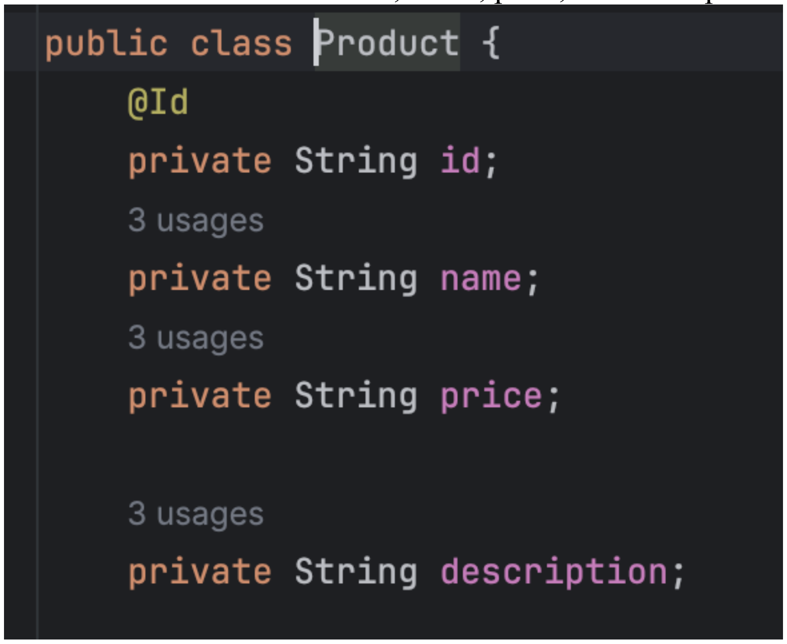
Group 2

UWIRAGIYE Aphrodis	M02258/2023
Morris Mulbah	M01928/2022
TUYISENGE CLENIE	M02317/2023
NDAGIJIMANA Eric	M02652/2023
Dushimimana Jean Honore	M02223/2023
ABAYIZERA YVES Bertrand	M02344/2023

Microservice Assignment report

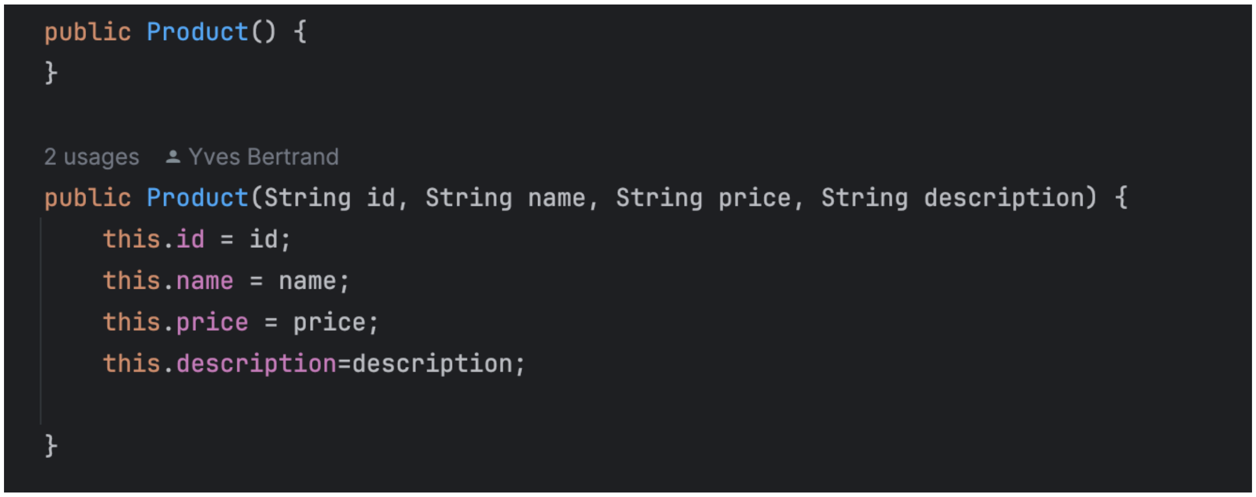
We have created two Microservices:

Product with attributes like ID, name, price, and description.



