MZUMBE UNIVERSITY

FACULTY OF SCIENCE AND TECHNOLOGY

(FST)

COURSE NAME	:	DISTRIBUTED SYSTEM
COURSE CODE	:	CSS 311
LECTURER NAME	:	MR. LUNODZO
TASK	:	GROUP ASSIGNMENT
PROJECT	:	09

GROUP MEMBERS

S/N	NAME	REGISTRATION NUMBER
1	KURHO K MATIKO	1432004/T.20
2	DANIEL M. MASUBO	14320039/T.20
3	DENIS D. ELIAS	14320014/T.20
4	RENATUS M. PIUS	14322049/T.20

TABLE OF CONTENTS

LIST OF FIGURES

ACCRONYMS AND ABREVIATION

IPC	Inter Process Communication
CRUD	Create, Retrieve, Update and Delete
HTTP	Hyper Text Transfer Protocol
API	Application Programming Interface
URL	Uniform Resource Locators

CHAPTER ONE

INTRODUCTION

1.1 Project Definition of terms

Microservices: are small applications with a single responsibility that can be deployed, scaled, and tested independently. Microservice architecture is a style of developing software as a collection of independent services. Each service is running on its own process that is independent from other processes and can be deployed separately from other services(Shafabakhsh et al., 2020)

Micro-Architecture: Microservices is a specific type of software architecture that breaks down a monolithic application into smaller, independent services.

RESTful APIs: Microservices communicate with each other using RESTful APIs, a type of web service that uses HTTP requests to perform CRUD operations.

Service Registry: A service registry is used to manage and discover services in a microservices architecture.

API Gateway: An API gateway acts as a single-entry point for all client requests, routing requests to the appropriate service

1.2 Project Description

This project aims to design and implement a microservices architecture for a Spring Boot application that manages the interactions between vendors, products, customers, orders, and inventory. The goal of the project is to create a flexible, scalable, and maintainable solution that can handle a high volume of transactions and provide real-time data to users.

1.3 Microservice qualities

Performance. Microservices in general present lower performance than monoliths. Final performance depends on several factors, including network aspects (latency for instance) and virtualization (deployment in virtual machines imposes additional performance overhead). **Reliability.** Again, microservices are, in general and by nature given their distributed essence, less reliable than monoliths.

Availability. In microservices architectures, the availability of a system depends on the availability of the microservices but also on the integration of such components. On the other hand, microservices reduce deployment time, increasing availability

Maintainability. Microservices are, by nature, independent, making maintainability one of the best aspects in this approach (Larrucea, XabierLarrucea, X., Santamaria, I., Colomo-palacios, R., & Ebert et al., 2021).

CHAPTER TWO

TECHNOLOGY USED

2.1 Backend Design

In this application, the following were the technologies used

Language used: JAVA

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible

Framework: Spring Boot

Spring Boot is a tool that makes developing web applications and microservices with Spring Framework faster and easier through three core capabilities:

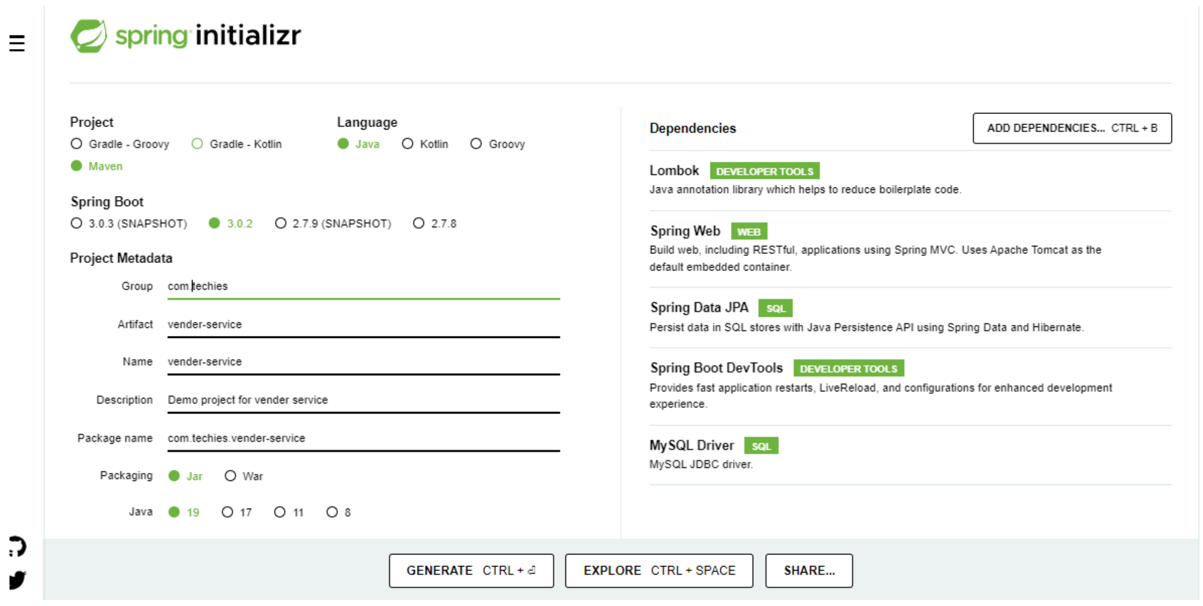
1. Autoconfiguration
2. An opinionated approach to configuration
3. The ability to create standalone applications

