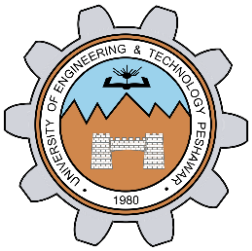
Project Report



Spring 2025

Group Members

Name	Registration Number
Hassan Zaib Jadoon	22PWCSE2144
Ahsan Raza	22PWCSE2099
Mutahhar Fayyaz	22PWCSE2176

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Engr. Summeyea Salahuddin

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Event Booking System - Project Report

Executive Summary

The Event Booking System is a comprehensive web application developed to revolutionize the way events are managed and tickets are booked. This project represents a complete evolution from a basic PHP application to a sophisticated, production-ready Laravel-based platform that serves real-world event management needs.

Project Scope

The system provides a complete end-to-end solution for event management, from creation and promotion to booking and attendance tracking. It serves multiple user types including event organizers, administrators, and general users seeking to attend events.

Key Deliverables

- **Production-Ready Application**: Fully deployed and operational system
- **Advanced Seat Management**: Individual seat selection and real-time availability tracking
- **Automated Email Notifications**: Professional communication system
- **Comprehensive Admin Panel**: Full event and user management capabilities
- **Mobile-Responsive Interface**: Cross-device compatibility
- **Secure Authentication System**: Role-based access control

Project Outcomes

The system successfully addresses the challenges of manual event management while providing a scalable, secure, and user-friendly platform that can handle real-world event booking scenarios with professional-grade features.

Introduction

Background

Traditional event booking systems often suffer from limitations such as manual processes, lack of real-time updates, poor user experience, and security vulnerabilities. The Event Booking System was conceived to address these challenges by providing a modern, comprehensive solution that leverages contemporary web technologies.

Problem Statement

The primary challenges in existing event booking systems include:

- Manual seat management leading to overbooking
- Lack of real-time availability updates
- Poor communication between organizers and attendees
- Inadequate Data Management
- Non-responsive interfaces limiting mobile access
- Complex booking processes deterring users

Project Objectives

Primary Objectives:

- Develop a user-friendly event booking platform
- Implement real-time seat management and availability tracking
- Create an automated communication system
- Ensure robust security and data protection
- Provide comprehensive administrative capabilities
- Deploy a production-ready application

Secondary Objectives:

- Demonstrate proficiency in modern web development frameworks
- Practice industry-standard security practices
- Create a scalable architecture for future enhancements
- Provide detailed documentation and testing procedures

Project Scope

The system encompasses complete event lifecycle management including event creation, booking and its confirmation, communication, and post-event analysis. It serves as a comprehensive platform for both small-scale and large-scale event management.

The main goal of this project is to make it easy for people to:

- See what events are happening
- Book tickets for events they want to attend
- Get a digital ticket in the form of QR code & Email

Project Overview

System Purpose

The Event Booking System serves as a centralized platform where event organizers can create and manage events while providing attendees with a seamless booking experience. The system eliminates the complexities of manual event management and provides automated solutions for common challenges.

Target Users

1. Event Organizers/Administrators

- Create and manage events
- Configure seating arrangements
- Monitor bookings and attendance
- Generate reports and analytics

2. General Users/Attendees

- Browse and search events

- Select specific seats
- Complete secure bookings
- Receive digital tickets
- Manage booking history

Key Benefits

For Event Organizers:

- Reduced administrative overhead
- Real-time booking monitoring
- Automated communication with attendees
- Professional event presentation
- Comprehensive reporting capabilities

For Attendees:

- Easy event discovery
- Flexible seat selection
- Instant booking confirmation
- Digital ticket convenience

For System Administrators:

- Centralized management
- Automated processes
- Performance monitoring
- Scalable architecture

What problem does it solve?

- Makes booking tickets easy and fast
- Reduces the need for physical ticket counters
- Keeps track of available seats automatically
- Provides digital tickets that can't be easily lost

System Features

For Regular Users:

1. **View Events**: See all upcoming events with details like date, time, location, and price
2. **Book Tickets**: Reserve seats for events they want to attend
3. **Get Digital Tickets**: Receive a QR code that serves as their ticket
4. **See Seat Availability**: Know how many seats are left for each event

For Administrators:

1. **Add New Events**: Create new events with all necessary details
2. **Manage Bookings**: See who has booked tickets

3. **Update Event Information**: Change event details if needed

Special Features:

- **QR Code Generation**: Each booking gets a unique QR code
- **Automatic Seat Management**: System automatically reduces available seats when someone books
- **Responsive Design**: Works well on computers, tablets, and phones
- **Security**: Protects against common web attacks

Technical Details

Programming Languages Used:

- **PHP**: For server-side logic (the brain of the website)
- **HTML**: For webpage structure
- **CSS**: For making the website look attractive
- **JavaScript**: Used a bit interactive features
- **SQL**: For database operations

Technologies and Libraries:

- **Bootstrap**: Makes the website look modern and work on all devices
- **PHPQRCode**: Creates QR codes for tickets
- **MySQL**: Stores all data

Development Tools

Visual Studio Code

- **Purpose**: Primary code editor
- **Extensions Used**:
    - PHP IntelliSense
    - Laravel Blade Snippets

Git & GitHub & GitHub Desktop

- **Purpose**: Version control system
- **Benefits**:
    - Collaborative development
    - Version tracking
    - Branch management
    - Deployment automation

XAMPP/Laravel Valet

- **Purpose**: Local development environment
- **Components**:

    - Apache web server
    - MySQL database
    - PHP interpreter
    - phpMyAdmin

System Components

1. Authentication System

- User registration and login
- Password reset functionality
- Session management
- Role-based access control

2. Event Management System

- Event CRUD operations
- Image upload and storage
- Status management
- Search and filtering

3. Seat Management System

- Seat configuration
- Real-time availability tracking
- Reservation system
- Pricing management
- Visual seat mapping

4. Booking System

- Booking creation and management
- Confirmation system
- History tracking
- Modification handling

5. Communication System

- Email notification service
- Template management
- Queue processing
- Attachment handling

6. Reporting System

- Analytics dashboard
- Performance metrics
- User statistics
- Revenue tracking

Data Flow Architecture

1. User Request Flow

User Interface → Route → Controller → Service → Model → Database

2. Response Flow

Database → Model → Service → Controller → View → User Interface

3. Email Notification Flow

Booking Event → Queue → Mail Service → Email Template → SMTP → User

Database Design

Conceptual Design

The database design follows a normalized approach ensuring data integrity, reducing redundancy, and maintaining referential consistency. The schema supports complex relationships between entities while maintaining performance.

Entity Relationship Diagram

Primary Entities:

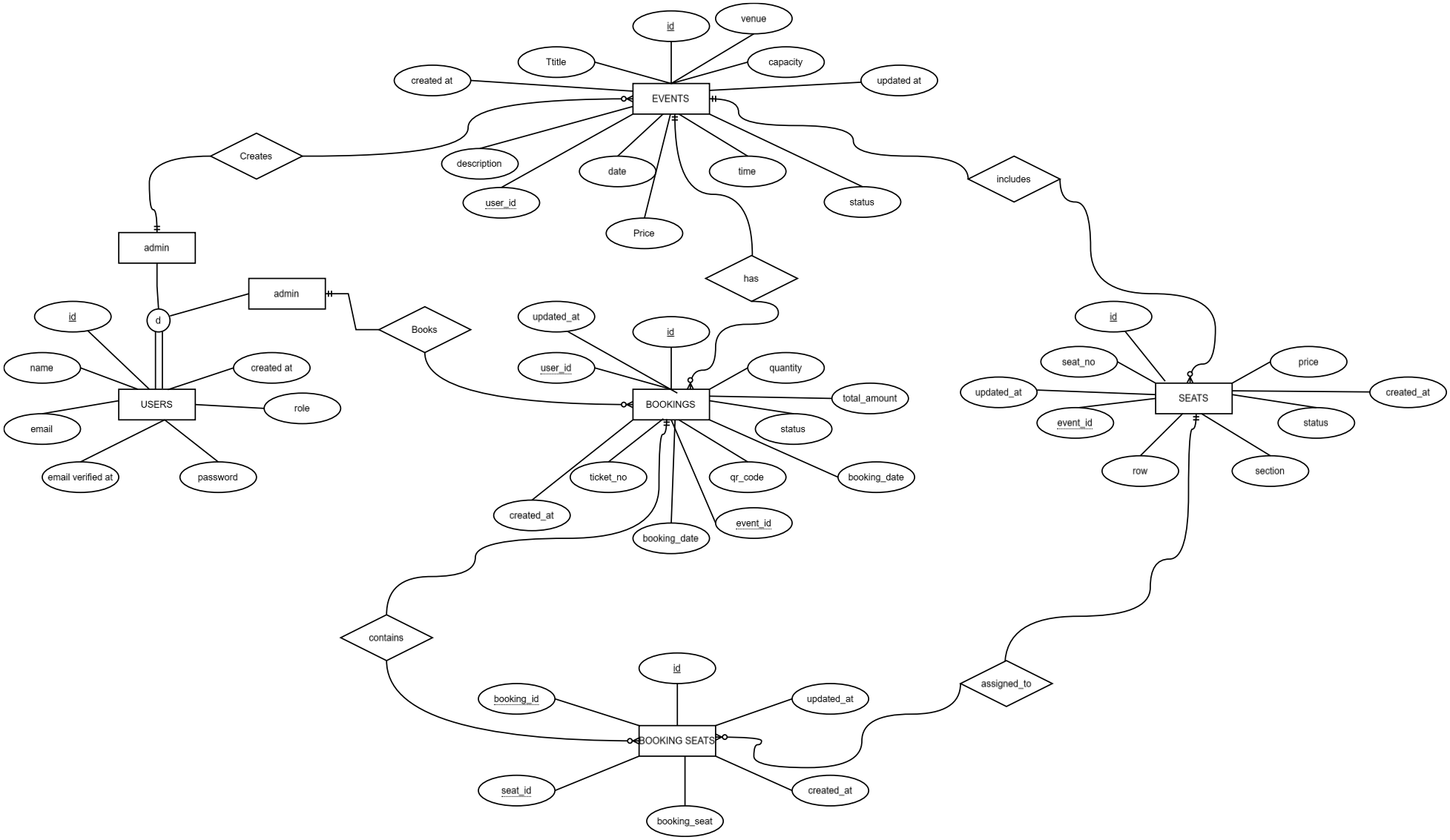
1. **Users** - System users (both regular users and administrators)
2. **Events** - Event information and management
3. **Seats** - Individual seat management for events
4. **Bookings** – Bookings details

Relationships:

- Users have many Events (one-to-many)
- Events have many Seats (one-to-many)
- Users have many Bookings (one-to-many)
- Events have many Bookings (one-to-many)

