

Practice Exercise

This document provides a list of exercises to be practiced by learners. Please raise feedback in Talent Next, should you have any queries.

Skill	Java Microservices
Proficiency	S2
Document Type	Lab Practice Exercises
Author	L & D
Current Version	3.0
Current Version Date	10-July-2023
Status	Active

Document Control

Version	Change Date	Change Description	Changed By
1.0	30-Oct-2019	Baseline version	Manpreet Singh Bindra
2.0	25-Sep-2022	Added the problem statements for the topics like creating table, multiple profiles, Actuator, AWS Parameter Store, Microservice with MySQL. Modified the detailed description to the existing problem statements.	Manpreet Singh Bindra
3.0	10-July-2023	Added the Spring Initializer (start.spring.io) to all the problem statements. Added the new problem statements for the topics like Spring Cloud Eureka Server, Spring Cloud Eureka Client, Spring Cloud Gateway, Axon Server, CQRS, Event Sourcing, Bean Validation, Message Dispatch Interceptor, Set Based Consistency, Handling Errors and Rollback Transaction, Orchestration based Saga, and Compensating Transaction.	Manpreet Singh Bindra

Contents

Practice Exercise	1
Document Control.....	2
Problem Statement 1: Basic Docker Commands.....	4
Problem Statement 2: Create Spring Boot Microservice for ProductService	5
Problem Statement 3: Configure Microservices with Eureka Service Registry Server	6
Problem Statement 4: Enable Dynamic Registration to Product Microservice	8
Problem Statement 5: Implementing Spring Cloud Gateway in Microservices	9
Problem Statement 6: Running Axon Server in Docker Container	13
Problem Statement 7: Implementing the CQRS & Event Sourcing Design Pattern in Product Microservice.....	15
Problem Statement 8: Validate Request Body, Bean Validation in Product Microservice	24
Problem Statement 9: Apply Command Validation using Message Dispatch Interceptor.....	26
Problem Statement 10: Set Based Consistency Validation in CQRS and Event Sourcing Application	28
Problem Statement 11: Handling Errors and Rollback Transaction with Axon	32
Problem Statement 12: Implementing the CQRS & Event Sourcing Design Pattern in Orders Microservice.....	37
Problem Statement 13: Orchestration based Saga – Reserve Product in Stock	43
Problem Statement 14: Orchestration based Saga – Fetch Payment Details.....	52
Problem Statement 15: Orchestration based Saga – Process User Payment.....	59
Problem Statement 16: Saga Compensating Transaction in Microservices.....	65

Note: Every Problem Statement start on a new page

Problem Statement 1: Basic Docker Commands

Try out some docker basic commands:

- 1) Write a command to pull an openjdk:8-apline and mysql image from registry to local machine.
- 2) Write a command to show all the images.
- 3) Write a command to run both the container and link the mysql server to openjdk:8-apline instance in interactive mode.
- 4) Write a command to run both the container and link the mysql server to openjdk:8-apline instance in detached mode.
- 5) Write a command to list all the registered containers that are running.
- 6) Write a command to show all the logs of the containers.
- 7) Write a command to interact with the containers.
- 8) Write a command to stop and start the container.
- 9) Write a command to create your own registry for faster performance.
- 10) Write a command to create your own image and list all images.
- 11) Write a command to push, pull and remove the image from your own registry.
- 12) Write a command to remove the stopped containers.
- 13) Write a command to list information about one or more networks.
- 14) Write a Dockerfile for the "Hello World" Java program and build the customize image.
- 15) Write a command to run the customize image and push the image to hub.docker.com.

Problem Statement 2: Create Spring Boot Microservice for ProductService

Mphasis got a requirement to create an eStoreApplication portal, wherein multiple users access the product details. We need to design the API for the application so we can achieve the interoperability feature in the application.

Technology stack:

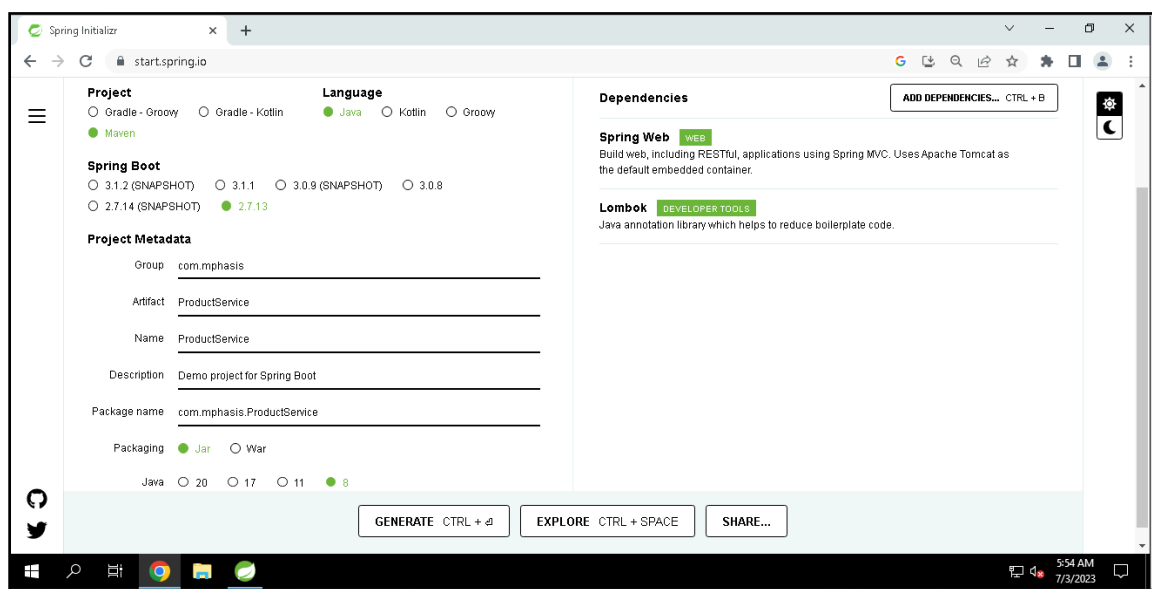
- Spring Web
- Lombok

Building a RESTful web application where CRUD operations to be carried out on entities.