Databases: The application uses MYSQL AND MONGODB databases where product and order microservices uses MONGODB and inventory, customer and vendor microservices uses MYSQL databases.

2.2 Tools Used

- **Spring Initializer**

  The Spring Initializer is ultimately a web application that can generate a Spring Boot project structure for you. It doesn't generate any application code, but it will give you a basic project structure and either a Maven or a Gradle build specification to build your code with. And can be accessed through URL https://start.spring.io/



Project
Gradle - Groovy | Gradle - Kotlin | [✓ Maven]

Language
[✓ Java] | Kotlin | Groovy

Spring Boot
3.0.3 (SNAPSHOT) | [✓ 3.0.2] | 2.7.9 (SNAPSHOT) | 2.7.8

Project Metadata

Group	com.techies
Artifact	vender-service
Name	vender-service
Description	Demo project for vender service
Package name	com.techies.vender-service
Packaging	[✓ Jar] War
Java	[✓ 19] 17 11 8

Dependencies

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

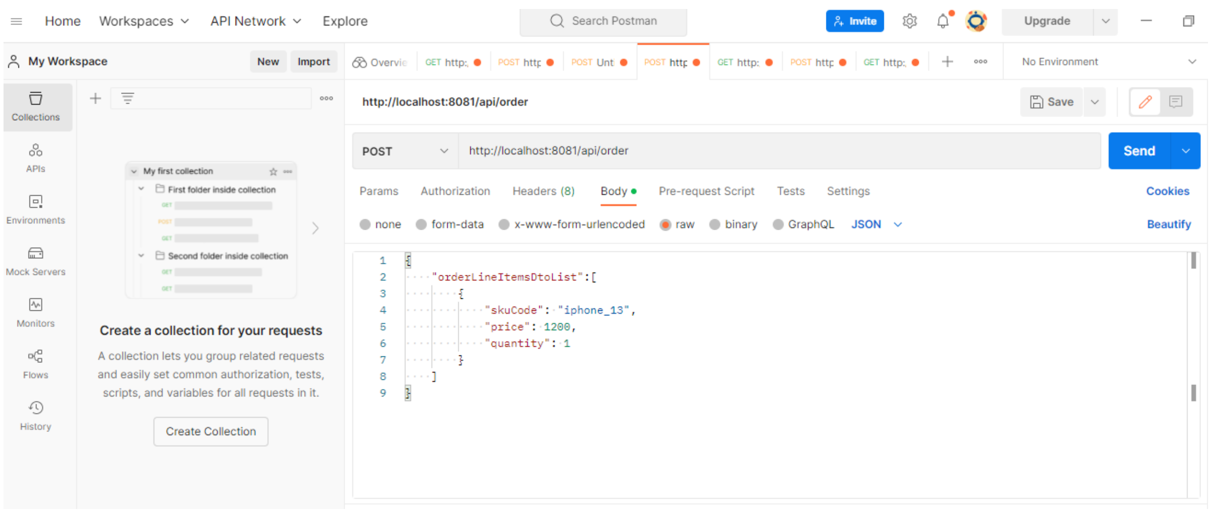
Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

MySQL Driver SQL
MySQL JDBC driver.