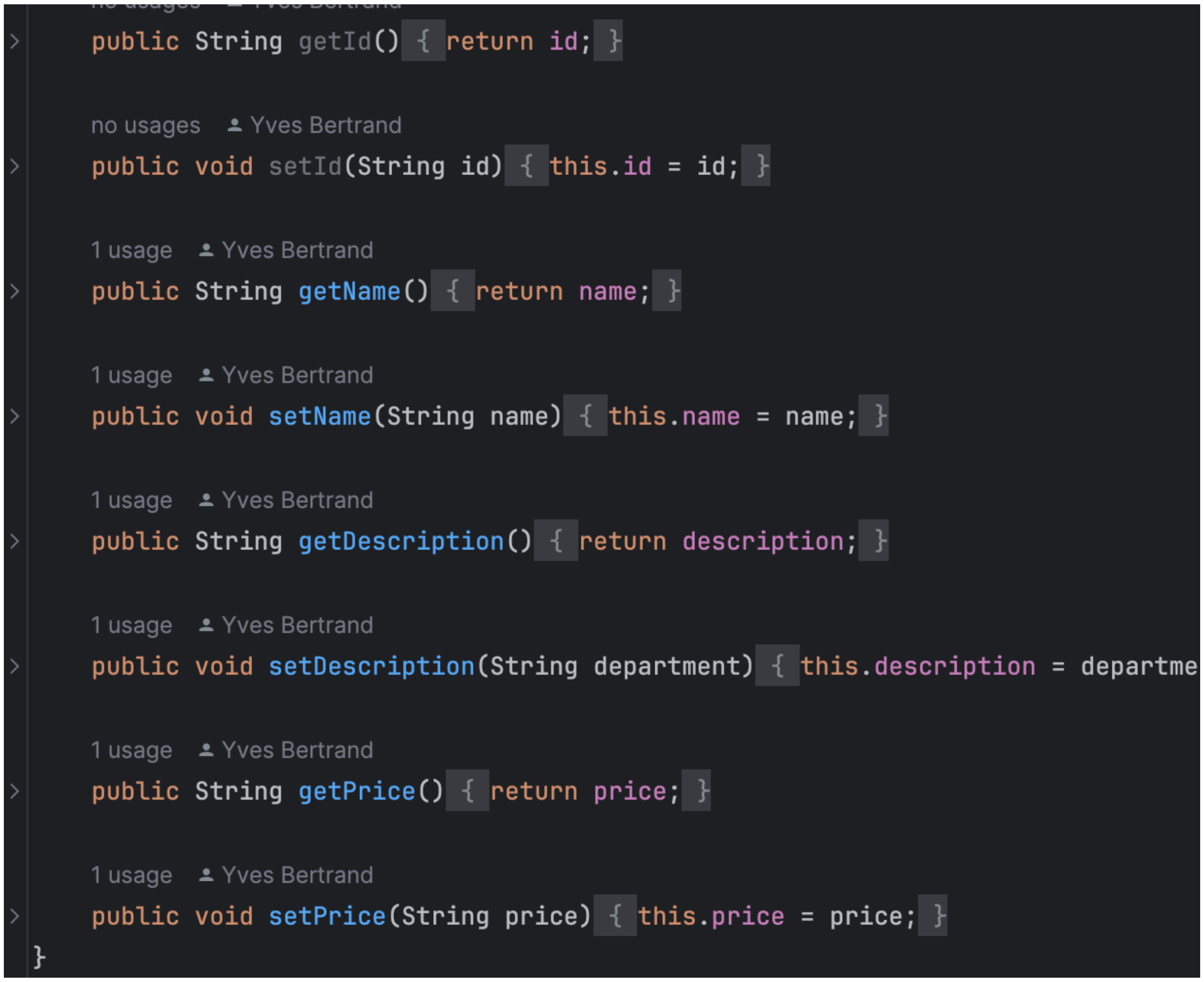
```java
public class Product {
    @Id
    private String id;
    3 usages
    private String name;
    3 usages
    private String price;

    3 usages
    private String description;
```

Then we have defined the constructors for the product



```java
public Product() {
}

2 usages   Yves Bertrand
public Product(String id, String name, String price, String description) {
    this.id = id;
    this.name = name;
    this.price = price;
    this.description=description;

}
```

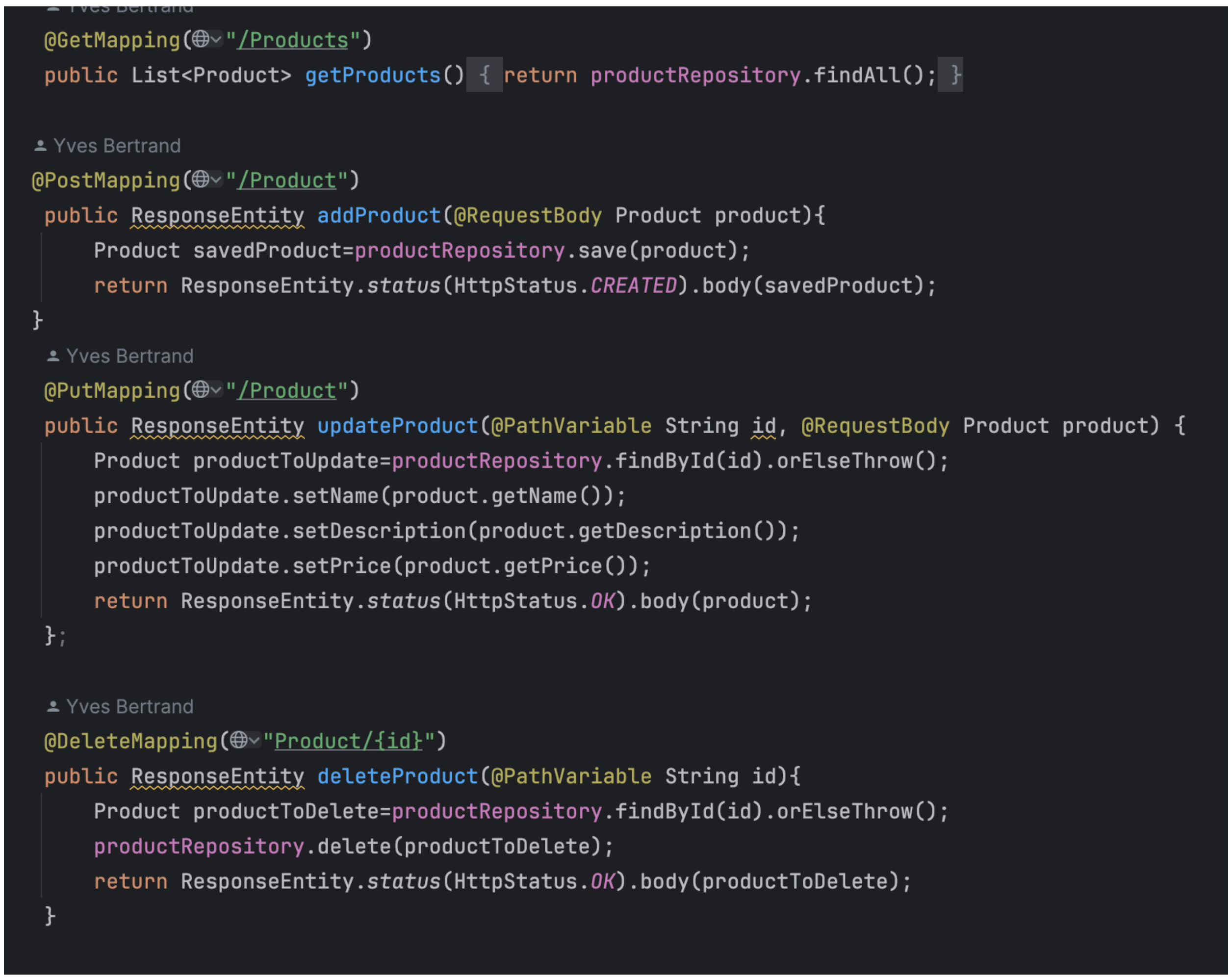
We also defined the setter and getters get data into and from the microservice.



```
    public String getId() { return id; }

    no usages   Yves Bertrand
    public void setId(String id) { this.id = id; }

    1 usage   Yves Bertrand
    public String getName() { return name; }

    1 usage   Yves Bertrand
    public void setName(String name) { this.name = name; }

    1 usage   Yves Bertrand
    public String getDescription() { return description; }

    1 usage   Yves Bertrand
    public void setDescription(String department) { this.description = departme

    1 usage   Yves Bertrand
    public String getPrice() { return price; }

    1 usage   Yves Bertrand
    public void setPrice(String price) { this.price = price; }
}
```