Initially we will have only controller layers into the application:

1. Create a new Project for **ProductService** using the **Spring Initializr** (start.spring.io).

Refer the below screenshots:



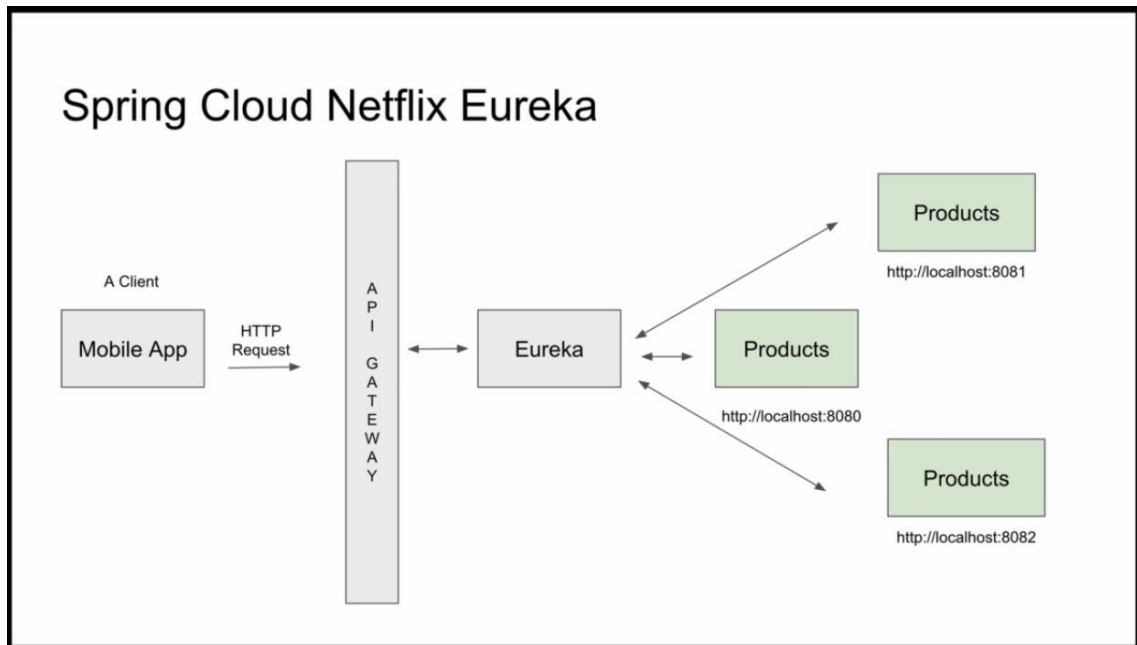
2. Select the **Spring Web**, **Lombok**, and **Eureka Discovery Client** dependencies.
3. Click on **GENERATE** button or **CTRL + Enter** to create the project structure.
4. Let's import the **ProductService** maven project in **STS**.
5. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
6. Let's start building our application.
7. Finally, create a **ProductController** will have the following Uri's:

URI	METHODS	Description
/products	POST	Return a String - "HTTP POST Method Handled"
/products	PUT	Return a String - "HTTP PUT Method Handled"
/products	GET	Return a String - "HTTP GET Method Handled"
/products	DELETE	Return a String - "HTTP DELETE Method Handled"

8. Running on a web server tier (using tomcat).

Problem Statement 3: Configure Microservices with Eureka Service Registry Server

The "eBookStore" application instance will expose a remote API such as HTTP/REST at a particular location (host and port). To overcome the challenge of dynamically changing service instances and their locations. The code deployers intended to create a service registry, which is a database containing information about services, their instances, and their locations.



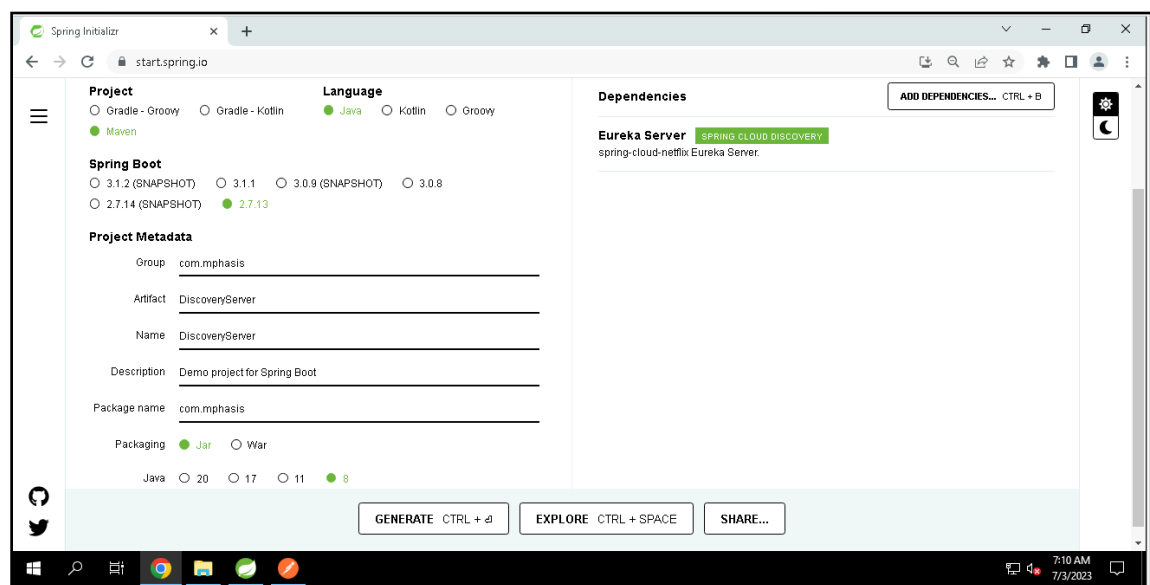
Technology stack:

- Spring Cloud Eureka Server

Steps for Spring Cloud Config Server:

1. Create a new Project for **DiscoveryServer** using the **Spring Initializr** (start.spring.io).

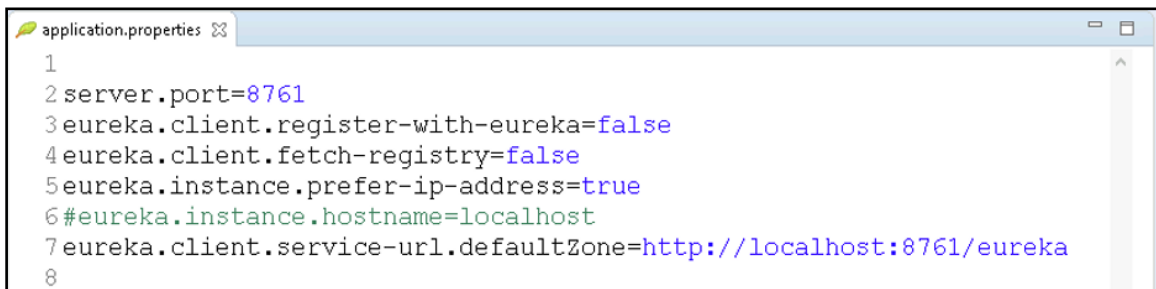
Refer the below screenshots:



2. Select the **Eureka Server** dependency.

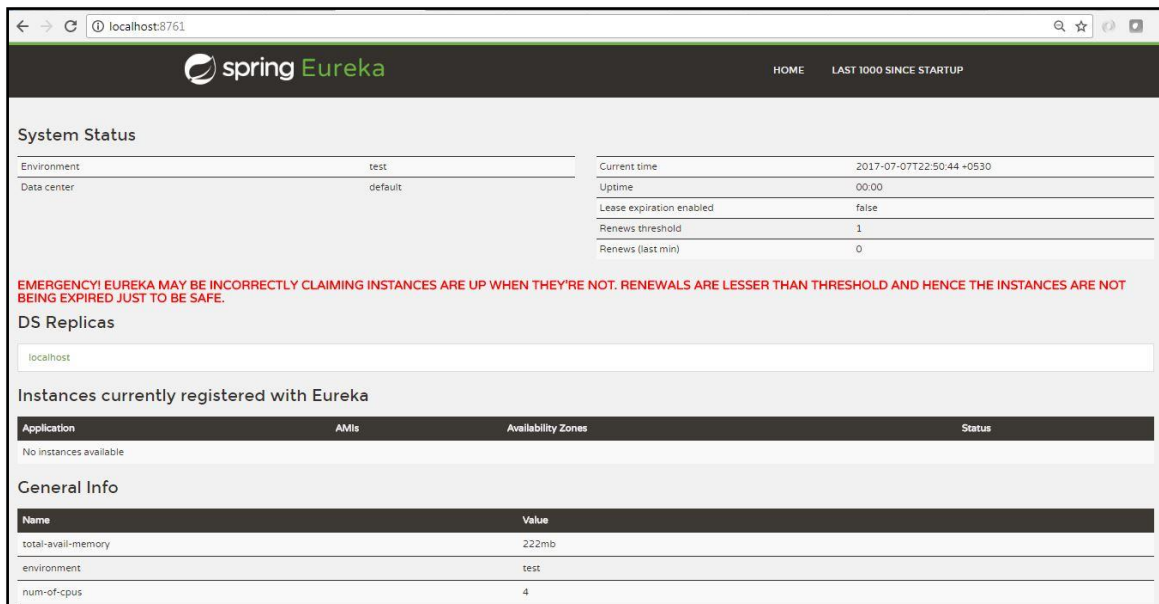
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

3. Click on **GENERATE** button or **CTRL + Enter** to create the project structure.
4. Let's import the **DiscoveryServer** maven project in **STS**.
5. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
6. In the Application class, Add **@EnableEurekaServer** annotation.
7. Ensure the server is running on 8761.



```
1
2 server.port=8761
3 eureka.client.register-with-eureka=false
4 eureka.client.fetch-registry=false
5 eureka.instance.prefer-ip-address=true
6 #eureka.instance.hostname=localhost
7 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
8
```

8. Start the application.
9. Verify the Eureka Server: <http://localhost:8761/>



The screenshot shows the Spring Eureka Server web interface at <http://localhost:8761/>. The interface includes a navigation bar with "HOME" and "LAST 1000 SINCE STARTUP". The main content area is divided into several sections:

- System Status:** A table showing system metrics.

System Status	
Environment	test
Data center	default
Current time	2017-07-07T22:50:44 +0530
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0
- EMERGENCY!** EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.
- DS Replicas:** A table showing the status of the data center replicas.

DS Replicas
localhost
- Instances currently registered with Eureka:** A table showing the status of registered instances.

Application	AMIs	Availability Zones	Status
No instances available			
- General Info:** A table showing general information about the server.

Name	Value
total-avail-memory	222mb
environment	test
num-of-cpus	4

Problem Statement 4: Enable Dynamic Registration to Product Microservice

Now we will configure the ProductService to register with Spring Cloud Eureka Server.

Technology stack:

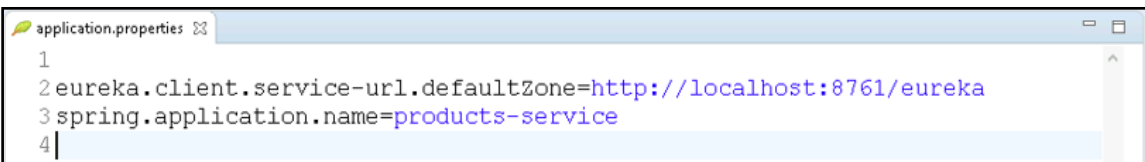
- Spring Cloud Eureka Client

Steps for Spring Cloud Config Client:

1. Refer the **ProductService** created in the problem statement – 2 and add the **Eureka Client** starter to the application.

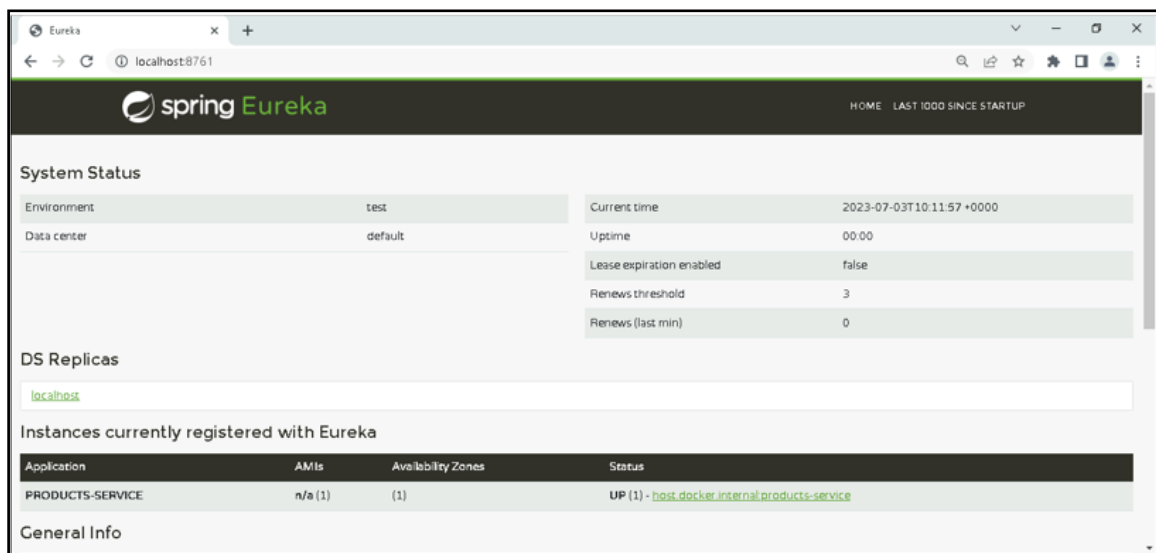
```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2. Include the **application name** and **eureka.client.serviceUrl.defaultZone** in the application.properties files. For the Product Service application to dynamically register to Discovery Server.



