- ***Arash Khajooei***
- ***[arash.khajooei@gmail.com]***

FastAPI Warehouse Management System

This FastAPI application represents my implementation of a Warehouse Management System (WMS). Through this system, I've designed a platform to efficiently oversee products, deliveries, exits, and transactions within a warehouse setting. The foundation of this application is built upon the powerful combination of ***SQLAlchemy*** for seamless database interaction and ***Pydantic*** for ensuring robust data validation.

Dependencies and Imports

The application starts with importing the required modules and libraries:

- **FastAPI:** This is the main module for creating the FastAPI application.
- **HTTPException:** This is used to raise HTTP exceptions with specific status codes and detail messages.
- **create_engine:** This is used to create a database engine using SQLAlchemy.
- **Column, Integer, String, etc.:** These are SQLAlchemy constructs for defining the database schema.
- **declarative_base:** This is used to create a base class for declarative models in SQLAlchemy.
- **sessionmaker:** This is used to create a session factory for creating database sessions.
- **relationship:** This is used to define relationships between tables in the database.
- **BaseModel:** This is used to define Pydantic models for data validation and serialization.
- **Optional, Union, List:** These are used to define types for function parameters.
- **date, datetime:** These are used for working with dates and times.

Application Structure

The FastAPI Warehouse Management System is organized into several sections:

- **Imports and Dependencies:** I start by importing the necessary modules and libraries. These include FastAPI, SQLAlchemy, Pydantic, and more. These modules are crucial for building the web application, defining the database schema, handling data validation, and managing dependencies.
- **Creating the FastAPI App:** I create the FastAPI app using the FastAPI class. This app handles incoming HTTP requests, processes them, and generates appropriate responses.
- **SQLAlchemy Setup:** I configure the connection to the database in this section. I specify the URL for the database and create an SQLAlchemy engine. Additionally, I define a session factory called SessionLocal to handle individual database sessions.
- **Pydantic Models:** I use Pydantic models for data validation and serialization. These models define the structure and data types of various entities within the application, such as products, deliveries, exits, and transactions.

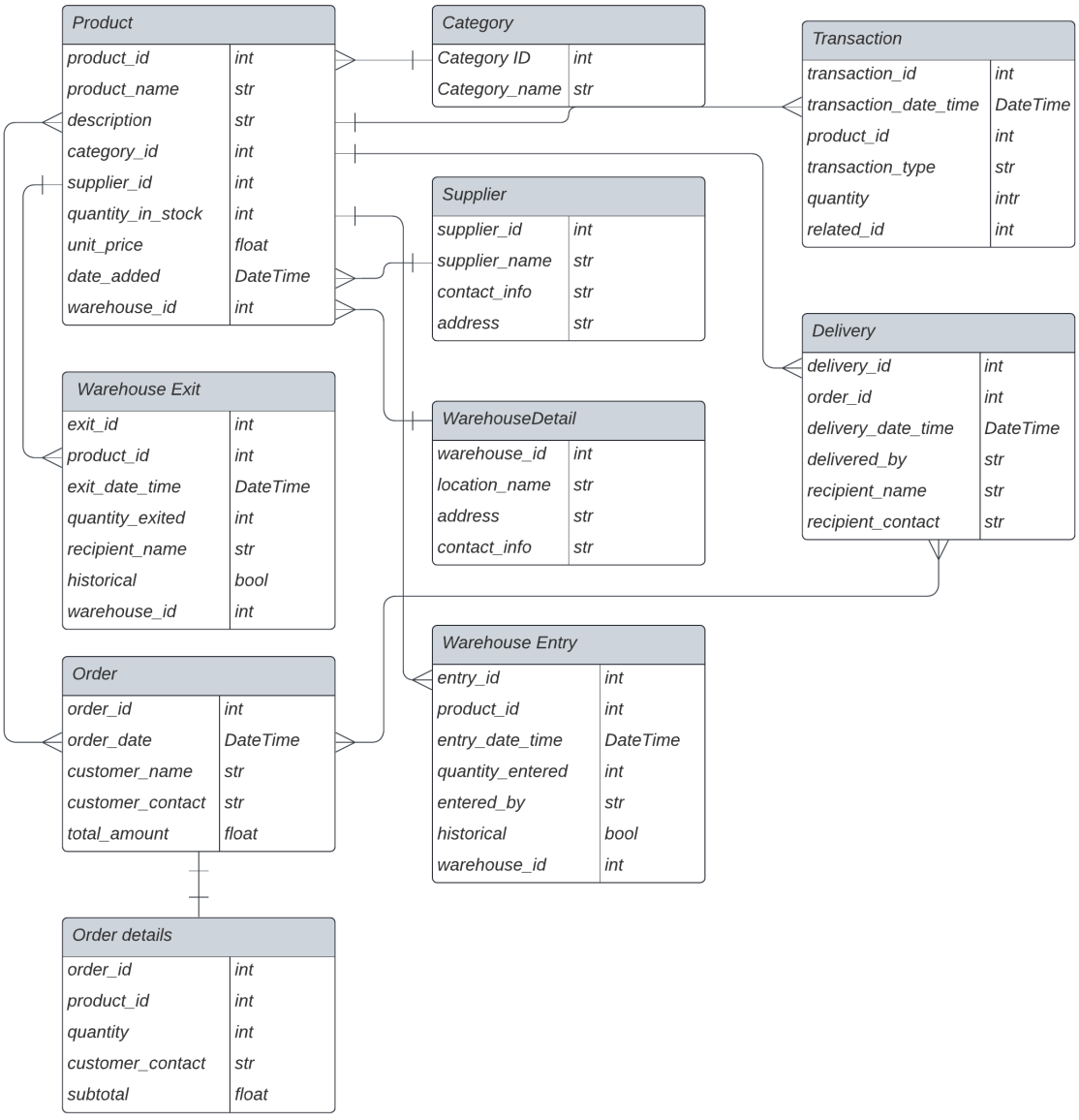
- **SQLAlchemy Models:** The SQLAlchemy models represent the database tables and their relationships. These models define the schema for storing information about products, suppliers, orders, deliveries, and other entities.
- **Creating Database Tables:** The SQLAlchemy models are used to create the corresponding database tables using the create_all method. This ensures that the database schema aligns with the application's model definitions.
- **Dependency for Database Session:** The get_db function serves as a dependency that provides a database session to the endpoint functions. This helps manage database connections and sessions efficiently.
- **Root Endpoint:** The root endpoint / is defined to provide a welcome message to users accessing the API.
- **Endpoints for CRUD Operations:** The application defines several endpoints for performing CRUD (Create, Read, Update, Delete) operations on different entities. These endpoints handle product creation, fetching products with deliveries, exiting products from the warehouse, delivering products to the warehouse, and fetching transaction records.