Figure 1 showing spring initializer interface

➢ **Postman**

Postman is an application used for API testing. It is an HTTP client that tests HTTP requests, utilizing a graphical user interface, through which we obtain different types of responses that need to be subsequently validated.



http://localhost:8081/api/order

POST http://localhost:8081/api/order

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
    "orderLineItemsDtoList":[
        {
            "skuCode": "iphone_13",
            "price": 1200,
            "quantity": 1
        }
    ]
}
```

Figure 2 showing postman interface

2.2 Frontend Design

CHAPTER THREE

MICROSERVICES

3.1 Microservice Lists

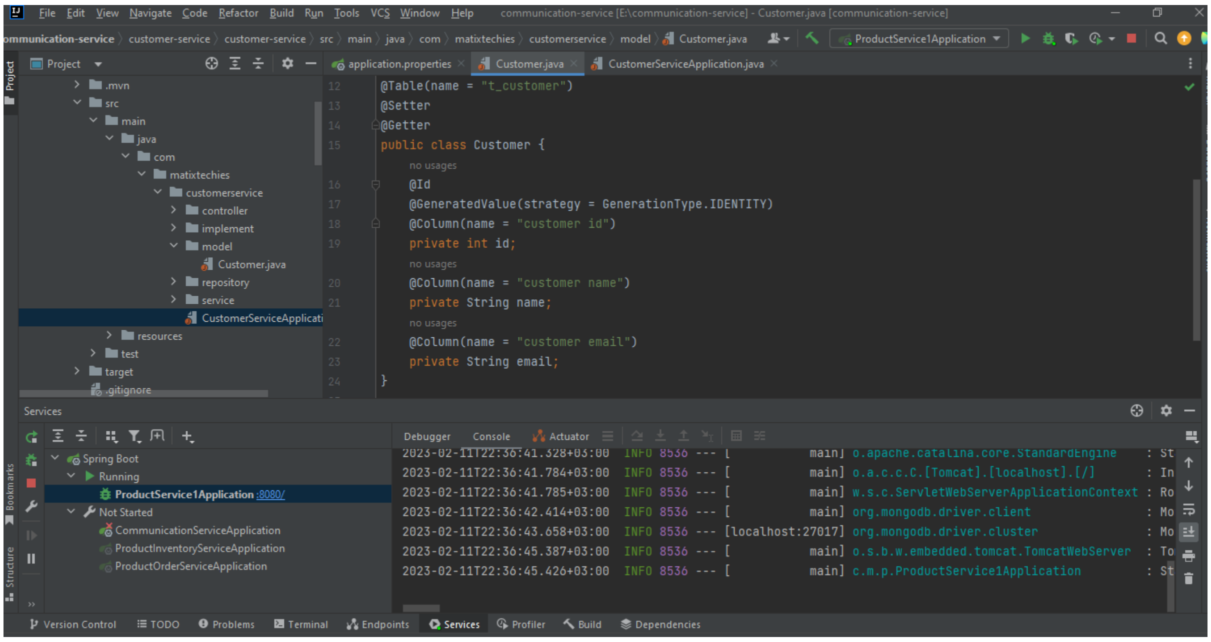
This project includes five microservices:

1. Customer Microservice
2. Order Microservice
3. Inventory Microservice
4. Vendor Microservice
5. Product Microservice

3.2 Microservice Descriptions

Customer Microservice

The customer microservice deals with all the information related to customers, such as customer names, contact information, and order history. It uses a relational database to store the data and exposes REST APIs to access the data. The customer microservice also implements basic CRUD operations, allowing administrators to add, update, and delete customer information.



