Project Overview: **Inventory Management**

As a backend developer, your task is to design and implement an **Inventory Management API** using **Django** and **Django REST Framework**. This API will allow users to manage inventory items by adding, updating, deleting, and viewing current inventory levels. You will build a fully functional API that stores inventory data, manages users, and tracks inventory levels, simulating a real-world scenario in managing inventory for a store. The project will cover key aspects such as CRUD operations, database management, and user authentication.

Functional Requirements:

1. **Inventory Item Management (CRUD)**:
   * Implement the ability to **Create**, **Read**, **Update**, and **Delete** (CRUD) inventory items.
   * Each inventory item should have the following attributes: Name, Description, Quantity, Price, Category, Date Added, and Last Updated.
   * Ensure validation for required fields like Name, Quantity, and Price.
2. **Users Management (CRUD)**:
   * Implement CRUD operations for users.
   * Each user should have a unique Username, Email, and Password.
   * Only authenticated users should be able to manage inventory (i.e., create, update, or delete items).
   * Implement permission checks to ensure that users can only manage their own inventory items.
3. **View Inventory Levels**:
   * Create an endpoint to view the **current inventory levels** for all items.
     + Display the current Quantity available for each inventory item.
     + Allow optional filters such as Category, Price Range, or Low Stock (e.g., items with quantity below a threshold).
4. **Track Inventory Changes**:
   * Log changes to inventory quantities (e.g., when an item is restocked or sold).
   * Provide an endpoint to view the **inventory change history** for each item, showing when quantities were updated and by whom.

Technical Requirements:

1. **Database**:
   * Use **Django ORM** to interact with the database.
   * Define models for **Inventory Items** and **Users**.
   * Ensure that inventory items are associated with the users who manage them.
2. **Authentication**:
   * Implement user authentication using **Django’s built-in authentication** system.
   * Users must be logged in to manage inventory items (i.e., create, update, or delete).
   * Optionally, implement token-based authentication (JWT) for more secure API access.
3. **API Design**:
   * Use **Django Rest Framework** (DRF) to design and expose API endpoints.
   * Follow RESTful principles by using appropriate HTTP methods (GET, POST, PUT, DELETE).
   * Ensure proper **error handling**, with appropriate HTTP status codes (e.g., 404 for not found, 400 for bad request).
4. **Deployment**:
   * Deploy the API on **Heroku** or **PythonAnywhere**.
   * Ensure the API is accessible, secure, and performs well in the deployed environment.
5. **Pagination and Sorting**:
   * Implement pagination for viewing inventory items to handle large datasets efficiently.
   * Provide sorting options, such as sorting items by Name, Quantity, Price, or Date Added.

Stretch Goals (Optional):

* **Low Stock Alerts**: Implement a feature where the system sends alerts (via email or in-app) when inventory levels drop below a certain threshold.
* **Inventory Categories**: Allow users to manage inventory categories dynamically, with the ability to add, update, and delete categories.
* **Supplier Management**: Add functionality for users to manage suppliers, with a relationship between inventory items and suppliers for tracking where items were sourced from.
* **Inventory Reports**: Generate detailed inventory reports that show total value of inventory, stock levels, and sales/restocking history.
* **Barcode Scanning Integration**: Implement support for barcode scanning, allowing users to add or update inventory by scanning item barcodes.
* **Stock Reordering**: Implement a feature that automatically suggests restocking certain items when their quantity drops below a predefined level.
* **Multi-Store Support**: Extend the system to manage inventory across multiple stores or locations, with separate inventory levels for each store.