Event Management System

Full Table of Contents

Chapter 6 — System Architecture & Project Structure
6.1 Why Django Was Chosen
6.2 High-Level System Architecture
6.3 Django App Structure in EMS
6.4 Data Flow Through the System
6.5 URLs, Views, Templates, and Logic Layer

Chapter 7 — Core Features & User Guide
7.1 Browsing Events
7.2 Exhibition Workflow (Announcement Only)
7.3 Conference Workflow (Free Reservation)
7.4 Concert & Sports Workflow (Paid Ticketing)
7.5 Viewing Booking History
7.6 Organizer Features
7.7 Admin Features
7.8 Example User Journeys

Chapter 8 — AI Integration (Ollama + Llama 3.1)
8.1 Rationale for AI Integration
8.2 AI Architecture
8.3 Data Flow of AI Requests
8.4 AI Commands
8.5 Utility for Different User Groups
8.6 Limitations of Local LLM Integration

Chapter 9 — Security & Privacy Considerations
9.1 Password Protection & Hashing
9.2 Role-Based Access Control
9.3 Payment Data Handling
9.4 Input Validation
9.5 Securing AI Interaction

Chapter 10 — Testing & Evaluation
10.1 Testing Approach
10.2 Functional Tests
10.3 Capacity and Booking Rule Tests
10.4 Payment Workflow Tests
10.5 AI Testing
10.6 Performance Evaluation

Chapter 11 — Limitations & Future Enhancements
11.1 Current Limitations
11.2 Planned Improvements
11.3 Future AI Capabilities
11.4 Potential for Real-World Deployment

Chapter 1 — Introduction

1.1 Background and Motivation

Events are an essential part of any academic institution or organization—whether they are exhibitions, conferences, concerts, or sports events. However, the methodology for communicating event information is often fragmented and inconsistent. Some events are announced through emails, others through physical posters, some via social media, and many through word-of-mouth. This decentralization makes it difficult for users to locate accurate information, recall dates, or discern whether registration or ticket purchase is required.

On the organizer side, the challenges are equally significant. Many organizers continue to rely on spreadsheets or manual lists to track registrations and ticket sales. Issues such as overbooking, missing data, inconsistent event information, and last-minute miscommunication are prevalent. A system that handles multiple event types with distinct requirements (e.g., free reservations versus paid tickets) becomes increasingly complex to manage manually.

To address these challenges, the Event Management System (EMS) was designed and implemented. EMS is a complete, centralized, and intelligent event platform. The primary goal is to streamline the event management process for all stakeholders: attendees, organizers, and administrators.

With EMS, users can browse events, register for conferences, purchase tickets for concerts or sports events, and interact with an AI assistant using natural language. Organizers can publish events and manage bookings, while administrators maintain control over venues, event status, and data integrity.

1.2 Objectives

The key objectives of the EMS project include:

User-Focused Objectives

- Provide a simple and user-friendly interface to browse all events in one central location.
- Clearly differentiate between event types (exhibition, conference, concert, sports).
- Allow users to register or purchase tickets efficiently.
- Enable natural-language interaction through an AI chatbot.

Organizer Objectives

- Allow organizers to create, edit, and publish events.
- Prevent scheduling conflicts and enforce venue capacities.
- Provide tools to manage bookings and view event statistics.
- Enable AI-assisted event description generation.

Admin Objectives

- Offer full oversight of all system data.
- Manage venues, organizers, and users.
- Enforce data integrity and business rules.
- Oversee the event lifecycle (draft to published to cancelled).

Technical Objectives

- Strictly follow the Entity-Relationship Diagram (ERD) during conceptual design.
- Implement the system using Django in a modular structure.
- Support a local AI model (Llama 3.1 via Ollama).
- Ensure the system is compatible with both macOS and Windows platforms.

The overall objective was to build a functioning system—not merely a conceptual design—and document it thoroughly from both technical and user-centered perspectives.

1.3 Scope of the System

The EMS system provides a full-stack solution for managing four distinct event types:

1. **Exhibitions**
    - Announcement only
    - No registration required
    - No limit on attendees
2.
3. **Conferences**
    - Free events
    - Reservation required
    - Capacity-limited
4.

5. **Concerts**
    - Paid ticketed events
    - Capacity-limited
    - Payment required
6.
7. **Sports Games**
    - Paid ticketed events
    - Capacity-limited
    - Payment required
8.

System Features Included:

- Event listing and filtering
- Detailed event pages
- Free conference reservations
- Ticket purchases for paid events
- Booking and payment tracking
- Organizer dashboards
- Admin management tools
- AI chatbot for event search and description generation
- Input validation and business rule enforcement

Features Not Included (Out of Scope):
These features may be added in future iterations but were excluded to maintain project focus:

- Real payment gateway processing
- Email/SMS notifications
- Multi-language support
- Mobile application
- Role-based dashboards beyond the main user categories

1.4 What Makes This Project Unique

While numerous event management tools exist, several factors distinguish this system:

AI Integration with a Local LLM
Instead of relying solely on dropdown filters and menus, users can query the system using natural language:

- "What events are happening today?"
- "Show me concerts with available tickets."
- "Write a short description for my Data Science Conference."
  Because the system utilizes a local Llama 3.1 model via Ollama, it ensures privacy, speed, and independence from cloud APIs.

Strict Alignment with ERD Requirements
The ERD was treated as a definitive blueprint. All entities, attributes, and relationships accurately reflect the conceptual design.

Cleanly Structured Django Project
Each domain of the system is separated into its own application. This modularity ensures the codebase is maintainable and comprehensible.

Cross-Platform Compatibility
The system was specifically engineered for ease of installation and operation on:

- macOS (specifically M1/M2 architectures)
- Windows (with full native support for Ollama)
  This ensures accessibility for a wide range of users.

1.5 How to Use This Documentation

This documentation is written in an accessible style to serve multiple audiences:

- **If you are installing the system:** Follow the installation chapter step-by-step, regardless of your operating system.
- **If you are a user:** Read the User Guide to learn how to browse, book, and interact with the system.
- **If you are an organizer:** Refer to the guide to understand how to create events and manage bookings.
- **If you are a developer or evaluator:** Review the conceptual ERD, physical schema, and architecture chapters to understand design choices.
- **If you are an instructor:** Evaluate the quality, completeness, and coherence of the system's design and documentation.

Chapter 2 — Problem Definition & Requirements

2.1 Core Problems with Existing Event Management

Before designing the Event Management System (EMS), an evaluation was conducted on how events are typically handled in environments such as universities, clubs, and community organizations. Several recurring issues motivated the creation of this system:

1. Event Information Is Scattered
Event announcements often appear across multiple channels:

- Emails
- Posters or flyers
- Group chats
- Verbal announcements
- Social media posts
  Users often do not know where to look, and organizers cannot rely on a single central source of information.

2. Different Event Types Have Different Requirements
Not all events function identically:

- Some require no registration.
- Some require free reservations.
- Some require paid tickets.
  Each event type possesses unique rules, and managing these manually introduces errors.

3. Manual Tracking Leads to Errors
Without a proper system:

- Overbooking is common.
- Duplicate entries occur.
- Payments may not match bookings.
- Critical information is lost.
  Organizers often rely on spreadsheets or paper forms, which are inefficient and unreliable.

4. Lack of Automation
Manual systems fail to provide:

- Automatic capacity enforcement
- Real-time updates
- Structured booking data
- Payment records
- Dashboards or analytics
  This creates administrative burdens that a digital system can easily alleviate.

5. No Natural-Language Interaction
Most event systems rely exclusively on dropdown filters and search bars. Users may not know exactly what to search for, yet they can easily formulate questions such as:

- "Are there any concerts this month?"
- "Is there a conference I can attend for free?"
- "Which events still have available seats?"
  Traditional search interfaces do not support this natural inquiry style.

6. No Unified System for Attendees, Organizers, and Admins
Most existing tools focus on a single user group (e.g., registration-only systems, payment-only systems, or admin-only dashboards). EMS integrates all three perspectives into one platform.

2.2 Target Users

Based on the identified problems, EMS is designed for three main user groups, each with unique needs:

1. Public Users / Attendees
These are individuals who browse and attend events. Their needs include:

- Viewing event details easily.
- Understanding whether an event is free or paid.
- Knowing if registration is required.
- Booking seats for conferences.
- Purchasing tickets for concerts and sports games.
- Asking questions through an AI assistant.
- Viewing past bookings.

2. Organizers
Organizers host events and require a structured, reliable system. Their needs include:

- Creating new events.
- Editing event details.
- Publishing or cancelling events.
- Ensuring no scheduling conflicts occur.
- Setting capacity limits.
- Monitoring reservations and ticket sales.
- Generating polished event descriptions (AI-assisted).

3. Administrators
Admins maintain the system infrastructure. Their needs include:

- Managing users and organizers.
- Creating and updating venues.
- Monitoring all events in the system.
- Handling booking or payment disputes.
- Ensuring data integrity.
- Overseeing cancellations and approvals.

2.3 Functional Requirements

To support all user types, the system must perform several core functions:

1. Event Management

- Create, edit, and delete events.
- Publish or cancel events.
- View event details.
- List all events with filtering options.

2. Booking Management

- Allow free conference reservations.
- Allow ticket purchases for concerts and sports.
- Prevent overbooking.
- Ensure capacity rules are applied.
- Store booking timestamps and statuses.

