Challenge: Railway Ticket Reservation API

Overview

Create a **RESTful API** that manages railway ticket reservations with the following features:

1. **Book a Ticket**
2. **Cancel a Ticket**
3. **Booked Tickets** (including passenger details & summary)
4. **Available Tickets** (including unoccupied details & summary)

Your solution should enforce the following **core constraints**:

- **63 confirmed berths** total.
- **9 RAC berths** that can hold **18 RAC tickets** (2 passengers per side-lower berth).
- **10 waiting-list** tickets maximum.
- If the waiting-list is full, the system must respond with something like `"No tickets available"`.
- Children under **age 5** do **not** get a berth but their details must still be stored.
- **Priority** for lower berths:
    - Passengers aged **60+**.
    - **Ladies with children**, if a lower berth is still available.
- **RAC** passengers are allocated **side-lower** berths.

Cancellation Logic:

- When a confirmed ticket is canceled, the next RAC ticket (if any) should become confirmed.
- Then, one waiting-list passenger (if any) should move to RAC.

Database Design

You'll use a **relational database** (like PostgreSQL/MySQL/SQLite). Demonstrate a clean schema (tables for passengers, tickets, berth allocations, etc.) that respects the constraints above.

Please ensure your solution:

1. **Validates** user inputs.
2. Follows **best practices** for SQL queries and relationships .
3. **Locks** or otherwise prevents exceeding the max seats in each category

---

API Endpoints (Suggested)

- **POST** `/api/v1/tickets/book`
- **POST** `/api/v1/tickets/cancel/{ticketId}`
- **GET** `/api/v1/tickets/booked`
- **GET** `/api/v1/tickets/available`

(You’re free to adjust naming or structure; just ensure you meet the core requirements.)

What We Expect

Functional Code

- Must run **locally** with minimal setup and usage instructions.

Correct Business Logic

- Especially around **seat allocation** (confirmed berths, RAC, waiting-list) and the **promotion** steps (RAC → confirmed, waiting-list → RAC).
- **Concurrency Handling**: Ensure two users **cannot** book the same ticket/berth at the same time.

Dockerized Deployment

- The project must run inside a Docker container.
- Include a `Dockerfile` and `docker-compose.yml` for easy setup and deployment.

Implementation Constraints

- **Languages**: Use **JavaScript (Node.js) or Python** (e.g., Django, Flask, FastAPI) **only**.
- **Database**: Must be a **Relational Database** (e.g., PostgreSQL, MySQL, SQLite).
- No other frameworks or tech stacks are allowed outside of the above.

Extra Points

- **In-Depth Documentation**: Additional docs (architecture diagrams, flowcharts, more sample requests, etc.).
- **Superior Coding Standards**: Well-structured code, consistent formatting, clear variable/function names, and DRY (Don’t Repeat Yourself) principles.