Mini Project: E-commerce API

After six weeks of learning coding, we have mastered the basics of Python programming. This project is the best summary of this period of learning.

The requirement for this project is:

1.Customer and CustomerAccount Management:

Create the CRUD (Create, Read, Update, Delete) endpoints for managing Customers and their associated CustomerAccounts:

2. Product Catalog:

Create the CRUD (Create, Read, Update, Delete) endpoints for managing Products:

3. Order Processing:

Develop comprehensive Orders Management functionality to efficiently handle customer orders, ensuring that customers can place, track, and manage their orders seamlessly.

4. Database Integration:

5. Data Validation and Error Handling:

6. User Interface (Postman):

7. GitHub Repository:

8. Effective Project Communication:

To save time and maintain consistency, we reusing the Flask-SQLAlchemy project as a foundation for Managing a Fitness Center Database.

This approach ensures a unified and efficient codebase, making it easier to integrate new features into the existing solution.

1. Create a new directory for the project:

2. Set up a virtual environment:

3. Activate the virtual environment:

4. Install Necessary Packages:

5. Create a Python file app.py for Flask application and set up the database connection.

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+mysqlconnector://root:8832@localhost/ecom'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

# init SQLAlchemy

db = SQLAlchemy(app)

# init Marshmallow

ma = Marshmallow(app)

1. Create the CRUD (Create, Read, Update, Delete) endpoints for managing Customers and their associated CustomerAccounts:

# Create new customer

@app.route('/customers', methods=['POST'])

def add\_customer():

# Get all customers

@app.route('/customers', methods=['GET'])

def get\_customers():

@app.route('/customers/<int:id>', methods=['GET'])

def get\_customer(id):

@app.route('/customers/<int:id>', methods=['PUT'])

def update\_customer(id):

# Delete customer

@app.route('/customers/<int:id>', methods=['DELETE'])

def delete\_customer(id):

2. Create the CRUD (Create, Read, Update, Delete) endpoints for managing Products:

View and Manage Product Stock Levels (Bonus): Create an endpoint that allows to view and manage the stock levels of each product in the catalog. Administrators should be able to see the current stock level and make adjustments as needed.

# Product

@app.route('/products', methods=['POST'])

def create\_product():

@app.route('/products', methods=['GET'])

def get\_all\_products():

@app.route('/products/<int:id>', methods=['GET'])

def get\_product(id):

@app.route('/products/<int:id>', methods=['PUT'])

def update\_product(id):

@app.route('/products/<int:id>', methods=['DELETE'])

def delete\_product(id):

Administrators can see which current stock level below the threshold and make adjustments as needed.

@app.route('/check\_stock\_levels', methods=['GET'])

def check\_stock\_levels():

Restock Products When Low (Bonus): Develop an endpoint that monitors product stock levels and triggers restocking when they fall below a specified threshold. Ensure that stock replenishment is efficient and timely.

@app.route('/restock', methods=['POST'])

def restock\_products():

3. Develop comprehensive Orders Management functionality to efficiently handle customer orders, ensuring that customers can place, track, and manage their orders seamlessly.  
**Place Order:**

**Retrieve Order:**

**Track Order:**

**Manage Order History (Bonus):**

@app.route('/customers/<int:customer\_id>/orders', methods=['GET'])

def get\_order\_history(customer\_id):

4. Database Integration:

5. Data Validation and Error Handling:

6. User Interface (Postman):

The most challenging are Develop comprehensive Orders Management function.

We need buildup a relationship between the Customers, Orders and Products in Models of Customers, Orders and Products.