```
1
2 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
3 spring.application.name=products-service
4
```

3. In the Application class, Add **@EnableEurekaClient/@EnableDiscoveryClient** annotation.
4. Ensure that the Discovery Server and Product Service is running.
5. Again, verify the Eureka Server: <http://localhost:8761/>



The screenshot shows the Spring Eureka Server web interface at localhost:8761. The page displays system status, DS replicas, and instances currently registered with Eureka.

System Status	
Environment	test
Data center	default
Current time	2023-07-03T10:11:57 +0000
Uptime	00:00
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

DS Replicas: localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCTS-SERVICE	n/a (1)	(1)	UP (1) - host.docker.internal/products-service

General Info

Problem Statement 5: Implementing Spring Cloud Gateway in Microservices

In this problem statement, we will use Spring Cloud Gateway to implement API Gateway. The Spring Cloud Gateway is a non-blocking API. A thread is always available to process the incoming request while using non-blocking API. These requests are then handled asynchronously in the background, and the response is returned once completed. When using Spring Cloud Gateway, no incoming request is ever blocked.

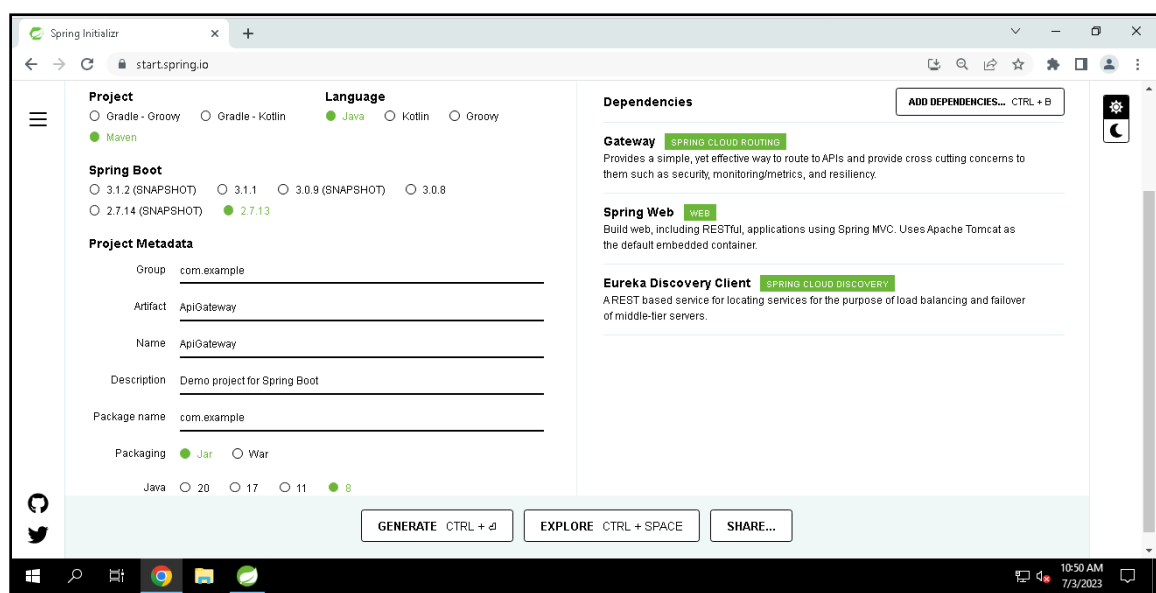
Technology stack:

- Spring Cloud Routing - Gateway

Steps for implementing API Gateway:

1. Ensure the Discovery Server and Product Service is running.
