Nest JS backend test

Scenario:

Zurich Malaysia is introducing a new motor insurance website where users are able to obtain the insurance premium instantly. The pricing of the product varies by vehicle's location, for example:

Product Code	Product Description	Location	Price (RM)
1000	Sedan	West Malaysia	300
1000	Sedan	East Malaysia	450
2000	SUV	West Malaysia	500
2000	SUV	East Malaysia	650

As a backend developer, you must build a pricing query API, where the user passes in the product id and location, and the API returns the premium. The API requires administrative functionalities to maintain the products and prices as well.

Task:

Build an API using NestJS, which consists of:

- Endpoints:
  - **GET** `/product`
  - Query parameter includes `productCode, location`
  - Accessible by all type of users
  - **POST** `/product`
  - Request body includes productCode, location, price
  - Admin access only
  - **PUT** `/product`
  - Query parameter includes productCode
    - Request body includes location and price
    - Admin access only
  - **DELETE** `/product`
  - query parameter includes product code
  - admin access only
- Upload your code to your personal GitHub account, and make it public for us to access and evaluate.
- Remember to include a ReadMe file to explain your solution.

Requirements:

- All endpoint must be accessible from Swagger
- All endpoints' requests and responses must be using DTO verify and standardize all communication.
- NestJS structure must include one module consisting of one controller and provider.
- Connect the API to a Postgresql database named MOTOR_INSURANCE_WEBSITE and PRODUCT table using typeORM to connect
- As a security requirement, header token/metadata must contain user role and be checked before accessing an endpoint. You are not requied to build the user table with roles.

Bonus points:

- Middleware is used for security and role checking purposes.
- API is deployment ready, eg; include a docker compose script.
- API is able to handle heavy load and high traffic requests.
- Unit tests with at least 80% code coverage.
- Usage of linter is encouraged.