The Department of Computer Science & Engineering (CSE)

(Even Sem 2024-25)

Project/ Internship Report On

"EventTix- Event Ticket Management"

(Major Project, MCA-P2401)

Submitted to

The Department of CSE

In partial fulfillment of the requirements for the award of the MCA

By

Praneet Kumar Pathak (SBU233465), Sem-IV, Section- 'A'
Manish Kumar Sharma (SBU233225), Sem-IV, Section-'A'

Under the guidance of

Dr. Avinash Kumar
Associate Professor, CSE, SBU
Department of CSE

Certificate

Certified that ***Praneet Kumar Pathak (SBU233465)*** *&* ***Manish Kumar Sharma (SBU233225)*** *of* ***MCA, Sem -IV '2023-2025'****,* have carried out the research work presented in this project entitled "**EventTix - Event Ticket Management**" for the award of **MCA** Degree from **Sarala Birla University, Ranchi** under my supervision during **Even Sem 2024-25 session**. The project embodies result of original work and studies carried out by Student himself and the contents of the project do not form the basis for the award of any other degree to the candidate or to anybody else.

Sign: _____

Date:

External Examiner

Sign: _____

Date:

Dr. Avinash Kumar

Associate Professor, SBU
Department of CSE
Sarala Birla University, Ranchi

Declaration

We, **Praneet Kumar Pathak (SBU233465)** and **Manish Kumar Sharma (SBU233225)**, student of **MCA Sem-IV, Batch(2023-25),** hereby declare that the report titled "**EventTix-Event Ticket Management**" which is submitted by us to the department of CSE , Sarala Birla University Jharkhand, in partial fulfillment of the requirement for the award of degree/ diploma in "**MCA**" , has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition. We further declare that the report is written by us and no part of the report is copied from any source(s) without being duly acknowledged.

Signature: _____

Date:

Praneet Kumar Pathak
SBU233465

Signature: _____

Date:

Manish Kumar Sharma
SBU233225

Acknowledgement

We express our sincere gratitude to my project guide **Dr. Avinash Kumar** for his/ her able guidance, continuous support and cooperation throughout my project, without which the present work would not have been possible. My endeavor stands incomplete without dedicating my gratitude to him/ she; he / she has contributed a lot towards successful completion of my project work.

We would like to acknowledge my indebtedness and deep sense of gratitude to Dr. Pankaj Kr Goswami sir, Dean-FoECS, SBU, Dr. Biswarup Samanta sir, Asso. Dean-FoECS & HoD – CSE&CA, SBU and my Program Coordinator, **Dr. Priyanka Srivastava** to encourage me to the highest peak and to provide me the opportunity to prepare the project.

We would also like to express my sincere thanks towards my teachers at SBU and my family and friends for their unending support, and tireless effort that kept me motivated throughout the completion of this project.

Name of the Students:

Praneet Kumar Pathak [SBU233465]
Manish Kumar Sharma [SBU233225]

Program: MCA
Semester: IV
Section: A
Batch: 2023 - 25

Table of Contents

Topic **Page Number**

THE DEPARTMENT OF "CSE & CA"
PROJECT SYNOPSIS

Student Name: Praneet Kumar Pathak & Manish Kumar Sharma

Enrollment No.: SBU233465 & SBU233225

Group No.: 23

Program & Branch: MCA **Batch:** 2023-25 **Semester:** IV **Section:** A

Academic Session: Even Semester 2024-25

Project Guide Details:

Guide Name: Dr. Avinash Kumar

Designation: Associate Professor, CSE, SBU

Project Information

1. Course Title: Major Project

2. Courses Code: MCA-P2401

3. Credit Unit: 6

4. Project/ Internship Duration:

a) Date of Project Commencement: 10th March, 2025

a) Date of Project Completion: 16th April, 2025

5. Approved Project Title: EventTix – Ticket Management System

6. Objectives:

The **EventTix Ticket Management System** is designed to provide a **comprehensive solution** for event organizers and attendees. The objectives of this project are as follows:

1. **Simplify Event Management for Admins**:
    - Provide a robust **Admin Dashboard** that enables easy creation, management, and deletion of events. Admins can easily update event details, such as event title, description, location, date, time, and ticket availability.
    - Empower admins to **track ticket sales** and monitor overall event performance in real time, offering insights into event success and areas that may need attention.
2. **Streamlined Ticket Booking for Users**:
    - Develop a **user-friendly platform** that allows attendees to browse through available events by categories, date, location, or popularity.
    - Provide an efficient **ticket booking system** where users can reserve tickets for events, view available slots, and select the number of tickets they wish to purchase.

3. **Secure and Reliable Ticket Verification**:
    - Implement **dynamic QR code generation** for each ticket booked. These QR codes will be unique to the user and event, ensuring that each ticket is **non-duplicable** and preventing unauthorized access to events.
    - Admins can easily **verify tickets** during event entry, either by scanning the dynamic QR code using a mobile device or by manually entering the ticket ID into the system.

Real-Time Updates:

- Utilize **Firebase** as a real-time backend to ensure that data such as ticket bookings, event updates, and user information are synchronized immediately across all users and administrators.
- Implement real-time Updates to keep users updated on ticket availability, upcoming events, or changes to event details, enhancing user engagement.

Seamless User Authentication:

- Integrate **Firebase Authentication** to offer a secure login and registration process for users and admins. This ensures that both admins and users can access their accounts with a safe and reliable authentication process.
- Users will be able to securely log in, view their past bookings, and manage their tickets, while admins can access advanced features, such as managing events and viewing detailed reports.

Ensure a Scalable and Flexible System:

- Design the system architecture with scalability in mind, ensuring that the platform can handle increasing traffic, events, and bookings as the user base grows.
- The system will be designed to accommodate future enhancements, such as integration with payment gateways, multi-language support, and additional user features, to enhance the overall experience.

7. Methodology to be adopted:

The EventTix Ticket Management System will be developed following an agile methodology, focusing on incremental development, user feedback, and constant refinement. The methodology will consist of the following stages:

1. Requirement Gathering and Analysis:

- **Feature Prioritization:** Based on these interviews, we will prioritize the most important features, such as event creation, user registration, ticket booking, and ticket verification.
- **System Requirements:** Define system architecture, technology stack (Firebase, HTML/CSS/JS), and third-party APIs (e.g., for dynamic QR code generation). The project will be developed using a cloud-native architecture, focusing on real-time data management.

2. System Design and Prototyping:

- **Database Design:** The backend will use Firebase Firestore as the real-time database to store user data, event details, and ticket information. We will define entities such as Users, Events, Bookings, and Tickets, as well as relationships between these entities.

- **Security Design:** Firebase's security rules will be configured to ensure that data is only accessible to authorized users. Admin access will be restricted to specific functions, while users will only have access to their own bookings.

3. Frontend and Backend Development:

- **Frontend Development: T**he frontend will be developed using HTML, CSS, and JavaScript, with a focus on creating a responsive design that works seamlessly across both desktop and mobile devices.
- **Backend Development:** Firebase will handle the authentication, real-time data management, and hosting of the application. We will use Firebase Authentication for user login and registration, Firebase Firestore for event and ticket data storage, and Firebase Hosting to deploy the web application.
- **QR Code Integration:** A third-party QR code generation API will be integrated to dynamically generate unique QR codes for each user's ticket.

4. Integration and Testing:

- **Unit Testing:** Testing individual components of the system, such as the event creation form, ticket booking process, and user login functionality.
- **Integration Testing:** Testing the end-to-end functionality of the application to ensure that the frontend communicates correctly with the backend, including user registration, event creation, ticket booking, and QR code generation.
- **Usability Testing:** Conduct user testing sessions to identify usability issues and improve the user interface based on feedback. This ensures that the system meets the needs of both admins and users.
- **Bug Fixing and Debugging:** Identify and resolve any bugs or performance issues that arise during testing.

5. Deployment and Post-Deployment Monitoring:

- **Deployment on Firebase Hosting:** Once the application is ready for production, it will be deployed to Firebase Hosting, providing secure and fast delivery of content to users.
- **Post-deployment Monitoring:** Continuously monitor system performance, user activity, and error logs to ensure the application is running smoothly. User feedback will be gathered to inform future updates and improvements.

6. Future Enhancements and Updates:

- **Feature Expansion:** Future updates will focus on adding features such as multi-language support, payment gateway integration, and the ability for admins to generate advanced reports.
- **Mobile App Version:** Eventually, we will develop a mobile app version of EventTix Platform.

8. Brief Summary of the project:

The **EventTix Ticket Management System** is a **comprehensive web application** designed to manage and streamline the process of event creation, ticket booking, and ticket verification for both event organizers and attendees. By leveraging **Firebase** for backend services such as real-time data storage, authentication, and hosting, the system ensures high performance, security, and ease of use.
For Admins, EventTix provides a full suite of management tools:

- **Event Management**: Admins can create new events, edit event details, and delete events once completed.

- **Ticket Management**: Admins can view bookings, track ticket sales, and verify tickets using **dynamic QR codes**.
- **Event Statistics**: Admins can access a dashboard displaying real-time data about ticket sales, booked tickets, and event attendance, helping them make data-driven decisions.

For Users, the system offers a simple and efficient booking experience:

- **Event Browsing and Ticket Booking**: Users can easily browse through available events, select tickets, and complete the booking process in a few steps.

- **Dynamic QR Code Generation**: Once a ticket is booked, the user receives a **unique QR code** that acts as a digital ticket, ensuring a secure and seamless entry to events.
- **Instant Confirmation**: Users receive an **email** and on-screen confirmation with event details and their ticket information after booking.

The system is designed to be **fully responsive**, allowing users to access and interact with the platform across multiple devices (desktop, tablet, mobile). The use of **Firebase** guarantees **real-time updates** and ensures that event and booking data is always up-to-date, improving both the user experience and administrative control.
The future of EventTix includes **enhancing reporting tools**, **payment gateway integration** for online payments, and providing **event reminders and notifications**. These features will provide a more complete and seamless solution for event organizers and attendees alike, enhancing the overall ticketing experience.

Signature with date
(Student)

Signature with date
(Project Guide)

List of figures

1. INTRODUCTION

Theoretical Background

The evolution of web technologies and cloud platforms has significantly transformed the way events are managed. Traditionally, event organizers relied on manual registration processes, printed tickets, and on-site verifications, which were time-consuming, error-prone, and inefficient. The growing demand for contactless, real-time, and secure solutions has led to the development of web-based ticketing systems. These systems provide a seamless interface for both organizers and attendees, ensuring better management, security, and user engagement.

Objective of the Project

The main objective of the EventTix Ticket Management System is to create a centralized and fully automated ticketing platform. This project aims to:

- Simplify the process of event creation and ticket booking.
- Allow secure, real-time, and dynamic QR code-based ticket generation.
- Enable role-based login for admins and users.
- Provide instant digital passes upon ticket booking.
- Eliminate the need for manual or paper-based systems.

Literature Review

Event management and ticketing systems like BookMyShow, Eventbrite, and MeraEvents provide various features, but they often lack customization for small or niche events. Moreover, they are mostly commercial platforms requiring service fees. EventTix, designed for academic and medium-scale events, focuses on open architecture, simplicity, and Firebase-based real-time functionalities that are more suitable for academic or internal organizational use.

Scope of the Study

The EventTix system supports:

- Admin role for creating and managing events.
- Real-time ticket bookings by users after login.
- Dynamic QR code generation for booked tickets.
- Ticket verification via QR scan or ticket ID. The system is developed using HTML, CSS, JavaScript, and Firebase (Firestore, Authentication, and Hosting). It is responsive and can be accessed via browsers on both desktop and mobile.

Limitations of the Study

- No online payment gateway integration.
- No auto-generated email/SMS ticket notifications.
- Only web-based version available (no native mobile app).
- No analytics or event performance reports.

2. PROBLEM ANALYSIS

Problem Definition

Event management, particularly ticketing, has traditionally been handled through manual processes or fragmented digital solutions. Physical tickets, spreadsheets, email-based registration, and manual guest verification still dominate many small- to medium-scale events. These practices often lead to issues such as long queues, ticket duplication, poor attendee tracking, and general inefficiency in operations.

Event organizers often struggle with coordinating event logistics, especially when it comes to monitoring ticket sales and ensuring secure entry at the venue. Simultaneously, attendees expect a modern, hassle-free experience with the ability to book tickets online, receive confirmations instantly, and gain quick access to venues using digital passes. In the absence of an integrated, real-time system, both stakeholders face major challenges — from organizational bottlenecks to poor user satisfaction. Furthermore, the COVID-19 pandemic accelerated the need for contactless systems, making traditional methods even more obsolete. With no centralized control and no real-time synchronization, the margin for error increases, ultimately impacting the credibility and success of events.

There is a pressing need for a secure, real-time, scalable system that allows for smooth event creation, efficient ticket booking, and reliable verification — all through a web interface that is both accessible and intuitive. This need serves as the foundation for the development of the EventTix Ticket Management System.

Requirement Analysis and Development

To ensure that the proposed solution meets user expectations and performs efficiently, the system requirements were broken down into functional and non-functional categories.

Functional Requirements

- **User Authentication**: The system must allow users to register and log in using secure credentials. Firebase Authentication is used to manage sessions securely.
- **Role-Based Access**: Users are assigned roles — either as an admin or a regular user. Access to features is based on these roles.
- **Event Creation and Management**: Admins can create, update, and delete events. Each event includes a title, description, date, location, and seat availability.
- **Ticket Booking**: Users can browse available events and book tickets after logging in. Each booking creates a record in the Firebase database.
- **Dynamic QR Code Generation**: After booking, each user receives a dynamic QR code representing their ticket. This code is unique and tied to the user and event.
- **Ticket Verification**: Admins can scan the QR code or manually enter the ticket ID to verify tickets. Verified tickets are marked to avoid re-entry.
- **Ticket Status Tracking**: Tickets have states such as "active", "used", or "invalid", managed in real-time.

Non-Functional Requirements

- **Real-Time Performance**: Firebase Firestore ensures real-time syncing of data across users and admins.
- **Security**: Firebase Authentication and role-based routing restrict unauthorized access.
- **Responsiveness**: The web interface is fully responsive and optimized for both desktop and mobile browsers.
- **Scalability**: The Firebase backend allows the system to scale easily for different event sizes

- and more users.
- **Availability**: Hosted on Firebase Hosting, the platform is available 24/7 with high uptime.
- **Maintainability**: Code is structured in a modular way using best practices in frontend development and database architecture.

Goals of Implementation

The development of EventTix was guided by clear implementation goals to ensure the success and usability of the system:

- **User-Centric Design**: Create a seamless and intuitive interface that simplifies the process of ticket booking and management.
- **Security and Accuracy**: Implement dynamic QR codes and secure database management to prevent ticket duplication and fraud.
- **Admin Efficiency**: Allow event organizers to manage multiple events, view ticket bookings, and verify entries with minimal manual effort.
- **Real-Time Syncing**: Ensure all data is updated instantly across devices and users without requiring manual refreshes or reloads.
- **Lightweight Deployment**: Use Firebase for both the backend and hosting to allow fast, serverless deployment and cost-efficiency.
- **Future Expandability**: Design the architecture in a way that allows easy integration of features like payment gateways, analytics, and push notifications.

The system provides the foundation for future enhancements, such as integration of payment gateways, reminder notifications, and event analytics, making it a scalable and future-ready solution. By reducing manual effort, increasing data security, and improving user convenience, EventTix bridges the gap between modern expectations and current limitations in the event management space.

3. SYSTEM IMPLIMENTATION DETAILS

Methodology Adopted

For the development of the EventTix Ticket Management System, the **Agile methodology** was adopted, ensuring iterative progress with regular testing and feedback loops. This approach allowed for flexibility in development and continuous improvements based on real-time feedback from stakeholders. The project was broken down into smaller tasks, each with clear milestones, ensuring the timely completion of features while maintaining quality.

The methodology involved several key stages:

- **Requirement Analysis**: Initial stages of the project involved gathering and defining the requirements through discussions with the project guide, as well as potential users. This provided a clear understanding of the needs of both the admin and user roles within the system.
- **Design and Prototyping**: Wireframes and mockups were created to visualize the user interface and design the user flow for both admins and users. Prototypes were shared for feedback, and adjustments were made to ensure a user-friendly and intuitive experience.
- **Development**: The development phase was initiated after finalizing the design, with the backend and frontend being developed concurrently. Frontend development was done using HTML, CSS, and JavaScript, while Firebase was chosen for real-time database management, authentication, and hosting.
- **Testing and Debugging**: After each feature was implemented, extensive unit tests were conducted to ensure functionality. These tests were followed by integration tests to verify that the frontend and backend were working seamlessly together. Bugs and issues were promptly addressed during each sprint.
- **Deployment**: The final system was deployed using Firebase Hosting, which provides a reliable, scalable, and serverless environment. Firebase Hosting ensured smooth hosting of the application with global access, providing efficient load times and data synchronization across different locations.

Hardware and Software Used

The system was developed with a specific hardware and software stack to ensure compatibility, scalability, and efficiency.

Hardware Requirements

- **Processor**: Intel i3 or higher
- **RAM**: 4GB minimum
- **Storage**: Sufficient storage space to accommodate development tools and project files (10GB or more)
- **Internet Connection**: Stable and high-speed internet connection for cloud-based development and testing

Software Requirements

- **Frontend Development**:
    - **HTML**: For creating the basic structure of the web pages.
    - **CSS**: For styling and layout design, ensuring the application is responsive and visually appealing.
    - **JavaScript**: For dynamic content, including the ticket booking functionality, event browsing, and QR code generation.
- **Backend Development**:
    - **Firebase**: A comprehensive backend solution that includes services like:
        - **Firebase Authentication**: For handling user logins and role-based authentication.
        - **Firebase Firestore**: A real-time NoSQL database for storing events, tickets, and user data.
        - **Firebase Hosting**: For deploying the application to the web and ensuring scalability and security.
- **Development Tools**:
    - **Visual Studio Code**: For writing and managing the project's source code.
    - **Postman**: For testing the API endpoints (if applicable) and ensuring communication with Firebase services.
- **Browser**:
    - The system was developed and tested using modern web browsers like Google Chrome and Mozilla Firefox to ensure compatibility and responsiveness across different platforms.

The combination of **Firebase** for backend management and **HTML, CSS, and JavaScript** for frontend development ensures that the application is both scalable and easy to manage. Additionally, Firebase provides real-time syncing capabilities, making it an ideal solution for applications that require instant updates and synchronization across devices.

Key Features Implemented

User Authentication

The authentication process is handled securely via **Firebase Authentication**. Users and admins are required to log in before they can access the system. Firebase's built-in authentication system handles user credentials and session management, providing a secure environment for both admin and user logins.

- **Admin Login**: Admins have access to a special dashboard to create, edit, and manage events, as well as verify tickets.
- **User Login**: Regular users can access the event browsing page and book tickets after logging in with their credentials.

Event Creation and Management

The admin dashboard, built with a clean and simple user interface, allows the admin to:

- Create new events, including setting the event's date, location, capacity, and description.
- Edit or delete events if needed.
- View all events and track the number of bookings for each.

This management system is connected to **Firebase Firestore**, where event data is stored and synced in real-time.

Ticket Booking

After logging in, users are able to browse the list of available events. Once they choose an event, they can proceed to book tickets. When a ticket is booked:

- The user receives a **unique ticket ID** and a **dynamic QR code** representing their ticket.
- The ticket and its status are stored in the Firebase database, and the ticket ID is linked to the event and user.

QR Code Generation

Each ticket booked by a user is assigned a **dynamic QR code**. This QR code contains the ticket's unique ID and event details, providing an easy and secure method of entry. The QR code is generated using a third-party **QR code generation API**, which dynamically creates codes that are unique to each ticket booking.

Ticket Verification

On the day of the event, admins can use the **QR code scanner** integrated into the admin panel to verify tickets:

- Scanning the QR code updates the ticket's status to "used" in real-time.
- Alternatively, the admin can manually enter the ticket ID to verify the ticket.

The verification system ensures that duplicate or invalid tickets cannot be used, providing a secure entry for event attendees.

System Architecture

The system is built on a **client-server architecture**. The frontend client is the user interface (UI) that interacts with the backend through API calls to Firebase services. All data — from user credentials to event details and ticket information — is securely stored in **Firebase Firestore**, which handles the real-time data synchronization.

- **Firebase Authentication**: Used for managing user and admin authentication.
- **Firebase Firestore**: Serves as the database for storing user details, event information, and ticket data.
- **Firebase Hosting**: Provides secure, fast, and scalable hosting for the web application.

The system architecture ensures that all data is synchronized in real-time, allowing seamless communication between users, admins, and the database. This also makes it scalable, as new events, tickets, and users can be added without impacting the system's performance.

4. DESIGNS

- **Flowchart:**

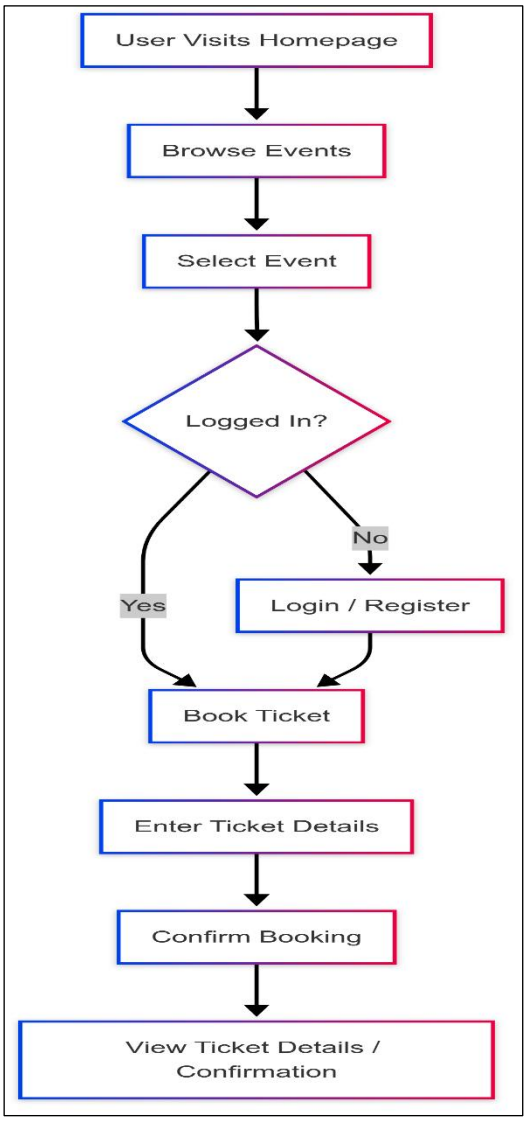
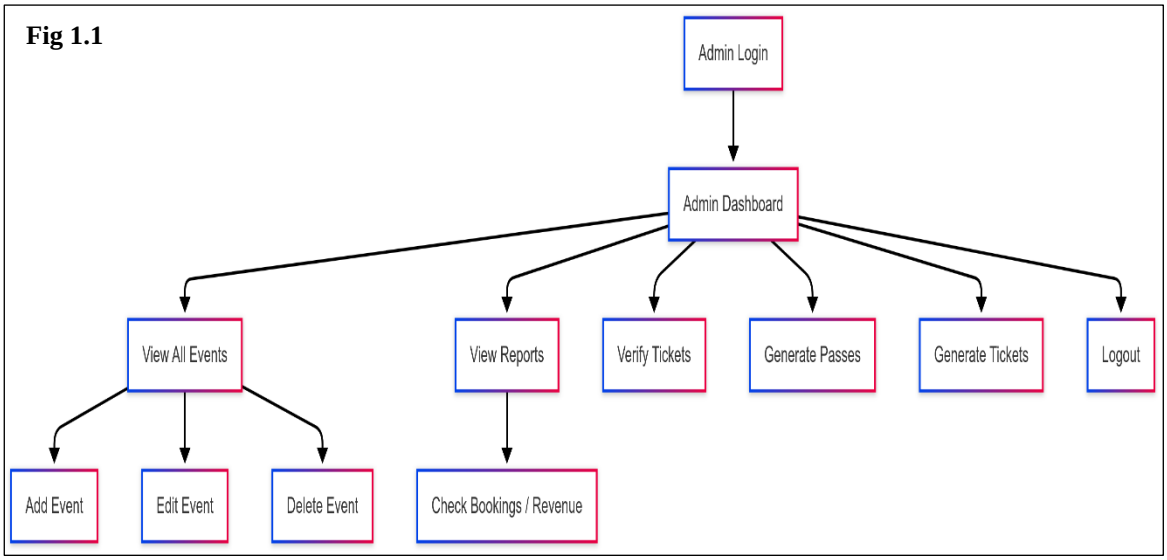


Fig 1.0

Fig 1.1



- **ER Diagram:**

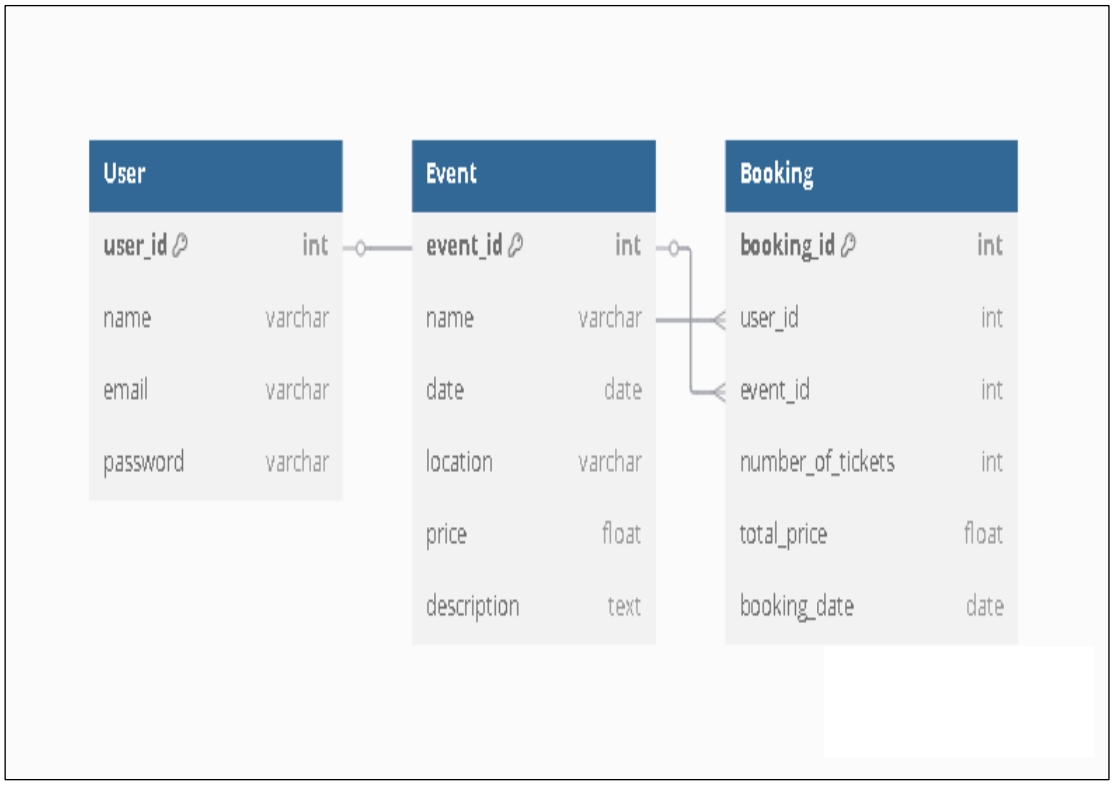


Fig 2.0

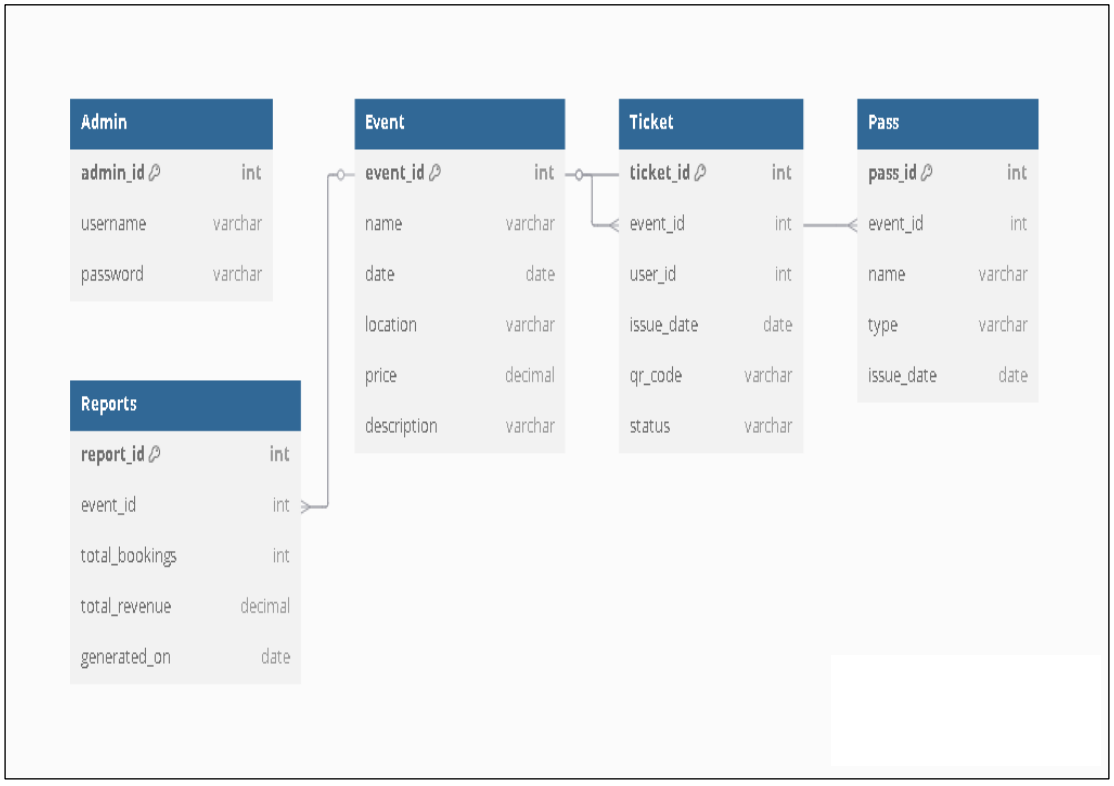


Fig 2.1

5. IMPLEMENTATION

System Development Process

The implementation of the **EventTix Ticket Management System** involved several phases, starting with the design of the user interface and followed by the integration of backend functionality. The development was carried out using **HTML, CSS, JavaScript**, and **Firebase** for real-time data synchronization, authentication, and hosting.

Frontend Implementation

The frontend of the EventTix system was developed using **HTML**, **CSS**, and **JavaScript** to ensure compatibility across devices and modern browsers. The website is designed to be responsive, ensuring a smooth user experience whether accessed from a desktop, tablet, or mobile device.

Key aspects of the frontend include:

- **Login Page**: The login page allows both users and admins to authenticate and access their respective dashboards. This page features simple login forms, where users enter their credentials (username and password), which are validated through Firebase Authentication.
- **User Dashboard**: Once logged in, users are directed to their dashboard, where they can browse available events and proceed to book tickets. Event details are displayed dynamically, retrieved from Firebase Firestore.
- **Admin Dashboard**: The admin dashboard provides an interface for event management, including event creation, editing, and deletion. It also allows admins to view the list of attendees and scan tickets for verification.
- **Ticket Booking Interface**: Users can book tickets for an event, and upon booking, they are provided with a **dynamic QR code** linked to their ticket ID. The ticket status is updated in real-time in the Firebase database.

Backend Implementation

The backend of the system relies heavily on **Firebase** services to handle all data storage, user authentication, and real-time syncing. The system's database is powered by **Firebase Firestore**, which stores event details, user profiles, and ticket data.

- **Firebase Authentication** is used to authenticate users, both admins and regular users. Once authenticated, users are granted access to their respective roles and functionalities.
- **Firebase Firestore** is used for storing all event-related data, such as event details, user ticket information, and bookings. The real-time nature of Firestore ensures that the admin and user interfaces are constantly updated as soon as changes are made — whether it's a new booking or an event update.
- **Firebase Hosting** is used to deploy the application, providing a reliable and scalable cloud-based solution for hosting the web application. Firebase Hosting ensures that the system can handle multiple users concurrently, with data stored on Google's secure servers.

Key Features Implemented

1. **Event Creation and Management**: Admins can create, update, and manage events. These events include key details such as the event name, description, date, time, and capacity.

2. **Ticket Booking**: Users can log in, browse the available events, and book tickets for their desired event. Each ticket booking generates a unique ticket ID and a dynamic QR code.

3. **Dynamic QR Code Generation**: Upon booking a ticket, users are provided with a unique dynamic QR code generated using a third-party API. This QR code is used for event entry verification.

4. **Ticket Verification**: Admins can verify tickets either by scanning the QR code or by manually entering the ticket ID. When a ticket is scanned or entered, its status is updated in the database to mark it as "used."

5. **Real-Time Data Synchronization**: The system uses **Firebase Firestore** for real-time data synchronization. As users book tickets, event data is updated immediately, ensuring that both the admin and user interfaces are always up-to-date.

Challenges Faced During Implementation

Throughout the implementation process, a few challenges were encountered:

- **Real-Time Data Syncing**: Ensuring that both admins and users see the updated ticket status and event data in real-time was initially a challenge. However, by leveraging **Firebase Firestore's real-time capabilities**, this issue was resolved, and data was synchronized across devices seamlessly.

- **QR Code Generation**: Initially, generating unique dynamic QR codes for each user proved tricky. However, integrating a third-party API solved this issue, and dynamic QR codes were generated correctly and securely for every ticket.

- **Ticket Verification**: Ensuring the integrity of the ticket verification process (avoiding duplicate tickets) was critical. Implementing real-time updates through Firebase ensured that once a ticket was marked as "used," it could not be used again.

Final System Features

The EventTix system now includes several important features:

- **Secure Authentication**: Users and admins can log in securely using Firebase Authentication.

- **Event Management**: Admins can create and manage events effortlessly, keeping track of all events in one place.

- **Ticket Booking**: Users can book tickets for events of their choice, and each booking generates a unique ticket ID and QR code.

- **QR Code Scanning**: Admins can scan or manually enter ticket IDs to verify tickets at the event.

- **Real-Time Data**: All data is stored in Firebase Firestore and updated in real time across users and admins.

Deployment

Once the system was fully implemented, it was deployed using **Firebase Hosting**, which offers a high level of security and performance. The deployment process ensured that the application could handle large numbers of concurrent users without performance degradation. Firebase Hosting also provided an easy-to-use environment for managing the deployment, ensuring the application was accessible online with minimal configuration.

