Designing, Evaluating, and Documenting Software Architectures

Mojtaba Shahin

Week 8: Lectorial – Part 2

Content

- Part 1
  - Attribute-Driven Design
  - The Steps of Attribute-Driven Design
  - A Case Study
- Part 2
  - Spring Boot and Microservices

Acknowledgements

- Most of the **texts** and **images** in the slides come from the following sources:
  - What Is a REST API? Examples, Uses, and Challenges (https://blog.postman.com/rest-api-examples/)
  - https://www.geeksforgeeks.org/advance-java/spring-boot/
  - https://azure.microsoft.com/en-au/resources/cloud-computing-dictionary/what-is-java-spring-boot
  - Build & Deploy a Production-Ready Patient Management System with Microservices: Java Spring Boot AWS https://www.youtube.com/watch?v=tseqdcFfTUY

Communication Styles

- Some of the interaction styles:
  - REST (Representational State Transfer)
  - RPC (Remote Procedure Call)
  - gRPC (Google Remote Procedure Call)
  - SOAP (Simple Object Access Protocol)
  - GraphQL
  - WebSocket

Frameworks and Tools for building microservices

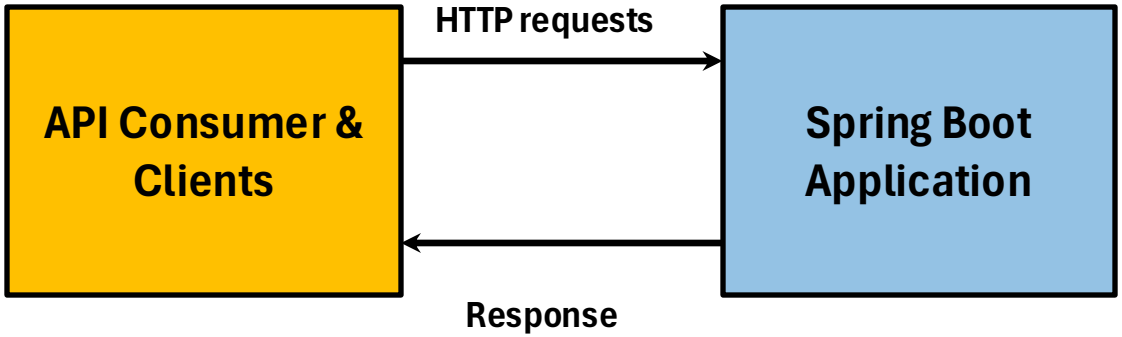
- Java
  - Spring Boot + Spring Cloud
- C#
  - ASP.NET Core
- Python
  - Flask

Spring Boot Framework*

- Spring Boot is a Java framework built on top of Spring that simplifies application development.
- Spring Boot comes with an embedded server, making applications production ready out of the box.
- It supports web apps, **REST APIs**, **microservices**, security and seamless cloud deployment.

