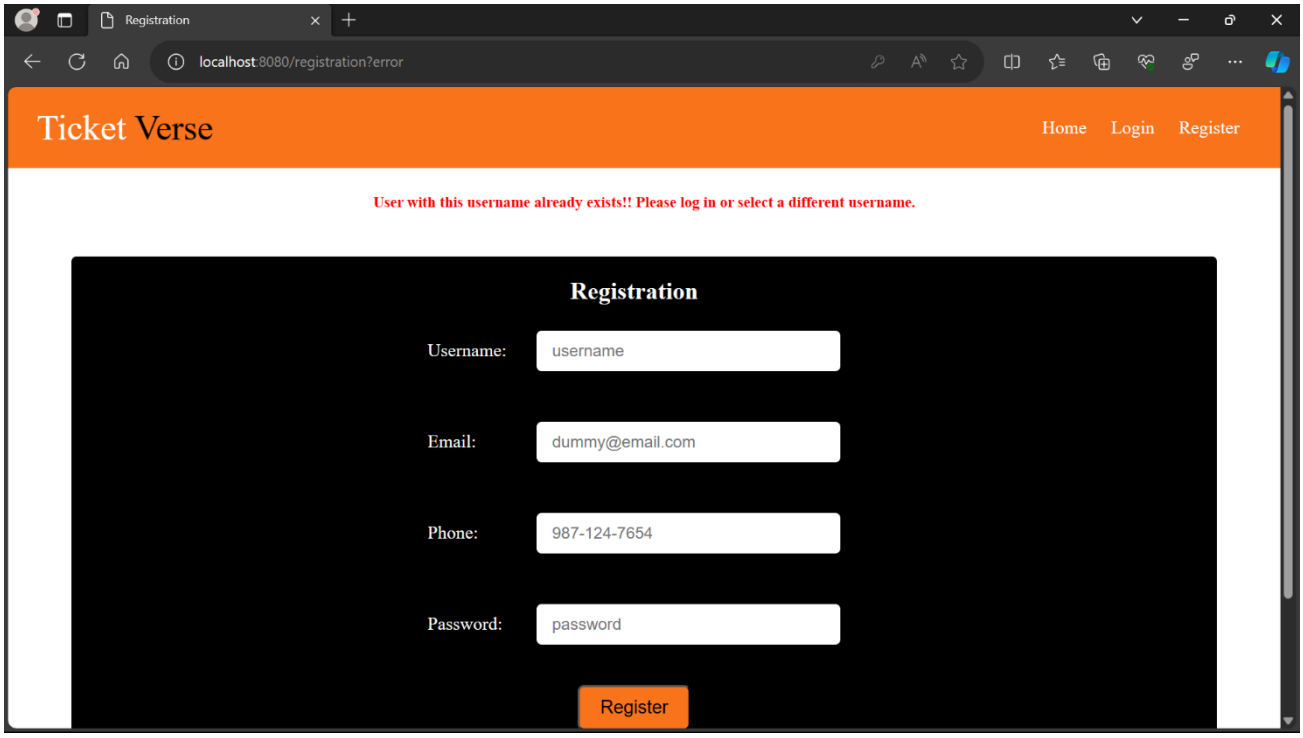
- **Login:**
  - **Successful Login:**



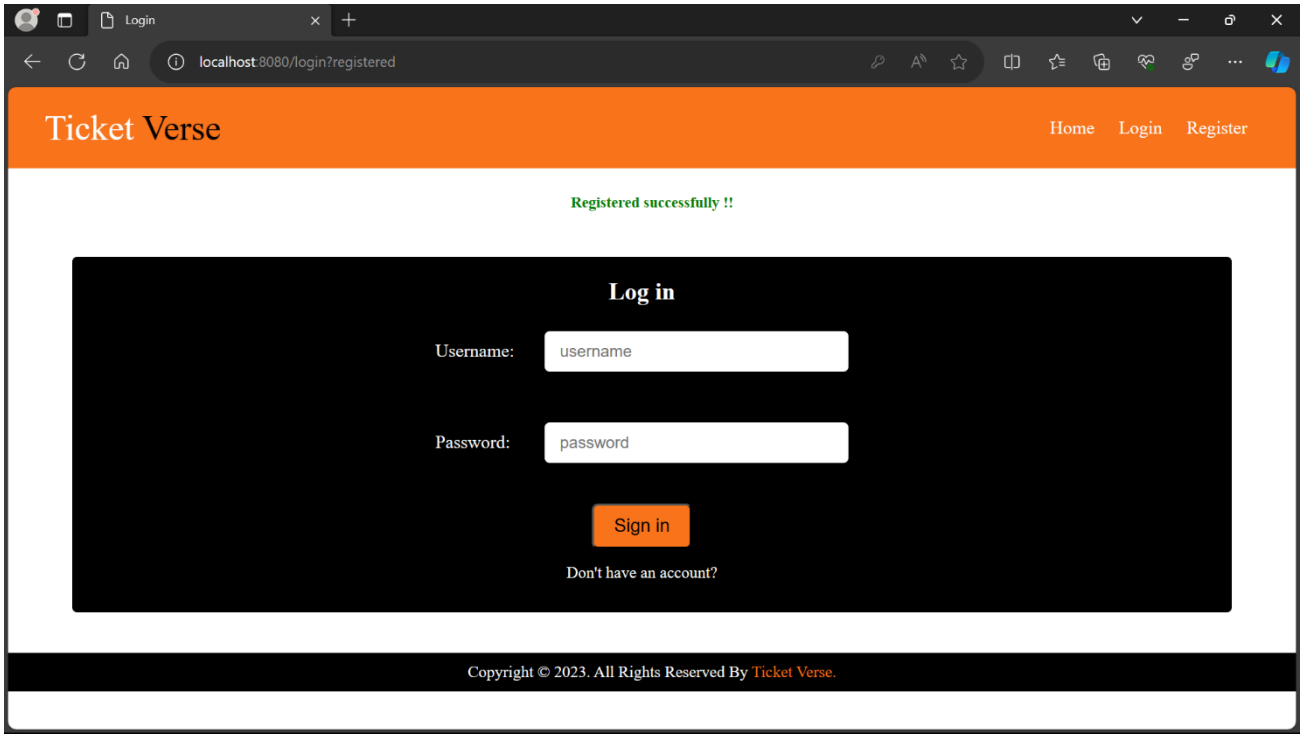