Coding:

//index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>EventTix - India's Premier Event Ticketing Platform</title>
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css">

</head>
<body>
   <!-- Loader -->
   <div class="loader-container">
      <div class="loader-content">
         <div class="loader-logo">EventTix</div>
         <div class="loader"></div>
         <p class="loader-text">Loading amazing events...</p>
      </div>
   </div>

   <!-- Navigation -->
   <nav class="navbar">
      <div class="nav-content">
         <a href="#" class="logo">EventTix</a>
         <div class="nav-links">
            <a href="#features"><i class="fas fa-star"></i>Features</a>
            <a href="#upcoming"><i class="fas fa-calendar"></i>Events</a>
            <a href="#about"><i class="fas fa-info-circle"></i>About</a>
            <a href="#contact"><i class="fas fa-envelope"></i>Contact</a>
         </div>
         <div class="nav-actions">
            <div class="profile-icon-wrapper" id="accountLink">
               <i class="fas fa-user-circle" style="font-size: 24px; color: white;"></i>
            </div>
            <button class="menu-btn">
               <div class="bar"></div>
               <div class="bar"></div>
               <div class="bar"></div>
            </button>
         </div>
      </div>
   </nav>

   <!-- Hero Section -->
   <section class="hero">
      <div class="hero-content">
         <h1>India's Premier Event Ticketing Platform</h1>
         <p>From Bollywood concerts to cricket matches, cultural festivals to comedy shows - book your tickets instantly!</p>
         <div class="hero-cta">
            <button class="cta-btn primary" onclick="handleExploreEvents()">
               <i class="fas fa-compass"></i> Explore Events
            </button>
            <a href="#trending" class="cta-btn secondary">Trending Now</a>
         </div>
         <div class="hero-stats">
            <div class="stat-item">
               <span class="stat-number">1M+</span>
               <span class="stat-text">Happy Customers</span>
            </div>
            <div class="stat-item">
               <span class="stat-number">10K+</span>
               <span class="stat-text">Events</span>
```

```
            </div>
            <div class="stat-item">
              <span class="stat-number">100+</span>
              <span class="stat-text">Cities</span>
            </div>
          </div>
        </div>
      </section>

      <!-- About Us Section -->
      <section class="about" id="about">
        <div class="section-header">
          <h2>About EventTix</h2>
          <p>Your trusted partner in event ticketing</p>
        </div>
        <div class="about-container">
          <!-- Story Section -->
          <div class="about-text">
            <h3><i class="fas fa-book"></i> Our Story</h3>
            <p>EventTix was founded with a vision to revolutionize event ticketing in India. We understand the passion and excitement that comes with attending live events, and we're here to make that experience seamless and memorable.</p>
            <p>Our platform serves millions of event enthusiasts across India, connecting them to a diverse range of events from cultural festivals to sporting events, concerts to comedy shows.</p>
          </div>

          <!-- Mission & Vision -->
          <div class="mission-vision">
            <div class="mission">
              <h3><i class="fas fa-bullseye"></i> Our Mission</h3>
              <p>To provide a seamless and secure ticketing platform that connects event organizers with their audience, making event discovery and booking accessible to everyone.</p>
            </div>
            <div class="vision">
              <h3><i class="fas fa-eye"></i> Our Vision</h3>
              <p>To become India's most trusted and preferred event ticketing platform, setting new standards in customer experience and technological innovation.</p>
            </div>
          </div>

          <!-- Achievements -->
          <div class="achievements">
            <div class="achievements-grid">
              <div class="achievement-item">
                <i class="fas fa-users"></i>
                <h4>1M+</h4>
                <p>Happy Customers</p>
              </div>
              <div class="achievement-item">
                <i class="fas fa-ticket-alt"></i>
                <h4>10K+</h4>
                <p>Events Managed</p>
              </div>
              <div class="achievement-item">
                <i class="fas fa-city"></i>
                <h4>100+</h4>
                <p>Cities Covered</p>
              </div>
              <div class="achievement-item">
                <i class="fas fa-handshake"></i>
                <h4>500+</h4>
                <p>Event Partners</p>
              </div>
            </div>
          </div>

          <!-- Team Section -->
          <div class="team-section">
            <h3>Designed & Developed By</h3>
            <div class="team-grid">
              <div class="team-member">
                <div class="member-image">
                  <img src="images/ms.jpeg" alt="Manish Sharma">
                  <div class="social-overlay">
                    <a href="https://www.linkedin.com/in/manishsharma18/" target="_blank"><i class="fab fa-linkedin"></i></a>
                  </div>
                </div>
                <div class="member-info">
                  <h4>Manish Kumar Sharma</h4>
```

```
                <p>Lead Developer</p>
                <div class="member-skills">
                  <span>Full Stack Development</span>
                  <span>UI/UX Design</span>
                </div>
              </div>
            </div>
            <div class="team-member">
              <div class="member-image">
                <img src="images/pkp.jpeg" alt="Pranit Pathak">
                <div class="social-overlay">
                  <a href="https://www.linkedin.com/in/pranit-kumar-pathak-81ab8b152/" target="_blank"><i class="fab fa-linkedin"></i></a>
                </div>
              </div>
              <div class="member-info">
                <h4>Pranit Kumar Pathak</h4>
                <p>UI/UX Designer</p>
                <div class="member-skills">
                  <span>Frontend Development</span>
                  <span>UI/UX Design</span>
                </div>
              </div>
            </div>
          </div>
        </div>
      </section>

      <!-- Features Section -->
      <section class="features" id="features">
        <div class="section-header">
          <h2>Why Choose EventTix</h2>
          <p>Your hassle-free ticket booking companion</p>
        </div>
        <div class="features-grid">
          <div class="feature-card">
            <i class="fas fa-bolt"></i>
            <h3>Instant Booking</h3>
            <p>Book tickets in seconds with our lightning-fast platform</p>
          </div>
          <div class="feature-card">
            <i class="fas fa-indian-rupee-sign"></i>
            <h3>Secure Payments</h3>
            <p>Multiple payment options including UPI, net banking, and cards</p>
          </div>
          <div class="feature-card">
            <i class="fas fa-qrcode"></i>
            <h3>Digital Tickets</h3>
            <p>Hassle-free entry with QR code-based e-tickets</p>
          </div>
          <div class="feature-card">
            <i class="fas fa-clock"></i>
            <h3>24/7 Support</h3>
            <p>Round-the-clock customer support in multiple Indian languages</p>
          </div>
          <!-- Add these inside the features-grid div -->
          <div class="feature-card">
            <i class="fas fa-mobile-alt"></i>
            <h3>Mobile First</h3>
            <p>Book tickets on the go with our mobile-optimized platform</p>
          </div>
          <div class="feature-card">
            <i class="fas fa-shield-alt"></i>
            <h3>Secure Platform</h3>
            <p>End-to-end encryption for all your transactions and data</p>
          </div>
        </div>
      </section>

      <!-- Trending Events Section -->
      <section id="trending" class="trending-events">
        <div class="section-header">
          <h2>Trending Now</h2>
          <p>Hot events that everyone's talking about</p>
        </div>
        <div id="trendingContainer" class="events-grid"></div>
      </section>
```

```
<section id="upcoming" class="upcoming-events">
  <div class="section-header">
    <h2>Upcoming Events</h2>
    <p>Mark your calendar for these exciting events</p>
  </div>
  <div id="upcomingContainer" class="events-grid"></div>
</section>

<!-- Contact Section -->
<section class="contact" id="contact">
  <div class="section-header">
    <h2>Get in Touch</h2>
    <p>We're here to help with any questions or concerns</p>
  </div>
  <div class="contact-container">
    <div class="contact-info">
      <div class="info-card">
        <i class="fas fa-map-marker-alt"></i>
        <h4>Our Location</h4>
        <p>KaramToli, Ranchi, 834001</p>
      </div>
      <div class="info-card">
        <i class="fas fa-phone"></i>
        <h4>Call Us</h4>
        <p>+91 9876543210</p>
        <p>Mon-Sat: 9:00 AM - 8:00 PM</p>
      </div>
      <div class="info-card">
        <i class="fas fa-envelope"></i>
        <h4>Email Us</h4>
        <p>support@eventtix.com</p>
        <p>business@eventtix.com</p>
      </div>
    </div>
    <div class="contact-form-container">
      <form id="contact-form" class="contact-form">
        <div class="form-group">
          <input type="text" name="name" placeholder="Your Name" required>
        </div>
        <div class="form-group">
          <input type="email" name="email" placeholder="Your Email" required>
        </div>
        <div class="form-group">
          <input type="text" name="subject" placeholder="Subject" required>
        </div>
        <div class="form-group">
          <textarea name="message" placeholder="Your Message" required></textarea>
        </div>
        <button type="submit">
          <i class="fas fa-paper-plane"></i>
          Send Message
        </button>
      </form>
    </div>
  </div>
</section>

<!-- Footer -->
<footer class="footer">
  <div class="footer-container">
    <div class="footer-grid">
      <div class="footer-section">
        <h3>EventTix</h3>
        <p>India's leading platform for booking event tickets. Download our app for the best experience.</p>
        <div class="app-buttons">
          <a href="#" class="app-btn app-download-link">
            <img src="https://upload.wikimedia.org/wikipedia/commons/7/78/Google_Play_Store_badge_EN.svg"
              alt="Get it on Google Play"
              style="height: 40px; width: auto;">
          </a>
          <a href="#" class="app-btn app-download-link">
            <img src="https://upload.wikimedia.org/wikipedia/commons/3/3c/Download_on_the_App_Store_Badge.svg"
              alt="Download on App Store"
              style="height: 40px; width: auto;">
          </a>
        </div>
      </div>
```

```
            <div class="footer-section">
                <h4>Popular Cities</h4>
                <ul>
                    <li><a href="#">Ranchi</a></li>
                    <li><a href="#">Delhi</a></li>
                    <li><a href="#">Bangalore</a></li>
                    <li><a href="#">Hyderabad</a></li>
                    <li><a href="#">Chennai</a></li>
                    <li><a href="#">Kolkata</a></li>
                </ul>
            </div>

            <div class="footer-section">
                <h4>Event Categories</h4>
                <ul>
                    <li><a href="#">Bollywood Concerts</a></li>
                    <li><a href="#">Comedy Shows</a></li>
                    <li><a href="#">Cricket Matches</a></li>
                    <li><a href="#">Classical Music</a></li>
                    <li><a href="#">Theatre Shows</a></li>
                    <li><a href="#">Food Festivals</a></li>
                </ul>
            </div>

            <div class="footer-section">
                <h4>Stay Updated</h4>
                <p>Subscribe to our newsletter for exclusive offers and updates.</p>
                <form class="newsletter-form">
                    <input type="email" placeholder="Enter your email" required>
                    <button type="submit">Subscribe</button>
                </form>
            </div>
        </div>

        <div class="footer-bottom">
            <div class="footer-links">
                <a href="about.html">About Us</a>
                <a href="terms.html">Terms & Conditions</a>
                <a href="privacy.html">Privacy Policy</a>
                <a href="refund.html">Refund Policy</a>
                <a href="faq.html">FAQ</a>
            </div>
            <div class="payment-methods">
                <i class="fab fa-cc-visa"></i>
                <i class="fab fa-cc-mastercard"></i>
                <i class="fab fa-google-pay"></i>
            </div>
            <p class="copyright">© 2024-2025 EventTix. All rights reserved. Made with ❤️ in India</p>
            <p class="publish-message">Want to publish an event with us? Drop a mail to publish@eventTix.in</p>
        </div>
    </div>
</footer>

<!-- Newsletter Submission Message -->
<div class="newsletter-message" id="newsletter-message">
    Thank you for subscribing to our newsletter!
</div>

<!-- Login Modal -->
<div id="loginModal" class="login-modal">
    <div class="login-modal-content">
        <div class="login-modal-header">
            <i class="fas fa-lock"></i>
            <h2>Login Required</h2>
        </div>
        <div class="login-modal-body">
            <p>Please login to view event details and book tickets.</p>
            <div class="login-benefits">
                <div class="benefit-item">
                    <i class="fas fa-ticket-alt"></i>
                    <span>Easy booking</span>
                </div>
                <div class="benefit-item">
                    <i class="fas fa-history"></i>
                    <span>Booking history</span>
                </div>
                <div class="benefit-item">
```

```
                <i class="fas fa-bell"></i>
                <span>Event updates</span>
            </div>
        </div>
    </div>
    <div class="login-modal-footer">
        <button class="modal-btn secondary" onclick="closeLoginModal()">Maybe Later</button>
        <button class="modal-btn primary" onclick="redirectToLogin()">Login Now</button>
    </div>
  </div>
</div>

<script>

    // Account link functionality
document.addEventListener('DOMContentLoaded', function() {
  const accountLink = document.querySelector('.profile-icon-wrapper');

  // Define the click handler function
  function handleAccountClick(e) {
    e.preventDefault();

    // Check login status at click time
    const currentLoginStatus = localStorage.getItem('isLoggedIn') === 'true';
    const currentUserRole = localStorage.getItem('userRole');

    if (currentLoginStatus) {
      if (currentUserRole === 'admin') {
        window.location.replace('admin.html');
      } else {
        window.location.replace('dashboard.html');
      }
    } else {
      window.location.replace('login.html');
    }
  }

  // Add click event listener to the profile icon wrapper
  accountLink.addEventListener('click', handleAccountClick);
});
  // Loader functionality
  window.addEventListener('load', function() {
    const loader = document.querySelector('.loader-container');
    setTimeout(() => {
      loader.style.opacity = '0';
      setTimeout(() => {
        loader.style.display = 'none';
      }, 500);
    }, 1000);
  });

  // Navigation functionality
  const menuBtn = document.querySelector('.menu-btn');
  const navLinks = document.querySelector('.nav-links');
  const navbar = document.querySelector('.navbar');

  menuBtn.addEventListener('click', () => {
    navLinks.classList.toggle('active');
  });

  window.addEventListener('scroll', () => {
    if (window.scrollY > 50) {
      navbar.classList.add('scrolled');
    } else {
      navbar.classList.remove('scrolled');
    }
  });

  // Features animation
  function animateFeatures() {
    const featureCards = document.querySelectorAll('.feature-card');

    const observer = new IntersectionObserver((entries) => {
      entries.forEach(entry => {
        if (entry.isIntersecting) {
          entry.target.classList.add('visible');
        }
      });
```

```
    }, { threshold: 0.1 });

    featureCards.forEach(card => {
        observer.observe(card);
    });
}

// Animate elements on scroll
function animateOnScroll() {
    const elements = document.querySelectorAll('.category-card, .event-card');

    const observer = new IntersectionObserver((entries) => {
        entries.forEach(entry => {
            if (entry.isIntersecting) {
                entry.target.style.opacity = '1';
                entry.target.style.transform = 'translateY(0)';
            }
        });
    }, { threshold: 0.1 });

    elements.forEach(element => {
        element.style.opacity = '0';
        element.style.transform = 'translateY(20px)';
        element.style.transition = 'opacity 0.6s ease, transform 0.6s ease';
        observer.observe(element);
    });
}

// Initialize animations
window.addEventListener('load', () => {
    animateFeatures();
    animateOnScroll();
});

// Newsletter form submission
document.querySelector('.newsletter-form').addEventListener('submit', function(e) {
    e.preventDefault();
    const email = this.querySelector('input').value;

    // Here you would typically send this to your server
    console.log('Newsletter subscription:', email);

    const message = document.getElementById('newsletter-message');
    message.style.display = 'block';
    setTimeout(() => {
        message.style.display = 'none';
    }, 3000);

    this.reset();
});

// Smooth scroll for footer links
document.querySelectorAll('.footer-links a').forEach(link => {
    link.addEventListener('click', function(e) {
        const href = this.getAttribute('href');
        if (href.startsWith('#')) {
            e.preventDefault();
            const target = document.querySelector(href);
            if (target) {
                target.scrollIntoView({
                    behavior: 'smooth'
                });
            }
        }
    });
});

// Contact form submission
document.getElementById('contact-form').addEventListener('submit', function(e) {
    e.preventDefault();
    const formData = new FormData(this);

    // Here you would typically send this to your server
    console.log('Contact form submission:');
    for (let [key, value] of formData.entries()) {
        console.log(key + ': ' + value);
    }
```

```
        alert('Thank you for your message. We will get back to you soon!');
        this.reset();
    });

    // Add this to your existing script
    document.addEventListener('DOMContentLoaded', function() {
        const menuBtn = document.querySelector('.menu-btn');
        const closeBtn = document.querySelector('.close-menu');
        const navLinks = document.querySelector('.nav-links');
        const links = document.querySelectorAll('.nav-links a');

        menuBtn.addEventListener('click', () => {
            navLinks.classList.add('active');
            document.body.style.overflow = 'hidden'; // Prevent scrolling when menu is open
        });

        closeBtn.addEventListener('click', () => {
            navLinks.classList.remove('active');
            document.body.style.overflow = ''; // Restore scrolling
        });

        // Close menu when clicking a link
        links.forEach(link => {
            link.addEventListener('click', () => {
                navLinks.classList.remove('active');
                document.body.style.overflow = '';
            });
        });

        // Close menu when clicking outside
        document.addEventListener('click', (e) => {
            if (navLinks.classList.contains('active') &&
                !navLinks.contains(e.target) &&
                !menuBtn.contains(e.target)) {
                navLinks.classList.remove('active');
                document.body.style.overflow = '';
            }
        });
    });
</script>
<script type="module">
    import { initializeApp } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-app.js";
    import { getFirestore, collection, getDocs, query, orderBy, limit } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js";
    import { getAuth, onAuthStateChanged } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js";

    const firebaseConfig = {
        apiKey: "AIzaSyCg7vwuwfN8oWSMagExshHCtMHnxzc7pH0",
        authDomain: "event-ticket-fc753.firebaseapp.com",
        projectId: "event-ticket-fc753",
        storageBucket: "event-ticket-fc753.firebasestorage.app",
        messagingSenderId: "216313948465",
        appId: "1:216313948465:web:4c4b8eb9fbb4257fe6e810",
        measurementId: "G-SGBZK0BYVL"
    };

    // Initialize Firebase
    const app = initializeApp(firebaseConfig);
    const db = getFirestore(app);
    const auth = getAuth(app);

    // Make auth available globally
    window.auth = auth;

    // Helper function to shuffle array
    function shuffleArray(array) {
        for (let i = array.length - 1; i > 0; i--) {
            const j = Math.floor(Math.random() * (i + 1));
            [array[i], array[j]] = [array[j], array[i]];
        }
        return array;
    }

    // Helper function to create event pairs
    function createEventPairs(events, containerId) {
        const container = document.getElementById(containerId);
        if (!events || events.length === 0) {
            container.parentElement.style.display = 'none';
            return;
```

```
        }

        container.innerHTML = '';

        // Add single-event class if there's only one event
        if (events.length === 1) {
            container.classList.add('single-event');
        } else {
            container.classList.remove('single-event');
        }

        // Create cards for all events
        events.forEach(event => {
            const eventCard = document.createElement('div');
            eventCard.className = 'event-card';
            eventCard.onclick = () => handleEventClick(event.id);

            eventCard.innerHTML = `
                <img src="${event.images?.[0] || 'placeholder.jpg'}" alt="${event.name}">
                <div class="event-info">
                    <div>
                        <h3>${event.name}</h3>
                        <p><i class="fas fa-calendar"></i> ${new Date(event.date).toLocaleDateString()}</p>
                        <p><i class="fas fa-map-marker-alt"></i> ${event.eventType === 'online' ? 'Online Event' : event.venueDetails?.venue ||
'TBA'}</p>
                    </div>
                </div>
            `;

            container.appendChild(eventCard);
        });

        container.parentElement.style.display = 'block';
    }

    // Update the loadAllEvents function
    async function loadAllEvents() {
        try {
            const querySnapshot = await getDocs(collection(db, 'events'));
            const allEvents = [];
            const now = new Date();

            querySnapshot.forEach((doc) => {
                const event = { id: doc.id, ...doc.data() };

                // Skip events without images
                if (!event.images || !Array.isArray(event.images) || event.images.length === 0) {
                    console.log('Skipped - No images:', event.name);
                    return;
                }

                // Skip if event date is not set
                if (!event.date) {
                    console.log('Skipped - No event date:', event.name);
                    return;
                }

                // Check scheduling rules
                if (event.scheduling) {
                    const publishDate = event.scheduling.publishDate ? new Date(event.scheduling.publishDate) : null;

                    // Skip if not yet published
                    if (publishDate && now < publishDate) {
                        console.log('Skipped - Not published yet:', event.name);
                        return;
                    }

                    // Skip if past hide date
                    if (event.scheduling.hideDate && now > new Date(event.scheduling.hideDate)) {
                        console.log('Skipped - Past hide date:', event.name);
                        return;
                    }
                }

                const eventDate = new Date(event.date);
                const bookingStartDate = event.bookingStartDate ? new Date(event.bookingStartDate) : null;
                const bookingEndDate = event.bookingEndDate ? new Date(event.bookingEndDate) : null;
```

```
            // Skip past events
            if (eventDate <= now) {
                console.log('Skipped - Past event:', event.name);
                return;
            }

            // Categorize event
            if (bookingStartDate && bookingStartDate > now) {
                event.section = 'upcoming';
                allEvents.push(event);
                console.log('Added to Upcoming:', event.name, 'Booking starts:', bookingStartDate);
            } else if ((!bookingStartDate || bookingStartDate <= now) &&
                    (!bookingEndDate || bookingEndDate > now)) {
                event.section = 'trending';
                allEvents.push(event);
                console.log('Added to Trending:', event.name, 'Booking period:',
                    bookingStartDate ? bookingStartDate : 'No start date',
                    'to',
                    bookingEndDate ? bookingEndDate : 'No end date');
            } else {
                console.log('Skipped - Not in any section:', event.name);
            }
        });

        console.log('Final event counts:', {
            total: allEvents.length,
            trending: allEvents.filter(e => e.section === 'trending').length,
            upcoming: allEvents.filter(e => e.section === 'upcoming').length
        });

        // Filter and display events
        const trendingEvents = shuffleArray(
            allEvents.filter(event => event.section === 'trending')
        );

        const upcomingEvents = allEvents
            .filter(event => event.section === 'upcoming')
            .sort((a, b) => new Date(a.bookingStartDate) - new Date(b.bookingStartDate));

        // Always show both sections
        document.getElementById('trending').style.display = 'block';
        document.getElementById('upcoming').style.display = 'block';

        // Display events
        if (trendingEvents.length > 0) {
            createEventPairs(trendingEvents, 'trendingContainer');
        } else {
            document.getElementById('trendingContainer').innerHTML = `
                <div style="text-align: center; padding: 2rem; width: 100%;">
                    <i class="fas fa-calendar-alt" style="font-size: 3rem; color: #ddd; margin-bottom: 1rem;"></i>
                    <p style="color: #666;">No events currently booking. Check back soon!</p>
                </div>
            `;
        }

        if (upcomingEvents.length > 0) {
            createEventPairs(upcomingEvents, 'upcomingContainer');
        } else {
            document.getElementById('upcomingContainer').innerHTML = `
                <div style="text-align: center; padding: 2rem; width: 100%;">
                    <i class="fas fa-clock" style="font-size: 3rem; color: #ddd; margin-bottom: 1rem;"></i>
                    <p style="color: #666;">No upcoming events at the moment. Stay tuned!</p>
                </div>
            `;
        }

    } catch (error) {
        console.error("Error loading events:", error);
        const errorMessage = `
            <div style="text-align: center; padding: 2rem; width: 100%;">
                <i class="fas fa-exclamation-circle" style="font-size: 3rem; color: #ff6b6b; margin-bottom: 1rem;"></i>
                <p style="color: #666;">Error loading events. Please try again later.</p>
            </div>
        `;
        document.getElementById('trendingContainer').innerHTML = errorMessage;
        document.getElementById('upcomingContainer').innerHTML = errorMessage;
    }
}
```

```
    // Helper function to hide all sections
    function hideAllSections() {
        ['trending', 'popular', 'upcoming'].forEach(id => {
            document.getElementById(id).style.display = 'none';
        });
    }

    // Update the load event listener
    window.addEventListener('load', () => {
        loadAllEvents();
    });

    // Add this function to handle event clicks
    function handleEventClick(eventId) {
        const isLoggedIn = localStorage.getItem('isLoggedIn') === 'true';
        if (!isLoggedIn) {
            // Show custom login modal
            document.getElementById('loginModal').style.display = 'block';
            // Store eventId for later use
            localStorage.setItem('pendingEventId', eventId);
        } else {
            // User is logged in, proceed to event details
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
            window.location.href = `${baseUrl}/event-details.html?id=${eventId}`;
        }
    }

    function closeLoginModal() {
        document.getElementById('loginModal').style.display = 'none';
        localStorage.removeItem('pendingEventId');
    }

    function redirectToLogin() {
        window.location.href = 'login.html';
    }

    // Make functions globally available
    window.handleEventClick = handleEventClick;
    window.closeLoginModal = closeLoginModal;
    window.redirectToLogin = redirectToLogin;

    // Add this after Firebase initialization
    onAuthStateChanged(auth, (user) => {
        // You can use this to update any UI elements based on auth state
        console.log('Auth state changed:', user ? 'User logged in' : 'User not logged in');
    });

    // Update the handleExploreEvents function to use the observer
    function handleExploreEvents() {
        const user = auth.currentUser;
        if (user) {
            window.location.replace('dashboard.html');
        } else {
            window.location.replace('login.html');
        }
    }

    // Make handleExploreEvents available globally
    window.handleExploreEvents = handleExploreEvents;

    // Add auth state change listener
    onAuthStateChanged(auth, (user) => {
        if (!user && window.location.pathname.includes('dashboard.html')) {
            window.location.replace('login.html');
        }
    });
</script>
<!-- Add this right before closing body tag -->
<nav class="bottom-nav">
    <a href="#" class="active">
        <i class="fas fa-home"></i>
        <span>Home</span>
    </a>
    <a href="#features">
        <i class="fas fa-star"></i>
        <span>Features</span>
    </a>
```

```
    <a href="#upcoming">
        <i class="fas fa-calendar"></i>
        <span>Events</span>
    </a>
    <a href="#about">
        <i class="fas fa-info-circle"></i>
        <span>About</span>
    </a>
    <a href="#contact">
        <i class="fas fa-envelope"></i>
        <span>Contact</span>
    </a>
</nav>

<script>
    // Replace the existing scroll spy code with this
    document.addEventListener('DOMContentLoaded', function() {
        const bottomNav = document.querySelector('.bottom-nav');
        const sections = document.querySelectorAll('section[id]');
        const navLinks = document.querySelectorAll('.bottom-nav a');

        function setActiveLink() {
            let found = false;
            const scrollPosition = window.scrollY + window.innerHeight / 2;

            // First, remove active class from all links
            navLinks.forEach(link => link.classList.remove('active'));

            // Check each section
            sections.forEach(section => {
                const sectionTop = section.offsetTop - 100; // Adjust for header
                const sectionBottom = sectionTop + section.offsetHeight;
                const sectionId = section.getAttribute('id');

                if (scrollPosition >= sectionTop && scrollPosition < sectionBottom) {
                    found = true;
                    // Find and activate corresponding nav link
                    const correspondingLink = document.querySelector(`.bottom-nav a[href="#${sectionId}"]`);
                    if (correspondingLink) {
                        correspondingLink.classList.add('active');
                    }
                }
            });

            // If we're at the top of the page, activate home
            if (window.scrollY < 100) {
                document.querySelector('.bottom-nav a[href="#"]').classList.add('active');
                found = true;
            }

            // If no section is active and we're not at the top, activate the closest section
            if (!found) {
                const closest = [...sections].reduce((prev, curr) => {
                    const prevDiff = Math.abs(prev.offsetTop - scrollPosition);
                    const currDiff = Math.abs(curr.offsetTop - scrollPosition);
                    return prevDiff < currDiff ? prev : curr;
                });

                const closestId = closest.getAttribute('id');
                const correspondingLink = document.querySelector(`.bottom-nav a[href="#${closestId}"]`);
                if (correspondingLink) {
                    correspondingLink.classList.add('active');
                }
            }
        }

        // Add click event listeners
        navLinks.forEach(link => {
            link.addEventListener('click', function(e) {
                e.preventDefault();

                const href = this.getAttribute('href');

                if (href === '#') {
                    window.scrollTo({
                        top: 0,
                        behavior: 'smooth'
                    });
```

```
            } else {
                const target = document.querySelector(href);
                if (target) {
                    const headerOffset = 80;
                    const elementPosition = target.offsetTop;
                    const offsetPosition = elementPosition - headerOffset;

                    window.scrollTo({
                        top: offsetPosition,
                        behavior: 'smooth'
                    });
                }
            }
        });
    });

    // Add scroll event listener with debounce
    let scrollTimeout;
    window.addEventListener('scroll', () => {
        if (scrollTimeout) {
            window.cancelAnimationFrame(scrollTimeout);
        }
        scrollTimeout = window.requestAnimationFrame(() => {
            setActiveLink();
        });
    });

    // Initial check for active section
    setActiveLink();
});
</script>

<!-- Add this function to handle random event section redirection -->
<script>
function redirectToRandomEventSection(e) {
    e.preventDefault();
    const eventSections = ['#trending', '#popular', '#upcoming'];
    const randomSection = eventSections[Math.floor(Math.random() * eventSections.length)];
    const target = document.querySelector(randomSection);

    if (target) {
        const headerOffset = 80;
        const elementPosition = target.offsetTop;
        const offsetPosition = elementPosition - headerOffset;

        window.scrollTo({
            top: offsetPosition,
            behavior: 'smooth'
        });

        // Update active states in bottom nav
        document.querySelectorAll('.bottom-nav a').forEach(link => {
            link.classList.remove('active');
            if (link.getAttribute('href') === '#events') {
                link.classList.add('active');
            }
        });
    }
}

// Add event listeners for both top and bottom nav Events links
document.addEventListener('DOMContentLoaded', function() {
    // For top navigation
    const topNavEventLink = document.querySelector('.nav-links a[href="#events"]');
    if (topNavEventLink) {
        topNavEventLink.addEventListener('click', redirectToRandomEventSection);
    }

    // For bottom navigation
    const bottomNavEventLink = document.querySelector('.bottom-nav a[href="#events"]');
    if (bottomNavEventLink) {
        bottomNavEventLink.addEventListener('click', redirectToRandomEventSection);
    }
});
</script>

<!-- Add this function to handle explore events button click -->
<script>
```

```
    function handleExploreEvents() {
        // Get current user
        const auth = getAuth();
        const user = auth.currentUser;

        if (user) {
            // User is signed in, redirect to dashboard
            window.location.replace('dashboard.html');
        } else {
            // No user is signed in, redirect to login
            window.location.replace('login.html');
        }
    }
    </script>
</body>
</html>
```

//scripts.js

```
/// Import Firebase modules
import {
    getFirestore,
    collection,
    addDoc,
    getDocs,
    deleteDoc,
    doc,
    getDoc,
    onSnapshot,
    updateDoc,
    query,
    orderBy,
    serverTimestamp,
    where,
    limit,
    writeBatch
} from "https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js";

import {
    signOut,
    onAuthStateChanged
} from "https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js";

// Add increment to your Firebase imports
import { increment } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js";

// Make sure to export increment
window.increment = increment;

// Make Firebase functions available globally
window.collection = collection;
window.addDoc = addDoc;
window.getDocs = getDocs;
window.deleteDoc = deleteDoc;
window.doc = doc;
window.getDoc = getDoc;
window.updateDoc = updateDoc;
window.onSnapshot = onSnapshot;
window.serverTimestamp = serverTimestamp;
window.query = query;
window.where = where;
window.orderBy = orderBy;
window.limit = limit;
window.writeBatch = writeBatch;

// Check authentication and initialize data
onAuthStateChanged(auth, (user) => {
    if (user) {
        console.log("User is signed in:", user.email);
        if (!user.email.includes('admin')) {
            window.location.href = 'index.html';
        }
        // Initialize data and listeners
        initializeData();
        initializeRealTimeListeners();
    } else {
        console.log("No user signed in");
        window.location.href = 'index.html';
    }
```

```
});

// Define handleLogout function
window.handleLogout = async function() {
    try {
        await signOut(auth);
        console.log("User signed out successfully");
        window.location.href = 'index.html';
    } catch (error) {
        console.error("Error signing out:", error);
        alert("Error signing out: " + error.message);
    }
};

// Initialize data function
async function initializeData() {
    try {
        await loadDashboard();
        await loadEvents();
        await loadTickets();
        await updateTicketStats();
    } catch (error) {
        console.error("Error initializing data:", error);
    }
}


function initializeDashboardListeners() {
    const eventsRef = collection(db, 'events');
    const ticketsRef = collection(db, 'tickets');

    const unsubscribeTickets = onSnapshot(ticketsRef, () => {
        if (document.getElementById('dashboard').style.display === 'block') {
            loadDashboard();
        }
    });

    const unsubscribeEvents = onSnapshot(eventsRef, () => {
        if (document.getElementById('dashboard').style.display === 'block') {
            loadDashboard();
        }
    });

    return () => {
        unsubscribeTickets();
        unsubscribeEvents();
    };
}
// Navigation function
async function showSection(sectionId) {
    try {
        console.log(`Showing section: ${sectionId}`);

        // Hide all sections first
        document.querySelectorAll('.content-section').forEach(section => {
            section.style.display = 'none';
        });

        // Show selected section
        const selectedSection = document.getElementById(sectionId);
        if (selectedSection) {
            selectedSection.style.display = 'block';
        }

        // Update menu items
        document.querySelectorAll('.menu-item').forEach(item => {
            item.classList.remove('active');
        });
        const menuItem = document.querySelector(`.menu-item[onclick*="${sectionId}"]`);
        if (menuItem) {
            menuItem.classList.add('active');
        }

        // Initialize section-specific content
        if (sectionId === 'dashboard') {
            await loadDashboard();
            initializeDashboardListeners();
        }
```

```
    } catch (error) {
        console.error(`Error showing section ${sectionId}:`, error);
    }
}

// Update populateEventSelector function to properly handle passes
async function populateEventSelector(selectorId) {
    try {
        const selector = document.getElementById(selectorId);
        if (!selector) return;

        selector.innerHTML = '<option value="">Select an Event</option>';
        const now = new Date();

        // Get events from Firestore
        const eventsRef = collection(db, 'events');
        const eventsSnapshot = await getDocs(eventsRef);

        eventsSnapshot.forEach(doc => {
            const event = doc.data();
            const bookingStart = new Date(event.bookingStartDate);
            const bookingEnd = new Date(event.bookingDeadline);

            // Only show events with open bookings
            if (now >= bookingStart && now <= bookingEnd) {
                const option = document.createElement('option');
                option.value = doc.id;
                option.textContent = event.name;
                selector.appendChild(option);
            }
        });
    } catch (error) {
        console.error("Error populating event selector:", error);
    }
}

// Make sure db is properly initialized and available
if (!window.db) {
    console.error('Firebase db not initialized');
}

// Make functions globally available
window.populateEventSelector = populateEventSelector;

// Dashboard function
async function loadDashboard() {
    try {
        const eventsRef = collection(db, 'events');
        const ticketsRef = collection(db, 'tickets');

        const [eventsSnapshot, ticketsSnapshot] = await Promise.all([
            getDocs(eventsRef),
            getDocs(ticketsRef)
        ]);

        let stats = {
            totalEvents: 0,
            upcomingEvents: 0,
            completedEvents: 0,
            totalTickets: 0,
            ticketsSold: 0,
            availableTickets: 0,
            totalRevenue: 0,
            ticketsScanned: 0
        };

        // Calculate stats
        eventsSnapshot.forEach(doc => {
            const event = doc.data();
            if (!event.deleted && event.status !== 'deleted') {
                stats.totalEvents++;

                if (new Date(event.date) > new Date()) {
                    stats.upcomingEvents++;
                } else {
                    stats.completedEvents++;
                }
```

```
        if (event.eventType === 'hybrid') {
            stats.totalTickets += (parseInt(event.tickets?.venue?.total) || 0) +
                                  (parseInt(event.tickets?.online?.total) || 0);
        } else {
            stats.totalTickets += parseInt(event.totalTickets) || 0;
        }
    }
});

// Calculate ticket stats
ticketsSnapshot.forEach(doc => {
    const ticket = doc.data();
    if (ticket.paymentStatus === 'completed') {
        const ticketCount = parseInt(ticket.ticketCount) || 1;
        stats.ticketsSold += ticketCount;
        stats.totalRevenue += parseFloat(ticket.totalPrice) || 0;

        if (ticket.used) {
            stats.ticketsScanned += ticketCount;
        }
    }
});

stats.availableTickets = Math.max(0, stats.totalTickets - stats.ticketsSold);

// Get dashboard container
const dashboardContent = document.getElementById('dashboard');
if (!dashboardContent) return;

// Create/update stats grid
let statsGrid = dashboardContent.querySelector('.stats-grid');
if (!statsGrid) {
    statsGrid = document.createElement('div');
    statsGrid.className = 'stats-grid';
    dashboardContent.appendChild(statsGrid);
}

// Update stats display
statsGrid.innerHTML = `
    <div class="stat-card">
        <h3>Total Events</h3>
        <div class="number" id="totalEvents">${stats.totalEvents}</div>
    </div>
    <div class="stat-card">
        <h3>Upcoming Events</h3>
        <div class="number" id="upcomingEvents">${stats.upcomingEvents}</div>
    </div>
    <div class="stat-card">
        <h3>Completed Events</h3>
        <div class="number" id="completedEvents">${stats.completedEvents}</div>
    </div>
    <div class="stat-card">
        <h3>Total Tickets</h3>
        <div class="number" id="totalTickets">${stats.totalTickets}</div>
    </div>
    <div class="stat-card">
        <h3>Tickets Sold</h3>
        <div class="number" id="ticketsSold">${stats.ticketsSold}</div>
    </div>
    <div class="stat-card">
        <h3>Available Tickets</h3>
        <div class="number" id="availableTickets">${stats.availableTickets}</div>
    </div>
    <div class="stat-card">
        <h3>Total Revenue</h3>
        <div class="number" id="totalRevenue">${formatCurrency(stats.totalRevenue)}</div>
    </div>
    <div class="stat-card">
        <h3>Tickets Scanned</h3>
        <div class="number" id="ticketsScanned">${stats.ticketsScanned}</div>
    </div>
`;

// Remove existing carousel
const existingCarousel = dashboardContent.querySelector('.events-carousel-section');
if (existingCarousel) {
    existingCarousel.remove();
```

```
}

// Create carousel section
const carouselSection = document.createElement('div');
carouselSection.className = 'events-carousel-section';
carouselSection.innerHTML = `
    <div class="carousel-container">
        <h2>Event Gallery</h2>
        <div class="swiper event-swiper">
            <div class="swiper-wrapper">
                <!-- Slides will be added here -->
            </div>
            <div class="swiper-pagination"></div>
        </div>
    </div>
`;

// Add carousel after stats grid
statsGrid.insertAdjacentElement('afterend', carouselSection);

// Add event slides
const swiperWrapper = carouselSection.querySelector('.swiper-wrapper');
if (swiperWrapper) {
    const eventsWithImages = eventsSnapshot.docs
        .map(doc => ({
            id: doc.id,
            ...doc.data(),
            imageUrl: Array.isArray(doc.data().images) ? doc.data().images[0] : doc.data().images
        }))
        .filter(event => {
            return event.imageUrl &&
                   typeof event.imageUrl === 'string' &&
                   event.imageUrl.trim() !== '' &&
                   !event.deleted &&
                   event.status !== 'deleted';
        });

    if (eventsWithImages.length > 0) {
        eventsWithImages.forEach(event => {
            const slide = document.createElement('div');
            slide.className = 'swiper-slide';
            slide.innerHTML = `
                <div class="event-slide">
                    <div class="event-slide-image">
                        <img src="${event.imageUrl}"
                             alt="${event.name}"
                             onerror="this.src='assets/images/event-placeholder.jpg'"
                             loading="lazy"/>
                    </div>
                    <div class="event-slide-info">
                        <h3>${event.name}</h3>
                        <p>${new Date(event.date).toLocaleDateString()}</p>
                    </div>
                </div>
            `;
            swiperWrapper.appendChild(slide);
        });

        // Initialize Swiper
        setTimeout(() => {
            new Swiper('.event-swiper', {
                slidesPerView: 1,
                spaceBetween: 0,
                loop: true,
                autoplay: {
                    delay: 5000,
                    disableOnInteraction: false,
                },
                pagination: {
                    el: '.swiper-pagination',
                    clickable: true,
                },
                effect: 'fade',
                fadeEffect: {
                    crossFade: true
                }
            });
        }, 100);
```

```
            }
        }

    } catch (error) {
        console.error("Error loading dashboard:", error);
    }
}
        // Update recent tickets display
        async function loadRecentTickets() {
            try {
                const recentTicketsList = document.getElementById('recentTicketsList');
                // Clear existing content
                recentTicketsList.innerHTML = '';

                // Get all active events
                const eventsRef = collection(db, 'events');
                const eventsSnapshot = await getDocs(eventsRef);
                const activeEventIds = new Set();

                eventsSnapshot.docs.forEach(doc => {
                    const event = doc.data();
                    if (!event.deleted && event.status !== 'deleted') {
                        activeEventIds.add(doc.id);
                    }
                });

                // Query recent tickets
                const ticketsRef = collection(db, 'tickets');
                const recentTicketsQuery = query(
                    ticketsRef,
                    orderBy('purchaseDate', 'desc'),
                    limit(5)
                );
                const ticketsSnapshot = await getDocs(recentTicketsQuery);

                // Process tickets
                for (const ticketDoc of ticketsSnapshot.docs) {
                    const ticket = ticketDoc.data();

                    // Skip tickets for deleted events
                    if (!activeEventIds.has(ticket.eventId)) continue;

                    const eventDoc = await getDoc(doc(db, 'events', ticket.eventId));
                    if (!eventDoc.exists()) continue;

                    const event = eventDoc.data();
                    const purchaseDate = new Date(ticket.purchaseDate).toLocaleDateString();

                    const ticketItem = document.createElement('div');
                    ticketItem.className = 'recent-item';
                    ticketItem.innerHTML = `
                        <div class="recent-item-title">${event.name}</div>
                        <div class="recent-item-details">
                            <span>${purchaseDate}</span>
                            <span>${ticket.ticketCount} ticket(s)</span>
                            <span>${formatCurrency(ticket.totalPrice)}</span>
                        </div>
                    `;
                    recentTicketsList.appendChild(ticketItem);
                }

                if (recentTicketsList.children.length === 0) {
                    recentTicketsList.innerHTML = '<div class="no-data">No recent tickets</div>';
                }