```
@Table(name = "t_customer")
@Setter
@Getter
public class Customer {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "customer id")
    private int id;
    no usages
    @Column(name = "customer name")
    private String name;
    no usages
    @Column(name = "customer email")
    private String email;
}
```

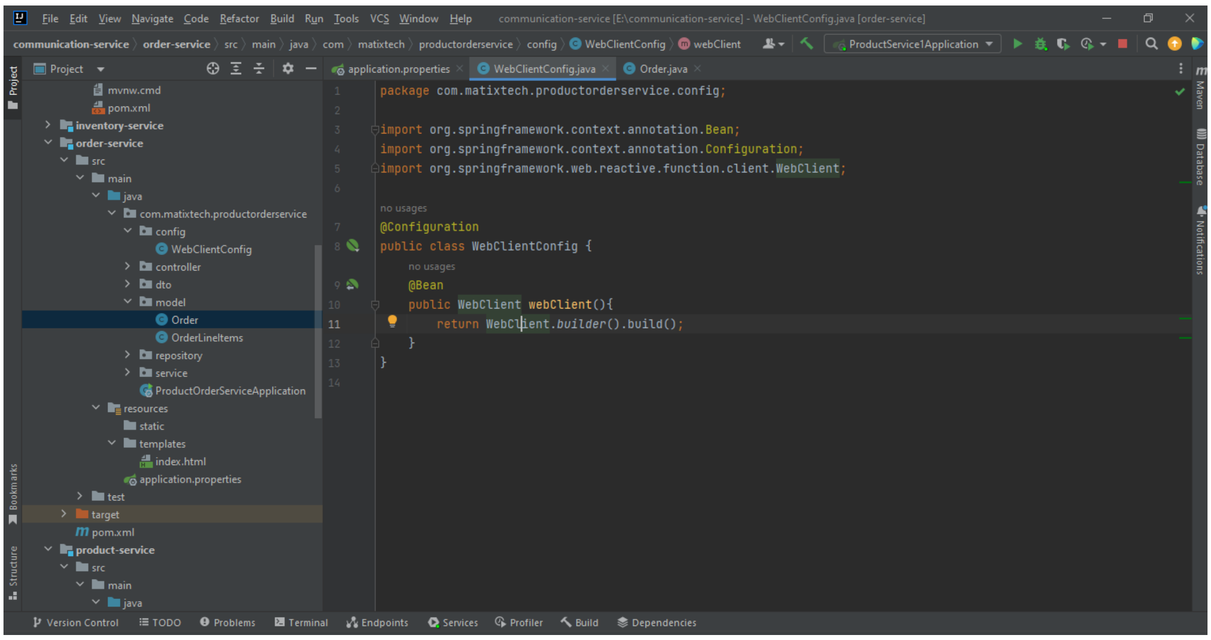
```
2023-02-11T22:36:41.328+03:00  INFO 8536 --- [           main] o.apache.catalina.core.StandardEngine    : St
2023-02-11T22:36:41.784+03:00  INFO 8536 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : In
2023-02-11T22:36:41.785+03:00  INFO 8536 --- [           main] w.s.c.ServletWebServerApplicationContext : Ro
2023-02-11T22:36:42.414+03:00  INFO 8536 --- [           main] org.mongodb.driver.client                : Mo
2023-02-11T22:36:43.658+03:00  INFO 8536 --- [localhost:27017] org.mongodb.driver.cluster               : Mo
2023-02-11T22:36:45.387+03:00  INFO 8536 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : To
2023-02-11T22:36:45.426+03:00  INFO 8536 --- [           main] c.m.p.ProductService1Application         : St
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
communication-service [E:\communication-service] - Customer.java [communication-service]
ommunication-service 〉 customer-service 〉 customer-service 〉 src 〉 main 〉 java 〉 com 〉 matixtechies 〉 customerservice 〉 model 〉 Customer.java
ProductService1Application
Project
.mvn
src
main
java
com
matixtechies
customerservice
controller
implement
model
Customer.java
repository
service
CustomerServiceApplicati
resources
test
target
.gitignore
application.properties
Customer.java
CustomerServiceApplication.java
Services
Spring Boot
Running
ProductService1Application :8080/
Not Started
CommunicationServiceApplication
ProductInventoryServiceApplication
ProductOrderServiceApplication
Debugger
Console
Actuator
Version Control
TODO
Problems
Terminal
Endpoints
Services
Profiler
Build
Dependencies

Figure 3 showing customer service

Order Microservice

The order microservice is necessary for managing all the information related to orders, such as order details, order status, and payment information. It uses a NoSQL database to store the data and exposes REST APIs to access the data. The order microservice also implements basic CRUD operations, allowing administrators to add, update, and delete order information.