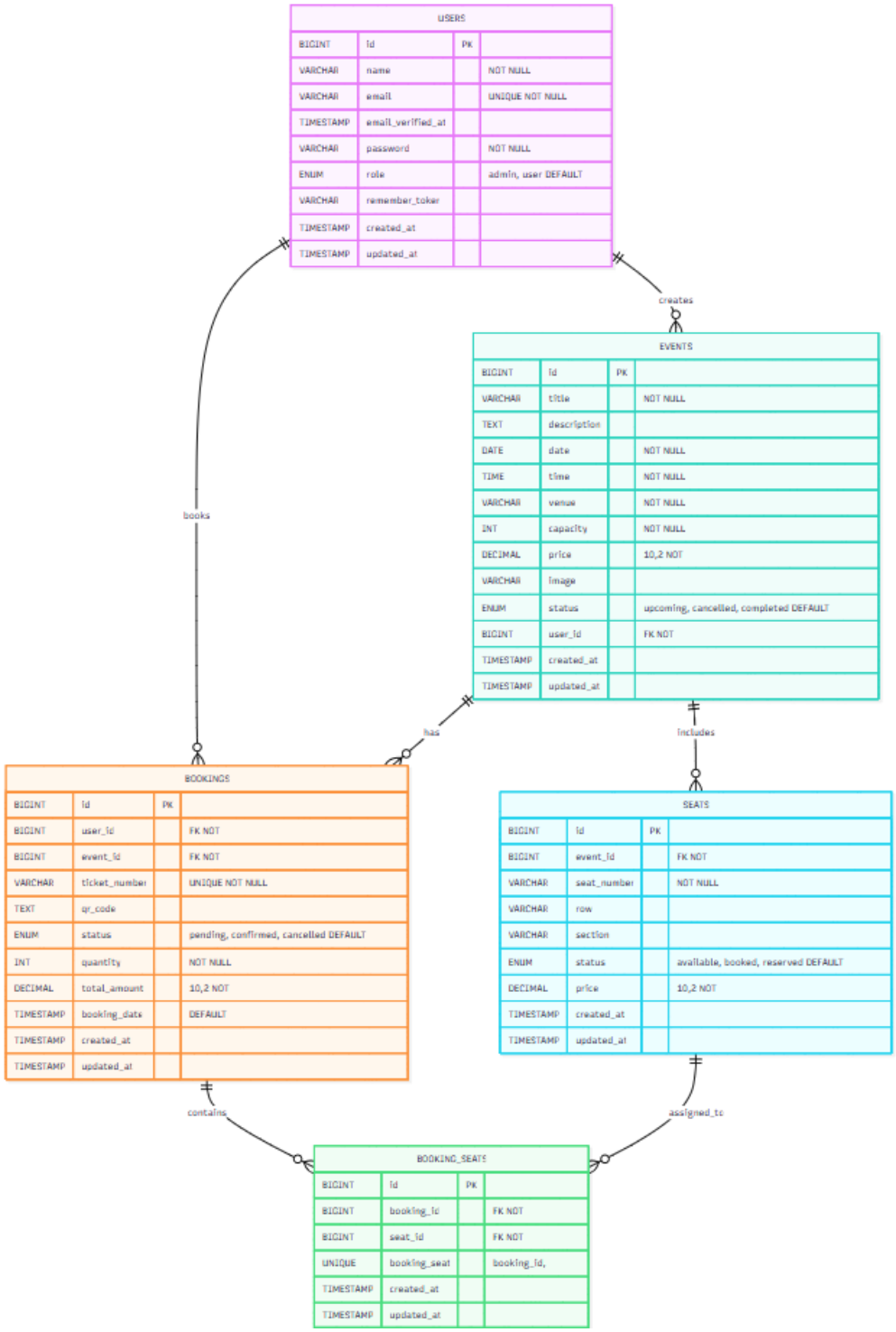
Finalized Conceptual Schema

Entity Relationship Diagram (ERD):



Enhanced ERD:

EERD includes additional attributes and relationships such as event ownership and booking status.



Business Rules:

- An admin can create multiple events.
- Each event is created and managed by one admin only.
- A customer can book a seat for an event without creating a user account.
- Each booking must be linked to one event.
- Each event must have a valid title, date, time, venue, capacity, and price before being listed.
- Events must have a status of either active, inactive, or cancelled. Only active events can be booked.

- The system must not allow seat capacity to be negative.
- Once all seats are booked or reserved, no new bookings can be made.
- Each seat is unique per event, identified by seat number.
- Seats must be marked as available, booked, or reserved. Booked seats cannot be double-booked.
- Each booking must generate a unique ticket number and QR code.
- The total amount for a booking is calculated based on the number of selected seats and seat price.
- Admins can edit or delete any event they created.
- When an event is deleted, all associated seats and bookings are also deleted.
- The system stores customer details such as name, email, and phone for every booking.
- A booking must include at least one seat.
- Only bookings with pending or confirmed status are considered active. Cancelled bookings release their seats.
- A user with role 'admin' can manage events. A user with role 'user' can only book events.
- Seats can have custom prices. If not set, the default event price is used.

Converted Relational Schema (from Conceptual Schema)

Based on the Entity Relationship Diagram (ERD), the conceptual schema has been converted into a relational schema and normalized up to **Third Normal Form (3NF)**. Each relation is designed to eliminate redundancy and maintain data integrity.

Relational Schema

1. **Admin**
   (admin_id, name, email, password, created_at)
   - Stores information about event administrators
   - Each admin can manage multiple events
2. **Event**
   (event_id, admin_id, title, description, date, time, venue, available_seats, price, created_at)
   - Represents individual events created by admins
   - admin_id is a foreign key referencing the Admin table
   - available_seats reflects remaining capacity
3. **Customer**
   (customer_id, name, email, phone)
   - Contains customer information for bookings
   - Supports booking without requiring full user registration
4. **Booking**
   (booking_id, event_id, customer_id, status, booking_date)
   - Links customers to events via bookings
   - event_id references the Event table
   - customer_id references the Customer table
   - status indicates current booking state (e.g., pending, confirmed)

Normalization to 3NF

To ensure the database is efficient, consistent, and free from redundancy, the schema was normalized to the **Third Normal Form (3NF)**. Below is the step-by-step normalization process based on the initial structure of event and booking data.

Step 1: Unnormalized Form (UNF)

In the unnormalized form, multiple data points such as seat numbers and customer information were stored together in a single record, resulting in redundancy and multivalued fields.

Example: BookingDetails(event_id, event_title, event_date, event_time, venue, admin_name, admin_email,

customer_name, customer_email, seat_number1, seat_number2, ..., status)

Step 2: First Normal Form (1NF)

In 1NF, all attributes must hold only **atomic** values, and repeating groups must be eliminated.

Revised Tables in 1NF:

- **users(user_id, name, email, password, role)**
- **events(event_id, title, date, time, venue, price, admin_id)**
- **seats(seat_id, event_id, seat_number, row, section, status, price)**
- **bookings(booking_id, event_id, user_id, status, total_amount, selected_seats)**

Note: The selected_seats field in bookings still violates 1NF as it stores multiple values in JSON format.

Step 3: Second Normal Form (2NF)

In 2NF, partial dependencies are removed. All non-key attributes must depend on the entire primary key. A new relation is introduced to handle the many-to-many relationship between bookings and seats.

Revised Tables in 2NF:

- **users(id, name, email, password, role)**
- **events(id, title, description, date, time, venue, price, capacity, user_id)**
- **seats(id, event_id, seat_number, row, section, status, price)**
- **bookings(id, event_id, user_id, ticket_number, qr_code, status, total_amount, booking_date)**
- **booking_seats(booking_id, seat_id)**

The booking_seats table replaces the JSON-based selected_seats field to ensure atomic values and proper foreign key relationships.

Step 4: Third Normal Form (3NF)

In 3NF, transitive dependencies are removed. All non-prime attributes must depend only on the primary key.

Final Relations in 3NF

1. **users**
   (id, name, email, password, role, email_verified_at, remember_token, created_at, updated_at)
2. **events**
   (id, title, description, date, time, venue, capacity, price, image, status, user_id, created_at, updated_at)
3. **seats**
   (id, event_id, seat_number, row, section, status, price, created_at, updated_at)
4. **bookings**
   (id, user_id, event_id, ticket_number, qr_code, status, quantity, total_amount, booking_date, created_at, updated_at)
5. **booking_seats** *(new table added for full normalization)*
   (booking_id, seat_id)

The database design has been successfully normalized to **Third Normal Form (3NF)** to ensure:

- Elimination of redundancy
- Improved data consistency and integrity
- Scalability and flexibility in querying booking and seating data

SQL Database Tables and Queries

Users Table

```
CREATE TABLE users (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  email_verified_at TIMESTAMP NULL,
  password VARCHAR(255) NOT NULL,
  role ENUM('user', 'admin') DEFAULT 'user',
  remember_token VARCHAR(100) NULL,
  created_at TIMESTAMP NULL,
  updated_at TIMESTAMP NULL
);
```

Purpose: Manages user authentication and authorization

Key Features:

- Unique email constraint
- Role-based access control
- Email verification support
- Remember token for persistent login
- Timestamps for audit trail

Events Table

```
CREATE TABLE events (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
```

```
    description TEXT NOT NULL,
    date DATE NOT NULL,
    time TIME NOT NULL,
    venue VARCHAR(255) NOT NULL,
    capacity INT NOT NULL,
    price DECIMAL(8,2) NOT NULL,
    image VARCHAR(255) NULL,
    status ENUM('active', 'inactive', 'cancelled') DEFAULT 'active',
    user_id BIGINT UNSIGNED NOT NULL,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Purpose: Stores comprehensive event information

Key Features:

- Event ownership through user_id
- Status management for event lifecycle
- Image support for event promotion
- Decimal precision for pricing
- Cascading delete for data integrity

Seats Table

```
CREATE TABLE seats (
    id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    event_id BIGINT UNSIGNED NOT NULL,
    seat_number VARCHAR(255) NOT NULL,
    row VARCHAR(255) NOT NULL,
    section VARCHAR(255) DEFAULT 'main',
    status ENUM('available', 'booked', 'reserved') DEFAULT 'available',
    price DECIMAL(8,2) NULL,
    created_at TIMESTAMP NULL,
    updated_at TIMESTAMP NULL,
    UNIQUE KEY unique_seat (event_id, seat_number),
    FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE
);
```

Purpose: Manages individual seat allocation and pricing **Key Features**:

- Unique seat identification per event
- Section-based organization
- Individual seat pricing capability
- Real-time status tracking

- Referential integrity with events

Bookings Table

