## **🧪 Sr. Python Developer Assignment**

### **Objective:**

Build a **Mini Event Management System** API with a focus on clean architecture, scalability, and data integrity.

### **📌 Problem Statement:**

You're tasked with building a backend for a simplified **Event Management System**. Users should be able to create events, register attendees, and view attendee lists per event.

### **🎯 Requirements:**

#### **API Endpoints to Implement:**

1. POST /events  
   * Creates a new event with fields: name, location, start\_time, end\_time, max\_capacity
2. GET /events  
   * Lists all upcoming events
3. POST /events/{event\_id}/register  
   * Registers an attendee (name, email) for a specific event
   * Prevents overbooking (should not exceed max\_capacity)
   * Prevent duplicate registrations for the same email
4. GET /events/{event\_id}/attendees  
   * Returns all registered attendees for an event

### **🛠️ Technical Expectations:**

* Use **Python** with **FastAPI**, **Flask**, or **Django**
* Use a real DB like **PostgreSQL** or **SQLite** (ORM preferred: SQLAlchemy/Django ORM)
* Follow **MVC or clean architecture principles**
* Add **basic input validations** and meaningful **error messages**
* Maintain **separation of concerns** (services, models, routes)
* Use **async** wherever applicable (FastAPI users)
* Timezone management: Even created in IST and on change of timezone all the slots should be changed accordingly
* Bonus:  
  + Implement pagination on attendee lists
  + Write unit tests using pytest or unittest
  + Add Swagger/OpenAPI documentation

### **📦 Deliverables:**

* A GitHub repo or zipped project folder with:  
  + Source code
  + README with setup instructions, assumptions, and sample API requests (Postman or cURL)
  + Database schema (migration or SQL file)

Submit a Loom Video with a walkthrough of the entire assignment (Mandatory)  
Timeline: Submit within 3 working days

### 

### **✅ Evaluation Criteria:**

* Code quality, modularity, and architecture
* API design, performance, and correctness
* Handling of edge cases (e.g., max capacity, duplicates)
* Use of best practices (DRY, naming, structure)
* Bonus: Async implementation, tests, documentation