            } catch (error) {
                console.error("Error loading recent tickets:", error);
                document.getElementById('recentTicketsList').innerHTML =
                    '<div class="error-message">Error loading recent tickets</div>';
            }
        }

// Add these functions to handle verification
async function verifyTicketManually() {
    const ticketId = document.getElementById('ticketIdInput').value.trim();
    if (!ticketId) {
        alert('Please enter a ticket ID');
        return;
```

```
    }

    const result = await verifyTicket(ticketId);
    showVerificationResult(result);
}

// Initialize QR Scanner
async function initQRScanner() {
    if (!window.html5QrcodeScanner) {
        window.html5QrcodeScanner = new Html5QrcodeScanner(
            "qrReader",
            { fps: 10, qrbox: { width: 250, height: 250 } }
        );
    }
}

// Start QR Scanner
async function startQRScanner() {
    try {
        await initQRScanner();
        window.html5QrcodeScanner.render(async (decodedText) => {
            const result = await verifyTicket(decodedText);
            showVerificationResult(result);
            await stopQRScanner();
        });
        document.querySelector('.scan-btn').style.display = 'none';
        document.querySelector('[onclick="stopQRScanner()"]').style.display = 'block';
    } catch (error) {
        console.error("Error starting QR scanner:", error);
        alert("Error starting scanner: " + error.message);
    }
}

// Stop QR Scanner
async function stopQRScanner() {
    try {
        if (window.html5QrcodeScanner) {
            await window.html5QrcodeScanner.clear();
        }
        document.querySelector('.scan-btn').style.display = 'block';
        document.querySelector('[onclick="stopQRScanner()"]').style.display = 'none';
    } catch (error) {
        console.error("Error stopping QR scanner:", error);
    }
}

// Make functions globally available
window.verifyTicketManually = verifyTicketManually;
window.initQRScanner = initQRScanner;
window.startQRScanner = startQRScanner;
window.stopQRScanner = stopQRScanner;

// Initialize passes section
async function initializePassesSection() {
    try {
        console.log('Starting passes section initialization...');
        await loadPassesStats();
        await loadAllPasses(); // This will now create and initialize filters
        console.log('Passes section initialized successfully');
    } catch (error) {
        console.error('Error in initializePassesSection:', error);
    }
}

// Setup role selector
function setupRoleSelector() {
    const roleSelector = document.getElementById('roleSelector');
    const otherRoleInput = document.getElementById('otherRoleInput');

    roleSelector.addEventListener('change', function() {
        otherRoleInput.style.display = this.value === 'other' ? 'block' : 'none';
    });
}

// Update loadPassesStats function
async function loadPassesStats() {
    try {
        const passesRef = collection(db, 'passes');
```

```
        const passesSnapshot = await getDocs(passesRef);

        let totalPasses = 0;
        let usedPasses = 0;
        let unusedPasses = 0;

        passesSnapshot.forEach(doc => {
            const pass = doc.data();
            totalPasses++;
            if (pass.used) {
                usedPasses++;
            } else {
                unusedPasses++;
            }
        });

        // Update stats display
        document.getElementById('totalPasses').textContent = totalPasses;
        document.getElementById('usedPasses').textContent = usedPasses;
        document.getElementById('unusedPasses').textContent = unusedPasses;

    } catch (error) {
        console.error("Error loading passes stats:", error);
    }
}

// Update generatePasses function
async function generatePasses() {
    try {
        const eventId = document.getElementById('eventSelectorForPasses').value;
        const roleSelector = document.getElementById('roleSelector');
        const customRoleInput = document.getElementById('customRole');
        const numberOfPasses = parseInt(document.getElementById('numberOfPasses').value);

        // Validation
        if (!eventId) {
            alert('Please select an event');
            return;
        }
        if (!roleSelector.value) {
            alert('Please select a role');
            return;
        }
        if (roleSelector.value === 'other' && !customRoleInput.value.trim()) {
            alert('Please enter a custom role');
            return;
        }
        if (!numberOfPasses || numberOfPasses < 1) {
            alert('Please enter a valid number of passes');
            return;
        }

        // Determine final role value
        const finalRole = roleSelector.value === 'other' ? customRoleInput.value.trim() : roleSelector.value;

        const batch = writeBatch(db);
        const passesRef = collection(db, 'passes');
        const generatedPasses = [];

        for (let i = 0; i < numberOfPasses; i++) {
            const passData = {
                eventId,
                role: finalRole,
                passIdentifier: generatePassIdentifier(),
                createdAt: serverTimestamp(),
                used: false
            };

            const newPassRef = doc(passesRef);
            batch.set(newPassRef, passData);
            generatedPasses.push(passData);
        }

        await batch.commit();

        // Display generated passes
        displayGeneratedPasses(generatedPasses);
```

```
        // Update stats
        await loadPassesStats();

        // Clear form
        customRoleInput.value = '';
        document.getElementById('numberOfPasses').value = '';
        document.getElementById('eventSelectorForPasses').value = '';
        roleSelector.value = '';
        document.getElementById('otherRoleInput').style.display = 'none';

        alert(`Successfully generated ${numberOfPasses} passes!`);

    } catch (error) {
        console.error("Error generating passes:", error);
        alert('Error generating passes: ' + error.message);
    }
}

// Generate unique pass identifier
function generatePassIdentifier() {
    const randomNum = generateRandomFiveDigits();
    const year = getCurrentYear();
    return `PASS-${randomNum}-${year}`;
}

// Update displayGeneratedPasses function
function displayGeneratedPasses(passes) {
    const container = document.getElementById('generatedPassesList');
    if (!container) {
        console.error('Generated passes container not found');
        return;
    }

    container.innerHTML = passes.map(pass => `
        <div class="generated-pass">
            <div class="pass-info">
                <p><strong>Pass ID:</strong> ${pass.passIdentifier}</p>
                <p><strong>Role:</strong> ${pass.role}</p>
            </div>
            <div class="pass-qr" id="qr-${pass.passIdentifier}"></div>
        </div>
    `).join('');

    // Generate QR codes for each pass
    passes.forEach(pass => {
        new QRCode(document.getElementById(`qr-${pass.passIdentifier}`), {
            text: pass.passIdentifier,
            width: 128,
            height: 128
        });
    });
}

// Make functions globally available
window.generatePasses = generatePasses;
window.togglePassesView = function(view) {
    document.querySelectorAll('.passes-section').forEach(section => {
        section.style.display = 'none';
    });
    document.getElementById(`${view}PassesSection`).style.display = 'block';

    document.querySelectorAll('.passes-toggle .toggle-btn').forEach(btn => {
        btn.classList.remove('active');
    });
    document.querySelector(`.toggle-btn[onclick*="${view}"]`).classList.add('active');
};

// Add cleanup verification function
function cleanupVerification() {
    if (window.html5QrcodeScanner) {
        window.html5QrcodeScanner.clear();
        window.html5QrcodeScanner = null;
    }
    // Reset verification result display
    const resultDiv = document.getElementById('verificationResult');
    if (resultDiv) {
        resultDiv.style.display = 'none';
    }
```

```
    // Reset manual input
    const ticketInput = document.getElementById('ticketIdInput');
    if (ticketInput) {
        ticketInput.value = '';
    }
}

// Update switchVerificationMethod function
function switchVerificationMethod(method) {
    try {
        // Stop QR scanner if it's running
        if (window.html5QrcodeScanner) {
            window.html5QrcodeScanner.clear();
            window.html5QrcodeScanner = null;
        }

        // Get all elements
        const tabButtons = document.querySelectorAll('.verification-tabs .tab-btn');
        const verificationMethods = document.querySelectorAll('.verification-method');
        const activeButton = document.querySelector(`.tab-btn[onclick*="'${method}'"]`);
        const activeMethod = document.getElementById(`${method}Verification`);
        const scanButton = document.querySelector('.scan-btn');
        const stopButton = document.querySelector('[onclick="stopQRScanner()"]');

        // Update tab buttons
        tabButtons.forEach(btn => btn.classList.remove('active'));
        if (activeButton) {
            activeButton.classList.add('active');
        }

        // Update verification methods display
        verificationMethods.forEach(el => {
            if (el) el.style.display = 'none';
        });
        if (activeMethod) {
            activeMethod.style.display = 'block';
        }

        // Handle QR scanner specific elements
        if (method === 'qr') {
            window.initQRScanner();
            if (scanButton) scanButton.style.display = 'block';
            if (stopButton) stopButton.style.display = 'none';
        }
    } catch (error) {
        console.error("Error switching verification method:", error);
    }
}

// Make functions globally available
window.cleanupVerification = cleanupVerification;
window.switchVerificationMethod = switchVerificationMethod;

// Add a variable to store the current ticket ID
let currentTicketId = null;

// Update showVerificationResult to store the ticket ID
window.showVerificationResult = function(result) {
    const resultDiv = document.getElementById('verificationResult');
    const statusSpan = resultDiv.querySelector('.result-status');
    const detailsDiv = resultDiv.querySelector('.ticket-details');
    const markUsedBtn = resultDiv.querySelector('.mark-used-btn');

    // Store the identifier (either ticket or pass)
    currentTicketId = result.ticket?.ticketIdentifier || result.pass?.passIdentifier || null;

    statusSpan.className = 'result-status ' + result.status;
    statusSpan.textContent = result.message;

    if ((result.ticket || result.pass) && result.event) {
        const item = result.ticket || result.pass;
        detailsDiv.innerHTML = `
            <h4>${result.ticket ? 'Ticket' : 'Pass'} Information</h4>
            <p><strong>ID:</strong> ${item.ticketIdentifier || item.passIdentifier}</p>
            <p><strong>Event:</strong> ${result.event.name}</p>
            <p><strong>Event Date:</strong> ${new Date(result.event.date).toLocaleString()}</p>
            ${result.ticket ? `
                <p><strong>Ticket Type:</strong> ${result.ticket.ticketType || 'Standard'}</p>
```

```
            <p><strong>Purchase Date:</strong> ${new Date(result.ticket.purchaseDate).toLocaleString()}</p>
            <p><strong>Purchased By:</strong> ${result.ticket.userEmail}</p>
          ` : `
            <p><strong>Pass Role:</strong> ${result.pass.role}</p>
            <p><strong>Created:</strong> ${new Date(result.pass.createdAt.toDate()).toLocaleString()}</p>
          `}
        `;
        markUsedBtn.style.display = result.status === 'valid' ? 'block' : 'none';
    } else {
        detailsDiv.innerHTML = '';
        markUsedBtn.style.display = 'none';
    }

    resultDiv.style.display = 'block';
};

// Update markTicketAsUsed to use the stored ticket ID
async function markTicketAsUsed() {
    try {
        if (!currentTicketId) {
            throw new Error('No valid ticket/pass selected');
        }

        // First try to find and update ticket
        const ticketsRef = collection(db, 'tickets');
        const ticketQuery = query(ticketsRef, where('ticketIdentifier', '==', currentTicketId));
        const ticketSnapshot = await getDocs(ticketQuery);

        if (!ticketSnapshot.empty) {
            const ticketDoc = ticketSnapshot.docs[0];
            await updateDoc(ticketDoc.ref, {
                used: true,
                usedDate: new Date().toISOString()
            });
            alert('Ticket marked as used successfully!');
        } else {
            // If not found in tickets, try passes
            const passesRef = collection(db, 'passes');
            const passQuery = query(passesRef, where('passIdentifier', '==', currentTicketId));
            const passSnapshot = await getDocs(passQuery);

            if (!passSnapshot.empty) {
                const passDoc = passSnapshot.docs[0];
                await updateDoc(passDoc.ref, {
                    used: true,
                    usedDate: serverTimestamp()
                });
                alert('Pass marked as used successfully!');
            } else {
                throw new Error('Ticket/Pass not found');
            }
        }

        document.getElementById('verificationResult').style.display = 'none';
        currentTicketId = null; // Reset the current ticket/pass ID

    } catch (error) {
        console.error("Error marking ticket/pass as used:", error);
        alert('Error marking ticket/pass as used: ' + error.message);
    }
}

// Add verify ticket function
async function verifyTicket(identifier) {
    try {
        // First check tickets collection
        const ticketsRef = collection(db, 'tickets');
        const ticketQuery = query(ticketsRef, where('ticketIdentifier', '==', identifier));
        const ticketSnapshot = await getDocs(ticketQuery);

        // If found in tickets
        if (!ticketSnapshot.empty) {
            const ticketDoc = ticketSnapshot.docs[0];
            const ticket = ticketDoc.data();

            // Get event details
            const eventDoc = await getDoc(doc(db, 'events', ticket.eventId));
            if (!eventDoc.exists()) {
```

```
            return {
                status: 'invalid',
                message: 'Event not found',
                ticket,
                event: null
            };
        }

        const event = eventDoc.data();
        const now = new Date();
        const eventDate = new Date(event.date);

        if (ticket.used) {
            return {
                status: 'used',
                message: 'Ticket already used',
                ticket,
                event
            };
        }

        if (eventDate < now) {
            return {
                status: 'expired',
                message: 'Event has ended',
                ticket,
                event
            };
        }

        return {
            status: 'valid',
            message: 'Valid Ticket',
            ticket,
            event
        };
    }

    // If not found in tickets, check passes collection
    const passesRef = collection(db, 'passes');
    const passQuery = query(passesRef, where('passIdentifier', '==', identifier));
    const passSnapshot = await getDocs(passQuery);

    // If found in passes
    if (!passSnapshot.empty) {
        const passDoc = passSnapshot.docs[0];
        const pass = passDoc.data();

        // Get event details
        const eventDoc = await getDoc(doc(db, 'events', pass.eventId));
        if (!eventDoc.exists()) {
            return {
                status: 'invalid',
                message: 'Event not found',
                pass,
                event: null
            };
        }

        const event = eventDoc.data();

        if (pass.used) {
            return {
                status: 'used',
                message: 'Pass already used',
                pass,
                event
            };
        }

        return {
            status: 'valid',
            message: 'Valid Pass',
            pass,
            event
        };
    }
```

```
        // If not found in either collection
        return {
            status: 'invalid',
            message: 'Invalid Ticket/Pass',
            ticket: null,
            pass: null,
            event: null
        };

    } catch (error) {
        console.error("Error verifying ticket/pass:", error);
        return {
            status: 'error',
            message: 'Error verifying ticket/pass',
            ticket: null,
            pass: null,
            event: null
        };
    }
}

// Make functions globally available
window.verifyTicket = verifyTicket;
window.markTicketAsUsed = markTicketAsUsed;
async function getRealTimeAvailability(eventId, ticketType) {
    try {
        const eventDoc = await getDoc(doc(db, 'events', eventId));
        if (!eventDoc.exists()) {
            throw new Error('Event not found');
        }
        const event = eventDoc.data();

        // Get all sold tickets
        const ticketsRef = collection(db, 'tickets');
        const ticketQuery = query(ticketsRef,
            where('eventId', '==', eventId),
            where('ticketType', '==', ticketType),
            where('paymentStatus', '==', 'completed')
        );
        const ticketSnapshot = await getDocs(ticketQuery);

        // Calculate total sold tickets
        let soldTickets = 0;
        ticketSnapshot.forEach(doc => {
            const ticket = doc.data();
            soldTickets += Number(ticket.ticketCount) || 0;
        });

        // Get total and available tickets based on event type
        let totalTickets = 0;
        let availableTickets = 0;

        if (event.eventType === 'hybrid') {
            if (ticketType === 'venue') {
                totalTickets = event.tickets.venue.total;
                availableTickets = event.tickets.venue.available;
            } else {
                totalTickets = event.tickets.online.total;
                availableTickets = event.tickets.online.available;
            }
        } else {
            totalTickets = event.totalTickets;
            availableTickets = event.availableTickets;
        }

        return {
            total: totalTickets,
            sold: soldTickets,
            available: availableTickets // Use the stored available count instead of calculating
        };
    } catch (error) {
        console.error('Error getting ticket availability:', error);
        throw error;
    }
}

// Function to update ticket type options based on selected event
async function updateTicketTypeOptions() {
```

```
    console.log('Updating ticket type options...');
    const eventId = document.getElementById('ticketEventSelector').value;
    const typeSelector = document.getElementById('ticketTypeSelector');
    const paymentModeSection = document.getElementById('paymentMode')?.parentElement;

    if (!eventId || !typeSelector) return;

    // Clear existing options
    typeSelector.innerHTML = '<option value="">Select Ticket Type</option>';

    try {
        const eventDoc = await getDoc(doc(db, 'events', eventId));
        const event = eventDoc.data();

        const isFreeEvent = event.pricingType === 'free' || event.price === 0;

        if (paymentModeSection) {
            paymentModeSection.style.display = isFreeEvent ? 'none' : 'block';
            if (isFreeEvent) {
                document.getElementById('paymentMode').value = 'free';
            }
        }

        if (event.eventType === 'hybrid') {
            // Get real-time availability for both types
            const venueAvailability = await getRealTimeAvailability(eventId, 'venue');
            const onlineAvailability = await getRealTimeAvailability(eventId, 'online');

            console.log('Current availability:', {
                venue: venueAvailability,
                online: onlineAvailability
            });

            // Add venue option if available
            if (venueAvailability.available > 0) {
                typeSelector.innerHTML += `
                    <option value="venue">
                        Venue Ticket (${isFreeEvent ? 'Free' : '₹' + event.tickets.venue.price})
                        - ${venueAvailability.available} available
                    </option>
                `;
            }

            // Add online option if available
            if (onlineAvailability.available > 0) {
                typeSelector.innerHTML += `
                    <option value="online">
                        Online Ticket (${isFreeEvent ? 'Free' : '₹' + event.tickets.online.price})
                        - ${onlineAvailability.available} available
                    </option>
                `;
            }
        } else {
            // For standard events
            const availability = await getRealTimeAvailability(eventId, 'standard');
            console.log('Current availability:', availability);

            if (availability.available > 0) {
                typeSelector.innerHTML += `
                    <option value="standard">
                        Standard Ticket (${isFreeEvent ? 'Free' : '₹' + event.price})
                        - ${availability.available} available
                    </option>
                `;
            }
        }

        // Update price display
        updateTicketPrice();

    } catch (error) {
        console.error("Error updating ticket options:", error);
    }
}
// Function to update ticket price display
function updateTicketPrice() {
    const quantity = parseInt(document.getElementById('ticketQuantity').value) || 0;
    const typeSelector = document.getElementById('ticketTypeSelector');
```

```
    const selectedOption = typeSelector.options[typeSelector.selectedIndex];

    let pricePerTicket = 0;
    if (selectedOption && selectedOption.text) {
        // Check if it's a free ticket
        if (selectedOption.text.includes('Free')) {
            pricePerTicket = 0;
        } else {
            // Extract price only if it's a paid ticket
            const priceMatch = selectedOption.text.match(/₹(\d+)/);
            pricePerTicket = priceMatch ? parseInt(priceMatch[1]) : 0;
        }
    }

    const totalPrice = pricePerTicket * quantity;

    document.getElementById('pricePerTicket').textContent = pricePerTicket === 0 ? 'Free' : `₹${pricePerTicket}`;
    document.getElementById('totalTicketPrice').textContent = totalPrice === 0 ? 'Free' : `₹${totalPrice}`;
}

// Function to generate ticket for user
async function generateTicketForUser() {
    try {
        const eventId = document.getElementById('ticketEventSelector').value;
        const ticketType = document.getElementById('ticketTypeSelector').value;
        const quantity = parseInt(document.getElementById('ticketQuantity').value);
        const userEmail = document.getElementById('ticketUserEmail').value.trim();
        const paymentMode = document.getElementById('paymentMode').value;

        // Validate inputs
        if (!eventId || !ticketType || !quantity || !userEmail) {
            const missingFields = [];
            if (!eventId) missingFields.push('Event');
            if (!ticketType) missingFields.push('Ticket Type');
            if (!quantity) missingFields.push('Quantity');
            if (!userEmail) missingFields.push('Email');

            alert(`Please fill in the following fields: ${missingFields.join(', ')}`);
            return;
        }

        // Check real-time availability
        const availability = await getRealTimeAvailability(eventId, ticketType);
        if (quantity > availability.available) {
            throw new Error(`Only ${availability.available} tickets available`);
        }

        // Get event details
        const eventDoc = await getDoc(doc(db, 'events', eventId));
        if (!eventDoc.exists()) {
            throw new Error('Event not found');
        }

        const event = eventDoc.data();
        const isFreeEvent = event.pricingType === 'free' || event.price === 0;

        // For paid events, handle payment
        if (!isFreeEvent && paymentMode === 'card') {
            showCardDetailsModal();
            return;
        }

        // Process ticket generation
        await processTicketGeneration(false);

    } catch (error) {
        console.error("Error generating ticket:", error);
        alert('Error generating ticket: ' + error.message);
    }
}

// Add function to update ticket stats
// Update the updateTicketStats function to handle filtered tickets

async function updateTicketStats(selectedEventId = null) {
    try {
        const ticketsRef = collection(db, 'tickets');
        const eventsRef = collection(db, 'events');
```

```
// Get event data first
let eventCapacity = 0;
if (selectedEventId && selectedEventId !== 'all') {
    const eventDoc = await getDoc(doc(eventsRef, selectedEventId));
    if (eventDoc.exists()) {
        const eventData = eventDoc.data();
        if (eventData.eventType === 'hybrid') {
            document.getElementById('venueTickets').value = eventData.tickets?.venue?.total || '';
            document.getElementById('onlineTickets').value = eventData.tickets?.online?.total || '';
            document.getElementById('venueTicketPrice').value = eventData.tickets?.venue?.price || '0';
            document.getElementById('onlineTicketPrice').value = eventData.tickets?.online?.price || '0';
        } else {
            document.getElementById('eventPrice').value = eventData.price || '0';
            document.getElementById('eventTickets').value = eventData.totalTickets || '';
        }
    }
} else {
    // If no event selected, sum up capacity of all events
    const eventsSnapshot = await getDocs(eventsRef);
    eventsSnapshot.forEach(doc => {
        const event = doc.data();
        if (!event.deleted && event.status !== 'deleted') {
            if (event.eventType === 'hybrid') {
                eventCapacity += (parseInt(event.tickets?.venue?.total) || 0) +
                                 (parseInt(event.tickets?.online?.total) || 0);
            } else {
                eventCapacity += parseInt(event.totalTickets) || 0;
            }
        }
    });
}

// Build ticket query
let ticketQuery;
if (selectedEventId && selectedEventId !== 'all') {
    ticketQuery = query(ticketsRef, where('eventId', '==', selectedEventId));
} else {
    ticketQuery = query(ticketsRef);
}

const ticketSnapshot = await getDocs(ticketQuery);

let stats = {
    totalTickets: eventCapacity,
    ticketsSold: 0,
    ticketsScanned: 0,
    totalRevenue: 0
};

// Calculate ticket stats
ticketSnapshot.forEach(doc => {
    const ticket = doc.data();
    if (ticket.paymentStatus === 'completed') {
        const ticketCount = parseInt(ticket.ticketCount) || 1;
        stats.ticketsSold += ticketCount;
        stats.totalRevenue += parseFloat(ticket.totalPrice) || 0;

        if (ticket.used) {
            stats.ticketsScanned += ticketCount;
        }
    }
});

// Calculate available tickets
stats.availableTickets = Math.max(0, stats.totalTickets - stats.ticketsSold);

// Update ticket section UI elements
const ticketElements = {
    totalTickets: document.getElementById('adminTotalTickets'),
    soldTickets: document.getElementById('adminSoldTickets'),
    availableTickets: document.getElementById('adminAvailableTickets'),
    usedTickets: document.getElementById('selectedEventUsedTickets'),
    revenue: document.getElementById('selectedEventTotalRevenue')
};

// Update elements if they exist
if (ticketElements.totalTickets) ticketElements.totalTickets.textContent = stats.totalTickets;
```

```
        if (ticketElements.soldTickets) ticketElements.soldTickets.textContent = stats.ticketsSold;
        if (ticketElements.availableTickets) ticketElements.availableTickets.textContent = stats.availableTickets;
        if (ticketElements.usedTickets) ticketElements.usedTickets.textContent = stats.ticketsScanned;
        if (ticketElements.revenue) ticketElements.revenue.textContent = formatCurrency(stats.totalRevenue);

        console.log('Ticket Page Stats:', stats);

    } catch (error) {
        console.error('Error updating ticket stats:', error);
    }
}

// Add event listeners for filters
document.addEventListener('DOMContentLoaded', () => {
    const eventSelector = document.getElementById('adminEventSelector');
    const searchInput = document.getElementById('adminTicketSearch');

    if (eventSelector) {
        eventSelector.addEventListener('change', () => {
            updateTicketStats(eventSelector.value);
        });
    }

    if (searchInput) {
        searchInput.addEventListener('input', debounce(() => {
            const selectedEventId = eventSelector?.value || 'all';
            updateTicketStats(selectedEventId);
        }, 300));
    }
});

document.addEventListener('DOMContentLoaded', () => {
    const eventSelector = document.getElementById('adminEventSelector');
    const searchInput = document.getElementById('adminTicketSearch');

    if (eventSelector) {
        eventSelector.addEventListener('change', () => {
            updateTicketStats(eventSelector.value);
        });
    }

    if (searchInput) {
        searchInput.addEventListener('input', debounce(() => {
            updateTicketStats(eventSelector?.value);
        }, 300));
    }

    // Initial load
    updateTicketStats('all');
});
// Helper function to clear ticket stats
// Helper function to clear ticket stats
function clearTicketStats() {
    const elements = {
        totalTickets: document.getElementById('selectedEventTotalTickets'),
        soldTickets: document.getElementById('selectedEventSoldTickets'),
        usedTickets: document.getElementById('selectedEventUsedTickets'),
        availableTickets: document.getElementById('selectedEventAvailableTickets'),
        totalRevenue: document.getElementById('selectedEventTotalRevenue')
    };

    // Only update elements that exist
    if (elements.totalTickets) elements.totalTickets.textContent = '0';
    if (elements.soldTickets) elements.soldTickets.textContent = '0';
    if (elements.usedTickets) elements.usedTickets.textContent = '0';
    if (elements.availableTickets) elements.availableTickets.textContent = '0';
    if (elements.totalRevenue) elements.totalRevenue.textContent = '₹0';
}

// Helper function for currency formatting
function formatCurrency(amount) {
    return '₹' + amount.toLocaleString('en-IN');
}

// Make functions globally available
window.closeCardModal = closeCardModal;
window.processCardPayment = processCardPayment;
window.updateTicketStats = updateTicketStats;
```

```
// Function to collect payment
async function collectPayment() {
    try {
        const ticketId = document.getElementById('paymentTicketId').value;
        const amount = parseFloat(document.getElementById('paymentAmount').value);
        const method = document.getElementById('paymentMethod').value;

        if (!ticketId || !amount || !method) {
            alert('Please fill in all fields');
            return;
        }

        const ticketsRef = collection(db, 'tickets');
        const q = query(ticketsRef, where('ticketIdentifier', '==', ticketId));
        const querySnapshot = await getDocs(q);

        if (querySnapshot.empty) {
            alert('Ticket not found');
            return;
        }

        const ticketDoc = querySnapshot.docs[0];
        await updateDoc(ticketDoc.ref, {
            paymentStatus: 'completed',
            paymentMethod: method,
            paymentDate: new Date().toISOString()
        });

        alert('Payment collected successfully!');

    } catch (error) {
        console.error("Error collecting payment:", error);
        alert('Error collecting payment: ' + error.message);
    }
}

// Make functions globally available
window.generateTicketForUser = generateTicketForUser;
window.updateTicketTypeOptions = updateTicketTypeOptions;
window.updateTicketPrice = updateTicketPrice;
window.collectPayment = collectPayment;

// Add event listeners
document.getElementById('ticketQuantity').addEventListener('input', updateTicketPrice);
document.getElementById('ticketTypeSelector').addEventListener('change', updateTicketPrice);

// Update loadAllPasses function with improved filtering
async function loadAllPasses(roleFilter = 'all', eventFilter = 'all') {
    try {
        const passesRef = collection(db, 'passes');
        const container = document.getElementById('viewPassesList');
        if (!container) {
            console.error('Passes list container not found');
            return;
        }

        // Create filters section if it doesn't exist
        let filtersSection = document.getElementById('passes-filters');
        if (!filtersSection) {
            filtersSection = document.createElement('div');
            filtersSection.id = 'passes-filters';
            filtersSection.className = 'filters-section';
            container.parentNode.insertBefore(filtersSection, container);
        }

        // Update filters UI
        filtersSection.innerHTML = `
            <div class="filter-controls">
                <select id="role-filter" class="filter-select">
                    <option value="all">All Roles</option>
                    <option value="staff">Staff</option>
                    <option value="vip">VIP</option>
                    <option value="media">Media</option>
                    <option value="other">Other Roles</option>
                    <option value="used">Used Passes</option>
                    <option value="unused">Unused Passes</option>
                </select>
```

```
            <select id="event-filter" class="filter-select">
                <option value="all">All Events</option>
                <!-- Events will be populated dynamically -->
            </select>
        </div>
    `;

    // Populate event filter
    const eventSelect = document.getElementById('event-filter');
    const eventsSnapshot = await getDocs(collection(db, 'events'));
    eventsSnapshot.forEach(doc => {
        const event = doc.data();
        const option = document.createElement('option');
        option.value = doc.id;
        option.textContent = event.name;
        eventSelect.appendChild(option);
    });

    // Set filter values if they exist
    if (roleFilter !== 'all') {
        document.getElementById('role-filter').value = roleFilter;
    }
    if (eventFilter !== 'all') {
        document.getElementById('event-filter').value = eventFilter;
    }

    // Add filter event listeners
    document.getElementById('role-filter').addEventListener('change', function() {
        loadAllPasses(this.value, document.getElementById('event-filter').value);
    });

    document.getElementById('event-filter').addEventListener('change', function() {
        loadAllPasses(document.getElementById('role-filter').value, this.value);
    });

    // Build query based on filters
    let q = passesRef;
    const conditions = [];

    if (roleFilter !== 'all') {
        if (roleFilter === 'used') {
            conditions.push(where('used', '==', true));
        } else if (roleFilter === 'unused') {
            conditions.push(where('used', '==', false));
        } else if (roleFilter === 'other') {
            // Filter for roles that are not in the standard roles list
            conditions.push(where('role', 'not-in', ['staff', 'vip', 'media']));
        } else {
            conditions.push(where('role', '==', roleFilter));
        }
    }

    if (eventFilter !== 'all') {
        conditions.push(where('eventId', '==', eventFilter));
    }

    if (conditions.length > 0) {
        q = query(passesRef, ...conditions);
    }

    // Get passes
    const passesSnapshot = await getDocs(q);

    // Clear and setup container
    container.innerHTML = `
        <div class="passes-actions">
            <button onclick="downloadSelectedPasses()" class="bulk-action-btn">
                <i class="fas fa-download"></i> Download Selected
            </button>
            <button onclick="deleteSelectedPasses()" class="bulk-action-btn">
                <i class="fas fa-trash"></i> Delete Selected
            </button>
        </div>
        <div class="passes-list"></div>
    `;

    const passesList = container.querySelector('.passes-list');
```

```
        // Display passes
        if (passesSnapshot.empty) {
            passesList.innerHTML = '<p class="no-results">No passes found</p>';
            return;
        }

        passesSnapshot.forEach(doc => {
            const pass = doc.data();
            const passElement = document.createElement('div');
            passElement.className = 'pass-item';
            passElement.innerHTML = `
                <div class="pass-checkbox">
                    <input type="checkbox" class="pass-select" data-pass-id="${doc.id}">
                </div>
                <div class="pass-details">
                    <p><strong>Pass ID:</strong> ${pass.passIdentifier}</p>
                    <p><strong>Role:</strong> ${pass.role}</p>
                    <p><strong>Event:</strong> <span id="event-${doc.id}">Loading...</span></p>
                    <p><strong>Status:</strong> ${pass.used ? 'Used' : 'Unused'}</p>
                    <p><strong>Created:</strong> ${pass.createdAt.toDate().toLocaleString()}</p>
                </div>
                <div class="pass-actions">
                    <button onclick="downloadPass('${doc.id}')" class="action-btn">
                        <i class="fas fa-download"></i> Download
                    </button>
                    <button onclick="deletePass('${doc.id}')" class="action-btn">
                        <i class="fas fa-trash"></i> Delete
                    </button>
                </div>
            `;
            passesList.appendChild(passElement);

            // Fetch and display event name
            fetchEventName(pass.eventId, `event-${doc.id}`);
        });

    } catch (error) {
        console.error("Error loading passes:", error);
        alert('Error loading passes: ' + error.message);
    }
}

// Function to download pass
async function downloadPass(passId) {
    const downloadBtn = document.querySelector(`button[onclick="downloadPass('${passId}')"]`);
    const originalText = downloadBtn.innerHTML;

    try {
        downloadBtn.innerHTML = '<i class="fas fa-spinner fa-spin"></i> Generating...';
        downloadBtn.disabled = true;

        // Get pass data
        const passDoc = await getDoc(doc(db, 'passes', passId));
        if (!passDoc.exists()) {
            throw new Error('Pass not found');
        }
        const pass = passDoc.data();

        // Create new jsPDF instance
        const pdf = new jsPDF({
            orientation: 'portrait',
            unit: 'mm',
            format: [210, 297] // A4 size
        });

        // Rest of your PDF generation code remains the same...
        pdf.setFont('helvetica');
        pdf.setFontSize(24);
        pdf.setTextColor(51, 51, 51);

        // Add header
        pdf.text('Event Tix', 105, 30, { align: 'center' });

        // Add role badge
        pdf.setFillColor(102, 126, 234);
        pdf.roundedRect(65, 40, 80, 10, 3, 3, 'F');
        pdf.setTextColor(255, 255, 255);
        pdf.setFontSize(12);
```

```
        pdf.text(pass.role.toUpperCase(), 105, 46, { align: 'center' });

        // Generate QR code
        const qrContainer = document.createElement('div');
        qrContainer.style.display = 'none';
        document.body.appendChild(qrContainer);

        new QRCode(qrContainer, {
            text: pass.passIdentifier,
            width: 128,
            height: 128
        });

        await new Promise(resolve => setTimeout(resolve, 200));

        // Add QR code to PDF
        const qrImage = qrContainer.querySelector('canvas').toDataURL();
        pdf.addImage(qrImage, 'PNG', 70, 60, 70, 70);
        document.body.removeChild(qrContainer);

        // Add pass details
        pdf.setTextColor(51, 51, 51);
        pdf.setFontSize(12);
        pdf.text(`Pass ID: ${pass.passIdentifier}`, 105, 150, { align: 'center' });
        pdf.text(`Access Level: ${pass.role.toUpperCase()}`, 105, 160, { align: 'center' });
        pdf.text(`Issued Date: ${new Date(pass.createdAt.toDate()).toLocaleDateString()}`, 105, 170, { align: 'center' });

        // Add footer
        pdf.setTextColor(220, 53, 69);
        pdf.setFontSize(14);
        pdf.text('AUTHORIZED ACCESS ONLY', 105, 200, { align: 'center' });

        // Save PDF
        pdf.save(`EventTix-Pass-${pass.passIdentifier}.pdf`);

        // Reset button state
        downloadBtn.innerHTML = originalText;
        downloadBtn.disabled = false;

    } catch (error) {
        console.error('Error generating pass:', error);
        alert('Failed to generate pass: ' + error.message);
        downloadBtn.innerHTML = originalText;
        downloadBtn.disabled = false;
    }
}

// Make function available globally
window.jsPDF = window.jspdf.jsPDF;
window.downloadPass = downloadPass;

// Add function to download selected passes
async function downloadSelectedPasses() {
    const selectedPasses = document.querySelectorAll('.pass-select:checked');
    if (selectedPasses.length === 0) {
        alert('Please select at least one pass to download');
        return;
    }

    try {
        for (const checkbox of selectedPasses) {
            const passId = checkbox.dataset.passId;
            await downloadPass(passId);
            // Add small delay between downloads to prevent browser issues
            await new Promise(resolve => setTimeout(resolve, 500));
        }
    } catch (error) {
        console.error("Error downloading passes:", error);
        alert('Error downloading passes: ' + error.message);
    }
}
// Add function to delete selected passes
async function deleteSelectedPasses() {
    const selectedPasses = document.querySelectorAll('.pass-select:checked');
    if (selectedPasses.length === 0) {
        alert('Please select at least one pass to delete');
        return;
    }
```

```
    if (!confirm(`Are you sure you want to delete ${selectedPasses.length} passes?`)) {
        return;
    }

    try {
        const batch = writeBatch(db);
        selectedPasses.forEach(checkbox => {
            const passId = checkbox.dataset.passId;
            batch.delete(doc(db, 'passes', passId));
        });

        await batch.commit();
        await loadPassesStats();

        // Get current filter values from the actual filter elements
        const roleFilter = document.getElementById('role-filter')?.value || 'all';
        const eventFilter = document.getElementById('event-filter')?.value || 'all';

        // Reload passes with current filters
        await loadAllPasses(roleFilter, eventFilter);

        alert('Selected passes deleted successfully');
    } catch (error) {
        console.error("Error deleting passes:", error);
        alert('Error deleting passes: ' + error.message);
    }
}

// Update setupPassSearch function
function setupPassSearch() {
    const searchInput = document.getElementById('passSearchInput');
    const filterSelector = document.getElementById('passFilterSelector');

    if (searchInput) {
        searchInput.addEventListener('input', debounce(() => {
            loadAllPasses(filterSelector.value, searchInput.value);
        }, 300));
    }

    if (filterSelector) {
        filterSelector.addEventListener('change', () => {
            loadAllPasses(filterSelector.value, searchInput?.value || '');
        });
    }
}

// Make sure to export all necessary functions
window.downloadPass = downloadPass;
window.downloadSelectedPasses = downloadSelectedPasses;
window.deletePass = deletePass;
window.deleteSelectedPasses = deleteSelectedPasses;
window.toggleAllPasses = toggleAllPasses;

// Add function to fetch event name
async function fetchEventName(eventId, elementId) {
    try {
        const eventDoc = await getDoc(doc(db, 'events', eventId));
        if (eventDoc.exists()) {
            const eventName = eventDoc.data().name;
            document.getElementById(elementId).textContent = eventName;
        }
    } catch (error) {
        console.error("Error fetching event name:", error);
    }
}

// Add function to toggle pass status
async function togglePassStatus(passId, newStatus) {
    try {
        await updateDoc(doc(db, 'passes', passId), {
            used: newStatus,
            usedDate: newStatus ? serverTimestamp() : null
        });
        await loadPassesStats();
        await loadAllPasses(
            document.getElementById('passFilterSelector').value,
            document.getElementById('passSearchInput').value
```

```
        );
    } catch (error) {
        console.error("Error updating pass status:", error);
        alert('Error updating pass status: ' + error.message);
    }
}

// Add debounce utility function
function debounce(func, wait) {
    let timeout;
    return function executedFunction(...args) {
        const later = () => {
            clearTimeout(timeout);
            func(...args);
        };
        clearTimeout(timeout);
        timeout = setTimeout(later, wait);
    };
}

// Make new functions globally available
window.togglePassStatus = togglePassStatus;
window.setupPassSearch = setupPassSearch;

// Add initialization code
document.addEventListener('DOMContentLoaded', () => {
    // Initialize event selector for passes
    const passesEventSelector = document.getElementById('eventSelectorForPasses');
    if (passesEventSelector) {
        populateEventSelector('eventSelectorForPasses');
    }

    // Initialize role selector
    const roleSelector = document.getElementById('roleSelector');
    if (roleSelector) {
        roleSelector.addEventListener('change', function() {
            const otherRoleInput = document.getElementById('otherRoleInput');
            if (otherRoleInput) {
                otherRoleInput.style.display = this.value === 'other' ? 'block' : 'none';
            }
        });
    }

    // Initialize passes view
    loadPassesStats();
    loadAllPasses();
});

// Make functions globally available
window.generatePasses = generatePasses;
window.populateEventSelector = populateEventSelector;
window.loadPassesStats = loadPassesStats;
window.loadAllPasses = loadAllPasses;

// Add deletePass function
async function deletePass(passId) {
    try {
        if (!confirm('Are you sure you want to delete this pass?')) {
            return;
        }

        await deleteDoc(doc(db, 'passes', passId));
        await loadPassesStats();

        // Get current filter values
        const roleFilter = document.getElementById('role-filter')?.value || 'all';
        const eventFilter = document.getElementById('event-filter')?.value || 'all';

        // Reload passes with current filters
        await loadAllPasses(roleFilter, eventFilter);

        alert('Pass deleted successfully');
    } catch (error) {
        console.error("Error deleting pass:", error);
        alert('Error deleting pass: ' + error.message);
    }
}
```

```
// Add toggle all passes function
function toggleAllPasses(checked) {
    document.querySelectorAll('.pass-select').forEach(checkbox => {
        checkbox.checked = checked;
    });
}

// Make new functions globally available
window.deletePass = deletePass;
window.toggleAllPasses = toggleAllPasses;

// Add some CSS to style the filters
const style = document.createElement('style');
style.textContent = `
    .filters-section {
        margin-bottom: 20px;
        padding: 15px;
        background: #f5f5f5;
        border-radius: 5px;
    }

    .filter-controls {
        display: flex;
        gap: 15px;
        align-items: center;
    }

    .filter-select {
        padding: 8px;
        border: 1px solid #ddd;
        border-radius: 4px;
        min-width: 150px;
    }

    .filter-select:focus {
        outline: none;
        border-color: #007bff;
    }
`;
document.head.appendChild(style);



// ... existing code ...