```
CREATE TABLE bookings (
  id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  user_id BIGINT UNSIGNED NOT NULL,
  event_id BIGINT UNSIGNED NOT NULL,
  ticket_number VARCHAR(255) UNIQUE NOT NULL,
  qr_code TEXT NOT NULL,
  status ENUM('pending', 'confirmed', 'cancelled') DEFAULT 'pending',
  quantity INT DEFAULT 1,
  selected_seats JSON NULL,
  total_amount DECIMAL(10,2) DEFAULT 0.00,
  booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  created_at TIMESTAMP NULL,
  updated_at TIMESTAMP NULL,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE
);
```

Purpose: Manages booking transactions and ticket generation

Key Features:

- Unique ticket number generation
- QR code storage for digital tickets
- JSON storage for flexible seat selection
- Comprehensive booking status tracking

METADATA

Table: users

Column Name	*Data Type*	*Constraints*	*Key Type*
id	BIGINT UNSIGNED	AUTO_INCREMENT, NOT NULL	PRIMARY KEY
name	VARCHAR(255)	NOT NULL	–
email	VARCHAR(255)	NOT NULL, UNIQUE	UNIQUE
email_verified_at	TIMESTAMP	NULL	–
password	VARCHAR(255)	NOT NULL	–
role	ENUM('user','admin')	DEFAULT 'user'	–
remember_token	VARCHAR(100)	NULL	–
created_at	TIMESTAMP	NULL	–
updated_at	TIMESTAMP	NULL	–

Table: events

Column Name	*Data Type*	*Constraints*	*Key Type*
id	BIGINT UNSIGNED	AUTO_INCREMENT, NOT NULL	PRIMARY KEY
title	VARCHAR(255)	NOT NULL	–
description	TEXT	NOT NULL	–
date	DATE	NOT NULL	–
time	TIME	NOT NULL	–
venue	VARCHAR(255)	NOT NULL	–
capacity	INT	NOT NULL	–
price	DECIMAL(8,2)	NOT NULL	–
image	VARCHAR(255)	NULL	–
status	ENUM('active','inactive','cancelled')	DEFAULT 'active'	–
user_id	BIGINT UNSIGNED	NOT NULL	FOREIGN KEY → users(id)
created_at	TIMESTAMP	NULL	–
updated_at	TIMESTAMP	NULL	–

Table: seats

Column Name	*Data Type*	*Constraints*	*Key Type*
id	BIGINT UNSIGNED	AUTO_INCREMENT, NOT NULL	PRIMARY KEY
event_id	BIGINT UNSIGNED	NOT NULL	FOREIGN KEY → events(id)
seat_number	VARCHAR(255)	NOT NULL	UNIQUE (event_id, seat_number)
row	VARCHAR(255)	NOT NULL	–
section	VARCHAR(255)	DEFAULT 'main'	–

status	ENUM('available','booked','reserved')	DEFAULT 'available'	–
price	DECIMAL(8,2)	NULL	–
created_at	TIMESTAMP	NULL	–
updated_at	TIMESTAMP	NULL	–

Table: bookings

Column Name	*Data Type*	*Constraints*	*Key Type*
id	BIGINT UNSIGNED	AUTO_INCREMENT, NOT NULL	PRIMARY KEY
user_id	BIGINT UNSIGNED	NOT NULL	FOREIGN KEY → users(id)
event_id	BIGINT UNSIGNED	NOT NULL	FOREIGN KEY → events(id)
ticket_number	VARCHAR(255)	NOT NULL, UNIQUE	UNIQUE
qr_code	TEXT	NOT NULL	–
status	ENUM('pending','confirmed','cancelled')	DEFAULT 'pending'	–
quantity	INT	DEFAULT 1	–
selected_seats	JSON	NULL	–
total_amount	DECIMAL(10,2)	DEFAULT 0.00	–
booking_date	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	–
created_at	TIMESTAMP	NULL	–
updated_at	TIMESTAMP	NULL	–

✅ KEYS Summary

Primary Keys

- users.id
- events.id
- seats.id
- bookings.id

Foreign Keys

- events.user_id → users.id
- seats.event_id → events.id
- bookings.user_id → users.id
- bookings.event_id → events.id

Unique Keys

- users.email
- bookings.ticket_number
- seats.event_id + seat_number (Composite Unique Constraint)

DATABASE QUERIES

Query 1: Select all events

```
SELECT * FROM events;
```

Query 2: Register a new user

```
INSERT INTO users (name, email, password)
VALUES ('Hassan Zaib', 'hassan@example.com', 'hashed_password');
```

Query 3: Book an event

```
INSERT INTO bookings (user_id, event_id, ticket_number, qr_code, status, quantity, total_amount)
VALUES (2, 5, 'TKT10005', 'QR_DATA_HERE', 'pending', 2, 2000.00);
```

Query 4: Admin login verification

```
SELECT * FROM users
WHERE email = 'admin@example.com'
AND password = 'hashed_password'
AND role = 'admin';
```

Query 5: Fetch all available seats for a specific event

```
SELECT * FROM seats
WHERE event_id = 5
AND status = 'available';
```

Query 6: Update seat status after booking

```
UPDATE seats
SET status = 'booked'
WHERE id IN (10, 11, 12);
```

Query 7: Get all bookings for a specific user

```
SELECT * FROM bookings
WHERE user_id = 2;
```

Query 8: Cancel a booking

```
UPDATE bookings
SET status = 'cancelled'
WHERE id = 7;
```

Query 9: View event details along with admin info

```
SELECT events.*, users.name AS admin_name, users.email AS admin_email
FROM events
JOIN users ON events.user_id = users.id;
```

Query 10: Delete an event (admin-only operation)

