PAD -- Project 1 Checkpoint 1

> ***Performed by:*** *Cristian Boris, group FAF-202*

Online Shoe Catalog and Warehouse Stock Management

Assess Application Suitability (2pts)

Modularity and Scalability:

An online shoe catalog and warehouse stock management system can benefit from microservices by breaking down the application into smaller, manageable modules. Each microservice can handle specific functionalities, such as catalog management and inventory tracking. This modularity enables easy scalability by adding or modifying microservices as needed.

Independent Development and Deployment:

Microservices allow different teams to work on distinct components independently. For instance, the catalog team can make updates or add new shoe listings without affecting the warehouse stock management service. This independent development and deployment facilitate faster feature releases and bug fixes.

Technology Stack Flexibility:

Different microservices can use different technology stacks and databases that best suit their requirements. For instance, the catalog microservice might use a NoSQL database for flexibility in handling product information, while the inventory management microservice could rely on a relational database for precise stock tracking.

Enhanced Performance:

By optimizing each microservice for its specific task, you can achieve enhanced performance. For instance, the catalog microservice can be fine-tuned for fast product searches and recommendations, while the warehouse stock management microservice can focus on efficient inventory updates.

Define Service Boundaries (2pts)

In my case, each microservice (catalog service and inventory service) will be separated into this way: the catalog microservice is only responsible for product management, i.e. catalog microservice is NOT giving you stocks, but only works with shoes (defining the shoes parameters, and adding or retrieving shoes to/from catalog). The inventory service is working only with product ID, and is storing Date, Quantity and OperationType for a given Product Id. (Figure 1.)

Choose Technology Stack and Communication Patterns (2pts)

In my specific case of an online shoe catalog and inventory management system, RESTful APIs should serve well for most interactions between these two microservices, as they are typically used for immediate, user-initiated actions like browsing products or checking stock availability. I will use F# for both microservices with Giraffe Framework. Gateway will be written in Golang.

Design Data Management (3pts)

Each service will have its own database. Catalog Service's Database will store (ShoeId, Color, Size, Price, Brand, Category, Model) while Inventory Service will store (productID, Date, Quantity, OperationType). Let's define all the endpoints for each service.

Catalog Service

GET /product

RESPONSE EXAMPLE : [{"ID":"1", "BRAND":"GUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"}, {"ID":"2", "BRAND":"GUCCI", "MODEL": "B323", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"}]

GET /product/:Id

RESPONSE EXAMPLE : {"ID":"1", "BRAND":"GUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"} || HTTP 404 NOT FOUND

#

POST /product

PAYLOAD EXAMPLE: {"BRAND":"GUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"}
RESPONSE EXAMPLE : {"ID":"1", "BRAND":"GUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"}

#

PUT /product/:Id

PAYLOAD EXAMPLE: {"BRAND":"AGUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"}
RESPONSE EXAMPLE : {"ID":"1", "BRAND":"AGUCCI", "MODEL": "B322", "CATEGORY":"FORMAL", "SIZE":"39", "COLOR":"BLACK" , "PRICE":"300"} || HTTP 404 NOT FOUND

#

DEL /product/:Id

RESPONSE EXAMPLE: HTTP 200 OK || HTTP 404 NOT FOUND

Inventory Service

POST /warehouse

PAYLOAD EXAMPLE: {"PRODUCTID":"1", "OPERATIONTYPE":"-1", "QUANTITY":"10", "DATE":"2023-09-17 12:34:56"}
RESPONSE EXAMPLE: HTTP 200 OK || HTTP 400 BAD REQUEST (NO SUCH QUANTITY IN STOCK TO RETRIEVE.)

#

GET /stock/:productId

RESPONSE EXAMPLE: 6

#

GET /turnaround/:productId/:opType

RESPONSE EXAMPLE: 5

#

GET /turnaround/:productId/:opType/:sinceWhen/:UntilWhen

RESPONSE EXAMPLE: 1

Set Up Deployment and Scaling (1pt)

Containerization with Docker:

> *Containerize each microservice independently using Docker, ensuring isolation of dependencies and consistent environments.*

Docker Compose for Orchestration:

> *Define microservices, configurations, and dependencies in a docker-compose.yml file, simplifying multi-service management. Leverage Docker Compose for service orchestration and networking setup.*

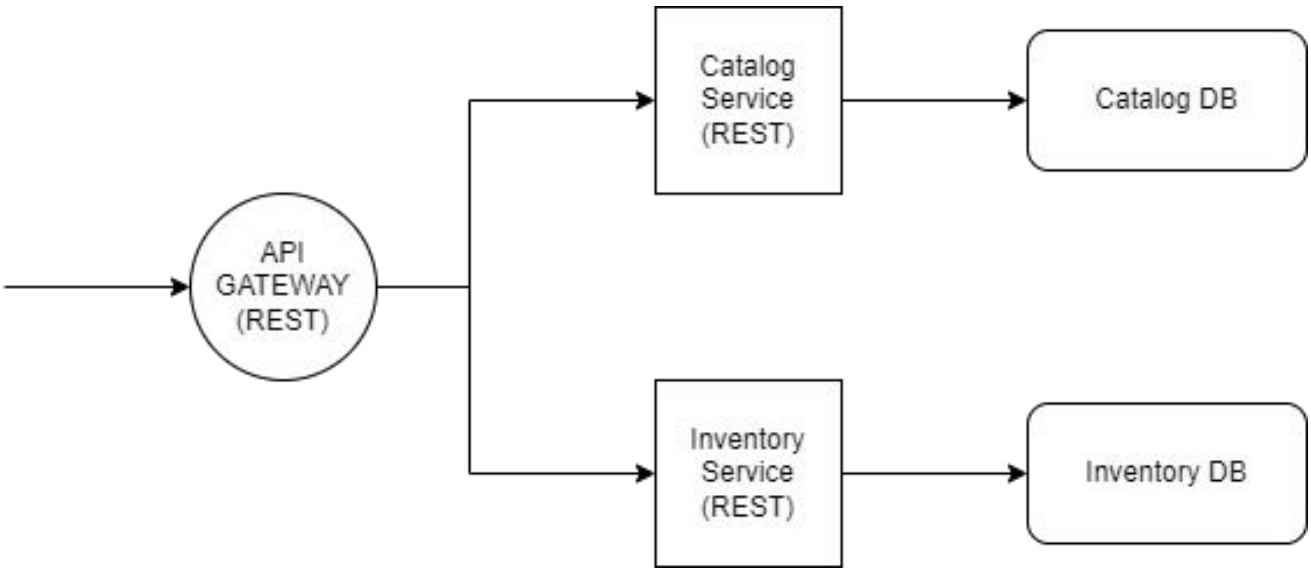

Figure 1.