Assignment Problem Statement: Develop Microservices-Based CRUD System Using NestJS

Objective

Develop a NestJS-based microservices architecture for a simple CRUD (Create, Read, Update, Delete) application with modular separation of concerns. The system should demonstrate:
✅ **Microservices architecture (NestJS + modular design).**
✅ **Database integration (PostgreSQL).**
✅ **API Gateway (REST).**
✅ **Unit & integration tests (Jest).**
✅ **GitHub repository with clear documentation.**

Requirements

1. System Architecture (Microservices)

- Use NestJS to structure the app into:
  - User Service (Handles user CRUD operations).
  - Product Service (Handles product inventory).
  - API Gateway (Routes requests to appropriate services).
- Services should communicate via:
  - REST/GraphQL (for external clients).
  - Message brokers (Redis/RabbitMQ) (optional for async tasks).

2. Core Functionality

User Service

- Create User: POST /users
- Get User: GET /users/:id
- Update User: PATCH /users/:id
- Delete User: DELETE /users/:id

Product Service

- Add Product: POST /products
- List Products: GET /products
- Update Stock: PATCH /products/:id/stock
- Delete Product: DELETE /products/:id

3. Database Integration

- Use PostgreSQL (TypeORM or Drizzle ORM).
- Each service should have its own database (or schema).

4. Testing

- Unit Tests (Jest)
  - Test service methods (e.g., UserService.create()).
- Integration Tests
  - Test API endpoints (GET /users, POST /products).
- E2E Tests
  - Verify full flow (e.g., create user → fetch user).

Expected Output

Repository Structure

```
crud-microservice/
├── user-service/        # User CRUD module
├── product-service/   # Product management
├── api-gateway/         # Routes requests
├── libs/                      # Shared utilities
├── tests/                    # Jest tests
├── docker-compose.yml    # Multi-container setup (optional)
└── README.md          # Setup & usage
```

Submission Guidelines

GitHub Repo:

- Link: https://github.com/<your-username>/nestjs-crud-microservices
- Include:
    - Modular NestJS services.
    - Docker setup (optional but encouraged).
    - README.md with API docs.

Evaluation Criteria:

- Correct microservices separation.
- Database integration (PostgreSQL).
- Test coverage ≥ 80%.