```
DELETE FROM events
WHERE id = 4
AND user_id = 1;
```

Implementation Details

Development Methodology

The project follows Agile development methodology with iterative development cycles, continuous integration, and test-driven development practices.

Development Phases:

1. **Planning Phase**: Requirements gathering and system design
2. **Development Phase**: Feature implementation and testing
3. **Testing Phase**: Comprehensive testing and bug fixing
4. **Deployment Phase**: Production deployment and monitoring
5. **Maintenance Phase**: Ongoing updates and improvements

Architecture Pattern:

The system follows **MVC (Model-View-Controller)** architectural pattern:

Model Layer:

- Database interaction through MySQL
- Data validation and sanitization
- Business logic implementation

View Layer:

- HTML templates with embedded PHP
- Bootstrap-based responsive design
- JavaScript for dynamic interactions

Controller Layer:

- PHP scripts handling requests
- Session management
- Input validation and processing

Key Implementation Features:

1. Database Connection Management
2. Prepared Statements for Security
3. Transaction Management
4. QR Code Generation
5. Email Automation
6. Input Validation and Sanitization

7. Session Management
8. Error Handling

How the System Works

1. User Interaction Flow

Step 1: Website Visit
The user visits the event booking platform, which displays a list of available events and options to **log in** or **register**.

Step 2: Registration (for New Users)
If the user is new, they click the **Register** button and fill out a form with their full name, email address, and password.
After successful registration, the user is automatically logged into the system.

Step 3: Login (for Existing Users)
If the user already has an account, they click the **Login** button and enter their credentials.
Upon successful login, they are redirected to their personal **Dashboard**.

2. User Dashboard Experience

- The **Dashboard** displays the user's recent and upcoming bookings.
- If no bookings exist, the booking section remains empty.
- From the Dashboard, users can navigate to the **Events** section to browse available events.

3. Viewing and Booking an Event

Step 4: Browsing Events
Users can explore all active events and view detailed information such as:

- Event title
- Description
- Date and time
- Venue
- Ticket price
- Total and available seats

Step 5: Selecting Seats
After clicking **"Book Now"**, the user is shown seat categories (e.g., **Balcony**, **Main**, **Normal**).
