Event Ticketing System

Overview

Build a fullstack event ticketing system using Angular for the frontend and Python for the backend. This assignment evaluates your database design skills, code quality, and ability to implement complex business logic while maintaining clean architecture.

Technical Stack

- **Frontend:** Angular with TypeScript
- **Backend:** Python (Flask or FastAPI recommended)
- **Database:** PostgreSQL or SQLite
- **ORM:** SQLAlchemy or similar

Database Schema Requirements

Design and implement the following entities with proper relationships:

Core Entities

```
None
Users
- id (Primary Key)
- name
- phone
- role (enum: 'customer', 'admin')

Events
- id (Primary Key)
- title
- description
- venue
- date_time
- capacity (integer)
```

```
- price (decimal)
- status (enum: 'active', 'cancelled', 'completed')
- created_at

Bookings
- id (Primary Key)
- user_id (Foreign Key → Users)
- event_id (Foreign Key → Events)
- quantity (integer)
- total_amount (decimal)
- booking_date
- status (enum: 'confirmed', 'cancelled')

Tickets
- id (Primary Key)
- booking_id (Foreign Key → Bookings)
- ticket_code (unique string)
- status (enum: 'active', 'used', 'cancelled')
```

Database Design Requirements

- Implement proper foreign key relationships
- Add appropriate constraints (check constraints for positive quantities, valid dates, etc.)
- Create strategic indexes for performance
- Ensure data integrity with proper cascading rules

Frontend Requirements (Angular)

1. Public Event Listing

- Display all active events in a clean, responsive layout
- Show event details: title, description, venue, date, available tickets, price
- Basic search functionality (by title or venue)
- Filter by date range

2. Event Detail & Booking

- Dedicated page for each event with full details
- Booking form with quantity selector
- Show current availability at page load
- Booking confirmation with ticket details

3. User Dashboard

- "My Bookings" page showing user's booking history
- Display booking details and individual ticket codes
- Allow booking cancellation (if implemented)

4. Admin Panel (Simple)

- Form to create new events
- List of all events with basic statistics (total bookings, revenue)
- View bookings for specific events

UI/UX Requirements

- Responsive design
- Clean, intuitive interface
- Loading states and error handling
- Form validation with user feedback

Backend Requirements (Python)

1. REST API Design

Implement the following endpoints with proper HTTP methods and status codes:

```
None
Events:
GET    /api/events              # List all active events
GET    /api/events/{id}         # Get event details
POST   /api/events              # Create new event (admin only)
PUT    /api/events/{id}         # Update event (admin only)

Bookings:
POST   /api/bookings            # Create new booking
```

```
GET    /api/bookings/user/{id}  # Get user's bookings
GET    /api/bookings/event/{id} # Get event's bookings (admin
only)
PUT    /api/bookings/{id}       # Update booking status

Users:
GET    /api/users               # List users (for booking form)
POST   /api/users               # Create test users
```

2. Business Logic Requirements

- **Ticket Availability:** Implement basic checks for available tickets
- **Data Validation:** Comprehensive input validation and sanitization
- **Ticket Code Generation:** Generate unique, secure ticket codes

3. Database Integration

- Implement database migrations/schema management
- Write optimized queries to avoid N+1 problems
- Use proper indexing for performance

Setup & Deployment Instructions

Required Deliverables

1. **Database Schema:** SQL file or migration scripts
2. **Backend Code:** Well-organized Python application
3. **Frontend Code:** Angular application with proper component structure
4. **README.md:** Comprehensive setup instructions
5. **Sample Data:** Script to populate test data including users and events

README Requirements

Include the following in your README:

- Project overview and architecture decisions
- Local setup instructions (step-by-step)
- API documentation (endpoints and examples)

- Design decisions and trade-offs
- Known limitations or areas for improvement

Guidelines & Notes

AI Usage Policy

- **AI assistance is encouraged** for boilerplate code, debugging, and research
- **Focus areas for evaluation:** Your design decisions, architecture choices, and problem-solving approach
- **Document your decisions:** Explain why you chose specific approaches in your README

Test Data

Create at least:

- 3 test users (1 admin, 2 customers)
- 5 sample events (mix of capacities and dates)
- Several sample bookings to demonstrate functionality

Optional Enhancements (Bonus Points)

If you finish early, consider adding:

- **Concurrency Control:** Prevent overbooking when multiple users book simultaneously
- Booking cancellation functionality
- Event status management (active/cancelled/completed)

Submission Format

- **Code Repository:** GitHub repository with clear commit history
- **Documentation:** Include architecture diagrams if helpful

Questions?

Document any assumptions you make about requirements in your README. Focus on building a robust, well-designed system that demonstrates your understanding of fullstack development principles and database design.

Deadline: 1 week from assignment date **Good luck!**