2. Now let's implement an API gateway that acts as a single-entry point for a collection of microservices.
3. Create a new Project for **ApiGateway** using the **Spring Initializr** (start.spring.io).

Refer the below screenshots:



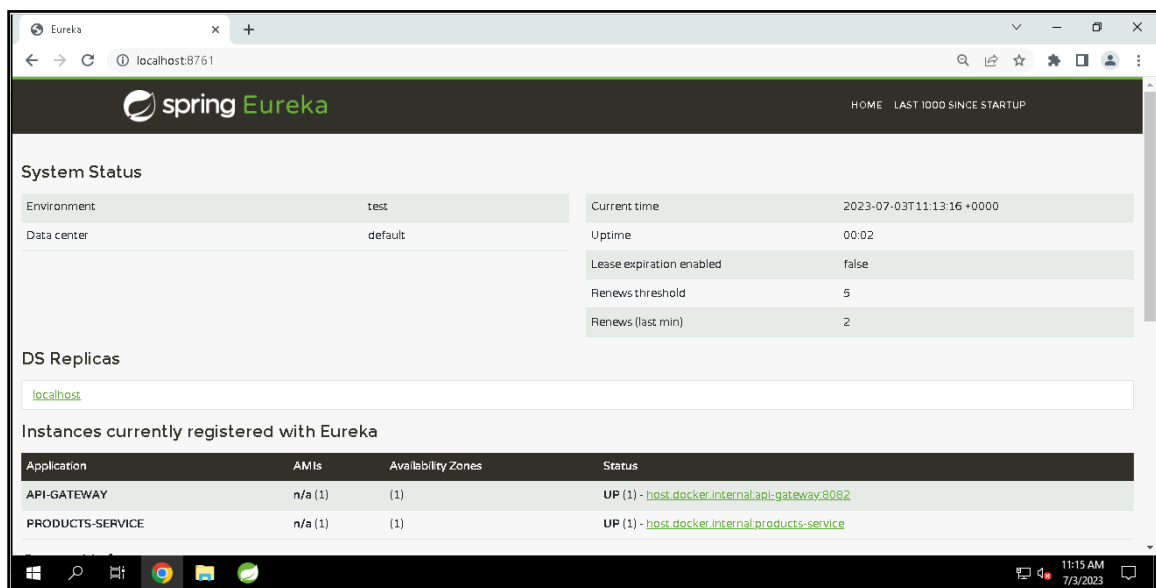
4. Select the **Gateway**, **Spring Web**, and **EurekaDiscoveryClient** dependencies.
5. Click on GENERATE button or CTRL + Enter to create the project structure.
6. Let's import the ApiGateway maven project in STS.
7. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
8. Add **@EnableEurekaClient/@EnableDiscoveryClient** in the Application class.

9. In application.properties file, enable the automatic mapping of gateway routes and add the application name and eureka client serviceUrl.

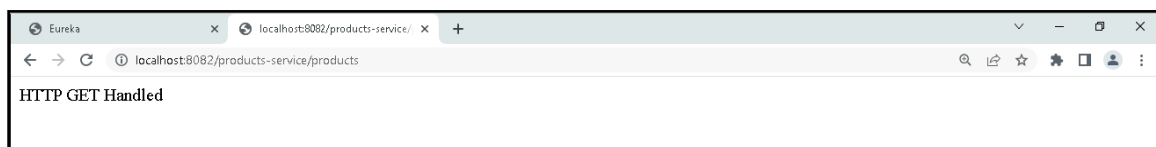
```
application.properties
1
2 spring.application.name=api-gateway
3 server.port=8082
4 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5
6 spring.cloud.gateway.discovery.locator.enabled=true
7 spring.cloud.gateway.discovery.locator.lower-case-service-id=true
8
```

10. Start the ApiGateway.

11. Check the proxy running instances is also registered with the Eureka Server.



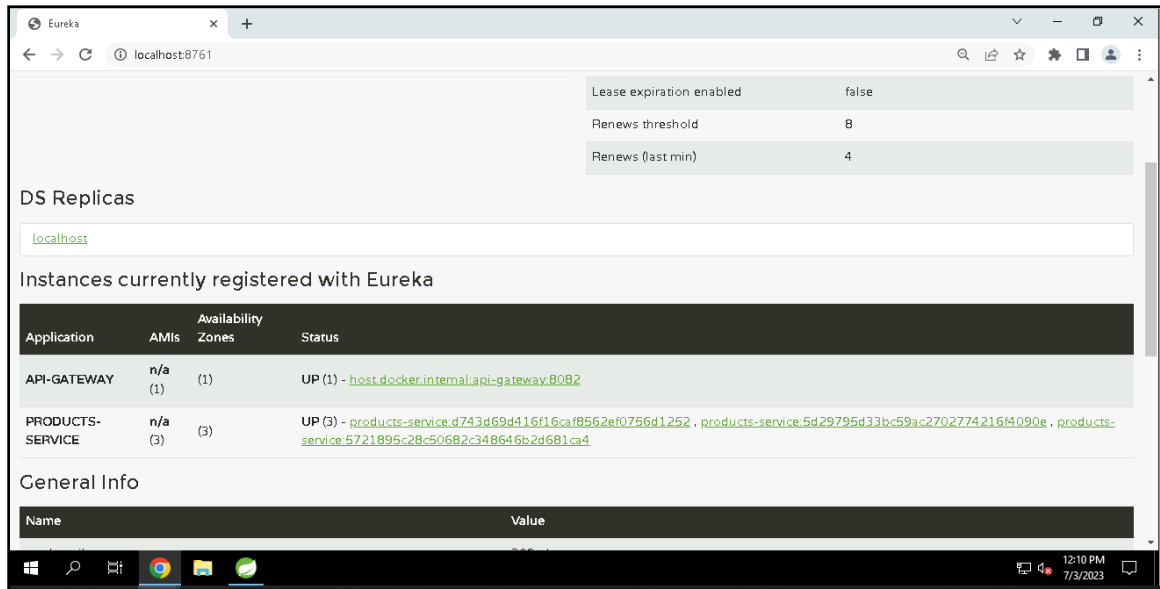
12. Test the Proxy: <http://localhost:8082/products-service/products>



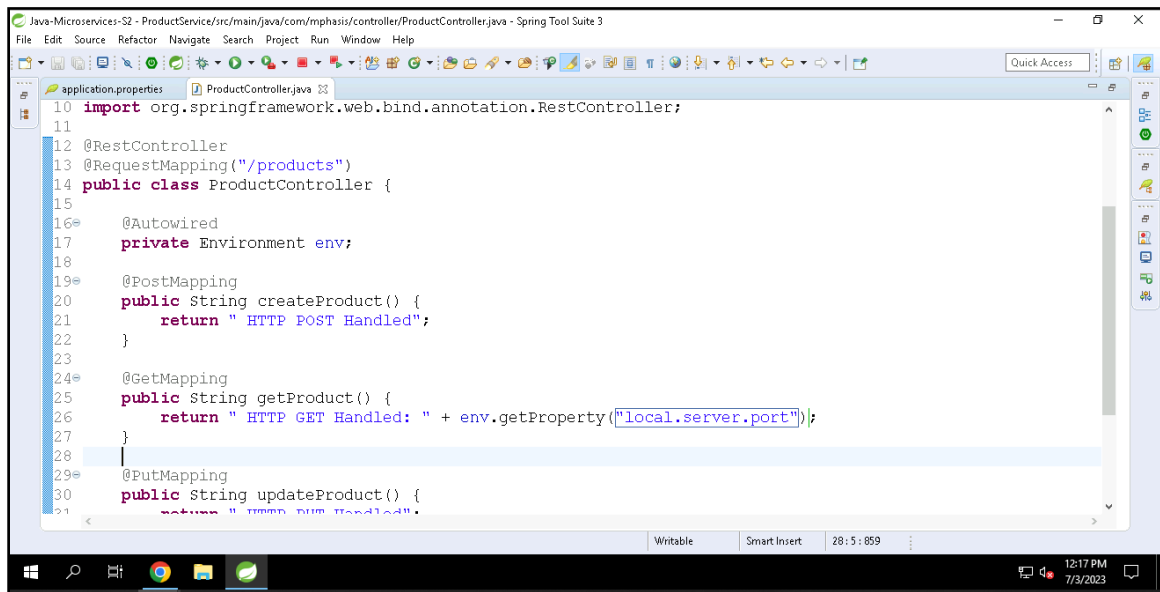
13. Let's add the random port and instance-id property to ProductService/application.properties file:

```
application.properties
1
2 server.port=0
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.application.name=products-service
5
6 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
7
```

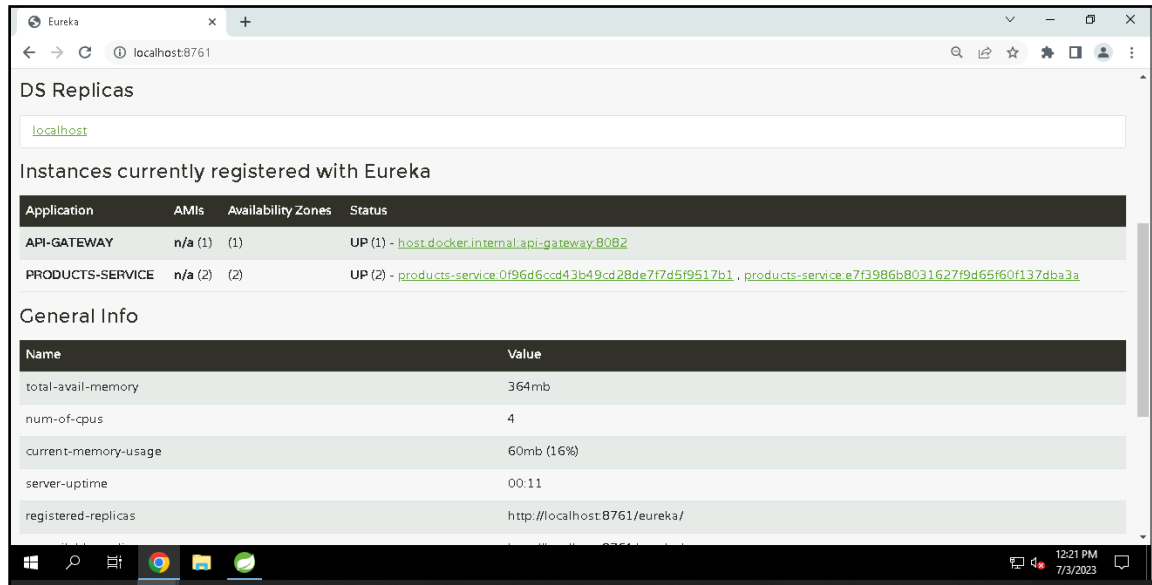
14. Restart the Discovery Server and execute Product Service three times, you will find 3 instances are running.



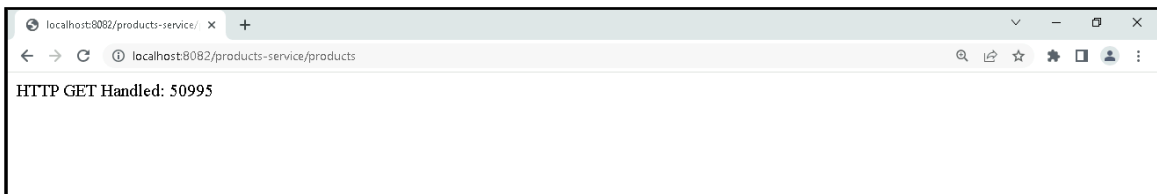
15. Test how the Load Balancing works.
16. Modify the code of GET handler method in ProductController class.



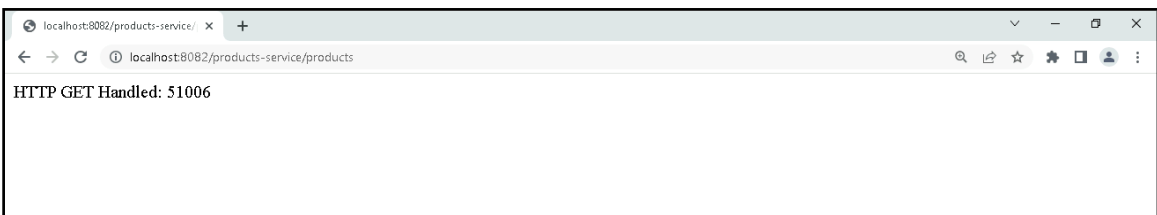
17. Restart the Discovery Server and execute Product Service two times, you will find 2 instances are running.
18. Restart the ApiGateway also.



19. Test the Proxy: <http://localhost:8082/products-service/products>



20. Refresh again:



Problem Statement 6: Running Axon Server in Docker Container

Axon Server is a zero-configuration message router and event store. The message router has a clear separation of different message types: **events**, **queries**, and **commands**. The event store is optimized to handle huge volumes of events without performance degradation.

Axon Server is initially built to support distributed Axon Framework microservices. Starting from Axon Framework version 4 Axon Server is the default implementation for the **CommandBus**, **EventBus/EventStore**, and **QueryBus** interfaces.

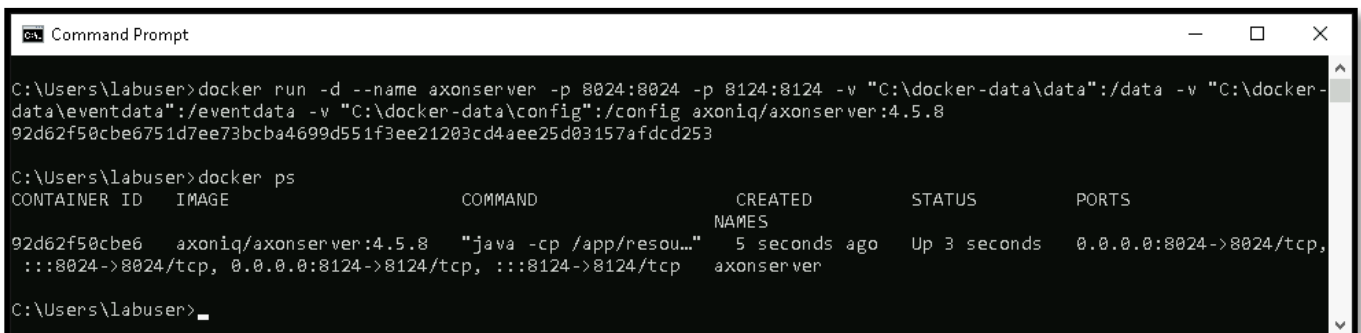
Steps for Running Axon Server in Docker Container:

1. Create a folder docker-data folder on C Drive and create three sub-folders: data, event, config.
2. The “/data” and “/eventdata” directories are created as volumes, and their data will be accessible on your local filesystem somewhere in Docker’s temporary storage tree. Alternatively, you can tell docker to use a specific directory, which will allow you to put it at a more convenient location. A third directory, not marked as a volume in the image, is important for our case: If you put an “axonserver.properties” file in “/config”, it can override the settings and add new ones.
3. Add the below properties to **axonserver.properties**:

```
server.port=8024
axoniq.axonserver.name=My Axon Server
axoniq.axonserver.hostname=localhost
axoniq.axonserver.devmode.enabled=true
```

4. Run the following command to start Axon Server in a Docker container:

```
docker run -d \
--name axonserver \
-p 8024:8024 \
-p 8124:8124 \
-v "C:\docker-data\data":/data \
-v "C:\docker-data/eventdata":/eventdata \
-v "C:\docker-data/config":/config \
axoniq/axonserver:4.5.8
```

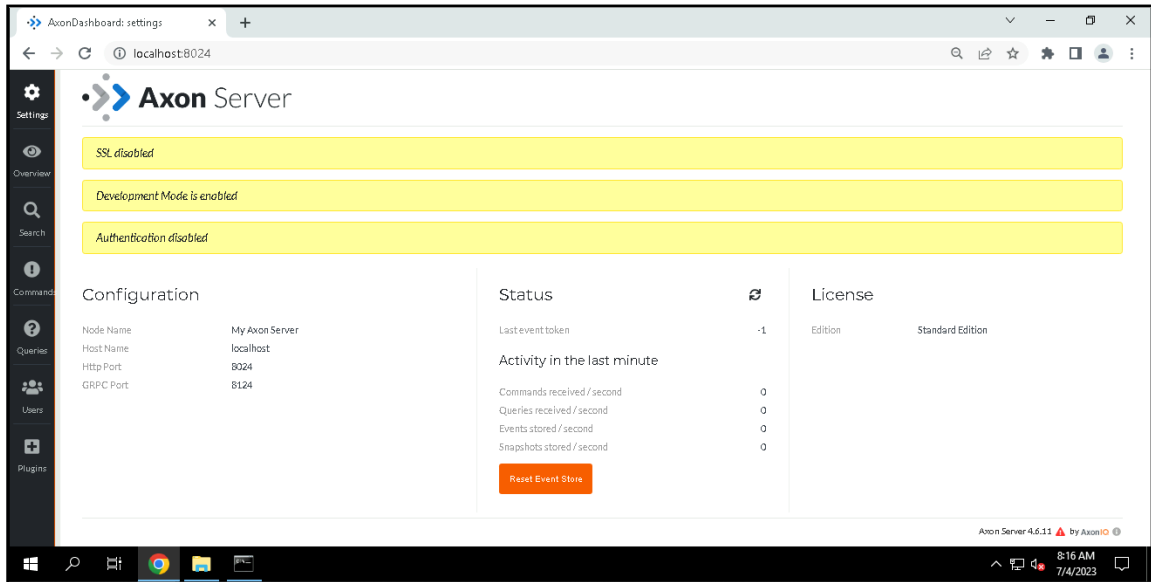


```
Command Prompt
C:\Users\labuser>docker run -d --name axonserver -p 8024:8024 -p 8124:8124 -v "C:\docker-data\data":/data -v "C:\docker-
data\eventdata":/eventdata -v "C:\docker-data\config":/config axoniq/axonserver:4.5.8
92d62f50cbe6751d7ee73bcba4699d551f3ee21203cd4aee25d03157afdc253

