**Event Management Platform - Complete Documentation**

1. **Project Overview**

This project is a Flask-based event management system designed to allow users to create, manage, and participate in events. It includes functionality for ticketing, email notifications, and payment integration. The system uses a modular approach to ensure maintainability and scalability

1. **Technology Stack**
   * Backend: Flask, Flask-SQLAlchemy, Flask-Mail, Flask-WTF-
   * Database: MySQL (with SQLAlchemy ORM)-
   * Frontend: HTML, CSS (not included in source files)- Email:
   * Flask-Mail (SMTP configured)- Payment: Stripe API integration-
   * Others: Redis (for caching/rate-limiting), Gunicorn (for deployment)
2. **Modules and Description**

1.***create\_database.py***:- Creates the MySQL database 'event\_management'.- Establishes initial connection to the MySQL server.

2. ***setup\_database.py***:- Initializes SQLAlchemy models and creates tables: \* User, Event, TicketType, PromoCode, Attendee, Payment, Referral

3. ***app.py*:-** Central Flask application with models, routes, and configurations.- Manages user sessions, event registration, ticket purchasing, and Stripe integration.

4**. *add\_ticket.py***:- Adds new ticket types to an existing event.- Requires event ID, ticket name, price, and quantity.

5. ***check\_events.py***:- Displays all created events along with their ticket types.

6. ***update\_event.py***:- Updates existing event details including: \* Description, start/end time, venue, max attendees.

7*.* ***test\_email.py***:- Sends a test email to confirm that Flask-Mail is properly configured.

8. ***app.log*:-** Logs application activity and errors for debugging.

**IV. Step-by-Step Workflow & Algorithms**

**Step 1: Database Setup**

*Algorithm:*

1. Run create\_database.py to create 'event\_management' DB
2. Run setup\_database.py to define and create tables.

**Step 2: Create Event**

*Algorithm:*

1. User registers or logs in via web form.

2. User fills event creation form with details. 3. Event is stored in the Event table.

**Step 3: Add Ticket Types**

*Algorithm:*

1. Organizer specifies event ID, ticket name, price, and quantity.

2. Script adds the ticket type to the DB using SQLAlchemy. **Step 4: Update Event Info**

*Algorithm:*

1. Fetch event using event ID.

2. Modify fields like description, venue, date/time.

3. Save changes using db.session.commit()

**Step 5: Email Testing**

*Algorithm:*

1. test\_email.py sends a sample email via Flask-Mail.

2. Uses app context and SMTP settings.

**Step 6: Check Events**

*Algorithm:*

1. check\_events.py queries all Event and TicketType objects. 2. Prints details in a structured format.

**V. Requirements and Deployment**:

* Dependencies listed in requirements.txt- Flask, Flask-Mail, SQLAlchemy, Stripe, gunicor
* Deployment via gunicorn with Redis for production

**VI. Conclusion:**

*This platform provides a complete, modular system to manage events, tickets, and participants. With built-in email and payment processing, it's ready for production use.*