Introduction.

Due to new regulations, restaurants are now requested to limit the number of tables and to only serve customers with prior reservations.

As a Software Engineer, a restaurant owner approached you to solve a reservation problem which the restaurant is facing. The problem is as follows:

- The restaurant has X number of tables, each having any number of seats between 1 - 12.
- More than one table in the restaurant may have the same number of seats.
- Each customer's group would like to sit at one table.
- To maximize profits, the restaurant owner allows customers to reserve only the minimal sized table of size equal to or more of the required seats.
    - For example:
        - Let's say the restaurant has 2 tables with 2 seats, 1 table with 4 seats, and 3 tables with 6 seats.
        - If a customer's group has a single person, the customer can only reserve the table with 2 seats. The customer cannot reserve any other table.
        - If a customer's group consists of 3 people the customer can only reserve the table with 4 seats.
- The restaurant opens every day at 12:00 PM and closes at 11:59 PM.

You're requested to implement a backend system that enables the restaurant workers to easily reserve tables based on the size of customer's groups. The backend system must satisfy the requirements specified below.

General Requirements.

- Implement the backend system in any of the following Languages/Frameworks (Only):
  - Python FastApi
  - NodeJs Express
- Design and implement a Postgres DB
- Use a Redis DB for caching roles for faster authorization
- Create an algorithm for the solution of the reservation problem, and model it as a flow chart or an activity diagram
- Implement the required functionality, and provide it as restful APIs
- Make sure to follow best practices of all technologies you use
- All provided APIs must have proper validation for business logic
- Design the DB and the APIs to satisfy the requirements, the requirements do not impose or suggest a specific design or implementation
- APIs must return proper and descriptive error messages for edge cases or validations
- All APIs must be documented in Postman, with proper use of postman's environment variables
- All deliverables must be submitted as a github repository
- The backend with all associated DBs must be containerized using docker. The system should be runnable just by the command: docker-compose up

Functional requirements.

Users & Authentication

- The system can be used by more than one user.
- Each user will have a name, an employee number, role, and a password.
  - There are two roles only: Admin and Employee
- Users must be authenticated (logged in) to use any of the functionality.
- Only admins are able to add new employees, by specifying their information.
  - Don't allow duplicate employee numbers.
  - Employee number consists of exactly 4 digits.
  - Passwords must be at least 6 characters long.
- Users can login using their employee number and password.
- Authentication should be using JWT.
- Authorization mechanism should use RedisDB for caching roles.

Table Management

Get restaurant tables

Only admins can retrieve restaurant tables.

- A list of tables is returned. Each table has:
    - Number.
        - An integer representing the number of the table, as identified by the restaurant's employees.
    - Number of seats.
        - An integer representing the number of seats for that table.
        - Can only be between 1-12 inclusive.

Add a restaurant table.

- Only admins can add tables to the restaurant.
- Must specify:
    - Number.
    - Number of seats.

Delete a restaurant table.

- Only admins can delete a restaurant's table.
- Do not allow a table to be deleted if the table has any reservations to it.

Reservations.

Check available time slots.

Restaurant employees and admins can easily check the available time slots for a customer by using this API.

- Restaurant Employee specifies the number of required seats for the customer.
- The system retrieves a list of time ranges in which the tables with minimum number of seats required for the customer are available.
    - Each time slot has a starting/ending time.
    - Only returns time slots in the future.
    - Only returns available time slots for the rest of the working hours of the day.
    - Examples:
        - If restaurant has only 1 table with 2 seats, the current time is 2:00 PM, there are no reservations for the current day, and the customer requested 2 seats:
            - Display all time slots that have a table with 2 seats available from now till the end of the day.
            - Result:
                - 2:00 PM - 11:59 PM
        - If restaurant has only 1 table with 4 seats, there are two reservations for the table with 4 seats at 4:00 PM - 4:30 and 5:30 PM - 5:45 PM, the current time is 1:00 PM, and the customer requested 3 seats:
            - Display all time slots that have a table with 4 seats available from now till the end of the day.
            - Result:
                - 1:00 PM - 4:00 PM
                - 4:30 PM - 5:30 PM
                - 5:45 PM - 11:59 PM
- Deny customers if they require a table with more seats than what the biggest table in the restaurant has.
- Deny customers if there are no available time slots for the day for the tables with required size.

Reserve a time slot.

Restaurant employees and admins can add a new reservation for a table at a specific time slot.

- The table is considered as reserved, and cannot be reserved by any other customer at the same or overlapping time slot.
- Restaurant employee specifies the table, the starting time, and the ending time.
- The starting and ending times must be within the restaurant's working hours.
- The system must make sure that the required table at the specified time slot is actually available for the customer.

Get reservations for today.

Restaurant employees and admins can view all reservations for the current working day.

- The API should support pagination to avoid loading a huge amount of reservations at once.
- Employees can sort the reservations by time in ascending or descending manner.

Get all reservations.

- Only admins can view all reservations for all times.
- API must support pagination to avoid loading huge amounts of reservations at once.
- Admins can filter reservation by table(s).
- Admins can filter reservations by a date range.

Delete a reservation.

Restaurant employees and admins can delete a specific reservation for the current working day.

- The API shouldn't allow the deletion of a reservation in the past.

Tests.

- The backend must contain integration tests for all reservations functionality.
- The tests must cover edge cases and different scenarios for reservations, business logic and validations.
- Tests must be runnable using a simple command.

Containerization.

- All services required to run the backend must be containerized using docker and docker-compose in a proper manner.
- The main DB must be automatically initiated on the first run (With one default admin user).

Submission.

- Include the following:
    - a readme file with instructions to run the code and tests locally.
    - All code files.
    - Algorithm diagram.
    - Postman Collection/Environment.
    - Any associated files that were created for this task.
- You must **send** or **commit** all of the previous deliverables in a git repository.