They select their desired seats and proceed to confirm the booking.

Step 6: Submitting the Booking
The system validates the booking request and updates the seat status to **booked**.
A booking record is stored, and the system generates a **unique ticket number** and **QR code** for entry.

4. Booking Confirmation and Notification

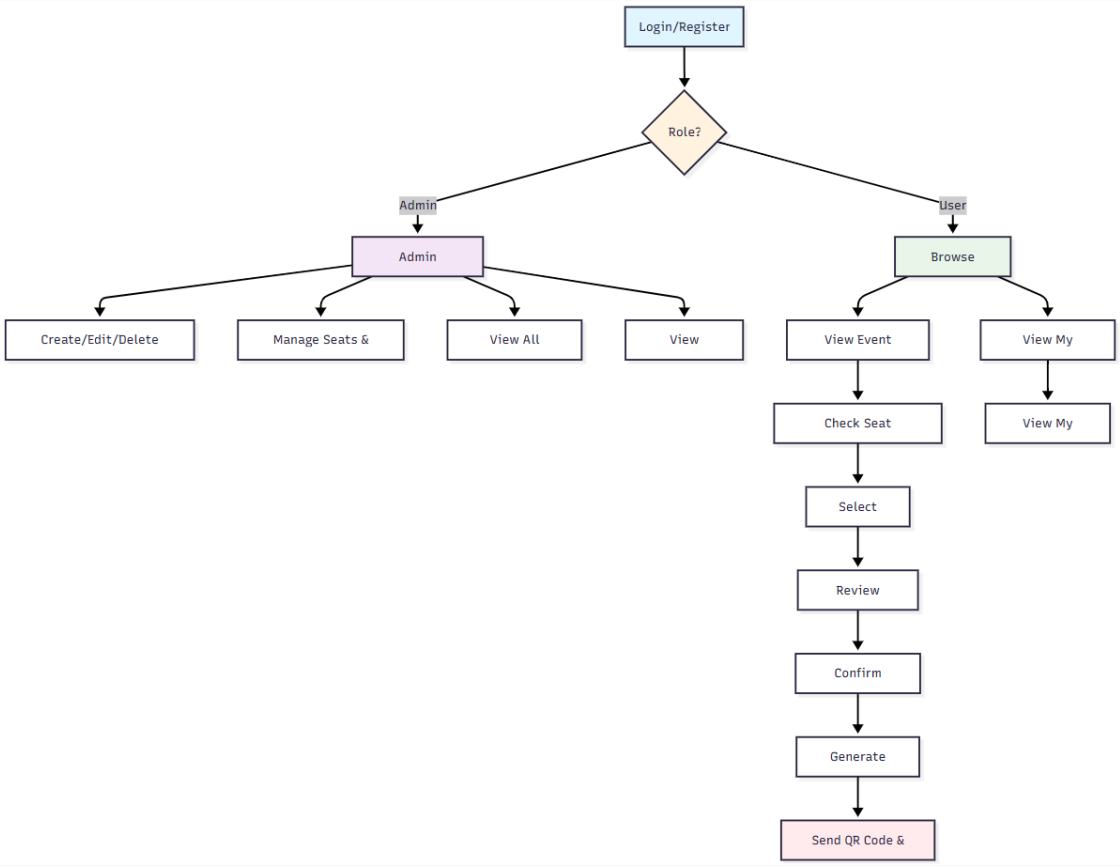
- A **confirmation page** is displayed to the user, showing ticket details and allowing them to **download the digital ticket**.

- A **confirmation email** is automatically sent to the user's **registered Gmail address**. This email includes the event details, seat information, ticket number, and QR code for scanning at the venue.

Behind the Scenes (System Logic)

1. **User Authentication**
   The system verifies login credentials and securely manages user sessions.
2. **Event & Seat Validation**
   The system checks seat availability and ensures the event exists.
3. **Transaction Processing**
   The system processes the booking, reserves the seats, and stores booking details in the database.
4. **QR Code Generation**
   A unique QR code is created for each booking to ensure secure event entry.
5. **Email Notification**
   Upon booking confirmation, an email is automatically sent to the user's registered Gmail account containing their ticket and event details.

Flow Chart Diagram:



User Interface

Design Features:

- **Clean and Modern**: Uses attractive colors and fonts
- **Easy Navigation**: Simple menu and clear buttons
- **Mobile Friendly**: Works well on phones and tablets
- **Visual Icons**: Uses symbols to make information easy to understand

Security Features

Protection Against Attacks:

1. **SQL Injection Prevention**: Uses prepared statements to prevent database attacks
2. **Input Validation**: Checks all user input for safety
3. **Data Sanitization**: Cleans user input before storing
4. **Session Management**: Secure handling of user sessions

Data Protection:

- **Form Validation**: Ensures users enter correct information
- **Error Handling**: Safely manages errors without exposing system details
- **Database Security**: Protects against unauthorized access

User Safety:

- **Safe Data Display**: Prevents malicious code in user content
- **Secure Booking Process**: Ensures booking integrity
- **Transaction Safety**: Uses database transactions for reliable booking

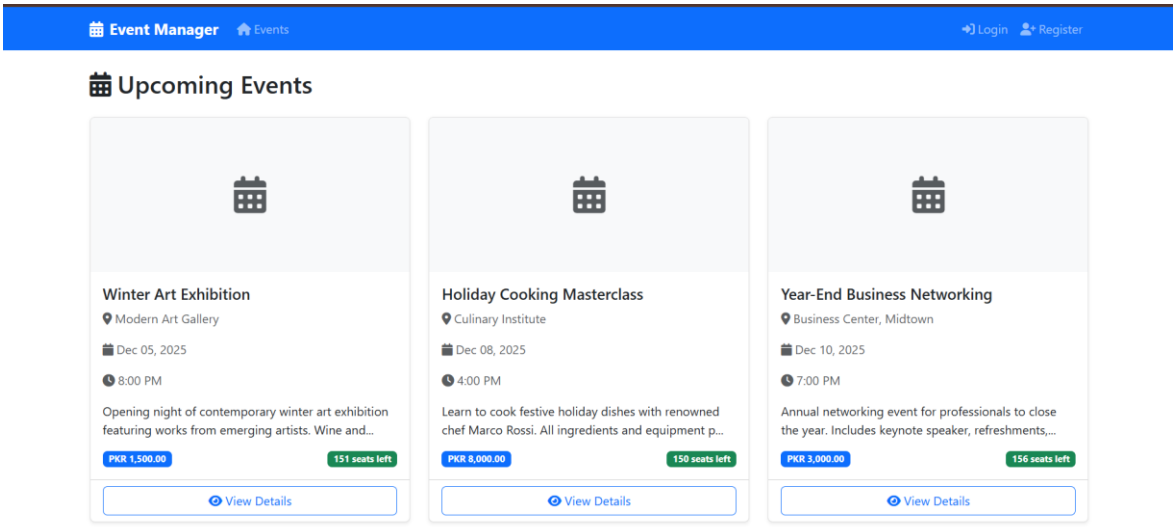
Learning Outcomes:

This project demonstrates knowledge of:

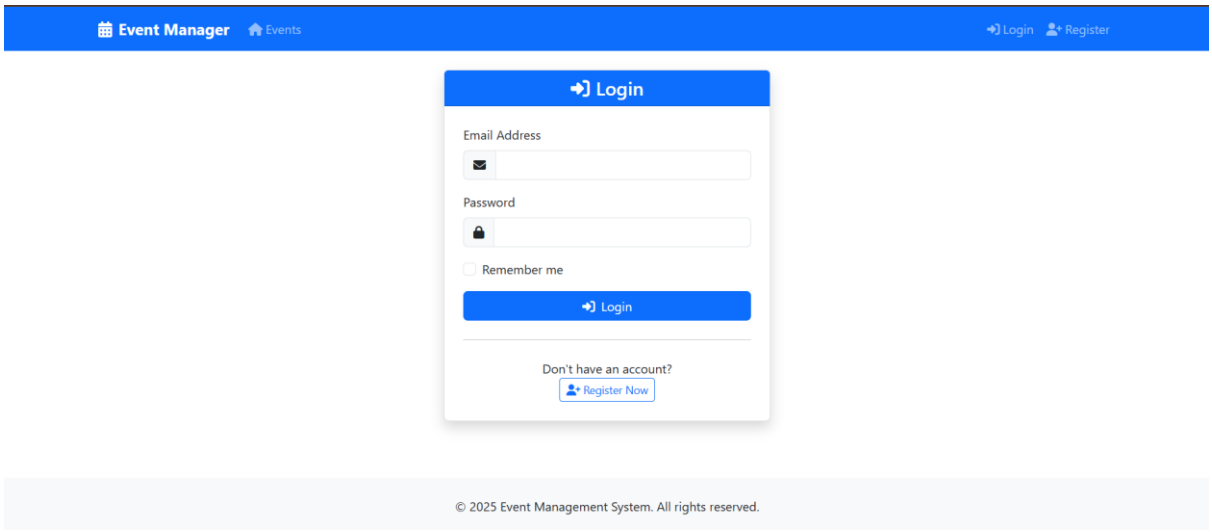
- Web development using PHP and MySQL
- Database design and management
- User interface design principles
- Security best practices
- Modern web technologies

The Event Booking System is a complete, functional web application that can be used in real-world scenarios for managing event bookings efficiently and securely.

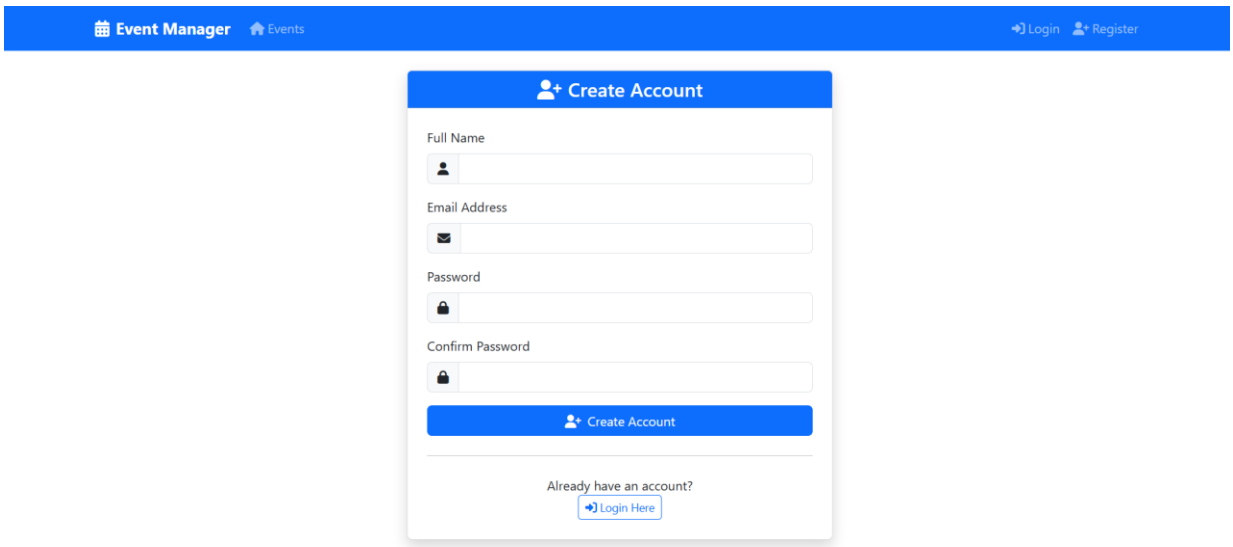
Homepage:



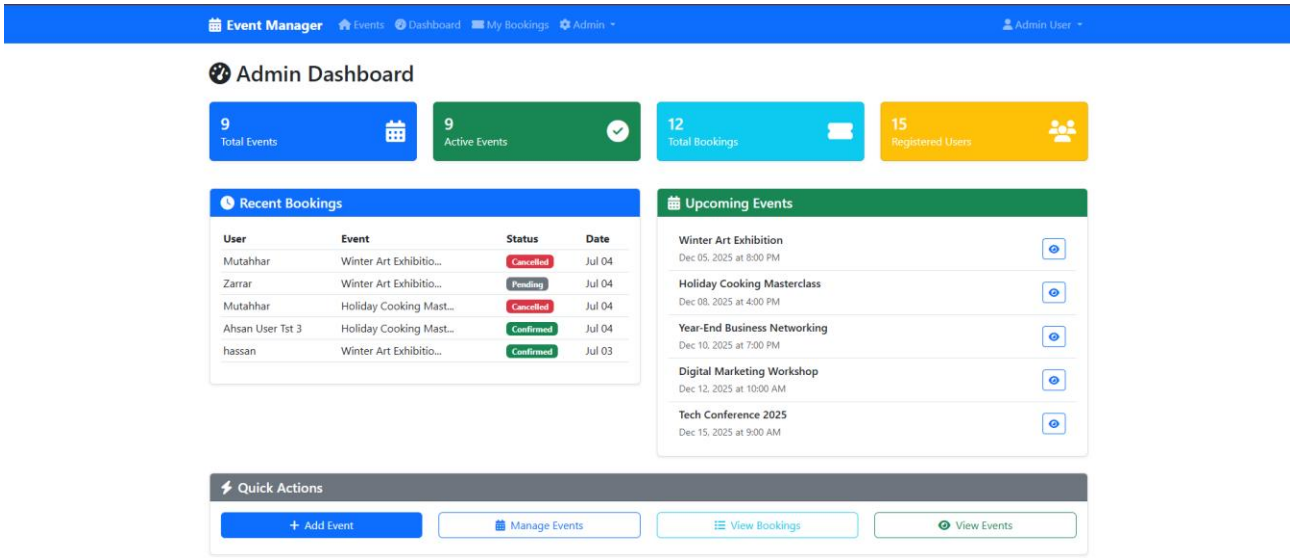
Login Page:



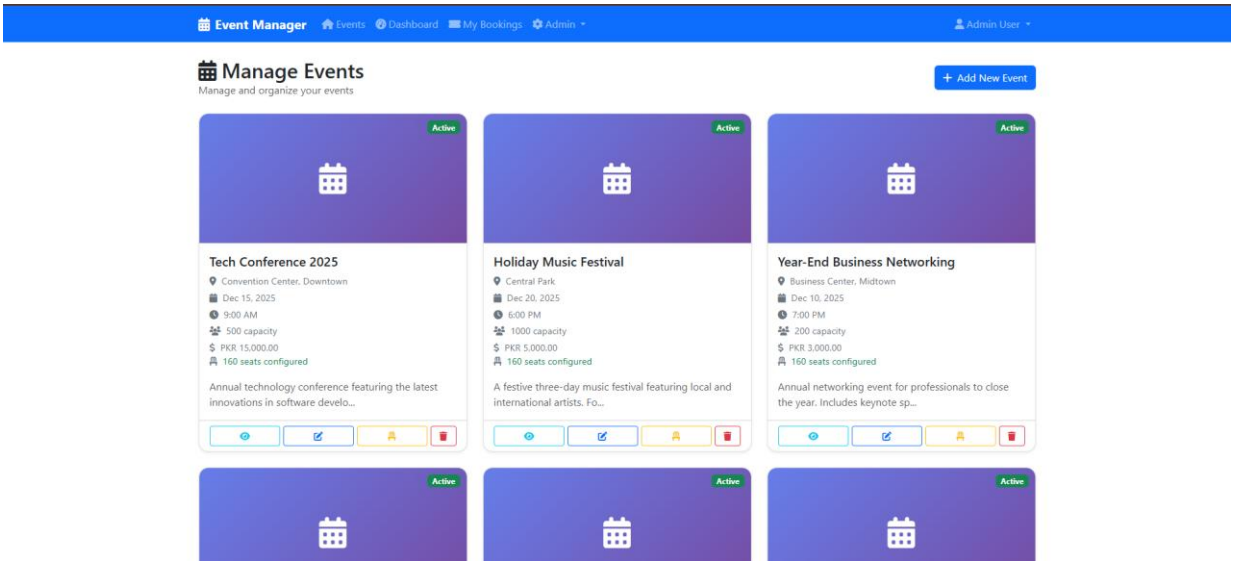
Registration Page:



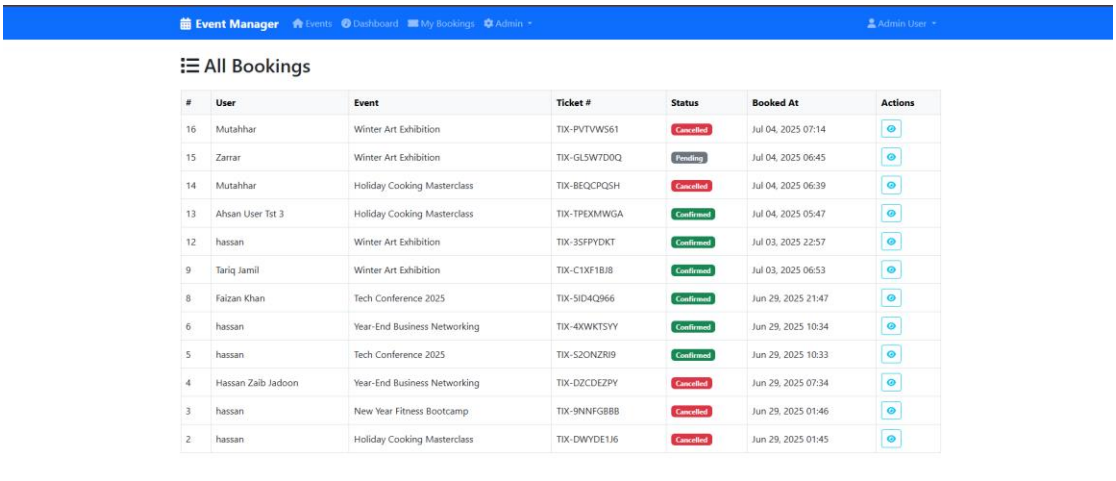
Admin Dashboard:



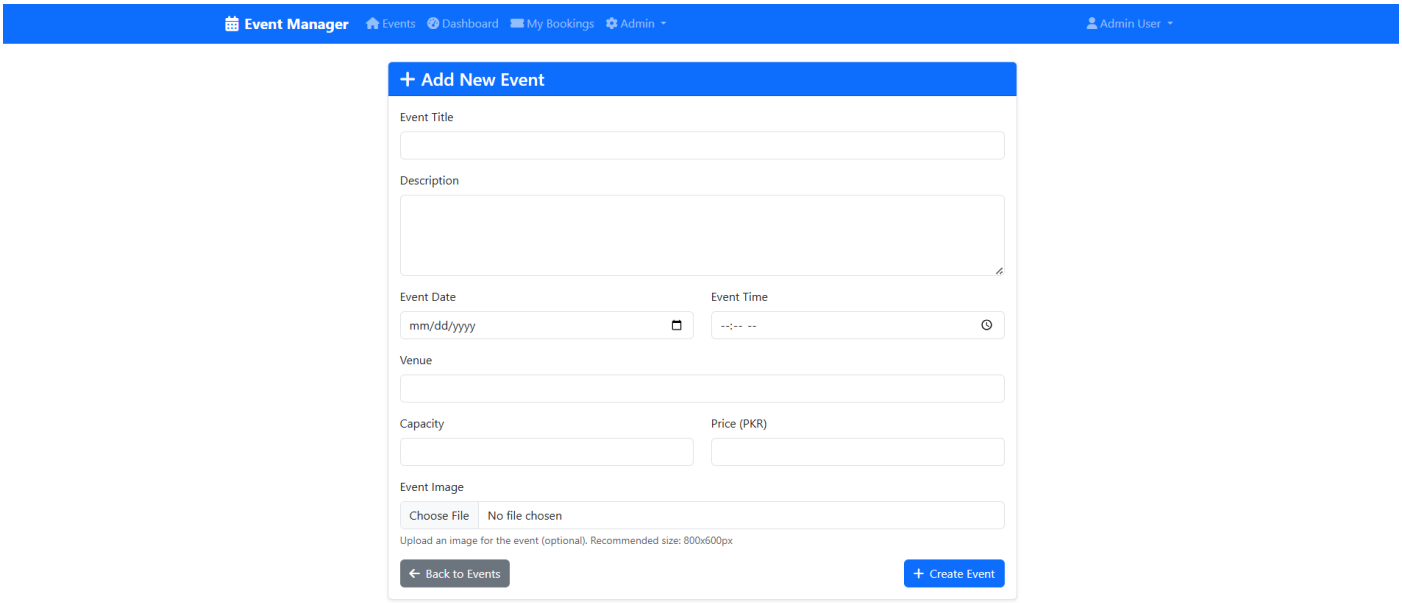
Manage Events Page:



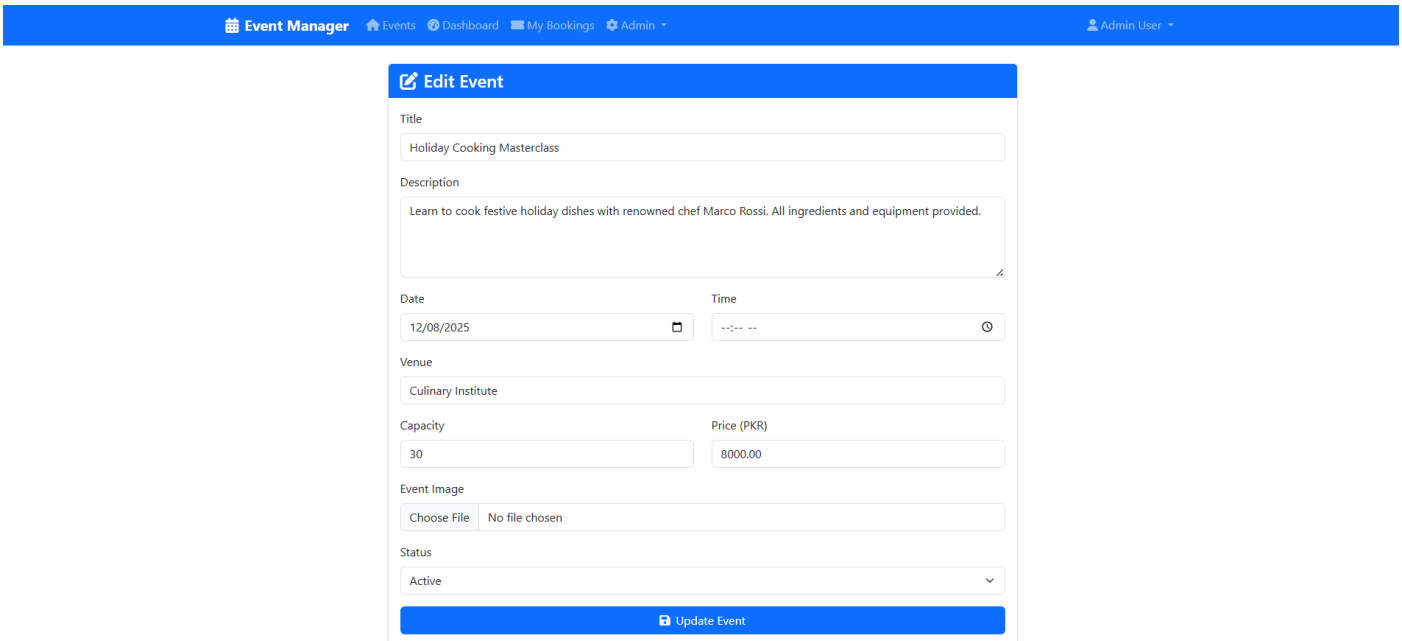
All Bookings Page:



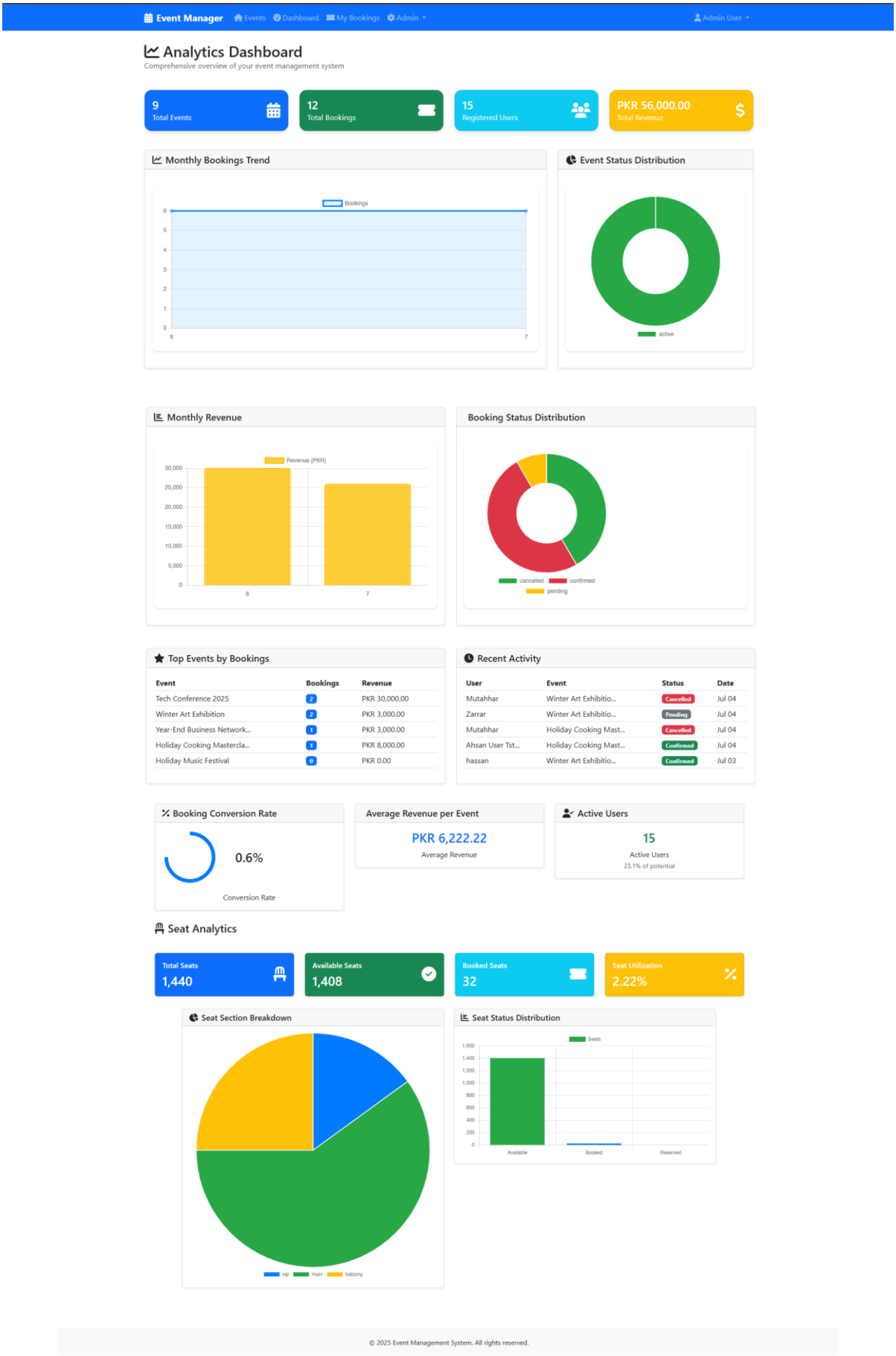
Creating New Event:



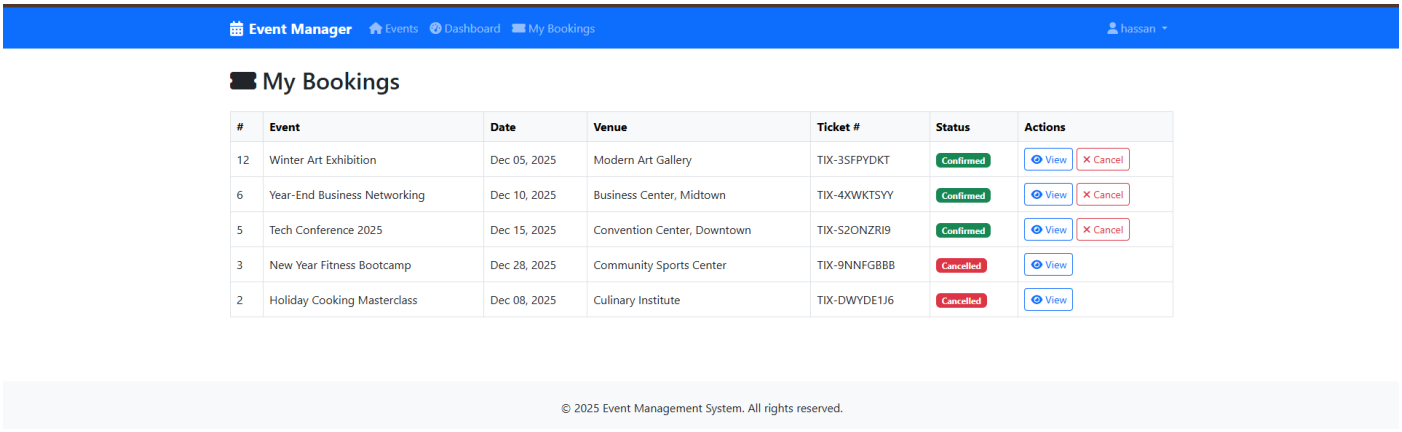
Editing an Event:



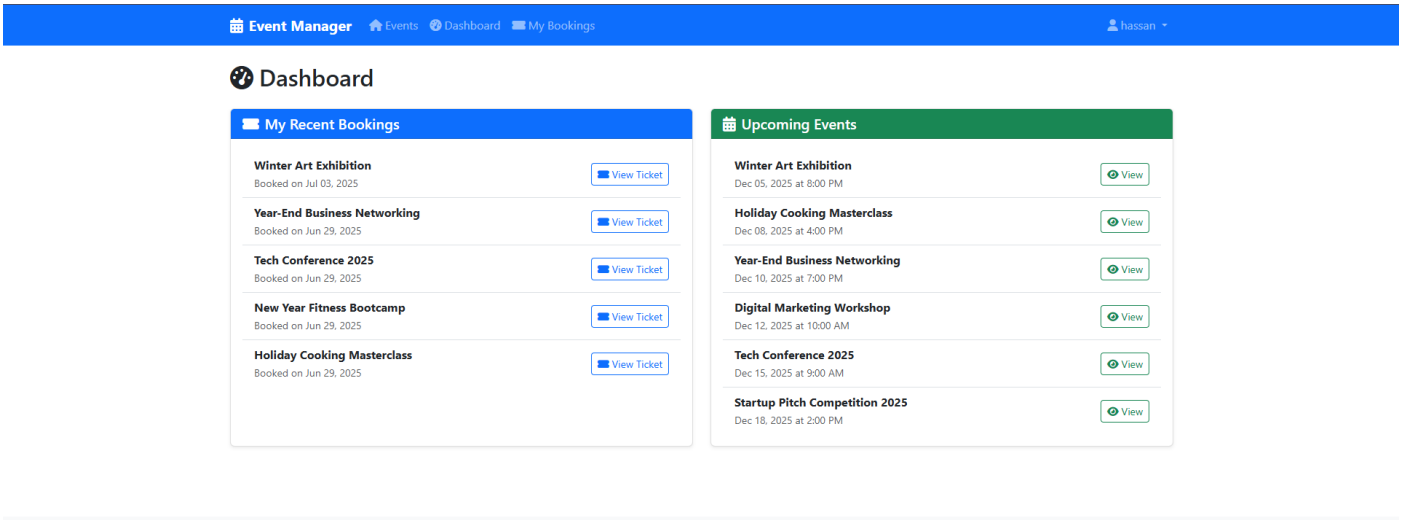
Analytics Dashboard:



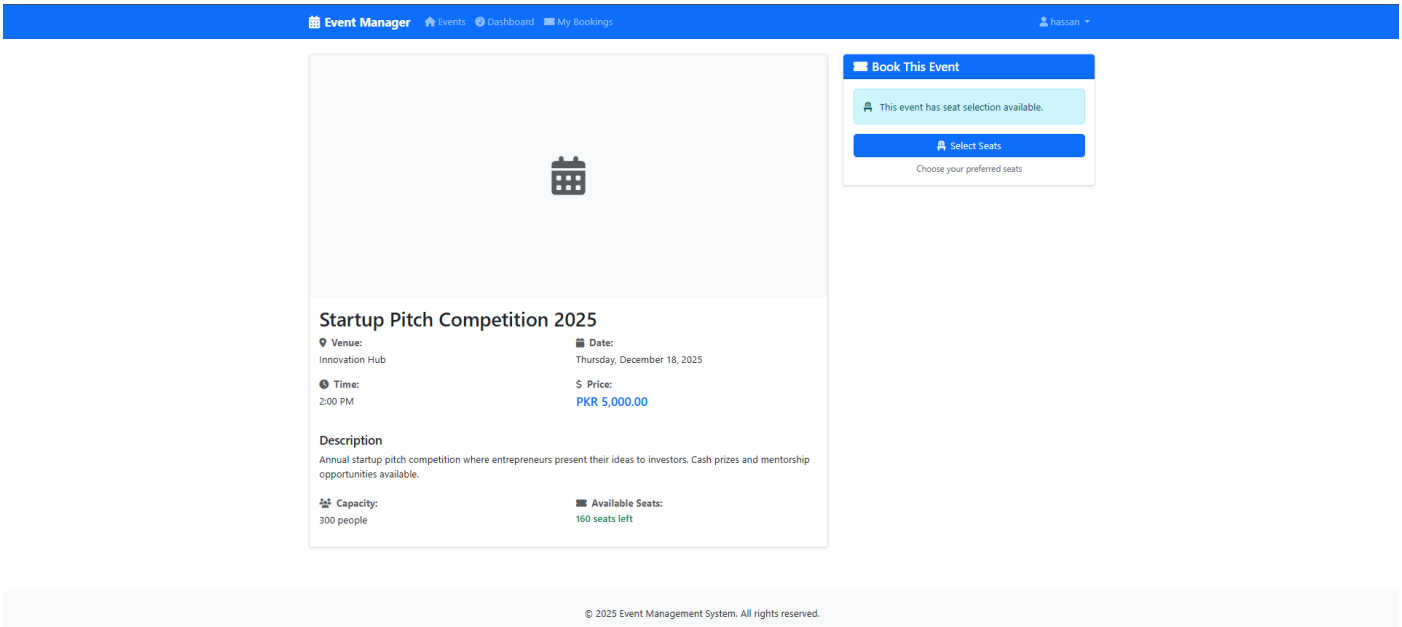
User My Booking Page:



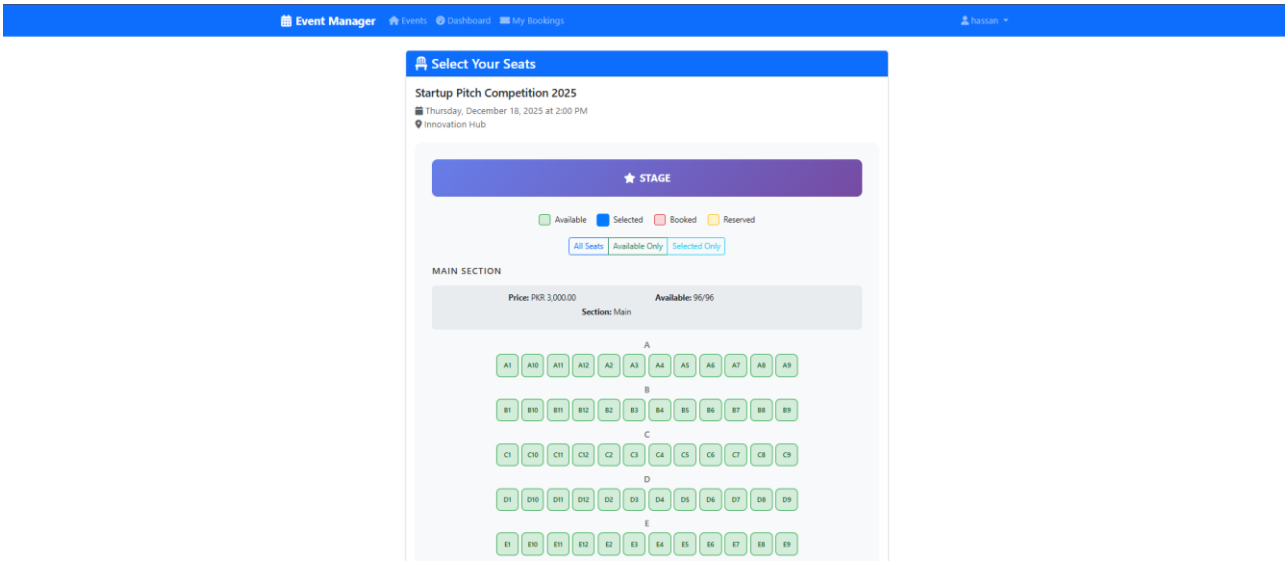
User Dashboard:



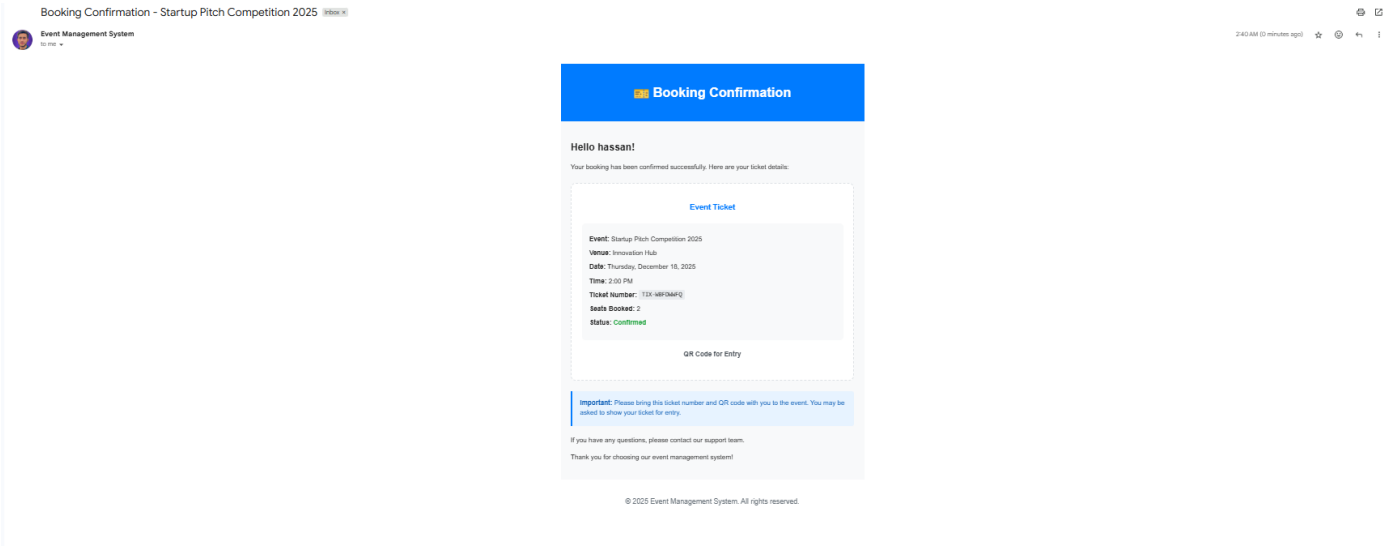
Event Booking Page:



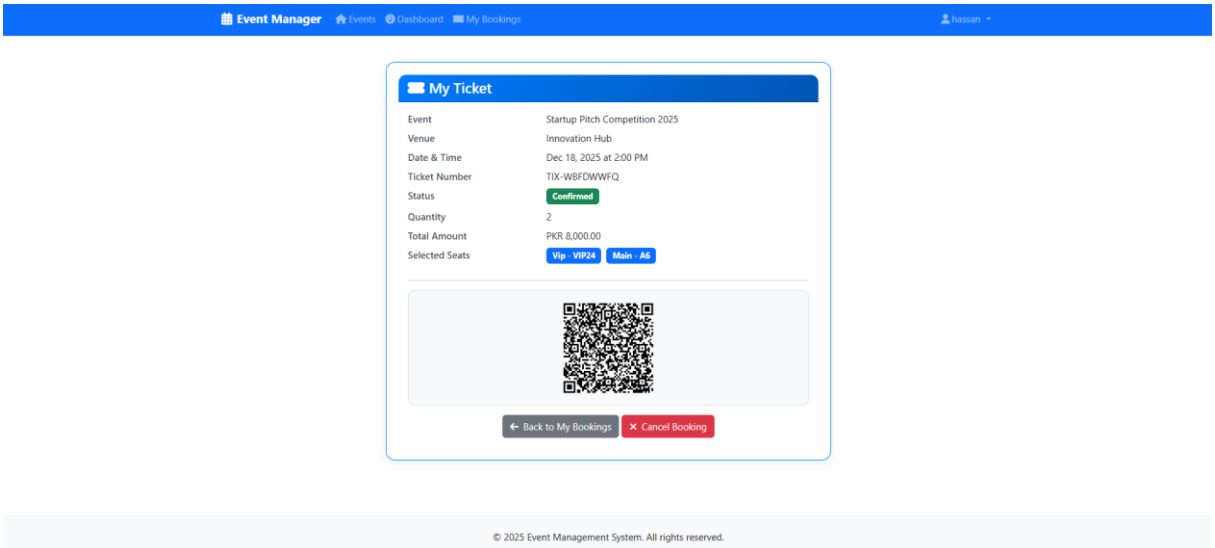
Seat Selection Page:



Confirmation Email:



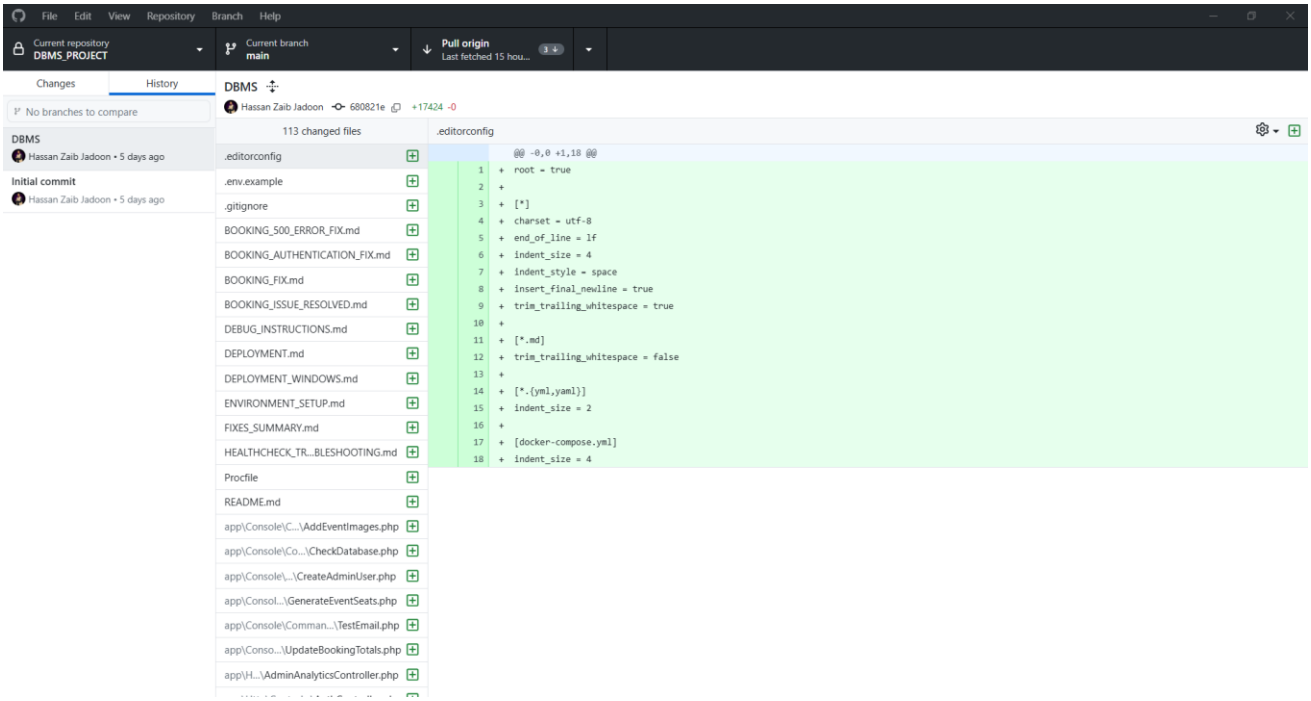
Qr Code:



6. DEPLOYMENT METHODOLGY

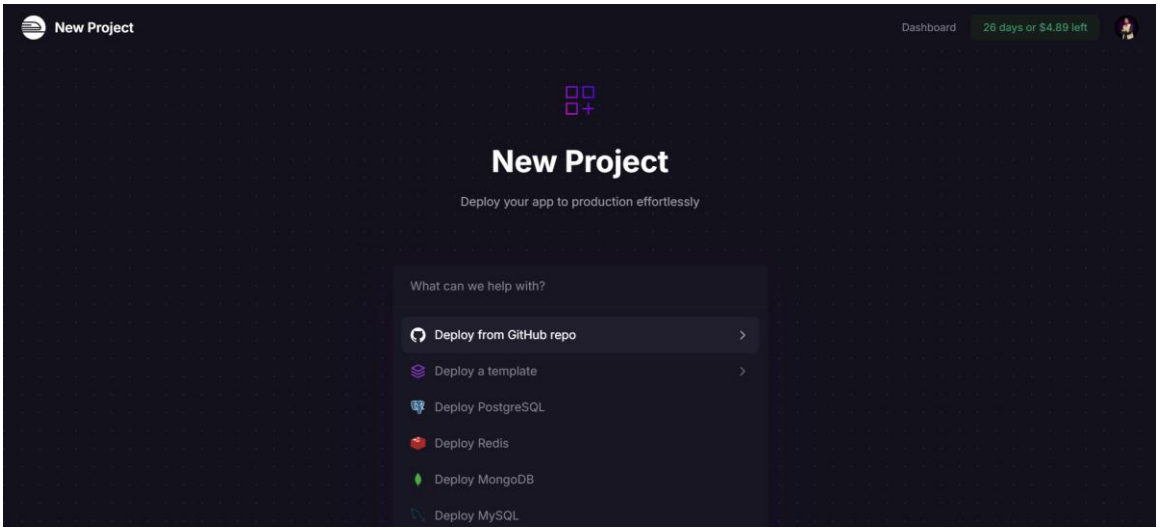
1. Push your project to Github:

- Create a private GitHub repository for your Event Management System project
- Initialize Git in your local project directory and add the remote repository
- Push your complete to the main branch
- Ensure all sensitive files like .env are included in .gitignore to maintain security



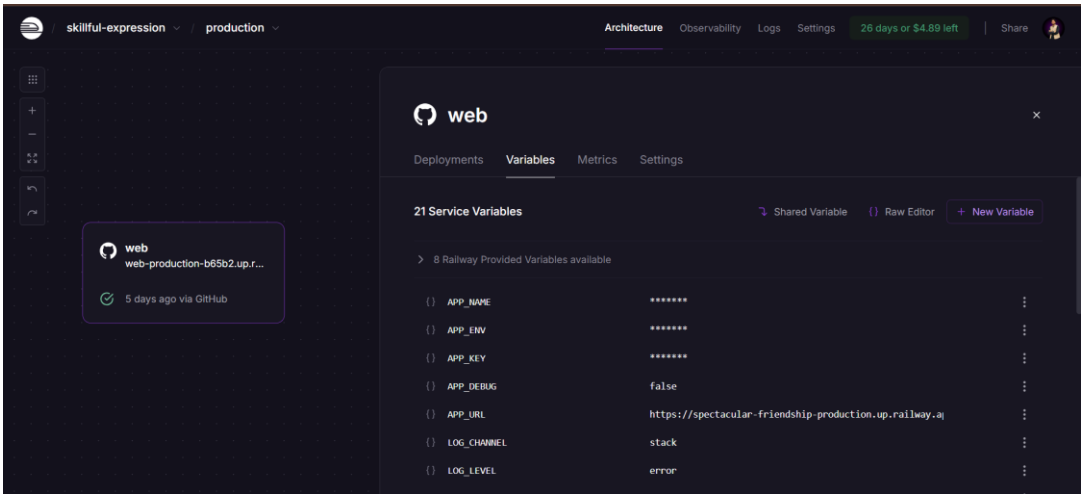
2. Create Railway Project from GitHub Repo

- Navigate to Railway.app and sign in with your GitHub account
- Click "New Project" from the dashboard and select "Deploy from GitHub"
- Authorize Railway to access your GitHub repositories
- Select your Event Management System repository and choose the main branch for deployment
- Railway will automatically detect your Laravel application and begin the setup process



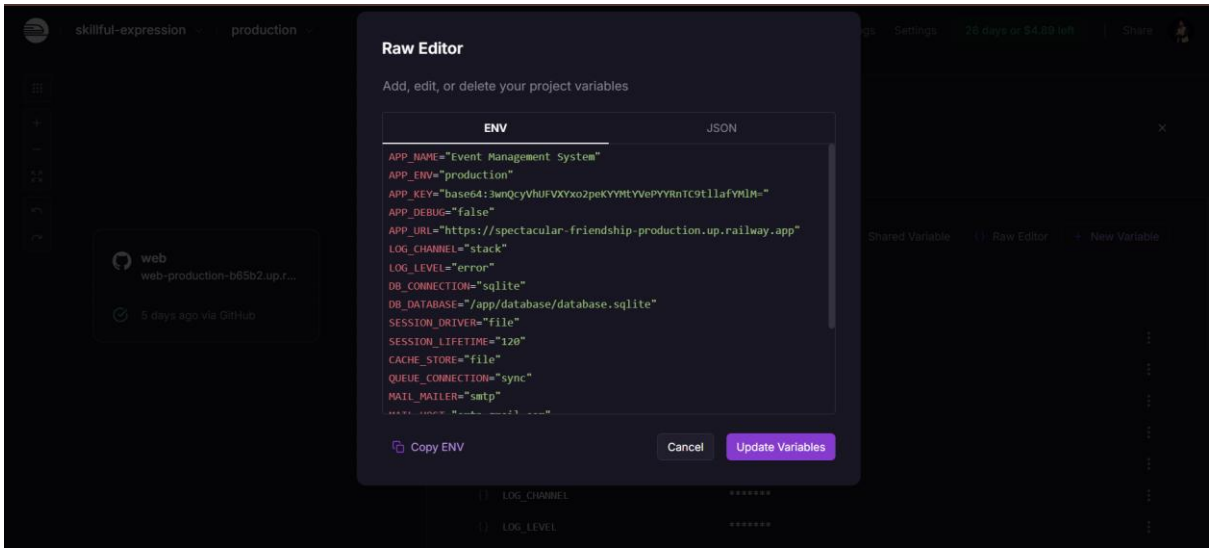
3. Provision the MySQL Database

- From your Railway project dashboard, click "New Service" and select "Database"
- Choose MySQL from the available database options
- Railway will automatically provision a new MySQL database instance
- The database connection details (host, port, username, password) will be generated automatically
