Technical Task [up to 6h]: Event Booking API - The candidate can use any framework or programming language of their choice.

Objective
Design and develop a RESTful API for an **Event Booking System** that allows users to manage event bookings while ensuring proper database design, request validation, and coding standards.

Requirements

The API should support the following functionalities:

- **Managing Events**
  - Users should be able to create, update, delete, and list events.
- **Managing Attendees**
  - Users should be able to register attendees and manage their information.
- **Booking System**
  - Users should be able to book an event.
  - The system should prevent overbooking and duplicate bookings.
- **Authentication & Authorization** (Implementation **not required**, only mention how it would be structured)
  - Assume that API consumers must be **authenticated** to manage events.
  - Attendees should be able to register without authentication.

Technical Expectations

- **Database Schema Design:** Define a relational schema to support the requirements.
- **Data Validation:** Ensure incoming requests are properly validated.
- **Serialization & Normalization:** Structure API responses in a clean and maintainable way.
- **Design Patterns & Best Practices:** Apply appropriate architectural patterns.
- **Error Handling:** Implement proper error responses for invalid requests.
- **Testing:** Provide unit and integration tests.
- **Documentation:** Include a README.md with setup instructions.

Guidelines

- The API **must** support event bookings, and the database should be structured accordingly.
- Locations should be **country-based** rather than specific addresses.
- Users should not be able to book the same event multiple times.
- Consider capacity limitations when booking an event.
- The API should return structured, meaningful responses.

The completed solution must be stored in a public repository (e.g., GitHub, GitLab, or Bitbucket) or a private repository with an invitation provided to access it.

Bonus (Optional Enhancements)

- Pagination and filtering for listing events.
- API documentation (Swagger/Postman).
- Docker support for easy deployment.

Evaluation Criteria

This task is designed to assess:

- **Database schema design** (relationships, constraints, and normalization).
- **Application architecture** (modularity, clean code, separation of concerns).
- **Implementation of validation and error handling.**
- **Use of design patterns and best practices.**
- **Testing approach (expecting meaningful test coverage).**

Technical task [optional bonus task – up to 2h]: Serverless Notification Service Architecture

Notice that **only architecture diagram is required** to illustrate the system's components, data flow, and interactions. No implementation or deployable code is required.

Objective

Design a **serverless notification service** architecture that processes events and delivers notifications through multiple channels, including **WebSockets, mobile push notifications, and email**. The system should be event-driven and support event enrichment when needed.

Acceptance Criteria

1. **Event Processing & Routing**

- The system **must** process events and trigger appropriate notifications.
- The system **must** support multiple notification channels (e.g., WebSocket, push notifications, email).
- The system **must** include a mechanism to **enrich events** with additional data when required before processing.

2. **Real-Time WebSocket Notifications**

- Users **must** receive real-time notifications in a web application.
- The system **must** maintain a mechanism to track active WebSocket connections.

3. **Mobile Push Notifications**

- The system **must** support push notifications for both **iOS** and **Android** devices.

4. **Email Notifications**

- The system **must** support email notifications with a **templating system**.
- Emails **must** support multiple languages and dynamic content placeholders.

5. **Event Enrichment**

- The system **must** determine whether an event requires enrichment before being processed.
- If enrichment is required, missing data **must** be retrieved from an external service/API.
- Enrichment **must** be performed before the event triggers a notification.

6. **Logging & Monitoring**

- The system **must** log relevant events for debugging and monitoring purposes.

Additional Notes

- The architecture should be designed using **AWS serverless services** (e.g., EventBridge, Lambda, SNS, SES, API Gateway, DynamoDB).
- The candidate should clearly represent **event enrichment** in their diagram, ensuring the system can fetch and inject missing data before event processing.
- The candidate **must** provide a **architecture diagram** illustrating the system's components, data flow, and interactions.
- The diagram **must** highlight how events are processed, enriched (if necessary), and routed to different notification channels.
- No implementation or deployable code is required.