3. Payment Management

- Record payment amounts.
- Store payment methods (cash, card, online).
- Link payments directly to bookings.
- Calculate total price (ticket quantity × unit price).
- Ensure that paid events cannot have bookings without payments.

4. Venue Management

- Add, edit, and remove venues.
- Ensure correct venue capacity.
- Assign venue types.
- Prevent scheduling conflicts.

5. User & Organizer Accounts

- User registration and login.
- Profile linking (User to Customer or Organizer).
- Organizer dashboards.
- Admin privileges.

6. AI Chatbot Integration

- Answer user questions about events.
- Search events using natural language.
- Suggest events automatically.
- Generate event descriptions for organizers.

7. Administrative Tools

- Access to all events, bookings, and payments.
- Editing rights across the system.
- Oversight of system consistency.

2.4 Non-Functional Requirements

To ensure the system is reliable and usable, several non-functional requirements must be met:

Usability

- Simple and intuitive web interface.
- Clear navigation flows.
- Easy-to-understand booking pages.

Performance

- Quick response times for viewing events.
- Efficient database operations.
- Fast AI responses when Ollama is running locally.

Security

- Password hashing.
- Role-based access control.
- Masking sensitive card details.
- Safe handling of AI inputs and outputs.

Portability
The system should run easily on both:

- macOS (especially M1/M2 chips).
- Windows (with native Ollama support).

Maintainability

- Clean separation into Django apps.
- Consistent naming and structure.
- Well-documented code and architecture.

2.5 Business Rules

These business rules were essential to the correctness of the EMS:

1. Event Types Behave Differently

- Exhibition: No booking required.
- Conference: Free booking required.
- Concert/Sports: Paid booking required.

2. Capacity Enforcement

- Total booked seats must be less than or equal to event capacity.
- Event capacity must be less than or equal to venue capacity.
- Bookings cannot be created if the event is full.

3. Payment Rules

- Concert and Sports bookings must have an associated payment.
- Unit price for conferences must be zero.
- Payment amount must equal the total booking price.

4. Booking Rules

- A booking must reference a valid event.
- Payments must reference a valid booking.
- An event must be published to accept bookings.

5. Role-Based Permissions

- Admins can manage all aspects of the system.
- Organizers can manage their own events.
- Users can only book events and view their own history.

6. AI Usage Rules

- AI cannot change database data directly.
- AI can only provide suggestions or answers.
- User inputs to AI must be sanitized.

Chapter 3 — System Setup & Installation (macOS + Windows)

3.1 Overview

This chapter explains how to install and run the Event Management System (EMS) on both macOS (specifically Apple Silicon) and Windows. The setup process is designed to be straightforward, allowing users with minimal technical experience to run the system smoothly.

The setup includes:

- Installing Python.
- Creating a virtual environment.
- Installing Django and required dependencies.
- Installing Ollama for AI.
- Pulling the Llama 3.1 model.
- Running database migrations.
- Starting the development server.

3.2 Hardware & Software Requirements

Before installing, ensure your machine meets the following requirements:

Hardware Requirements

- Modern Mac or PC.
- 8 GB RAM minimum (16 GB recommended for LLM performance).
- 5–8 GB free disk space.
- Increasing RAM aids in faster AI inference.

Software Requirements

- Python 3.10 or higher (Python 3.11 used in development).
- Django 4.2 (installed via requirements.txt).
- SQLite (included with Python).
- Ollama installed locally.
- Llama 3.1 model.
- pip (Python package manager).
- A modern web browser.

Python Dependencies
These packages are installed automatically via pip install -r requirements.txt. The requirements file includes:

- asgiref
- django==4.2.26
- sqlparse
- requests
- certifi
- charset-normalizer

- idna
- urllib3

These libraries support Django, HTTP requests to the AI, and database utilities.

3.3 Installing EMS on macOS (Apple Silicon)

Step 1 — Check Python Installation
Run:

<*Bash*>
downloadcontent_copy
expand_less
  python3 --version

If your version is 3.10+, you may proceed. If not, install the latest Python from the official website.

Step 2 — Navigate to the Project Folder
Assuming your project structure is EMS_PROJECT/backend/, move into the backend folder:

<*Bash*>
downloadcontent_copy
expand_less
  cd EMS_PROJECT/backend

Step 3 — Create and Activate a Virtual Environment
This keeps dependencies isolated.

<*Bash*>
downloadcontent_copy
expand_less
  python3 -m venv venv
source venv/bin/activate

You should now see (venv) before your terminal prompt.

Step 4 — Install All Dependencies

<*Bash*>
downloadcontent_copy
expand_less
  pip install -r requirements.txt

Step 5 — Apply Database Migrations
This creates all required tables (Customer, Organizer, Venue, Event, Booking, Payment, Admin) plus Django implementation tables.

<Bash>
downloadcontent_copy
expand_less
```
  python manage.py migrate
```

Optionally, create an admin user:

<Bash>
downloadcontent_copy
expand_less
```
  python manage.py createsuperuser
```

Step 6 — Install Ollama (AI Runtime)
Download it from the official Ollama website. Once installed, pull the Llama 3.1 model:

<Bash>
downloadcontent_copy
expand_less
```
  ollama pull llama3.1
```

This downloads the model to your machine.

Step 7 — Run the Django Development Server

<Bash>
downloadcontent_copy
expand_less
```
  python manage.py runserver
```

Open http://127.0.0.1:8000/ to view the EMS homepage.

3.4 Installing EMS on Windows (with Native Ollama Support)

Ollama now supports Windows natively, eliminating the need for WSL.

Step 1 — Install Python
Download the installer from the official Python website. **Important:** Check the box Add Python to PATH.
Verify installation:

<*Bash*>
downloadcontent_copy
expand_less
  python --version

Step 2 — Navigate to the Backend Folder

<*Bash*>
downloadcontent_copy
expand_less
  cd EMS_PROJECT\backend

Step 3 — Create and Activate a Virtual Environment

<*Bash*>
downloadcontent_copy
expand_less
  python -m venv venv
venv\Scripts\activate

Your command prompt will now show (venv).

Step 4 — Install Dependencies

<*Bash*>
downloadcontent_copy
expand_less
  pip install -r requirements.txt

Step 5 — Apply Migrations

<*Bash*>
downloadcontent_copy
expand_less
  python manage.py migrate
python manage.py createsuperuser

Step 6 — Install Ollama (Native Windows)
Download the Windows installer from the Ollama website.
Verify installation:

<*Bash*>
downloadcontent_copy
expand_less
  ollama --version

Pull the Llama model:

<*Bash*>
downloadcontent_copy
expand_less
  ollama pull llama3.1

Step 7 — Start the Django Server

<*Bash*>
downloadcontent_copy
expand_less
  python manage.py runserver

Open http://127.0.0.1:8000/.

3.5 Initializing the Database

Your EMS uses SQLite; no separate database engine installation is required. To initialize:

<*Bash*>
downloadcontent_copy
expand_less
  python manage.py migrate

Django automatically creates tables for:

- admin
- accounts_userprofile
- ai_pendingaiaction
- authentication tables
- All EMS entities

You can inspect the database using python manage.py dbshell or a GUI like DB Browser for SQLite.

3.6 Running Django

Once migrations are applied, start the server:

<Bash>
downloadcontent_copy
expand_less
  python manage.py runserver

Visit http://127.0.0.1:8000/. You should see the Homepage, Event listings, Menus, and the "Chat with AI Assistant" link.

3.7 Testing the Installation

Test the following to confirm correct installation:

1. **Homepage loads.**
2. **Events page loads.**
3. **Admin panel loads** at /admin.
4. **AI connection test:**
   Run: curl http://localhost:11434/api/chat
   If you receive a JSON response, your LLM is running.
5. **Log into admin.**
6. **Create a venue.**
7. **Create an event.**

If all tests pass, the EMS environment is fully active.

Chapter 4 — Conceptual Database Design

4.1 Overview of the Design Process

Before implementing the EMS in Django, the conceptual database structure was designed to ensure data flow, business rules, and entity relationships were properly understood. This was achieved through an Entity-Relationship Diagram (ERD), which acts as the blueprint for the system.

The goal was to design a clean, normalized, and accurate representation of how events, users, bookings, and payments interact. The conceptual model includes seven core entities:

- ADMIN
- CUSTOMER
- ORGANIZER
- VENUE
- EVENT
- BOOKING
- PAYMENT

4.2 Entity Descriptions

1. ADMIN
Stores system administrators who have elevated privileges.

- **Purpose:** To manage the entire EMS, including events, venues, organizers, and system oversight.
- **Key Attributes:** admin_id (Unique identifier), username, email, password_hash, role (Admin/Staff).

2. CUSTOMER
Represents the users who attend events or make bookings.

- **Purpose:** To store attendee information consistently.
- **Key Attributes:** customer_id (Unique identifier), name, email, phone (optional).

3. ORGANIZER
Organizers host and manage events.

- **Purpose:** To properly represent event creators.
- **Key Attributes:** organizer_id (Unique identifier), name, email, phone (optional).

4. VENUE
Represents physical locations where events are hosted.

- **Purpose:** To define event locations with capacities and types.
- **Key Attributes:** venue_id (Unique identifier), name, address, capacity, type (Exhibition/Conference/Concert/Sports Game).
- *Note:* The "type" attribute enforces the rule that venues are suitable only for certain event types.