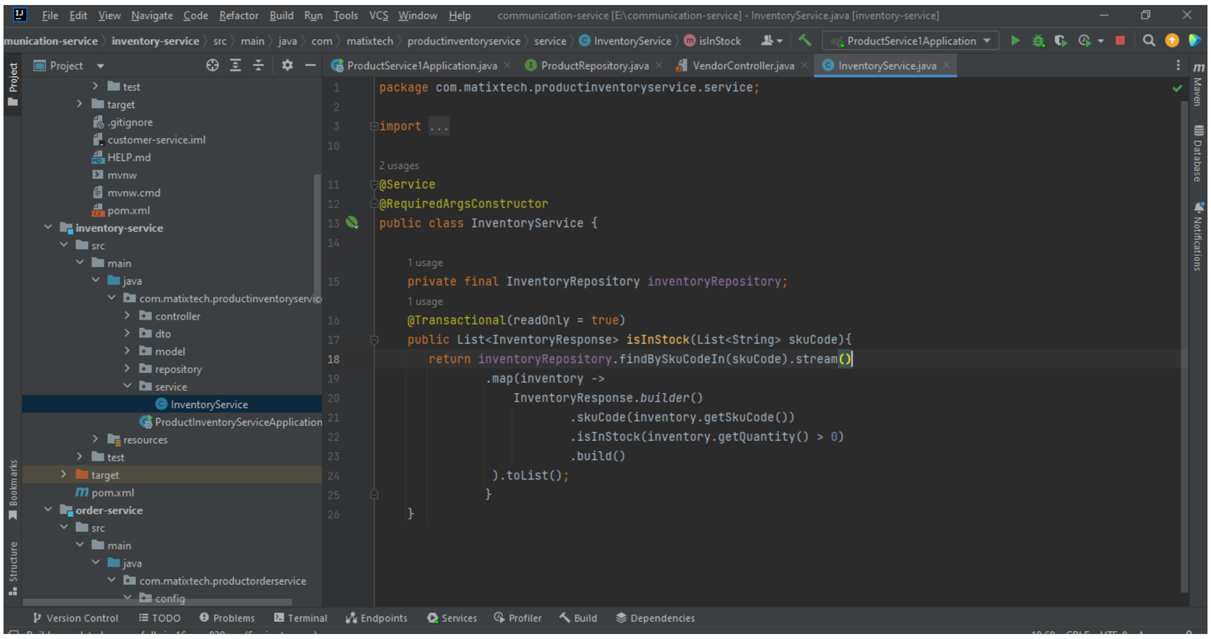


```java
package com.matixtech.productorderservice.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.reactive.function.client.WebClient;

@Configuration
public class WebClientConfig {
    @Bean
    public WebClient webClient(){
        return WebClient.builder().build();
    }
}
```

Figure 4 showing order service

Inventory Microservice

The inventory microservice is responsible for managing the availability of products in the inventory. It communicates with the product microservice to get product information and the order microservice to get order information. Based on this information, the inventory microservice updates the availability of products in real-time.

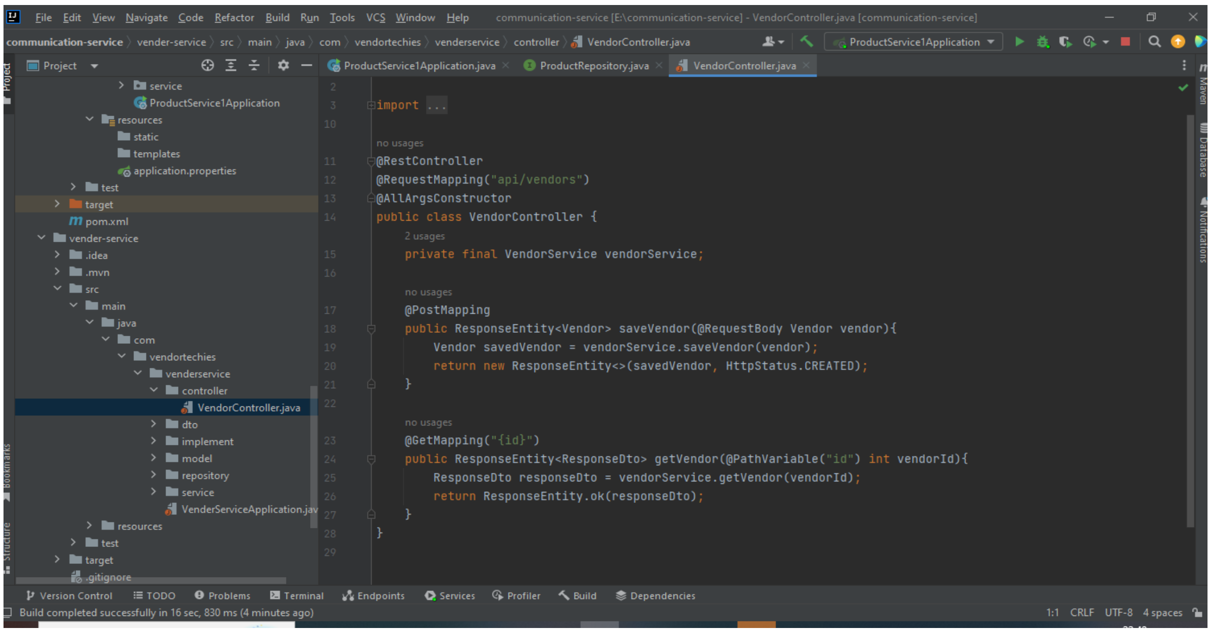


```java
package com.matixtech.productinventoryservice.service;

import ...

@Service
@RequiredArgsConstructor
public class InventoryService {

    private final InventoryRepository inventoryRepository;
    @Transactional(readOnly = true)
    public List<InventoryResponse> isInStock(List<String> skuCode){
       return inventoryRepository.findBySkuCodeIn(skuCode).stream()
               .map(inventory ->
                   InventoryResponse.builder()
                           .skuCode(inventory.getSkuCode())
                           .isInStock(inventory.getQuantity() > 0)
                           .build()
                ).toList();
               }
    }
```

