**Role challenge**

**Duration: 3** hours

**Task: Develop a simplified appointment system with NESTJS**

**Description:**

Develop a RESTful API for an appointment scheduling system that supports creating, viewing, and canceling appointments, with configurable slots and operational parameters.

**Business Logic:**

* Each appointment slot is 30 minutes long.
* Appointments are available from 9 AM to 6 PM on weekdays.
* Prevent double bookings for the same slot.

**Expected output:**

* API to
  + Retrieve the available slots based on the selected date.
    - The response should list the available slots and the time. Given that each appointment slot is 30 minutes and only 1 available slots

[

{

"date": "2024-04-04",

"time": "10:00",

"available\_slots": 1

},

{

"date": "2024-04-04",

"time": "10:30",

"available\_slots": 1

},

{

"date": "2024-04-04",

"time": "11:00",

"available\_slots": 0

},

...

]

* + Book the appointment based on the available slot.
    - Validate if the slot is available
    - Available slot should be deducted upon appointment made successfully

**Additional Requirements:**

**Basic Level Configuration:**

* Allow configuration of the appointment slot duration (minimum 5 minutes).
* Enable setting the maximum number of slots per appointment (1 to 5 slots).
* Configure operational hours and days for scheduling appointments.

**Advanced Level Configuration (Database Enhancements):**

* Implement functionality to set days off (e.g., public holidays).
* Allow setting unavailable hours within operational days (e.g., lunch breaks).

**Note: You are freely to use your imagination on how the configuration like, can be either using env, json file or database. You also may wear a product hats to put an assumption for the cases, but make sure you note it down and showcase.**

**Tech stack:**

Framework: NestJS

Language: Typescript

Database: ORM (Preferrable) Mikroorm as framework and any sql (Postgres is preferrable)

Submission:

Via github public repo

**Assessment Criteria:**

Functionality: API meets all specified features and handles edge cases.

Code Quality and Organization: Clear structure, naming conventions, and documentation.

Error Handling: Proper response codes and messages for invalid requests.

Persistence: Effective use of a database for storing appointment and configuration data.

Security: Basic considerations for API security (e.g., input validation).

Documentation: A brief description on your implementation. Additional diagrams would be great to demonstrate your system design.

*Tips: You may check out Calendly or Google Appointment Schedule as reference*