**NestJs Backend**  
  
**This is a professional assignment. Please generate the complete production-grade NestJS backend project and full documentation strictly following the instructions below.**

**🚀 Build a Production-Ready E-commerce Backend using NestJS and MongoDB**

**1. Architecture & Best Practices**

* Use **NestJS modular architecture**:
  + **Controllers**, **Services**, **Modules**, **DTOs**, **Guards**, **Pipes**, **Exception Filters**, **Interceptors**.
* Implement **JWT Authentication** using **Passport.js** (real JWT token generation, not mock).
* Enable **CORS** and secure HTTP headers using **Helmet**.
* Set up **Global Exception Handling**.
* Use **NestJS Logger** and a **Global Logging Interceptor** for structured request/response logging.
* Follow **strict code organization** with proper **inline code comments**.

**2. Functional Modules**

* **User Module**:
  + Register (with **bcrypt password hashing**).
  + Login (generate real JWT token).
  + Profile Update.
  + Forgot Password / Reset Password flow.
* **Product Module**:
  + Create, Read, Update, Delete (CRUD).
  + Search and Filter by **category**, **brand**, and **price**.
* **Cart Module**:
  + Add item to cart.
  + Update item quantity.
  + Remove item from cart.
  + View all cart items.
* **Order Module**:
  + Place Order.
  + View Order History.
  + Cancel Order.
* **Payment Module**:
  + Simulate Mock Payment flow (success and failure).

**3. Seeding (Database Initialization)**

* Create **seed files** for:
  + Users
  + Products
  + Categories
  + Carts
  + Orders
* Seed data expectations:
  + **Passwords must be hashed** using **bcrypt** in the seed data.
  + Include realistic sample data (e.g., product images, prices).
* Implement a **SeedService**:
  + On application startup, **auto-insert** seed data if collections are empty.
* Use a **Named MongoDB Connection** called mongodbconn:
  + In MongooseModule.forRoot, specify { connectionName: 'mongodbconn' }.
  + In all MongooseModule.forFeature and @InjectModel, always use { connectionName: 'mongodbconn' }.

**4. Unit Testing**

* Use **Jest** for unit testing.
* Write tests for:
  + Controllers
  + Services
  + Guards
  + Filters
  + Interceptors
* Test Coverage Requirements:
  + Achieve **at least 95% code coverage**.
  + Exclude index.ts, \*.module.ts, and \*.schema.ts files from coverage.
* Cover both **successful cases** and **failure/error cases**.

**5. API Documentation**

* Integrate **Swagger UI** at the endpoint /api/docs.
* Generate and export a **complete Postman Collection** covering all APIs.
* Swagger should include:
  + Request examples
  + Response examples
  + Authentication details (JWT required fields).

**6. README.md Documentation**

* Create a **professional README** with the following sections:
  + Project Overview
  + Technology Stack
  + Database Setup (local MongoDB or MongoDB Atlas)
  + Environment Setup Guide (with .env examples)
  + How to Seed the Database
  + How to Access Swagger Documentation
  + How to Import and Use the Postman Collection
  + How to Run Unit Tests and View Coverage Report
  + Explain clearly the use of **named connection mongodbconn** in Mongoose setup.

**7. Environment File (.env) Example**

bash

CopyEdit

MONGODB\_URI=mongodb://localhost:27017/ecommerce

JWT\_SECRET=your\_super\_secret\_key

PORT=3000

**8. Additional Important Requirements**

* All protected APIs must **strictly require** a valid **JWT token**.
* Use **bcrypt** for all password encryption and comparison.
* Always use the **named connection mongodbconn** for database access in all models and services.
* Minimum seed data:
  + **5 users**
  + **10 products**
  + **Relevant sample carts and orders**
* If any error or missing functionality is identified during generation, **propose concise fixes** and apply them properly.