# E-Commerce Platform Microservice

**Introduction:**

The goal of this project is to create a microservice for a e-commerce platform and with the micro service implementation, have independent services for handling individual functionality like managing products, users, reviews, discounts, and orders. For this project I used fastAPI to implement each service. The microservice approach ensures each service is scalable and independent from other services allowing for services to be created and changed independently without affecting other service implementation.

**Use Case Definitions:**

The use cases for this platform are centered around how a customer and employee would use this platform to manage their products online. Each service is responsible for managing a specific operation in the system. Customers should be able to create and manage their accounts, browse products,  order products, track deliveries, and add reviews for products they have ordered. Employees should be able to manage products and discounts for products

**Service and Data Architecture**

The architecture is designed around microservices, each responsible for a specific function:

- Product Service: Manages product-related information like product name, description, links, and images
- Order Service: Handles order creation, updating, and tracking orders. Contains product information on what was ordered, amounts ordered, and delivery dates.
- Discount Service: Manages discount information and calculates pricing after applying discounts.
- Customer Service: Manages customer reviews and review information like rating, name, and review contents

**Service Communication**

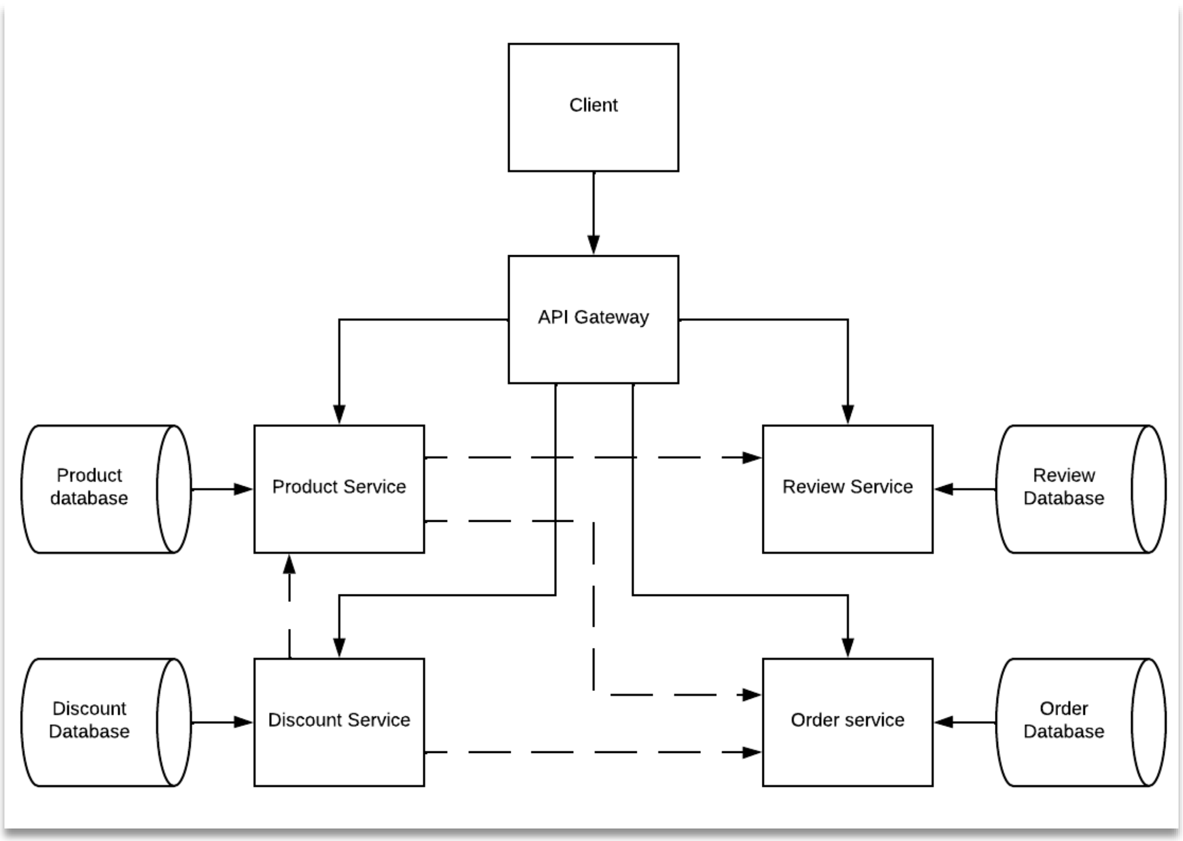
The services communicate using REST APIs, allowing them to send HTTP requests to each other. For instance, the product service makes HTTP requests to the discount service to get discount details on the product and the order service gets information from the product service to get product details and the discount service to get up to date pricing for the order. Services communicate via RESTful API calls using HTTP to make GET, POST, PUT, and DELETE requests and uses HTTPX for asynchronous communication between services, ensuring efficient data retrieval.

**Data Management**

Each service has its own database for storing data relevant to its operation. I am using MongoDB for storing products, orders, reviews, and discounts information. Services are loosely coupled, and the interaction is handled via HTTP requests, ensuring that services can scale independently.

**Architecture**

The architecture consists of 4 services and their databases. The client connects to the API gateway which they can navigate to any of the services they need to use. Each service has a database that only it can access. The connections between services allow each to get the data from another service without going directly to the dataset potentially changing the data of a different service.



**Delivered Product**

- Product Service: Manages product creation, updates, and retrieval. This service interacts with the Discount Service to retrieve discount details for products when creating a new product or updating a product.
- Order Service: Handles order creation and calculates total cost by interacting with the Product Service and the Discount Service. It tracks order status and updates the delivery information.
- Discount Service: Provides discount information based on the product ID. It manages discount percentages, actual prices, and discounted prices.
- Review Service: Allows users to submit reviews of products. Reviews are associated with a product ID and user id and stored with user details.

## What I Learned

- **Microservices Design**: I gained experience in designing microservices-based architecture where each service is responsible for a specific domain. This approach offers flexibility and scalability, allowing each service to evolve independently.
- **Inter-Service Communication**: I learned how to use HTTP requests for communication between services and how to handle asynchronous tasks efficiently using libraries like HTTPX.
- **Database Integration**: Each service has its own database, which required managing multiple databases and ensuring that services could interact with each other via API calls.

## If I Had More Time

If I had more time I would expand the services to include a user service which would include user authentication and creation to better show how a customer would interact with this architecture. I would also want to revamp some of my databases as there some information I'm missing that I want to include like the order service doesn't include the name of the product, adding a verify purchase to reviews and make a better GUI to visualize looking through and ordering products.