*Source: https://www.geeksforgeeks.org/advance-java/spring-boot/

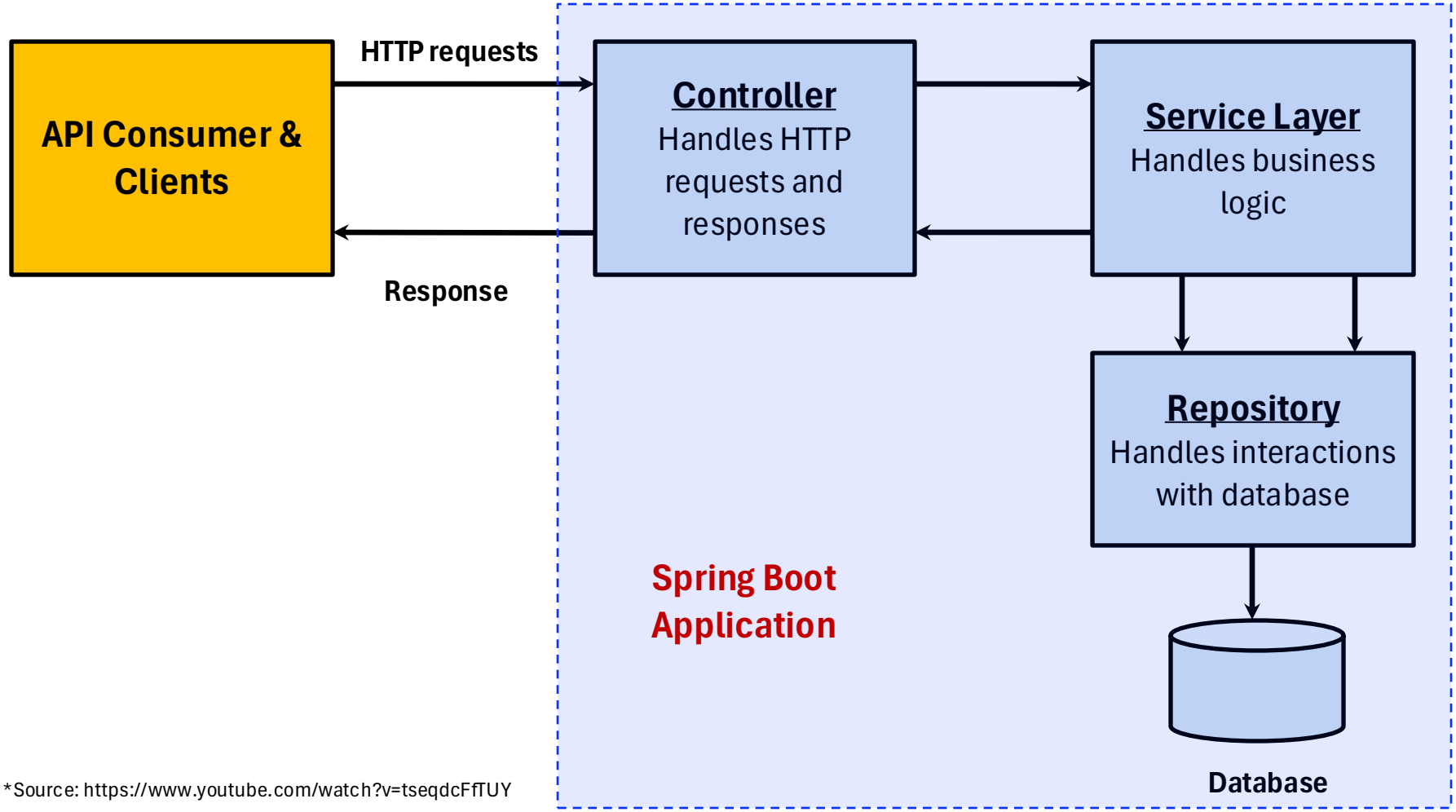
Spring Boot Architecture*



*Source: https://www.youtube.com/watch?v=tseqdcFfTUY

*Source: https://www.geeksforgeeks.org/springboot/spring-boot-architecture/

Spring Boot Architecture



*Source: https://www.youtube.com/watch?v=tseqdcFfTUY

*Source: https://www.geeksforgeeks.org/springboot/spring-boot-architecture/

REST

- ***REpresentational State Transfer (REST).***
- REST is a protocol for web services. It imposes six constraints on the interactions between elements:
  - ***Uniform interface***. All interactions use the same form (typically HTTP). Resources are specified via URIs (Uniform Resource Identifier).
  - ***Client-server***. The actors are clients and the resource providers are servers using the client-server pattern.
  - ***Stateless***. All client-server interactions are stateless.
  - ***Cacheable***. Caching is applied to resources when applicable.
  - ***Layered architecture***. The "server" can be broken into multiple elements, which may be deployed independently.
  - ***Code on demand (optional)***. It is possible for the server to provide code to the client to be executed. JavaScript is an example

REST

- **REST is resource-oriented**
  - To understand how REST APIs work, it is critical to understand **resources**.
  - A resource can be any information that could be named, such as a document or image, a collection of other resources, and more.
  - REST uses a resource identifier to recognise the specific resource involved in an interaction between components.