- **Invalid Credentials:**



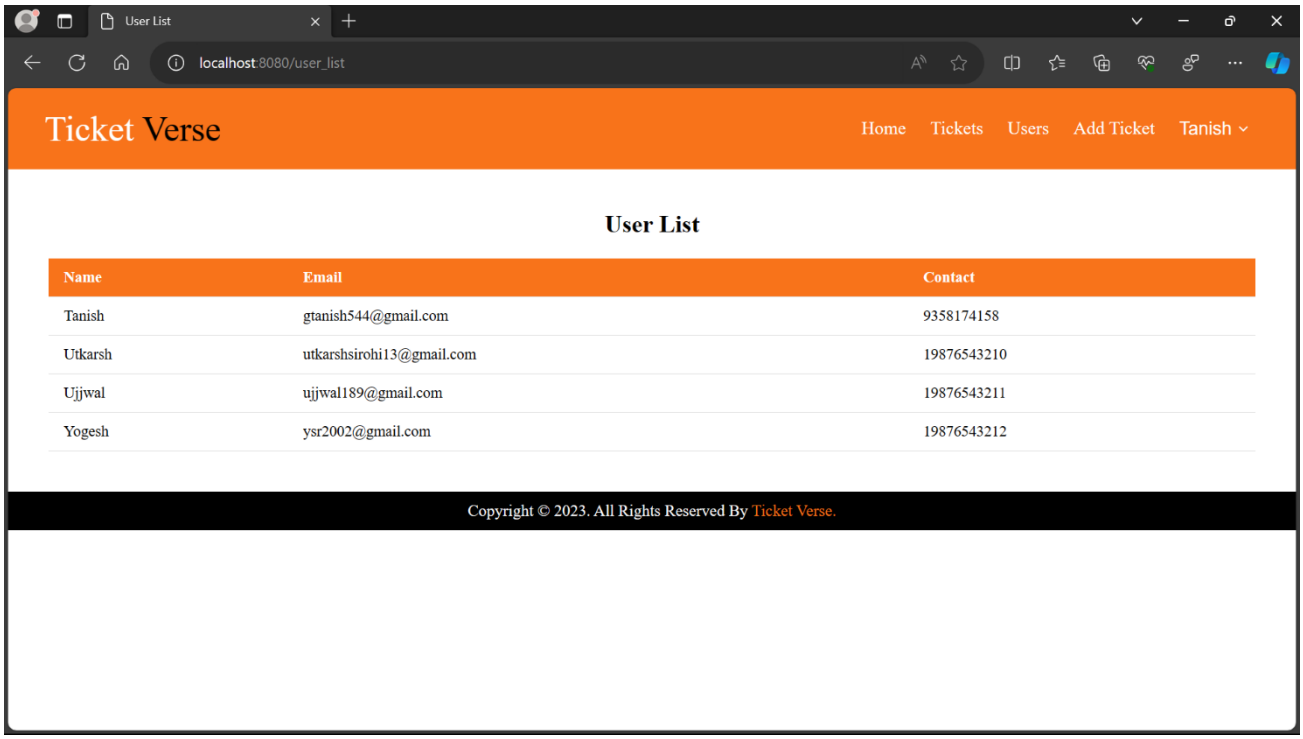
- **Register:**
  - **If user with username exist:**



