SMART RANKS

User Story – User, Product, and Invoice Management Application

1. General Description

Develop a web application for managing users, products, and invoices using the following technologies:

- **Backend:** RESTful API in **Node.js** with **NestJS**.
- **Frontend: Angular** application consuming the API.
- **Database: MongoDB** for data storage.
- **Deployment:** Hosted on a cloud service (**AWS, Azure, or Google Cloud**).

The application must follow **SOLID** principles, design patterns, and **Clean Code** best practices.

---

2. Functional Requirements

2.1. Entity Management (CRUD)

The application must manage three main entities:

1. **Users**
   - Suggested fields: `id, name, email, password, role (admin/user)`.
   - CRUD operations with role-based restrictions:
     - **Admin:** Can create, update, and delete users.
     - **User:** Can only update their own profile.
2. **Products**
   - Suggested fields: `id, name, description, price, stock, status (active/inactive)`.
   - CRUD operations with restrictions:
     - **Admin:** Can create and update products, including stock.
     - **Users:** Can only view product details.
     - **Validation:** Purchases cannot proceed if stock is insufficient.
3. **Invoices**
   - Suggested fields: `id, user_id, products (list of purchased products and quantities), total, date`.
   - CRUD operations with role-based access:
     - **Users:** Can generate invoices when making purchases.
     - **Admin:** Can retrieve invoice details by ID.

SMART RANKS

2.2. Endpoints and Business Logic

- **Users:** Registration and authentication with role-based access (JWT).
- **Products:**
  - Only admins can add new products.
  - Stock updates cannot result in negative values.
- **Invoices:**
  - Only users can generate invoices upon purchase.
  - Stock availability must be verified before processing a purchase.
- **Analytical Queries:** Endpoint to check the number of purchases a user made in the last month.

2.3. Security and Access Control

- **Authentication: JWT** for user validation.
- **Authorization:** Role-based access control for endpoints.
- **Restrictions:** Only users can make purchases; admins manage users and products.

---

3. Non-Functional Requirements

3.1. Development Best Practices

- Modular, reusable, and testable code.
- Implementation of **SOLID** principles and design patterns where applicable.
- **Documentation:**
  - A **detailed README** explaining technologies, setup instructions, and usage.

3.2. Error Handling and Validations

- Implement **global middleware** for exception handling in the backend.
- Data validation and error control in all endpoints.