Inventory Management System - Project Requirements

1. Project Overview

This project is designed to help you practice building a full-stack web application using **Django** for the backend and SQL for database management. You are required to create an **Inventory Management System**, allowing users to manage products, categories, stock levels, and sales. The **UI** will consume APIs from the Django backend, and you have the freedom to use any frontend framework such as **React** or **Vanilla JavaScript**.

2. Project Scope and Objective

The primary goal of this project is to provide hands-on experience with:

- Building a backend API with **Django**.
- Creating, reading, updating, and deleting (CRUD) operations for inventory items.
- Implementing **authentication** (login, registration, and role-based access).
- Designing a simple **UI** that interacts with the Django backend.
- Managing a **SQL database** to store inventory, user, and sales data.

3. Key Features

You are required to implement the following core features:

3.1 User Authentication

- **User Registration**: New users should be able to register with a username, email, and password.
- **Login/Logout**: Existing users should be able to log in with their credentials. Upon login, they should see their dashboard.
- **Role Management**: Implement two types of users:
    - **Admin**: Can manage all aspects of the inventory (add, update, delete items).
    - **User**: Can only view the products but not modify them.

3.2 Inventory Management

- **Add Products**: Admin users can add new products with details like name, category, price, quantity, and description.
- **View Products**: Both admin and regular users can view the list of available products, including their stock levels.
- **Update Products**: Admin users can modify existing product information such as quantity or price.
- **Delete Products**: Admin users can remove products from the inventory.

3.3 Categories

- **Add Category**: Admin users can categorize products by adding and managing categories.
- **View Category**: All users can view products categorized under specific categories.

3.4 Stock Management

- **View Stock**: Users should be able to view product stock levels (e.g., how many items are in stock).
- **Low Stock Notification**: If the stock level of a product falls below a set threshold (e.g., 5 units), an indicator should show this on the UI.

3.5 Sales Management (Optional)

- Admin users can track sales made through the system, updating stock quantities accordingly.

4. Technical Requirements

- **Backend**:
  - **Django** framework (latest stable version).
  - **RESTful APIs**: Create APIs for managing user authentication, inventory, categories, and products.
  - **Database**: Use SQL (PostgreSQL/MySQL/SQLite).
- **Frontend**:
  - You are free to use any framework or library, such as **React**, **Vue.js**, or even **Vanilla JavaScript**.
  - The frontend should consume the APIs from the Django backend for displaying data and sending form submissions (e.g., login, product creation).
- **Database Schema**:
  - **Users**: Username, email, password, and role.
  - **Products**: Name, category, price, quantity, description, and stock level.
  - **Categories**: Name and description.
  - **Sales** (Optional): Product ID, quantity sold, and date.

5. Detailed Task Breakdown

Backend (Django):

1. **Setup Django Project**:
   - Create a new Django project and app for inventory management.
   - Configure the SQL database.
2. **User Authentication**:
   - Implement user registration and login using Django’s authentication system.
   - Add role-based permissions (admin and user roles).
3. **Create API Endpoints**:
   - **/register**: For user registration.

    - **/login**: For user login.
    - **/products**: For viewing the product list (accessible by both roles).
    - **/products/add**: For adding new products (admin only).
    - **/products/update/<id>**: For updating product information (admin only).
    - **/products/delete/<id>**: For deleting a product (admin only).
    - **/categories**: For viewing and managing categories.

Frontend (React/Vanilla JS):

1. **Design the UI**:
    - Create a simple, user-friendly interface to display product and category information.
    - Implement forms for login, product creation, and category management.
2. **API Integration**:
    - Use **fetch** or **Axios** to interact with Django's REST APIs.
    - Ensure the UI updates dynamically when interacting with the backend (e.g., displaying newly added products without refreshing).
3. **Navigation and Routing** (if using React):
    - Implement routing for different pages (e.g., Login, Product List, Add Product).

Testing:

- Ensure all API endpoints are functional.
- Test the UI by interacting with the backend to ensure data is fetched and updated correctly.

6. Submission and Deadline Expectations

- **Deadline**: The completed project must be submitted in **7 days**.
- You should submit:
    - The complete source code (backend and frontend).
    - Documentation explaining how to set up the project locally (installation steps, environment setup, etc.).
    - A brief description of your approach to solving the problem.

7. Additional Notes/Guidance

- **Focus on practical experience**: This project is designed to help you practice key web development skills. Aim for clean, maintainable code and follow best practices for security (e.g., handling passwords securely).
- **Version control**: Use **Git** to track your progress. Commit regularly and push your changes to a **GitHub** repository.
- **User experience**: Although this is a practice project, consider how the users (admin and regular) will interact with the system. Design the UI to be intuitive and responsive.