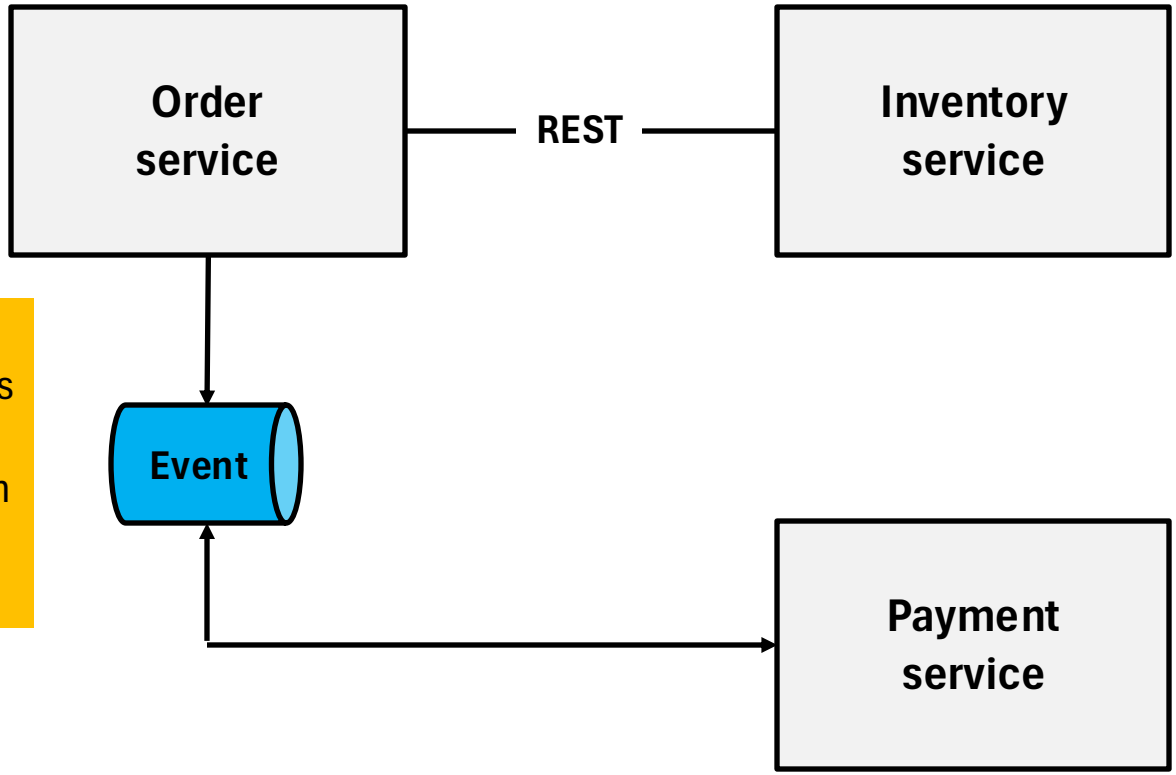
HTTP Command		CRUD Operation
POST	→	**CREATE a new resource**
GET	→	**READ data about an existing resource**
PUT	→	**UPDATE an existing resource**
DELETE	→	**DELETE an existing resource**

Expose a service's methods through REST API

- A service can make its methods available to client applications (frontends, other services, etc) via HTTP endpoints.
- Client applications can call those endpoints using standard HTTP methods (**GET**, **POST**, **PUT**, **DELETE**).
- Communication between a service and its clients usually follows the request–response model, with data typically exchanged in JSON format,