Figure 5 showing inventory service

Vendor Microservice

This service manage the information related to vendors, such as vendor name, contact information, and product offerings. It uses a relational database to store the data and exposes REST APIs to access the data. The vendor microservice also implements basic CRUD operations, allowing administrators to add, update, and delete vendor information.



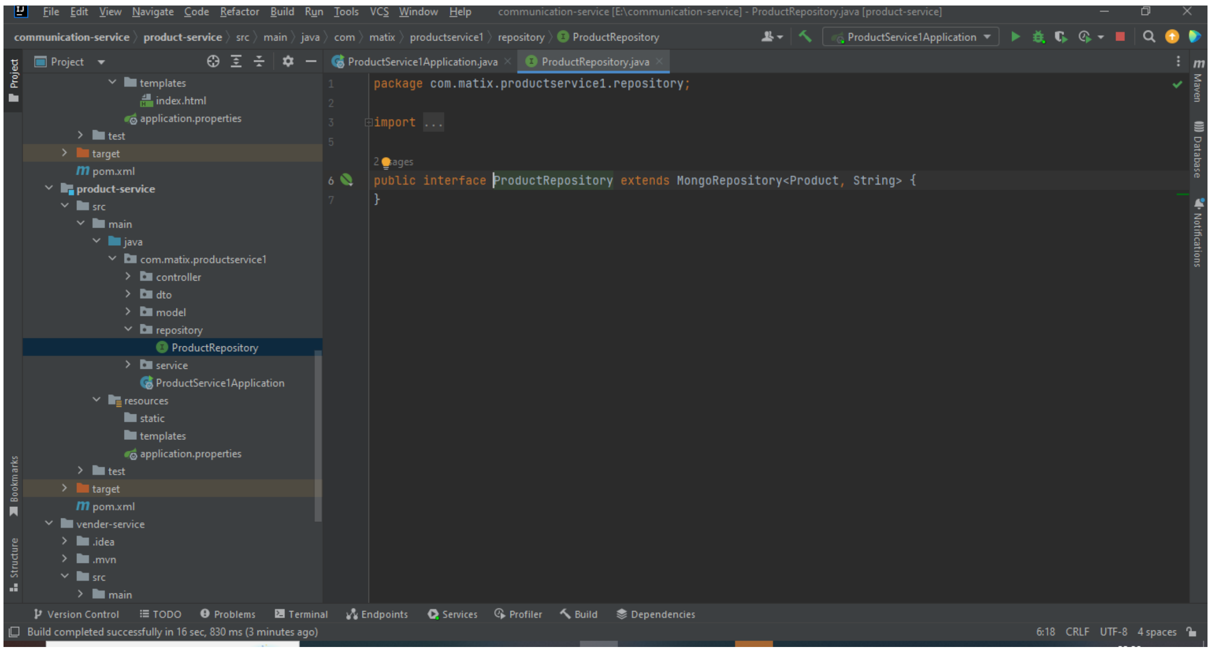
```java
import ...

@RestController
@RequestMapping("api/vendors")
@AllArgsConstructor
public class VendorController {
    private final VendorService vendorService;

    @PostMapping
    public ResponseEntity<Vendor> saveVendor(@RequestBody Vendor vendor){
        Vendor savedVendor = vendorService.saveVendor(vendor);
        return new ResponseEntity<>(savedVendor, HttpStatus.CREATED);
    }

    @GetMapping("{id}")
    public ResponseEntity<ResponseDto> getVendor(@PathVariable("id") int vendorId){
        ResponseDto responseDto = vendorService.getVendor(vendorId);
        return ResponseEntity.ok(responseDto);
    }
}
```