5. EVENT
Represents a scheduled event.

- **Purpose:** To store details about events and connect them to venues and organizers.

- **Key Attributes:** event_id, organizer_id (FK), venue_id (FK), title, start_time, end_time, capacity, status (Draft/Published/Cancelled).

6. BOOKING
Represents reservations or tickets purchased for an event.

- **Purpose:** To track seats/tickets requested, payment price, and status.
- **Key Attributes:** booking_id, event_id (FK), customer_id (FK), ticket_qty, unit_price, total_price (calculated), status, booked_at.

7. PAYMENT
Represents payments completed for bookings.

- **Purpose:** To store financial transactions associated with paid bookings.
- **Key Attributes:** payment_id, booking_id (FK), customer_id (FK), amount, method, card_details (masked), paid_at.

4.3 Relationships Between Entities

The EMS ERD establishes the following relationships:

1. **One Organizer → Many Events:** An organizer can host multiple events; an event belongs to one organizer.
2. **One Venue → Many Events:** A venue can host many events over time.
3. **One Event → Many Bookings:** Users may reserve multiple seats; many customers book the same event.
4. **One Customer → Many Bookings:** A customer can make multiple bookings.
5. **One Booking → One Payment:** Each booking has one corresponding payment record.
6. **Admin Roles Are Separate:** Admins are system-level users distinct from organizers or customers.

4.4 Data Dictionary Summary

The data dictionary clarifies field names, data types, constraints, and rules. For example:

- ticket_qty must be between 1–10.
- event status must be Draft, Published, or Cancelled.
- booking status must be Pending, Approved, or Cancelled.
- payment method must be Cash, Card, or Online.

4.5 Conceptual ERD Explanation

- **EVENT** sits at the center, linked to **ORGANIZER** and **VENUE** via one-to-many relationships.

- **CUSTOMER** connects to **BOOKING**, and each booking links to a specific event.
- **PAYMENT** links to a booking for one-to-one tracking.
- **ADMIN** exists separately for system management.

Chapter 5 — Physical Database Implementation

5.1 Overview of Final Schema

The conceptual ERD was translated into a physical database using Django's ORM (Object Relational Mapping). The physical schema is realized in the db.sqlite3 file. While the conceptual ERD provided the structure, the physical schema includes implementation details such as foreign keys, authentication tables, and specific field types.

5.2 Core Tables Implemented

Django generated tables reflecting the conceptual entities:

1. **admin table:** Corresponds to the ADMIN entity.
2. **customer table:** Stores user info (name, email, phone).
3. **organizer table:** Stores organizer info and links to a Django user account.
4. **venue table:** Stores location, capacity, and compatibility.
5. **event table:** Stores details, foreign keys, status, plus implementation fields like description and base ticket price.
6. **booking table:** Stores reservation data, calculated prices, and timestamps.
7. **payment table:** Stores masked payment details and timestamps.

5.3 Additional Implementation Tables

The physical implementation required tables not present in the conceptual ERD:

1. **accounts_userprofile:** Links Django's built-in authentication (auth_user) to domain entities (CUSTOMER and ORGANIZER). It manages logins and user roles.
2. **ai_pendingaiaction:** Stores AI requests, including user, action type, payload, and timestamps, to support asynchronous AI handling.
3. **Django Authentication & Session Tables:** auth_user, auth_group, django_session, etc., supporting login and session management.

5.4 Differences Between Conceptual and Physical Schema

1. **EVENT table includes new fields:** description (for rich text) and ticket_price (stored at the event level).
2. **ORGANIZER table includes user_id:** A foreign key linking the organizer to a Django account for authentication.
3. **accounts_userprofile table:** Added to link Django users to profiles.
4. **ai_pendingaiaction table:** Added for AI feature support.
5. **Nullability:** Some fields (e.g., capacity for exhibitions, phone numbers) were made nullable.
6. **Primary Keys:** Django automatically generates auto-incrementing integer PKs.

5.5 How the ORM Enforces Schema Rules

1. **Foreign Key Integrity:** Deleting a Venue or Customer cascades appropriately to associated records.
2. **Business Logic Enforcement:** Model methods validate overbooking, payment requirements, and zero-pricing for conferences.
3. **Migration System:** Ensures safe schema modifications and consistency.
4. **Index Creation:** Foreign keys are indexed for performance.
5. **Field Validation:** Data types, ranges, and formats are validated before saving.

Chapter 6 — System Architecture & Project Structure

6.1 Why I Chose Django

Django was selected for several reasons:

1. **Rapid Development:** Provides a powerful ORM, authentication, admin dashboard, and routing.
2. **"Batteries Included":** Includes database integration, security (hashing, CSRF), and session management.
3. **CRUD Support:** Simplifies operations for Events, Customers, and Bookings.
4. **External Service Integration:** Easily integrates with the local AI model via HTTP/JSON.
5. **Documentation:** Strong community support and documentation.

6.2 High-Level System Architecture

The architecture follows a layered approach:

- **Layer 1 — Presentation (Frontend):** HTML templates, CSS, forms, and the chatbot interface.
- **Layer 2 — Application (Django):** Views, Forms, Models, URL routing, and business logic.
- **Layer 3 — Database:** SQLite storage for all entities and logs.
- **Layer 4 — AI Integration:** A dedicated Python client (ai_client.py) connecting to the local Ollama/Llama 3.1 API.

6.3 Django App Structure in EMS

The system is split into modular apps:

1. **ems_core:** Main application layer containing settings, root URLs, homepage, and AI endpoints.
2. **accounts:** User and profile management (registration, login, profile linking).
3. **events:** Event and venue management (CRUD, listing, dashboards).
4. **bookings:** Reservations, payments, and business rule enforcement.

6.4 Data Flow Through the System

1. **User Action:** User interacts with the UI (e.g., clicks an event).
2. **URL Routing:** Django maps the request to a view (e.g., /events/23/).
3. **View Logic:** Fetches data, validates input, applies rules, and renders a response.
4. **Model Interaction:** Queries or updates the database.
5. **Template Rendering:** Presents data via HTML.
6. **Response:** The page is returned to the user.

6.5 URLs, Views, Templates, and Logic Layer

Django uses an MVC-like pattern:

- **URLs:** Map paths to views.
- **Views:** Handle logic, validation, and permissions.
- **Templates:** Render the UI using Bootstrap/Tailwind.
- **Models:** Define table structures and relationships.
- **Forms:** Validate user input and handle errors.

Chapter 7 — Core Features & User Guide

7.1 Browsing Events

The homepage displays a list of published events with titles, types, dates, venues, and a "View Details" button. Users can browse without logging in and filter events by type (Exhibitions, Conferences, Concerts, Sports Games).

7.2 Exhibition Workflow (Announcement Only)

Exhibitions are open events.

- **Capabilities:** View details, description, venue, date/time, and ask the AI.
- **Restrictions:** No booking, reservation, or payment is required.

7.3 Conference Workflow (Free Reservation)

Conferences require free reservations and are capacity-limited.

- **Process:** Click "Reserve Seat", enter quantity (1-10).
- **System Check:** Verifies event and venue capacity.
- **Result:** Booking created with unit_price = 0 and total_price = 0.

7.4 Concert & Sports Workflow (Paid Ticketing)

These events require paid tickets.

- **Process:** Select ticket quantity. System calculates total_price. Select payment method (Cash, Card, Online).
- **System Check:** Verifies capacity and payment status.
- **Result:** Booking created, followed by a linked payment record. Card details are masked.

7.5 Viewing Booking History

Logged-in users can view their history, including event name, date, ticket quantity, status (Pending/Approved/Cancelled), and masked payment details.

7.6 Organizer Features

Organizers have a dedicated dashboard to:

- **Create Events:** Define title, venue, time, capacity, type, price, and description.
- **Manage Workflow:** Save as Draft, Publish, or Cancel.
- **Manage Bookings:** View, approve, or cancel bookings; track capacity and revenue.
- **AI Assistance:** Generate event descriptions automatically.

7.7 Admin Features

Admins have full system oversight:

- Create/manage venues.
- Oversee all events, bookings, and payments.
- Ensure data integrity and resolve disputes.
- Access the Django admin panel for direct data management.

7.8 Example User Journeys

1. **Reserving a Conference:** User filters by Conference, selects an event, reserves 2 seats. System confirms capacity and creates a free booking.

Buying Concert Tickets: User selects 2 tickets (
49.99each).Systemcalculatestotal(49.99 each). System calculates total
(49.99each).Systemcalculatestotal(

2. 99.98). User pays via Card. Booking and Payment records are created.
3. **Organizer Creating Event:** Organizer logs in, creates "Tech Innovation Fair" (Exhibition), uses AI for description, saves as Draft, then Publishes.
4. **AI Query:** User asks "Show me sports events this month." AI lists matching events.

Chapter 8 — AI Integration (Ollama + Llama 3.1)

8.1 Why I Added AI to the EMS

Traditional event systems rely on rigid menus and filters. AI was integrated to provide:

1. **Natural Language Search:** Users can ask questions like "What concerts are happening this weekend?"
2. **Content Generation:** Organizers can generate professional descriptions effortlessly.
3. **Accessibility:** Simplifies navigation for non-technical users.
4. **Privacy:** Local processing via Llama 3.1 ensures no data leaves the system.