Example: e-Commerce App

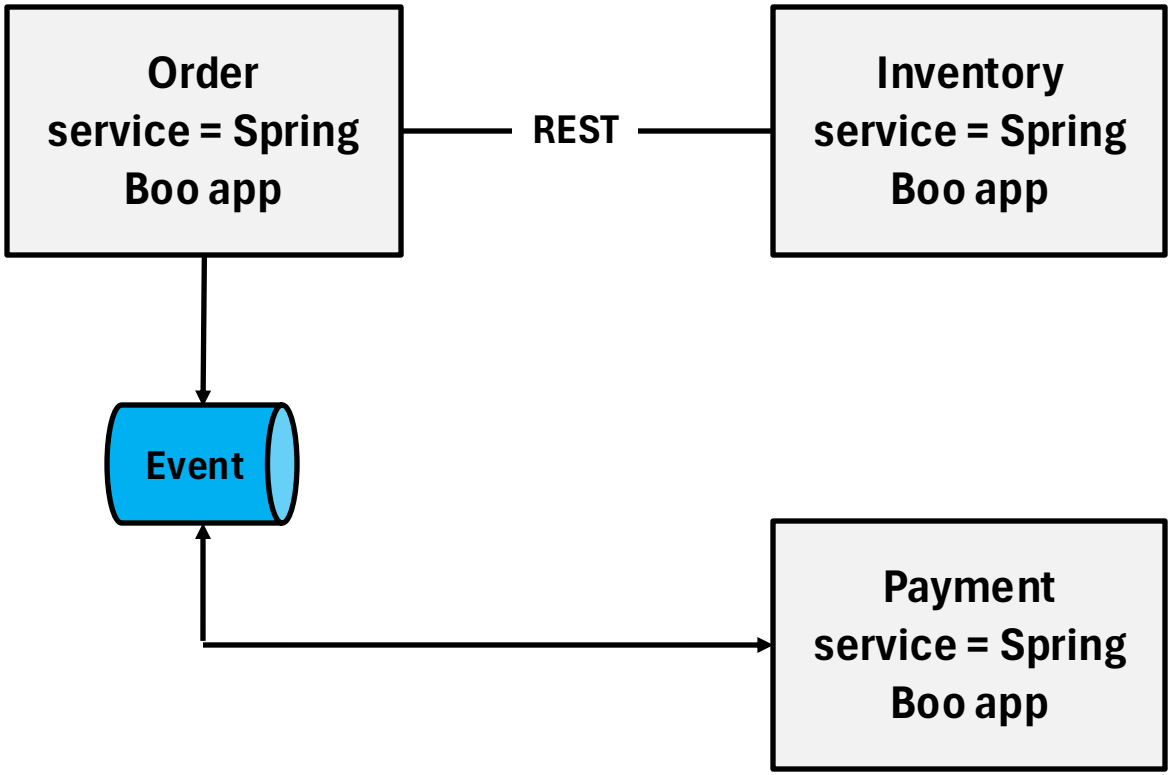
Inventory service creates, deletes, updates, etc., products. It also checks the quantity of each product and reduces or increases its quantity.



Order service creates, deletes, updates orders and enables users to search orders based on order ID and product/item names

Payment service does the payment process.

Example: e-Commerce App with Spring Boot



Expose the methods of Inventory service

Controller Layer and REST APIs

Product Entity

```
public class Product {

    private int productId;
    private String productName;
    private int productQuantity;
```

Expose the methods of Inventory service

@RestController
Exposes service methods as **HTTP endpoints**.
Handles requests, calls the service, and returns the response.

```
@RestController
@RequestMapping("/inventory")
public class InventoryServiceController {
```

@RequestMapping annotation is to map requests to controllers methods.
Variants of @**RequestMapping:**
@GetMapping
@PostMapping
@PutMapping
@DeleteMapping
@PatchMapping

Expose the methods of Inventory service

@GetMapping maps HTTP **GET** requests to the *getProductsInInventory* method. Since we haven't specified a path (@GetMapping("/something")), it inherits the **class-level @RequestMapping path**.

Example URL: /inventory

```
@GetMapping
public ResponseEntity <List <Product> > getProductsInInventory (){

    return ResponseEntity.ok(dummyInventoryServiceLayer.getAllProductsInInventory());
}
```

ResponseEntity allows you to control HTTP status codes, headers, and body.

ResponseEntity.ok(...) → sends a 200 OK with the **data**.

Expose the methods of Inventory service

Example URL: /inventory

```
@GetMapping
public ResponseEntity <List <Product> > getProductsInInventory (){

    return ResponseEntity.ok(dummyInventoryServiceLayer.getAllProductsInInventory());
}
```

Sample response

```
[
    {
        "productId": 1,
        "productName": "Product1",
        "productQuantity": 10
    },
    {
        "productId": 2,
        "productName": "Product2",
        "productQuantity": 8
    },
    {
        "productId": 3,
        "productName": "Product3",
        "productQuantity": 20
    },
```

Expose the methods of Inventory service

@PutMapping maps HTTP **PUT** requests to the *updateProduct* method.

