

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Code Documentation | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Elastic Event Platform |  | | | |
|  | | | | A Scalable and Fault-Tolerant Cloud-Based Event Management Solution—Software and Data Engineering—Mini Project  |  |  | | --- | --- | | Name | Roll Number | | Prateek Singhal | M22AIE215 | | Aryan Kumar | M23CSA510 | | Harsh Parashar | M22AIE210 | |  | | | |
|  | | |  | | |  | | |

Table of Contents

[Code Documentation 1](#_Toc178628436)

[Elastic Event Platform 1](#_Toc178628437)

[A Scalable and Fault-Tolerant Cloud-Based Event Management Solution 1](#_Toc178628438)

[— 1](#_Toc178628439)

[Software and Data Engineering 1](#_Toc178628440)

[— 1](#_Toc178628441)

[Mini Project 1](#_Toc178628442)

[Introduction 3](#_Toc178628443)

[Scenarios (Use Case View) 3](#_Toc178628444)

[Logical View (Design and Structure) 3](#_Toc178628445)

[Architectural Patterns 4](#_Toc178628446)

[Classes 5](#_Toc178628447)

[Development View (Implementation) 6](#_Toc178628448)

### Introduction

The Cloud-Based Event Management System lets you create, run, book tickets, and pay securely for events. For multi-tenant environments, it guarantees scalability, fault tolerance, and effective performance by using cloud concepts including auto-scaling, containerizing, and load balancing.

Github link for the project is <https://github.com/pscss/event-management>

Project deployment link is (endpoint) : <https://34.134.2.203:8001/docs#>

### Scenarios (Use Case View)

**Actors**

* **Event Manager (Admin User)**: Can create, update, and manage events.
* **Normal User**: Can browse events, book tickets, and make payments.

**Use Cases:**

1. **User Registration**:
   * Admin User Registration: Provides user and company details.
   * Normal User Registration: Provides user details.
2. **Event Management (Admin User)**:
   * Create Event
   * Update Event
   * View Event Details
3. **Event Browsing (Normal User)**:
   * Search Events
   * View Event Details
4. **Ticket Booking (Normal User)**:
   * Select Event
   * Choose Quantity
   * Book Tickets
5. **Payment Processing (Normal User)**:
   * Make Payment
   * Receive Payment Confirmation

### Logical View (Design and Structure)

1. **Models (Data Layer):**

* User
* Company
* Event
* Booking
* Payment

1. **Data Access Layer (DAL):**

* UserDAL
* CompanyDAL
* EventDAL
* BookingDAL
* PaymentDAL

1. **REST API Endpoints:**

* User Endpoints
* Event Endpoints
* Booking Endpoints
* Payment Endpoints

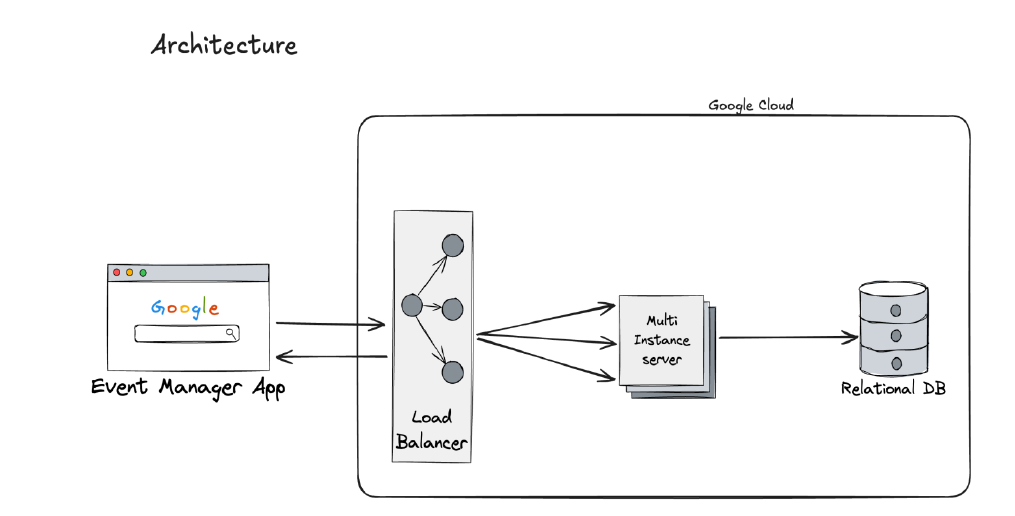
**Relationships:**

* User can have many Bookings.
* Booking is associated with one Event and one User.
* Event can have many Bookings.
* Payment is associated with one Booking.
* User may belong to one Company (if Admin).

