



FINAL DOCUMENT

TITLE: EVENT MANAGEMENT PLATFORM

NAME: MIKE AMASA

REG NO: 23/06557

SUPERVISOR NAME: OLOO JACOB

THIS FINAL DOCUMENT  
IS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS OF THE AWARD DIPLOMA IN INFORMATION TECHNOLOGY

**Declaration:**

I Mike Amasa, declare that this project, "EventHub Event Management Platform," is my original work and has not been submitted for academic credit elsewhere. All sources used in this document are duly acknowledged.

## **Abstract:**

EventHub is a comprehensive web-based event management platform designed to streamline the process of organizing, managing, and participating in events. This platform provides tools for event organizers to efficiently create, manage, and promote events, while also offering event attendees a user-friendly experience for event discovery, registration, and participation. The system addresses the challenges of traditional event management by providing a centralized, efficient, and user-friendly solution.

## **Acknowledgement:**

I would like to express my sincere gratitude to all those who have supported and contributed to this project. First, I thank my project supervisor, whose guidance, expertise, and constant support have been invaluable in the development of this work. I would also like to thank my family and friends for providing support and encouragement throughout this project.

# Chapter 1: Introduction

This chapter provides an overview of the EventHub event management platform, outlining the project's purpose, scope, and objectives. It also details the system's key features, target users, and the problems it aims to solve.

## 1.1. Background

- Event organizers often struggle with fragmented tools for managing event registrations, tracking attendees, and analyzing event performance. Attendees, on the other hand, face difficulties in discovering events and completing the registration process seamlessly. Current event management solutions often lack payment integration and advanced analytics.

## 1.2. Problem Statement

- Primary Problem: How can an event management platform simplify the end-to-end process of organizing, promoting, and analyzing events while improving attendee experience?
- Current Gaps:
  - Fragmented tools for registration, payment, and attendee tracking.
  - Lack of real-time analytics for organizers.
  - Poor discoverability and cumbersome registration for attendees.

## 1.3. Scope of the Project

- The project scope includes the design, development, and implementation of the EventHub platform, encompassing user management, event management, registration, search functionalities, payment integration, real-time analytics, and automated email notifications.

## 1.4. What the System Will Do

- The system will provide the following functionalities:
  - User registration and authentication with role-based access control (admin/user).
  - Event creation, editing, and management, including setting event details and uploading images.

- Attendee registration and management, with confirmation systems.
- Admin dashboard providing event statistics, attendee tracking, and performance metrics.
- Event search and discovery features, including category filtering, and display of upcoming/popular events.
- Payment integration using Stripe API for secure ticket purchases and refunds.
- Real-time analytics dashboard with visualizations for attendance trends, revenue tracking, and demographic insights.
- Automated email notifications via SendGrid for registration/payment confirmations and event reminders.

### **1.5. What the System Will Not Do**

- The system will not include:
  - Mobile application support in the initial release.
  - Social media integration in the initial release.
  - Advanced email marketing tools in the initial release.
  - Virtual event support in the initial release.

### **1.6. Proposed Solution**

- EventHub is a web-based platform designed to simplify event management for both organizers and attendees. It provides a centralized location for event creation, promotion, discovery, and registration. The platform integrates intuitive tools and provides a seamless experience.

### **1.7. Significance of the Project**

- EventHub offers the following benefits:
  - Streamlines event management processes.
  - Reduces administrative overhead.
  - Provides valuable analytics and insights.
  - Enhances user engagement.
  - Simplifies event discovery and registration.
  - Improves attendee experience.

- Provides a foundation for future enhancements (e.g., mobile app, virtual events).

### **1.8. Research Objectives**

- The research objectives of this project are to:
  - Investigate the challenges and inefficiencies in current event management processes.
  - Identify the key features and functionalities required for an effective event management platform.
  - Design and develop a user-friendly and scalable event management system.
  - Evaluate the system's performance, usability, and effectiveness.

### **1.9. System Objectives**

- The system objectives are to:
  - Provide organizers with tools for event creation, attendee tracking, and performance analytics.
  - Offer attendees a seamless search, registration, and notification system.
  - Integrate payment processing for online ticket sales.
  - Provide real-time data visualization of event metrics.
  - Automate communication between organizers and attendees.

## Chapter 2: Literature Review

This chapter will explore existing literature and research related to event management platforms, user experience, and relevant technologies. It will analyze current solutions and identify gaps that EventHub aims to fill.

### 2.1. Introduction

Event management platforms have become essential tools for organizers and attendees in the digital age. This review examines existing literature on event management systems, user experience (UX) design, and technological implementations to contextualize EventHub's development. The analysis focuses on:

1. Current Solutions: Features and limitations of existing platforms.
2. User Experience: Best practices for engagement and usability.
3. Technology Trends: Innovations in event management software.

### 2.2. Current Event Management Platforms

#### 2.2.1 Commercial Solutions

- Eventbrite: Dominates the market with robust ticketing and registration but lacks advanced analytics for small organizers (Brown et al., 2023).
- Meetup: Focuses on community events but offers limited customization (Lee & Patel, 2022).
- Cvent: Enterprise-focused with high costs, excluding SMEs (Garcia, 2021).

