**Part 1: Database Design and SQL (PostgreSQL)**

Task 1: Database Schema Design

Given a hypothetical scenario for an e-commerce platform, design a PostgreSQL database schema to store information about products, users, orders, and transactions.

Include tables, relationships, primary keys, foreign keys, and any necessary constraints.

Explain the reasoning behind your design choices.

Users Table:

This table stores information about users who interact with the e-commerce platform.

Attributes:

user\_id (Primary Key)

username

email

password

Reasoning: Users are a fundamental aspect of any e-commerce platform. Storing user information allows for authentication, order tracking, and personalized experiences.

Products Table:

This table stores information about the products available for sale.

Attributes:

product\_id (Primary Key)

name

description

price

stock\_available

Reasoning: Products are central to an e-commerce platform. Storing product information enables users to browse, search, and purchase items.

Orders Table:

This table stores information about orders placed by users.

Attributes:

order\_id (Primary Key)

user\_id (Foreign Key referencing Users table)

product\_id(Foreign Key referencing Product table)

status (e.g., 'Pending', 'Shipped', 'Delivered')

Reasoning: Orders represent transactions between users and the platform. Storing order details facilitates order management and tracking.

One-to-Many Relationships:

Users to Orders: One user can place multiple orders, but each order is associated with only one user. This is a one-to-many relationship.

Products to Orders: One product can place multiple orders

Many-to-One Relationships:

Orders to User and Orders to Products :As for the my covenant, design the data base.

Task 2: SQL Queries

Write SQL queries to perform the following tasks:

Retrieve the total number of orders for a specific user.

Get the top 5 products based on the total number of units sold.

Update the status of an order to 'Shipped' based on the order ID.

Provide a list of customers who have not made any purchases.

1.Retrieve the total number of orders for a specific user:

Query: SELECT COUNT(\*) AS total\_orders FROM Orders WHERE user\_id = specific\_user\_id;

2.Get the top 5 products based on the total number of units sold

Query: SELECT p.product\_id, p.name, SUM(o.quantity) AS total\_units\_sold FROM Products p

JOIN Orders o ON p.product\_id = o.product\_id GROUP BY p.product\_id, p.name

ORDER BY total\_units\_sold DESC LIMIT 5;

3. Update the status of an order to 'Shipped' based on the order ID

Query: UPDATE Orders SET status = 'Shipped' WHERE order\_id = specific\_order\_id;

4. Provide a list of customers who have not made any purchases.

Query: SELECT u.user\_id, u.username FROM Users u LEFT JOIN Orders o ON u.user\_id = o.user\_id WHERE o.order\_id IS NULL;

**Part 2: Backend Development (NestJS)**

Task 3: API Development

Create a simple NestJS application with the following features:

CRUD operations for managing products (create, read, update, delete).

Authentication middleware using JWT tokens.

An endpoint to retrieve a list of orders with product details for a specific user.

To Access the this url: https://localhost:3031/order/{ specific\_user\_id}/producs

Task 4: Middleware and Error Handling

Implement middleware to log incoming requests, including the endpoint, method, and timestamp.

Create error handling middleware to handle common HTTP errors gracefully.

Middleware implement in the src\utility\middleware\current-user.middleware.ts

Error/Exception handle done by with try catch in same method.

I mplemented the JWT token in Signing with proper email and password, it generate the

JWT, with Middleware is implements. This is url to implement https://localhost:3031/users/signin

Task 5: Unit Testing

Write unit tests for one of the API endpoints, ensuring both success and error scenarios are covered.

Unit test implement in this location src\users\users.controller.spec.ts

**Part 3: Frontend Development (Next.js)**

Task 6: Page Creation

Build a simple Next.js application with the following pages:

Product listing page displaying all available products.

Product detail page with information about a selected produts.

Created the project name is ECOMNEXT

When acess the <https://localhost:3000/>

It shows the welcome page, in that show the products link ,one click that link, it shows the all products and if you click the select the product it show the alert show the details of the page.

Task 7: API Integration

Integrate the NestJS API developed earlier into the Next.js application to fetch and display product information.

With help of the fetch(), made call to NestJs end point

Visit the compones/page.stx

Project Urls

Signup user

<http://localhost:3001/users/signup>

Sample data

{

”username”:”admin”,

“emial”:”[admin@mail.com](mailto:admin@mail.com)”

“password”:”admin”

}

Signup user

<http://localhost:3001/users/signin>

Sample data { “email”:[admin@mail.com](mailto:admin@mail.com),”password”:”admin”}

Product CURD

Product Get, Here Http method type Get

<http://localhost:3001/product>

Product Get By Id, Here Http method type Get

<http://localhost:3001/product>/{Product\_id}

Product Update Here Http method type Patch

[http://localhost:3001/product/update/{product\_id}](http://localhost:3001/product/update/%7bproduct_id%7d)

Product Delete Here http method type is Delete

[http://localhost:3001/{p](http://localhost:3001/%7bp)roduct\_id}

Ecommer(Nest Project) url : <https://github.com/Ajay-527/Nestproject>

Ecomnest(Next Project) url: <https://github.com/Ajay-527/NextProject>

CotextApi Acctive Url: <https://github.com/Ajay-527/contextApiProject>

Documentation Url: <https://github.com/Ajay-527/Documentation>