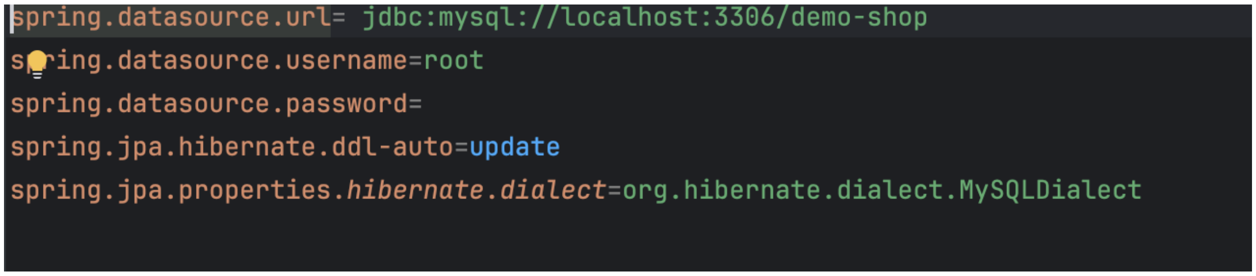
We proceeded with creating a controller class to be used to implement requests to our microservice.

We have defined our GET mapping, POST mapping, PUT mapping, DELETE mapping to view, add, update, and delete data from the microservice.



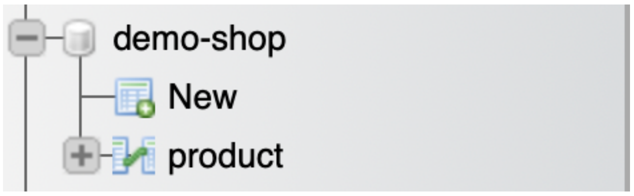
```
@GetMapping("/Products")
public List<Product> getProducts() { return productRepository.findAll(); }

Yves Bertrand
@PostMapping("/Product")
public ResponseEntity addProduct(@RequestBody Product product){
    Product savedProduct=productRepository.save(product);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedProduct);
}
Yves Bertrand
@PutMapping("/Product")
public ResponseEntity updateProduct(@PathVariable String id, @RequestBody Product product) {
    Product productToUpdate=productRepository.findById(id).orElseThrow();
    productToUpdate.setName(product.getName());
    productToUpdate.setDescription(product.getDescription());
    productToUpdate.setPrice(product.getPrice());
    return ResponseEntity.status(HttpStatus.OK).body(product);
};

Yves Bertrand
@DeleteMapping("Product/{id}")
public ResponseEntity deleteProduct(@PathVariable String id){
    Product productToDelete=productRepository.findById(id).orElseThrow();
    productRepository.delete(productToDelete);
    return ResponseEntity.status(HttpStatus.OK).body(productToDelete);
}
```

We also defined our application properties with our SQL information to be connected to our DB.

