Coding Challenge: Event Booking System

Objective

You are required to build an **Event Booking System** consisting of:

- A backend service using **FastAPI** with an **SQLite** database or any other database you are comfortable with.
- A **ReactJS** frontend that interacts with the API.
- A **Python SDK** (generated using OpenAPI Generator CLI).
- **Automation scripts** (PowerShell, Shell, or any other script you are comfortable with) to simplify setup and execution.

Your solution should be **fully functional**, adhere to **OpenAPI standards**, and demonstrate **best practices** in backend, frontend, database, and SDK development.

Please leverage any means available on the internet, such as **LLMs, open-source projects, or any other resources**, to enhance your solution. Using your **imagination to add additional functionality** is a plus.

📌 Tasks & Requirements

1. Backend Development (FastAPI & Database)

Implement a FastAPI service with the following endpoints:

- **POST** `/events/` → Create a new event.
- **GET** `/events/` → Retrieve all events.
- **GET** `/events/{event_id}` → Retrieve a specific event.
- **POST** `/events/{event_id}/book` → Book a seat for an event.
- **DELETE** `/bookings/{booking_id}` → Cancel a booking.

Requirements:

- Use **SQLite** or any other database you are comfortable with.
- Ensure the API follows **OpenAPI standards** (**Swagger UI docs can be accessible**).
- Implement **error handling** (e.g., `404` for non-existent events, `400` for overbooking).
- Write **unit tests** for the backend.

Trick Logic in Backend (Hidden Complexity)

Each event has a **limited number of seats**:

- The system should **not allow booking more than available seats**.
- If an event is **fully booked**, new booking requests should be rejected.
- **Cancellations** should free up seats, allowing new bookings.

📌 Example: ✅ Event A has **5 seats** → **5 bookings allowed** ❌ If **6th person tries to book**, API should return an error. ✅ If someone **cancels**, **1 seat becomes available**.

Your implementation should correctly **reject overbookings and allow seat reallocation**.

---

2. Database (SQLite or Other Database)

Create two database tables:

```sql
CREATE TABLE events (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    total_seats INTEGER NOT NULL,
    available_seats INTEGER NOT NULL CHECK (available_seats >= 0),
    date TIMESTAMP NOT NULL
);

CREATE TABLE bookings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    event_id INTEGER NOT NULL,
    user_name TEXT NOT NULL,
    booking_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE
);
```

- Provide a **sample SQL file** (`seed_data.sql`) to populate initial test data.

---

3. Frontend Client (ReactJS)

Develop a **ReactJS-based frontend**.

Use **Axios** for making API calls.

The frontend should allow users to:

✅ View all events. ✅ Book a seat for an event. ✅ Cancel a booking. ✅ See available seats in real time.

Requirements:

- The frontend should interact **only via API calls** (no direct database access).
- **Style is not a priority**, but **functionality must work correctly**.

📌 **Bonus:**

- Implement **real-time seat availability updates**.

---

4. Platform SDK (Generated via OpenAPI Generator CLI)

Instead of manually writing an SDK, you must **generate** a **Python SDK** using the **OpenAPI Generator CLI**.

Instructions to Generate SDK

1. **Install OpenAPI Generator CLI:**

```
npm install -g @openapitools/openapi-generator-cli
```

2. **Run the following command to generate the SDK:**

```
openapi-generator-cli generate -i http://localhost:8000/openapi.json -g python -o event_sdk
```

Modify and Test the SDK:

- After generation, **modify the SDK** if needed.
- Write a **sample script** to **test SDK functionalities**.

📌 **Hint:**

- Ensure your **FastAPI server is running** before running the generator.

- The OpenAPI spec will be available at `http://localhost:8000/openapi.json`.

Expected SDK Usage (After Generation & Testing):

```
from event_sdk.api.events_api import EventsApi
from event_sdk import ApiClient

client = ApiClient()
events_api = EventsApi(client)

# Retrieve all events
events = events_api.get_events()

print(events)
```

---

5. Development Setup Script

Create an automation script (PowerShell, Shell, or any other script you are comfortable with) that:

✅ **Sets up a virtual environment**. ✅ **Installs Python dependencies** (`requirements.txt`). ✅ **Initializes the database** (**applies migrations**). ✅ **Installs ReactJS dependencies**.

---

6. Execution Script

Create an automation script (PowerShell, Shell, or any other script you are comfortable with) to start the full application.

✅ **Start the FastAPI backend**. ✅ **Start the React frontend**.

🎯 Evaluation Criteria

✅ **Code Quality:** Well-structured, clean, and modular code. ✅ **Correct API Implementation:** API should work correctly and follow OpenAPI standards. ✅ **Proper Error Handling:** Handles bad inputs and errors gracefully. ✅ **Platform SDK Generation:** SDK must be generated correctly using OpenAPI Generator CLI. ✅ **Frontend Integration:** The ReactJS frontend should correctly communicate with the backend. ✅ **Automation Scripts:** Setup and execution scripts should work as expected. ✅ **Unit Tests:** Backend should include test cases. ✅ **Documentation:** Include a README with installation and execution steps. ✅ **Backend Trick Logic:** Ensure candidates correctly implement the seat booking logic.

📦 Deliverables

By the end of the task, you should submit the following:

- ✅ **Backend (FastAPI) source code**
- ✅ **Database schema & migration files**
- ✅ **ReactJS frontend code**
- ✅ **Python SDK** (generated via OpenAPI Generator CLI)
- ✅ **Setup & Execution Scripts** (PowerShell, Shell, or any other script)
- ✅ **Unit tests for backend**
- ✅ **README explaining how to set up and run the project**

Good luck! 🚀