### **🍽️ Тестовое задание: API-сервис бронирования столиков в ресторане**

#### **📌 Цель**

Разработать REST API для бронирования столиков в ресторане. Сервис должен позволять создавать, просматривать и удалять брони, а также управлять столиками и временными слотами.

### **✅ Функциональные требования**

#### **Модели:**

1. **Table** – столик в ресторане:  
   * id: int
   * name: str (например, "Table 1")
   * seats: int (количество мест)
   * location: str (например, "зал у окна", "терраса")
2. **Reservation** – бронь:  
   * id: int
   * customer\_name: str
   * table\_id: int (FK на Table)
   * reservation\_time: datetime
   * duration\_minutes: int

#### **Методы API:**

* **Столики:**
  + GET /tables/ — список всех столиков
  + POST /tables/ — создать новый столик
  + DELETE /tables/{id} — удалить столик
* **Брони:**
  + GET /reservations/ — список всех броней
  + POST /reservations/ — создать новую бронь
  + DELETE /reservations/{id} — удалить бронь

#### **Логика бронирования:**

* Нельзя создать бронь, если в указанный временной слот столик уже занят (пересечение по времени и table\_id).
* Бронь может длиться произвольное количество минут.
* Валидации должны обрабатываться на уровне API (например, конфликт брони должен выдавать ошибку с пояснением).

### **⚙️ Технические требования**

* Использовать **FastAPI** как основной фреймворк.
* Работа с БД через **SQLAlchemy** или **SQLModel**.
* Использовать **PostgreSQL**.
* Использовать **Alembic** для миграций.
* Приложение должно быть обернуто в **Docker**.
* Использовать docker-compose для запуска всех компонентов.
* Структура проекта должна быть модульной: routers/, models/, schemas/, services/, и т.п.
* Код должен быть легко расширяемым.
* Приветствуется: логгирование, покрытие базовых сценариев тестами (на pytest).

### **📦 Что нужно предоставить**

* Ссылку на репозиторий с проектом (GitHub / GitLab).
* README.md с инструкцией по запуску (docker-compose up) и кратким описанием проекта.

### **🧠 Оценка будет производиться по:**

* Архитектуре проекта.
* Качества и читаемости кода.
* Корректности бизнес-логики (в частности, проверка на конфликты броней).
* Владению Docker и docker-compose.
* Наличию тестов и валидных миграций.