- Note down these credentials as they will be needed for environment configuration



4. Update Environment Variables

- Navigate to the "Variables" tab in your Railway project settings
- Copy the Railway-generated database credentials including:
  - DB_HOST: The database server hostname
  - DB_PORT: The port number for database connection
  - DB_DATABASE: The database name (usually "railway")
  - DB_USERNAME: Database username (typically "root")
  - DB_PASSWORD: Auto-generated secure password
- Add additional Laravel environment variables such as APP_KEY, APP_ENV, and APP_DEBUG
- Configure frontend environment variables for API connection
- Railway automatically injects these variables at runtime, eliminating the need for local .env files



5. Deploy the Application

- Return to the "Deployments" tab in your Railway dashboard
- Click "Deploy" to manually trigger a deployment, or simply push new commits to your main branch
- Railway will automatically:
    - Pull the latest code from your GitHub repository
    - Build your Laravel application using the detected framework
    - Install dependencies using Composer for backend and npm for frontend
    - Run database migrations to set up your table structure
    - Launch your Event Management System on a live URL
- Monitor the deployment logs to ensure successful completion
- Test the application functionality once deployment is complete

Deployment Links:

- **Website Link:** https://event-management-system-production-b4b8.up.railway.app/events
    - Main application URL where users can access the Event Management System

- **MySQL Database:** Login to Railway using CLI and run this command:

  mysql://root:xKp9mDaNPNTzCubkMSGQXaJvjMHItuVl@containers-us-west-1.railway.app:7834/railway

    - Direct database connection string for administrative access
    - Can be used with MySQL client tools or command line interface
- **Admin Panel:** https://event-management-system-production-b4b8.up.railway.app/login
    - Administrative interface for managing events, users, and bookings
    - Restricted access requiring admin credentials
- **Admin Credentials:**
    - **Email:** admin@example.com
    - **Password:** ahsan
    - Default admin account for system management and testing
- **User Registration:** https://event-management-system-production-b4b8.up.railway.app/register
    - Public registration page for new users to create accounts

Additional Configuration:

Frontend Environment Variables:

env