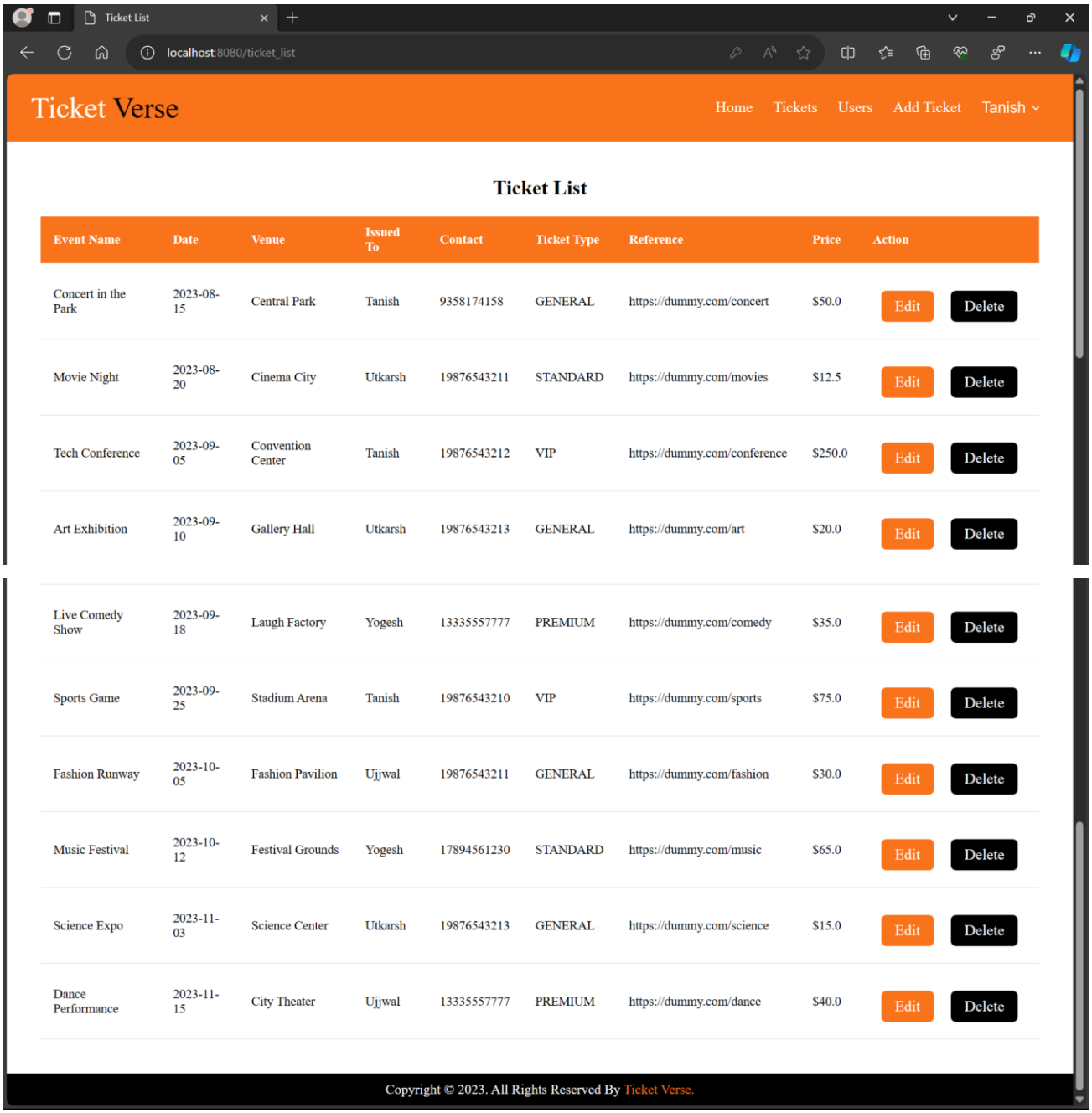
- **If user with username not exist:**



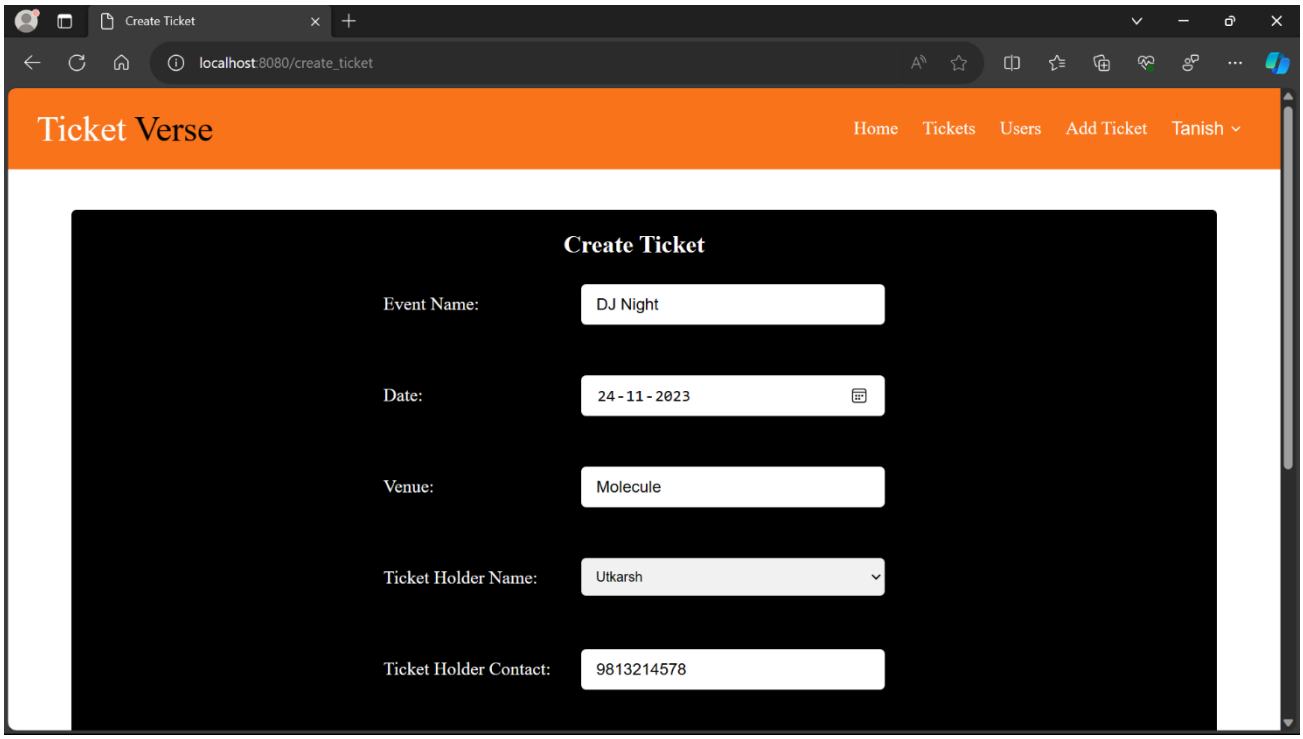
- **When logged in as an admin:**
  - **View Users List:**