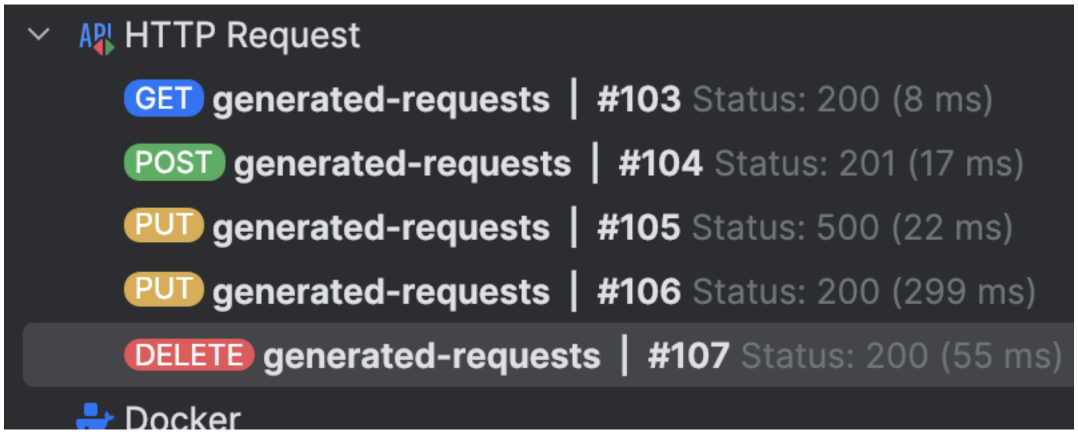


```
spring.datasource.url= jdbc:mysql://localhost:3306/demo-shop
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

We also created Manually our DB in our local host under name demo-shop for the product.

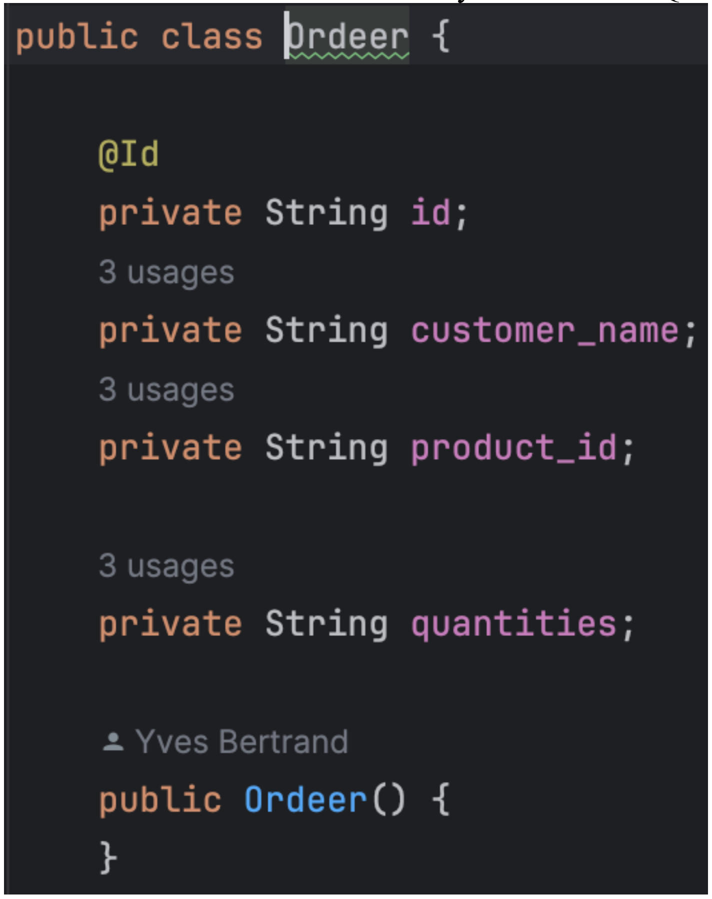


We were able to do our requests.



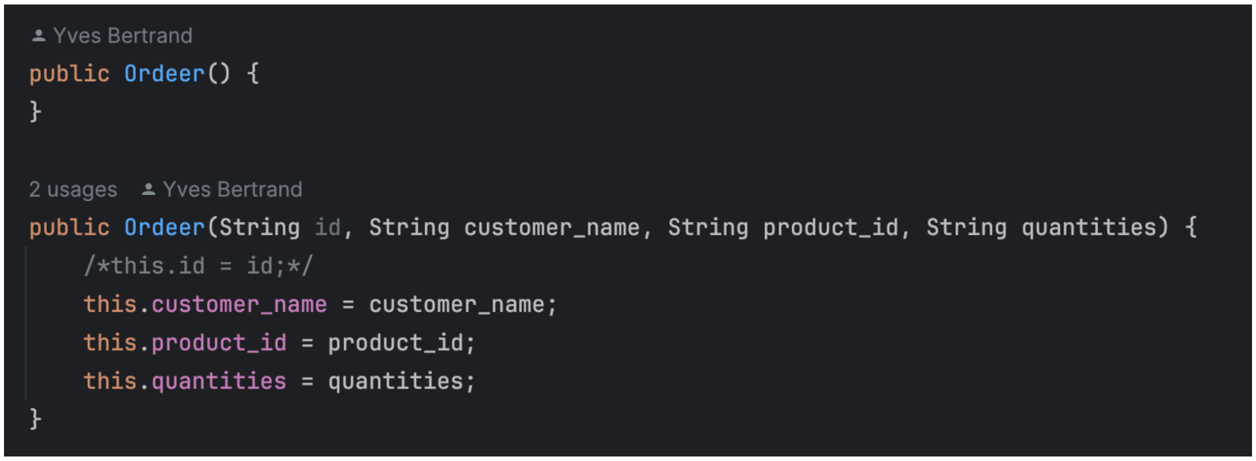
```
HTTP Request
    GET generated-requests | #103 Status: 200 (8 ms)