```
VITE_API_URL=https://event-management-system-production-12a4.up.railway.app/api
VITE_APP_NAME=Event Management System
```

- VITE_API_URL: Connects frontend to backend API endpoints
- VITE_APP_NAME: Application name displayed in the user interface

Backend Environment Variables:

env

APP_NAME=EventManagementSystem

APP_ENV=production

APP_KEY=base64:generated_key_here

APP_DEBUG=false

APP_URL=https://event-management-system-production-12a4.up.railway.app

DB_CONNECTION=mysql

DB_HOST=containers-us-west-1.railway.app

DB_PORT=7834

DB_DATABASE=railway

DB_USERNAME=root

DB_PASSWORD=xKp9mDaNPNTzCubkMSGQXaJvjMHItuVl

- Laravel application configuration for production environment
- Database connection parameters provided by Railway MySQL service
- Security settings optimized for live deployment

Challenges Faced and Solutions

Challenge	Solution Implemented
Designing a normalized database structure	Applied 1NF, 2NF, and 3NF techniques for clean and scalable schema
Managing seat selection with real-time availability	Used a seats table with status tracking (available, booked, etc.)
Handling dynamic QR code generation	Implemented text-based placeholders; future implementation can include QR libraries
Limited time for payment gateway integration	Bookings are confirmed without actual transactions; placeholder for future integration
User email notifications	Simulated sending confirmation to Gmail accounts; integration possible with SMTP or third-party services
UI/UX limitations without frontend framework	Basic design was created using AI-based tools like Stitch for simplicity

Feature Enhancements (Future Scope)

The system can be enhanced in the future with the following features:

- **Payment Gateway Integration**: Integrate services like Stripe, JazzCash, or PayPal to allow secure ticket purchases.
- **Email Notification System**: Set up a live email system using services like Mailgun, Gmail SMTP, or SendGrid.
- **Admin Panel Dashboard**: More detailed analytics and booking insights for event creators.
- **PDF Ticket Generation**: Automatically generate downloadable PDF tickets with event and QR code details.
- **Mobile Responsiveness**: Enhance frontend layout to support mobile users and cross-device compatibility.

- **User Reviews & Ratings**: Allow users to rate events and share feedback post-attendance.

Final Note

This project provided hands-on experience in designing and implementing a functional database-driven system, applying real-world constraints and workflow logic. It also demonstrated the collaborative role of AI and cloud tools in modern development processes. Challenges were met with strategic solutions, and the system offers a strong foundation for future enhancements.

Conclusion

The Event Booking System successfully achieves its goal of providing an easy-to-use platform for booking event tickets online. The system offers:

Key Achievements:

- **User-Friendly Interface**: Simple and attractive design that anyone can use
- **Reliable Functionality**: Secure and efficient booking process
- **Modern Features**: QR code tickets and responsive design
- **Robust Security**: Protection against common web vulnerabilities

Benefits Delivered:

- Makes ticket booking convenient for users
- Reduces manual work for event organizers
- Provides digital solution for ticket management
- Offers professional appearance for events

References

The following resources and tools were used during the research, design, and development phases of this project, adhering to best practices in academic and technical writing:

1. Anthropic. *Claude AI*. Large Language Model used for conceptual brainstorming and content structuring.
2. OpenAI. *ChatGPT*. Used for technical validation, SQL query optimization, and documentation formatting.
3. Mermaid.js (Mermister). Used for drawing ER diagrams, flowcharts, and visual process representation.
4. Stitch Designer by Google. Used to create a simple UI/UX mockup and layout structure for the website.
5. MySQL Documentation. For syntax validation and understanding constraints, relationships, and best schema practices.
6. Deployment Documentation. For managing user authentication and MVC structure, if relevant.

Appendices

Here's the Deployment Link of the Deployed Website;

https://event-management-system-production-b4b8.up.railway.app/

GitHub Repository Here:

https://github.com/hassanzaibjadoon/DBMS_PROJECT