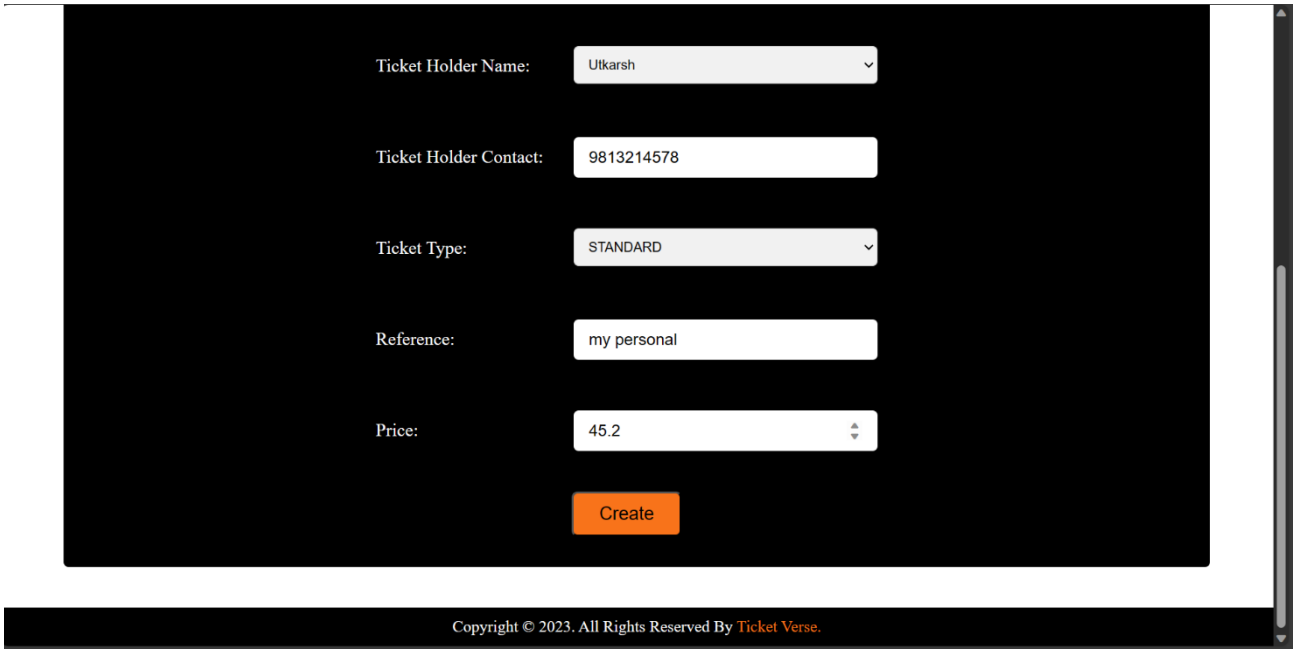


- **View Booked Tickets:**

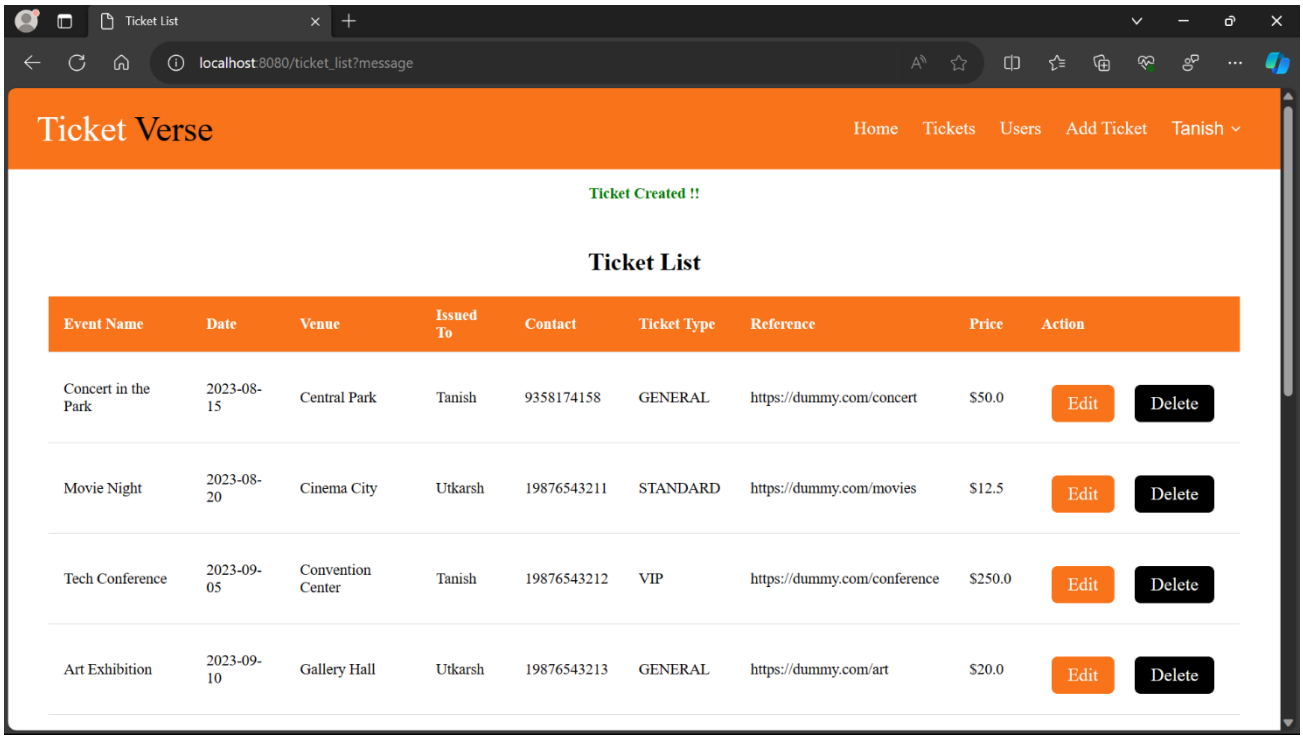


- **Book New Ticket:**

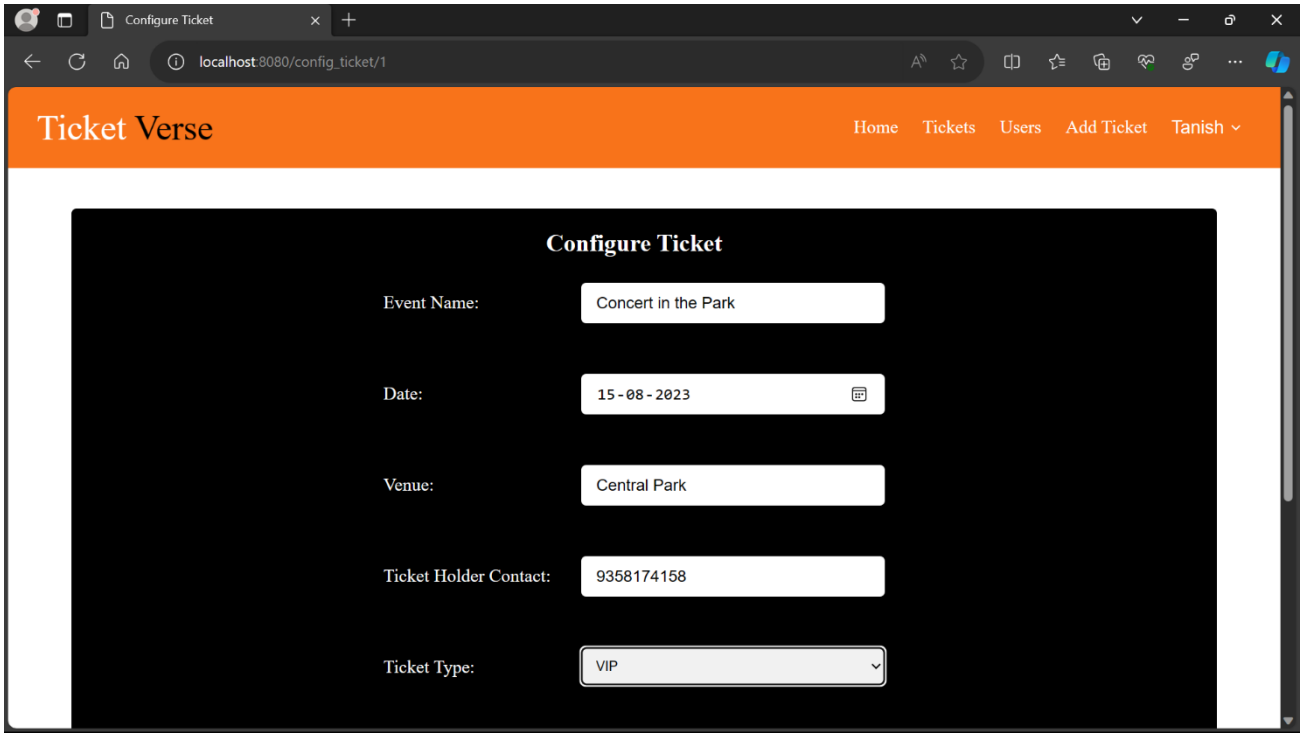


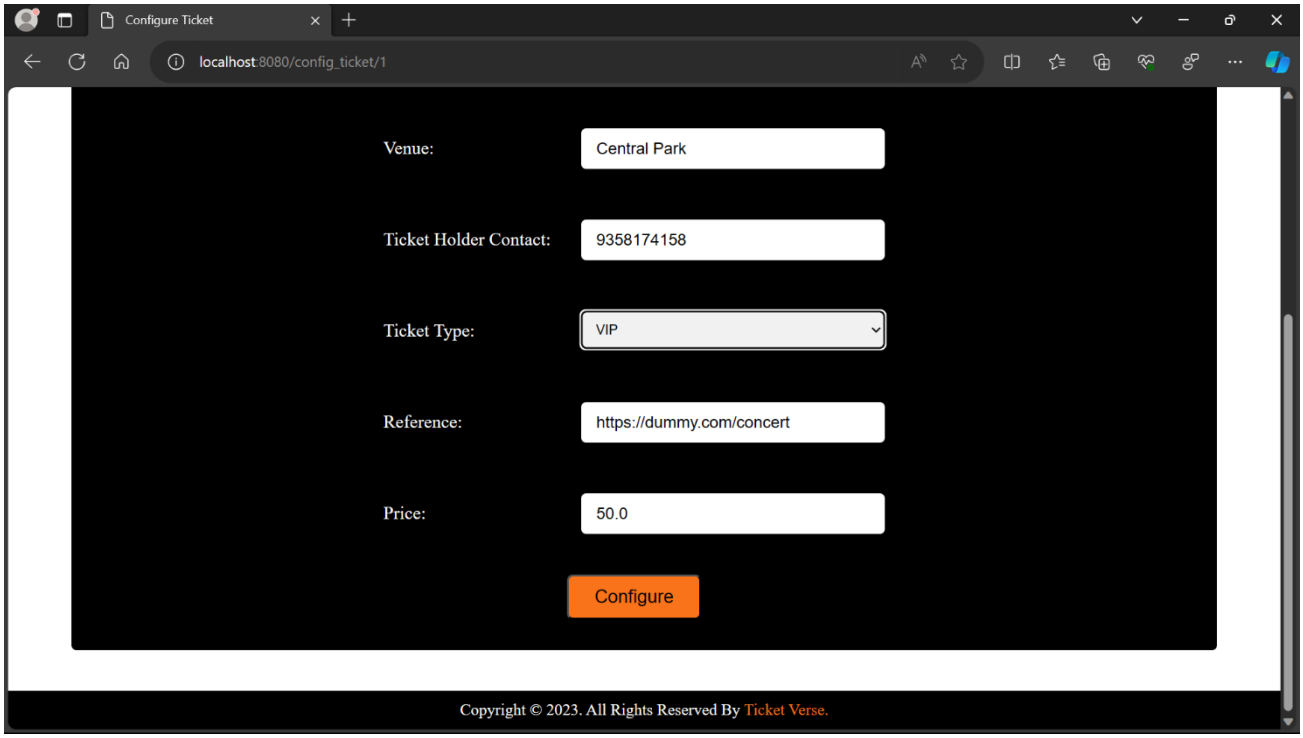


- **When Ticket Added Successfully**

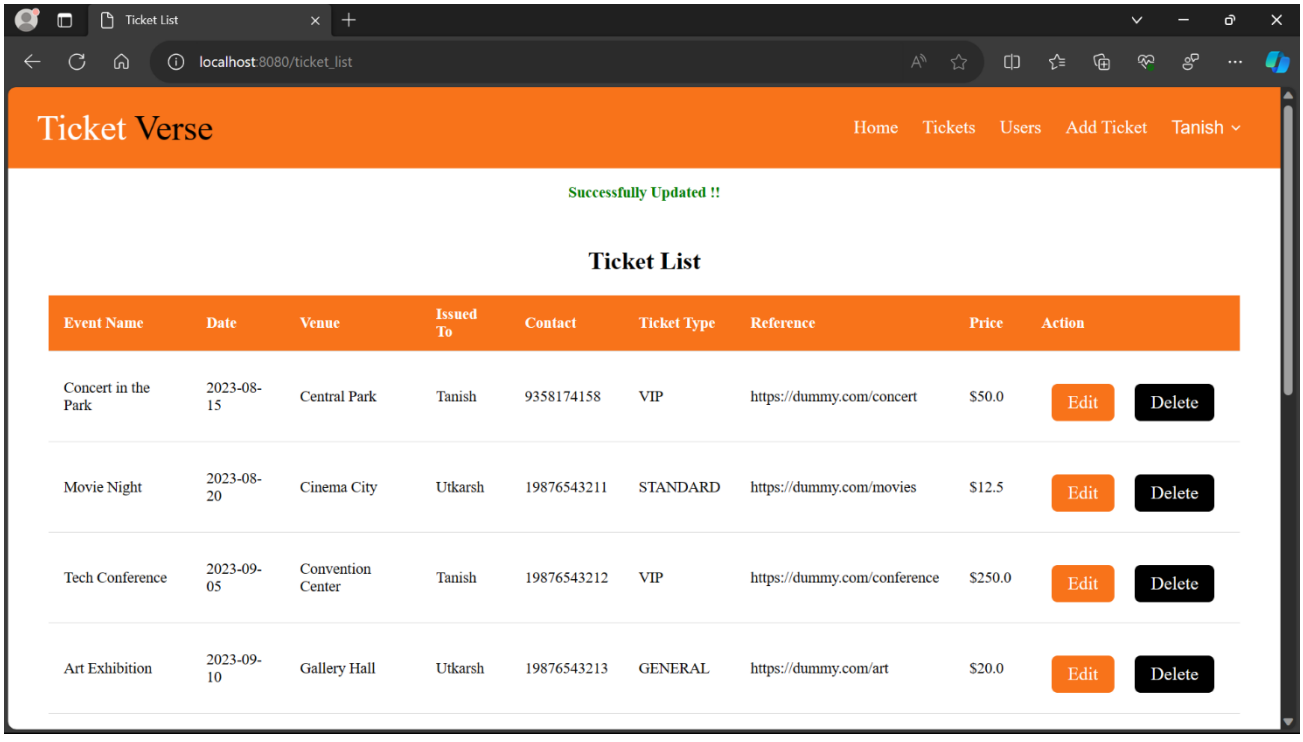


- **Edit Booked Ticket Details:**

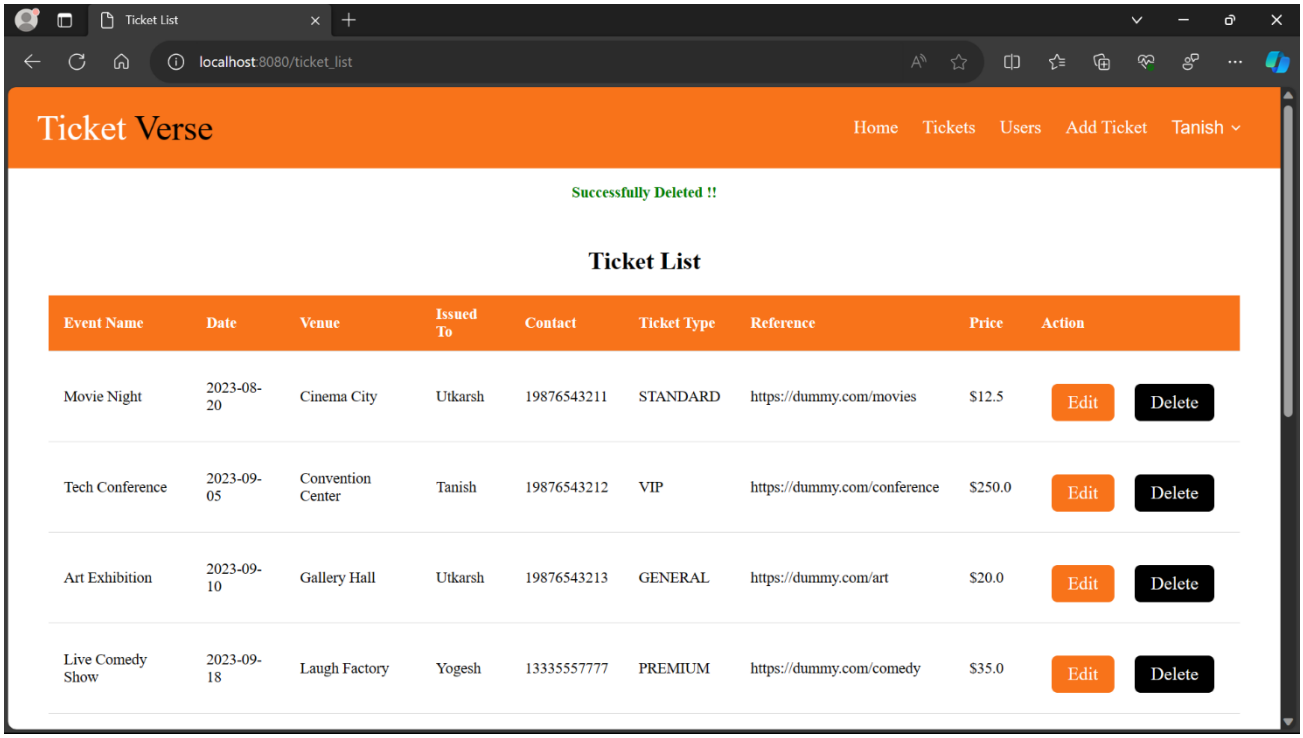




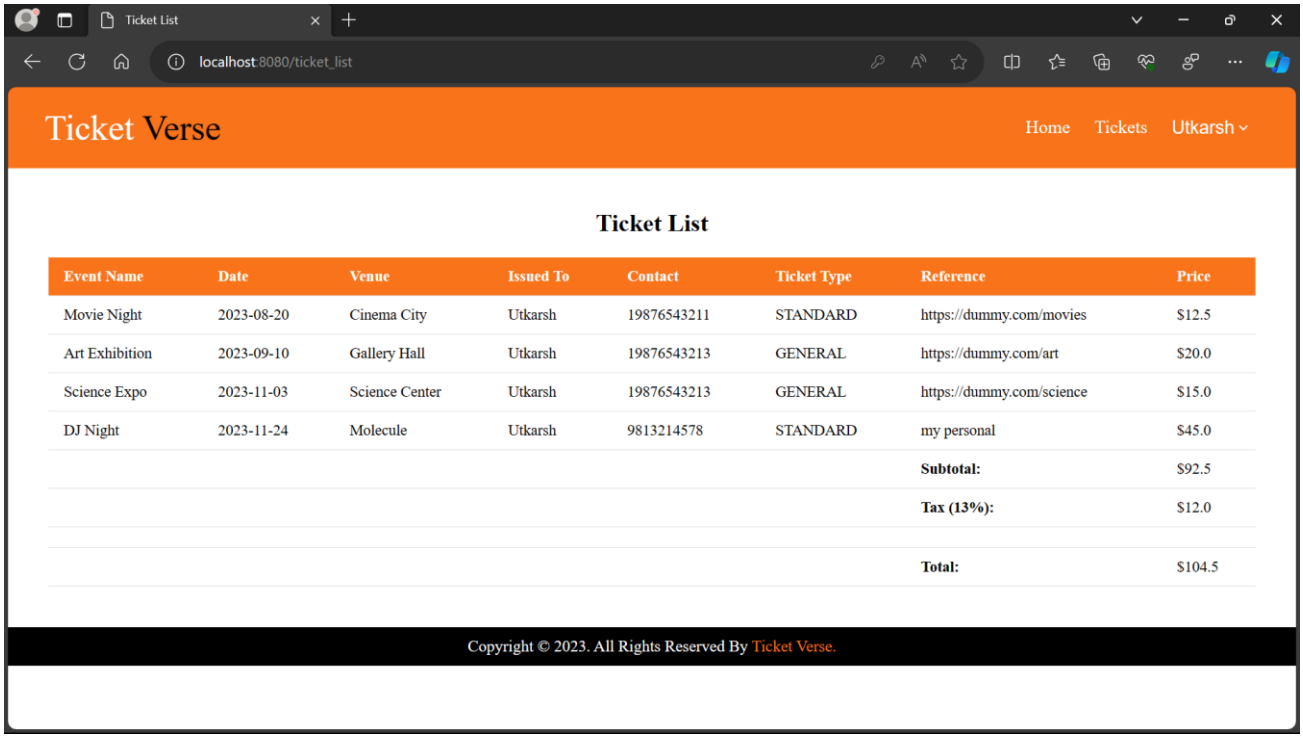
- **When Ticket edited successfully:**



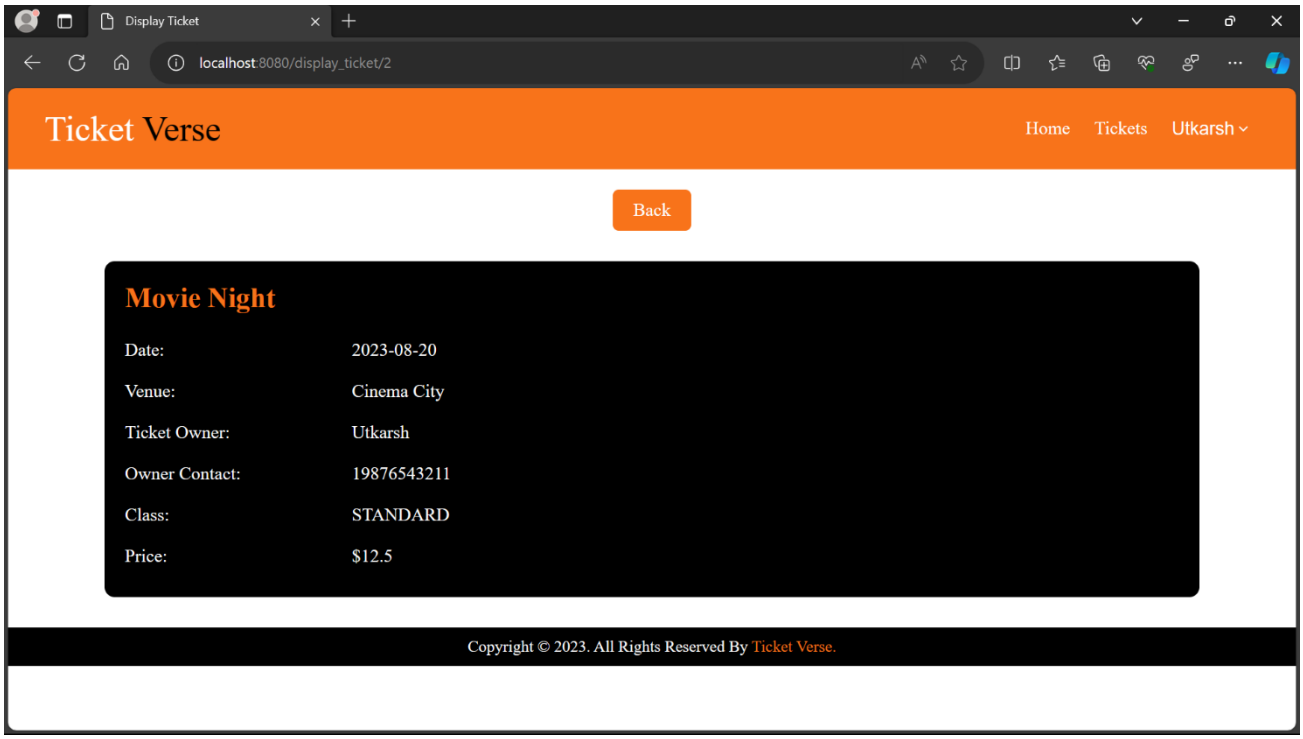
- **Delete Booked Ticket:**



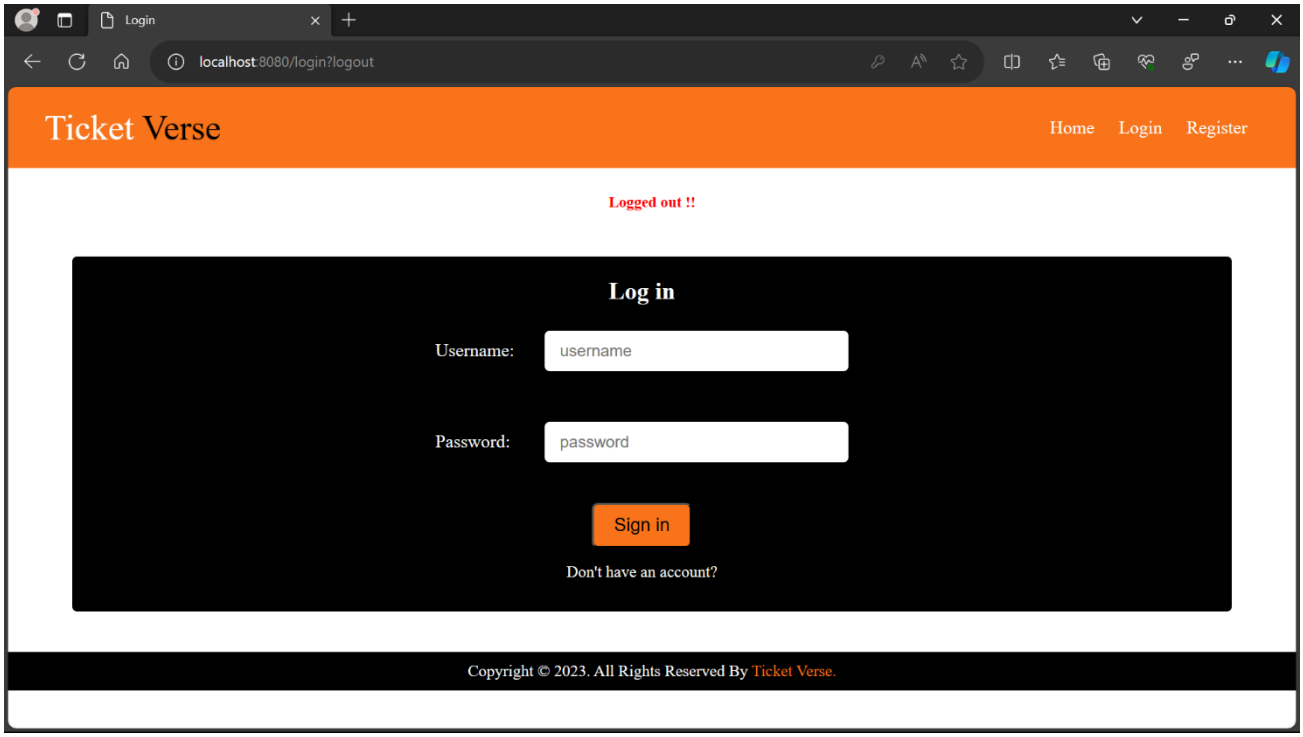
- **When logged in as a user:**
  - **View Booked Tickets:**