Build completed successfully in 16 sec, 830 ms (4 minutes ago)

Figure 6 showing vendor service

Product Microservice

The product microservice is used to manage all the information related to products, such as product name, description, price, and availability. It uses a NoSQL database to store the data and exposes REST APIs to access the data. The product microservice also implements basic CRUD operations, allowing administrators to add, update, and delete product information.



```
package com.matix.productservice1.repository;

import ...

2 usages
public interface ProductRepository extends MongoRepository<Product, String> {
}
```

```
Build completed successfully in 16 sec, 830 ms (3 minutes ago)
```

Figure 7 showing product service

Communication of microservices

Microservices communicate with each other using APIs, for example, RESTful APIs is used to allow communication using the HTTP protocol. In a microservices architecture, each service exposes a set of RESTful APIs that can be used by other services to access its data and functionality. For example, The Order Microservice will communicate with the Customer Microservice to retrieve customer information and with the Inventory Microservice to check the availability of products. The Inventory Microservice will communicate with the Product Microservice to retrieve product information and with the Order Microservice to update the availability of products. The Vendor Microservice will communicate with the Product Microservice to retrieve information about the products offered by the vendor.

These relationships ensure that all the microservices have the necessary information to perform their respective tasks and that the information is consistent and up-to-date across all the microservices

Microservice	**Relation to other microservices**
Vendor	-communicate with customer
Product	communicates with Inventory
Customer	-Responds to vendor
Order	-Communicates with Inventory
Inventory	-Responds to Order

The table above shows how our microservices communicate which Order Microservice communicates with the Customer Microservice to retrieve customer information, and with the Inventory Microservice to check the availability of products. The Inventory Microservice communicates with the Product Microservice to retrieve product information and with the Order Microservice to update the availability of products.