// Add event listeners when document loads
document.addEventListener('DOMContentLoaded', () => {
    // Initialize event selector for tickets
    const ticketEventSelector = document.getElementById('ticketEventSelector');
    if (ticketEventSelector) {
        ticketEventSelector.addEventListener('change', updateTicketTypeOptions);
        populateEventSelector('ticketEventSelector');
    }

    // Add listeners for ticket quantity and type
    const ticketQuantity = document.getElementById('ticketQuantity');
    if (ticketQuantity) {
        ticketQuantity.addEventListener('input', updateTicketPrice);
    }

    const ticketTypeSelector = document.getElementById('ticketTypeSelector');
    if (ticketTypeSelector) {
        ticketTypeSelector.addEventListener('change', updateTicketPrice);
    }
});

// Make functions globally available

window.updateTicketTypeOptions = updateTicketTypeOptions;
window.updateTicketPrice = updateTicketPrice;



// Function to show card details modal
function showCardDetailsModal() {
    // Create modal HTML with unique ID and additional fields
    const modalHTML = `
```

```
    <div id="ticketCardDetailsModal" class="modal">
        <div class="modal-content">
            <span class="close" onclick="closeCardModal()">&times;</span>
            <h3>Enter Card Details</h3>
            <div class="form-group">
                <label>Card Number</label>
                <input type="text" id="cardNumber" maxlength="16" placeholder="Enter 16-digit card number">
                <small class="helper-text">Enter 16-digit number without spaces</small>
            </div>
            <div class="form-group">
                <label>Name on Card</label>
                <input type="text" id="cardName" placeholder="Enter name as shown on card">
            </div>
            <div class="card-extra-details">
                <div class="form-group expiry-date">
                    <label>Expiry Date</label>
                    <input type="text" id="cardExpiry" maxlength="5" placeholder="MM/YY">
                    <small class="helper-text">Format: MM/YY (e.g., 12/26)</small>
                </div>
                <div class="form-group cvv">
                    <label>CVV</label>
                    <input type="password" id="cardCvv" maxlength="3" placeholder="***">
                    <small class="helper-text">3-digit security code</small>
                </div>
            </div>
            <button onclick="processCardPayment()" class="pay-now-btn">Pay Now</button>
            <div class="loader" style="display: none;">Processing payment...</div>
        </div>
    </div>
`;

// Add modal to document
document.body.insertAdjacentHTML('beforeend', modalHTML);

// Add modal styles
const modalStyles = document.createElement('style');
modalStyles.textContent = `
    #ticketCardDetailsModal {
        display: block;
        position: fixed;
        z-index: 1000;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        background-color: rgba(0,0,0,0.5);
    }

    #ticketCardDetailsModal .modal-content {
        background-color: white;
        margin: 10% auto;
        padding: 25px;
        border-radius: 8px;
        width: 90%;
        max-width: 500px;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
    }

    #ticketCardDetailsModal .close {
        float: right;
        font-size: 24px;
        font-weight: bold;
        cursor: pointer;
        color: #666;
    }

    #ticketCardDetailsModal .close:hover {
        color: #000;
    }

    #ticketCardDetailsModal .form-group {
        margin-bottom: 20px;
    }

    #ticketCardDetailsModal label {
        display: block;
        margin-bottom: 5px;
        font-weight: 500;
```

```
            color: #333;
        }

        #ticketCardDetailsModal input {
            width: 100%;
            padding: 10px;
            border: 1px solid #ddd;
            border-radius: 4px;
            font-size: 16px;
        }

        #ticketCardDetailsModal input:focus {
            border-color: #007bff;
            outline: none;
            box-shadow: 0 0 0 2px rgba(0,123,255,0.25);
        }

        #ticketCardDetailsModal .card-extra-details {
            display: grid;
            grid-template-columns: 1fr 1fr;
            gap: 20px;
        }

        #ticketCardDetailsModal .helper-text {
            display: block;
            margin-top: 5px;
            font-size: 12px;
            color: #666;
        }

        #ticketCardDetailsModal .pay-now-btn {
            background-color: #007bff;
            color: white;
            padding: 12px 24px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            width: 100%;
            font-size: 16px;
            margin-top: 10px;
            transition: background-color 0.2s;
        }

        #ticketCardDetailsModal .pay-now-btn:hover {
            background-color: #0056b3;
        }

        #ticketCardDetailsModal .loader {
            text-align: center;
            margin-top: 15px;
            color: #666;
            display: flex;
            align-items: center;
            justify-content: center;
            gap: 10px;
        }

        #ticketCardDetailsModal .loader::after {
            content: '';
            display: inline-block;
            width: 20px;
            height: 20px;
            border: 2px solid #f3f3f3;
            border-top: 2px solid #3498db;
            border-radius: 50%;
            animation: spin 1s linear infinite;
        }

        @keyframes spin {
            0% { transform: rotate(0deg); }
            100% { transform: rotate(360deg); }
        }
    `;
    document.head.appendChild(modalStyles);

    // Add input validation listeners
    setupCardValidation();
}
```

```
// Function to setup card validation
function setupCardValidation() {
    // Card number validation
    document.getElementById('cardNumber').addEventListener('input', function(e) {
        this.value = this.value.replace(/\D/g, '').slice(0, 16);
    });

    // Expiry date validation
    document.getElementById('cardExpiry').addEventListener('input', function(e) {
        let value = this.value.replace(/\D/g, '');
        if (value.length >= 2) {
            value = value.slice(0, 2) + '/' + value.slice(2, 4);
        }
        this.value = value;
    });

    // CVV validation
    document.getElementById('cardCvv').addEventListener('input', function(e) {
        this.value = this.value.replace(/\D/g, '').slice(0, 3);
    });
}

// Function to close card modal
function closeCardModal() {
    const modal = document.getElementById('ticketCardDetailsModal');
    if (modal) {
        modal.remove();
    }
}

// Function to process card payment
async function processCardPayment() {
   try{// Get all card details
    const cardNumber = document.getElementById('cardNumber').value;
    const cardName = document.getElementById('cardName').value;
    const cardExpiry = document.getElementById('cardExpiry').value;
    const cardCvv = document.getElementById('cardCvv').value;

    // Validate all fields
    if (!/^\d{16}$/.test(cardNumber)) {
        alert('Please enter a valid 16-digit card number');
        return;
    }
    if (!cardName.trim()) {
        alert('Please enter the name on card');
        return;
    }
    if (!/^\d{2}\/\d{2}$/.test(cardExpiry)) {
        alert('Please enter a valid expiry date (MM/YY)');
        return;
    }
    if (!/^\d{3}$/.test(cardCvv)) {
        alert('Please enter a valid 3-digit CVV');
        return;
    }

    // Show loader
    const loader = document.querySelector('#ticketCardDetailsModal .loader');
    const payButton = document.querySelector('#ticketCardDetailsModal .pay-now-btn');
    loader.style.display = 'block';
    loader.textContent = 'Processing payment...';
    payButton.disabled = true;

    // Simulate payment processing
    await new Promise(resolve => setTimeout(resolve, 500));

    // Update loader text
    loader.textContent = 'Generating ticket...';

    // Process ticket generation
    await processTicketGeneration();

    // Close modal after successful ticket generation
    closeCardModal();

} catch (error) {
    console.error("Error processing payment:", error);
```

```
    alert('Error processing payment: ' + error.message);

    // Re-enable pay button and hide loader on error
    const loader = document.querySelector('#ticketCardDetailsModal .loader');
    const payButton = document.querySelector('#ticketCardDetailsModal .pay-now-btn');
    if (loader) loader.style.display = 'none';
    if (payButton) payButton.disabled = false;
}
}

// Function to process ticket generation
async function processTicketGeneration(isCardPayment = false) {
    try {
        const eventId = document.getElementById('ticketEventSelector').value;
        const ticketType = document.getElementById('ticketTypeSelector').value;
        const quantity = parseInt(document.getElementById('ticketQuantity').value);
        const userEmail = document.getElementById('ticketUserEmail').value.trim();
        const paymentMode = document.getElementById('paymentMode').value;

        // Check real-time availability again before proceeding
        const availability = await getRealTimeAvailability(eventId, ticketType);
        if (quantity > availability.available) {
            throw new Error(`Only ${availability.available} tickets available`);
        }

        // Get event details
        const eventDoc = await getDoc(doc(db, 'events', eventId));
        if (!eventDoc.exists()) {
            throw new Error('Event not found');
        }

        const event = eventDoc.data();
        const isFreeEvent = event.pricingType === 'free' || event.price === 0;

        // Calculate price
        let ticketPrice = 0;
        if (!isFreeEvent) {
            if (event.eventType === 'hybrid') {
                ticketPrice = ticketType === 'venue' ?
                    event.tickets.venue.price :
                    event.tickets.online.price;
            } else {
                ticketPrice = event.price;
            }
        }

        const totalPrice = ticketPrice * quantity;

        // Create ticket document
        const ticketData = {
            eventId,
            ticketType,
            ticketCount: quantity,
            userEmail,
            ticketIdentifier: generateTicketId(),
            purchaseDate: new Date().toISOString(),
            paymentStatus: 'completed',
            paymentMethod: isFreeEvent ? 'free' : paymentMode,
            used: false,
            totalPrice: isFreeEvent ? 0 : totalPrice,
            pricePerTicket: isFreeEvent ? 0 : ticketPrice,
            adminGenerated: true,
            generatedBy: auth.currentUser.email
        };

        // Start batch write
        const batch = writeBatch(db);

        // Add ticket document
        const newTicketRef = doc(collection(db, 'tickets'));
        batch.set(newTicketRef, ticketData);

        // Update event available tickets
        const eventRef = doc(db, 'events', eventId);
        if (event.eventType === 'hybrid') {
            const updateField = ticketType === 'venue' ?
                'tickets.venue.available' : 'tickets.online.available';
            batch.update(eventRef, {
```

```
            [updateField]: increment(-quantity),
            lastUpdated: serverTimestamp()
        });
    } else {
        batch.update(eventRef, {
            availableTickets: increment(-quantity),
            lastUpdated: serverTimestamp()
        });
    }

    await batch.commit();

    // Show success message
    alert(`Successfully generated ticket!\nTicket ID: ${ticketData.ticketIdentifier}`);

    // Clear form and update UI
    clearTicketForm();
    await updateTicketTypeOptions();

  } catch (error) {
    console.error("Error processing ticket:", error);
    throw error;
  }
}

// Initialize real-time listeners
function initializeTicketListeners() {
  const ticketsRef = collection(db, 'tickets');

  onSnapshot(ticketsRef, (snapshot) => {
    snapshot.docChanges().forEach(async (change) => {
      if (change.type === "added" || change.type === "modified") {
        const ticketData = change.doc.data();
        const currentEventId = document.getElementById('ticketEventSelector')?.value;
        if (currentEventId === ticketData.eventId) {
          await updateTicketTypeOptions();
        }
      }
    });
  });
}

  // Also listen for event changes
  const eventsRef = collection(db, 'events');
  onSnapshot(eventsRef, (snapshot) => {
    snapshot.docChanges().forEach(async (change) => {
      if (change.type === "modified") {
        const eventData = change.doc.data();
        const currentEventId = document.getElementById('ticketEventSelector')?.value;

        if (currentEventId === change.doc.id) {
          console.log('Updating generate ticket section due to event change');
          await updateTicketTypeOptions();
          await updateTicketStats(currentEventId);
        }
      }
    });
  });

// Make functions globally available
window.getRealTimeAvailability = getRealTimeAvailability;
window.updateTicketTypeOptions = updateTicketTypeOptions;
window.processTicketGeneration = processTicketGeneration;

// Initialize when document loads
document.addEventListener('DOMContentLoaded', () => {
  initializeTicketListeners();
});

// Function to clear ticket form
function clearTicketForm() {
  const elements = {
    ticketEventSelector: document.getElementById('ticketEventSelector'),
    ticketTypeSelector: document.getElementById('ticketTypeSelector'),
    ticketQuantity: document.getElementById('ticketQuantity'),
    ticketUserEmail: document.getElementById('ticketUserEmail'),
    paymentMode: document.getElementById('paymentMode'),
    pricePerTicket: document.getElementById('pricePerTicket'),
```

```
        totalTicketPrice: document.getElementById('totalTicketPrice')
    };

    // Clear form values
    if (elements.ticketEventSelector) elements.ticketEventSelector.value = '';
    if (elements.ticketTypeSelector) elements.ticketTypeSelector.value = '';
    if (elements.ticketQuantity) elements.ticketQuantity.value = '1';
    if (elements.ticketUserEmail) elements.ticketUserEmail.value = '';
    if (elements.paymentMode) elements.paymentMode.value = '';

    // Reset price displays
    if (elements.pricePerTicket) elements.pricePerTicket.textContent = '₹0';
    if (elements.totalTicketPrice) elements.totalTicketPrice.textContent = '₹0';
}

// Add these utility functions near the top of admin.js, after the imports
function generateRandomFiveDigits() {
    return Math.floor(10000 + Math.random() * 90000).toString();
}

function getCurrentYear() {
    return new Date().getFullYear().toString();
}

function generateTicketId() {
    const randomNum = generateRandomFiveDigits();
    const year = getCurrentYear();
    return `TIX-${randomNum}-${year}`;
}


window.generateTicketId = generateTicketId;

// Add this function to handle real-time updates for generate ticket section
function initializeGenerateTicketListeners() {
    const ticketsRef = collection(db, 'tickets');

    // Listen for any ticket changes
    onSnapshot(ticketsRef, (snapshot) => {
        snapshot.docChanges().forEach(async (change) => {
            if (change.type === "added" || change.type === "modified") {
                const ticketData = change.doc.data();
                // Get current selected event in generate ticket section
                const currentEventId = document.getElementById('ticketEventSelector')?.value;

                if (currentEventId === ticketData.eventId) {
                    console.log('Updating generate ticket section due to ticket change');
                    await updateTicketTypeOptions();
                    await updateTicketStats(currentEventId);
                }
            }
        });
    });

    // Also listen for event changes
    const eventsRef = collection(db, 'events');
    onSnapshot(eventsRef, (snapshot) => {
        snapshot.docChanges().forEach(async (change) => {
            if (change.type === "modified") {
                const eventData = change.doc.data();
                const currentEventId = document.getElementById('ticketEventSelector')?.value;

                if (currentEventId === change.doc.id) {
                    console.log('Updating generate ticket section due to event change');
                    await updateTicketTypeOptions();
                    await updateTicketStats(currentEventId);
                }
            }
        });
    });
}

// Update the DOMContentLoaded event listener
document.addEventListener('DOMContentLoaded', () => {
    // ... existing initialization code ...

    // Initialize generate ticket listeners if we're on that section
    if (document.getElementById('generateTicket')?.style.display !== 'none') {
```

```
        initializeGenerateTicketListeners();
    }
});

    // Add an event listener to sanitize pasted content
    document.getElementById('eventDescription').addEventListener('paste', function (event) {
        event.preventDefault(); // Prevent the default paste action

        // Get plain text from the clipboard
        const text = (event.clipboardData || window.clipboardData).getData('text');

        // Insert the plain text into the textarea
        const start = this.selectionStart;
        const end = this.selectionEnd;
        this.value = this.value.substring(0, start) + text + this.value.substring(end);

        // Place the cursor after the inserted text
        this.selectionStart = this.selectionEnd = start + text.length;
    });

    function openEditModal(eventId, eventData) {
        document.getElementById('editingEventId').value = eventId;
        try {
            console.log("Opening edit modal for event:", eventId);
            console.log("Event data received:", eventData);

            isEditing = true;
            currentEditingEventId = eventId;

            // Basic fields
            document.getElementById('eventName').value = eventData.name || '';

            // Handle description - clear existing content first
            const descriptionEditor = document.getElementById('eventDescription');
            descriptionEditor.innerHTML = eventData.description || '';

            // Handle rules - clear existing content first
            const rulesEditor = document.getElementById('eventRules');
            if (rulesEditor) {
                rulesEditor.innerHTML = eventData.rules || '';
            }

            // Rest of your existing code...
            document.getElementById('eventDate').value = eventData.date ? new Date(eventData.date).toISOString().slice(0, 16) : '';
            document.getElementById('eventPrice').value = eventData.price || 0;
            document.getElementById('eventType').value = eventData.eventType || '';

            // Handle event type specific fields
            handleEventTypeChange();

            if (eventData.eventType === 'online' || eventData.eventType === 'hybrid') {
                document.getElementById('meetingPlatform').value = eventData.onlineDetails?.platform || '';
document.getElementById('meetingLink').value = eventData.onlineDetails?.meetingLink || '';
                document.getElementById('meetingId').value = eventData.meetingId || '';
                document.getElementById('meetingPassword').value = eventData.meetingPassword || '';
            }

            if (eventData.eventType === 'venue' || eventData.eventType === 'hybrid') {
                document.getElementById('eventPlace').value = eventData.venueDetails?.place || '';
                document.getElementById('eventVenue').value = eventData.venueDetails?.venue || '';
            }

            document.querySelector('.event-form').scrollIntoView({ behavior: 'smooth' });

        } catch (error) {
            console.error("Detailed error opening edit modal:", error);
            alert('Error loading event data for editing');
        }
    }

document.addEventListener('DOMContentLoaded', () => {
    const eventDescription = document.getElementById('eventDescription');
    if (eventDescription) {
        eventDescription.addEventListener('paste', function (event) {
            event.preventDefault();
            const text = (event.clipboardData || window.clipboardData).getData('text');
            const start = this.selectionStart;
            const end = this.selectionEnd;
```

```
            this.value = this.value.substring(0, start) + text + this.value.substring(end);
            this.selectionStart = this.selectionEnd = start + text.length;
        });
    }
});
// Add this function to initialize rich text formatting toolbar
function initializeRichTextEditor(editorId) {
    const editor = document.getElementById(editorId);
    if (!editor) return;

    // Create toolbar
    const toolbar = document.createElement('div');
    toolbar.className = 'editor-toolbar';
    toolbar.innerHTML = `
        <button type="button" onclick="formatDoc('bold')" title="Bold">
            <i class="fas fa-bold"></i>
        </button>
        <button type="button" onclick="formatDoc('italic')" title="Italic">
            <i class="fas fa-italic"></i>
        </button>
        <button type="button" onclick="formatDoc('underline')" title="Underline">
            <i class="fas fa-underline"></i>
        </button>
        <button type="button" onclick="formatDoc('insertunorderedlist')" title="Bullet List">
            <i class="fas fa-list-ul"></i>
        </button>
    `;

    // Insert toolbar before editor
    editor.parentNode.insertBefore(toolbar, editor);

    // Add paste event handler
    editor.addEventListener('paste', function(e) {
        e.preventDefault();
        const text = e.clipboardData.getData('text/html') || e.clipboardData.getData('text/plain');
        document.execCommand('insertHTML', false, text);
    });
}

// Add this function to handle formatting
function formatDoc(command) {
    document.execCommand(command, false, null);
}
// Add this function to handle paste events in contenteditable fields
function handleContentEditablePaste(e) {
    e.preventDefault();

    // Get plain text from clipboard
    let text = '';
    if (e.clipboardData || window.clipboardData) {
        // Get HTML content if available
        text = (e.clipboardData || window.clipboardData).getData('text/html') ||
               (e.clipboardData || window.clipboardData).getData('text/plain');
    }

    // Insert at cursor position
    const selection = window.getSelection();
    if (selection.getRangeAt && selection.rangeCount) {
        const range = selection.getRangeAt(0);
        range.deleteContents();

        // Create wrapper div to handle HTML content
        const div = document.createElement('div');
        div.innerHTML = text;

        // Insert each child node
        const fragment = document.createDocumentFragment();
        let child;
        while (child = div.firstChild) {
            fragment.appendChild(child);
        }
        range.insertNode(fragment);

        // Move cursor to end
        range.collapse(false);
        selection.removeAllRanges();
        selection.addRange(range);
    }
```

```
}

// Add this to your DOMContentLoaded event listener
document.addEventListener('DOMContentLoaded', () => {
    const descriptionField = document.getElementById('eventDescription');
    if (descriptionField) {
        // Add paste event listener
        descriptionField.addEventListener('paste', handleContentEditablePaste);