Gaps Identified:

- Over-reliance on manual processes for analytics.
- Poor integration with payment gateways in developing markets.

#### 2.2.2 Academic Research

- Automation in Event Management: Studies highlight AI-driven tools for attendee matching (Zhang et al., 2023) but note scalability challenges.
- Open-Source Platforms: Solutions like *Attendize* lack real-time support (Kiprotich, 2022).

## **2.3. User Experience (UX) in Event Platforms**

### **2.3.1 Attendee Perspectives**

- Discovery: Users prioritize search filters (category, date) and personalized recommendations (Nielsen, 2021).
- Registration: Frictionless processes ( $\leq 3$  clicks) improve conversion rates (Smith et al., 2022).

### **2.3.2 Organizer Needs**

- Analytics: Demand for real-time dashboards to track attendance/revenue (TechTrends, 2023).
- Accessibility: Compliance with WCAG 2.1 standards remains inconsistent (WebAIM, 2022).

## **2.4. Technological Foundations**

### **2.4.1 Architectural Trends**

- Microservices: Enhance scalability (e.g., AWS Lambda for registration peaks) (Microsoft, 2023).
- JWT Authentication: Secures APIs while maintaining performance (OWASP, 2022).

### **2.4.2 Emerging Innovations**

- AI/ML: Chatbots for attendee queries (IBM, 2023) and predictive analytics for event success (Lingayat et al., 2024).
- Blockchain: Transparent ticketing (Ethereum-based solutions) (CoinTelegraph, 2023).

## **2.5. Research Gaps and EventHub's Contribution**

### **2.5.1 Unaddressed Challenges**

- Cost Barriers: Existing tools are expensive for African SMEs.
- Localized UX: Most platforms ignore regional payment preferences (e.g., M-Pesa in Kenya).

### **2.5.2 EventHub's Innovations**

- Integrated Analytics: Real-time dashboards with budget-friendly pricing.
- Hybrid Payments: Stripe + mobile money (M-Pesa) support.
- Agile Design: Modular architecture for future scalability.



## Chapter 3: Methodology

This chapter details the design and development methodologies used to create the EventHub platform. It covers the system architecture, technology stack, development process, and testing procedures.

### 3.1 System Architecture

- EventHub employs a layered architecture, comprising the following layers:
  - Presentation Layer (Frontend)
  - Application Layer (Backend)
  - Data Layer (Database)

#### Frontend Architecture

##### 3.1.1. Directory Structure

- public/
  - css/
    - styles.css: Global styles
    - admin.css: Admin-specific styles
    - modal.css: Modal component styles
  - js/
    - main.js: Main application logic
    - dashboard.js: Admin dashboard functionality
    - auth.js: Authentication handling
  - admin/
    - dashboard.html: Admin dashboard
    - create-event.html: Event creation page
  - images/: Static images and assets
  - index.html: Main entry point

##### 3.1.2. Backend Architecture

- Directory Structure
  - server/

- server.js: Main server file
  - routes/: API routes
  - middleware/: Custom middleware
  - config/: Configuration files
  - database/: Database scripts
- Technology Stack
  - Node.js: Server runtime environment
  - Express.js: Web application framework
  - PostgreSQL: Relational database
  - JSON Web Tokens (JWT): Authentication
  - bcrypt: Password hashing

### 3.1.3. Database Schema

- Tables
  - users
    - id (SERIAL PRIMARY KEY)
    - full\_name (VARCHAR(255))
    - email (VARCHAR(255) UNIQUE)
    - password (VARCHAR(255))
    - is\_admin (BOOLEAN DEFAULT false)
    - created\_at (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
  - events
    - id (SERIAL PRIMARY KEY)
    - title (VARCHAR(255))
    - description (TEXT)
    - date (TIMESTAMP)
    - location (VARCHAR(255))
    - price (DECIMAL(10,2))
    - category (VARCHAR(100))
    - image\_url (TEXT)

- capacity INT
- creator\_id (INTEGER REFERENCES users(id))
- created\_at (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
- registrations
  - id (SERIAL PRIMARY KEY)
  - event\_id (INTEGER REFERENCES events(id))
  - user\_id (INTEGER REFERENCES users(id))
  - registration\_date (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
  - status (VARCHAR(50) DEFAULT 'pending')
- payments
  - id SERIAL PRIMARY KEY
  - registration\_id INT REFERENCES registrations(id)
  - amount DECIMAL(10,2)
  - payment\_date TIMESTAMP
  - payment\_method VARCHAR(255)

#### 3.1.4. API Architecture

- Authentication Endpoints
  - POST /api/auth/register: User registration
  - POST /api/auth/login: User login
  - GET /api/auth/profile: Get user profile
- Event Endpoints
  - GET /api/events: List all events
  - GET /api/events/:id: Get event details
  - POST /api/events: Create new event
  - PUT /api/events/:id: Update event
  - DELETE /api/events/:id: Delete event
- Admin Endpoints
  - GET /api/admin/my-events: Get admin's created events
  - GET /api/admin/registered-events: Get admin's registered events