Example URL: /inventory

```
@PutMapping
public ResponseEntity <Product> updateProduct (@RequestBody Product newProduct) {

    Product product = dummyInventoryServiceLayer.updateProduct(newProduct);
    if (product != null) {
        return ResponseEntity.ok(product);
    }
    else {
        return ResponseEntity.notFound().build();
    }
}
```

@RequestBody Product newProduct
Tells Spring to **deserialize the incoming JSON request body** into a **Product** object.

If the product does not exist → return **404 Not Found**.

Expose the methods of Inventory service

Example URL: /inventory

```
@PutMapping
public ResponseEntity <Product> updateProduct (@RequestBody Product newProduct) {

    Product product = dummyInventoryServiceLayer.updateProduct(newProduct);
    if (product != null) {
        return ResponseEntity.ok(product);
    }
    else {
        return ResponseEntity.notFound().build();
    }
}
```

Sample request

```
{
    "productId": 7,
    "productName": "Product7",
    "productQuantity": 182
}
```

Expose the methods of Inventory service

@DeleteMapping maps **DELETE requests** to the *deleteProduct* method.
{id} is a **path variable** — a dynamic segment of the URL.

Example URL: /inventory/id/1

```
@DeleteMapping("/id/{id}")
public ResponseEntity <Void> deleteProduct (@PathVariable int id) {

    if (dummyInventoryServiceLayer.deleteProduct(id)) {
        return ResponseEntity.noContent().build();
    }
    else
        return ResponseEntity.notFound().build();
}
```

@PathVariable int id
Binds the {**id**} from the URL to the method parameter id.
Spring automatically converts the path segment to an int.

Indicates the request was successful but **there's no response body**.

Expose the methods of Inventory service

@PostMapping("/check-quantities") maps **HTTP POST** requests to the *getQuantityOfProducts* method.

Example URL: /inventory/check-quantities

```
@PostMapping("/check-quantities")
public ResponseEntity <Map<String,Integer>> getQuantityOfProducts (@RequestBody List<String> productNames) {

    Map<String, Integer> result = new HashMap<>();

    for (String name : productNames) {
        result.put(name, dummyInventoryServiceLayer.getQuantityOfProduct(name));
    }

    return ResponseEntity.ok(result);
}
```

@RequestBody List<String> productNames
Expects a **JSON array of product names** in the request body.
Spring automatically deserializes it into **a List<String>.**

Expose the methods of Inventory service

Example URL: /inventory/check-quantities

```
@PostMapping("/check-quantities")
public ResponseEntity <Map<String,Integer>> getQuantityOfProducts (@RequestBody List<String> productNames) {

    Map<String, Integer> result = new HashMap<>();

    for (String name : productNames) {
        result.put(name, dummyInventoryServiceLayer.getQuantityOfProduct(name));
    }

    return ResponseEntity.ok(result);
}
```

Sample request

```
[
  "Product2",
  "Product5",
  "Product9"
]
```

Sample response

```
{
    "Product5": 200,
    "Product2": 0,
    "Product9": 21
}
```

Expose the methods of Inventory service

Example URL: /inventory/reduce

```
@PutMapping("/reduce")
public ResponseEntity <Void> reduceInventory(@RequestBody List<ProductQuantityDto> items) {

    boolean allReduced = true;
    for (ProductQuantityDto item : items) {
        boolean success = dummyInventoryServiceLayer.reduceInventory(item.getProductName(), item.getQuantity());
        if (!success) {
            allReduced = false;
        }
    }
    if (allReduced) {
        return ResponseEntity.ok().build();
    } else {
        return ResponseEntity.badRequest().build();
    }
}
```

ResponseEntity.ok().build() → HTTP **200 OK** with no body.

ResponseEntity.badRequest().build() → HTTP **400 Bad Request**, indicating that some operation failed.