Mechanisms

Here's how the FastAPI Warehouse Management System operates:

- **Product Creation:** Users can create new products by sending a POST request to /products/. This endpoint expects a JSON payload containing product information. Upon creation, the product details are stored in the database, and a corresponding delivery entry is created to mark the product's arrival in the warehouse.
- **Fetching Products with Deliveries:** Users can fetch a list of products along with their associated delivery details. They can specify date ranges using query parameters to filter the results to products delivered within a specific timeframe. The application retrieves the relevant data from the database and structures it for response.
- **Exiting Products from Warehouse:** Users can initiate the process of removing products from the warehouse by sending a POST request to /products/{product_id}/exit/. This decreases the quantity of the specified product in stock, creates a warehouse exit entry, and records a transaction.
- **Delivering Products to Warehouse:** Users can deliver products to the warehouse by sending a POST request to /products/{product_id}/deliver/. This increases the quantity of the specified product in stock, creates a warehouse entry, and records a transaction.
- **Fetching Product Transactions:** Users can fetch a list of transactions associated with a specific product by sending a GET request to /products/{product_id}/transactions/. The application retrieves the relevant transaction records from the database and returns them as a response.

Entity Relationship Diagram (ERD)



This diagram depicts the main entities in the FastAPI Warehouse Management System and their relationships. Here's a brief explanation of each entity:

- **Category:** Represents product categories.

- **Supplier:** Represents suppliers of products.
- **WarehouseDetail:** Represents warehouse details such as location, address, and contact information.
- **Product:** Represents individual products with their details, including name, description, category, supplier, quantity in stock, unit price, and more.
- **Transaction:** Records various types of transactions related to products, such as additions, exits, and deliveries.
- **Delivery:** Represents deliveries of products to the warehouse, including delivery details and recipient information.
- **WarehouseExit:** Represents products leaving the warehouse, including exit details and recipient information.
- **WarehouseEntry:** Represents products entering the warehouse, including entry details.

The relationships are represented as lines connecting the entities:

- Products are associated with a Category, a Supplier, and a Warehouse.
- Transactions reference Products to track changes.
- Deliveries, Exits, and Entries all reference Products and may have associated Transactions.
- WarehouseExits and WarehouseEntries are associated with a Warehouse.

- **One-to-Many Relationship: Category to Products**
    - One category can be associated with many products, but each product can be associated with only one category.
    - This relationship is represented in the ProductModel entity as category_id, which is a foreign key referencing the category_id in the CategoryModel.
- **One-to-Many Relationship: Supplier to Products**
    - One supplier can be associated with many products, but each product can be associated with only one supplier.
    - This relationship is represented in the ProductModel entity as supplier_id, which is a foreign key referencing the supplier_id in the SupplierModel.
- **One-to-Many Relationship: WarehouseDetail to Products**
    - One warehouse can be associated with many products, but each product can be associated with only one warehouse.
    - This relationship is represented in the ProductModel entity as warehouse_id, which is a foreign key referencing the warehouse_id in the WarehouseDetailModel.
- **One-to-Many Relationship: Product to Transactions**
    - One product can have many transactions, but each transaction is associated with only one product.
    - This relationship is represented in the TransactionModel entity as product_id, which is a foreign key referencing the product_id in the ProductModel.
- **One-to-Many Relationship: Product to Deliveries**
    - One product can have multiple delivery records, but each delivery record is associated with only one product.