    POST generated-requests | #104 Status: 201 (17 ms)
    PUT generated-requests | #105 Status: 500 (22 ms)
    PUT generated-requests | #106 Status: 200 (299 ms)
    DELETE generated-requests | #107 Status: 200 (55 ms)
Docker
```

Order with attributes like ID, customer_name, product_id, and quantities.
We named it ordeer as order is a key word in MYSQL and it was creating confusion.



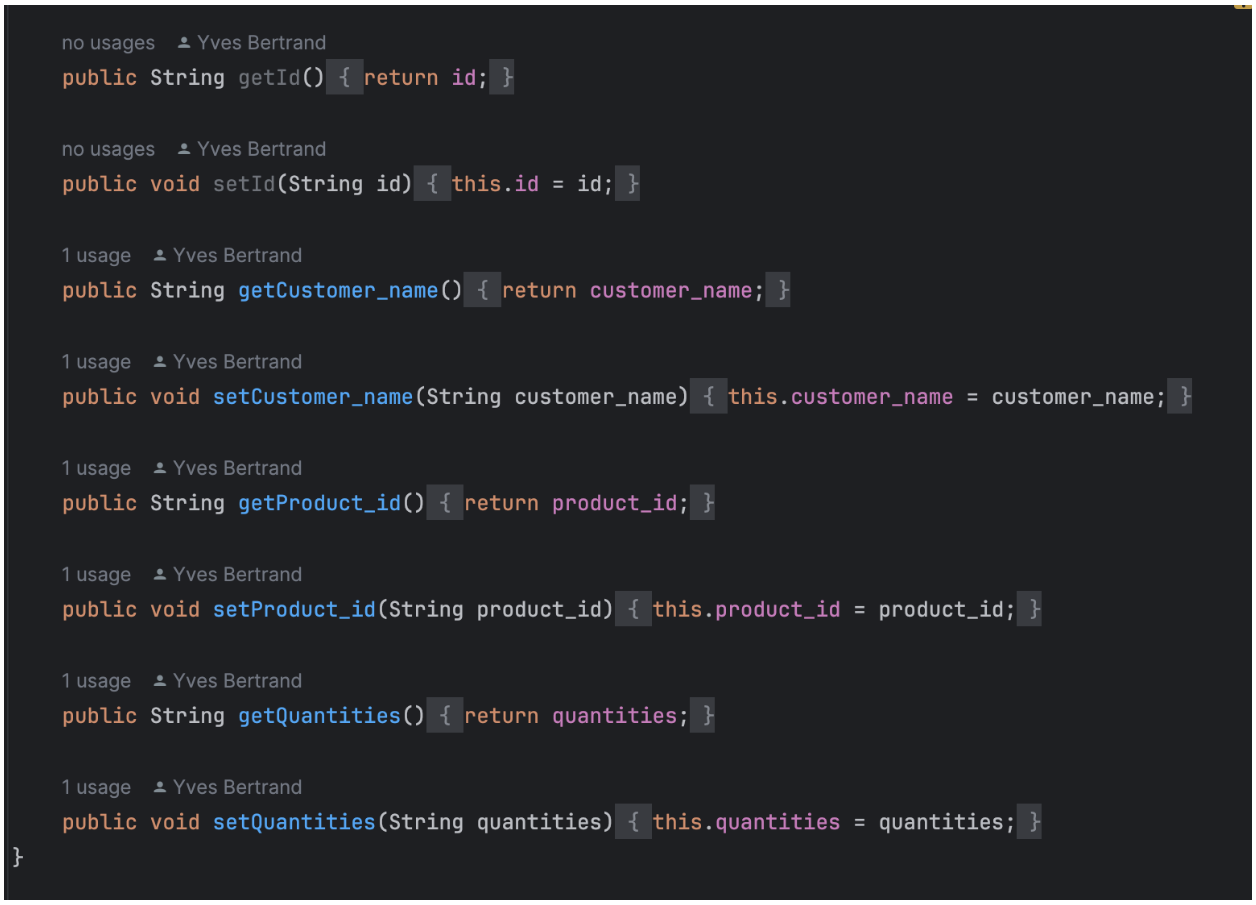
```java
public class Ordeer {