Expose the methods of Inventory service

Example URL: /inventory/reduce

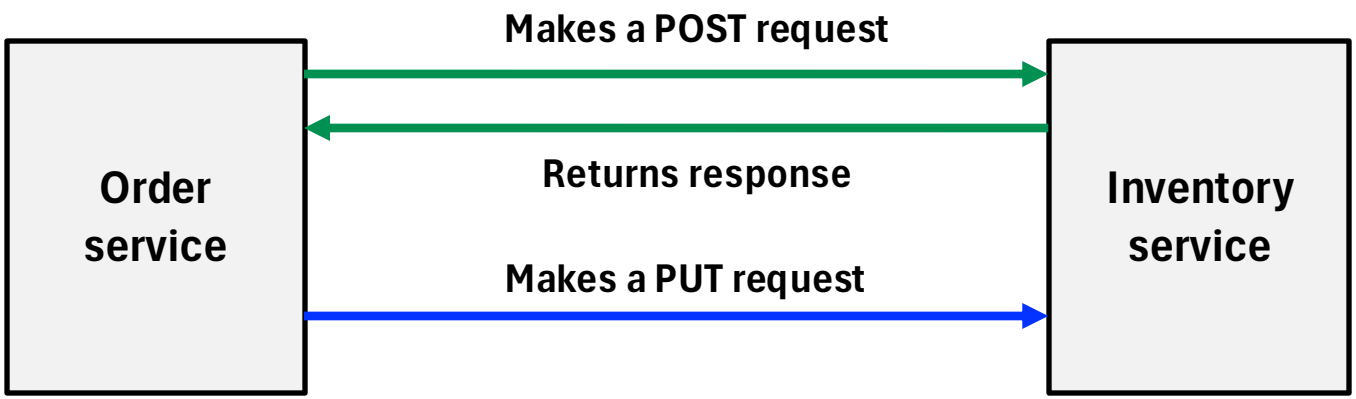
```
@PutMapping("/reduce")
public ResponseEntity <Void> reduceInventory(@RequestBody List<ProductQuantityDto> items) {

    boolean allReduced = true;
    for (ProductQuantityDto item : items) {
        boolean success = dummyInventoryServiceLayer.reduceInventory(item.getProductName(), item.getQuantity());
        if (!success) {
            allReduced = false;
        }
    }
    if (allReduced) {
        return ResponseEntity.ok().build();
    } else {
        return ResponseEntity.badRequest().build();
    }
}
```

Sample request

```
[
  {"productName": "Product1", "quantity": 2},
  {"productName": "Product4", "quantity": 3},
  {"productName": "Product8", "quantity": 2}
]
```

Communication between Order and Inventory services

To create an order, the Order Service should know product quantities in the Inventory to ensure the requested product are available for purchase.



After an order is successfully created, the Inventory Service should update its database by deducting the ordered product quantities.

Expose the methods of Order service

It calls **dummyOrderServiceLayer.createOrder(order)** in the Service Layer:

Example URL: /order/create

```
@PostMapping ("/create")
public ResponseEntity<Void> createOrder(@RequestBody Order order) {
    if (dummyOrderServiceLayer.createOrder(order))
        return ResponseEntity.noContent().build();
    else
        return  ResponseEntity.status(status:500).build();
}
```

Expose the methods of Order service

Example URL: /order/create

```
@PostMapping ("/create")
public ResponseEntity<Void> createOrder(@RequestBody Order order) {
    if (dummyOrderServiceLayer.createOrder(order))
        return ResponseEntity.noContent().build();
    else
        return  ResponseEntity.status(status:500).build();
}
```

Sample request

```
{
    "orderId": 14,
    "products": [
        {
            "productName": "Product7",
            "quantity": 3
        },
        {
            "productName": "Product2",
            "quantity": 2
        }
        ],
        "totalPrice": 239.75,
        "orderDate": "2021-09-09"
}
```

