**Stock Management System GraphQL API:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Project Description:**

Develop a GraphQL API for managing stock information, including the ability to search for stock, add/modify existing stock details, and remove stock items.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**User Stories:**

1. As a user, I want to retrieve a list of existing stocks.

2. As a user, I want to search for a specific item and view its available stock quantity.

3. As a user, I want to add a new item to the stock.

4. As a user, I want to update/modify the quantity of an existing item in the stock.

5. As a user, I want to remove an item from the stock.

6. As a user, I want to view a list of available suppliers.

7. As a user, I want to view a list of available categories.

8. As a user, I want to record a stock transaction. I should provide the transaction type, date, quantity, stock item, and customer information.

**Non-Functional Requirements:**

**Data Schema Design:**

Design the database schema to accurately represent the stock management system, including tables, relationships, and constraints. Ensure the schema aligns with the GraphQL schema and supports the required operations efficiently.

**Data Integrity:**

Enforce data integrity rules at the database level to maintain the consistency and accuracy of stock data. Implement constraints, such as unique constraints, foreign key constraints, and data validation rules, to prevent invalid or inconsistent data entry.

**Database Connectivity:**

Establish a secure and efficient connection between the GraphQL API and the database.

**Code Organization**:

Follow Python best practices for code organization, including module structure, naming conventions, and separation of concerns. Use packages and modules to group related functionality together.

**Object-Oriented Programming**:

Utilize object-oriented programming principles to design and structure the codebase. Encapsulate related data and behavior within classes and use inheritance and composition where appropriate.

**Error Handling:**

Implement proper error handling mechanisms using exception handling to handle errors and exceptions. Provide informative error messages to assist with troubleshooting and debugging.

**Code Readability:**

Write clean, readable, and well-documented code. Follow the Python style guide (PEP 8) for naming conventions, indentation, comments, and docstrings.

**Input Validation:**

Validate user input and perform data sanitization to ensure the integrity and security of the application. Use appropriate Python libraries or regular expressions to validate and sanitize input data.

**Unit Testing:**

Write unit tests to verify the correctness of individual components or functions within the codebase. Use testing frameworks unittest to automate the testing process.

**Virtual Environment Creation:**

The Stock Management system should support the creation of a dedicated virtual environment for development and deployment purposes. The virtual environment should be isolated and independent, ensuring that the system's dependencies and configurations do not conflict with other Python packages or system dependencies.

**Package Management**:

Use a package management tool like pip to manage dependencies and ensure consistent package versions. Define dependencies in a requirements.txt or a similar file for easy installation and reproducibility.

**Schema Design:**

Define a GraphQL schema that represents the stock management system, including the necessary types, queries, and mutations.

**Error Handling**:

Implement error handling mechanisms in the API to provide appropriate error messages for incorrect input or encountered errors.

**Input Validation**:

Validate user input and enforce data integrity rules in the GraphQL resolvers to ensure correct and valid data entry.

**Performance**:

Optimize the API's performance by efficiently fetching and manipulating stock data, minimizing unnecessary data transfers.

**Localization:**

Utilize appropriate date and time formatting based on the user's locale when displaying timestamps or other time-related information.

**Scalability**:

Design the API to handle a growing number of users and increasing stock data without compromising performance or functionality.

**Documentation**:

Create comprehensive documentation for the GraphQL API, including schema definitions, available queries, mutations, input types, and usage examples.

**Sorting and Filtering:**

Enable sorting and filtering capabilities for the stock list, allowing users to sort stocks by attributes such as name, quantity, or date added, and filter stocks based on specific criteria.