ONLINE PRODUCT ORDERING APPLICATION RELATION LIFE CYCLE

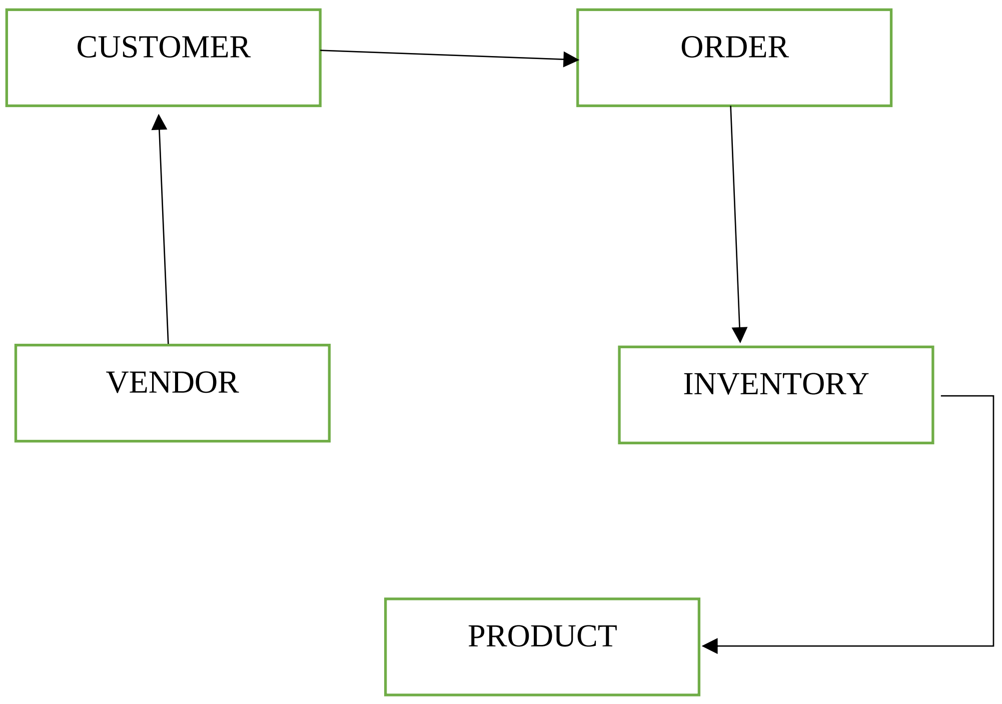


Figure 8 showing microservices relations

Inter process communication in microservices

In a monolithic application, components invoke one another via language-level method or function calls. In contrast, a microservices-based application is a distributed system running on multiple machines. Each service instance is typically a process. Consequently, as the following diagram shows, services must interact using an inter-process communication (IPC) mechanism

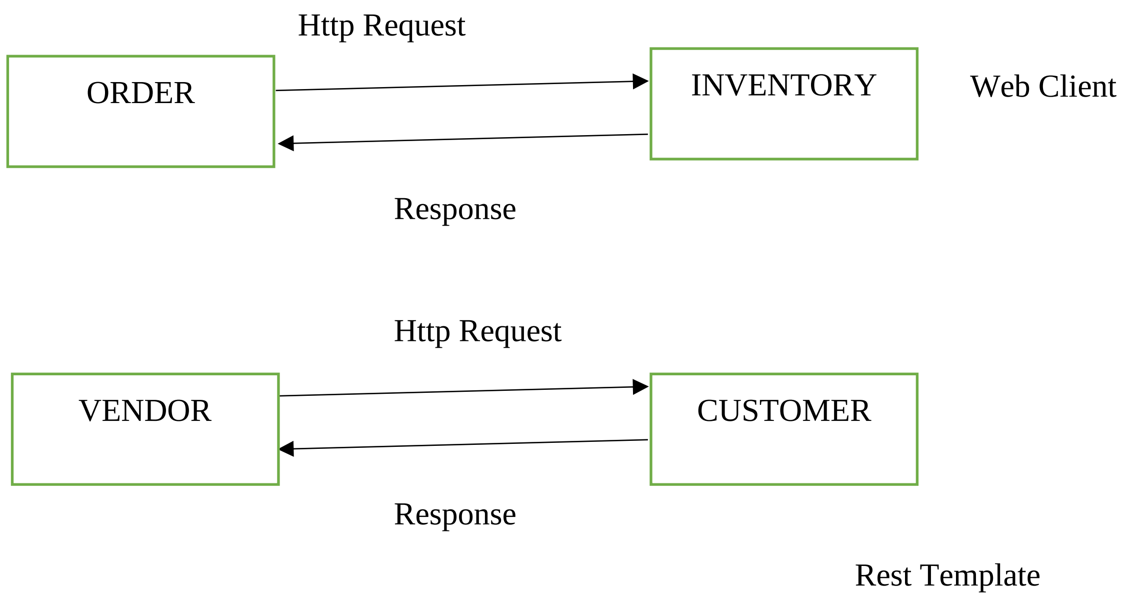


Figure 9 showing microservices IPC

CHAPTER FOUR

CONCLUSION

In conclusion, the implementation of a microservices-based application for managing vendors, products, customers, orders, and inventory can bring several benefits, such as increased scalability, improved reliability, and faster time to market. By breaking down the application into smaller, independent services, each with its own database, the system can handle increased load by adding more instances of a specific service. This also allows for easier maintenance and evolution of the system, as changes can be made to individual services without affecting the rest of the application.

The use of Spring Boot, a popular Java framework for building microservices, made it easier to implement the various services and their RESTful APIs. The use of a service registry and an API gateway allowed for efficient communication and management of the services, making it easier for clients to access the various functionalities of the system.

REFERENCES

Larrucea, X., Santamaria, I., Colomo-Palacios, R., & Ebert, C. (2018). Microservices. *IEEE Software*, *35*(3), 96-100.

Shafabakhsh, B., Lagerström, R., & Hacks, S. (2020). Evaluating the Impact of Inter Process Communication in Microservice Architectures. In *QuASoQ@ APSEC* (pp. 55-63).

https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture