# Development Process for Event Management System API

1. **Project Initialization:**

* **Requirements Gathering:** The project started with defining the basic features needed for an Event Management System, including event creation, attendee management, and status updates
* **Technology Selection:**
  + **FastAPI** was chosen for its speed and simplicity in building APIs
  + **SQLAlchemy ORM** was used to interact with MySQL database and manage database schema
  + **Pydantic** for data validation and serialization
  + **Pytest** was selected for writing automated tests for each API endpoint
  + **Uvicorn** was used as the ASGI server for running the FastAPI application
* **Project Setup:**
  + Created a python virtual environment
  + Installed necessary libraries using pip and prepared a requirements.txt file for managing dependencies

1. **Database Setup:**

* **Configured the MySQL database:** Set up MySQL as the backend storage system to handle event and attendee data for the Event Management System
* **Database Creation:** Created a database named event\_management in MySQL Workbench to store all the event-related data
* **Table Creation:** Designed and created two primary tables in the event\_management database:
  + **event** table to store event details (name, description, start\_time, end\_time, etc.)
  + **attendees** table to store attendee details (name, email, phone number, event id, etc.)
* **Relationships Setup:** 
  + Established a relationship between the Event and Attendee models using SQLAlchemy’s foreign key mechanism, linking the attendees.event\_id column to the events.event\_id column. This allows the application to associate each attendee with a specific event

1. **Backend Development:**

* **API Development (FastAPI):** Designed and developed the backend using FastAPI, which provides a fast and efficient framework for creating RESTful APIs. Created endpoints for adding, retrieving, updating, and deleting events and attendees
* **CRUD Operations:** 
  + Implemented Create, Read, Update, and Delete (CRUD) operations for both events and attendees to manage the lifecycle of events and the attendees associated with them
  + Developed endpoints like:
    - **POST** /events/:to create a new event
    - **GET** /events/{event\_id}: to fetch event details
    - **PUT** /attendees/{attendee\_id}: to update attendee details
    - **DELETE** /attendees/{attendee\_id}: to delete an attendee

1. **Validation and Error Handling:**

* Implemented Pydantic models to validate input data for both Event and Attendee endpoints
* Added error handling using FastAPI’s HTTPException to manage common API errors like 404 (not found)

1. **Testing:**

* **Unit Tests for Business Logic:** 
  + **Registration Limits:** Validated that the event’s maximum attendee limit is enforced. If an attempt is made to register more than the allowed number of attendees, the system return an appropriate error response
  + **Check-ins:** Ensured that attendee check-ins are correctly handled
  + **Automatic Status Updates:** Verified that event status automatically updates based on conditions (e.g., status changing from “scheduled” to “completed” once the event ends)
* **API Endpoint Testing:**
  + **Validation of API Behaviour:** Implemented tests to ensure that the API endpoints behave as expected under different conditions, including valid and invalid inputs
* **Test Case Coverage:**
  + **Event and Attendee Creation:** Ensured events and attendees can be successfully created
  + **Event and Attendee Retrieval:** Verified that the correct data is returned when retrieving events and attendees
  + **Updating Event and Attendee details:** Ensured that attendee and event details can be updated correctly
  + **Deleting Events and Attendees:** Tested the deletion of events and attendees, and verified their removal from the database
  + **Edge Cases & Error Handling:** Validated that edge cases (e.g., registration beyond attendee limit, non-existing attendee or event) are handled properly

1. **Deployment and Configuration**

* **Uvicorn Setup:**
  + Configured Uvicorn to serve the FastAPI application in production
  + Command to run the application: uvicorn app.main:app –reload
* **MySQL Configuration:** 
  + Ensured the MySQL database was properly configured with secure access credentials
  + The database connection is pooled using SQLAlchemy to optimize resource usage and support a scalable application

1. **Documentation**

* **API Documentation:** Utilized FastAPI’s built-in interactive documentation, which allows users to test endpoints directly through the UI (Swagger UI)
* **Project Documentation:** Thoroughly documented the project setup, API endpoints, database schema, test cases and detailed instructions on how to run the project and execute the test cases

1. **Version Control& Deployment**

* Deployed the application in a local development environment, ensuring it connects to the MySQL database and handles all API requests
* Commited the changes frequently to GitHub with clear commit messages
* Kept the README.md file updated with setup instructions and API documentation