    @Id
    private String id;
    3 usages
    private String customer_name;
    3 usages
    private String product_id;

    3 usages
    private String quantities;

    Yves Bertrand
    public Ordeer() {
    }
```

Then we have defined the constructors for the order



```
Yves Bertrand
public Ordeer() {
}

2 usages   Yves Bertrand
public Ordeer(String id, String customer_name, String product_id, String quantities) {
    /*this.id = id;*/
    this.customer_name = customer_name;
    this.product_id = product_id;
    this.quantities = quantities;
}
```

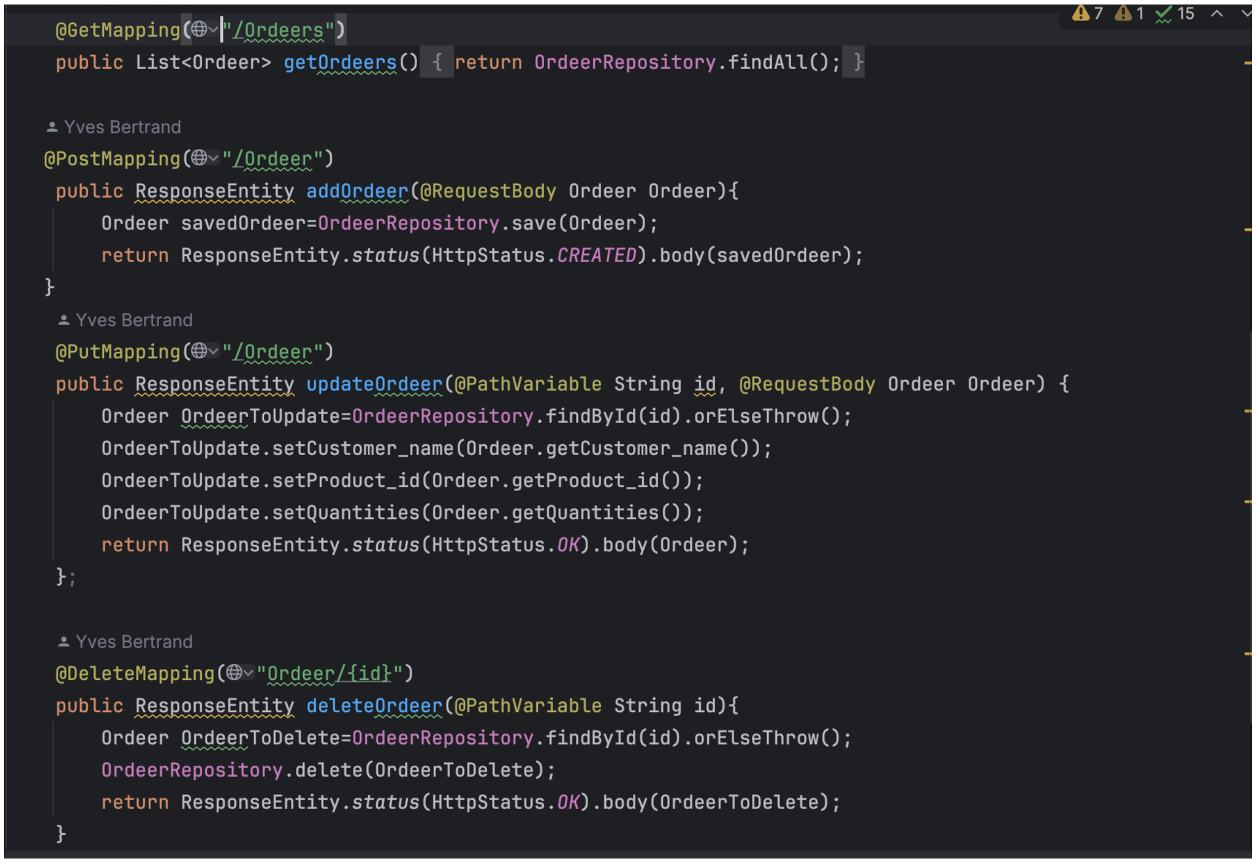
We also defined the setter and getters get data into and from the microservice.



```
    no usages   Yves Bertrand
    public String getId() { return id; }

    no usages   Yves Bertrand
    public void setId(String id) { this.id = id; }

    1 usage   Yves Bertrand
    public String getCustomer_name() { return customer_name; }

    1 usage   Yves Bertrand
    public void setCustomer_name(String customer_name) { this.customer_name = customer_name; }

    1 usage   Yves Bertrand
    public String getProduct_id() { return product_id; }

    1 usage   Yves Bertrand
    public void setProduct_id(String product_id) { this.product_id = product_id; }

