Inventory Management System - Documentation

# 1. Introduction and Concept

An Inventory Management System is designed to track and manage stock levels, purchases, and sales. It helps organizations monitor available inventory, track stock movement, and maintain updated records. Key functions include tracking available items, their quantity, and updates in stock over time.

# 2. Database Schemas and Entity Descriptions

## 2.1 Item Table

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary Key |
| name | String | Name of the item |
| description | Text | Optional details |
| quantity | Integer | Quantity in stock |
| price | Float | Price per unit |
| category\_id | ForeignKey | Linked to Category |
| supplier\_id | ForeignKey | Linked to Supplier |
| created\_at | DateTime | Auto timestamp |
| updated\_at | DateTime | Auto timestamp |

## 2.2 Category Table

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary Key |
| name | String | Category name |
| description | Text | Optional |

## 2.3 Supplier Table

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary Key |
| name | String | Supplier name |
| contact\_info | String | Phone/email |
| address | Text | Address |

## 2.4 User Table

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary Key |
| username | String | Unique login name |
| hashed\_password | String | Secure password |
| role | String | admin / viewer |

## 2.5 StockTransaction Table

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| id | Integer | Primary Key |
| item\_id | ForeignKey | Linked to Item |
| change\_type | String | add, remove, adjust |
| quantity | Integer | Quantity changed |
| user\_id | ForeignKey | Who made the change? |
| timestamp | DateTime | When it happened |
| notes | Text | Optional remark |

## 2.6 Table Summary

|  |  |  |
| --- | --- | --- |
| Table Name | Required? | Purpose |
| items | Yes | Store inventory items |
| categories | Yes | Group similar items |
| suppliers | Yes | Manage supplier info |
| users | Optional | For login/auth (JWT/token) |
| stock\_transactions | Optional | History of changes (audit/logs) |

# 4. Implementation Process

The system is structured in a modular format for scalability and clarity. Key files and their responsibilities include:

- main.py: Entry point for FastAPI app  
 - models.py: Contains ORM table definitions  
 - schemas.py: Data validation models using Pydantic  
 - crud.py: Database operation functions  
 - settings.py: Environment configuration management  
 - database.py: Async session and engine setup  
 - streamlit\_app.py: Visualization and reporting  
 - routers/inventory.py: API route definitions  
 - .env: Environment-specific secrets  
 - requirements.txt: Dependency file

# 5. API Route Definitions

|  |  |  |  |
| --- | --- | --- | --- |
| ENTITY | OPERATION | ROUTE | DESCRIPTION |
| Category | Create | /inventory/categories/ (POST) | Add a new category |
| Category | Get All | /inventory/categories/ (GET) | List categories |
| Category | Get One | /inventory/categories/{id} (GET) | Get specific category |
| Supplier | Create | /inventory/suppliers/ (POST) | Add a new supplier |
| Supplier | Get All | /inventory/suppliers/ (GET) | List suppliers |
| Supplier | Get One | /inventory/suppliers/{id} (GET) | Get specific supplier |
| Item | Create | /inventory/items/ (POST) | Add a new item |
| Item | Get All | /inventory/items/ (GET) | List all items |
| Item | Get One | /inventory/items/{id} (GET) | Get specific item |
| Item | Update | /inventory/items/{id} (PUT) | Update item details |
| Item | Delete | /inventory/items/{id} (DELETE) | Delete an item from stock |

# 6. Additional Implementation Notes

The application uses asynchronous functions for improved performance. Libraries used include `asyncpg` and `aiosqlite`. All CRUD operations and router handlers are implemented using `async def` for better concurrency handling. `settings.py` is utilized to load environment variables using Pydantic’s BaseSettings class.

# 7. Models and Schema of Database

## 7.1. ORM Data Models (models.py)

Defines SQLAlchemy-based classes mapping to database tables with relationships.

* **Category**: Represents item groups. One-to-many relationship with items.
* **Supplier**: Stores vendor details. Linked to multiple items.
* **Item**: Core entity holding inventory data like quantity, price, category, and supplier references.
* **User**: Manages authentication with roles (admin/viewer). Linked to stock transactions.
* **StockTransaction**: Logs inventory changes with metadata like type, quantity, user, and timestamp.

ORM relationships enable data consistency and referential integrity.

## 7.2. Data Validation Models (schemas.py)

Pydantic models for input validation and API response formatting.

* **Category / Supplier / Item Schemas**: Handle create/read operations. Ensure correct data types and structure.
* **StockTransaction Schemas**: Structure for inventory audit records.
* **User Schemas**: Manage registration, login, and roles.
* **Auth Schemas**: Handle JWT-based authentication and token handling.

These schemas enhance data safety, serialization, and auto-doc generation in FastAPI.

# 8. Project Directory Setup