### Architectural Patterns

1. **Model-View-Controller (MVC) Pattern**

* Models: Represent the data and business logic (ORM models using SQLAlchemy).
* Views: API responses in a web API context.
* Controllers: FastAPI endpoints that handle HTTP requests.



1. **Repository Pattern (Implemented in DAL)**

* Each DAL class acts as a repository for its respective model, encapsulating data access logic.

### Classes

***User***:

**Attributes:** name, email, country\_code, phone\_number, role, company\_id, keycloak\_id, username

**Relationships**: Belongs to Company, has many Bookings

***Company:***

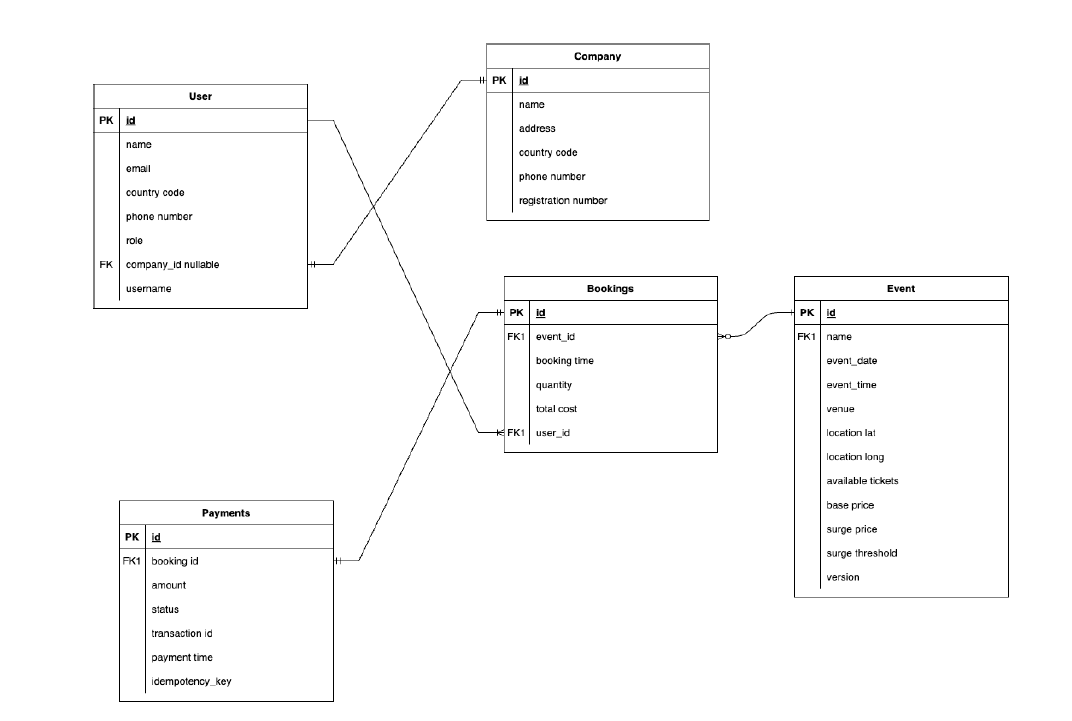
**Attributes:** name, address, email, country\_code, phone\_number, registration\_number

**Relationships:** Has many Users

**Event:**

**Attributes:** name, event\_date, event\_time, venue, location\_lat, location\_long, available\_tickets, base\_price, surge\_price, surge\_threshold, version

**Relationships:** Has many Bookings



**Booking:**

**Attributes:** event\_id, user\_id, booking\_time, quantity, total\_cost

**Relationships:** Belongs to User, belongs to Event, has many Payments

**Payment:**

**Attributes:** booking\_id, amount, status, transaction\_id, payment\_time, idempotency\_key

**Relationships:** Belongs to Booking

### Development View (Implementation)

#### Code Organization:

* **Packages/Folders**:
* **event\_manager**:
  + api: FastAPI endpoint definitions.
  + models: SQLAlchemy ORM models.
  + schemas: Pydantic models for data validation.
  + dal: Data Access Layer and business logic.
  + core: Configuration and utility functions.
  + main.py: Application entry point.
  + config.py: Configuration settings.

**Technologies and Frameworks:**

* **Programming Language**: Python 3.10
* **Web Framework**: FastAPI
* **ORM**: SQLAlchemy (with async support)
* **Database**: PostgreSQL
* **Asynchronous Programming**: asyncio
* **IDE**: Visual Studio Code

**Development Tools:**

* **Package Management**: Poetry
* **Version Control**: Git
* **Testing**: Pytest, pytest-asyncio
* **Linters and Formatters**: flake8, black
* **Virtual Environments**: Managed by Poetry

#### Physical View (Deployment)

**Platform:** Google Cloud Platform (GCP)

**Deployment Methods:**

* **Containers**: Docker for containerization.
* **Virtual Machines**: Deployed on GCP Compute Engine VMs.