    1 usage   Yves Bertrand
    public String getQuantities() { return quantities; }

    1 usage   Yves Bertrand
    public void setQuantities(String quantities) { this.quantities = quantities; }
}
```

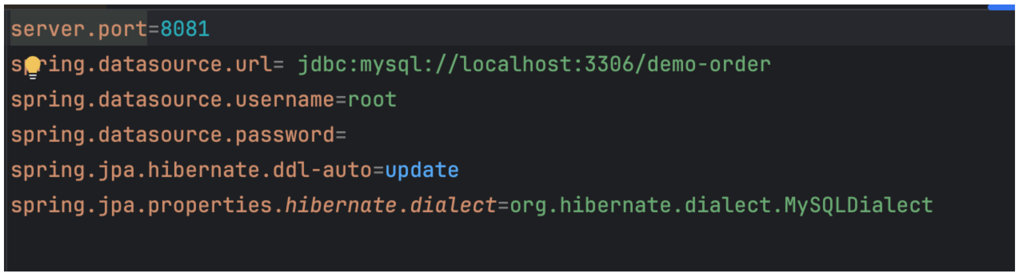
We proceeded with creating a controller class to be used to implement requests to our microservice.

We have defined our GET mapping, POST mapping, PUT mapping, DELETE mapping to view, add, update, and delete data from the microservice.



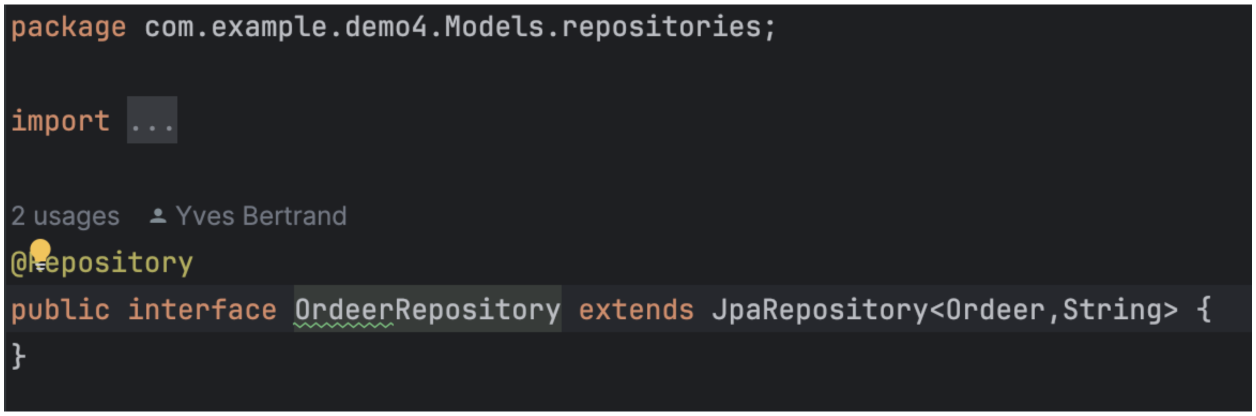
```
@GetMapping("/Ordeers")
public List<Ordeer> getOrdeers() { return OrdeerRepository.findAll(); }

Yves Bertrand
@PostMapping("/Ordeer")
public ResponseEntity addOrdeer(@RequestBody Ordeer Ordeer){
    Ordeer savedOrdeer=OrdeerRepository.save(Ordeer);
    return ResponseEntity.status(HttpStatus.CREATED).body(savedOrdeer);
}
Yves Bertrand
@PutMapping("/Ordeer")
public ResponseEntity updateOrdeer(@PathVariable String id, @RequestBody Ordeer Ordeer) {
    Ordeer OrdeerToUpdate=OrdeerRepository.findById(id).orElseThrow();
    OrdeerToUpdate.setCustomer_name(Ordeer.getCustomer_name());
    OrdeerToUpdate.setProduct_id(Ordeer.getProduct_id());
    OrdeerToUpdate.setQuantities(Ordeer.getQuantities());
    return ResponseEntity.status(HttpStatus.OK).body(Ordeer);
};

Yves Bertrand
@DeleteMapping("Ordeer/{id}")
public ResponseEntity deleteOrdeer(@PathVariable String id){
    Ordeer OrdeerToDelete=OrdeerRepository.findById(id).orElseThrow();
    OrdeerRepository.delete(OrdeerToDelete);
    return ResponseEntity.status(HttpStatus.OK).body(OrdeerToDelete);
}
```

We also defined our application properties with our SQL information to be connected to our DB.



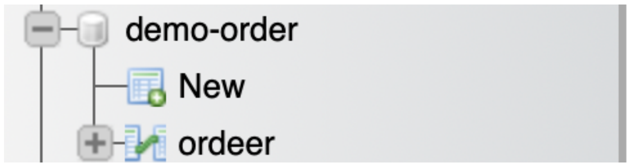
```
server.port=8081
spring.datasource.url= jdbc:mysql://localhost:3306/demo-order
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

We also created a repository to create our table in the database

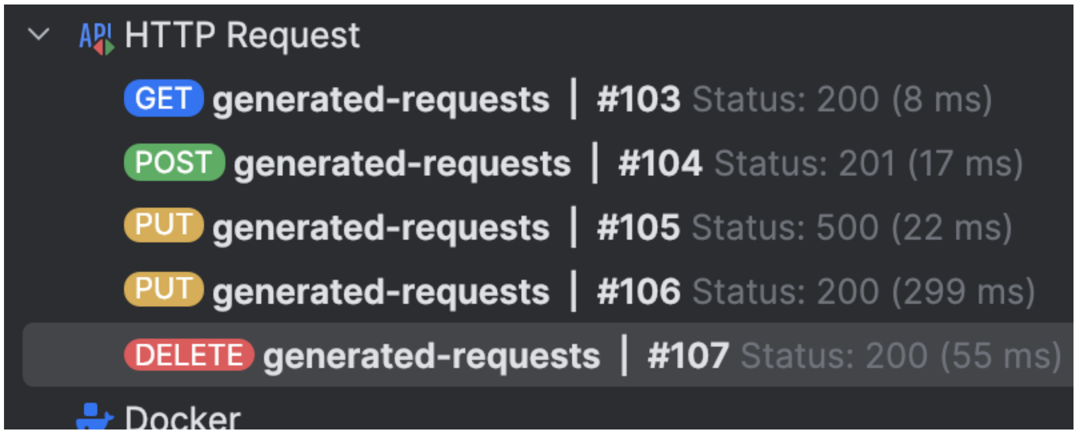


```
package com.example.demo4.Models.repositories;

import ...

2 usages   Yves Bertrand
@Repository
public interface OrdeerRepository extends JpaRepository<Ordeer,String> {
}
```

We also created Manually our DB in our local host under name demo-order for the product.