        // Add keyboard shortcuts for formatting
        descriptionField.addEventListener('keydown', function(e) {
            if (e.ctrlKey || e.metaKey) {
                switch(e.key) {
                    case 'b':
                        e.preventDefault();
                        document.execCommand('bold', false, null);
                        break;
                    case 'i':
                        e.preventDefault();
                        document.execCommand('italic', false, null);
                        break;
                    case 'u':
                        e.preventDefault();
                        document.execCommand('underline', false, null);
                        break;
                }
            }
        });
    }
});
```

//admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Panel</title>
    <link rel="stylesheet" href="styles/admin.css">
    <script src="https://unpkg.com/html5-qrcode@2.3.8/html5-qrcode.min.js"></script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/all.min.css">
    <script src="https://cdn.rawgit.com/davidshimjs/qrcodejs/gh-pages/qrcode.min.js"></script>
    <script src="https://html2canvas.hertzen.com/dist/html2canvas.min.js"></script>
    <!-- Add Swiper CSS and JS -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.css" />
<script src="https://cdn.jsdelivr.net/npm/swiper@11/swiper-bundle.min.js"></script>
    <!-- Add this CSS style block in the head section -->
    <style>
        @media screen and (max-width: 768px) {
            .content-section {
                padding-bottom: 80px; /* Add padding at bottom of each section */
                margin-bottom: 20px; /* Additional margin for visual separation */
            }

            /* Ensure the last element in each section has proper spacing */
            .content-section > *:last-child {
                margin-bottom: 60px;
            }

            /* Additional spacing for specific sections that might need more room */
            #verification, #generateTicket, #passes {
                padding-bottom: 100px; /* Extra padding for sections with more interactive elements */
            }

            /* Ensure bottom elements don't get cut off by the mobile nav */
            .button-group, .admin-ticket-actions, .generate-btn {
                margin-bottom: 80px;
            }

            /* Additional spacing for create event form */
            #createEventSection {
                padding-bottom: 120px; /* Extra padding for the create event form */
            }

            /* Ensure form buttons are visible */
            .form-group:last-child {
                margin-bottom: 100px; /* Space for form buttons */
            }
```

```
/* Additional space for buttons when editing */
.create-btn, .cancel-btn {
    margin-bottom: 20px;
}

/* Container for form buttons */
.form-group[style*="display: flex"] {
    margin-bottom: 120px; /* Extra space for button container */
    padding-bottom: 20px;
}

/* Hide event gallery on mobile */
.event-images {
    display: none;
}

/* Optional: Add a message indicating gallery is hidden on mobile */
.event-images::before {
    content: 'Gallery available on desktop view';
    display: block;
    text-align: center;
    color: #666;
    padding: 10px;
    font-size: 0.9em;
    font-style: italic;
}

/* Hide event gallery on mobile - using more specific selector */
.event-card .event-images,
.event-card-content .event-images,
.event-section .event-images {
    display: none !important;
}

/* Optional: Add a message indicating gallery is hidden on mobile */
.event-card .event-images::before,
.event-card-content .event-images::before,
.event-section .event-images::before {
    content: 'Gallery available on desktop view';
    display: block;
    text-align: center;
    color: #666;
    padding: 10px;
    font-size: 0.9em;
    font-style: italic;
}

/* Hide event gallery on mobile - using all possible combinations */
.event-images,
div.event-images,
.event-card .event-images,
.event-card-content .event-images,
.event-section .event-images,
[class*="event"] .event-images {
    display: none !important;
    visibility: hidden !important;
    height: 0 !important;
    overflow: hidden !important;
}

/* Optional: Add a message indicating gallery is hidden on mobile */
.event-images::before,
div.event-images::before {
    content: 'Gallery available on desktop view';
    display: block;
    text-align: center;
    color: #666;
    padding: 10px;
    font-size: 0.9em;
    font-style: italic;
}

/* Hide event gallery on mobile */
.events-carousel-section {
    display: none !important;
}
```

```
            /* Optional: Add a message indicating gallery is hidden on mobile */
            .events-carousel-section::before {
                content: 'Gallery available on desktop view';
                display: block;
                text-align: center;
                color: #666;
                padding: 10px;
                font-size: 0.9em;
                font-style: italic;
            }
        }
    </style>
    <!-- Add this near the top with other script imports -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>
    <script>
        // Initialize jsPDF globally
        window.jsPDF = window.jspdf.jsPDF;
        console.log('jsPDF initialized:', !!window.jsPDF);
    </script>
</head>
<body>
    <!-- Mobile Top Navbar -->
    <div class="mobile-top-nav">
        <div class="mobile-top-title" onclick="window.location.href='index.html'" style="cursor: pointer;">EventTix Admin</div>
        <button onclick="logout()" class="mobile-logout-btn">
            <i class="fas fa-sign-out-alt"></i>
        </button>
    </div>
    <div class="container">
        <div class="sidebar">
            <div class="logo">
                <h2 onclick="window.location.href='index.html'" style="cursor: pointer;">EventTix Admin</h2>
            </div>

            <div class="menu">
                <div class="menu-item active" onclick="showSection('dashboard')">
                    <i class="fas fa-tachometer-alt"></i>
                    <span>Dashboard</span>
                </div>

                <div class="menu-item" onclick="showSection('events')">
                    <i class="fas fa-calendar-alt"></i>
                    <span>Events</span>
                </div>

                <div class="menu-item" onclick="showSection('tickets')">
                    <i class="fas fa-ticket-alt"></i>
                    <span>Tickets</span>
                </div>

                <div class="menu-item" onclick="showSection('verification')">
                    <i class="fas fa-check-circle"></i>
                    <span>Verify Tickets</span>
                </div>

                <div class="menu-item" onclick="showSection('passes')">
                    <i class="fas fa-id-card"></i>
                    <span>Pass Management</span>
                </div>

                <div class="menu-item" onclick="showSection('generateTicket')">
                    <i class="fas fa-plus-circle"></i>
                    <span>Generate Ticket</span>
                </div>


            </div>

            <div class="user-info">
                <span id="userEmail"></span>
                <div class="sidebar-actions">
                    <a href="index.html" class="home-btn" title="Go to Home">

                    </a>
                    <button id="logoutBtn" class="logout-btn" title="Logout">
                        <i class="fas fa-sign-out-alt"></i>
                    </button>
                </div>
```

```
    </div>
</div>

<div class="main-content">
    <!-- Dashboard Section -->
    <div id="dashboard" class="content-section active">
        <h2>Dashboard</h2>

        <div class="stats-grid">
            <div class="stat-card">
                <h3>Total Events</h3>
                <div class="number" id="totalEvents">0</div>
            </div>
            <div class="stat-card">
                <h3>Upcoming Events</h3>
                <div class="number" id="upcomingEvents">0</div>
            </div>
            <div class="stat-card">
                <h3>Completed Events</h3>
                <div class="number" id="completedEvents">0</div>
            </div>
            <div class="stat-card">
                <h3>Total Tickets</h3>
                <div class="number" id="totalTickets">0</div>
            </div>
            <div class="stat-card">
                <h3>Tickets Sold</h3>
                <div class="number" id="ticketsSold">0</div>
            </div>
            <div class="stat-card">
                <h3>Available Tickets</h3>
                <div class="number" id="availableTickets">0</div>
            </div>
            <div class="stat-card">
                <h3>Total Revenue</h3>
                <div class="number" id="totalRevenue">₹0</div>
            </div>
            <div class="stat-card">
                <h3>Tickets Scanned</h3>
                <div class="number" id="ticketsScanned">0</div>
            </div>
        </div>
    </div>

    <!-- Events Section -->
    <div id="events" class="content-section">
        <h2>Manage Events</h2>

        <!-- Add toggle buttons -->
        <div class="events-toggle">
            <button class="toggle-btn active" onclick="toggleEventsView('view')">
                <i class="fas fa-list"></i> View Events
            </button>
            <button class="toggle-btn" onclick="toggleEventsView('create')">
                <i class="fas fa-plus"></i> Create Event
            </button>
        </div>

        <!-- Create Event Form (Initially Hidden) -->
        <div id="createEventSection" class="event-section" style="display: none;">
            <div class="event-form">
                <input type="hidden" id="editingEventId">
                <div class="form-group">
                    <label>Event Name</label>
                    <input type="text" id="eventName" required>
                </div>
                <div class="form-group">
                    <label>Event Category</label>
                    <select id="eventCategory" onchange="handleEventCategoryChange()" required>
                        <option value="">Select Category</option>
                        <option value="corporate">Corporate Events</option>
                        <option value="social">Social Events</option>
                        <option value="entertainment">Entertainment</option>
                        <option value="sports">Sports & Fitness</option>
                        <option value="educational">Educational</option>
                        <option value="community">Community Events</option>
                        <option value="other">Other</option>
                    </select>
```

```
</div>
<div class="form-group" id="otherCategoryGroup" style="display: none;">
  <label>Custom Category Name</label>
  <input type="text" id="otherCategoryName" placeholder="Enter category name">
</div>
<div class="form-group">
  <label>Event Pricing Type</label>
  <select id="eventPricingType" onchange="handlePricingTypeChange()" required>
    <option value="">Select Pricing Type</option>
    <option value="paid">Paid Event</option>
    <option value="free">Free Event</option>
  </select>
</div>
<div class="form-group">
  <label for="eventDescription">Event Description</label>
  <div
    id="eventDescription"
    name="eventDescription"
    class="form-textarea"
    contenteditable="true"
    style="border: 1px solid #ccc; padding: 10px; min-height: 120px;">
  </div>
  <div class="helper-text">Enter a detailed description of your event</div>
</div>
<div class="form-group">
  <label>Event Date</label>
  <input type="datetime-local" id="eventDate" required>
</div>
<!-- Event Scheduling Section -->
<div class="scheduling-section">
  <h3>Event Scheduling</h3>
  <div class="schedule-checkbox">
    <input type="checkbox" id="enablePublishSchedule" onchange="toggleScheduleSection('publish')">
    <label for="enablePublishSchedule">Schedule Event Publishing</label>
  </div>
  <div id="publishScheduleSection" class="schedule-details" style="display: none;">
    <input type="datetime-local" id="publishDate" class="schedule-input">
    <div class="helper-text">Event will become visible to users at this time</div>
  </div>

  <div class="schedule-checkbox">
    <input type="checkbox" id="enableHideSchedule" onchange="toggleScheduleSection('hide')">
    <label for="enableHideSchedule">Schedule Event Hiding</label>
  </div>
  <div id="hideScheduleSection" class="schedule-details" style="display: none;">
    <input type="datetime-local" id="hideDate" class="schedule-input">
    <div class="helper-text">Event will be hidden from users at this time</div>
  </div>
</div>
<div class="form-group">
  <label>Booking Start Date</label>
  <input type="datetime-local" id="bookingStartDate" required>
</div>
<div class="form-group">
  <label>Booking Deadline</label>
  <input type="datetime-local" id="bookingDeadline" required>
</div>
<div class="form-group">
  <label>Price (₹)</label>
  <input type="number"
      id="eventPrice"
      min="0"
      step="0.01"
      value="0"
      required>
</div>
<div class="form-group">
  <label class="required">Event Type</label>
  <select id="eventType" onchange="handleEventTypeChange()" required>
    <option value="">Select Event Type</option>
    <option value="venue">Venue (Physical Location)</option>
    <option value="online">Online Event</option>
    <option value="hybrid">Hybrid (Both Online & Venue)</option>
  </select>
</div>
<div id="onlineEventDetails" class="form-section" style="display: none;">
  <h3>Online Event Details</h3>
  <div class="form-group">
```

```
                        <label class="required">Meeting Platform</label>
                        <select id="meetingPlatform" required>
                            <option value="">Select Platform</option>
                            <option value="zoom">Zoom</option>
                            <option value="gmeet">Google Meet</option>
                            <option value="teams">Microsoft Teams</option>
                            <option value="other">Other</option>
                        </select>
                    </div>
                    <div class="form-group">
                        <label class="required">Meeting Link</label>
                        <input type="url" id="meetingLink" placeholder="https://..." required>
                        <div class="helper-text">This will be shared with attendees after booking</div>
                    </div>
                    <div class="form-group">
                        <label>Meeting ID (if applicable)</label>
                        <input type="text" id="meetingId" placeholder="Meeting ID">
                    </div>
                    <div class="form-group">
                        <label>Meeting Password (if applicable)</label>
                        <input type="text" id="meetingPassword" placeholder="Meeting Password">
                    </div>
                </div>
                <div id="venueEventDetails" class="form-section" style="display: none;">
                    <h3>Venue Details</h3>
                    <div class="form-group">
                        <label class="required">Event Place</label>
                        <input type="text" id="eventPlace" placeholder="Enter event location" required>
                    </div>
                    <div class="form-group">
                        <label class="required">Venue Details</label>
                        <textarea id="eventVenue" placeholder="Enter detailed venue information (e.g., hall number, floor, landmarks)"
required></textarea>
                    </div>
                </div>
                <div id="imageUrlsContainer">
                    <div class="form-group image-input">
                        <label>Image URL #1</label>
                        <input type="text" class="eventImage" placeholder="Enter image URL">
                        <button type="button" onclick="addImageField()" class="add-image-btn">+ Add Another Image</button>
                    </div>
                </div>
                <div id="ticketingSection" class="form-section" style="display: none;">
                    <h3>Ticket Configuration</h3>

                    <!-- For non-hybrid events -->
                    <div class="form-group" id="standardTicketsSection">
                        <label class="required">Total Available Tickets</label>
                        <input type="number" id="eventTickets" min="1" required>
                    </div>

                    <!-- For hybrid events -->
                    <div id="hybridTicketsSection" style="display: none;">
                        <div class="form-group" id="venueTicketsSection">
                            <label class="required">Venue Tickets</label>
                            <input type="number" id="venueTickets" min="0" required>
                            <div class="helper-text">Number of tickets available for physical attendance</div>
                            <div class="form-group">
                                <label class="required">Venue Ticket Price (₹)</label>
                                <input type="number"
                                    id="venueTicketPrice"
                                    min="0"
                                    step="0.01"
                                    value="0"
                                    required>
                            </div>
                        </div>

                        <div class="form-group" id="onlineTicketsSection">
                            <label class="required">Online Tickets</label>
                            <input type="number" id="onlineTickets" min="0" step="0.01" required>
                            <div class="helper-text">Number of tickets available for online attendance</div>
                            <div class="form-group">
                                <label class="required">Online Ticket Price (₹)</label>
                                <input type="number"
                                    id="onlineTicketPrice"
                                    min="0"
                                    step="0.01"
```

```
                        value="0"
                        required>
                    </div>
                  </div>
                </div>
              </div>
              <!-- Add this after the event description field in your form -->
              <div class="form-group">
                <label>Rules (if applicable)</label>
                <div
                  id="eventRules"
                  name="eventRules"
                  class="form-textarea"
                  contenteditable="true"
                  style="border: 1px solid #ccc; padding: 10px; min-height: 120px;">
                </div>
                <div class="helper-text">Add any rules or guidelines for the event</div>
              </div>
              <div class="form-group" style="display: flex; gap: 10px;">
                <button type="button" onclick="handleCreateEvent()" class="create-btn">Create Event</button>
                <button type="button" onclick="cancelEdit()" class="cancel-btn" style="display: none;">Cancel Edit</button>
              </div>
            </div>
          </div>

          <!-- View Events Section (Initially Visible) -->
          <div id="viewEventSection" class="event-section">
            <!-- Add filter controls -->
            <div class="event-filters">
              <input type="text"
                  id="eventSearchInput"
                  placeholder="Search events..."
                  oninput="filterEvents()">
              <select id="eventTypeFilter" onchange="filterEvents()">
                <option value="">All Types</option>
                <option value="venue">Venue</option>
                <option value="online">Online</option>
                <option value="hybrid">Hybrid</option>
              </select>
              <select id="eventStatusFilter" onchange="filterEvents()">
                <option value="">All Status</option>
                <option value="upcoming">Upcoming</option>
                <option value="ongoing">Ongoing</option>
                <option value="completed">Completed</option>
              </select>
            </div>

            <div class="event-list" id="eventList">
              <!-- Events will be listed here -->
            </div>
          </div>
        </div>

        <!-- Tickets Section -->
        <div id="tickets" class="content-section">
          <h2>Ticket Management</h2>

          <!-- Stats card with original class names -->
          <div class="stats-card">
            <div class="stats-grid">
              <div class="stat-item">
                <h4>Total Tickets</h4>
                <p id="adminTotalTickets">0</p>
              </div>
              <div class="stat-item">
                <h4>Tickets Sold</h4>
                <p id="adminSoldTickets">0</p>
              </div>
              <div class="stat-item">
                <h4>Available Tickets</h4>
                <p id="adminAvailableTickets">0</p>
              </div>
              <div class="stat-item">
        <h4>Tickets Used</h4>
        <p id="selectedEventUsedTickets">0</p>
      </div>
              <div class="stat-item">
                <h4>Total Revenue</h4>
```

```
                    <p id="selectedEventTotalRevenue">₹0</p>
                </div>
            </div>
        </div>

        <!-- Existing ticket filters and list -->
        <div class="ticket-filters">
            <select id="adminEventSelector">
                <option value="">All Events</option>
            </select>
            <input type="text"
                id="adminTicketSearch"
                placeholder="Search tickets...">
        </div>

        <div id="adminTicketList" class="ticket-list">
            <!-- Tickets will be populated here -->
        </div>
    </div>

    <!-- Verification Section -->
<div id="verification" class="content-section">
    <h2>Ticket Verification</h2>

    <div class="verification-tabs">
        <button class="tab-btn active" onclick="switchVerificationMethod('manual')">
            <i class="fas fa-keyboard"></i> Manual Entry
        </button>
        <button class="tab-btn" onclick="switchVerificationMethod('qr')">
            <i class="fas fa-qrcode"></i> Scan QR Code
        </button>
    </div>

    <div id="qrVerification" class="verification-method">
        <div id="qrReader"></div>
        <button class="scan-btn" onclick="startQRScanner()">
            <i class="fas fa-camera"></i> Start Scanner
        </button>
        <button onclick="stopQRScanner()" style="display: none;">Stop Scanner</button>
    </div>

    <div id="manualVerification" class="verification-method" style="display: none;">
        <div class="form-group">
            <label>Enter Ticket ID</label>
            <input type="text" id="ticketIdInput" placeholder="Enter ticket ID">
            <button onclick="verifyTicketManually()">Verify Ticket</button>
        </div>
    </div>

    <div id="verificationResult" class="verification-result" style="display: none;">
        <h3>Verification Result</h3>
        <span class="result-status"></span>
        <div class="ticket-details"></div>
        <button onclick="markTicketAsUsed()" class="mark-used-btn" style="display: none;">
            Mark as Used
        </button>
    </div>
</div>
<!-- Add this new section after your other content sections -->
<div id="passes" class="content-section">
    <h2>Pass Management</h2>

    <!-- Stats Section -->
    <div class="stats-card">
        <div class="stats-grid">
            <div class="stat-item">
                <h4>Total Passes</h4>
                <p id="totalPasses">0</p>
            </div>
            <div class="stat-item">
                <h4>Used Passes</h4>
                <p id="usedPasses">0</p>
            </div>
            <div class="stat-item">
                <h4>Unused Passes</h4>
                <p id="unusedPasses">0</p>
            </div>
        </div>
```

```
    </div>

    <!-- Toggle Buttons -->
    <div class="passes-toggle">
        <button class="toggle-btn active" onclick="togglePassesView('generate')">
            Generate Passes
        </button>
        <button class="toggle-btn" onclick="togglePassesView('view')">
            View Passes
        </button>
    </div>

    <!-- Generate Passes Section -->
    <div id="generatePassesSection" class="passes-section">
        <div class="form-group">
            <label>Select Event</label>
            <select id="eventSelectorForPasses" required>
                <option value="">Select an Event</option>
            </select>
        </div>

        <div class="form-group">
            <label>Role</label>
            <select id="roleSelector" required>
                <option value="">Select Role</option>
                <option value="staff">Staff</option>
                <option value="vip">VIP</option>
                <option value="media">Media</option>
                <option value="other">Other</option>
            </select>
        </div>

        <div class="form-group" id="otherRoleInput" style="display: none;">
            <label>Custom Role</label>
            <input type="text" id="customRole" placeholder="Enter custom role">
        </div>

        <div class="form-group">
            <label>Number of Passes</label>
            <input type="number" id="numberOfPasses" min="1" value="1" required>
        </div>

        <button onclick="generatePasses()" class="generate-btn">Generate Passes</button>

        <div id="generatedPassesList" class="generated-passes">
            <!-- Generated passes will appear here -->
        </div>
    </div>

    <!-- View Passes Section -->
    <div id="viewPassesSection" class="passes-section" style="display: none;">
        <div id="viewPassesList">
            <!-- Existing passes will be listed here -->
        </div>
    </div>
</div>
<!-- Generate Ticket Section -->
<div id="generateTicket" class="content-section">
    <h2>Generate Ticket</h2>

    <div class="ticket-generation-form">
        <div class="form-group">
            <label>Select Event</label>
            <select id="ticketEventSelector">
                <option value="">Select an Event</option>
            </select>
        </div>

        <div class="form-group">
            <label>Ticket Type</label>
            <select id="ticketTypeSelector">
    <option value="">Select Ticket Type</option>
</select>
<small class="helper-text" style="color: #666; font-size: 0.8em; margin-top: 4px; display: block;">
    Ticket types will only appear if tickets are available for the selected event
</small>
        </div>
```

```
        <div class="form-group">
            <label>Quantity</label>
            <input type="number" id="ticketQuantity" min="1" value="1" required>
        </div>

        <div class="form-group">
            <label>User Email</label>
            <input type="email" id="ticketUserEmail" placeholder="Enter user email" required>
        </div>
        <div class="form-group">
            <label>Payment Mode</label>
            <select id="paymentMode" required>
                <option value="">Select Payment Mode</option>
                <option value="cash">Cash</option>
                <option value="card">Card</option>
                <option value="free" style="display: none;">Free</option>
            </select>
        </div>

        <div class="price-summary">
            <p>Price per ticket: <span id="pricePerTicket">₹0</span></p>
            <p>Total Amount: <span id="totalTicketPrice">₹0</span></p>
        </div>

        <button onclick="generateTicketForUser()" class="generate-btn">Generate Ticket</button>
    </div>
</div>



    <script>
    // Navigation
    function showSection(sectionId) {
        // Hide all sections
        document.querySelectorAll('.content-section').forEach(section => {
            section.style.display = 'none';
        });

        // Show selected section
        const selectedSection = document.getElementById(sectionId);
        if (selectedSection) {
            selectedSection.style.display = 'block';
        }

        // Update active menu item for both sidebar and mobile nav
        document.querySelectorAll('.menu-item, .mobile-nav-item').forEach(item => {
            item.classList.remove('active');
        });

        // Activate sidebar menu item
        const menuItem = document.querySelector(`.menu-item[onclick*="${sectionId}"]`);
        if (menuItem) {
            menuItem.classList.add('active');
        }

        // Activate mobile nav item
        const mobileNavItem = document.querySelector(`.mobile-nav-item[onclick*="${sectionId}"]`);
        if (mobileNavItem) {
            mobileNavItem.classList.add('active');
        }

        // Handle section-specific initialization
        switch (sectionId) {
            case 'dashboard':
                loadDashboard();
                break;
            case 'events':
                loadEvents();
                break;
            case 'tickets':
                loadTickets();
                break;
            case 'verification':
                switchVerificationMethod('manual');
                break;
        }
```

```
    // Clean up verification when switching away
    if (sectionId !== 'verification') {
        cleanupVerification();
    }
}

// Ensure the dashboard is active on page load
document.addEventListener('DOMContentLoaded', () => {
    showSection('dashboard');
});



    // Format currency in INR
    function formatCurrency(amount) {
        return new Intl.NumberFormat('en-IN', {
            style: 'currency',
            currency: 'INR',
            minimumFractionDigits: 0,
            maximumFractionDigits: 0
        }).format(amount);
    }

    // Events Functions
    let imageFieldCount = 1;

    function addImageField() {
        imageFieldCount++;
        const container = document.getElementById('imageUrlsContainer');
        const div = document.createElement('div');
        div.className = 'form-group image-input';
        div.innerHTML = `
            <label>Image URL #${imageFieldCount}</label>
            <input type="text" class="eventImage" placeholder="Enter image URL">
            <button type="button" onclick="removeImageField(this)" class="remove-image-btn">Remove</button>
        `;
        container.appendChild(div);
    }

    function removeImageField(button) {
        button.parentElement.remove();
        imageFieldCount--;
    }

    // Add this variable to track editing state
    let isEditing = false;
    let currentEditingEventId = null;

    // Update the edit button click handler
    function openEditModal(eventId, eventData) {
document.getElementById('editingEventId').value = eventId;
try {
    console.log("Opening edit modal for event:", eventId);
    console.log("Event data received:", eventData);

    isEditing = true;
    currentEditingEventId = eventId;

    // Basic fields
    document.getElementById('eventName').value = eventData.name || '';
    document.getElementById('eventCategory').value = eventData.category || '';
    document.getElementById('eventPricingType').value = eventData.pricingType || 'paid';

    // Clear and set description content
    const descriptionEditor = document.getElementById('eventDescription');
    descriptionEditor.innerHTML = ''; // Clear first
    descriptionEditor.innerHTML = eventData.description || '';

    // Clear and set rules content
    const rulesEditor = document.getElementById('eventRules');
    if (rulesEditor) {
        rulesEditor.innerHTML = ''; // Clear first
        rulesEditor.innerHTML = eventData.rules || '';
    }

    // Dates
    document.getElementById('eventDate').value = eventData.date ? new Date(eventData.date).toISOString().slice(0, 16) : '';
```

```
    document.getElementById('bookingStartDate').value = eventData.bookingStartDate ? new
Date(eventData.bookingStartDate).toISOString().slice(0, 16) : '';
    document.getElementById('bookingDeadline').value = eventData.bookingDeadline ? new
Date(eventData.bookingDeadline).toISOString().slice(0, 16) : '';

    // Event type and pricing
    const eventTypeSelect = document.getElementById('eventType');
    eventTypeSelect.value = eventData.eventType || '';
    handleEventTypeChange(); // This will show/hide relevant sections

    // Handle pricing based on event type
    if (eventData.eventType === 'hybrid') {
        // Set hybrid event prices and tickets
        document.getElementById('venueTickets').value = eventData.tickets?.venue?.total || '';
        document.getElementById('onlineTickets').value = eventData.tickets?.online?.total || '';
        document.getElementById('venueTicketPrice').value = eventData.tickets?.venue?.price || '0';
        document.getElementById('onlineTicketPrice').value = eventData.tickets?.online?.price || '0';
    } else {
        // Set standard event price and tickets
        document.getElementById('eventPrice').value = eventData.price || '0';
        document.getElementById('eventTickets').value = eventData.totalTickets || '';
    }

    // Handle online event details
    if (eventData.eventType === 'online' || eventData.eventType === 'hybrid') {
        document.getElementById('meetingPlatform').value = eventData.onlineDetails?.platform || '';
        document.getElementById('meetingLink').value = eventData.onlineDetails?.meetingLink || '';
        document.getElementById('meetingId').value = eventData.onlineDetails?.meetingId || '';
        document.getElementById('meetingPassword').value = eventData.onlineDetails?.meetingPassword || '';
    }

    // Handle venue details
    if (eventData.eventType === 'venue' || eventData.eventType === 'hybrid') {
        document.getElementById('eventPlace').value = eventData.venueDetails?.place || '';
        document.getElementById('eventVenue').value = eventData.venueDetails?.venue || '';
    }

    // Handle scheduling data
    if (eventData.scheduling) {
        const publishSchedule = document.getElementById('enablePublishSchedule');
        const hideSchedule = document.getElementById('enableHideSchedule');

        if (eventData.scheduling.publishDate) {
            publishSchedule.checked = true;
            document.getElementById('publishScheduleSection').style.display = 'block';
            document.getElementById('publishDate').value = new Date(eventData.scheduling.publishDate).toISOString().slice(0, 16);
        }

        if (eventData.scheduling.hideDate) {
            hideSchedule.checked = true;
            document.getElementById('hideScheduleSection').style.display = 'block';
            document.getElementById('hideDate').value = new Date(eventData.scheduling.hideDate).toISOString().slice(0, 16);
        }
    }

    // Handle images
    const imageContainer = document.getElementById('imageUrlsContainer');
    imageContainer.innerHTML = '';

    if (eventData.images && Array.isArray(eventData.images)) {
        eventData.images.forEach((imageUrl, index) => {
            const div = document.createElement('div');
            div.className = 'form-group image-input';
            div.innerHTML = `
                <label>Image URL #${index + 1}</label>
                <input type="text" class="eventImage" value="${imageUrl}" placeholder="Enter image URL">
                ${index === 0 ?
                    `<button type="button" onclick="addImageField()" class="add-image-btn">+ Add Another Image</button>` :
                    `<button type="button" onclick="removeImageField(this)" class="remove-image-btn">Remove</button>`
                }
            `;
            imageContainer.appendChild(div);
        });
        imageFieldCount = eventData.images.length;
    }

    // Update form buttons
    const cancelBtn = document.querySelector('.cancel-btn');
```

```
        const createBtn = document.querySelector('.create-btn');
        if (cancelBtn) cancelBtn.style.display = 'block';
        if (createBtn) createBtn.textContent = 'Update Event';

        // Scroll form into view
        document.querySelector('.event-form').scrollIntoView({ behavior: 'smooth' });

    } catch (error) {
        console.error("Error in openEditModal:", error);
        alert('Error loading event data for editing: ' + error.message);
    }
}

    // Update the handleCreateEvent function to handle both create and update
    async function handleCreateEvent() {
    try {
        if (!validateEventDates()) {
            return;
        }

        const editingEventId = document.getElementById('editingEventId')?.value;
        const isEditing = !!editingEventId;

        // Get description content with HTML formatting
        const descriptionContent = document.getElementById('eventDescription').innerHTML;
        const rulesContent = document.getElementById('eventRules')?.innerHTML || '';

        const eventData = {
            name: document.getElementById('eventName').value.trim(),
            description: descriptionContent, // Store HTML content
            rules: rulesContent, // Store HTML content
            category: document.getElementById('eventCategory').value,
            pricingType: document.getElementById('eventPricingType').value,
            date: new Date(document.getElementById('eventDate').value).toISOString(),
            bookingStartDate: new Date(document.getElementById('bookingStartDate').value).toISOString(),
            bookingDeadline: new Date(document.getElementById('bookingDeadline').value).toISOString(),
            eventType: document.getElementById('eventType').value,
            price: parseFloat(document.getElementById('eventPrice').value) || 0,
            updatedAt: serverTimestamp(),

            // Add scheduling if enabled
            scheduling: {
                publishDate: document.getElementById('enablePublishSchedule').checked ?
                    new Date(document.getElementById('publishDate').value).toISOString() : null,
                hideDate: document.getElementById('enableHideSchedule').checked ?
                    new Date(document.getElementById('hideDate').value).toISOString() : null
            },

            // Add images
            images: Array.from(document.getElementsByClassName('eventImage'))
                .map(input => input.value.trim())
                .filter(url => url !== '')
        };

        // Only add these fields for new events
        if (!isEditing) {
            eventData.eventId = generateEventId();
            eventData.createdAt = serverTimestamp();
        }

        // Handle custom category
        if (eventData.category === 'other') {
            eventData.customCategory = document.getElementById('otherCategoryName').value.trim();
        }

        // Handle event type specific details
        if (eventData.eventType === 'online' || eventData.eventType === 'hybrid') {
            eventData.onlineDetails = {
                platform: document.getElementById('meetingPlatform').value,
                meetingLink: document.getElementById('meetingLink').value.trim(),
                meetingId: document.getElementById('meetingId').value.trim(),
                meetingPassword: document.getElementById('meetingPassword').value.trim()
            };
        }

        if (eventData.eventType === 'venue' || eventData.eventType === 'hybrid') {
            eventData.venueDetails = {
                place: document.getElementById('eventPlace').value.trim(),
```

```
            venue: document.getElementById('eventVenue').value.trim()
        };
    }

    // Handle ticket configuration
    if (eventData.eventType === 'hybrid') {
        eventData.tickets = {
            venue: {
                total: parseInt(document.getElementById('venueTickets').value) || 0,
                available: parseInt(document.getElementById('venueTickets').value) || 0,
                price: parseFloat(document.getElementById('venueTicketPrice').value) || 0
            },
            online: {
                total: parseInt(document.getElementById('onlineTickets').value) || 0,
                available: parseInt(document.getElementById('onlineTickets').value) || 0,
                price: parseFloat(document.getElementById('onlineTicketPrice').value) || 0
            }
        };
    } else {
        eventData.totalTickets = parseInt(document.getElementById('eventTickets').value) || 0;
        eventData.availableTickets = parseInt(document.getElementById('eventTickets').value) || 0;
    }

    // Handle free events
    if (eventData.pricingType === 'free') {
        eventData.price = 0;
        if (eventData.eventType === 'hybrid') {
            eventData.tickets.venue.price = 0;
            eventData.tickets.online.price = 0;
        }
    }

    // Update or create event in Firebase
    if (isEditing) {
        await updateDoc(doc(db, 'events', editingEventId), eventData);
        alert('Event updated successfully!');
    } else {
        await addDoc(collection(db, 'events'), eventData);
        alert('Event created successfully!');
    }

    // Reset form and view
    clearEventForm();
    toggleEventsView('view');
    loadEvents();

  } catch (error) {
    console.error('Error handling event:', error);
    alert(`Error ${isEditing ? 'updating' : 'creating'} event: ${error.message}`);
  }
}

    // Update clearEventForm to reset editing state
    function clearEventForm() {
  try {
    // Clear basic fields
    document.getElementById('eventName').value = '';
    document.getElementById('eventCategory').value = '';
    document.getElementById('eventPricingType').value = '';

    // Clear rich text editors
    const descriptionEditor = document.getElementById('eventDescription');
    const rulesEditor = document.getElementById('eventRules');
    if (descriptionEditor) descriptionEditor.innerHTML = '';
    if (rulesEditor) rulesEditor.innerHTML = '';

    // Clear dates
    document.getElementById('eventDate').value = '';
    document.getElementById('bookingStartDate').value = '';
    document.getElementById('bookingDeadline').value = '';

    // Clear prices and tickets
    document.getElementById('eventPrice').value = '';
    document.getElementById('eventTickets').value = '';
    document.getElementById('venueTickets').value = '';
    document.getElementById('onlineTickets').value = '';
    document.getElementById('venueTicketPrice').value = '';
    document.getElementById('onlineTicketPrice').value = '';
```

```
        // Clear event type and related fields
        const eventTypeSelect = document.getElementById('eventType');
        eventTypeSelect.value = '';
        handleEventTypeChange();

        // Clear scheduling
        document.getElementById('enablePublishSchedule').checked = false;
        document.getElementById('enableHideSchedule').checked = false;
        document.getElementById('publishScheduleSection').style.display = 'none';
        document.getElementById('hideScheduleSection').style.display = 'none';

        // Reset image container
        const imageContainer = document.getElementById('imageUrlsContainer');
        imageContainer.innerHTML = `
            <div class="form-group image-input">
                <label>Image URL #1</label>
                <input type="text" class="eventImage" placeholder="Enter image URL">
                <button type="button" onclick="addImageField()" class="add-image-btn">+ Add Another Image</button>
            </div>
        `;
        imageFieldCount = 1;

        // Reset buttons
        const cancelBtn = document.querySelector('.cancel-btn');
        const createBtn = document.querySelector('.create-btn');
        if (cancelBtn) cancelBtn.style.display = 'none';
        if (createBtn) createBtn.textContent = 'Create Event';

        // Reset editing state
        isEditing = false;
        currentEditingEventId = null;

    } catch (error) {
        console.error("Error in clearEventForm:", error);
    }
}

        async function loadEvents() {
            try {
                console.log("Starting to load events...");
                const eventList = document.getElementById('eventList');
                if (!eventList) {
                    console.error("Event list element not found");
                    return;
                }

                eventList.innerHTML = '<p>Loading events...</p>';
                const eventsRef = collection(db, 'events');

                console.log("Fetching events...");
                const snapshot = await getDocs(eventsRef);

                console.log("Events fetched:", snapshot.size);
                eventList.innerHTML = '';

                if (snapshot.empty) {
                    console.log("No events found");
                    eventList.innerHTML = '<p>No events found. Create your first event!</p>';
                    return;
                }

                snapshot.forEach((doc) => {
                    console.log("Processing event:", doc.id);
                    const event = doc.data();
                    const eventCard = createEventCard(event, doc.id);
                    eventList.appendChild(eventCard);
                });

            } catch (error) {
                console.error("Detailed error loading events:", error);
                const eventList = document.getElementById('eventList');
                if (eventList) {
                    eventList.innerHTML = `
                        <div class="error-message">
                            <p>Error loading events. Please try again.</p>
                            <button onclick="loadEvents()" class="retry-btn">Retry</button>
                        </div>
```

```
            `;
        }
    }
}

// Helper function to create event cards
function createEventCard(event, eventId) {
    try {
        const div = document.createElement('div');
        div.className = 'event-card';
        div.setAttribute('data-event-id', eventId);

        // Determine event visibility status
        const now = new Date();
        let visibilityStatus = '';

        if (event.scheduling) {
            if (event.scheduling.hideDate && new Date(event.scheduling.hideDate) < now) {
                visibilityStatus = `
                    <span class="visibility-status archived">
                        <i class="fas fa-archive"></i>
                        Archived
                    </span>
                `;
            } else if (event.scheduling.publishDate && new Date(event.scheduling.publishDate) > now) {
                visibilityStatus = `
                    <span class="visibility-status scheduled">
                        <i class="fas fa-clock"></i>
                        Scheduled
                    </span>
                `;
            } else {
                visibilityStatus = `
                    <span class="visibility-status published">
                        <i class="fas fa-globe"></i>
                        Published
                    </span>
                `;
            }
        } else {
            visibilityStatus = `
                <span class="visibility-status published">
                    <i class="fas fa-globe"></i>
                    Published
                </span>
            `;
        }

        const eventTypeBadge = `
            <span class="event-type-badge ${event.eventType}">
                ${event.eventType ? event.eventType.charAt(0).toUpperCase() + event.eventType.slice(1) : 'Unknown'}
            </span>
        `;

        const ticketInfoHTML = event.eventType === 'hybrid' ? `
            <div class="ticket-type-info">
                <h4>Ticket Information</h4>
                <div class="venue-tickets">
                    <strong>Venue Tickets:</strong>
                    <p>Available: ${event.tickets?.venue?.available || 0}/${event.tickets?.venue?.total || 0}</p>
                    <p>Price: ₹${event.tickets?.venue?.price?.toFixed(2) || '0.00'}</p>
                </div>
                <div class="online-tickets">
                    <strong>Online Tickets:</strong>
                    <p>Available: ${event.tickets?.online?.available || 0}/${event.tickets?.online?.total || 0}</p>
                    <p>Price: ₹${event.tickets?.online?.price?.toFixed(2) || '0.00'}</p>
                </div>
            </div>
        ` : `
            <div class="ticket-type-info">
                <h4>Ticket Information</h4>
                <p>Available Tickets: ${event.availableTickets || 0}/${event.totalTickets || 0}</p>
                <p>Price: ₹${event.price?.toFixed(2) || '0.00'}</p>
            </div>
        `;

        const onlineDetailsHTML = (event.eventType === 'online' || event.eventType === 'hybrid') && event.onlineDetails ? `
            <div class="event-details-section">
```

```
        <h4>Online Event Details</h4>
        <p><strong>Platform:</strong> ${event.onlineDetails.platform || 'Not specified'}</p>
        <p class="secured-info">Meeting link and access details will be shared after booking</p>
    </div>
` : '';

const venueDetailsHTML = (event.eventType === 'venue' || event.eventType === 'hybrid') && event.venueDetails ? `
    <div class="event-details-section">
        <h4>Venue Details</h4>
        <p><strong>Location:</strong> ${event.venueDetails.place || 'Not specified'}</p>
        <p><strong>Venue:</strong> ${event.venueDetails.venue || 'Not specified'}</p>
    </div>
` : '';

const bookingStatus = getBookingStatus(event);
const bookingStatusHTML = `
    <div class="booking-status ${bookingStatus.status}">
        ${bookingStatus.message}
    </div>
`;

const imagesHTML = event.images && Array.isArray(event.images) ? `
    <div class="event-images">
        ${event.images.map(img => `<img src="${img}" alt="Event image" onerror="this.src='placeholder.jpg'">`).join('')}
    </div>
` : '';

const descriptionHTML = `
    <div class="event-description-section">
        <h4>About This Event</h4>
        <div class="formatted-description">
            ${event.description || 'No description available'}
        </div>
    </div>
`;

const rulesHTML = event.rules ? `
    <div class="event-details-section">
        <h4>Event Rules</h4>
        <div class="formatted-content">
            ${event.rules}
        </div>
    </div>
` : '';

const limitationsHTML = event.limitations ? `
    <div class="event-details-section">
        <h4>Event Limitations</h4>
        <div class="formatted-content">
            ${event.limitations}
        </div>
    </div>
` : '';

div.innerHTML = `
    <div class="event-card-content">
        <div class="event-header">
            <h3>
                ${event.name || 'Unnamed Event'}
                ${eventTypeBadge}
                ${visibilityStatus}
            </h3>
            <div class="event-id">ID: ${event.eventId || eventId}</div>
        </div>

        ${imagesHTML}

        <div class="event-info">
            <div class="event-meta">
                <p><strong>Event Date:</strong> ${new Date(event.date).toLocaleString()}</p>
                <p><strong>Booking Period:</strong>
                    ${new Date(event.bookingStartDate).toLocaleString()} -
                    ${new Date(event.bookingDeadline).toLocaleString()}
                </p>
            </div>

            ${descriptionHTML}
            ${rulesHTML}
```

```
                    ${limitationsHTML}
                    ${ticketInfoHTML}
                    ${onlineDetailsHTML}
                    ${venueDetailsHTML}
                </div>

                ${bookingStatusHTML}

                <div class="button-group">
                    <button onclick="handleEditClick('${eventId}', ${JSON.stringify(event).replace(/"/g, '&quot;')})" class="edit-btn">Edit</button>
                    <button onclick="deleteEvent('${eventId}')" class="delete-btn">Delete</button>
                </div>
            </div>
        `;

        return div;

    } catch (error) {
        console.error("Error creating event card:", error, event);
        const errorCard = document.createElement('div');
        errorCard.className = 'event-card error';
        errorCard.innerHTML = `
            <div class="error-message">
                <p>Error displaying this event</p>
                <p class="error-details">${error.message}</p>
            </div>
        `;
        return errorCard;
    }
}

// Updated getBookingStatus function
function getBookingStatus(event) {
    try {
        const now = new Date();
        const bookingStartDate = new Date(event.bookingStartDate);
        const bookingDeadline = new Date(event.bookingDeadline);

        if (now < bookingStartDate) {
            return {
                status: 'not-started',
                message: 'Booking Not Started Yet',
                canBook: false
            };
        }

        if (now > bookingDeadline) {
            return {
                status: 'closed',
                message: 'Booking Closed',
                canBook: false
            };
        }

        if (event.eventType === 'hybrid') {
            const venueAvailable = event.tickets?.venue?.available > 0;
            const onlineAvailable = event.tickets?.online?.available > 0;

            if (!venueAvailable && !onlineAvailable) {
                return {
                    status: 'sold-out',
                    message: 'All Tickets Sold Out',
                    canBook: false
                };
            }

            return {
                status: 'active',
                message: `Book Now (${venueAvailable ? 'Venue' : ''}${venueAvailable && onlineAvailable ? ' & ' : ''}${onlineAvailable ? 'Online' : ''} Available)`,
                canBook: true,
                venueAvailable,
                onlineAvailable
            };
        } else {
            if (event.availableTickets <= 0) {
                return {
                    status: 'sold-out',
```

```
                message: 'Sold Out',
                canBook: false
            };
        }

        return {
            status: 'active',
            message: 'Book Now',
            canBook: true
        };
    }
  } catch (error) {
    console.error("Error getting booking status:", error);
    return {
        status: 'error',
        message: 'Status Unavailable',
        canBook: false
    };
  }
}

// Add this new function to handle edit button clicks
function handleEditClick(eventId, eventData) {
  try {
    console.log("Edit clicked for event:", eventId);
    console.log("Event data:", eventData);

    // Set the editing flags
    isEditing = true;
    currentEditingEventId = eventId; // Make sure this is the document ID from Firestore

    toggleEventsView('create');
    openEditModal(eventId, eventData);

    // Update button text
    const createBtn = document.querySelector('.create-btn');
    if (createBtn) {
        createBtn.textContent = 'Update Event';
    }

    // Show cancel button
    const cancelBtn = document.querySelector('.cancel-btn');
    if (cancelBtn) {
        cancelBtn.style.display = 'block';
    }

    // Update toggle buttons
    document.querySelectorAll('.toggle-btn').forEach(btn => {
        btn.classList.remove('active');
    });
    document.querySelector('.toggle-btn[onclick*="create"]').classList.add('active');

  } catch (error) {
    console.error("Error handling edit click:", error);
    alert("Error opening edit form: " + error.message);
  }
}

// Make the function available globally
window.handleEditClick = handleEditClick;

// Add this helper function for event status
function getEventStatus(event) {
  const now = new Date();
  const startDate = new Date(event.bookingStartDate);
  const deadlineDate = new Date(event.bookingDeadline);
  const eventDate = new Date(event.date);

  if (now < startDate) {
    return '<span class="status-upcoming">Booking Not Started</span>';
  } else if (now > deadlineDate || now > eventDate) {
    return '<span class="status-closed">Booking Closed</span>';
  } else if (event.availableTickets <= 0) {
    return '<span class="status-sold-out">Sold Out</span>';
  } else {
    return '<span class="status-active">Booking Open</span>';
  }
}
```

```
    // Make sure to initialize everything when the page loads
    document.addEventListener('DOMContentLoaded', () => {
        console.log("DOM Content Loaded");
        const auth = getAuth();

        auth.onAuthStateChanged((user) => {
  if (!user) {
    // User is signed out, redirect to index
    localStorage.clear();
    sessionStorage.clear();
    const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
    window.location.replace(`${baseUrl}index.html`);
  } else if (!user.email.toLowerCase().includes('admin')) {
    // User is not an admin, redirect to regular dashboard
    localStorage.clear();
    sessionStorage.clear();
    const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
    window.location.replace(`${baseUrl}/dashboard.html`);
  }
});
    });

    // Update the deleteEvent function
    async function deleteEvent(eventId) {
        try {
            if (!confirm('Are you sure you want to delete this event? This will also delete all associated tickets.')) {
                return;
            }

            // Show loading state
            const deleteBtn = document.querySelector(`button[onclick="deleteEvent('${eventId}')"]`);
            const originalText = deleteBtn.innerHTML;
            deleteBtn.innerHTML = '<i class="fas fa-spinner fa-spin"></i> Deleting...';
            deleteBtn.disabled = true;

            // First, get all tickets associated with this event
            const ticketsRef = collection(db, 'tickets');
            const ticketQuery = query(ticketsRef, where('eventId', '==', eventId));
            const ticketSnapshot = await getDocs(ticketQuery);

            // Delete all associated tickets
            const ticketDeletions = ticketSnapshot.docs.map(doc =>
                deleteDoc(doc.ref)
            );

            // Wait for all ticket deletions to complete
            await Promise.all(ticketDeletions);

            // Then delete the event
            await deleteDoc(doc(db, 'events', eventId));

            // Remove the event card from UI
            const eventCard = document.querySelector(`[data-event-id="${eventId}"]`);
            if (eventCard) {
                eventCard.remove();
            }

            alert('Event and associated tickets have been deleted successfully');

        } catch (error) {
            console.error('Error deleting event:', error);
            alert('Error deleting event: ' + error.message);

            // Restore delete button
            if (deleteBtn) {
                deleteBtn.innerHTML = originalText;
                deleteBtn.disabled = false;
            }
        }
    }

    // Tickets function
    async function loadTickets() {
  try {
    // Get event selector element
    const eventSelector = document.getElementById('adminEventSelector');
```

```
        // Get all active events
        const eventsRef = collection(db, 'events');
        const eventsSnapshot = await getDocs(eventsRef);
        const activeEvents = {};

        // Clear existing options
        eventSelector.innerHTML = '<option value="">All Events</option>';

        // Populate events dropdown and activeEvents object
        eventsSnapshot.forEach(doc => {
            const event = doc.data();
            if (!event.deleted && event.status !== 'deleted') {
                activeEvents[doc.id] = event;
                const option = document.createElement('option');
                option.value = doc.id;
                option.textContent = event.name;
                eventSelector.appendChild(option);
            }
        });

        // Initial load of all tickets with empty filters
        await filterTickets('', activeEvents, '');

        // Add event listeners for filtering
        document.getElementById('adminTicketSearch').addEventListener('input', async (e) => {
            await filterTickets(e.target.value, activeEvents, eventSelector.value);
        });

        eventSelector.addEventListener('change', async (e) => {
            await filterTickets(
                document.getElementById('adminTicketSearch').value,
                activeEvents,
                e.target.value
            );
        });

    } catch (error) {
        console.error("Error loading tickets:", error);
        alert("Error loading tickets: " + error.message);
    }
}

        // Add function to update stats display
        function updateTicketStats(totalTickets, totalSold, totalRevenue, totalUsed, availableTickets) {
    document.getElementById('adminTotalTickets').textContent = totalTickets;
    document.getElementById('adminSoldTickets').textContent = totalSold;
    document.getElementById('adminAvailableTickets').textContent = availableTickets;
    document.getElementById('adminUsedTickets').textContent = totalUsed;
    document.getElementById('adminTotalRevenue').textContent = formatCurrency(totalRevenue);
}

        // Add function to filter tickets
        async function filterTickets(searchTerm = '', activeEvents = {}, eventId = '') {
    try {
        const ticketsRef = collection(db, 'tickets');
        const ticketsSnapshot = await getDocs(ticketsRef);
        let filteredTickets = [];
        let totalTickets = 0;      // Will store total tickets from events
        let totalSold = 0;         // Will store total tickets sold
        let totalRevenue = 0;
        let totalUsed = 0;

        // First, calculate total tickets from events
        if (eventId) {
            // If specific event is selected
            const event = activeEvents[eventId];
            if (event) {
                if (event.eventType === 'hybrid') {
                    totalTickets = (event.tickets?.venue?.total || 0) + (event.tickets?.online?.total || 0);
                } else {
                    totalTickets = event.totalTickets || 0;
                }
            }
        } else {
            // If no specific event is selected, sum up all events
            Object.values(activeEvents).forEach(event => {
                if (event.eventType === 'hybrid') {
                    totalTickets += (event.tickets?.venue?.total || 0) + (event.tickets?.online?.total || 0);
```

```
            } else {
                totalTickets += event.totalTickets || 0;
            }
        });
    }

    // Process tickets for sold and used counts
    ticketsSnapshot.forEach(doc => {
        const ticket = doc.data();
        const event = activeEvents[ticket.eventId];

        if (event) {
            const searchString = `
                ${event.name || ''}
                ${ticket.eventId || ''}
                ${ticket.userEmail || ''}
                ${ticket.ticketIdentifier || ''}
            `.toLowerCase();

            const matchesSearch = !searchTerm || searchString.includes(searchTerm.toLowerCase());
            const matchesEvent = !eventId || ticket.eventId === eventId;

            if (matchesSearch && matchesEvent) {
                filteredTickets.push({
                    id: doc.id,
                    ...ticket,
                    eventName: event.name
                });

                if (ticket.paymentStatus === 'completed') {
                    const ticketCount = Number(ticket.ticketCount) || 1;
                    totalSold += ticketCount;
                    totalRevenue += Number(ticket.totalPrice) || 0;

                    if (ticket.used) {
                        totalUsed += ticketCount;
                    }
                }
            }
        }
    });

    // Calculate available tickets
    const availableTickets = totalTickets - totalSold;

    // Update stats display
    const statsElements = {
        totalTickets: document.getElementById('adminTotalTickets'),
        soldTickets: document.getElementById('adminSoldTickets'),
        availableTickets: document.getElementById('adminAvailableTickets'),
        usedTickets: document.getElementById('selectedEventUsedTickets'),
        revenue: document.getElementById('selectedEventTotalRevenue')
    };

    // Only update if elements exist
    if (statsElements.totalTickets) statsElements.totalTickets.textContent = totalTickets;
    if (statsElements.soldTickets) statsElements.soldTickets.textContent = totalSold;
    if (statsElements.availableTickets) statsElements.availableTickets.textContent = availableTickets;
    if (statsElements.usedTickets) statsElements.usedTickets.textContent = totalUsed;
    if (statsElements.revenue) statsElements.revenue.textContent = formatCurrency(totalRevenue);


    // Display filtered tickets
    const ticketList = document.getElementById('adminTicketList');
    if (ticketList) {
        if (filteredTickets.length === 0) {
            ticketList.innerHTML = '<p class="no-tickets">No tickets found</p>';
        } else {
            // Inside filterTickets function, update the ticket list HTML generation:
ticketList.innerHTML = filteredTickets.map(ticket => `
    <div class="admin-ticket-card">
        <div class="admin-ticket-info">
            <h3>${ticket.eventName}</h3>
            <p>Ticket ID: ${ticket.ticketIdentifier}</p>
            <p>User: ${ticket.userEmail}</p>
            <p>Quantity: ${ticket.ticketCount || 1}</p>
            <p>Status: ${ticket.used ? 'Used' : 'Not Used'}</p>
            <p>Payment: ${ticket.paymentStatus}</p>
```

```
            <p>Price: ${formatCurrency(ticket.totalPrice || 0)}</p>
        </div>
        <div class="admin-ticket-actions">
            <button onclick="showTicketDetails('${ticket.id}')" class="admin-view-btn">
                <i class="fas fa-eye"></i> View Details
            </button>
        </div>
    </div>
`).join('');
        }
    }

    } catch (error) {
        console.error("Error filtering tickets:", error);
    }
}
document.addEventListener('DOMContentLoaded', () => {
    const eventSelector = document.getElementById('adminEventSelector');
    const searchInput = document.getElementById('adminTicketSearch');

    if (eventSelector) {
        eventSelector.addEventListener('change', () => {
            const searchTerm = searchInput?.value || '';
            filterTickets(searchTerm, window.activeEvents || {}, eventSelector.value);
        });
    }

    if (searchInput) {
        searchInput.addEventListener('input', () => {
            const eventId = eventSelector?.value || '';
            filterTickets(searchInput.value, window.activeEvents || {}, eventId);
        });
    }
});

        // Add function to display tickets
        function displayTickets(tickets) {
    const ticketList = document.getElementById('adminTicketList');

    if (!ticketList) {
        console.error('Ticket list container not found');
        return;
    }

    if (tickets.length === 0) {
        ticketList.innerHTML = '<p class="no-tickets">No tickets found</p>';
        return;
    }

    let html = '';
    tickets.forEach(ticket => {
        const status = getTicketStatus(ticket);
        html += `
            <div class="admin-ticket-card">
                <div class="admin-ticket-info">
                    <div class="admin-ticket-detail">
                        <span class="admin-ticket-label">Ticket ID</span>
                        <span class="admin-ticket-value">${ticket.ticketIdentifier}</span>
                    </div>
                    <div class="admin-ticket-detail">
                        <span class="admin-ticket-label">Event</span>
                        <span class="admin-ticket-value">${ticket.eventName}</span>
                    </div>
                    <div class="admin-ticket-detail">
                        <span class="admin-ticket-label">User</span>
                        <span class="admin-ticket-value">${ticket.userEmail || 'N/A'}</span>
                    </div>
                    <div class="admin-ticket-detail">
                        <span class="admin-ticket-label">Purchase Date</span>
                        <span class="admin-ticket-value">${new Date(ticket.purchaseDate).toLocaleString()}</span>
                    </div>
                    <div class="admin-ticket-detail">
                        <span class="admin-ticket-label">Quantity</span>
                        <span class="admin-ticket-value">
                            <span class="admin-ticket-quantity">
                                <i class="fas fa-ticket-alt"></i> ${ticket.ticketCount || 1}
                            </span>
                        </span>
```

```
                </div>
                <div class="admin-ticket-detail">
                    <span class="admin-ticket-label">Price</span>
                    <span class="admin-ticket-value">
                        ₹${ticket.totalPrice || 0}
                    </span>
                </div>
                <div class="admin-ticket-detail">
                    <span class="admin-ticket-label">Status</span>
                    <span class="admin-status-badge admin-status-${status.toLowerCase()}">${status}</span>
                </div>
            </div>
            <div class="admin-ticket-actions">
                <button onclick="showTicketDetails('${ticket.id}')" class="admin-view-btn">
                    View Details
                </button>
            </div>
        </div>
    `;
  });

  ticketList.innerHTML = html;
}

    // Add function to get ticket status
    function getTicketStatus(ticket) {
        const now = new Date();
        const eventDate = new Date(ticket.eventDate);

        if (ticket.used) {
            return 'Used';
        } else if (eventDate < now) {
            return 'Expired';
        } else if (eventDate.toDateString() === now.toDateString()) {
            return 'Active';
        } else {
            return 'Upcoming';
        }
    }

    // Update the showTicketDetails function
    async function showTicketDetails(ticketId) {
        try {
            const ticketDoc = await getDoc(doc(db, 'tickets', ticketId));
            if (!ticketDoc.exists()) {
                throw new Error('Ticket not found');
            }

            const ticket = ticketDoc.data();

            const eventDoc = await getDoc(doc(db, 'events', ticket.eventId));
            const event = eventDoc.exists() ? eventDoc.data() : null;

            const quantity = ticket.ticketCount || 1;
            const unitPrice = (ticket.totalPrice || 0) / quantity;

            const qrCodeUrl = `https://api.qrserver.com/v1/create-qr-code/?size=200x200&data=${ticket.ticketIdentifier || ticketIdentifier}`;

            const modalHtml = `
                <div class="admin-ticket-modal">
                    <div class="admin-ticket-modal-content">
                        <div class="admin-ticket-modal-header">
                            <h2>Ticket Details</h2>
                            <span class="admin-close-modal">&times;</span>
                        </div>

                        <div class="admin-ticket-details-card">
                            <div class="admin-ticket-header">
                                <div class="admin-ticket-title">
                                    <h3>${event?.name || 'Event'}</h3>
                                    <span class="admin-ticket-type">${ticket.ticketType || 'Standard'}</span>
                                </div>
                                <div class="admin-ticket-persons">
                                    <i class="fas fa-users"></i> ${quantity} Person${quantity > 1 ? 's' : ''}
                                </div>
                            </div>

                            <div class="admin-ticket-grid">
```

```
                <div class="admin-detail-group">
                  <label>Ticket ID</label>
                  <span>${ticket.ticketIdentifier || ticketIdentifier}</span>
                </div>
                <div class="admin-detail-group">
                  <label>Purchaser Email</label>
                  <span>${ticket.userEmail || 'N/A'}</span>
                </div>
                <div class="admin-detail-group">
                  <label>Purchase Date</label>
                  <span>${new Date(ticket.purchaseDate).toLocaleString()}</span>
                </div>
                <div class="admin-detail-group">
                  <label>Event Date</label>
                  <span>${event ? new Date(event.date).toLocaleString() : 'N/A'}</span>
                </div>
                <div class="admin-detail-group">
                  <label>Total Price</label>
                  <span class="admin-price">
                    ₹${ticket.totalPrice || 0}
                    <small>(₹${unitPrice.toFixed(2)} × ${quantity})</small>
                  </span>
                </div>
                <div class="admin-detail-group">
                  <label>Status</label>
                  <span class="admin-status-badge admin-status-${getTicketStatus(ticket).toLowerCase()}">
                    ${getTicketStatus(ticket)}
                  </span>
                </div>
              </div>

              <div class="admin-ticket-qr-section">
                <div class="admin-qr-code">
                  <img src="${qrCodeUrl}" alt="Ticket QR Code" class="admin-ticket-qr">
                  <div class="admin-qr-id">ID: ${ticket.ticketIdentifier || ticketIdentifier}</div>
                </div>
              </div>

              ${event?.eventType === 'venue' || event?.eventType === 'hybrid' ? `
                <div class="admin-venue-details">
                  <h4>Venue Information</h4>
                  <div class="admin-detail-group">
                    <label>Venue</label>
                    <span>${event?.venueDetails?.venue || 'N/A'}</span>
                  </div>
                  <div class="admin-detail-group">
                    <label>Location</label>
                    <span>${event?.venueDetails?.place || 'N/A'}</span>
                  </div>
                </div>
              ` : ''}

              ${event?.eventType === 'online' || event?.eventType === 'hybrid' ? `
                <div class="admin-online-details">
                  <h4>Online Access</h4>
                  <div class="admin-detail-group">
                    <label>Platform</label>
                    <span>${event?.onlineDetails?.platform || 'N/A'}</span>
                  </div>
                  <div class="admin-detail-group secured-info">
                    <label>Access Details</label>
                    <span>Will be shared before the event</span>
                  </div>
                </div>
              ` : ''}
            </div>
          </div>
        </div>
    `;

    const modalContainer = document.createElement('div');
    modalContainer.innerHTML = modalHtml;
    document.body.appendChild(modalContainer);

    const closeBtn = modalContainer.querySelector('.admin-close-modal');
    closeBtn.onclick = () => modalContainer.remove();

    modalContainer.querySelector('.admin-ticket-modal').onclick = (e) => {
```

```
            if (e.target === modalContainer.querySelector('.admin-ticket-modal')) {
                modalContainer.remove();
            }
        };

    } catch (error) {
        console.error("Error showing ticket details:", error);
        alert("Error loading ticket details: " + error.message);
    }
}

// Function to download ticket as PNG
async function downloadTicket() {
    try {
        const ticketCard = document.getElementById('ticketCard');
        const canvas = await html2canvas(ticketCard, {
            scale: 2,
            logging: false,
            useCORS: true
        });

        const link = document.createElement('a');
        link.download = 'event-ticket.png';
        link.href = canvas.toDataURL('image/png');
        link.click();
    } catch (error) {
        console.error("Error downloading ticket:", error);
        alert("Error downloading ticket. Please try again.");
    }
}

// Make functions globally available
window.showTicketDetails = showTicketDetails;
window.downloadTicket = downloadTicket;

// Make getDoc available globally
window.getDoc = getDoc;

// Make functions globally available
window.showSection = showSection;
window.handleCreateEvent = handleCreateEvent;
window.deleteEvent = deleteEvent;
window.logout = handleLogout;
window.addImageField = addImageField;
window.removeImageField = removeImageField;
window.loadDashboard = loadDashboard;
window.loadEvents = loadEvents;
window.loadTickets = loadTickets;

// Make sure to call these functions when the page loads
document.addEventListener('DOMContentLoaded', () => {
    loadDashboard();
    loadEvents();
    loadTickets();
});

// Add real-time updates for events
function initializeRealTimeListeners() {
    const eventsRef = collection(db, 'events');
    const eventsQuery = query(eventsRef, orderBy('createdAt', 'desc'));

    const unsubscribe = onSnapshot(eventsQuery, (snapshot) => {
        const eventList = document.getElementById('eventList');
        eventList.innerHTML = '';

        if (snapshot.empty) {
            eventList.innerHTML = '<p>No events found</p>';
            return;
        }

        snapshot.forEach((doc) => {
            const event = doc.data();
            const eventCard = createEventCard(event, doc.id);
            eventList.appendChild(eventCard);
        });
    }, (error) => {
        console.error("Error in real-time updates:", error);
    });
```

```
    window.addEventListener('beforeunload', unsubscribe);
}

// Make functions globally available
window.openEditModal = openEditModal;
window.handleCreateEvent = handleCreateEvent;
window.clearEventForm = clearEventForm;
window.initializeRealTimeListeners = initializeRealTimeListeners;

// Initialize real-time updates when the page loads
document.addEventListener('DOMContentLoaded', () => {
    initializeRealTimeListeners();
});

// Add this after your Firebase initialization
auth.onAuthStateChanged((user) => {
    if (user) {
        console.log("Logged in user:", user.email);
        console.log("User ID:", user.uid);
    } else {
        console.log("No user logged in");
    }
});

// Add this logging code
auth.onAuthStateChanged((user) => {
    if (user) {
        console.log("Current user:", {
            email: user.email,
            uid: user.uid,
            emailVerified: user.emailVerified
        });
    } else {
        console.log("No user logged in");
    }
});

// Add this CSS for better statistics display

document.head.appendChild(style);

// Make sure to call loadDashboard when needed
document.addEventListener('DOMContentLoaded', () => {
    loadDashboard();
});

// Add real-time updates for dashboard
    // Get references to Firebase collections
const eventsRef = collection(db, 'events');
const ticketsRef = collection(db, 'tickets');

// Listen for ticket changes
onSnapshot(ticketsRef, async (snapshot) => {
    if (document.getElementById('dashboard').style.display === 'block') {
        await loadDashboard();
    }
});

// Listen for event changes
onSnapshot(eventsRef, async (snapshot) => {
    if (document.getElementById('dashboard').style.display === 'block') {
        await loadDashboard();
    }
});


// Make functions globally available
window.loadDashboard = loadDashboard;
window.initializeDashboardListeners = initializeDashboardListeners;

// Initialize dashboard listeners
document.addEventListener('DOMContentLoaded', () => {
    initializeDashboardListeners();
});

// Add these functions to your existing JavaScript
let allEvents = [];
```

```
let currentFilter = 'all';

// Function to generate unique event ID
function generateEventId() {
    const timestamp = new Date().getTime().toString(36);
    const randomStr = Math.random().toString(36).substring(2, 7);
    return `EVT-${timestamp}-${randomStr}`.toUpperCase();
}

// Load all events initially
async function loadFilteredEvents() {
    try {
        const eventsSnapshot = await getDocs(collection(db, 'events'));
        allEvents = [];

        eventsSnapshot.forEach(doc => {
            allEvents.push({
                id: doc.id,
                ...doc.data()
            });
        });

        applyFilters('all');
    } catch (error) {
        console.error("Error loading events:", error);
        document.getElementById('filteredEventsList').innerHTML = `
            <div class="error-message">
                Error loading events. Please try again.
                <button onclick="loadFilteredEvents()" class="retry-btn">Retry</button>
            </div>
        `;
    }
}

// Apply filters to events
function applyFilters(filterType = currentFilter) {
    currentFilter = filterType;

    document.querySelectorAll('.filter-btn').forEach(btn => {
        btn.classList.remove('active');
        if (btn.textContent.toLowerCase().includes(filterType.toLowerCase())) {
            btn.classList.add('active');
        }
    });

    const searchTerm = document.getElementById('eventSearchInput').value.toLowerCase();
    const dateFilter = document.getElementById('eventDateFilter').value;
    const now = new Date();
    const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());

    let filteredEvents = allEvents.filter(event => {
        const eventDate = new Date(event.date);
        const eventName = (event.name || '').toLowerCase();
        const eventId = (event.eventId || '').toLowerCase();

        if (searchTerm && !eventName.includes(searchTerm) && !eventId.includes(searchTerm)) {
            return false;
        }

        if (dateFilter) {
            const filterDate = new Date(dateFilter);
            const eventDateOnly = new Date(eventDate.getFullYear(), eventDate.getMonth(), eventDate.getDate());
            if (eventDateOnly.getTime() !== filterDate.getTime()) {
                return false;
            }
        }

        switch (filterType) {
            case 'upcoming':
                return eventDate > now;
            case 'today':
                return eventDate.getDate() === today.getDate() &&
                       eventDate.getMonth() === today.getMonth() &&
                       eventDate.getFullYear() === today.getFullYear();
            case 'past':
                return eventDate < now;
            default:
                return true;
```

```
            }
        });

        filteredEvents.sort((a, b) => new Date(b.date) - new Date(a.date));

        displayFilteredEvents(filteredEvents);
    }

    // Display filtered events
    function displayFilteredEvents(events) {
        const container = document.getElementById('filteredEventsList');

        if (events.length === 0) {
            container.innerHTML = `
                <div class="no-events-message">
                    <p>No events found matching your criteria.</p>
                </div>
            `;
            return;
        }

        let html = `
            <div class="events-grid">
                ${events.map(event => `
                    <div class="event-card">
                        <div class="event-header">
                            <div>
                                <h3>${event.name || 'Unnamed Event'}</h3>
                                <div class="event-id">ID: ${event.eventId || 'No ID'}</div>
                            </div>
                            ${getEventStatusBadge(event)}
                        </div>
                        <div class="event-details">
                            <p><strong>Date:</strong> ${new Date(event.date).toLocaleString()}</p>
                            <p><strong>Location:</strong> ${event.place || 'Not specified'}</p>
                            <p><strong>Venue:</strong> ${event.venue || 'Not specified'}</p>
                            <p><strong>Price:</strong> ₹${(event.price || 0).toFixed(2)}</p>
                            <p><strong>Available Tickets:</strong> ${event.availableTickets || 0}/${event.totalTickets || 0}</p>
                        </div>
                        <div class="event-actions">
                            <button onclick="handleEditClick('${event.id}', ${JSON.stringify(event).replace(/"/g, '&quot;')})" class="edit-
btn">Edit</button>
                            <button onclick="deleteEvent('${event.id}')" class="delete-btn">Delete</button>
                        </div>
                    </div>
                `).join('')}
            </div>
        `;

        container.innerHTML = html;
    }

    // Get event status badge
    function getEventStatusBadge(event) {
        const eventDate = new Date(event.date);
        const now = new Date();

        if (eventDate < now) {
            return '<span class="status-badge past">Past</span>';
        } else if (event.availableTickets <= 0) {
            return '<span class="status-badge sold-out">Sold Out</span>';
        } else {
            return '<span class="status-badge upcoming">Upcoming</span>';
        }
    }

    // Make functions globally available
    window.generateEventId = generateEventId;
    window.loadFilteredEvents = loadFilteredEvents;
    window.applyFilters = applyFilters;
    window.showSection = showSection;

    function handleEventTypeChange() {
        const eventType = document.getElementById('eventType').value;
        const onlineDetails = document.getElementById('onlineEventDetails');
        const venueDetails = document.getElementById('venueEventDetails');
        const ticketingSection = document.getElementById('ticketingSection');
        const standardTicketsSection = document.getElementById('standardTicketsSection');
```

```
    const hybridTicketsSection = document.getElementById('hybridTicketsSection');
    const priceField = document.getElementById('eventPrice').parentElement;

    onlineDetails.style.display = 'none';
    venueDetails.style.display = 'none';
    ticketingSection.style.display = 'none';
    standardTicketsSection.style.display = 'none';
    hybridTicketsSection.style.display = 'none';
    priceField.style.display = 'block';

    switch (eventType) {
        case 'online':
            onlineDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'venue':
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'hybrid':
            onlineDetails.style.display = 'block';
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            hybridTicketsSection.style.display = 'block';
            priceField.style.display = 'none';
            break;
    }
}

// Add validation for dates
function validateEventDates() {
    const eventDate = new Date(document.getElementById('eventDate').value);
    const bookingStartDate = new Date(document.getElementById('bookingStartDate').value);
    const bookingDeadline = new Date(document.getElementById('bookingDeadline').value);
    const now = new Date();

    document.getElementById('eventDate').min = now.toISOString().slice(0, 16);

    if (eventDate < now) {
        alert('Event date must be in the future');
        return false;
    }

    if (bookingStartDate > eventDate) {
        alert('Booking start date must be before event date');
        return false;
    }

    if (bookingDeadline > eventDate) {
        alert('Booking deadline must be before or same as event date');
        return false;
    }

    if (bookingDeadline < bookingStartDate) {
        alert('Booking deadline must be after booking start date');
        return false;
    }

    return true;
}

// Add validation for tickets
function validateTickets() {
    const eventType = document.getElementById('eventType').value;
    let isValid = true;

    if (eventType === 'hybrid') {
        const venueTickets = parseInt(document.getElementById('venueTickets').value) || 0;
        const onlineTickets = parseInt(document.getElementById('onlineTickets').value) || 0;
        const venuePrice = parseFloat(document.getElementById('venueTicketPrice').value) || 0;
        const onlinePrice = parseFloat(document.getElementById('onlineTicketPrice').value) || 0;

        if (venueTickets <= 0 && onlineTickets <= 0) {
            alert('Please specify ticket capacity for at least one ticket type');
            isValid = false;
        }
```

```
            if (venueTickets > 0 && venuePrice <= 0) {
                alert('Please specify a valid price for venue tickets');
                isValid = false;
            }

            if (onlineTickets > 0 && onlinePrice <= 0) {
                alert('Please specify a valid price for online tickets');
                isValid = false;
            }
        } else {
            const totalTickets = parseInt(document.getElementById('eventTickets').value) || 0;
            const price = parseFloat(document.getElementById('eventPrice').value) || 0;

            if (totalTickets <= 0) {
                alert('Total tickets must be greater than 0');
                isValid = false;
            }

            if (price <= 0) {
                alert('Please specify a valid ticket price');
                isValid = false;
            }
        }

        return isValid;
    }

    function cancelEdit() {
  try {
    // Clear the form
    clearEventForm();

    // Switch back to view mode
    toggleEventsView('view');

    // Reset editing state
    isEditing = false;
    currentEditingEventId = null;

    // Clear rich text editors
    const editors = ['eventDescription', 'eventRules'];
    editors.forEach(editorId => {
        const editor = document.getElementById(editorId);
        if (editor) {
            editor.innerHTML = '';
        }
    });

    // Hide cancel button
    const cancelBtn = document.querySelector('.cancel-btn');
    if (cancelBtn) {
        cancelBtn.style.display = 'none';
    }

    // Reset create button text
    const createBtn = document.querySelector('.create-btn');
    if (createBtn) {
        createBtn.textContent = 'Create Event';
    }

  } catch (error) {
    console.error("Error in cancel edit:", error);
  }
}

    // Add to your existing global assignments
    window.cancelEdit = cancelEdit;

    // Update the switchVerificationMethod function
    function switchVerificationMethod(method) {
        if (window.html5QrcodeScanner) {
            window.html5QrcodeScanner.clear();
            window.html5QrcodeScanner = null;
        }

        document.querySelectorAll('.tab-btn').forEach(btn => {
            btn.classList.remove('active');
```

```
    });
    document.querySelector(`.tab-btn[onclick="window.switchVerificationMethod('${method}')"]`).classList.add('active');

    document.querySelectorAll('.verification-method').forEach(el => {
        el.style.display = 'none';
    });
    document.getElementById(`${method}Verification`).style.display = 'block';

    if (method === 'qr') {
        window.initQRScanner();
        document.querySelector('.scan-btn').style.display = 'block';
        document.querySelector('[onclick="window.stopQRScanner()"]').style.display = 'none';
    }
}

window.showVerificationResult = function(result) {
    const resultDiv = document.getElementById('verificationResult');
    const statusSpan = resultDiv.querySelector('.result-status');
    const detailsDiv = resultDiv.querySelector('.ticket-details');
    const markUsedBtn = resultDiv.querySelector('.mark-used-btn');

    statusSpan.className = 'result-status ' + result.status;
    statusSpan.textContent = result.message;

    if (result.ticket && result.event) {
        detailsDiv.innerHTML = `
            <h4>Ticket Information</h4>
            <p><strong>Ticket ID:</strong> ${result.ticket.ticketId}</p>
            <p><strong>Event:</strong> ${result.event.name}</p>
            <p><strong>Event Date:</strong> ${new Date(result.event.date).toLocaleString()}</p>
            <p><strong>Ticket Type:</strong> ${result.ticket.ticketType || 'Standard'}</p>
            <p><strong>Purchase Date:</strong> ${new Date(result.ticket.purchaseDate).toLocaleString()}</p>
            <p><strong>Purchased By:</strong> ${result.ticket.userEmail}</p>
        `;
        markUsedBtn.style.display = result.status === 'valid' ? 'block' : 'none';
    } else {
        detailsDiv.innerHTML = '';
        markUsedBtn.style.display = 'none';
    }

    resultDiv.style.display = 'block';
};

window.switchVerificationMethod = async function(method) {
    try {
        if (window.html5QrcodeScanner) {
            await window.html5QrcodeScanner.stop();
            window.html5QrcodeScanner = null;
        }

        document.querySelectorAll('.tab-btn').forEach(btn => {
            btn.classList.remove('active');
        });
        document.querySelector(`.tab-btn[onclick="window.switchVerificationMethod('${method}')"]`).classList.add('active');

        document.querySelectorAll('.verification-method').forEach(el => {
            el.style.display = 'none';
        });
        document.getElementById(`${method}Verification`).style.display = 'block';

        if (method === 'qr') {
            await window.initQRScanner();
            document.querySelector('.scan-btn').style.display = 'block';
            document.querySelector('[onclick="window.stopQRScanner()"]').style.display = 'none';
        }
    } catch (error) {
        console.error("Error switching verification method:", error);
        alert("Error: " + error.message);
    }
};

if (typeof Html5QrCode === 'undefined') {
    console.error('HTML5QrCode library not loaded. Please check the script inclusion.');
}

function toggleEventsView(view) {
    document.querySelectorAll('.toggle-btn').forEach(btn => {
        btn.classList.remove('active');
```

```
    });
    document.querySelector(`.toggle-btn[onclick*="${view}"]`).classList.add('active');

    const createSection = document.getElementById('createEventSection');
    const viewSection = document.getElementById('viewEventSection');

    if (view === 'create') {
        createSection.style.display = 'block';
        viewSection.style.display = 'none';
    } else {
        createSection.style.display = 'none';
        viewSection.style.display = 'block';
        loadEvents();
    }
}

function filterEvents() {
    const searchTerm = document.getElementById('eventSearchInput').value.toLowerCase();
    const typeFilter = document.getElementById('eventTypeFilter').value;
    const statusFilter = document.getElementById('eventStatusFilter').value;
    const eventList = document.getElementById('eventList');
    const eventCards = eventList.querySelectorAll('.event-card');
    const now = new Date();

    eventCards.forEach(card => {
        let showCard = true;

        const eventName = card.querySelector('h3').textContent.toLowerCase();
        const eventId = card.querySelector('.event-id').textContent.toLowerCase();
        const eventType = card.querySelector('.event-type-badge')?.textContent.toLowerCase() || '';
        const eventDateText = card.querySelector('.event-meta p:first-child')?.textContent;
        const eventDate = eventDateText ? new Date(eventDateText.split(': ')[1]) : now;

        if (searchTerm && !eventName.includes(searchTerm) && !eventId.includes(searchTerm)) {
            showCard = false;
        }

        if (typeFilter && !eventType.includes(typeFilter.toLowerCase())) {
            showCard = false;
        }

        if (statusFilter) {
            const eventTime = eventDate.getTime();
            const nowTime = now.getTime();
            const oneDayMs = 24 * 60 * 60 * 1000;

            switch (statusFilter) {
                case 'upcoming':
                    if (eventTime <= nowTime) showCard = false;
                    break;
                case 'ongoing':
                    if (eventTime < nowTime || eventTime > (nowTime + oneDayMs)) showCard = false;
                    break;
                case 'completed':
                    if (eventTime > nowTime) showCard = false;
                    break;
            }
        }

        card.style.display = showCard ? '' : 'none';
    });
}

// Make functions globally available
window.toggleEventsView = toggleEventsView;
window.filterEvents = filterEvents;

// Add this function for PNG download
function downloadTicketPNG(ticketId) {
    try {
        const ticketCard = document.getElementById('ticketCard');
        if (!ticketCard) {
            throw new Error('Ticket card element not found');
        }

        html2canvas(ticketCard).then(canvas => {
            const link = document.createElement('a');
            link.download = `ticket-${ticketId}.png`;
```

```
            link.href = canvas.toDataURL('image/png');

            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        });

    } catch (error) {
        console.error("Error downloading ticket:", error);
        alert("Error downloading ticket: " + error.message);
    }
}

// Update the button in showTicketDetails function
// Replace the download button HTML with this:
`<div class="admin-ticket-actions">
    <button onclick="downloadTicketPNG('${ticketId}')" class="admin-download-btn">
        <i class="fas fa-download"></i> Download Ticket (PNG)
    </button>
</div>`

// Update the verification function
async function verifyTicket(ticketId) {
    try {
        // Query tickets collection using ticketIdentifier
        const ticketsRef = collection(db, 'tickets');
        const q = query(ticketsRef, where('ticketIdentifier', '==', ticketId));
        const querySnapshot = await getDocs(q);

        if (querySnapshot.empty) {
            return {
                status: 'invalid',
                message: 'Invalid Ticket',
                ticket: null,
                event: null
            };
        }

        const ticketDoc = querySnapshot.docs[0];
        const ticket = ticketDoc.data();

        // Get event details
        const eventDoc = await getDoc(doc(db, 'events', ticket.eventId));
        if (!eventDoc.exists()) {
            return {
                status: 'invalid',
                message: 'Event not found',
                ticket,
                event: null
            };
        }

        const event = eventDoc.data();
        const now = new Date();
        const eventDate = new Date(event.date);

        if (ticket.used) {
            return {
                status: 'used',
                message: 'Ticket already used',
                ticket,
                event
            };
        }

        if (eventDate < now) {
            return {
                status: 'expired',
                message: 'Event has ended',
                ticket,
                event
            };
        }

        return {
            status: 'valid',
            message: 'Valid Ticket',
            ticket,
```

```
            event
        };
    } catch (error) {
        console.error("Error verifying ticket:", error);
        return {
            status: 'error',
            message: 'Error verifying ticket',
            ticket: null,
            event: null
        };
    }
}

// Update the mark ticket as used function
async function markTicketAsUsed() {
    try {
        const ticketId = document.querySelector('.ticket-details p:first-child strong').nextSibling.textContent.trim();

        // Query to find the ticket document using ticketIdentifier
        const ticketsRef = collection(db, 'tickets');
        const q = query(ticketsRef, where('ticketIdentifier', '==', ticketId));
        const querySnapshot = await getDocs(q);

        if (!querySnapshot.empty) {
            const ticketDoc = querySnapshot.docs[0];
            await updateDoc(ticketDoc.ref, {
                used: true,
                usedDate: new Date().toISOString()
            });

            alert('Ticket marked as used successfully!');
            document.getElementById('verificationResult').style.display = 'none';
        } else {
            throw new Error('Ticket not found');
        }
    } catch (error) {
        console.error("Error marking ticket as used:", error);
        alert('Error marking ticket as used: ' + error.message);
    }
}

// Add these functions to handle image management
function handleImageUpload(event) {
    const files = event.target.files;
    const imageGrid = document.getElementById('imageGrid');

    Array.from(files).forEach(file => {
        const reader = new FileReader();
        reader.onload = function(e) {
            const imageItem = document.createElement('div');
            imageItem.className = 'image-item';
            imageItem.innerHTML = `
                <img src="${e.target.result}" alt="Event image">
                <div class="image-actions">
                    <button class="image-action-btn" onclick="moveImage(this, 'up')" title="Move Up">
                        <i class="fas fa-arrow-up"></i>
                    </button>
                    <button class="image-action-btn" onclick="moveImage(this, 'down')" title="Move Down">
                        <i class="fas fa-arrow-down"></i>
                    </button>
                    <button class="image-action-btn delete" onclick="deleteImage(this)" title="Delete">
                        <i class="fas fa-trash"></i>
                    </button>
                </div>
            `;
            imageGrid.appendChild(imageItem);
        };
        reader.readAsDataURL(file);
    });
}

function moveImage(button, direction) {
    const imageItem = button.closest('.image-item');
    const grid = imageItem.parentElement;

    if (direction === 'up' && imageItem.previousElementSibling) {
        grid.insertBefore(imageItem, imageItem.previousElementSibling);
    } else if (direction === 'down' && imageItem.nextElementSibling) {
```

```
            grid.insertBefore(imageItem.nextElementSibling, imageItem);
        }
    }

    function deleteImage(button) {
        if (confirm('Are you sure you want to delete this image?')) {
            button.closest('.image-item').remove();
        }
    }

    // Add event listener for image upload
    document.getElementById('imageUpload').addEventListener('change', handleImageUpload);
</script>

<!-- Add this modal HTML just before closing body tag -->
<div id="editEventModal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2>Edit Event</h2>
        <div class="form-group">
            <label>Event Name</label>
            <input type="text" id="editEventName" required>
        </div>
        <div class="form-group">
            <label>Description</label>
            <textarea id="editEventDescription" required></textarea>
        </div>
        <div class="form-group">
            <label>Event Date</label>
            <input type="datetime-local" id="editEventDate" required>
        </div>
        <div class="form-group">
            <label>Booking Start Date</label>
            <input type="datetime-local" id="editBookingStartDate" required>
        </div>
        <div class="form-group">
            <label>Booking Deadline</label>
            <input type="datetime-local" id="editBookingDeadline" required>
        </div>
        <div class="form-group">
            <label>Price (₹)</label>
            <input type="number" id="editEventPrice" step="0.01" required>
        </div>
        <div class="form-group">
            <label>Total Tickets</label>
            <input type="number" id="editEventTickets" required min="1">
        </div>
        <div id="editImageUrlsContainer">
            <!-- Image inputs will be added here -->
        </div>
        <div class="form-group" style="display: flex; gap: 10px;">
            <button type="button" onclick="updateEvent()" class="edit-btn">Update Event</button>
            <button type="button" onclick="cancelEdit()" class="cancel-btn" style="display: none;">Cancel Edit</button>
        </div>
    </div>
</div>

<script>
auth.onAuthStateChanged((user) => {
    if (user) {
        console.log('User is signed in:', user.email);
        document.getElementById('userEmail').textContent = user.email;
    } else {
        console.log('User is signed out');
        window.location.href = 'login.html';
    }
});
</script>

<script>
async function loadRecentEvents() {
    try {
        const eventsRef = collection(db, 'events');
        const recentEventsQuery = query(eventsRef, orderBy('createdAt', 'desc'), limit(5));
        const recentEventsSnapshot = await getDocs(recentEventsQuery);

        const recentEventsList = document.getElementById('recentEventsList');
        recentEventsList.innerHTML = '';
```

```
            for (const eventDoc of recentEventsSnapshot.docs) {
                const event = eventDoc.data();
                const eventDate = new Date(event.date).toLocaleDateString();
                const eventStatus = getEventStatus(event);

                const eventItem = document.createElement('div');
                eventItem.className = 'recent-item';
                eventItem.innerHTML = `
                    <div class="recent-item-title">${event.name}</div>
                    <div class="recent-item-details">
                        <span>${eventDate}</span>
                        <span>${event.eventType}</span>
                        <span>${eventStatus}</span>
                    </div>
                `;
                recentEventsList.appendChild(eventItem);
            }
        } catch (error) {
            console.error("Error loading recent events:", error);
        }
    }

    // Make sure these functions are globally available
    window.loadRecentEvents = loadRecentEvents;
    window.getEventStatus = getEventStatus;
    </script>
<script src="https://unpkg.com/html5-qrcode@2.3.8/html5-qrcode.min.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js"></script>
<script src="https://html2canvas.hertzen.com/dist/html2canvas.min.js"></script>
<script type="module">
    import { initializeApp } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-app.js";
    import { getFirestore } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js";
    import { getAuth } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js";

    const firebaseConfig = {
        apiKey: "AIzaSyCg7vwuwfN8oWSMagExshHCtMHnxzc7pH0",
        authDomain: "event-ticket-fc753.firebaseapp.com",
        projectId: "event-ticket-fc753",
        storageBucket: "event-ticket-fc753.firebasestorage.app",
        messagingSenderId: "216313948465",
        appId: "1:216313948465:web:4c4b8eb9fbb4257fe6e810",
        measurementId: "G-SGBZK0BYVL"
    };

    // Initialize Firebase
    const app = initializeApp(firebaseConfig);
    window.db = getFirestore(app);
    window.auth = getAuth(app);

    // Define the function
</script>

<script>
    // Add this after Firebase initialization
    document.addEventListener('DOMContentLoaded', function() {
    const logoutBtn = document.getElementById('logoutBtn');
    if (logoutBtn) {
        logoutBtn.addEventListener('click', async function() {
            try {
                // First sign out from Firebase
                await auth.signOut();

                // Clear all storage
                localStorage.clear();
                sessionStorage.clear();

                // Get the base URL for GitHub Pages
                const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';

                // Force redirect to index page with correct path
                window.location.replace(`${baseUrl}/index.html`);

                // Prevent back navigation
                window.history.pushState(null, '', `${baseUrl}/index.html`);
```

```
            } catch (error) {
                console.error("Logout error:", error);
                // Force redirect even if there's an error
                const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
                window.location.replace(`${baseUrl}/index.html`);
            }
        });
    }
});
</script>

<!-- Add reference to external JavaScript file -->
<script type="module" src="scripts/admin.js"></script>
    <!-- Loader iframe -->

    <!-- Add this function to handle scheduler sections -->
    <script>
    function toggleScheduleSection(type) {
        const section = document.getElementById(`${type}ScheduleSection`);
        const checkbox = document.getElementById(`enable${type.charAt(0).toUpperCase() + type.slice(1)}Schedule`);

        if (checkbox.checked) {
            section.style.display = 'block';
            // Set minimum date to today for publish date
            if (type === 'publish') {
                const today = new Date();
                today.setMinutes(today.getMinutes() - today.getTimezoneOffset());
                document.getElementById('publishDate').min = today.toISOString().slice(0, 16);
            }
            // Set minimum date to event date for hide date
            if (type === 'hide') {
                const eventDate = document.getElementById('eventDate').value;
                if (eventDate) {
                    document.getElementById('hideDate').min = eventDate;
                }
            }
        } else {
            section.style.display = 'none';
            document.getElementById(`${type}Date`).value = '';
        }
    }

    // Add event listener for event date change
    document.getElementById('eventDate').addEventListener('change', function() {
        const hideDate = document.getElementById('hideDate');
        if (hideDate && this.value) {
            hideDate.min = this.value;
        }
    });

    // Make the function globally available
    window.toggleScheduleSection = toggleScheduleSection;
    </script>

    <script>
    async function handleEventSelect() {
        const eventSelect = document.getElementById('eventSelect');
        const ticketInfoDiv = document.getElementById('ticketInfo');
        const generateBtn = document.getElementById('generateTicketBtn');

        if (!eventSelect.value) {
            ticketInfoDiv.innerHTML = '';
            generateBtn.disabled = true;
            return;
        }

        try {
            // Get event details
            const eventDoc = await getDoc(doc(db, 'events', eventSelect.value));
            if (!eventDoc.exists()) {
                throw new Error('Event not found');
            }

            const event = eventDoc.data();
            const now = new Date();
            const bookingStart = new Date(event.bookingStartDate);
            const bookingEnd = new Date(event.bookingDeadline);
```

```
// Check if booking is allowed
if (now < bookingStart) {
    ticketInfoDiv.innerHTML = `
        <div class="alert alert-warning">
            <i class="fas fa-clock"></i>
            Booking hasn't started yet. Starts on ${bookingStart.toLocaleDateString()} at ${bookingStart.toLocaleTimeString()}
        </div>
    `;
    generateBtn.disabled = true;
    return;
}

if (now > bookingEnd) {
    ticketInfoDiv.innerHTML = `
        <div class="alert alert-error">
            <i class="fas fa-times-circle"></i>
            Booking period has ended
        </div>
    `;
    generateBtn.disabled = true;
    return;
}

// Check ticket availability
let availabilityHTML = '';
let canGenerate = false;

if (event.eventType === 'hybrid') {
    const venueAvailable = event.tickets?.venue?.available || 0;
    const onlineAvailable = event.tickets?.online?.available || 0;

    availabilityHTML = `
        <div class="ticket-availability">
            <h4>Ticket Availability:</h4>
            <div class="availability-item ${venueAvailable > 0 ? 'available' : 'sold-out'}">
                <span>Venue Tickets:</span>
                <span>${venueAvailable} available</span>
                <span>Price: ₹${event.tickets?.venue?.price || 0}</span>
            </div>
            <div class="availability-item ${onlineAvailable > 0 ? 'available' : 'sold-out'}">
                <span>Online Tickets:</span>
                <span>${onlineAvailable} available</span>
                <span>Price: ₹${event.tickets?.online?.price || 0}</span>
            </div>
        </div>
    `;
    canGenerate = venueAvailable > 0 || onlineAvailable > 0;
} else {
    const available = event.availableTickets || 0;
    availabilityHTML = `
        <div class="ticket-availability">
            <h4>Ticket Availability:</h4>
            <div class="availability-item ${available > 0 ? 'available' : 'sold-out'}">
                <span>Available Tickets:</span>
                <span>${available}</span>
                <span>Price: ₹${event.price || 0}</span>
            </div>
        </div>
    `;
    canGenerate = available > 0;
}

ticketInfoDiv.innerHTML = `
    <div class="event-info">
        <h3>${event.name}</h3>
        <p><i class="fas fa-calendar"></i> ${new Date(event.date).toLocaleDateString()}</p>
        ${availabilityHTML}
    </div>
`;

generateBtn.disabled = !canGenerate;
if (!canGenerate) {
    ticketInfoDiv.innerHTML += `
        <div class="alert alert-error">
            <i class="fas fa-exclamation-circle"></i>
            No tickets available for this event
        </div>
```

```
            `;
        }

    } catch (error) {
        console.error('Error fetching event details:', error);
        ticketInfoDiv.innerHTML = `
            <div class="alert alert-error">
                <i class="fas fa-exclamation-circle"></i>
                Error fetching event details
            </div>
        `;
        generateBtn.disabled = true;
    }
}
</script>

<script>
function toggleMoreMenu() {
    const moreMenu = document.getElementById('moreMenu');
    const moreButton = document.querySelector('.mobile-nav-item[onclick="toggleMoreMenu()"]');

    moreMenu.classList.toggle('active');
    moreButton.classList.toggle('active');

    // Update the More button icon
    const moreIcon = moreButton.querySelector('i');
    if (moreMenu.classList.contains('active')) {
        moreIcon.classList.remove('fa-ellipsis-h');
        moreIcon.classList.add('fa-times');
    } else {
        moreIcon.classList.remove('fa-times');
        moreIcon.classList.add('fa-ellipsis-h');
    }
}

// Close more menu when clicking outside
document.addEventListener('click', function(event) {
    const moreMenu = document.getElementById('moreMenu');
    const moreButton = document.querySelector('.mobile-nav-item[onclick="toggleMoreMenu()"]');

    if (!moreButton.contains(event.target) && !moreMenu.contains(event.target)) {
        moreMenu.classList.remove('active');
        moreButton.classList.remove('active');
        const moreIcon = moreButton.querySelector('i');
        moreIcon.classList.remove('fa-times');
        moreIcon.classList.add('fa-ellipsis-h');
    }
});
</script>

<script>
// Global logout function
function logout() {
    try {
        // First sign out from Firebase
        auth.signOut().then(() => {
            // Clear all storage
            localStorage.clear();
            sessionStorage.clear();

            // Get the base URL for GitHub Pages
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';

            // Force redirect to index page with correct path
            window.location.replace(`${baseUrl}/index.html`);

            // Prevent back navigation
            window.history.pushState(null, '', `${baseUrl}/index.html`);

        }).catch((error) => {
            console.error("Error signing out:", error);
            // Force redirect even if there's an error
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
            window.location.replace(`${baseUrl}/index.html`);
        });
    } catch (error) {
        console.error("Logout error:", error);
        // Force redirect even if there's an error
```

```
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
            window.location.replace(`${baseUrl}/index.html`);
        }
    }

    // Make logout function globally available
    window.logout = logout;
</script>

</body>
</html>

<!-- Add this right before closing body tag -->
<div class="mobile-nav">
    <div class="mobile-nav-items">
        <div class="mobile-nav-item active" onclick="showSection('dashboard')">
            <i class="fas fa-tachometer-alt"></i>
            <span>Dashboard</span>
        </div>

        <div class="mobile-nav-item" onclick="showSection('events')">
            <i class="fas fa-calendar-alt"></i>
            <span>Events</span>
        </div>

        <div class="mobile-nav-item" onclick="showSection('tickets')">
            <i class="fas fa-ticket-alt"></i>
            <span>Tickets</span>
        </div>

        <div class="mobile-nav-item" onclick="showSection('passes')">
            <i class="fas fa-id-card"></i>
            <span>Passes</span>
        </div>

        <div class="mobile-nav-item" onclick="toggleMoreMenu()">
            <i class="fas fa-ellipsis-h"></i>
            <span>More</span>
        </div>
    </div>

    <!-- Move more menu inside mobile-nav -->
    <div class="more-menu" id="moreMenu">
        <div class="mobile-nav-item" onclick="showSection('verification')">
            <i class="fas fa-check-circle"></i>
            <span>Verify</span>
        </div>
        <div class="mobile-nav-item" onclick="showSection('generateTicket')">
            <i class="fas fa-plus-circle"></i>
            <span>Generate</span>
        </div>
    </div>
</div>

<script>
    // Functions for handling event category and pricing type
    function handleEventCategoryChange() {
        const categorySelect = document.getElementById('eventCategory');
        const otherCategoryGroup = document.getElementById('otherCategoryGroup');
        const otherCategoryInput = document.getElementById('otherCategoryName');

        if (categorySelect.value === 'other') {
            otherCategoryGroup.style.display = 'block';
            otherCategoryInput.required = true;
        } else {
            otherCategoryGroup.style.display = 'none';
            otherCategoryInput.required = false;
            otherCategoryInput.value = '';
        }
    }

    function handlePricingTypeChange() {
    const pricingType = document.getElementById('eventPricingType').value;
    const priceInputs = [
        document.getElementById('eventPrice'),
        document.getElementById('venueTicketPrice'),
        document.getElementById('onlineTicketPrice')
    ];
```

```
    priceInputs.forEach(input => {
        if (input) {
            if (pricingType === 'free') {
                input.value = '0';
                input.readOnly = true;
            } else {
                input.value = '';
                input.readOnly = false;
            }
        }
    });
}
        let currentFilter = 'all';

        // Function to generate unique event ID
        function generateEventId() {
            const timestamp = new Date().getTime().toString(36);
            const randomStr = Math.random().toString(36).substring(2, 7);
            return `EVT-${timestamp}-${randomStr}`.toUpperCase();
        }

        // Load all events initially
        async function loadFilteredEvents() {
            try {
                const eventsSnapshot = await getDocs(collection(db, 'events'));
                allEvents = [];

                eventsSnapshot.forEach(doc => {
                    allEvents.push({
                        id: doc.id,
                        ...doc.data()
                    });
                });

                applyFilters('all');
            } catch (error) {
                console.error("Error loading events:", error);
                document.getElementById('filteredEventsList').innerHTML = `
                    <div class="error-message">
                        Error loading events. Please try again.
                        <button onclick="loadFilteredEvents()" class="retry-btn">Retry</button>
                    </div>
                `;
            }
        }

        // Apply filters to events
        function applyFilters(filterType = currentFilter) {
            currentFilter = filterType;

            document.querySelectorAll('.filter-btn').forEach(btn => {
                btn.classList.remove('active');
                if (btn.textContent.toLowerCase().includes(filterType.toLowerCase())) {
                    btn.classList.add('active');
                }
            });

            const searchTerm = document.getElementById('eventSearchInput').value.toLowerCase();
            const dateFilter = document.getElementById('eventDateFilter').value;
            const now = new Date();
            const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());

            let filteredEvents = allEvents.filter(event => {
                const eventDate = new Date(event.date);
                const eventName = (event.name || '').toLowerCase();
                const eventId = (event.eventId || '').toLowerCase();

                if (searchTerm && !eventName.includes(searchTerm) && !eventId.includes(searchTerm)) {
                    return false;
                }

                if (dateFilter) {
                    const filterDate = new Date(dateFilter);
                    const eventDateOnly = new Date(eventDate.getFullYear(), eventDate.getMonth(), eventDate.getDate());
                    if (eventDateOnly.getTime() !== filterDate.getTime()) {
                        return false;
                    }
```

```
            }

            switch (filterType) {
                case 'upcoming':
                    return eventDate > now;
                case 'today':
                    return eventDate.getDate() === today.getDate() &&
                           eventDate.getMonth() === today.getMonth() &&
                           eventDate.getFullYear() === today.getFullYear();
                case 'past':
                    return eventDate < now;
                default:
                    return true;
            }
        });

        filteredEvents.sort((a, b) => new Date(b.date) - new Date(a.date));

        displayFilteredEvents(filteredEvents);
    }

    // Display filtered events
    function displayFilteredEvents(events) {
        const container = document.getElementById('filteredEventsList');

        if (events.length === 0) {
            container.innerHTML = `
                <div class="no-events-message">
                    <p>No events found matching your criteria.</p>
                </div>
            `;
            return;
        }

        let html = `
            <div class="events-grid">
                ${events.map(event => `
                    <div class="event-card">
                        <div class="event-header">
                            <div>
                                <h3>${event.name || 'Unnamed Event'}</h3>
                                <div class="event-id">ID: ${event.eventId || 'No ID'}</div>
                            </div>
                            ${getEventStatusBadge(event)}
                        </div>
                        <div class="event-details">
                            <p><strong>Date:</strong> ${new Date(event.date).toLocaleString()}</p>
                            <p><strong>Location:</strong> ${event.place || 'Not specified'}</p>
                            <p><strong>Venue:</strong> ${event.venue || 'Not specified'}</p>
                            <p><strong>Price:</strong> ₹${(event.price || 0).toFixed(2)}</p>
                            <p><strong>Available Tickets:</strong> ${event.availableTickets || 0}/${event.totalTickets || 0}</p>
                        </div>
                        <div class="event-actions">
                            <button onclick="handleEditClick('${event.id}', ${JSON.stringify(event).replace(/"/g, '&quot;')})" class="edit-
btn">Edit</button>
                            <button onclick="deleteEvent('${event.id}')" class="delete-btn">Delete</button>
                        </div>
                    </div>
                `).join('')}
            </div>
        `;

        container.innerHTML = html;
    }

    // Get event status badge
    function getEventStatusBadge(event) {
        const eventDate = new Date(event.date);
        const now = new Date();

        if (eventDate < now) {
            return '<span class="status-badge past">Past</span>';
        } else if (event.availableTickets <= 0) {
            return '<span class="status-badge sold-out">Sold Out</span>';
        } else {
            return '<span class="status-badge upcoming">Upcoming</span>';
        }
    }
```

```
// Make functions globally available
window.generateEventId = generateEventId;
window.loadFilteredEvents = loadFilteredEvents;
window.applyFilters = applyFilters;
window.showSection = showSection;

function handleEventTypeChange() {
    const eventType = document.getElementById('eventType').value;
    const onlineDetails = document.getElementById('onlineEventDetails');
    const venueDetails = document.getElementById('venueEventDetails');
    const ticketingSection = document.getElementById('ticketingSection');
    const standardTicketsSection = document.getElementById('standardTicketsSection');
    const hybridTicketsSection = document.getElementById('hybridTicketsSection');
    const priceField = document.getElementById('eventPrice').parentElement;

    onlineDetails.style.display = 'none';
    venueDetails.style.display = 'none';
    ticketingSection.style.display = 'none';
    standardTicketsSection.style.display = 'none';
    hybridTicketsSection.style.display = 'none';
    priceField.style.display = 'block';

    switch (eventType) {
        case 'online':
            onlineDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'venue':
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'hybrid':
            onlineDetails.style.display = 'block';
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            hybridTicketsSection.style.display = 'block';
            priceField.style.display = 'none';
            break;
    }
}

// Add validation for dates
function validateEventDates() {
    const eventDate = new Date(document.getElementById('eventDate').value);
    const bookingStartDate = new Date(document.getElementById('bookingStartDate').value);
    const bookingDeadline = new Date(document.getElementById('bookingDeadline').value);
    const now = new Date();

    document.getElementById('eventDate').min = now.toISOString().slice(0, 16);

    if (eventDate < now) {
        alert('Event date must be in the future');
        return false;
    }

    if (bookingStartDate > eventDate) {
        alert('Booking start date must be before event date');
        return false;
    }

    if (bookingDeadline > eventDate) {
        alert('Booking deadline must be before or same as event date');
        return false;
    }

    if (bookingDeadline < bookingStartDate) {
        alert('Booking deadline must be after booking start date');
        return false;
    }

    return true;
}

// Add validation for tickets
function validateTickets() {
```

```
        const eventType = document.getElementById('eventType').value;
        let isValid = true;

        if (eventType === 'hybrid') {
            const venueTickets = parseInt(document.getElementById('venueTickets').value) || 0;
            const onlineTickets = parseInt(document.getElementById('onlineTickets').value) || 0;
            const venuePrice = parseFloat(document.getElementById('venueTicketPrice').value) || 0;
            const onlinePrice = parseFloat(document.getElementById('onlineTicketPrice').value) || 0;

            if (venueTickets <= 0 && onlineTickets <= 0) {
                alert('Please specify ticket capacity for at least one ticket type');
                isValid = false;
            }

            if (venueTickets > 0 && venuePrice <= 0) {
                alert('Please specify a valid price for venue tickets');
                isValid = false;
            }

            if (onlineTickets > 0 && onlinePrice <= 0) {
                alert('Please specify a valid price for online tickets');
                isValid = false;
            }
        } else {
            const totalTickets = parseInt(document.getElementById('eventTickets').value) || 0;
            const price = parseFloat(document.getElementById('eventPrice').value) || 0;

            if (totalTickets <= 0) {
                alert('Total tickets must be greater than 0');
                isValid = false;
            }

            if (price <= 0) {
                alert('Please specify a valid ticket price');
                isValid = false;
            }
        }

        return isValid;
    }

    function cancelEdit() {
  try {
    // Clear the form
    clearEventForm();

    // Switch back to view mode
    toggleEventsView('view');

    // Reset editing state
    isEditing = false;
    currentEditingEventId = null;

    // Clear rich text editors
    const editors = ['eventDescription', 'eventRules'];
    editors.forEach(editorId => {
        const editor = document.getElementById(editorId);
        if (editor) {
            editor.innerHTML = '';
        }
    });

    // Hide cancel button
    const cancelBtn = document.querySelector('.cancel-btn');
    if (cancelBtn) {
        cancelBtn.style.display = 'none';
    }

    // Reset create button text
    const createBtn = document.querySelector('.create-btn');
    if (createBtn) {
        createBtn.textContent = 'Create Event';
    }

  } catch (error) {
    console.error("Error in cancel edit:", error);
  }
}
```

```
// Add to your existing global assignments
window.cancelEdit = cancelEdit;

// Update the switchVerificationMethod function
function switchVerificationMethod(method) {
    if (window.html5QrcodeScanner) {
        window.html5QrcodeScanner.clear();
        window.html5QrcodeScanner = null;
    }

    document.querySelectorAll('.tab-btn').forEach(btn => {
        btn.classList.remove('active');
    });
    document.querySelector(`.tab-btn[onclick="window.switchVerificationMethod('${method}')"]`).classList.add('active');

    document.querySelectorAll('.verification-method').forEach(el => {
        el.style.display = 'none';
    });
    document.getElementById(`${method}Verification`).style.display = 'block';

    if (method === 'qr') {
        window.initQRScanner();
        document.querySelector('.scan-btn').style.display = 'block';
        document.querySelector('[onclick="window.stopQRScanner()"]').style.display = 'none';
    }
}

window.showVerificationResult = function(result) {
    const resultDiv = document.getElementById('verificationResult');
    const statusSpan = resultDiv.querySelector('.result-status');
    const detailsDiv = resultDiv.querySelector('.ticket-details');
    const markUsedBtn = resultDiv.querySelector('.mark-used-btn');

    statusSpan.className = 'result-status ' + result.status;
    statusSpan.textContent = result.message;

    if (result.ticket && result.event) {
        detailsDiv.innerHTML = `
            <h4>Ticket Information</h4>
            <p><strong>Ticket ID:</strong> ${result.ticket.ticketId}</p>
            <p><strong>Event:</strong> ${result.event.name}</p>
            <p><strong>Event Date:</strong> ${new Date(result.event.date).toLocaleString()}</p>
            <p><strong>Ticket Type:</strong> ${result.ticket.ticketType || 'Standard'}</p>
            <p><strong>Purchase Date:</strong> ${new Date(result.ticket.purchaseDate).toLocaleString()}</p>
            <p><strong>Purchased By:</strong> ${result.ticket.userEmail}</p>
        `;
        markUsedBtn.style.display = result.status === 'valid' ? 'block' : 'none';
    } else {
        detailsDiv.innerHTML = '';
        markUsedBtn.style.display = 'none';
    }

    resultDiv.style.display = 'block';
};

window.switchVerificationMethod = async function(method) {
    try {
        if (window.html5QrcodeScanner) {
            await window.html5QrcodeScanner.stop();
            window.html5QrcodeScanner = null;
        }

        document.querySelectorAll('.tab-btn').forEach(btn => {
            btn.classList.remove('active');
        });
        document.querySelector(`.tab-btn[onclick="window.switchVerificationMethod('${method}')"]`).classList.add('active');

        document.querySelectorAll('.verification-method').forEach(el => {
            el.style.display = 'none';
        });
        document.getElementById(`${method}Verification`).style.display = 'block';

        if (method === 'qr') {
            await window.initQRScanner();
            document.querySelector('.scan-btn').style.display = 'block';
            document.querySelector('[onclick="window.stopQRScanner()"]').style.display = 'none';
if (input) {
```

```
            if (pricingType === 'free') {
                input.value = '0';
                input.readOnly = true;
            } else {
                input.value = '';
                input.readOnly = false;
            }
        }
    });
}
        let currentFilter = 'all';

        // Function to generate unique event ID
        function generateEventId() {
            const timestamp = new Date().getTime().toString(36);
            const randomStr = Math.random().toString(36).substring(2, 7);
            return `EVT-${timestamp}-${randomStr}`.toUpperCase();
        }

        // Load all events initially
        async function loadFilteredEvents() {
            try {
                const eventsSnapshot = await getDocs(collection(db, 'events'));
                allEvents = [];

                eventsSnapshot.forEach(doc => {
                    allEvents.push({
                        id: doc.id,
                        ...doc.data()
                    });
                });

                applyFilters('all');
            } catch (error) {
                console.error("Error loading events:", error);
                document.getElementById('filteredEventsList').innerHTML = `
                    <div class="error-message">
                        Error loading events. Please try again.
                        <button onclick="loadFilteredEvents()" class="retry-btn">Retry</button>
                    </div>
                `;
            }
        }

        // Apply filters to events
        function applyFilters(filterType = currentFilter) {
            currentFilter = filterType;

            document.querySelectorAll('.filter-btn').forEach(btn => {
                btn.classList.remove('active');
                if (btn.textContent.toLowerCase().includes(filterType.toLowerCase())) {
                    btn.classList.add('active');
                }
            });

            const searchTerm = document.getElementById('eventSearchInput').value.toLowerCase();
            const dateFilter = document.getElementById('eventDateFilter').value;
            const now = new Date();
            const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());

            let filteredEvents = allEvents.filter(event => {
                const eventDate = new Date(event.date);
                const eventName = (event.name || '').toLowerCase();
                const eventId = (event.eventId || '').toLowerCase();

                if (searchTerm && !eventName.includes(searchTerm) && !eventId.includes(searchTerm)) {
                    return false;
                }

                if (dateFilter) {
                    const filterDate = new Date(dateFilter);
                    const eventDateOnly = new Date(eventDate.getFullYear(), eventDate.getMonth(), eventDate.getDate());
                    if (eventDateOnly.getTime() !== filterDate.getTime()) {
                        return false;
                    }
                }

                switch (filterType) {
```

```
                case 'upcoming':
                    return eventDate > now;
                case 'today':
                    return eventDate.getDate() === today.getDate() &&
                        eventDate.getMonth() === today.getMonth() &&
                        eventDate.getFullYear() === today.getFullYear();
                case 'past':
                    return eventDate < now;
                default:
                    return true;
            }
        });

        filteredEvents.sort((a, b) => new Date(b.date) - new Date(a.date));

        displayFilteredEvents(filteredEvents);
    }

    // Display filtered events
    function displayFilteredEvents(events) {
        const container = document.getElementById('filteredEventsList');

        if (events.length === 0) {
            container.innerHTML = `
                <div class="no-events-message">
                    <p>No events found matching your criteria.</p>
                </div>
            `;
            return;
        }

        let html = `
            <div class="events-grid">
                ${events.map(event => `
                    <div class="event-card">
                        <div class="event-header">
                            <div>
                                <h3>${event.name || 'Unnamed Event'}</h3>
                                <div class="event-id">ID: ${event.eventId || 'No ID'}</div>
                            </div>
                            ${getEventStatusBadge(event)}
                        </div>
                        <div class="event-details">
                            <p><strong>Date:</strong> ${new Date(event.date).toLocaleString()}</p>
                            <p><strong>Location:</strong> ${event.place || 'Not specified'}</p>
                            <p><strong>Venue:</strong> ${event.venue || 'Not specified'}</p>
                            <p><strong>Price:</strong> ₹${(event.price || 0).toFixed(2)}</p>
                            <p><strong>Available Tickets:</strong> ${event.availableTickets || 0}/${event.totalTickets || 0}</p>
                        </div>
                        <div class="event-actions">
                            <button onclick="handleEditClick('${event.id}', ${JSON.stringify(event).replace(/"/g, '&quot;')})" class="edit-
btn">Edit</button>
                            <button onclick="deleteEvent('${event.id}')" class="delete-btn">Delete</button>
                        </div>
                    </div>
                `).join('')}
            </div>
        `;

        container.innerHTML = html;
    }

    // Get event status badge
    function getEventStatusBadge(event) {
        const eventDate = new Date(event.date);
        const now = new Date();

        if (eventDate < now) {
            return '<span class="status-badge past">Past</span>';
        } else if (event.availableTickets <= 0) {
            return '<span class="status-badge sold-out">Sold Out</span>';
        } else {
            return '<span class="status-badge upcoming">Upcoming</span>';
        }
    }

    // Make functions globally available
    window.generateEventId = generateEventId;
```

```
window.loadFilteredEvents = loadFilteredEvents;
window.applyFilters = applyFilters;
window.showSection = showSection;

function handleEventTypeChange() {
    const eventType = document.getElementById('eventType').value;
    const onlineDetails = document.getElementById('onlineEventDetails');
    const venueDetails = document.getElementById('venueEventDetails');
    const ticketingSection = document.getElementById('ticketingSection');
    const standardTicketsSection = document.getElementById('standardTicketsSection');
    const hybridTicketsSection = document.getElementById('hybridTicketsSection');
    const priceField = document.getElementById('eventPrice').parentElement;

    onlineDetails.style.display = 'none';
    venueDetails.style.display = 'none';
    ticketingSection.style.display = 'none';
    standardTicketsSection.style.display = 'none';
    hybridTicketsSection.style.display = 'none';
    priceField.style.display = 'block';

    switch (eventType) {
        case 'online':
            onlineDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'venue':
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            standardTicketsSection.style.display = 'block';
            break;
        case 'hybrid':
            onlineDetails.style.display = 'block';
            venueDetails.style.display = 'block';
            ticketingSection.style.display = 'block';
            hybridTicketsSection.style.display = 'block';
            priceField.style.display = 'none';
            break;
    }
}

// Add validation for dates
function validateEventDates() {
    const eventDate = new Date(document.getElementById('eventDate').value);
    const bookingStartDate = new Date(document.getElementById('bookingStartDate').value);
    const bookingDeadline = new Date(document.getElementById('bookingDeadline').value);
    const now = new Date();

    document.getElementById('eventDate').min = now.toISOString().slice(0, 16);

    if (eventDate < now) {
        alert('Event date must be in the future');
        return false;
    }

    if (bookingStartDate > eventDate) {
        alert('Booking start date must be before event date');
        return false;
    }

    if (bookingDeadline > eventDate) {
        alert('Booking deadline must be before or same as event date');
        return false;
    }

    if (bookingDeadline < bookingStartDate) {
        alert('Booking deadline must be after booking start date');
        return false;
    }

    return true;
}

// Add validation for tickets
function validateTickets() {

        }
    } catch (error) {
```

```
        console.error("Error switching verification method:", error);
        alert("Error: " + error.message);
    }
};

if (typeof Html5QrCode === 'undefined') {
    console.error('HTML5QrCode library not loaded. Please check the script inclusion.');
}

function toggleEventsView(view) {
    document.querySelectorAll('.toggle-btn').forEach(btn => {
        btn.classList.remove('active');
    });
    document.querySelector(`.toggle-btn[onclick*="${view}"]`).classList.add('active');

    const createSection = document.getElementById('createEventSection');
    const viewSection = document.getElementById('viewEventSection');

    if (view === 'create') {
        createSection.style.display = 'block';
        viewSection.style.display = 'none';
    } else {
        createSection.style.display = 'none';
        viewSection.style.display = 'block';
        loadEvents();
    }
}

function filterEvents() {
    const searchTerm = document.getElementById('eventSearchInput').value.toLowerCase();
    const typeFilter = document.getElementById('eventTypeFilter').value;
    const statusFilter = document.getElementById('eventStatusFilter').value;
    const eventList = document.getElementById('eventList');
    const eventCards = eventList.querySelectorAll('.event-card');
    const now = new Date();

    eventCards.forEach(card => {
        let showCard = true;

        const eventName = card.querySelector('h3').textContent.toLowerCase();
        const eventId = card.querySelector('.event-id').textContent.toLowerCase();
        const eventType = card.querySelector('.event-type-badge')?.textContent.toLowerCase() || '';
        const eventDateText = card.querySelector('.event-meta p:first-child')?.textContent;
        const eventDate = eventDateText ? new Date(eventDateText.split(': ')[1]) : now;

        if (searchTerm && !eventName.includes(searchTerm) && !eventId.includes(searchTerm)) {
            showCard = false;
        }

        if (typeFilter && !eventType.includes(typeFilter.toLowerCase())) {
            showCard = false;
        }

        if (statusFilter) {
            const eventTime = eventDate.getTime();
            const nowTime = now.getTime();
            const oneDayMs = 24 * 60 * 60 * 1000;

            switch (statusFilter) {
                case 'upcoming':
                    if (eventTime <= nowTime) showCard = false;
                    break;
                case 'ongoing':
                    if (eventTime < nowTime || eventTime > (nowTime + oneDayMs)) showCard = false;
                    break;
                case 'completed':
                    if (eventTime > nowTime) showCard = false;
                    break;
            }
        }

        card.style.display = showCard ? '' : 'none';
    });
}

// Make functions globally available
window.toggleEventsView = toggleEventsView;
window.filterEvents = filterEvents;
```

```
// Add this function for PNG download
function downloadTicketPNG(ticketId) {
    try {
        const ticketCard = document.getElementById('ticketCard');
        if (!ticketCard) {
            throw new Error('Ticket card element not found');
        }

        html2canvas(ticketCard).then(canvas => {
            const link = document.createElement('a');
            link.download = `ticket-${ticketId}.png`;
            link.href = canvas.toDataURL('image/png');

            document.body.appendChild(link);
            link.click();
            document.body.removeChild(link);
        });

    } catch (error) {
        console.error("Error downloading ticket:", error);
        alert("Error downloading ticket: " + error.message);
    }
}

// Update the button in showTicketDetails function
// Replace the download button HTML with this:
`<div class="admin-ticket-actions">
    <button onclick="downloadTicketPNG('${ticketId}')" class="admin-download-btn">
        <i class="fas fa-download"></i> Download Ticket (PNG)
    </button>
</div>`

// Update the verification function
async function verifyTicket(ticketId) {
    try {
        // Query tickets collection using ticketIdentifier
        const ticketsRef = collection(db, 'tickets');
        const q = query(ticketsRef, where('ticketIdentifier', '==', ticketId));
        const querySnapshot = await getDocs(q);

        if (querySnapshot.empty) {
            return {
                status: 'invalid',
                message: 'Invalid Ticket',
                ticket: null,
                event: null
            };
        }

        const ticketDoc = querySnapshot.docs[0];
        const ticket = ticketDoc.data();

        // Get event details
        const eventDoc = await getDoc(doc(db, 'events', ticket.eventId));
        if (!eventDoc.exists()) {
            return {
                status: 'invalid',
                message: 'Event not found',
                ticket,
                event: null
            };
        }

        const event = eventDoc.data();
        const now = new Date();
        const eventDate = new Date(event.date);

        if (ticket.used) {
            return {
                status: 'used',
                message: 'Ticket already used',
                ticket,
                event
            };
        }

        if (eventDate < now) {
```

```
            return {
                status: 'expired',
                message: 'Event has ended',
                ticket,
                event
            };
        }

        return {
            status: 'valid',
            message: 'Valid Ticket',
            ticket,
            event
        };
    } catch (error) {
        console.error("Error verifying ticket:", error);
        return {
            status: 'error',
            message: 'Error verifying ticket',
            ticket: null,
            event: null
        };
    }
}

// Update the mark ticket as used function
async function markTicketAsUsed() {
    try {
        const ticketId = document.querySelector('.ticket-details p:first-child strong').nextSibling.textContent.trim();

        // Query to find the ticket document using ticketIdentifier
        const ticketsRef = collection(db, 'tickets');
        const q = query(ticketsRef, where('ticketIdentifier', '==', ticketId));
        const querySnapshot = await getDocs(q);

        if (!querySnapshot.empty) {
            const ticketDoc = querySnapshot.docs[0];
            await updateDoc(ticketDoc.ref, {
                used: true,
                usedDate: new Date().toISOString()
            });

            alert('Ticket marked as used successfully!');
            document.getElementById('verificationResult').style.display = 'none';
        } else {
            throw new Error('Ticket not found');
        }
    } catch (error) {
        console.error("Error marking ticket as used:", error);
        alert('Error marking ticket as used: ' + error.message);
    }
}

// Add these functions to handle image management
function handleImageUpload(event) {
    const files = event.target.files;
    const imageGrid = document.getElementById('imageGrid');

    Array.from(files).forEach(file => {
        const reader = new FileReader();
        reader.onload = function(e) {
            const imageItem = document.createElement('div');
            imageItem.className = 'image-item';
            imageItem.innerHTML = `
                <img src="${e.target.result}" alt="Event image">
                <div class="image-actions">
                    <button class="image-action-btn" onclick="moveImage(this, 'up')" title="Move Up">
                        <i class="fas fa-arrow-up"></i>
                    </button>
                    <button class="image-action-btn" onclick="moveImage(this, 'down')" title="Move Down">
                        <i class="fas fa-arrow-down"></i>
                    </button>
                    <button class="image-action-btn delete" onclick="deleteImage(this)" title="Delete">
                        <i class="fas fa-trash"></i>
                    </button>
                </div>
            `;
            imageGrid.appendChild(imageItem);
```

```
            };
            reader.readAsDataURL(file);
        });
    }

    function moveImage(button, direction) {
        const imageItem = button.closest('.image-item');
        const grid = imageItem.parentElement;

        if (direction === 'up' && imageItem.previousElementSibling) {
            grid.insertBefore(imageItem, imageItem.previousElementSibling);
        } else if (direction === 'down' && imageItem.nextElementSibling) {
            grid.insertBefore(imageItem.nextElementSibling, imageItem);
        }
    }

    function deleteImage(button) {
        if (confirm('Are you sure you want to delete this image?')) {
            button.closest('.image-item').remove();
        }
    }

    // Add event listener for image upload
    document.getElementById('imageUpload').addEventListener('change', handleImageUpload);
</script>

<!-- Add this modal HTML just before closing body tag -->
<div id="editEventModal" class="modal">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2>Edit Event</h2>
        <div class="form-group">
            <label>Event Name</label>
            <input type="text" id="editEventName" required>
        </div>
        <div class="form-group">
            <label>Description</label>
            <textarea id="editEventDescription" required></textarea>
        </div>
        <div class="form-group">
            <label>Event Date</label>
            <input type="datetime-local" id="editEventDate" required>
        </div>
        <div class="form-group">
            <label>Booking Start Date</label>
            <input type="datetime-local" id="editBookingStartDate" required>
        </div>
        <div class="form-group">
            <label>Booking Deadline</label>
            <input type="datetime-local" id="editBookingDeadline" required>
        </div>
        <div class="form-group">
            <label>Price (₹)</label>
            <input type="number" id="editEventPrice" step="0.01" required>
        </div>
        <div class="form-group">
            <label>Total Tickets</label>
            <input type="number" id="editEventTickets" required min="1">
        </div>
        <div id="editImageUrlsContainer">
            <!-- Image inputs will be added here -->
        </div>
        <div class="form-group" style="display: flex; gap: 10px;">
            <button type="button" onclick="updateEvent()" class="edit-btn">Update Event</button>
            <button type="button" onclick="cancelEdit()" class="cancel-btn" style="display: none;">Cancel Edit</button>
        </div>
    </div>
</div>

<script>
auth.onAuthStateChanged((user) => {
    if (user) {
        console.log('User is signed in:', user.email);
        document.getElementById('userEmail').textContent = user.email;
    } else {
        console.log('User is signed out');
        window.location.href = 'login.html';
    }
```

```
    });
    </script>

    <script>
    async function loadRecentEvents() {
        try {
            const eventsRef = collection(db, 'events');
            const recentEventsQuery = query(eventsRef, orderBy('createdAt', 'desc'), limit(5));
            const recentEventsSnapshot = await getDocs(recentEventsQuery);

            const recentEventsList = document.getElementById('recentEventsList');
            recentEventsList.innerHTML = '';

            for (const eventDoc of recentEventsSnapshot.docs) {
                const event = eventDoc.data();
                const eventDate = new Date(event.date).toLocaleDateString();
                const eventStatus = getEventStatus(event);

                const eventItem = document.createElement('div');
                eventItem.className = 'recent-item';
                eventItem.innerHTML = `
                    <div class="recent-item-title">${event.name}</div>
                    <div class="recent-item-details">
                        <span>${eventDate}</span>
                        <span>${event.eventType}</span>
                        <span>${eventStatus}</span>
                    </div>
                `;
                recentEventsList.appendChild(eventItem);
            }
        } catch (error) {
            console.error("Error loading recent events:", error);
        }
    }

    // Make sure these functions are globally available
    window.loadRecentEvents = loadRecentEvents;
    window.getEventStatus = getEventStatus;
    </script>
<script src="https://unpkg.com/html5-qrcode@2.3.8/html5-qrcode.min.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/1.4.1/html2canvas.min.js"></script>
<script src="https://html2canvas.hertzen.com/dist/html2canvas.min.js"></script>
<script type="module">
    import { initializeApp } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-app.js";
    import { getFirestore } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-firestore.js";
    import { getAuth } from "https://www.gstatic.com/firebasejs/10.8.0/firebase-auth.js";

    const firebaseConfig = {
        apiKey: "AIzaSyCg7vwuwfN8oWSMagExshHCtMHnxzc7pH0",
        authDomain: "event-ticket-fc753.firebaseapp.com",
        projectId: "event-ticket-fc753",
        storageBucket: "event-ticket-fc753.firebasestorage.app",
        messagingSenderId: "216313948465",
        appId: "1:216313948465:web:4c4b8eb9fbb4257fe6e810",
        measurementId: "G-SGBZK0BYVL"
    };

    // Initialize Firebase
    const app = initializeApp(firebaseConfig);
    window.db = getFirestore(app);
    window.auth = getAuth(app);

    // Define the function
</script>

<script>
    // Add this after Firebase initialization
    document.addEventListener('DOMContentLoaded', function() {
    const logoutBtn = document.getElementById('logoutBtn');
    if (logoutBtn) {
        logoutBtn.addEventListener('click', async function() {
            try {
                // First sign out from Firebase
                await auth.signOut();
```

```
            // Clear all storage
            localStorage.clear();
            sessionStorage.clear();

            // Get the base URL for GitHub Pages
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';

            // Force redirect to index page with correct path
            window.location.replace(`${baseUrl}/index.html`);

            // Prevent back navigation
            window.history.pushState(null, '', `${baseUrl}/index.html`);

        } catch (error) {
            console.error("Logout error:", error);
            // Force redirect even if there's an error
            const baseUrl = window.location.pathname.includes('/Event-Ticket-Management') ? '/Event-Ticket-Management' : '';
            window.location.replace(`${baseUrl}/index.html`);
        }
    });
  }
});
</script>

<!-- Add reference to external JavaScript file -->
<script type="module" src="scripts/admin.js"></script>
  <!-- Loader iframe -->

  <!-- Add this function to handle scheduler sections -->
  <script>
  function toggleScheduleSection(type) {
    const section = document.getElementById(`${type}ScheduleSection`);
    const checkbox = document.getElementById(`enable${type.charAt(0).toUpperCase() + type.slice(1)}Schedule`);

    if (checkbox.checked) {
      section.style.display = 'block';
      // Set minimum date to today for publish date
      if (type === 'publish') {
        const today = new Date();
        today.setMinutes(today.getMinutes() - today.getTimezoneOffset());
        document.getElementById('publishDate').min = today.toISOString().slice(0, 16);
      }
      // Set minimum date to event date for hide date
      if (type === 'hide') {
        const eventDate = document.getElementById('eventDate').value;
        if (eventDate) {
          document.getElementById('hideDate').min = eventDate;
        }
      }
    } else {
      section.style.display = 'none';
      document.getElementById(`${type}Date`).value = '';
    }
  }

  // Add event listener for event date change
  document.getElementById('eventDate').addEventListener('change', function() {
    const hideDate = document.getElementById('hideDate');
    if (hideDate && this.value) {
      hideDate.min = this.value;
    }
  });

  // Make the function globally available
  window.toggleScheduleSection = toggleScheduleSection;
  </script>

  <script>
  async function handleEventSelect() {
    const eventSelect = document.getElementById('eventSelect');
    const ticketInfoDiv = document.getElementById('ticketInfo');
    const generateBtn = document.getElementById('generateTicketBtn');

    if (!eventSelect.value) {
      ticketInfoDiv.innerHTML = '';
      generateBtn.disabled = true;
      return;
    }
```

```
try {
    // Get event details
    const eventDoc = await getDoc(doc(db, 'events', eventSelect.value));
    if (!eventDoc.exists()) {
        throw new Error('Event not found');
    }

    const event = eventDoc.data();
    const now = new Date();
    const bookingStart = new Date(event.bookingStartDate);
    const bookingEnd = new Date(event.bookingDeadline);

    // Check if booking is allowed
    if (now < bookingStart) {
        ticketInfoDiv.innerHTML = `
            <div class="alert alert-warning">
                <i class="fas fa-clock"></i>
                Booking hasn't started yet. Starts on ${bookingStart.toLocaleDateString()} at ${bookingStart.toLocaleTimeString()}
            </div>
        `;
        generateBtn.disabled = true;
        return;
    }

    if (now > bookingEnd) {
        ticketInfoDiv.innerHTML = `
            <div class="alert alert-error">
                <i class="fas fa-times-circle"></i>
                Booking period has ended
            </div>
        `;
        generateBtn.disabled = true;
        return;
    }

    // Check ticket availability
    let availabilityHTML = '';
    let canGenerate = false;

    if (event.eventType === 'hybrid') {
        const venueAvailable = event.tickets?.venue?.available || 0;
        const onlineAvailable = event.tickets?.online?.available || 0;

        availabilityHTML = `
            <div class="ticket-availability">
                <h4>Ticket Availability:</h4>
                <div class="availability-item ${venueAvailable > 0 ? 'available' : 'sold-out'}">
                    <span>Venue Tickets:</span>
                    <span>${venueAvailable} available</span>
                    <span>Price: ₹${event.tickets?.venue?.price || 0}</span>
                </div>
                <div class="availability-item ${onlineAvailable > 0 ? 'available' : 'sold-out'}">
                    <span>Online Tickets:</span>
                    <span>${onlineAvailable} available</span>
                    <span>Price: ₹${event.tickets?.online?.price || 0}</span>
                </div>
            </div>
        `;
        canGenerate = venueAvailable > 0 || onlineAvailable > 0;
    } else {
        const available = event.availableTickets || 0;
        availabilityHTML = `
            <div class="ticket-availability">
                <h4>Ticket Availability:</h4>
                <div class="availability-item ${available > 0 ? 'available' : 'sold-out'}">
                    <span>Available Tickets:</span>
                    <span>${available}</span>
                    <span>Price: ₹${event.price || 0}</span>
                </div>
            </div>
        `;
        canGenerate = available > 0;
    }

    ticketInfoDiv.innerHTML = `
        <div class="event-info">
            <h3>${event.name}</h3>
```

```
                <p><i class="fas fa-calendar"></i> ${new Date(event.date).toLocaleDateString()}</p>
                ${availabilityHTML}
            </div>
        `;

        generateBtn.disabled = !canGenerate;
        if (!canGenerate) {
            ticketInfoDiv.innerHTML += `
                <div class="alert alert-error">
                    <i class="fas fa-exclamation-circle"></i>
                    No tickets available for this event
                </div>
            `;
        }

    } catch (error) {
        console.error('Error fetching event details:', error);
        ticketInfoDiv.innerHTML = `
            <div class="alert alert-error">
                <i class="fas fa-exclamation-circle"></i>
                Error fetching event details


// Update openEditModal to handle category and pricing type
const originalOpenEditModal = window.openEditModal;
window.openEditModal = function(eventId, eventData) {
    try {
        if (originalOpenEditModal) {
            originalOpenEditModal(eventId, eventData);
        }

        // Set category
        const categorySelect = document.getElementById('eventCategory');
        if (categorySelect) {
            if (eventData.category === 'other') {
                categorySelect.value = 'other';
                document.getElementById('otherCategoryName').value = eventData.customCategory || '';
                document.getElementById('otherCategoryGroup').style.display = 'block';
            } else {
                categorySelect.value = eventData.category || '';
            }
        }

        // Set pricing type
        const pricingTypeSelect = document.getElementById('eventPricingType');
        if (pricingTypeSelect) {
            pricingTypeSelect.value = eventData.pricingType || 'paid';
            handlePricingTypeChange();
        }

    } catch (error) {
        console.error("Error in enhanced openEditModal:", error);
    }
};

// Update clearEventForm to reset new fields
const originalClearEventForm = window.clearEventForm;
window.clearEventForm = function() {
    if (originalClearEventForm) {
        originalClearEventForm();
    }

    // Reset category fields
    const categorySelect = document.getElementById('eventCategory');
    const otherCategoryGroup = document.getElementById('otherCategoryGroup');
    const otherCategoryInput = document.getElementById('otherCategoryName');

    if (categorySelect) categorySelect.value = '';
    if (otherCategoryGroup) otherCategoryGroup.style.display = 'none';
    if (otherCategoryInput) otherCategoryInput.value = '';

    // Reset pricing type
    const pricingTypeSelect = document.getElementById('eventPricingType');
    if (pricingTypeSelect) {
        pricingTypeSelect.value = '';
        handlePricingTypeChange();
    }
};
```

```
    // Make functions globally available
    window.handleEventCategoryChange = handleEventCategoryChange;
    window.handlePricingTypeChange = handlePricingTypeChange;
</script>

<script>
    // Function to generate random 5-digit number
    function generateRandomFiveDigits() {
        return Math.floor(10000 + Math.random() * 90000).toString();
    }

    // Function to get current year
    function getCurrentYear() {
        return new Date().getFullYear().toString();
    }

    // Update event ID generation
    function generateEventId() {
        const randomNum = generateRandomFiveDigits();
        const year = getCurrentYear();
        return `EVT-${randomNum}-${year}`;
    }

    // Update ticket ID generation
    function generateTicketId() {
        const randomNum = generateRandomFiveDigits();
        const year = getCurrentYear();
        return `TIX-${randomNum}-${year}`;
    }

    // Update pass ID generation
    function generatePassId() {
        const randomNum = generateRandomFiveDigits();
        const year = getCurrentYear();
        return `PASS-${randomNum}-${year}`;
    }

    // Make functions globally available
    window.generateEventId = generateEventId;
    window.generateTicketId = generateTicketId;
    window.generatePassId = generatePassId;
</script>



<script>


    // Update the showSection function to load active events when switching to generate ticket section
    const originalShowSection = window.showSection;
    window.showSection = function(sectionId) {
        if (originalShowSection) {
            originalShowSection(sectionId);
        }

        if (sectionId === 'generateTicket') {
            loadActiveEvents();
        }
    };

    // Make the function globally available
    window.loadActiveEvents = loadActiveEvents;
</script>

</body>
</html>
```

6. RESULTS

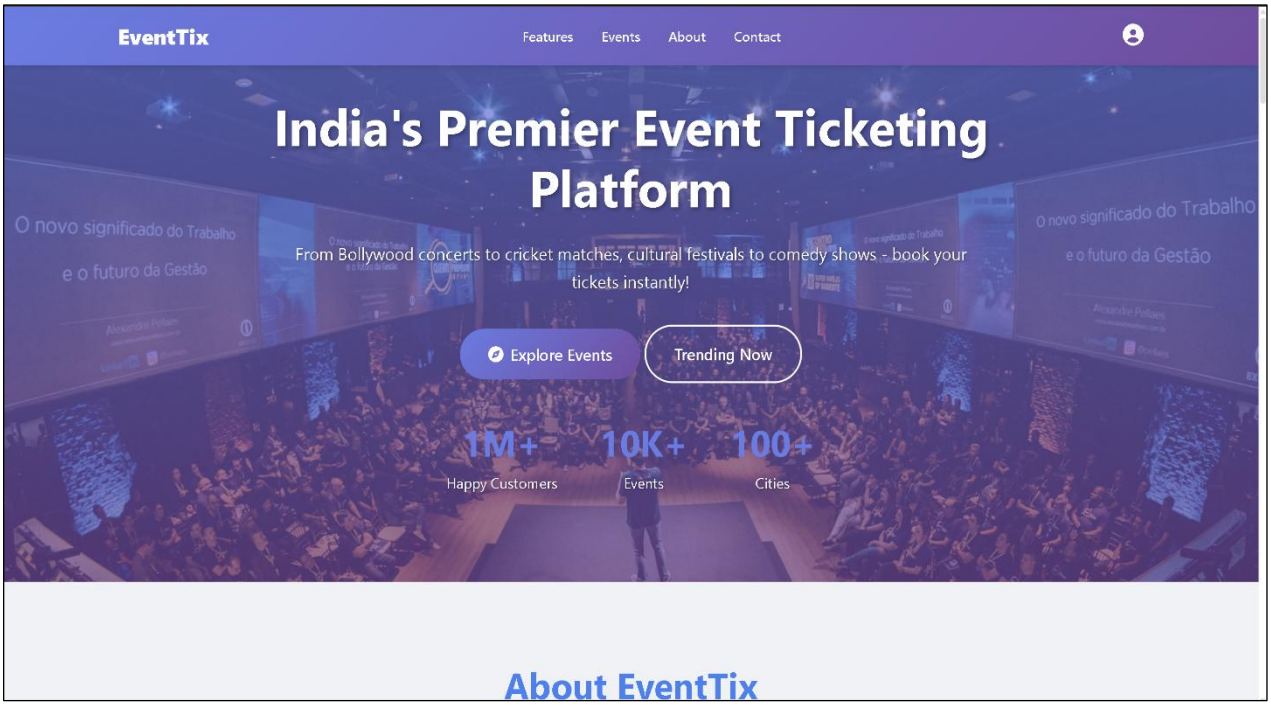


Fig 3.0

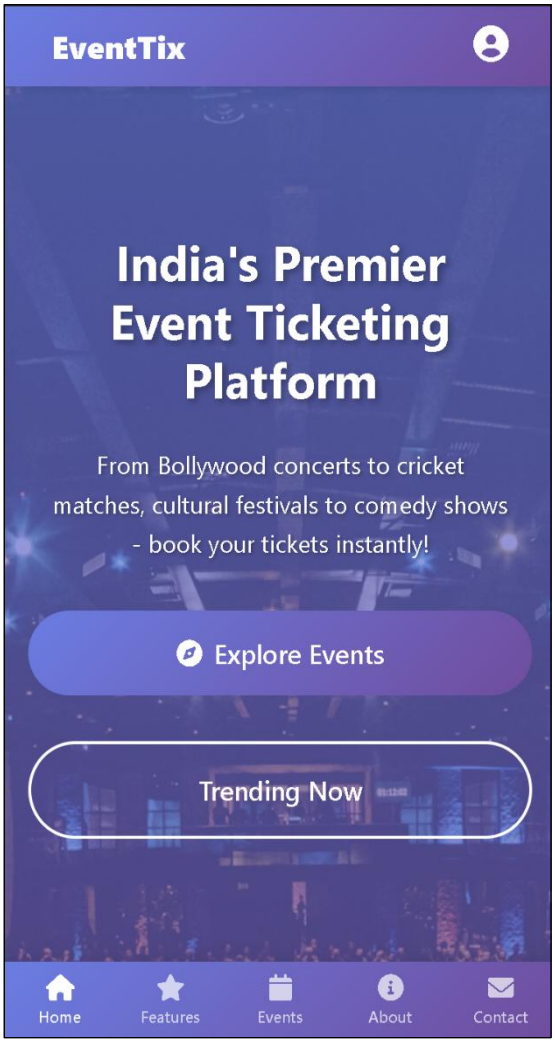
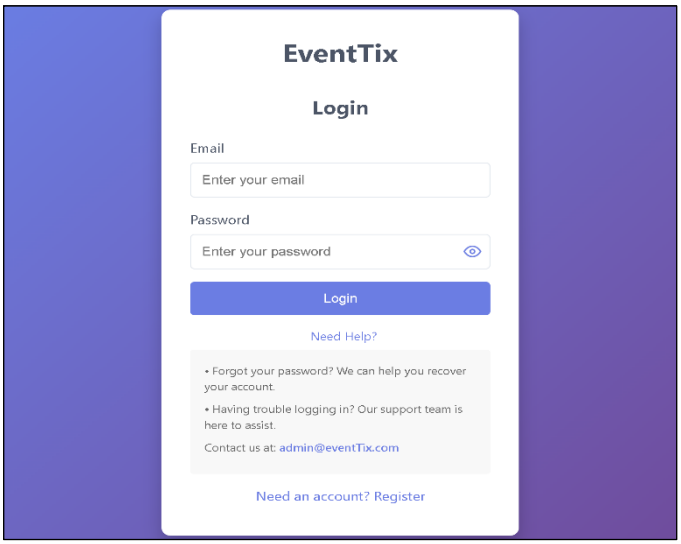


Fig 3.1



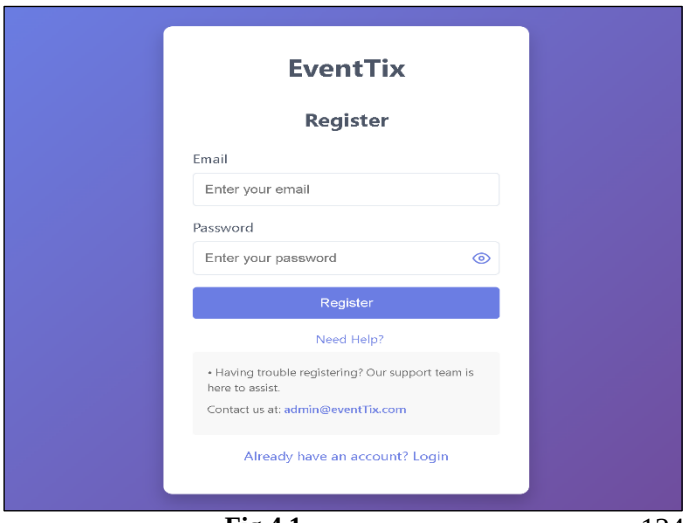


Fig 4.1

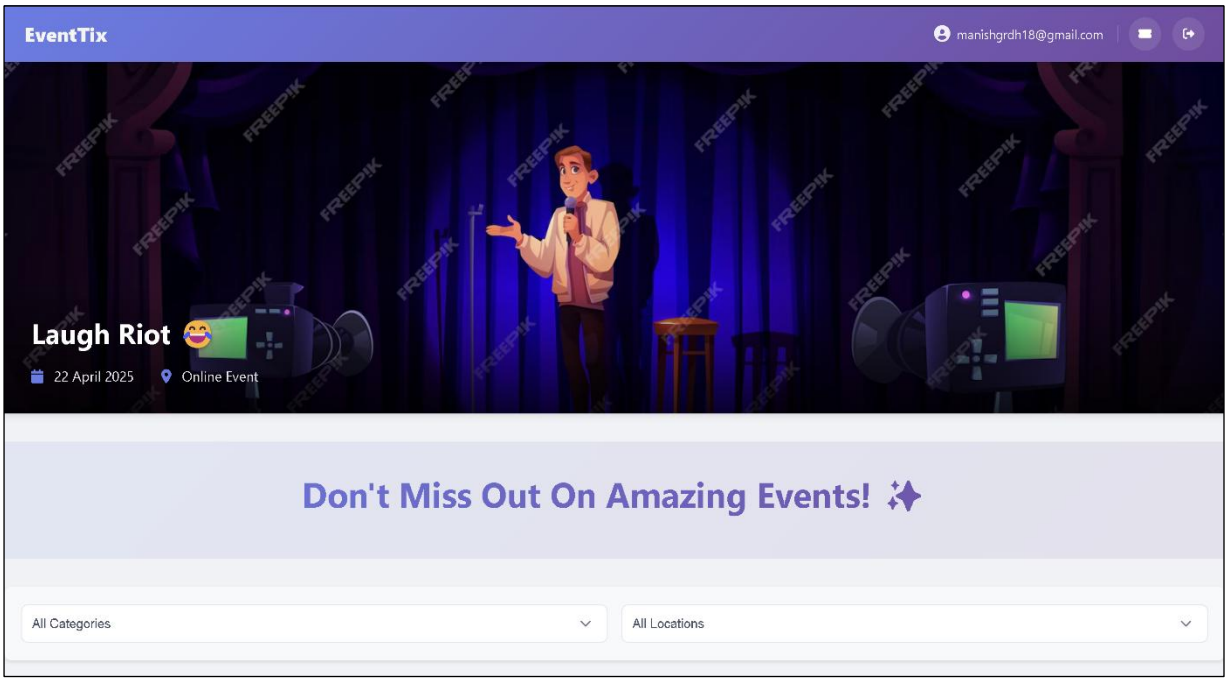


Fig 5.0

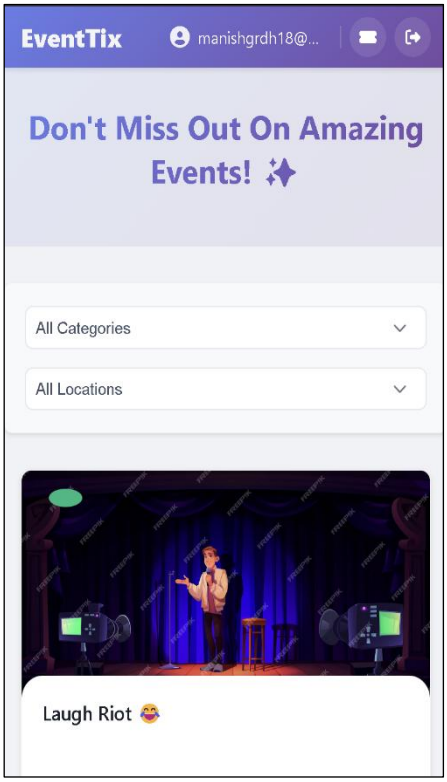


Fig 5.1

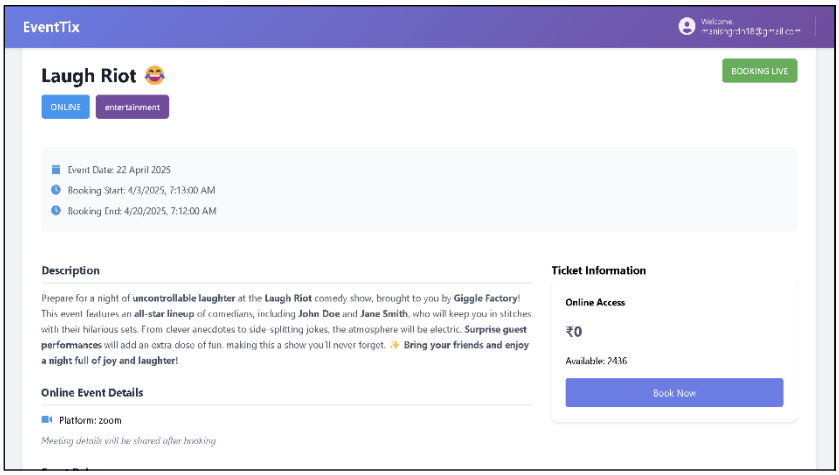


Fig 6.0

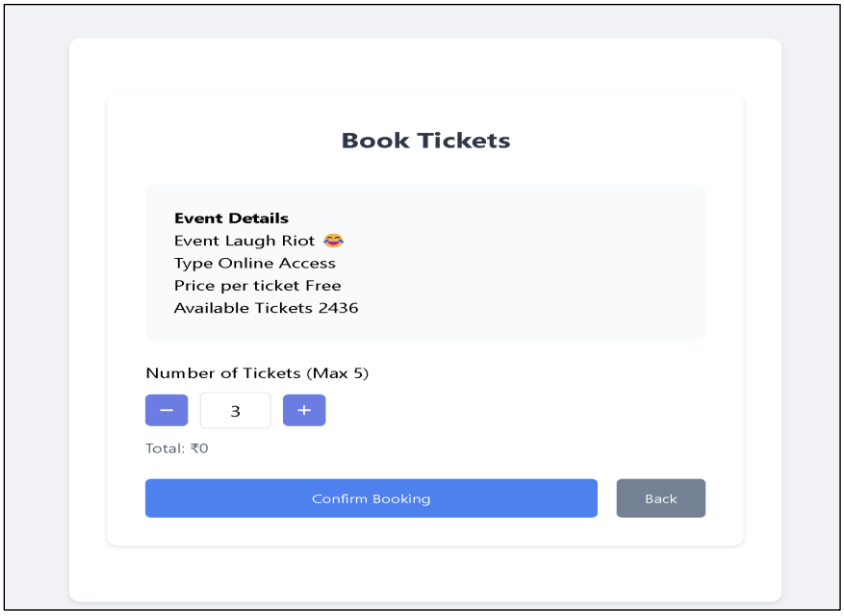


Fig 7.0

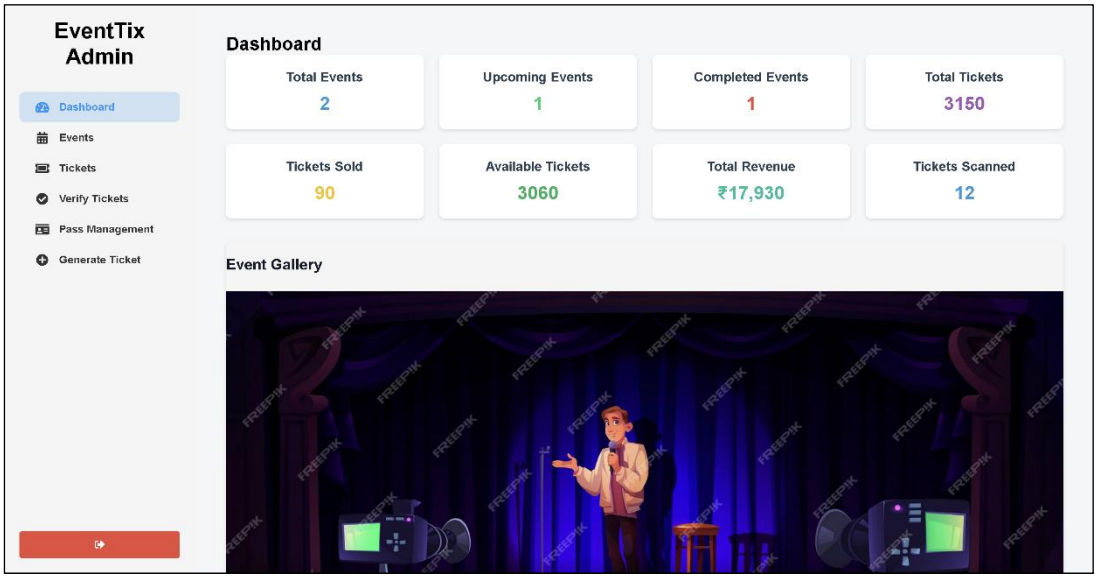


Fig 10.0

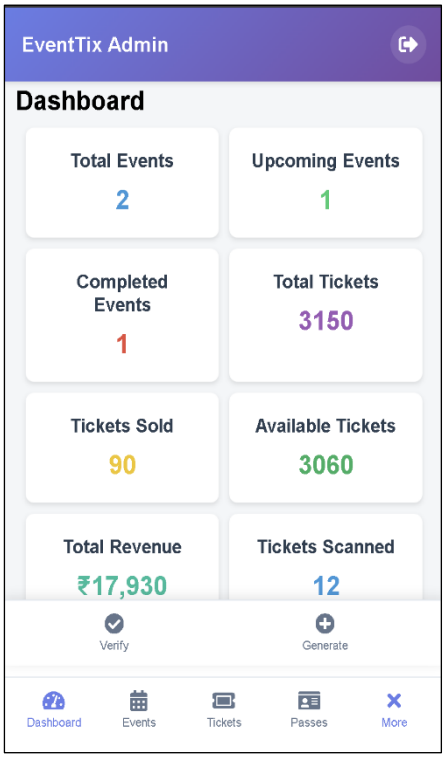


Fig 10.1

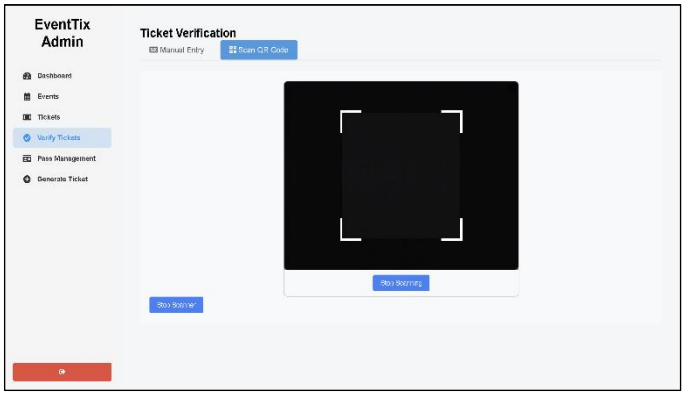


Fig 10.2

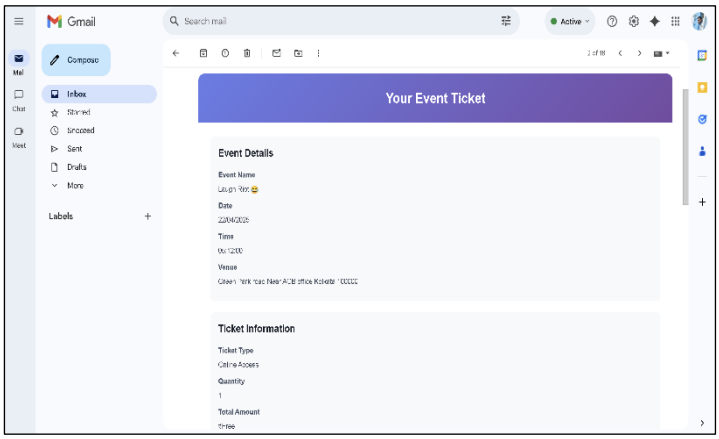


Fig 11.0

7. CONCLUSION

The **EventTix Ticket Management System** is a comprehensive, web-based application designed to streamline the process of event management and ticket booking. Throughout the development process, the system has evolved into a robust solution for both administrators and users, offering essential functionalities such as event creation, real-time ticket booking, and dynamic QR code generation for secure ticket verification.

The use of **Firebase** as the backend for authentication, data storage, and hosting has been a major factor in ensuring the system's scalability and performance. By leveraging **Firebase Firestore**, we were able to implement real-time data synchronization, which allows both users and admins to interact with the system without the need for manual page refreshes. This real-time capability enhances the user experience and makes the event management process more efficient.

On the user side, the system provides an intuitive and easy-to-navigate interface, enabling users to log in, browse available events, and book tickets seamlessly. Upon booking, users instantly receive a unique ticket with a dynamic QR code, ensuring security and preventing ticket duplication. Admins, on the other hand, benefit from a comprehensive dashboard that enables them to create, update, and manage events, track ticket sales, and verify tickets through QR scanning.

One of the key achievements of this project is the integration of **dynamic QR code generation**, which plays a crucial role in preventing ticket fraud. The ticket verification process is simple and reliable, either through QR code scanning or manual ticket ID entry, ensuring the validity of each ticket at the event.

The system's modular design, built using **HTML**, **CSS**, **JavaScript**, and **Firebase**, ensures that it can be easily extended in the future to include additional features such as payment integration, email notifications, and advanced analytics for event tracking. Additionally, the system is scalable, meaning it can accommodate large numbers of users and events without compromising performance.

In conclusion, the **EventTix Ticket Management System** successfully addresses the challenges of traditional ticketing systems by offering a secure, scalable, and user-friendly solution. It simplifies the event management process for organizers and enhances the overall experience for users. The project demonstrates how modern web technologies can be harnessed to create efficient solutions for real-world problems in event management.

8. FUTURE SCOPE

While the **EventTix Ticket Management System** provides a solid foundation for event ticketing and management, there are several enhancements and additional features that could be implemented in the future to further improve the system's functionality and user experience. Below are some potential areas for future development:

1. Payment Gateway Integration

Currently, the system allows users to book tickets, but it lacks a mechanism for processing payments. In the future, integrating a secure payment gateway like **PayPal**, **Stripe**, or **Razorpay** would allow users to complete their transactions directly on the platform. This would enable event organizers to collect payments for tickets and further automate the ticketing process.

2. Real-time Event Analytics and Reporting

One of the significant future enhancements could be the addition of real-time event analytics and reporting features. Admins could gain insights into ticket sales, audience demographics, and trends related to the event's popularity. This would help event organizers to make more informed decisions and optimize future event planning. Analytics could include data on ticket sales per day, the most popular event categories, and geographical distribution of ticket buyers.

3. Mobile Application for Enhanced Accessibility

Currently, the EventTix system is web-based, but the future scope includes the development of a **mobile application** (for both Android and iOS). A mobile app would provide better accessibility, allowing users and admins to manage events, book tickets, and verify tickets on the go. The mobile version could offer features such as push notifications for event reminders and updates.

4. Advanced Ticketing Features

The future could also see the inclusion of **VIP tickets**, **early bird tickets**, and **discount codes**. Users would be able to purchase different types of tickets for the same event, each with different benefits, such as access to exclusive areas or early access to the event. This feature would provide event organizers with greater flexibility in their pricing strategies.

5. Event Collaboration and Partnerships

The platform could be expanded to allow **event collaboration** and **partnerships** between multiple event organizers. This would enable joint events or multi-day conferences, where users can buy tickets for multiple events as part of a bundle, thus expanding the platform's use cases.

6. Social Media Integration

Integration with popular social media platforms like **Facebook**, **Twitter**, and **Instagram** could be introduced. Users could share their booked tickets, upcoming events, or even the events they are attending directly on their social media profiles. This feature would also benefit event organizers by providing free advertising, expanding the reach of their events through social media sharing.

7. Personalized Event Recommendations

By leveraging machine learning or AI, the system could be enhanced to offer **personalized event recommendations** to users based on their past event attendance, preferences, or interests. This would increase user engagement by suggesting events that align with their tastes, encouraging more frequent usage of the platform.

8. User Reviews and Ratings for Events

Users could be allowed to leave reviews and rate events after they attend them. This would provide valuable feedback to event organizers, and other potential attendees could benefit from this information when deciding whether to attend an event. Reviews and ratings could also help maintain the quality and reputation of events listed on the platform.

9. Enhanced Security Features

Although the system currently uses **dynamic QR codes** for ticket verification, further security measures can be added. For example, **two-factor authentication (2FA)** could be implemented for both users and admins to enhance account security. Additionally, the system could use **biometric authentication** (such as fingerprint or face recognition) for ticket verification at the event, adding another layer of security.

10. Integration with Event Marketing Tools

Future integration with **email marketing tools** (e.g., **MailChimp**) and **CRM** systems would allow event organizers to directly communicate with attendees. They could send reminders, newsletters, or promotional materials, further improving the overall user experience and event marketing efforts.

11. Multiple Language Support

To cater to a global audience, the platform could introduce **multi-language support**. This would make the system accessible to a wider demographic, including users who speak languages other than English. Event organizers could also customize the platform to support regional languages, making the system more inclusive and user-friendly.

12. Support for Virtual and Hybrid Events

With the growing trend of virtual and hybrid events, the system could be extended to accommodate online events. Users could attend live-streamed events through the platform, and tickets for these events could also include **virtual passes**. The integration of video streaming tools such as **Zoom** or **YouTube** would allow users to participate remotely.

The **EventTix Ticket Management System** offers a great starting point for modernizing event ticketing, but there is still considerable room for growth. By incorporating features such as payment integration, real-time analytics, mobile support, personalized recommendations, and security enhancements, the system can become even more versatile and user-friendly. As technology continues to evolve, these enhancements will help EventTix keep pace with the changing demands of the event management industry, allowing it to stay competitive in the market.

In conclusion, the future scope of this project is vast, with opportunities for expansion into new domains, including mobile app development, social media integration, and virtual event support. These improvements will enhance the platform’s appeal to a broader range of users and event organizers, ensuring its continued relevance and success in the years to come.

9. LIMITATIONS

While the **EventTix Ticket Management System** offers a robust solution for event management and ticket booking, there are some limitations that were encountered during the development and implementation phases. These limitations could impact the system's functionality in certain use cases and highlight areas where future improvements can be made.

1. Lack of Payment Integration

One of the primary limitations of the system is the absence of payment gateway integration. Although users can browse events and book tickets, the system does not currently allow for secure payment transactions. This prevents the system from fully supporting paid events and limiting the platform's potential for use in commercial event management. The addition of a payment gateway like **PayPal**, **Stripe**, or **Razorpay** is crucial for ensuring secure and efficient payment processing.

2. Scalability Challenges

While the system is built using **Firebase** and is scalable for small to medium-sized events, it may face performance issues when dealing with very large events with thousands of attendees. **Firebase Firestore**, although fast and efficient for real-time operations, can sometimes face limitations in handling extremely high concurrent requests and large volumes of data, especially during peak event times. Implementing advanced database optimization techniques and considering more robust server-side architectures could help overcome this limitation in the future.

3. Limited Event Analytics

The system currently lacks advanced event analytics, such as attendee demographics, sales trends, and audience engagement metrics. Admins can track basic information like the number of tickets sold, but there is no built-in tool to generate comprehensive reports or provide insights into the success of an event. Adding real-time event analytics and reporting features would significantly enhance the platform's functionality for event organizers.

4. No Mobile Application

The current system is web-based, which restricts users and event organizers to accessing the platform only through a browser. In the mobile-first world, a significant portion of users access services via smartphones. The absence of a mobile application means users cannot book tickets, verify tickets, or check event details on the go. The lack of a mobile version limits accessibility and user convenience, which could be addressed by developing a cross-platform mobile application.

5. Single Event Type and Limited Ticket Customization

The system currently supports a single ticket type for each event. There is no option for event organizers to create **VIP**, **early bird**, or **discounted** tickets with different privileges or prices. This limitation restricts the flexibility of event organizers in terms of pricing and ticket offerings. Enabling multiple ticket types for each event, along with customizable ticket features, would provide a more dynamic solution for event organizers.

6. Basic Ticket Verification System

While the **dynamic QR code generation** offers a secure way of verifying tickets, the ticket verification system could be expanded further. The current verification process only supports QR code scanning or manual ticket ID entry. There is no integration with real-time ticket scanning devices or dedicated **ticket validation terminals**, which may be necessary for large-scale events with a high volume of attendees. Future improvements could include integrating scanning devices or mobile apps specifically designed for event check-ins.

7. Lack of User Feedback and Rating System

The system does not currently support a feature for users to leave feedback or rate the events they attend. This lack of a review or rating mechanism makes it harder for event organizers to gather valuable feedback about their events and for potential attendees to make informed decisions. A built-in **rating and review system** could significantly enhance user interaction and allow organizers to improve future events based on attendee feedback.

8. No Support for Virtual and Hybrid Events

The platform currently only supports in-person events, which limits its applicability to events that are conducted online or in a hybrid format (both physical and virtual). The COVID-19 pandemic has demonstrated the need for platforms to support **virtual** and **hybrid events**. The lack of integration with live streaming services or online event platforms makes the system less flexible in catering to a diverse range of event types.

9. Dependence on Internet Connectivity

Since the system is cloud-based and requires internet connectivity, users and admins can only access the platform when they have a stable internet connection. For users with unreliable internet access, this may pose a barrier to booking tickets or managing events. Offering some offline functionality or ensuring that the platform remains available in low-connection areas could improve accessibility, particularly in regions with poor internet infrastructure.

10. Security Concerns with Personal Data

Although the platform uses **Firebase Authentication** and **dynamic QR codes** for user verification and event security, there are still concerns regarding the handling of sensitive user data, such as personal details and payment information (when integrated in the future). Ensuring that the system complies with **data protection regulations** like **GDPR** or **CCPA** is important to safeguard user privacy. Additional measures, such as **data encryption** and **multi-factor authentication (MFA)**, should be considered to ensure the highest level of security for users' personal information.

11. Limited Internationalization Support

The platform currently supports only English, limiting its use in non-English-speaking regions. To appeal to a global audience, the system could benefit from **multi-language support** to cater to users in different countries. Expanding the platform’s language options would significantly increase its global accessibility and widen the user base.

While the **EventTix Ticket Management System** serves as a comprehensive and functional solution for managing events and ticketing, it does have several limitations that need to be addressed in the future. These include the lack of payment gateway integration, scalability challenges, mobile accessibility, and a more robust ticketing and verification system. By addressing these limitations, the system can evolve into a more versatile platform that better meets the needs of users and event organizers.

Despite these limitations, the system offers great potential for future enhancements, and with the right updates and improvements, it can provide an even more seamless and powerful event management solution in the future.

10. ANNEXURE – I: References

1. **Firebase Documentation**. (2025). Retrieved from https://firebase.google.com/docs
2. **MDN Web Docs**. (2025). HTML: HyperText Markup Language. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTML
3. **W3Schools**. (2025). CSS Tutorial. Retrieved from https://www.w3schools.com/css/
4. **JavaScript.info**. (2025). JavaScript Tutorial. Retrieved from https://javascript.info/
5. **QR Code Generator**. (2025). QR Code Documentation. Retrieved from https://www.qr-code-generator.com/
6. **Chaudhary, A., & Sharma, R.** (2022). Event Ticketing Systems: Challenges and Opportunities. *Journal of Digital Systems & Management*, 15(3), 67-85.
7. **Google Developers**. (2025). Firebase Realtime Database: Build rich, collaborative applications. Retrieved from https://firebase.google.com/docs/database
8. **Agarwal, N.**, & **Kumar, P.** (2023). *Web-based Event Management and Ticketing System: A Review*. *International Journal of Computer Applications*, 45(2), 21-35.

11. ANNEXURE – II: Weekly Progress Reports

(All original & signed WPRs to be attached here)

12. ANNEXURE – III: Internship/ Project Certificate

(Copy of internship/ project certificate to be attached here)

YHills
Learning Beyond Expectations

CERTIFICATE
OF COMPLETION

CYBER SECURITY
CERTIFIED

PROUDLY PRESENTED TO

PRANIT KUMAR PATHAK

has successfully completed **Cyber Security** course from 01/03/2024 to 30/04/2024 at YHills.

YHILLS EDUTECH PVT. LTD.

22/05/2024

DATE

AMAN KUMAR, CO-FOUNDER

Certificate ID : YHIcc-5008520

Issued Date: *01-08-2024*

CIN No- U85500HR2023PTC115118

CERTIFICATE
OF INTERNSHIP

UNIFIED MENTOR
YOUR SKILL, SUCCESS & JOURNEY

Manish Kumar Sharma

For successfully completing two months internship as Web Development Intern at Unified Mentor Pvt Ltd. Dated from 01-06-2024 to 01-08-2024
During the internship we found him/her consistent & hard-working. We wish them all the best for their future endeavors.

Verify at:

Paras Grover
Paras Grover
Director

CERTIFIED ISO 9001:2015 COMPANY

Sanket Patil
Sanket Patil
Awarded By

AN ISO **9001:2015** Certified Company