8.2 AI Architecture

The modular architecture consists of:

1. **User Interface:** Chatbot window.
2. **Django View (ai_chat_view):** Handles requests/responses.
3. **AI Client (ai_client.py):** Manages HTTP requests to Ollama and error handling.
4. **Ollama Server:** Local engine running at localhost:11434.
5. **Llama 3.1 Model:** The neural model processing natural language.

8.3 How AI Requests Flow Through the System

1. User types a message.
2. Django sends the request to ai_client.py.
3. Client sends a JSON payload via HTTP POST to Ollama.
4. Llama 3.1 generates a response.
5. Django formats and returns the response to the UI.

8.4 AI Commands You Can Try

- **Search:** "Show me all concerts happening this month," "List free conferences tomorrow."
- **Availability:** "Which events have available seats?" "Is the Data Science Conference full?"
- **Financial:** "What is the price for Homecoming tickets?"
- **Content:** "Write a friendly description for my Cultural Exhibition."
- **Admin:** "Summarize events with low attendance."

8.5 How AI Helps Different Users

- **Attendees:** Find events via natural language.
- **Organizers:** Generate content and improve descriptions.
- **Admins:** Gain insights and summaries of system data.

8.6 Limitations of Local LLM Integration

1. **No Direct Database Access:** AI cannot query SQL directly for safety.
2. **No Real-Time SQL Generation:** AI provides natural language answers, not database commands.
3. **Variability:** Responses may vary or be slightly inaccurate.
4. **Resource Usage:** Requires significant RAM/CPU.
5. **No Fine-Tuning:** Uses general knowledge, not domain-specific training.

Chapter 9 — Security & Privacy Considerations

9.1 Password Protection & Hashing

EMS uses Django's authentication system, employing **PBKDF2** and **Salted Hashes** to ensure passwords are never stored in plain text.

9.2 Role-Based Access Control

- **Public Users:** View events, book tickets, view history.
- **Organizers:** Manage own events and bookings.
- **Admins:** Full system access. Permissions are enforced at the View, Template, and URL levels.

9.3 Payment Data Handling

To avoid PCI compliance issues, payments are conceptual.

- No real payment gateway is used.
- **Masked Details:** Card numbers are stored as **** **** **** 4242.
- **Relationship Integrity:** Payments must match bookings and prices exactly.

9.4 Input Validation

- **Form Validation:** Checks required fields, data types, and formats.
- **Server-Side Validation:** Enforced even if frontend checks are bypassed.
- **Business Logic:** Validates capacity and pricing rules.

9.5 Keeping AI Interaction Secure

- **No Direct DB Access:** AI operates only on provided context.
- **Sanitized Input:** User inputs are sanitized to prevent injection attacks.
- **Local Processing:** No external API calls; user data stays private.

Chapter 10 — Testing & Evaluation

10.1 Testing Approach

The system underwent Functional, Validation, Integration, UI/UX, Performance, and Error Handling testing.

10.2 Functional Tests

- **Event Listing:** Verified page loads correctly.
- **Event Details:** Verified correct data display.
- **Conference Booking:** Verified free booking logic (price = 0).
- **Paid Booking:** Verified ticket calculation and payment creation.
- **Booking History:** Verified correct retrieval of user data.
- **Organizer/Admin Actions:** Verified event and venue creation.

10.3 Capacity and Booking Rule Tests

- **Overbooking:** Attempting to book beyond capacity triggers an error.
- **Cancelled Events:** Booking attempts are rejected.
- **Exhibitions:** Booking options are hidden.
- **Conference Pricing:** Enforced as free.
- **Paid Events:** Payment is mandatory.

10.4 Payment Workflow Tests

- **Linkage:** Payment correctly links to Booking.
- **Calculation:** Total price = quantity × unit price.
- **Masking:** Card details are stored masked.
- **Negative Values:** Negative ticket quantities are rejected.

10.5 AI Testing

- **Natural Language:** AI correctly identifies intent (e.g., "Show concerts").
- **Description Generation:** AI produces coherent text.
- **Unknown Requests:** AI handles unrelated queries gracefully.
- **Response Time:** Acceptable performance (< 2 seconds).

10.6 Performance Evaluation

- **Page Load:** Instant homepage and listing rendering.
- **Database:** Fast queries via indexed foreign keys.
- **AI:** Quick response via Ollama on supported hardware.

Chapter 11 — Limitations & Future Enhancements

11.1 Current Limitations

1. **No Real Payment Gateway:** Payments are "record only"; no Stripe/PayPal integration.
2. **No Notification System:** No emails or SMS are sent.
3. **Limited Analytics:** Basic dashboards without deeper insights.
4. **Basic UI/UX:** Functional but lacks advanced interactivity (e.g., animations).
5. **Local AI Limits:** No domain fine-tuning or direct SQL access.
6. **Single Organizer:** Events map to only one organizer.
7. **No Mobile App:** Web-only interface.
8. **Limited Conflict Detection:** Does not fully prevent venue scheduling overlaps.
9. **No Seat-Level Reservation:** Reserves counts, not specific seats.

11.2 Planned Improvements

1. **Payment Integration:** Add Stripe/Razorpay.
2. **Notification System:** Email confirmations and push notifications.
3. **Enhanced Analytics:** Charts and heatmaps for organizers.
4. **Mobile-Friendly:** Improved responsive design or mobile app.
5. **Multi-Organizer Support:** Allow committees/clubs to host events.
6. **Real-Time Availability:** Live updates via WebSockets.
7. **Staff Management:** Volunteer tracking.
8. **AI Enhancements:** Personalized recommendations and overbooking predictions.

11.3 Future AI Capabilities

- **Conversational Booking:** Booking tickets directly via chat.
- **Voice Support:** Speech-to-text integration.
- **Content Editing:** Tone adjustment and grammar checks.
- **Auto-Summaries:** Automatic reporting on revenue/attendance.
- **Recommendations:** User-interest based suggestions.

11.4 Potential for Real-World Deployment

With minimal enhancements (PostgreSQL, payment gateways, notifications, and security hardening), EMS could be deployed for universities, clubs, or community centers.

Chapter 12 — Conclusion

12.1 Project Summary

The development of the Event Management System (EMS) represents a comprehensive solution to the inefficiencies and fragmentation often found in traditional event planning workflows. By centralizing event discovery, registration, and management into a unified web-based platform, the EMS successfully bridges the operational gaps between attendees, organizers, and administrators.

The system utilizes a robust Django backend architecture, ensuring modularity, security, and scalability. A distinguishing feature of this project is the integration of a local Large Language Model (Llama 3.1 via Ollama). This integration provides users with natural language search capabilities and organizers with content generation tools, all while maintaining strict data privacy by eliminating reliance on external cloud APIs.

12.2 Achievement of Objectives

The project has successfully met the technical and functional objectives outlined at the inception of the study:

- **User-Centric Design:** The system provides a seamless interface for browsing and booking diverse event types, from free conferences to paid concerts. The problem of "scattered information" has been resolved by creating a single source of truth for all event data.
- **Data Integrity:** By strictly adhering to the conceptual Entity-Relationship Diagram (ERD) during the physical implementation, the system ensures consistent handling of complex relationships between venues, capacities, and bookings.
- **Operational Efficiency:** Organizers are equipped with dashboards that automate capacity enforcement and status tracking, significantly reducing the likelihood of manual errors such as overbooking.
- **Security and Privacy:** The implementation of industry-standard security practices, including PBKDF2 password hashing, role-based access control, and payment data masking, ensures that the system is safe for deployment in academic or institutional settings.

12.3 Final Remarks

The Event Management System demonstrates that effective software solutions do not require complex dependencies on third-party cloud services. By leveraging open-source technologies and local AI processing, the EMS offers a sustainable, cost-effective, and private alternative to commercial event platforms.

While the current iteration serves as a functional prototype with defined limitations, the modular code structure provides a solid foundation for future enhancements, such as mobile application development and real-time analytics. Ultimately, this project illustrates the practical application of database theory and modern web frameworks to solve real-world logistical challenges.

Appendices

Appendix A — ERD (Descriptive Version)

Entities and Their Relationships

ADMIN

- admin_id (PK), username, email, password_hash, role.
- *Purpose:* System management.

CUSTOMER

- customer_id (PK), name, email, phone.
- *Purpose:* Attendee booking.

ORGANIZER

- organizer_id (PK), name, email, phone.
- *Relationship:* Hosts many events.

VENUE

- venue_id (PK), name, address, capacity, type.
- *Relationship:* Hosts many events.

EVENT

- event_id (PK), organizer_id (FK), venue_id (FK), title, start_time, end_time, capacity, status.
- *Relationships:* Linked to Organizer, Venue, and Bookings.

BOOKING

- booking_id (PK), event_id (FK), customer_id (FK), ticket_qty, unit_price, total_price, status, booked_at.
- *Relationships:* Links Customer to Event.

PAYMENT

- payment_id (PK), booking_id (FK), customer_id (FK), amount, method, card_details, paid_at.
- *Relationships:* One-to-one with Booking.

Appendix B — Data Dictionary

ADMIN Table

- admin_id: PK (int) - Unique identifier.
- username: varchar - Login name.
- email: varchar - Email address.
- password_hash: varchar - Encrypted password.
- role: varchar - Admin/Staff.

CUSTOMER Table

- customer_id: PK - Unique identifier.
- name: varchar - Full name.
- email: varchar - Contact email.
- phone: varchar - Phone number (optional).