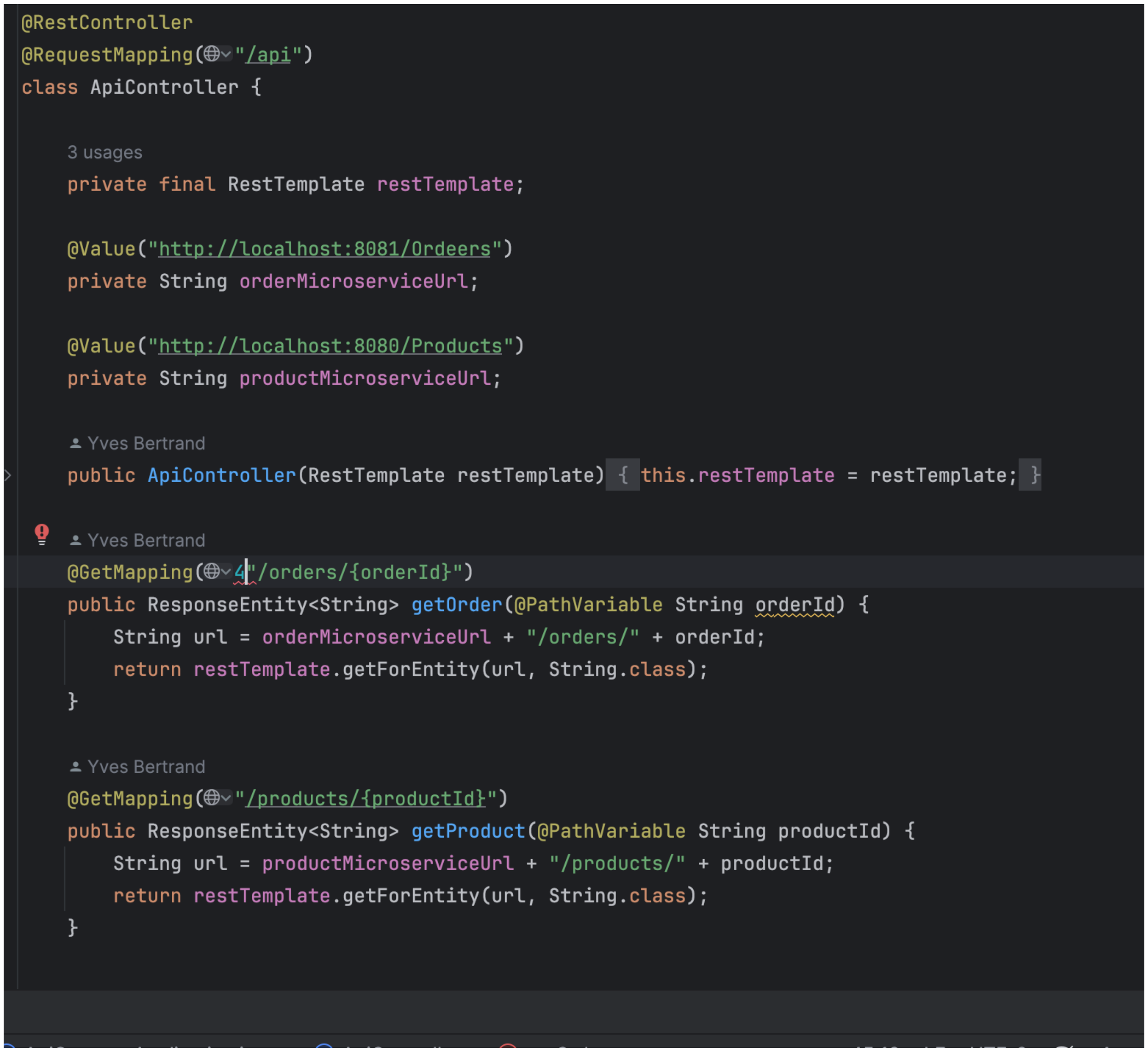


We were able to do our requests.



```
HTTP Request
    GET generated-requests | #103 Status: 200 (8 ms)
    POST generated-requests | #104 Status: 201 (17 ms)
    PUT generated-requests | #105 Status: 500 (22 ms)
    PUT generated-requests | #106 Status: 200 (299 ms)
    DELETE generated-requests | #107 Status: 200 (55 ms)
Docker
```

To connect both microservices, we have created an APIGateway



```
@RestController
@RequestMapping("/api")
class ApiController {

    3 usages
    private final RestTemplate restTemplate;

    @Value("http://localhost:8081/Ordeers")
    private String orderMicroserviceUrl;

    @Value("http://localhost:8080/Products")
    private String productMicroserviceUrl;

    Yves Bertrand
    public ApiController(RestTemplate restTemplate) { this.restTemplate = restTemplate; }

    Yves Bertrand
    @GetMapping(4"/orders/{orderId}")
    public ResponseEntity<String> getOrder(@PathVariable String orderId) {
        String url = orderMicroserviceUrl + "/orders/" + orderId;
        return restTemplate.getForEntity(url, String.class);
    }

    Yves Bertrand
    @GetMapping("/products/{productId}")
    public ResponseEntity<String> getProduct(@PathVariable String productId) {
        String url = productMicroserviceUrl + "/products/" + productId;
        return restTemplate.getForEntity(url, String.class);
    }
```

Each microservice works on its own port to have them work simultaneously.

Thank you…..