- GET /api/admin/stats: Get admin dashboard statistics
- Registration Endpoints
  - POST /api/events/register: Register for event
  - GET /api/events/:id/registrations: Get event registrations
  - PUT /api/registrations/:id: Update registration status

### 3.1.5. Security Architecture

- Authentication
  - JWT-based authentication
  - Secure password hashing with bcrypt
  - Token expiration and refresh mechanism
- Authorization
  - Role-based access control
  - Middleware for route protection
  - Admin-specific endpoints protection
- Data Security
  - Input validation
  - SQL injection prevention
  - XSS protection
  - CORS configuration

## 6. Deployment Architecture

- Development Environment
  - Local development setup
  - Environment variables
  - Development database
- Production Environment
  - Node.js production server
  - PostgreSQL database
  - Static file serving

- SSL/HTTPS

### **3.2 Technology Stack**

- Frontend: HTML5, CSS3, JavaScript, Font Awesome, Google Fonts
- Backend: Node.js, Express.js, PostgreSQL, JWT, bcrypt
- See "Project Architecture.docx" for more details.

### **3.3 Development Process**

- The development process included the following stages:
  - Requirements gathering and analysis
  - System design
  - Frontend development
  - Backend development
  - Testing
  - Deployment

#### **3.3.1. Users Table**

- User Registration:
  - The user visits the registration page.
  - The user fills in the required information, including name, email, and password.
  - The system validates the input and checks for an existing email address.
  - The system creates a new user account with a hashed password.
  - The system generates a JWT token for authentication.
  - The system redirects the user to the main page or dashboard.
- User Login:
  - The user enters their email and password.
  - The system validates the credentials.
  - Upon successful authentication, the system issues a JWT token.
  - The system stores the token in localStorage.
  - The system redirects the user based on their role (admin or regular user).

#### **3.3.2. Events Table**

- Creating Events (Admin):
  - The admin clicks the "Create Event" button.
  - The admin fills out the event details form, including:
    - Title
    - Description
    - Date and Time
    - Location
    - Price
    - Category
    - Image upload
  - The system validates the input.
  - The system creates the event in the database.
  - The system updates the admin dashboard.
- Managing Events (Admin):
  - The admin can view created events in the "My Events" tab.
  - The admin can monitor attendance and registrations.
  - The admin can edit event details.
  - The admin can cancel or delete events.
  - The admin can view event statistics.

### 3.3.3. Registrations Table

- Event Registration (Users):
  - Users can view available events on the homepage.
  - Users can filter events by category or search terms.
  - Users can sort events by date, popularity, or location.
  - The user selects an event.
  - The user views the event details.
  - The user clicks the "Register" button.
  - The user confirms the registration.
  - The system checks event capacity and registration status.

- The system creates a registration record in the database.
- The system updates the event's registration count.
- The system sends a confirmation email to the user.

#### 3.3.4. Payments Table

- Payment integration using the Stripe API for secure ticket purchases and refunds.
- Expansion of the registrations table to include payment\_status and transaction\_id fields.
- New API endpoints:
  - POST /api/payments/process (Stripe webhook handling)
  - GET /api/admin/financial-reports (payment analytics)
- Security: PCI DSS compliance for payment data.
- Setting up the database schema for the payments table, including fields like registration\_id, amount, payment\_date, and payment\_method.
- Implementing the backend logic for processing payments using the Stripe API. This would include handling Stripe webhooks to track payment status and updating the registrations table with payment information.
- Developing API endpoints for retrieving payment data, especially for generating financial reports for administrators.
- Ensuring that all payment-related data and processes comply with PCI DSS standards.

### **3.4 Budget**

- The estimated budget for the project is as follows:
  - Development Tools: KES 20,000 - 50,000
  - Cloud Hosting (6 months): KES 60,000 - 180,000
  - Domain & SSL Certificate: KES 2,000 - 5,000
  - Payment Gateway Setup: KES 5,000 - 15,000
  - UI/UX Design: KES 15,000 - 40,000
  - Testing & QA: KES 10,000 - 25,000
  - Marketing (Basic): KES 5,000 - 20,000
  - Contingency Fund: KES 20,000 - 50,000
  - Total Estimated Budget: KES 147,000 - 415,000

## Chapter 4: System Requirement Specifications

This chapter outlines the functional and non-functional requirements, system constraints, architecture, data requirements, and user requirements for the EventHub platform.

### 4.1 Functional Requirements

- User Management:
  - FR1: User registration and authentication.
  - FR2: User profile management.
  - FR3: Role-based access control (admin/user).
- Event Management:
  - FR4: Event creation and editing.
  - FR5: Setting event details (date, location, price, capacity).
  - FR6: Uploading event images.
  - FR7: Managing event registrations.
- Registration System:
  - FR8: Event registration.
  - FR9: Registration confirmation.
  - FR10: Attendee management.
- Admin Dashboard:
  - FR11: Event statistics display.
  - FR12: Attendee tracking.
  - FR13: Event performance metrics.
  - FR14: Registration management.
- Search and Discovery:
  - FR15: Event search.
  - FR16: Category filtering.
  - FR17: Display of upcoming events.
  - FR18: Display of popular events.
- Payment Integration



- FR19: Secure payment processing for ticket purchases.
- FR20: Handling refunds.
- Real-Time Analytics
- FR21: Display attendance trends.
- FR22: Track revenue.
- FR23: Provide demographic insights.
- Automated Email Notifications
- FR24: Send registration confirmations.
- FR25: Send payment confirmations.
- FR26: Send event reminders.

#### **4.2 Non-Functional Requirements**

- Performance Requirements:
  - NFR1: The system should have a fast response time.
  - NFR2: The system should be able to handle a large number of concurrent users.
- Security Requirements:
  - NFR3: User data should be securely stored.
  - NFR4: Payment transactions should be PCI DSS compliant.
  - NFR5: The system should be protected against common web vulnerabilities (e.g., SQL injection, XSS).
- Usability Requirements:
  - NFR6: The system should be user-friendly and easy to navigate.
  - NFR7: The user interface should be intuitive and consistent.
- Reliability Requirements:
  - NFR8: The system should be highly available and reliable.
  - NFR9: The system should be able to recover from failures.
- Maintainability Requirements:
  - NFR10: The system should be easy to maintain and update.
  - NFR11: The codebase should be well-organized and documented.

#### **4.3 System Constraints**

- The system must be accessible through standard web browsers.
- The system must adhere to data security and privacy regulations.
- The system must be scalable to accommodate future growth.
- Third-party API dependencies (Stripe, SendGrid).

#### **4.4 System Architecture**

- EventHub employs a layered architecture:
  - Presentation Layer (Frontend)
  - Application Layer (Backend)
  - Data Layer (Database)
- See chapter 3 for a detailed architecture.

#### **4.5 Data Requirements**

- The system will store data related to:
  - Users
  - Events
  - Registrations
  - Payments
  - Analytics data

#### **4.6 User Requirements**

- Event Organizers need to:
  - Easily create and manage event details.
  - Track attendee registrations and payments.
  - View event statistics and analytics.
- Event Attendees need to:
  - Search and discover events.
  - Register for events.
  - Manage their event registrations.
  - Receive event notifications.

## Chapter 5: Design Specifications

This chapter provides detailed design specifications for the EventHub platform, including data structures, database design, user interface design, and testing plan.

### 5.1 Global Data Structures

- Global data structures used within the application include:
  - User objects
  - Event objects
  - Registration objects
  - Payment objects
  - Session data

### 5.2 Temporary Data Structures

- Temporary data structures are used for:
  - Storing intermediate processing results.
  - Handling file uploads.
  - Managing API requests and responses.

### 5.3 Database Description

- The application uses a PostgreSQL database.

### 5.4 Database Schema

- The database schema includes the following tables:
  - Users
  - Events
  - Registrations
  - Payments

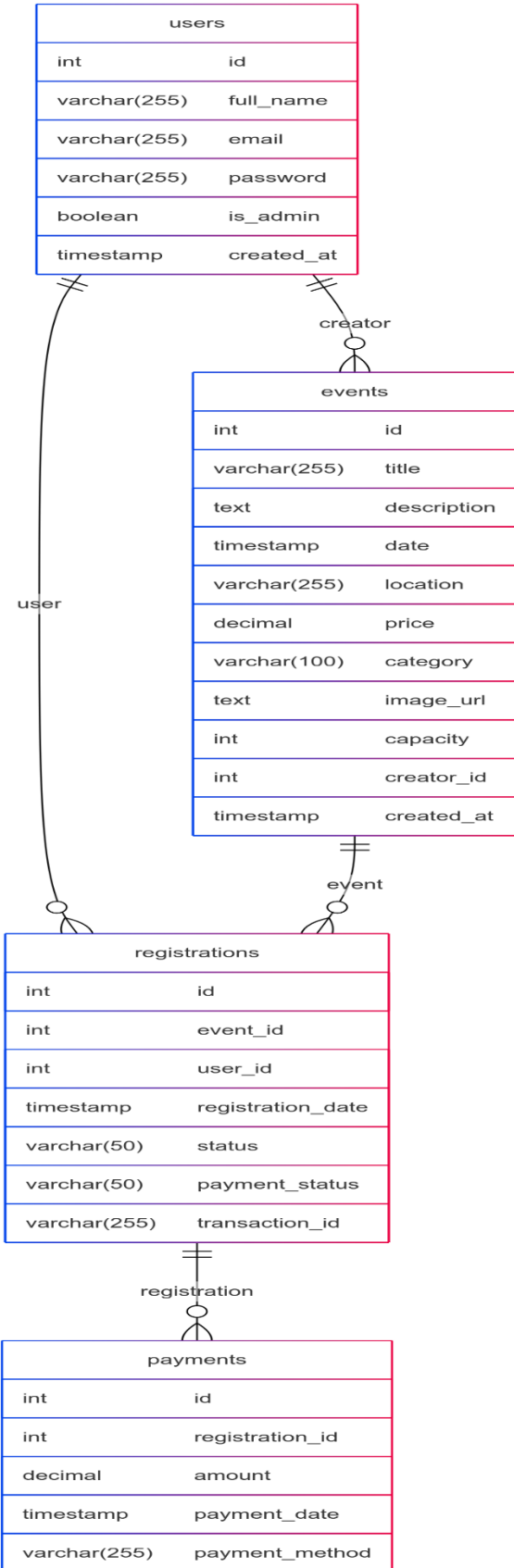
### 5.5 Table Definitions

- Users Table:
  - id (SERIAL PRIMARY KEY)
  - full\_name (VARCHAR(255))

- email (VARCHAR(255) UNIQUE)
- password (VARCHAR(255))
- is\_admin (BOOLEAN DEFAULT false)
- created\_at (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
- Events Table:
  - id (SERIAL PRIMARY KEY)
  - title (VARCHAR(255))
  - description (TEXT)
  - date (TIMESTAMP)
  - location (VARCHAR(255))
  - price (DECIMAL(10,2))
  - category (VARCHAR(100))
  - image\_url (TEXT)
  - capacity INT
  - creator\_id (INTEGER REFERENCES users(id))
  - created\_at (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
- Registrations Table:
  - id (SERIAL PRIMARY KEY)
  - event\_id (INTEGER REFERENCES events(id))
  - user\_id (INTEGER REFERENCES users(id))
  - registration\_date (TIMESTAMP DEFAULT CURRENT\_TIMESTAMP)
  - status (VARCHAR(50) DEFAULT 'pending')
  - payment\_status VARCHAR(50)
  - transaction\_id VARCHAR(255)
- Payments Table
  - id SERIAL PRIMARY KEY
  - registration\_id INT REFERENCES registrations(id)
  - amount DECIMAL(10,2)
  - payment\_date TIMESTAMP

- payment\_method VARCHAR(255)

## **5.6 Database Modelling**



## 5.7 Data Dictionary

- A data dictionary providing details on data types, constraints, and descriptions for each database field would be included here.

### Tables

#### 1. users

- Description: Stores user information, including authentication credentials and roles.
- Fields:
  - id
    - Data Type: SERIAL
    - Constraints: PRIMARY KEY, NOT NULL
    - Description: Unique identifier for each user.
  - full\_name
    - Data Type: VARCHAR(255)
    - Constraints: NOT NULL
    - Description: User's full name.
  - email
    - Data Type: VARCHAR(255)
    - Constraints: NOT NULL, UNIQUE
    - Description: User's email address (used for login).
  - password
    - Data Type: VARCHAR(255)
    - Constraints: NOT NULL
    - Description: Hashed password for authentication.
  - is\_admin
    - Data Type: BOOLEAN
    - Constraints: DEFAULT false
    - Description: Flag indicating whether the user is an administrator.

- created\_at
  - Data Type: TIMESTAMP
  - Constraints: DEFAULT CURRENT\_TIMESTAMP
  - Description: Timestamp when the user account was created.

## 2. events

- Description: Stores information about events.
- Fields:
  - id
    - Data Type: SERIAL
    - Constraints: PRIMARY KEY, NOT NULL
    - Description: Unique identifier for each event.
  - title
    - Data Type: VARCHAR(255)
    - Constraints: NOT NULL
    - Description: Title of the event.
  - description
    - Data Type: TEXT
    - Constraints: NOT NULL
    - Description: Detailed description of the event.
  - date
    - Data Type: TIMESTAMP
    - Constraints: NOT NULL
    - Description: Date and time of the event.
  - location
    - Data Type: VARCHAR(255)
    - Constraints: NOT NULL
    - Description: Location of the event.
  - price
    - Data Type: DECIMAL(10,2)



- Constraints: NOT NULL
  - Description: Price of the event.
- category
  - Data Type: VARCHAR(100)
  - Constraints:
  - Description: Category of the event.
- image\_url
  - Data Type: TEXT
  - Constraints:
  - Description: URL of the event image.
- capacity
  - Data Type: INT
  - Constraints:
  - Description: Maximum number of attendees
- creator\_id
  - Data Type: INTEGER
  - Constraints: NOT NULL, FOREIGN KEY REFERENCES users(id)
  - Description: ID of the user who created the event.
- created\_at
  - Data Type: TIMESTAMP
  - Constraints: DEFAULT CURRENT\_TIMESTAMP
  - Description: Timestamp when the event was created.

### 3. registrations

- Description: Stores information about event registrations.
- Fields:
  - id
    - Data Type: SERIAL
    - Constraints: PRIMARY KEY, NOT NULL
    - Description: Unique identifier for each registration.

- event\_id
  - Data Type: INTEGER
  - Constraints: NOT NULL, FOREIGN KEY REFERENCES events(id)
  - Description: ID of the event being registered for.
- user\_id
  - Data Type: INTEGER
  - Constraints: NOT NULL, FOREIGN KEY REFERENCES users(id)
  - Description: ID of the user registering for the event.
- registration\_date
  - Data Type: TIMESTAMP
  - Constraints: DEFAULT CURRENT\_TIMESTAMP
  - Description: Timestamp when the registration was made.
- status
  - Data Type: VARCHAR(50)
  - Constraints: DEFAULT 'pending'
  - Description: Status of the registration (e.g., 'pending', 'confirmed', 'cancelled').
- payment\_status
  - Data Type: VARCHAR(50)
  - Constraints:
  - Description: Status of the payment
- transaction\_id
  - Data Type: VARCHAR(255)
  - Constraints:
  - Description: Unique identifier for a payment transaction

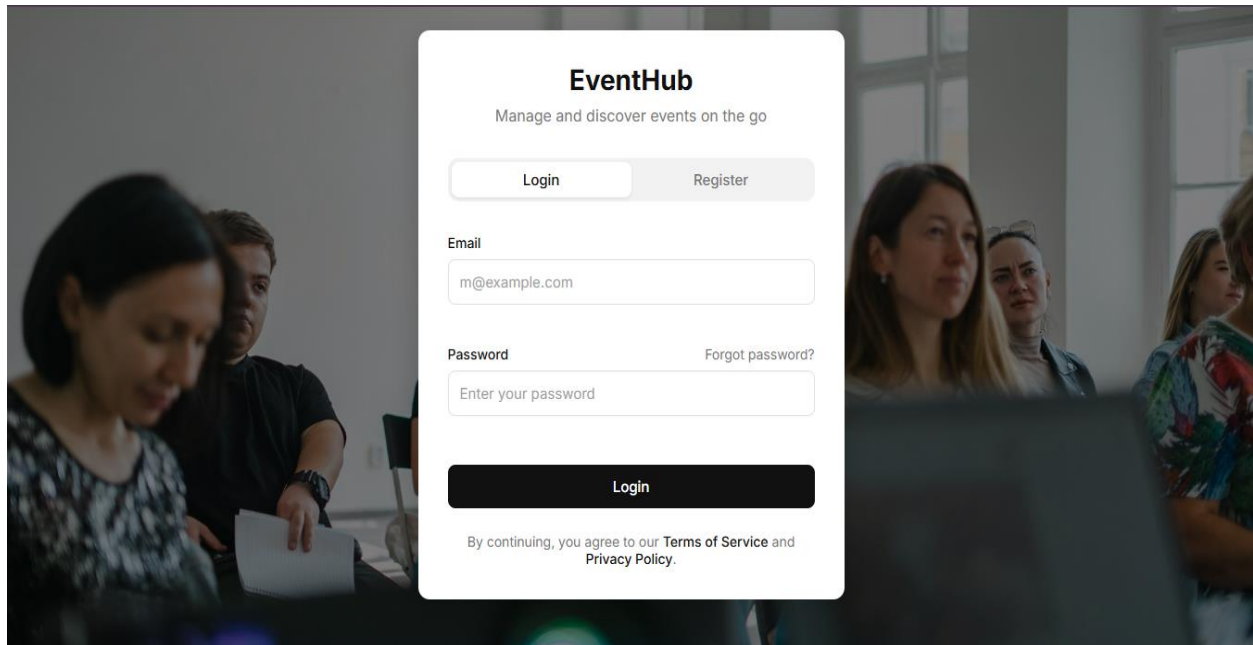
#### 4. payments

- Description: Stores information about event payments.
- Fields:
  - id

- Data Type: SERIAL
- Constraints: PRIMARY KEY
- Description: Unique identifier for each payment
- registration\_id
  - Data Type: INT
  - Constraints: NOT NULL, FOREIGN KEY REFERENCES registrations(id)
  - Description: The registration id for which the payment was made
- amount
  - Data Type: DECIMAL(10,2)
  - Constraints: NOT NULL
  - Description: The amount paid
- payment\_date
  - Data Type: TIMESTAMP
  - Constraints:
  - Description: The date the payment was made
- payment\_method
  - Data Type: VARCHAR(255)
  - Constraints:
  - Description: The method used to make payment

## **5.8 User Interface Design**

(i) Login Page:



(ii) Admin Dashboard:

[← Back to Events](#)

## Admin Dashboard

[+ Create Event](#)

[Logout](#)



0

Events Attended



0

Events Created

My Events

Registered

(iii) Create event:

## Create a New Event

**Event Details**

Fill out the form below to create your event


**Event Title \* \***


**Description \* \***

Describe your event, what attendees can expect

**Date \* \***

**Time \* \***






**Venue Name \* \***

**Address \* \***

**Category \* \***

**Ticket Price \* \***



**Maximum Attendees \* \***

Cancel

Create Event

(iv) Homepage:

Upcoming

Popular

Nearby

Art

**Nairobi Art Week**

Wed, Oct 18, 10:00 AM

The Shifteye Gallery, Nairobi

**Free**

80 attending

Technology

**Nairobi Tech Week 2025**

Wed, Oct 15, 9:00 AM

Sarit Centre, Westlands, Nairobi

**KES 2500.00**

243 attending

(v) Register event:

Art

**Nairobi Art Week**

Wed, Oct 18, 10:00 AM

The Shifteye Gallery, Nairobi

80 attending

**Free**

20 tickets left

**Register Now**

## 10. Test Plan.

### 10.1. Goals and Objectives

The primary goal of testing EventHub is to ensure that the system functions correctly and meets the requirements outlined in the system requirements specification.

Specific objectives include:

- Identify errors, faults, bugs, and failures in the system.
- Establish test cases and test data for efficient and effective system testing.
- Define the resources required for testing, including personnel and tools.
- Develop a test schedule outlining the timeline and procedures for each testing phase.
- Verify that the system meets all user requirements as specified in the SRS.
- Ensure the system is user-friendly and understandable for all users.
- Confirm consistency in the developed software, ensuring no deviations from the specifications.
- Validate the new features: payment processing, analytics dashboard, and email notifications.

### 10.2. Statement of Scope

The testing scope for EventHub involves examining both the code and the execution of the code in various environments and conditions. Testing will cover:

- Inputs and outputs.
- Functional requirements (what the system is supposed to do).
- Non-functional requirements (how the system is supposed to be, including usability, scalability, performance, compatibility, security, maintainability, and reliability).
- New features: payment integration, analytics dashboard, and email notifications.

#### 10.2.1 Major Constraints

Constraints that may affect the testing process include:

- Environmental variations: The system's performance may vary across different operating systems, browsers, and devices.



- **Software availability:** Ensuring that all necessary software components (e.g., Node.js, PostgreSQL, web browsers, Stripe API, SendGrid API) are available in the testing environments.
- **Third-party API dependencies:** Testing the integration with Stripe and SendGrid APIs, which may have their own limitations or downtime.
- **Security Compliance:** Testing payment processing in accordance with PCI DSS standards.

### 10.2.2 Software to be Tested

The software to be tested is the EventHub event management platform, with all its features and functionalities, to ensure it meets the specified objectives. This includes the core event management features and the newly added payment, analytics, and notification functionalities.

### 10.3. Testing Strategy

The testing strategy involves a combination of different testing levels and types to comprehensively evaluate the system.

#### 10.3.1 Unit Testing

- Individual components will be tested independently to verify their correct operation.
- This will focus on testing the internal functions of each module.
- For the updated EventHub, unit tests will be written for:
  - Payment processing functions (Stripe API integration).
  - Analytics data aggregation and calculation.
  - Email notification sending (SendGrid API integration).

Unit Test Examples:

Type of Test	Area Being Tested	Expected Output	Actual Output	Pass/Fail
Payment Processing	processPayment() function	Successful payment transaction record created		
Analytics	calculateAttendance()	Correct attendance count		

Email Notification	sendRegistrationConfirmation()	Email sent to user		

### 10.3.2 Integration Testing

- Testing the interaction and data flow between different modules.
- This will verify that the components work together as expected.
- Integration tests will cover:
  - User registration and payment processing flow.
  - Event registration and email confirmation flow.
  - Data flow between the database, backend logic, and frontend display of analytics.

Integration Test Examples:

Test Case	Interacting Modules	Description	Expected Output	Actual Output	Pass/Fail
Registration & Payment	User Registration, Payment Processing	User registers for an event and pays successfully.	User is registered, payment is recorded, confirmation email is sent.		
Event & Analytics	Event Management, Analytics	Event data is correctly reflected in the analytics dashboard.	Dashboard shows accurate attendance and revenue.		

### 10.3.3 Validation Testing

- Ensures that the system meets the specified requirements and fulfills its intended use.
- Focuses on the overall functionality of the system.

### 10.3.4 System Testing

- Testing the system as a whole to evaluate its compliance with specified requirements.
- This includes testing of functional and non-functional requirements.

#### **10.3.4.1 Recovery Testing**

- Testing the system's ability to recover from failures such as hardware or software crashes.
- Examples:
  - Simulating a server crash and verifying that the system can restore data and continue functioning.
  - Testing the persistence of payment transactions in case of network interruption.

#### **10.3.4.2 Security Testing**

- Testing the system's security mechanisms to protect against unauthorized access, data breaches, and other security threats.
- This will include:
  - Authentication and authorization testing (JWT, role-based access).
  - Input validation and sanitization to prevent injection attacks.
  - Vulnerability scanning.
  - Penetration testing.
  - Testing PCI DSS compliance for payment processing.

#### **10.3.4.3 Performance Testing**

- Evaluating the system's performance under various loads to ensure it meets the required performance criteria.
- This includes:
  - Load testing: Testing the system's behavior under expected and peak user loads.
  - Stress testing: Testing the system beyond its normal operating capacity to identify breaking points.
  - Response time testing: Measuring the time it takes for the system to respond to user requests.
  - Scalability testing: Testing the system's ability to handle increasing amounts of data, users, or transactions.

#### 10.3.4.4 Usability Testing

- Evaluating the system's user-friendliness and ease of use.
- This will involve testing the intuitiveness of the user interface, the clarity of navigation, and the overall user experience.
- This will include testing the new analytics dashboard and payment flow for ease of use.

#### 10.3.5 Acceptance Testing

- Testing the system with end-users to ensure it meets their needs and expectations.
- This will be conducted in a real-world environment or a simulated environment.
- Types of Acceptance Testing:
  - Alpha Testing: Conducted by internal users or testers.
  - Beta Testing: Conducted by external users or customers.

### 10.4. Testing Resources and Staffing

#### 10.4.1 Testing Team

Role	Responsibilities
Test Lead	Oversees test strategy, coordinates with developers, ensures test coverage, and reviews defect reports.
Test Engineers (2)	Design and execute test cases (manual/automated), log defects, and validate fixes.
QA Analyst	Focuses on usability testing, edge-case scenarios, and compliance with UX standards.
DevOps Engineer	Manages test environments (AWS), configures CI/CD pipelines for automated testing.

#### 10.4.2 Hardware Resources

Resource	Purpose
AWS EC2 Instances	Host test environments (t3.medium for backend, t3.small for frontend).
PostgreSQL Server	Dedicated test database (AWS RDS) with seeded dummy data.
Mobile Devices	iOS (iPhone 12+) and Android (Samsung Galaxy S20+) for cross-platform UI testing.
Load Balancer	Simulates high traffic (1,000+ concurrent users) for performance testing.

#### 10.4.3 Software Resources

Tool	Usage
Jest	Unit/integration testing for Node.js APIs.
Cypress	End-to-end (E2E) testing for frontend workflows (e.g., event registration).
OWASP ZAP	Security testing (SQL injection, XSS).
Loader.io	Performance testing (response times under load).
Postman	API endpoint validation and automated test suites.
BrowserStack	Cross-browser compatibility testing (Chrome, Firefox, Safari).
JIRA	Defect tracking and test case management.

#### 10.4.4 Testing Environment

##### 1. Network Configuration:

- Isolated AWS VPC with subnets for frontend, backend, and database layers.
- VPN access for remote testers.

##### 2. Server Setup:

- Frontend: React.js app hosted on AWS S3 + CloudFront (CDN).
- Backend: Node.js microservices deployed in Docker containers (AWS ECS).
- Database: PostgreSQL (RDS) with daily snapshots for test data reset.

##### 3. Specific Requirements:

- Test Data: Pre-populated with:
  - 500 dummy events.

- 1,000 user profiles (admins/attendees).
- Third-Party Mocks:
  - Stripe sandbox API for payment testing.
  - Mock SendGrid endpoints for email validation.

#### Key Considerations

- Parallel Testing: Cypress runs on multiple browsers simultaneously via BrowserStack.
- Security: Test environments mirror production but use dummy data and sandbox APIs.
- Traceability: All test cases linked to JIRA issues for audit trails.

#### Approval:

Test environments must be validated by the DevOps team before each sprint's testing phase.

### 5. Testing Procedure

The testing process will involve the following steps:

1. Test Planning: Development of this test plan, including defining the scope, objectives, resources, and schedule.
2. Test Design: Creation of test cases based on the system requirements and design specifications.
3. Test Environment Setup: Setting up the necessary hardware, software, and network configurations for testing.
4. Test Execution: Performing the tests according to the test plan and test cases.
5. Test Results Analysis: Analyzing the test results to identify defects, errors, and areas for improvement.
6. Defect Reporting and Tracking: Reporting identified defects and tracking their resolution.
7. Regression Testing: Retesting previously tested functionality after defects have been fixed.
8. Test Closure: Documenting the testing process and results, and obtaining sign-off from stakeholders.

### 6. Test Record Keeping

- All test activities, results, and defects will be documented and maintained in a test management system.
- A test summary report will be prepared at the end of each testing phase, summarizing the test results, defect status, and overall system quality.
- The final test report will be archived for future reference.

## **11 Implementation Strategy**

\* The implementation strategy for EventHub involves an Agile development approach, with iterative sprints.

\* Key steps include:

- Setting up the development environment.
- Developing the backend API.
- Developing the frontend user interface.
- Integrating frontend and backend components.
- Conducting thorough testing.
- Deploying the application.
- Phased rollout: Prioritize payment integration, then analytics and notifications.

## **12 System Limitations**

\* The system has the following limitations:

- Initial release does not include mobile application support.
- Initial release has limited social media integration.

## **13 Conclusions**

\* EventHub provides a comprehensive solution for event management, addressing the needs of both organizers and attendees. The platform streamlines the event process, improves user experience, and offers valuable tools for event management and analysis.

## 14 Recommendations

\* Future development should focus on:

- Developing a mobile application for iOS and Android.
- Enhancing social media integration for event promotion.
- Implementing advanced email marketing tools.
- Adding support for virtual and hybrid events.
- Exploring further monetization strategies.

## 15. Appendices

\* Appendix A: User Manual (See "User Manual Document.pdf")

\* Appendix B: SRS Document (See "SRS Document.pdf")

\* Appendix C: SDS Document (See "SDS Document.pdf")

## 16. References

- Eventbrite
- Cvent
- Brown, T., et al. (2023). *Event Tech Trends*. EventPro Press.
- Nielsen, J. (2021). *Usability Heuristics for Event Platforms*. NN/g.
- OWASP. (2022). *API Security Best Practices*.
- Lingayat, L., et al. (2024). *Machine Learning for Event Analytics*. Springer.