ORGANIZER Table

- organizer_id: PK - Unique identifier.
- name: varchar - Organizer name.
- email: varchar - Contact email.
- phone: varchar - Phone number (optional).

VENUE Table

- venue_id: PK - Unique identifier.
- name: varchar - Venue name.
- address: varchar - Physical address.
- capacity: int - Max seating.
- type: varchar - Allowed event type.

EVENT Table

- event_id: PK - Unique identifier.

- organizer_id: FK - Link to Organizer.
- venue_id: FK - Link to Venue.
- title: varchar - Event title.
- start_time: datetime - Start.
- end_time: datetime - End.
- capacity: int - Event capacity.
- status: varchar - Draft/Published/Cancelled.

BOOKING Table

- booking_id: PK - Unique identifier.
- event_id: FK - Link to Event.
- customer_id: FK - Link to Customer.
- ticket_qty: int - Tickets booked.
- unit_price: decimal - Price per ticket.
- total_price: decimal - Total cost.
- status: varchar - Pending/Approved/Cancelled.
- booked_at: datetime - Timestamp.

PAYMENT Table

- payment_id: PK - Unique identifier.
- booking_id: FK - Link to Booking.
- customer_id: FK - Link to Customer.
- amount: decimal - Payment amount.
- method: varchar - Cash/Card/Online.
- card_details: varchar - Masked info.
- paid_at: datetime - Timestamp.

Appendix C — SQL Query Examples

1. Select all concerts

code SQL
downloadcontent_copy
expand_less

```
  SELECT * FROM event WHERE type = 'Concert';
```

2. Find all bookings for a specific event

code SQL
downloadcontent_copy
expand_less

```
  SELECT * FROM booking WHERE event_id = 203;
```

3. Get all payments made by a specific customer

```
code SQL
downloadcontent_copy
expand_less
  SELECT * FROM payment WHERE customer_id = 12;
```

4. Count bookings for each event

```
code SQL
downloadcontent_copy
expand_less
  SELECT event_id, COUNT(*) AS total_bookings FROM booking GROUP BY event_id;
```

5. Find available seats for an event

```
code SQL
downloadcontent_copy
expand_less
  SELECT capacity - (
   SELECT COALESCE(SUM(ticket_qty), 0)
   FROM booking
   WHERE event_id = e.event_id
) AS seats_remaining
FROM event e
WHERE event_id = 45;
```

Appendix D — AI Prompt Cheat Sheet

Event Search Prompts

- Show me concerts happening this month.
- Find free conferences this weekend.
- What exhibitions are available today?
- List sports events with available tickets.

Organizer Prompts

- Write a description for my Cultural Night 2025 event.
- Generate a professional summary for the Photography Expo.

- Improve the tone of this event description.

Booking/Payment Prompts

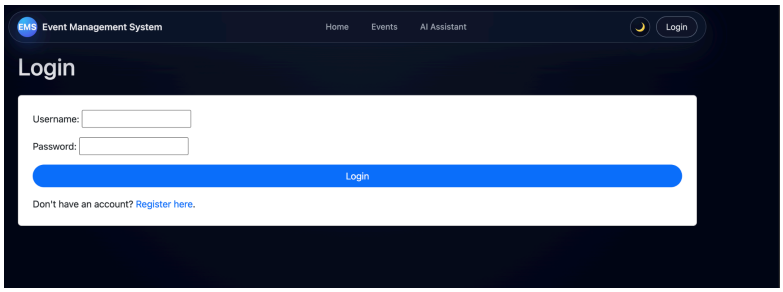
- Summarize all bookings for the Data Science Conference.
- How many seats are remaining for the Homecoming Concert?
- What events require paid tickets?

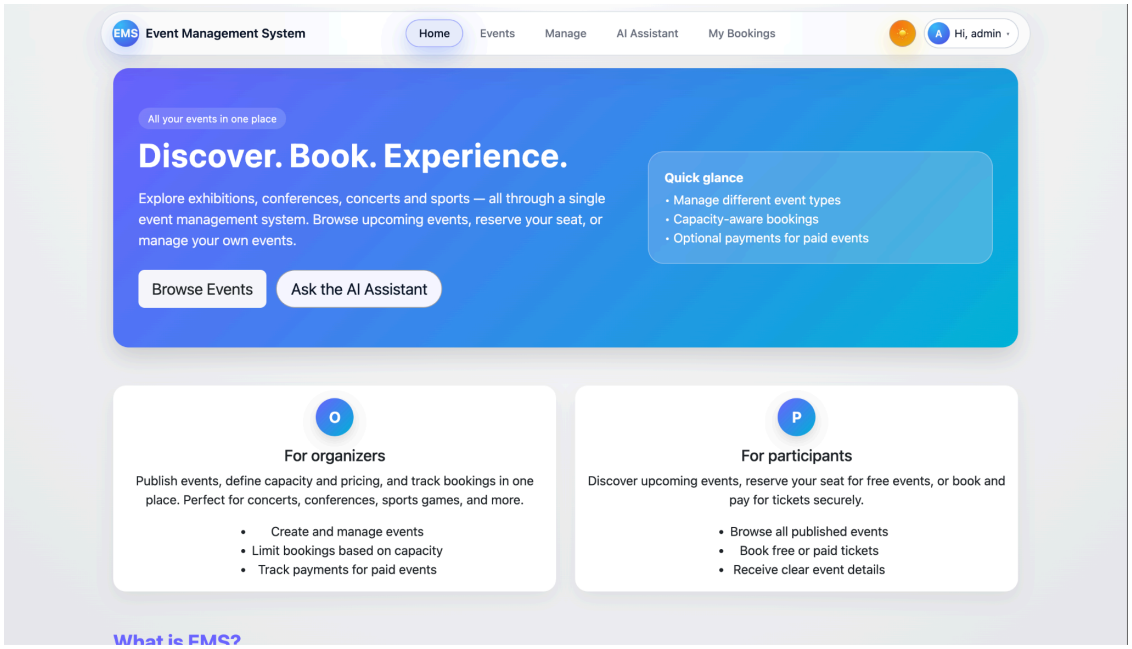
Admin Prompts

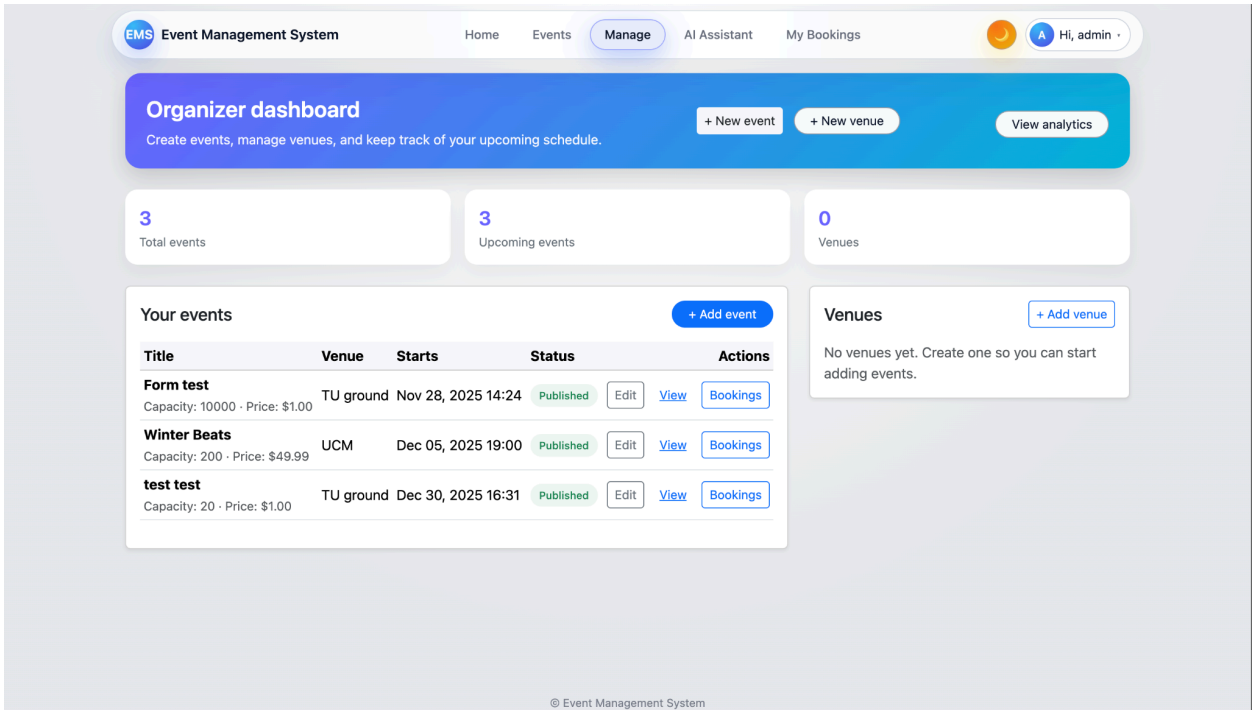
- List events that are nearly full.
- Summarize payments for Event ID 203.
- Which organizers have the most upcoming events?

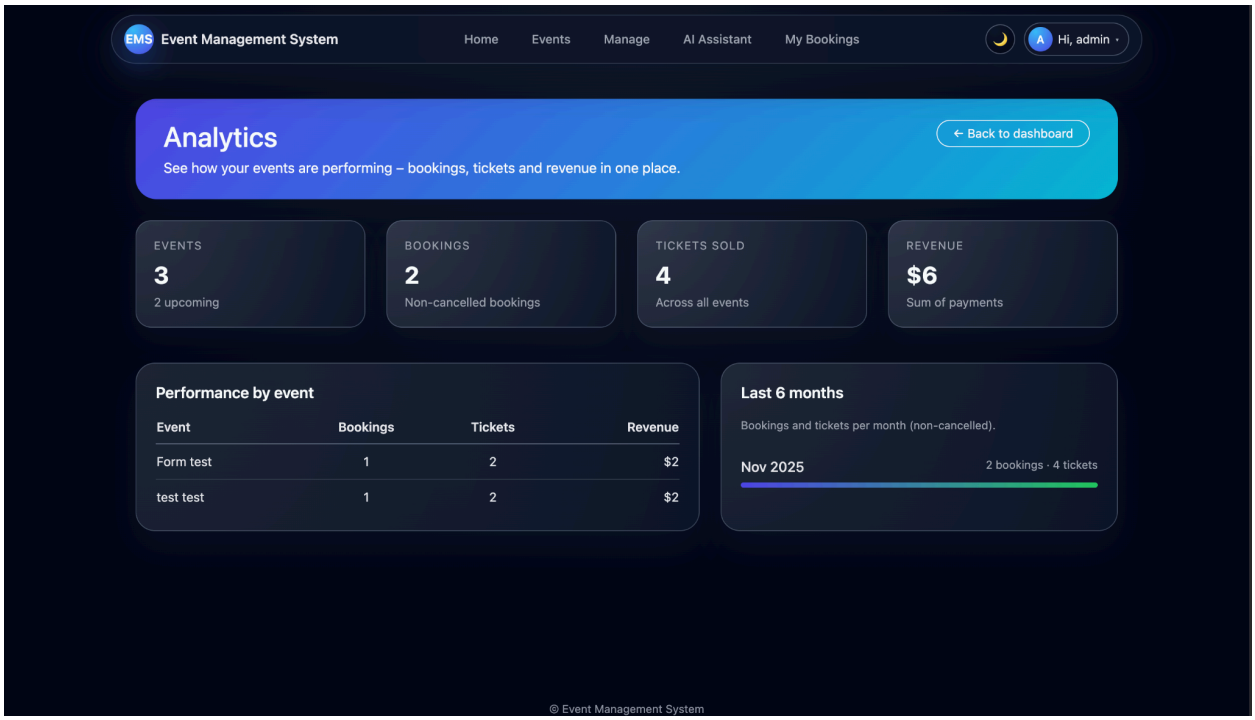
Appendix E — Screenshots of EMS Pages

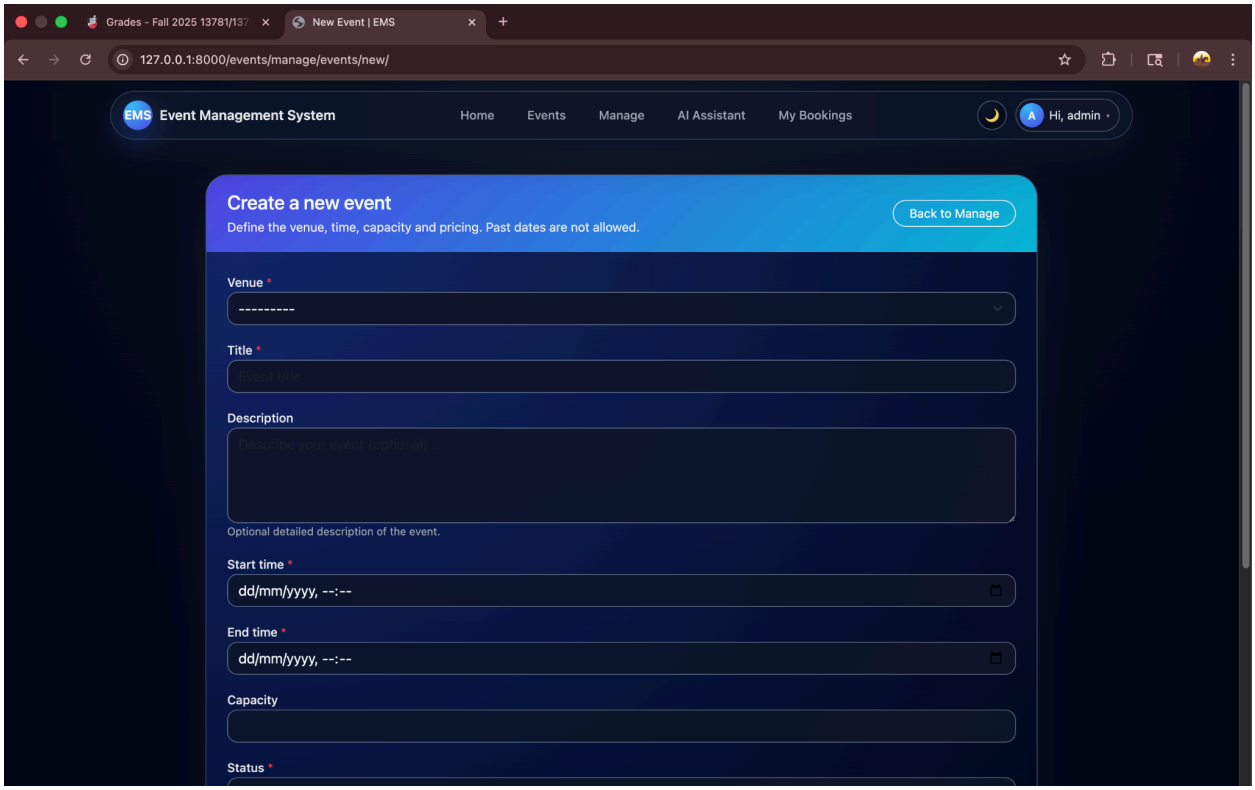
Screenshots are omitted from this document but are recommended for the final report to demonstrate the working system. Suggested screenshots include:


EMS Event Management System
Home
Events
AI Assistant
Login
Login
Username:
Password:
Login
Don't have an account? Register here.

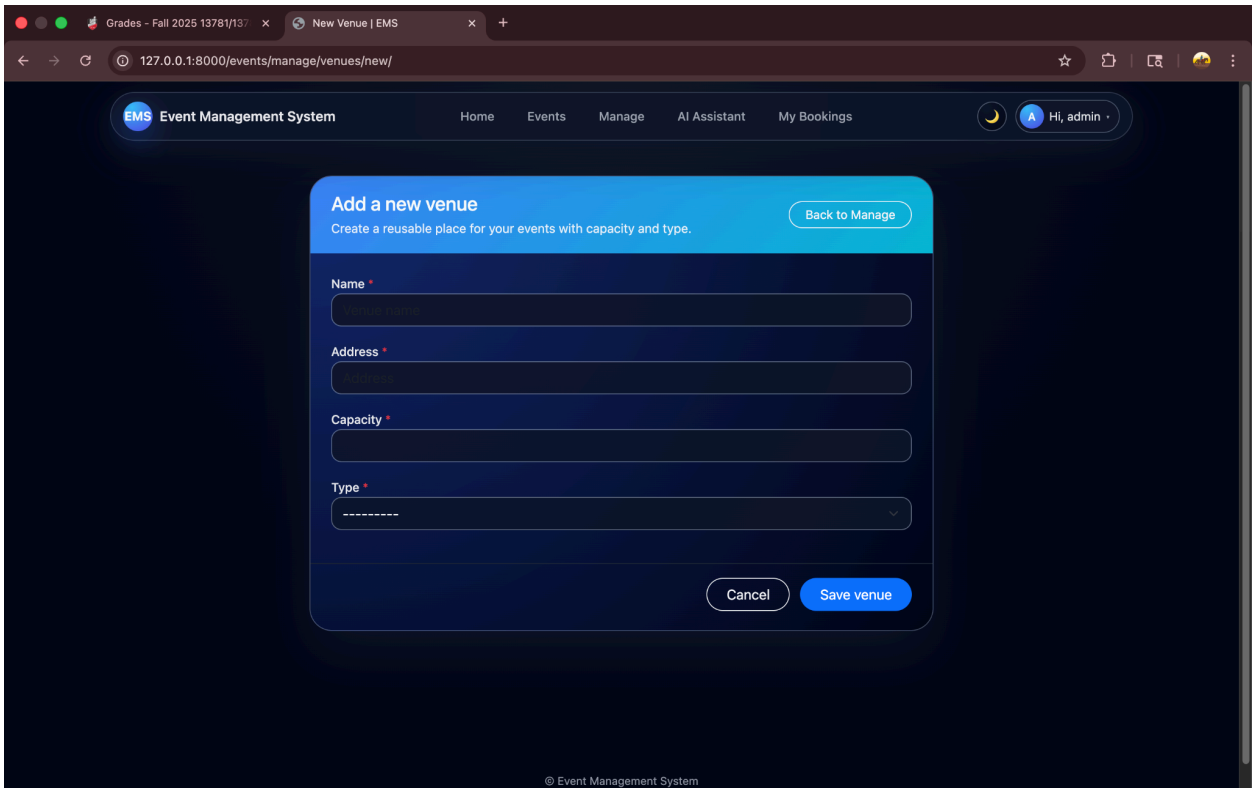

EMS Event Management System
Home
Events
Manage
AI Assistant
My Bookings
A Hi, admin
All your events in one place
Discover. Book. Experience.
Explore exhibitions, conferences, concerts and sports — all through a single event management system. Browse upcoming events, reserve your seat, or manage your own events.
Browse Events
Ask the AI Assistant
Quick glance
• Manage different event types
• Capacity-aware bookings
• Optional payments for paid events
O
For organizers
Publish events, define capacity and pricing, and track bookings in one place. Perfect for concerts, conferences, sports games, and more.
• Create and manage events
• Limit bookings based on capacity
• Track payments for paid events
P
For participants
Discover upcoming events, reserve your seat for free events, or book and pay for tickets securely.
• Browse all published events
• Book free or paid tickets
• Receive clear event details
What is EMS?


EMS
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
A
Hi, admin
Organizer dashboard
Create events, manage venues, and keep track of your upcoming schedule.
+ New event
+ New venue
View analytics
3
Total events
3
Upcoming events
0
Venues
Your events
+ Add event
Title
Venue
Starts
Status
Actions
Form test
Capacity: 10000 · Price: $1.00
TU ground
Nov 28, 2025 14:24
Published
Edit
View
Bookings
Winter Beats
Capacity: 200 · Price: $49.99
UCM
Dec 05, 2025 19:00
Published
Edit
View
Bookings
test test
Capacity: 20 · Price: $1.00
TU ground
Dec 30, 2025 16:31
Published
Edit
View
Bookings
Venues
+ Add venue
No venues yet. Create one so you can start adding events.
© Event Management System

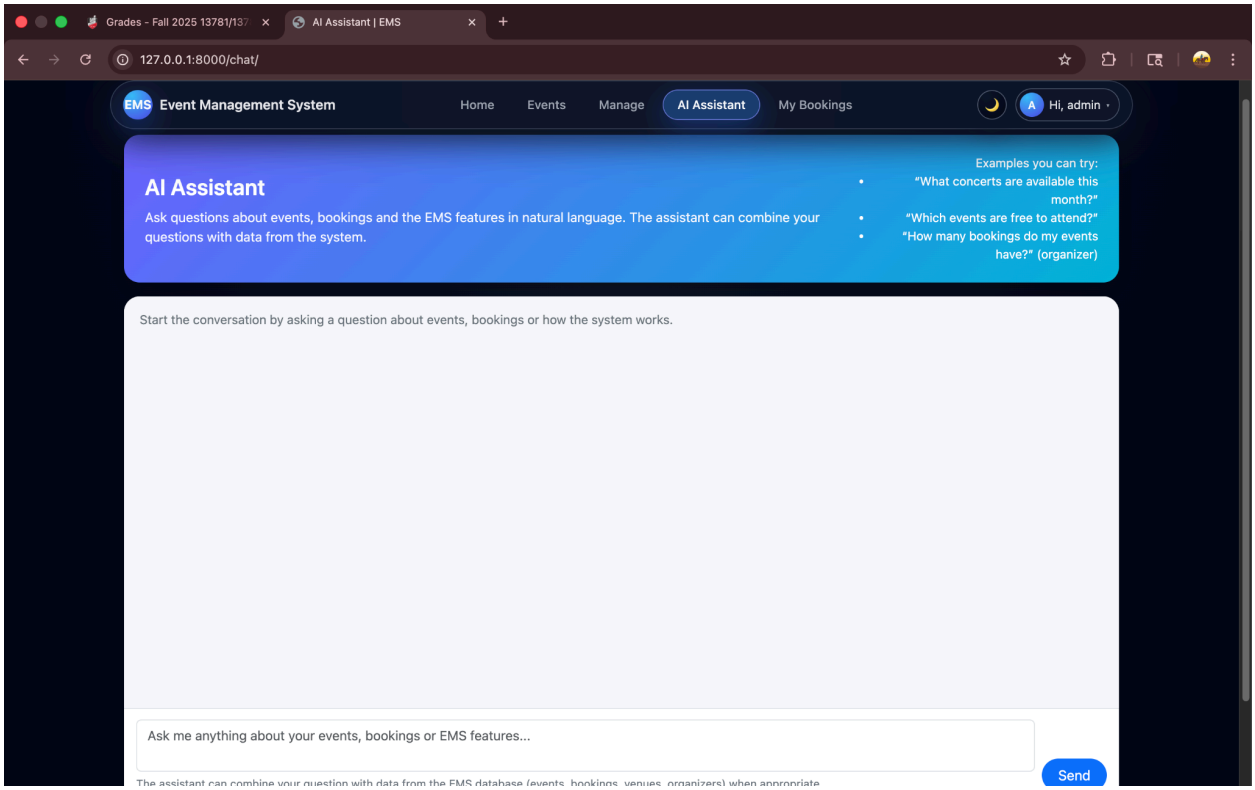

EMS
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
A
Hi, admin
Analytics
See how your events are performing – bookings, tickets and revenue in one place.
← Back to dashboard
EVENTS
3
2 upcoming
BOOKINGS
2
Non-cancelled bookings
TICKETS SOLD
4
Across all events
REVENUE
$6
Sum of payments
Performance by event
Event
Bookings
Tickets
Revenue
Form test
1
2
$2
test test
1
2
$2
Last 6 months
Bookings and tickets per month (non-cancelled).
Nov 2025
2 bookings · 4 tickets
© Event Management System

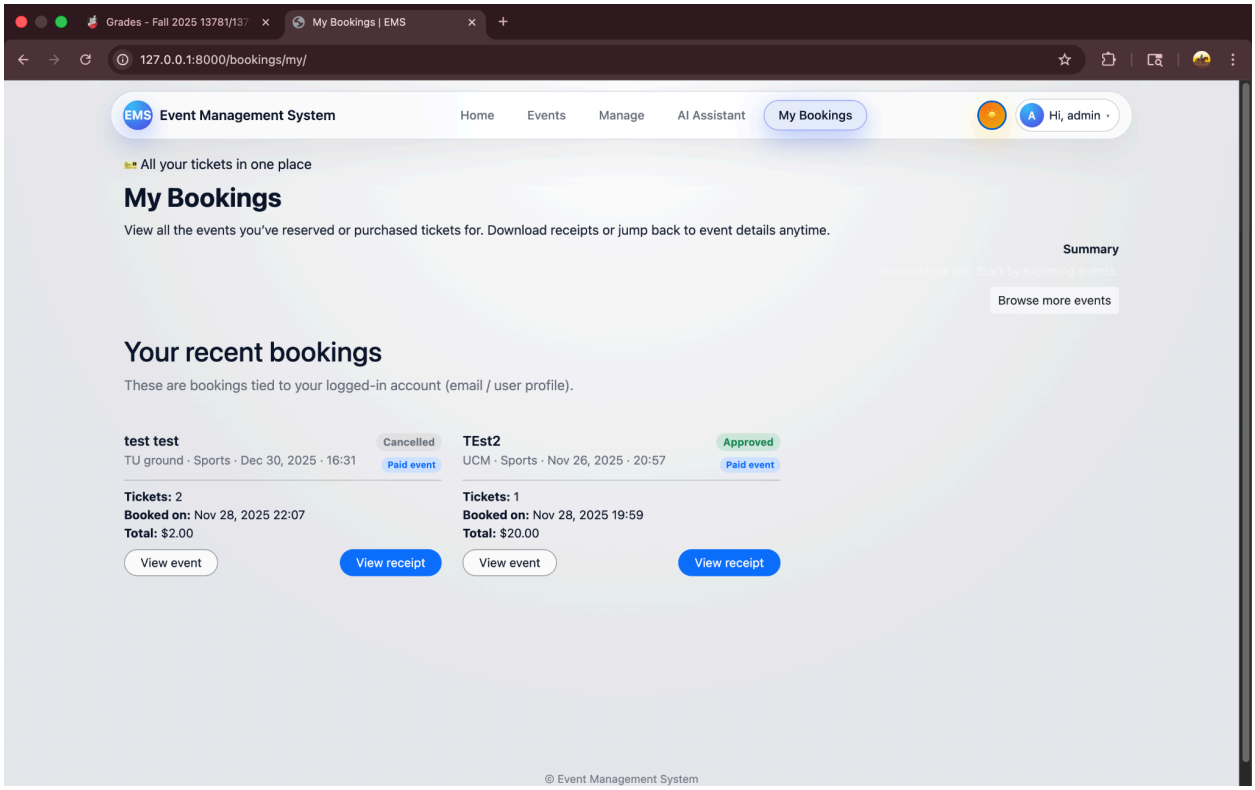

Grades - Fall 2025 13781/137
New Event | EMS
127.0.0.1:8000/events/manage/events/new/
EMS Event Management System
Home
Events
Manage
AI Assistant
My Bookings
Hi, admin
Create a new event
Define the venue, time, capacity and pricing. Past dates are not allowed.
Back to Manage
Venue *

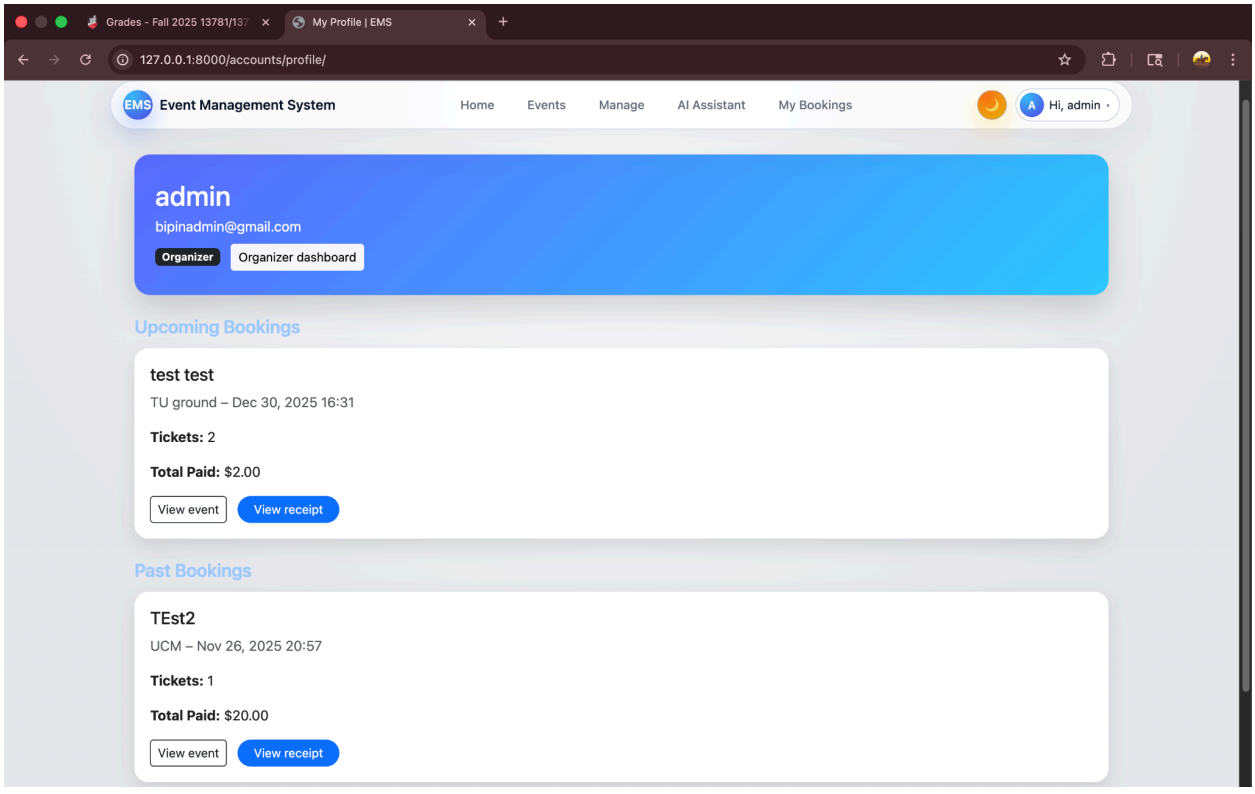
Title *
Event title
Description
Describe your event (optional)
Optional detailed description of the event.
Start time *
dd/mm/yyyy, --:--
End time *
dd/mm/yyyy, --:--
Capacity
Status *

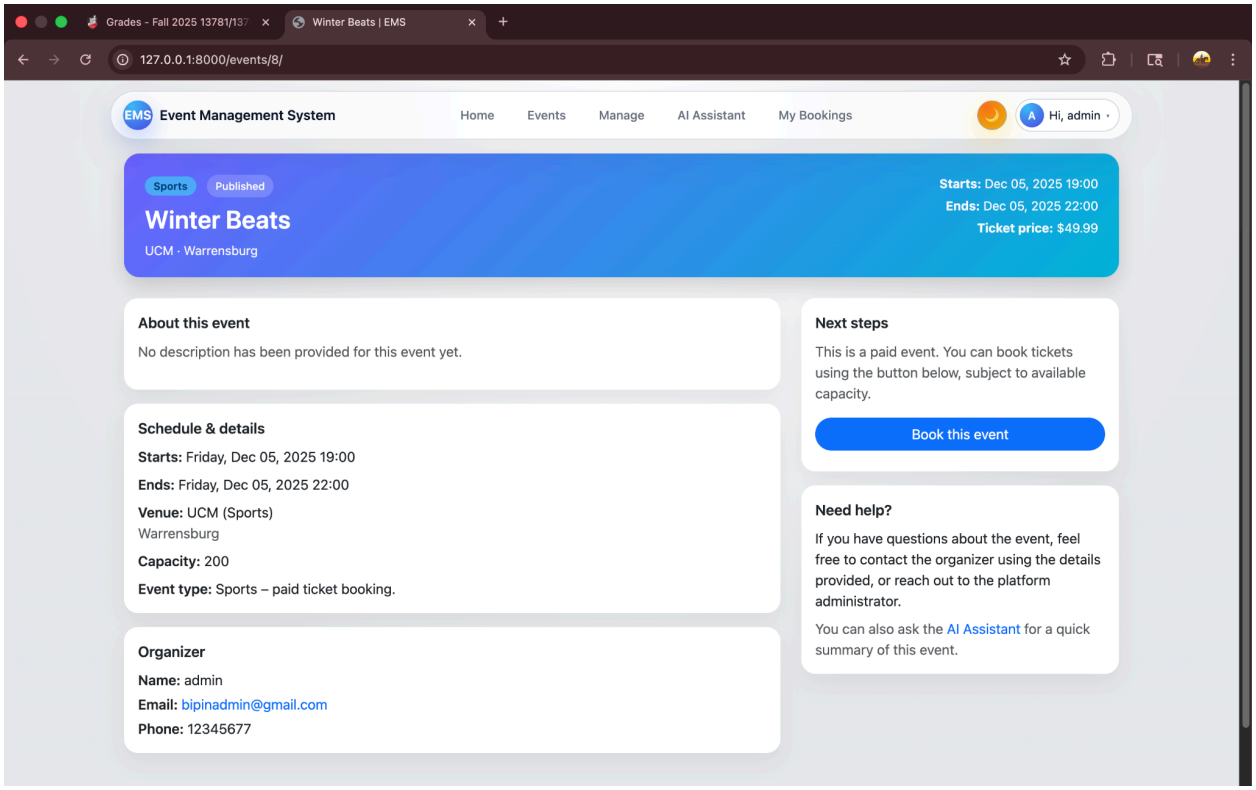

Grades - Fall 2025 13781/137
New Venue | EMS
127.0.0.1:8000/events/manage/venues/new/
EMS Event Management System
Home
Events
Manage
AI Assistant
My Bookings
Hi, admin
Add a new venue
Create a reusable place for your events with capacity and type.
Back to Manage
Name *
Venue name
Address *
Address
Capacity *
Type *

Cancel
Save venue
© Event Management System


Grades - Fall 2025 13781/137
AI Assistant | EMS
127.0.0.1:8000/chat/
EMS
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
A
Hi, admin
AI Assistant
Ask questions about events, bookings and the EMS features in natural language. The assistant can combine your questions with data from the system.
Examples you can try:
"What concerts are available this month?"
"Which events are free to attend?"
"How many bookings do my events have?" (organizer)
Start the conversation by asking a question about events, bookings or how the system works.
Ask me anything about your events, bookings or EMS features...
Send
The assistant can combine your question with data from the EMS database (events, bookings, venues, organizers) when appropriate


Grades - Fall 2025 13781/137
My Bookings | EMS
127.0.0.1:8000/bookings/my/
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
Hi, admin
All your tickets in one place
My Bookings
View all the events you've reserved or purchased tickets for. Download receipts or jump back to event details anytime.
Summary
Browse more events
Your recent bookings
These are bookings tied to your logged-in account (email / user profile).
test test
Cancelled
TU ground · Sports · Dec 30, 2025 · 16:31
Paid event
Tickets: 2
Booked on: Nov 28, 2025 22:07
Total: $2.00
View event
View receipt
TEst2
Approved
UCM · Sports · Nov 26, 2025 · 20:57
Paid event
Tickets: 1
Booked on: Nov 28, 2025 19:59
Total: $20.00
View event
View receipt
© Event Management System


Grades - Fall 2025 13781/137
My Profile | EMS
127.0.0.1:8000/accounts/profile/
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
Hi, admin
admin
bipinadmin@gmail.com
Organizer
Organizer dashboard
Upcoming Bookings
test test
TU ground – Dec 30, 2025 16:31
Tickets: 2
Total Paid: $2.00
View event
View receipt
Past Bookings
TEst2
UCM – Nov 26, 2025 20:57
Tickets: 1
Total Paid: $20.00
View event
View receipt


Grades - Fall 2025 13781/137
Winter Beats | EMS
127.0.0.1:8000/events/8/
EMS
Event Management System
Home
Events
Manage
AI Assistant
My Bookings
A
Hi, admin
Sports
Published
Winter Beats
UCM · Warrensburg
Starts: Dec 05, 2025 19:00
Ends: Dec 05, 2025 22:00
Ticket price: $49.99
About this event
No description has been provided for this event yet.
Schedule & details
Starts: Friday, Dec 05, 2025 19:00
Ends: Friday, Dec 05, 2025 22:00
Venue: UCM (Sports)
Warrensburg
Capacity: 200
Event type: Sports – paid ticket booking.
Organizer
Name: admin
Email: bipinadmin@gmail.com
Phone: 12345677
Next steps
This is a paid event. You can book tickets using the button below, subject to available capacity.
Book this event
Need help?
If you have questions about the event, feel free to contact the organizer using the details provided, or reach out to the platform administrator.
You can also ask the AI Assistant for a quick summary of this event.