**Infrastructure Components:**

1. **Load Balancer**: Distributes traffic across multiple application instances.
2. **Application Instances**:
   * Docker Containers running the FastAPI application.
   * Deployed across multiple VMs or container instances.
3. **Database**:
   * PostgreSQL, hosted on a managed service or VM.
4. **Payment Gateway**:
   * External service (e.g., Stripe) for processing payments.

#### Infrastructure Components

1. **Load Balancer**

* Distributes traffic to multiple instances of the application.
* Ensures high availability and scalability.

1. **Application Instances**

* Docker Containers: Running the FastAPI application.
* Deployed across multiple VMs or container instances.

3. **Database**

* PostgreSQL: Hosted on a managed service or VM.
* Accessible by application instances within the same network.

4. **Payment Gateway**

* External service (i.e., Stripe) for processing payments.
* Communicates over HTTPS.

5**. Nodes:**

* *Load Balancer* 
  + Connected to the Internet.
  + Forwards requests to Application Instances.
* *Application Server(s)* 
  + Docker containers running the FastAPI app.
  + Connected to the Database and Payment Gateway.
* *Database Server* 
  + PostgreSQL database instance.
* *Payment Gateway* 
  + External service accessible over the Internet.

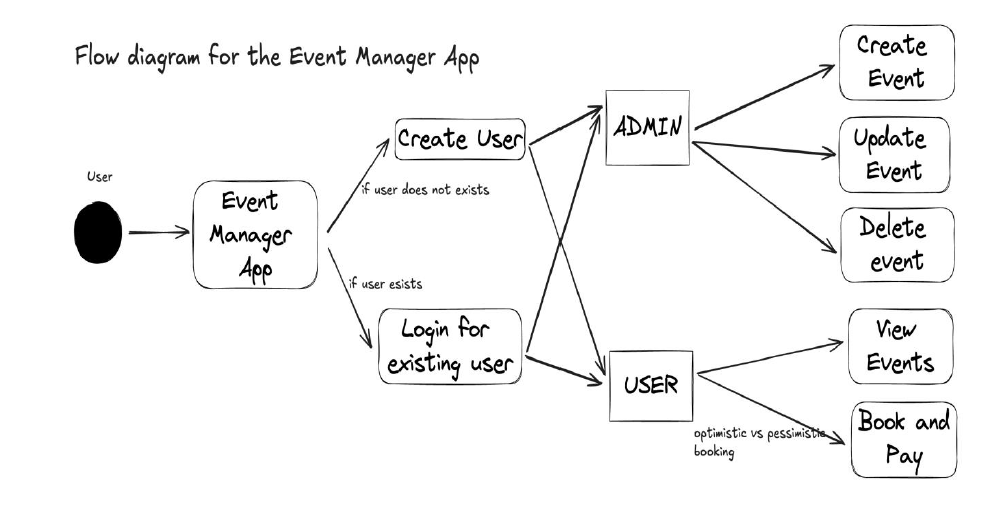
6. **Cross-Cutting Concerns**

* **Security** 
  + Authentication and Authorization
  + Keycloak: Opensource Identity and Access Management.
  + Handles user authentication.
  + Manages user roles (Admin, User).
  + Encryption
  + Communication over HTTPS.
  + Sensitive data encrypted at rest (database encryption features).
* **Error Handling and Logging** 
  + Exception Handling
  + Custom exceptions for business logic errors.
  + Global exception handlers in FastAPI.
* **Logging** 
  + Standard logging using Python's `logging` module.

**7. Configuration Management**

* **Environment Variables** 
  + Sensitive configurations (e.g., database credentials) managed via environment variables.
* **Configuration Files** 
  + Separate configuration files for different environments (development, staging, production).

#### Process Flow



#### Architectural Patterns

* ModelViewController (MVC)
* Separates concerns, making the application easier to maintain and scale.
* Models: Data representations and business logic.
* Views: API responses.
* Controllers: API endpoints handling requests.
* Repository Pattern
  + DAL Layer abstracts data access.
  + Promotes loose coupling between the business logic and data storage.

**Design Patterns**

* Dependency Injection
* Utilized via FastAPI's `Depends` mechanism.
* Facilitates testing and modularity.
* Singleton Pattern
  + Applied for configurations or shared resources like database connections.
* Asynchronous Patterns
  + Leveraging async/await for nonblocking I/O operations.
  + Improves performance and scalability under concurrent load.

**Enterprise Integration Patterns**

* Gateway Pattern
  + The application acts as a gateway to external services like the Payment Gateway.
  + Handles communication and error handling with external APIs.