createOrder method in the Service Layer

```java
public boolean createOrder (Order order) {


    // First check the quantity of the ordered products in inventory

    // Exteract product names form order
    List <String> productNames = new ArrayList<>();
    for (OrderLine item : order.getProducts()) {
        productNames.add(item.getProductName());
    }

    // Check available quantities
    String inventoryURL = "http://localhost:4001/inventory";
    HttpEntity<List<String>> request = new HttpEntity<>(productNames);
    ResponseEntity<Map<String, Integer>> response = restTemplate.exchange(
            inventoryURL +"/check-quantities",
            HttpMethod.POST,
            request,
            new ParameterizedTypeReference<Map<String, Integer>>() {}
    );
    Map<String, Integer> products_availableQuantities = response.getBody();


    // Verify order can be fulfilled
    for (OrderLine item : order.getProducts()) {
        int availableInInventory = products_availableQuantities.getOrDefault(item.getProductName(), defaultValue:0);
        if (availableInInventory < item.getQuantity()) {
            return false;
        }
    }
    orders.add(order);

    // Reduce inventory
    Set<OrderLine> itemsToReduce = order.getProducts();
    HttpEntity<Set<OrderLine>> reduceRequest = new HttpEntity<>(itemsToReduce);
    ResponseEntity<Void> reduceResponse = restTemplate.exchange(
        inventoryURL + "/reduce",
        HttpMethod.PUT,
        reduceRequest,
        responseType:Void.class);

    return true;
}
```

createOrder method in the Service Layer

This part of code calls the **Inventory Service's endpoint** (@PostMapping("/check-quantities")) to return the quantity of each product in the Order.

```
// Check available quantities
String inventoryURL = "http://localhost:4001/inventory";
HttpEntity<List<String>> request = new HttpEntity<>(productNames);
ResponseEntity<Map<String, Integer>> response = restTemplate.exchange(
        inventoryURL +"/check-quantities",
        HttpMethod.POST,
        request,
        new ParameterizedTypeReference<Map<String, Integer>>() {}
);
Map<String, Integer> products_availableQuantities = response.getBody();
```

restTemplate.exchange(...) makes the POST request to the Inventory Service.
When you call restTemplate.exchange(...), you must pass the request in an **HttpEntity**.

HttpEntity<T> is a Spring class that represents an **HTTP request or response entity**.

createOrder method in the Service Layer

```java
// Check available quantities
String inventoryURL = "http://localhost:4001/inventory";
HttpEntity<List<String>> request = new HttpEntity<>(productNames);
ResponseEntity<Map<String, Integer>> response = restTemplate.exchange(
        inventoryURL +"/check-quantities",
        HttpMethod.POST,
        request,
        new ParameterizedTypeReference<Map<String, Integer>>() {}
);
Map<String, Integer> products_availableQuantities = response.getBody();
```

response

```json
[
  {"productName": "Product2", "quantity": 20},
  {"productName": "Product7", "quantity": 16}
]
```

request

```java
[
  "Product7",
  "Product2"
]
```

createOrder method in the Service Layer

This part of code checks if the quantity of each ordered product is equal or less than the quantity of each product in the Inventory. If not, return false, else it will create an order.

```
// Verify order can be fulfilled
for (OrderLine item : order.getProducts()) {
    int availableInInventory = products_availableQuantities.getOrDefault(item.getProductName(), defaultValue:0);
    if (availableInInventory < item.getQuantity()) {
        return false;
    }
}
orders.add(order);
```

createOrder method in the Service Layer

This part of code calls the **Inventory Service's endpoint** (@PutMapping("/reduce")) to reduce the quantity of each product in the inventory.

```
// Reduce inventory
Set<OrderLine> itemsToReduce = order.getProducts();
HttpEntity<Set<OrderLine>> reduceRequest = new HttpEntity<>(itemsToReduce);
ResponseEntity<Void> reduceResponse = restTemplate.exchange(
    inventoryURL + "/reduce",
    HttpMethod.PUT,
    reduceRequest,
    responseType:Void.class);

return true;
```

```
[
  {"productName": "Product2", "quantity": 2},
  {"productName": "Product7", "quantity": 3}
]
```

References

- What Is a REST API? Examples, Uses, and Challenges (https://blog.postman.com/rest-api-examples/)
- https://www.geeksforgeeks.org/advance-java/spring-boot/
- https://azure.microsoft.com/en-au/resources/cloud-computing-dictionary/what-is-java-spring-boot
- Build & Deploy a Production-Ready Patient Management System with Microservices: Java Spring Boot AWS https://www.youtube.com/watch?v=tseqdcFfTUY