    - This relationship is represented in the DeliveryModel entity as product_id, which is a foreign key referencing the product_id in the ProductModel.
- **One-to-Many Relationship: Product to WarehouseEntries**
    - One product can have multiple warehouse entry records, but each warehouse entry is associated with only one product.
    - This relationship is represented in the WarehouseEntryModel entity as product_id, which is a foreign key referencing the product_id in the ProductModel.
- **One-to-Many Relationship: Product to WarehouseExits**
    - One product can have multiple warehouse exit records, but each warehouse exit is associated with only one product.
    - This relationship is represented in the WarehouseExitModel entity as product_id, which is a foreign key referencing the product_id in the ProductModel.
- **Many-to-One Relationship: Products to Category, Supplier, Warehouse**
    - Many products can be associated with one category, supplier, and warehouse.
    - These relationships are represented through the category, supplier, and warehouse attributes in the ProductModel.
- **Many-to-Many Relationship: Products to Orders**
    - Products can be associated with many orders, and orders can contain multiple products.
    - This relationship is indirectly represented through the OrderDetailsModel, which serves as a join table between ProductModel and OrderModel.
- **Many-to-Many Relationship: Orders to Deliveries**
    - Orders can have multiple delivery records, and deliveries can be associated with multiple orders.
    - This relationship is represented through the deliveries attribute in the OrderModel.

How to run the code:

Prerequisites:

- Make sure you have Python installed on your system (Python 3.6 or higher).
- Install the required packages by running pip install fastapi uvicorn sqlalchemy pydantic.

Tutorial: Running FastAPI Application with Uvicorn Server and Swagger UI

HERE IS A SIMPLE TUTORIAL WHICH I PROVIDE TO USE THE FASTAPI Swagger UI

Download and Organize Your Code:

- Download your code and save it in a folder on your local machine.
- Organize the code files and make sure you have all the required imports and dependencies.

Navigate to the Code Directory:

- Open a terminal or command prompt.
- Use the cd command to navigate to the directory where you saved your code.

Running the Uvicorn Server:

- In the terminal, run the following command to start the Uvicorn server:

uvicorn main:app --host 0.0.0.0 --port 8000

- This command tells Uvicorn to run the FastAPI app defined in the main.py file. The --host and --port flags specify the host address and port number. Change these values if needed.

Access Swagger UI:

- Once the server is up and running, open your web browser.
- Navigate to the following URL to access the Swagger UI for your API:

http://localhost:8000/docs

- We'll see the Swagger UI interface that allows you to interact with your API endpoints.

Explore and Test Endpoints:

- In the Swagger UI, you'll see a list of available endpoints and methods.
- Click on an endpoint to expand its details.
- Click the "Try it out" button to test the endpoint. You can enter parameter values and click "Execute" to see the response.

Creating Products:

- To create a product, scroll down in Swagger UI to find the "POST /products/" endpoint.
- Click on the endpoint to expand it.
- Click the "Try it out" button.
- In the "Request Body" section, provide the necessary details for the product.
- Click "Execute" to create the product. You'll see the response with the created product details.

Viewing Products with Deliveries:

- To view products with deliveries within a specific date range, find the "GET /products/" endpoint.
- Click on the endpoint to expand it.
- Enter the desired start and end dates in the query parameters.
- Click "Execute" to fetch the products along with delivery information for the specified date range.

Exiting and Delivering Products:

- To exit a product from the warehouse or deliver a product, find the corresponding "POST /products/{product_id}/exit/" or "POST /products/{product_id}/deliver/" endpoints.
- Click on the endpoint to expand it.
- Enter the required information in the request body (for exits) or parameters (for deliveries).
- Click "Execute" to perform the exit or delivery action. You'll see the updated product details in the response.

Testing Transactions:

- To test transactions, find the "GET /transactions/" endpoint.
- Click on the endpoint to expand it.
- Click "Execute" to fetch the transaction records.

Exploring Other Endpoints:

- Feel free to explore other endpoints and test different actions using the Swagger UI.

Stopping the Server:

- When you're done testing, go back to the terminal where the Uvicorn server is running.
- Press Ctrl + C to stop the server.

The source of project also has provided on GitHub:

https://github.com/arashkhgit/FastAPI

SCAN ME

Thank you for giving me this opportunity to join your team. I am excited about the possibility of joining your company and contributing my skills to your innovative projects. I am eager to be part of your talented workforce and make a positive impact. Looking forward to your decision.

Sincerely,

ARASH KHAJOOEI

8 AUGUST 2023