- **View Ticket:**



- **Logout**



5 Future Outlook & Bibliography

5.1Scope of the Project

The scope of the Event & Movie Ticket Booking System extends beyond its initial implementation. Future enhancements and potential areas for expansion include:

- **Mobile Application Development:** Extend the system's accessibility by developing a mobile application to cater to a broader audience.
- **Integration with Additional Payment Gateways:** Enhance user convenience by integrating with a variety of secure payment gateways to provide users with more options.
- **Advanced User Engagement Features:** Implement features such as personalized recommendations, social media integration, and loyalty programs to enhance user engagement.

5.2Limitations

While the Event & Movie Ticket Booking System addresses several challenges in traditional ticketing methods, it has its limitations:

- **Dependency on Internet Connectivity:** The system relies on internet connectivity, which may pose challenges in regions with limited access.
- **Event and Movie Database Updates:** Keeping the database up-to-date with the latest events and movies requires continuous efforts and collaboration with event organizers.

5.3Conclusion

The development of the Event & Movie Ticket Booking System using the Spring Boot framework in Java has successfully resulted in a user-friendly,

efficient, and secure platform for booking tickets. The system's intuitive interface, real-time updates, and secure payment processing contribute to a positive user experience. While the project achieves its intended objectives, continuous improvements and adaptations to emerging technologies will be essential to stay competitive in the dynamic domain of event and movie ticketing.

5.4Bibliography

1. ***Horstmann, Cay S.*** (2018). Core Java Volume I – Fundamentals (11th Edition). Pearson.
    - This comprehensive book provides a deep dive into Java fundamentals, making it an essential reference for Java development.

2. ***Long, Josh.*** (2020). Reactive Spring. O'Reilly Media.
    - For a more advanced understanding of Spring Boot and its reactive programming capabilities, "Reactive Spring" by Josh Long is a valuable resource.

3. ***Sierra, Kathy, and Bates, Bert.*** (2014). Head First Java (2nd Edition). O'Reilly Media.
    - "Head First Java" is an engaging and beginner-friendly book that provides an introduction to Java programming concepts.

5.5Web References

- Spring Boot Documentation.

  https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

- Thymeleaf Documentation.

  https://www.thymeleaf.org/documentation.html