E-commerce Admin API

GitHub Repository: https://github.com/hashaam-011/Backend-Development

A FastAPI-based backend API for e-commerce management, providing detailed insights into sales, revenue, and inventory status, as well as allowing new product registration.

Features

- **Sales Management**
  - Retrieve, filter, and analyze sales data
  - Analyze revenue (daily, weekly, monthly, annual)
  - Compare revenue across periods and categories
  - Get sales data by date range, product, and category
- **Inventory Management**
  - View current inventory status
  - Low stock alerts
  - Update inventory levels
  - Track inventory changes over time
- **Product Management**
  - Register new products
  - View, update, and delete products

Technical Stack

- Python 3.8+
- FastAPI
- PostgreSQL (or MySQL, with minor changes)
- SQLAlchemy
- Pydantic

Setup Instructions

1. **Clone the repository**

```
git clone https://github.com/hashaam-011/Backend-Development.git
cd ecommerce-admin-api
```

2. **Create and activate a virtual environment**

```
python -m venv venv
# Windows
.\venv\Scripts\activate
# Linux/Mac
source venv/bin/activate
```

3. **Install dependencies**

```
pip install -r requirements.txt
```

4. **Configure environment variables**

   - Create a `.env` file in the root directory:

   ```
   DB_HOST=localhost
   DB_PORT=5432
   DB_NAME=ecommerce
   DB_USER=postgres
   DB_PASS=your_password
   ```

5. **Initialize the database**

```
python init_db.py
```

6. **Populate demo data**

```
python populate_demo_data.py
```

7. **Run the application**

```
uvicorn app.main:app --reload
```

API Endpoints

Base URL: `http://localhost:8000/api/v1`

Products

- `GET /products/` — List all products

- `POST /products/` — Create new product
- `GET /products/{id}` — Get product details
- `PUT /products/{id}` — Update product
- `DELETE /products/{id}` — Delete product

Sales

- `POST /sales/` — Record new sale
- `GET /sales/range` — Get sales by date range
- `GET /sales/revenue` — Get revenue analysis (daily, weekly, monthly, annual)
- `GET /sales/product/{product_id}` — Get sales for a specific product

Inventory

- `GET /inventory/` — List all inventory items
- `PUT /inventory/{product_id}` — Update inventory level
- `GET /inventory/low-stock` — Get low stock alerts

Database Documentation

Database Schema Overview

This project uses a normalized relational database schema to support e-commerce admin operations. The main tables and their purposes are:

1. Categories

- **Purpose:** Stores product categories (e.g., Electronics, Home Appliances).
- **Fields:**
  - `id` (Primary Key): Unique identifier for each category.
  - `name` (String): Name of the category.

2. Products

- **Purpose:** Stores all products available for sale.
- **Fields:**
  - `id` (Primary Key): Unique identifier for each product.
  - `name` (String): Product name.
  - `category_id` (Foreign Key): References `categories.id` to associate a product with a category.
  - `price` (Float): Product price.
  - `created_at` (DateTime): Timestamp when the product was added.

3. Sales

- **Purpose:** Records each sale transaction.
- **Fields:**
  - `id` (Primary Key): Unique identifier for each sale.
  - `product_id` (Foreign Key): References `products.id` to indicate which product was sold.

- `quantity` (Integer): Number of units sold.
- `total_price` (Float): Total price for the sale.
- `sale_date` (DateTime): Timestamp of the sale.

4. Inventory

- **Purpose:** Tracks current stock levels for each product.
- **Fields:**
  - `id` (Primary Key): Unique identifier for each inventory record.
  - `product_id` (Foreign Key): References `products.id` to indicate which product's inventory is being tracked.
  - `stock_level` (Integer): Current stock level.
  - `updated_at` (DateTime): Timestamp of the last inventory update.

Relationships

- **products.category_id → categories.id** Each product belongs to a category.

- **sales.product_id → products.id** Each sale is linked to a specific product.

- **inventory.product_id → products.id** Each inventory record is linked to a specific product.

Indexing & Normalization

- All primary keys and foreign keys are indexed for optimized query performance.
- The schema is normalized to avoid redundancy and maintain data consistency.

Demo Data

To populate the database with sample data for Amazon and Walmart products, run:

```
python populate_demo_data.py
```

Testing

- Use Swagger UI at http://localhost:8000/docs for interactive API testing.
- Or use Postman with the provided endpoints.