C:\Users\labuser>docker ps
CONTAINER ID   IMAGE               COMMAND                  CREATED        STATUS        PORTS
92d62f50cbe6   axoniq/axonserver:4.5.8   "java -cp /app/resou..."   5 seconds ago   Up 3 seconds   0.0.0.0:8024->8024/tcp,
:::8024->8024/tcp, 0.0.0.0:8124->8124/tcp, :::8124->8124/tcp
axonserver
```

Java Microservices – S2 – Practice Exercise

Access the Axon Server, via, <http://localhost:8024>



The screenshot shows the Axon Server dashboard in a web browser. The browser's address bar displays 'localhost:8024'. The dashboard has a dark sidebar on the left with navigation options: Settings, Overview, Search, Command, Queries, Users, and Plugins. The main content area is titled 'Axon Server' and features three yellow status bars at the top: 'SSL disabled', 'Development Mode is enabled', and 'Authentication disabled'. Below these, the dashboard is divided into three columns: Configuration, Status, and License. The Configuration column lists 'Node Name' as 'My Axon Server', 'Host Name' as 'localhost', 'Http Port' as '8024', and 'GRPC Port' as '8124'. The Status column shows 'Last event token' as '-1' and 'Activity in the last minute' with metrics for Commands, Queries, Events, and Snapshots, all at '0'. A 'Reset Event Store' button is located at the bottom of the Status column. The License column indicates the 'Edition' is 'Standard Edition'. The bottom of the dashboard shows 'Axon Server 4.6.11' and 'By AxonIO'.

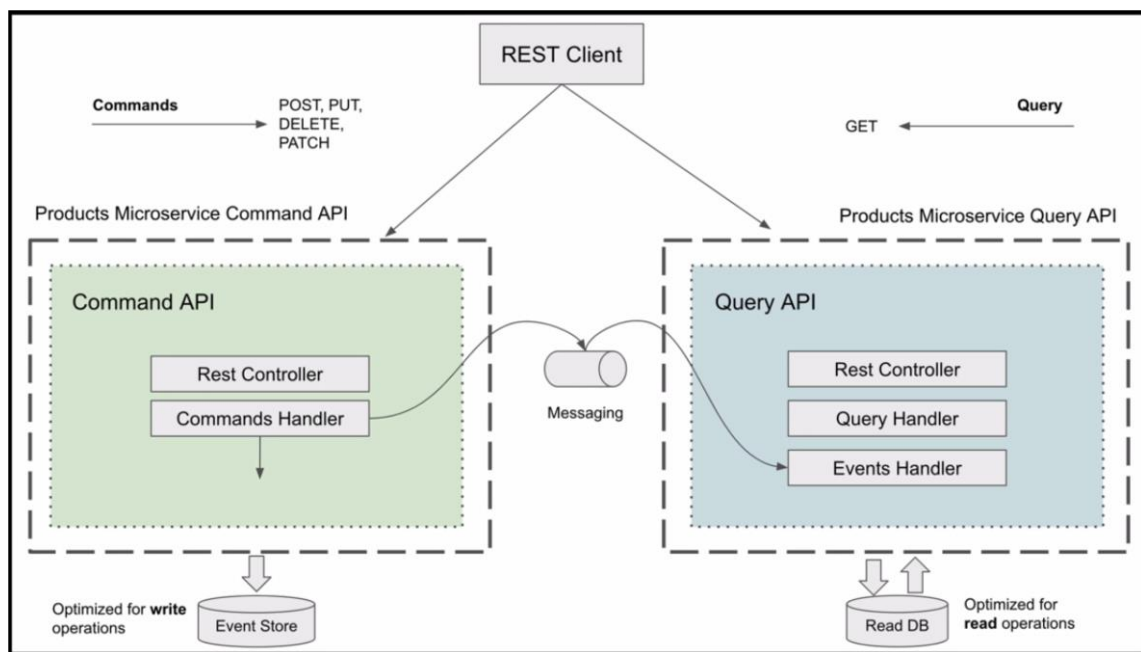
Configuration		Status	License
Node Name	My Axon Server	Last event token	-1
Host Name	localhost	Activity in the last minute	
Http Port	8024	Commands received / second	0
GRPC Port	8124	Queries received / second	0
		Events stored / second	0
		Snapshots stored / second	0
		Reset Event Store	
		Edition	Standard Edition

Problem Statement 7: Implementing the CQRS & Event Sourcing Design Pattern in Product Microservice

Now in the era of distributed system, you run a large-scale ecommerce store. You have a large user base who query your system for products much more than they buy them. In other words, your system has more read requests than write requests. As a result, you'd prefer to handle the high load of read requests separately from the relative low write requests. Also, suppose you need to write complex queries to read data from the same database that you write to, and this might impact its performance. Assume you also want to add additional security while writing data to the database. How do you design your system to cater these specific use cases?

CQRS (Command Query Responsibility Segregation) design pattern addresses these concerns. On a high level, you separate your read and write systems and keep them in sync.

Here is a diagram explaining the same:

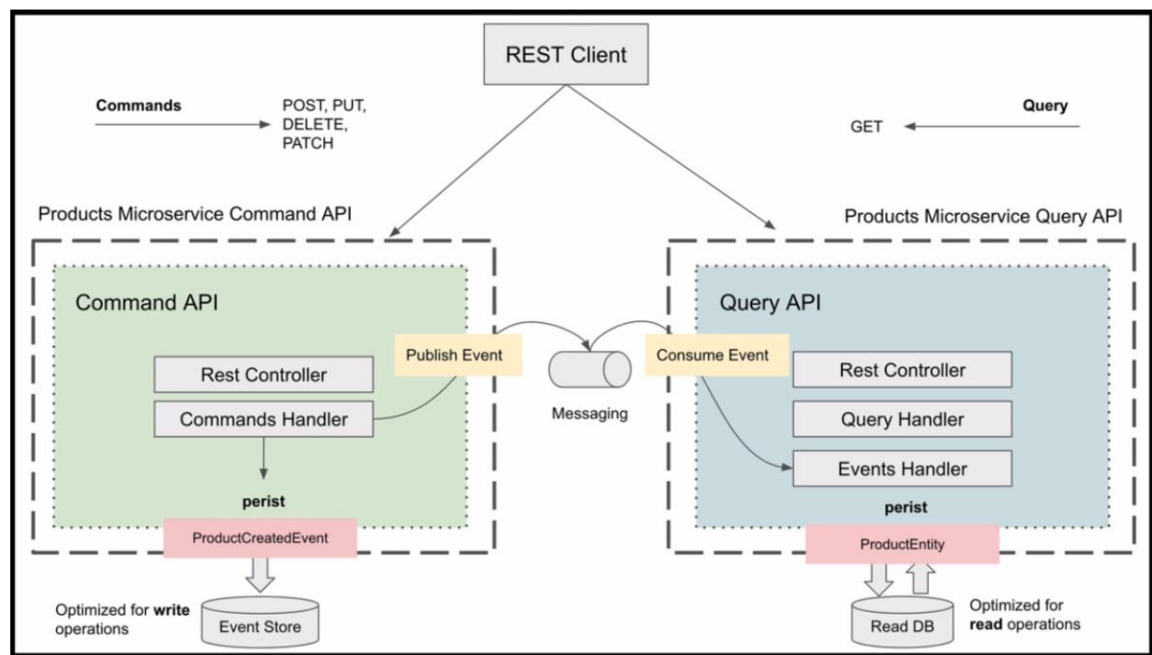


In some scenarios though you would want more than the current state, you might need all the states which the customer entry went through. For such cases, the design pattern “**Event Sourcing**” helps.

When a client application sends a POST request to create a Product, the Command Handler will handle the Command and Product Create Event will be created. And will be persisted into a database which is now going to be called **Event Store**. The ProductCreatedEvent will be published to all the other components who are interested in this event to consume it.

The **Events Handler** in the Query API on the right side, will consume the ProductCreatedEvent and read database will be updated with a new Record.

Here is a diagram explaining the same:



Now the difference between the Event Store on the left side and Read DB on the right side is that the Event Store database will contain the record of every single event that took place for the product but the read database on the right side will only contain 1 single record which is the latest state of the Product Details Entity.

We have an **historical data** that we can use for **audit purposes**, or we can use to reconstruct the state of the Object at any given time.

Technology stack:

- Spring Web
- Spring Data JPA
- H2 Database
- Spring Cloud Eureka Client
- Lombok
- Axon Spring Boot Starter
- Google Guava
- Spring Boot Starter Validation

Steps for implementing CQRS and Event Sourcing using Axon Server:

1. Refer the **ProductService** updated in the problem statement – 4.

2. Add the Spring Web, Spring Data JPA, H2 Database, Spring Cloud Eureka Client, Lombok, Axon Spring Boot Starter, Google Guava, and Spring Boot Starter Validation dependencies in pom.xml.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.axonframework</groupId>
  <artifactId>axon-spring-boot-starter</artifactId>
  <version>4.5.8</version>
</dependency>

<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>30.1-jre</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

3. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
4. Will have a separate package for Command API (com.mphasis.command) and Query API (com.mphasis.query).
5. Update the ProductController class.
6. The method that accepts the HTTP Post request, should accept the CreateProductRestModel payload as a request body.

7. The CreateProductRestModel annotated with @Data and should have the following fields:
private String title;
private BigDecimal price;
private Integer quantity;
8. This controller class should use the **Axon's CommandGateway** and publish the CreateProductCommand.
9. The CreateProductCommand annotated with @Data, @Builder and should have the following fields:
private final String productId;
private final String title;
private final BigDecimal price;
private final Integer quantity;

Where:
productId - is a randomly generated value. For example, UUID.randomUUID().toString() and annotated with **@TargetAggregateIdentifier**.
Should return the productId as String. If the exception raised by published code, handle and return the Localized Message as String.
10. Create a new class called ProductAggregate and make it handle the CreateProductCommand using **@CommandHandler** and publish the ProductCreatedEvent.
11. The ProductCreatedEvent class annotated with @Data and should have the following fields:
private String productId;
private String title;
private BigDecimal price;
private Integer quantity;
12. Apply the below validation to price and title in the Command Handler.

```
@CommandHandler
public ProductAggregate(CreateProductCommand createProductCommand) {
    // Validate Create Product Command

    if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
        throw new IllegalArgumentException("Price cannot be less or equal than zero");
    }

    if (createProductCommand.getTitle() == null
        || createProductCommand.getTitle().isEmpty()) {
        throw new IllegalArgumentException("Title cannot be empty");
    }
}
```

13. Use **AggregateLifecycle.apply(Object payload)** which apply a **DomainEventMessage** with given payload without metadata. Applying events means they are immediately applied (published) to the aggregate and scheduled for publication to other event handlers.

```
ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();
BeanUtils.copyProperties(createProductCommand, productCreatedEvent);
AggregateLifecycle.apply(productCreatedEvent);
```

14. The ProductAggregate class should also have an **@EventSourcingHandler** method that sets values for all fields in the ProductAggregate.

CQRS Persisting Event in the Product database:

15. Add the below DB properties in application.properties:

```
application.properties
2 server.port=0
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.application.name=products-service
5 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
8
9 spring.h2.console.settings.web-allow-others=true
10
11 spring.datasource.url=jdbc:h2:mem:mphasisdb
12 spring.datasource.driver-class-name=org.h2.Driver
13 spring.datasource.username=sa
14 spring.datasource.password=password
15
16 #Accessing the H2 Console
17 spring.h2.console.enabled=true
18 spring.h2.console.path=/h2-console
```

16. Create a new @Component class called ProductEventsHandler inside com.mphasis.query package.
17. Create a new JPA Repository called ProductRepository inside com.mphasis.core.data package and inject it into ProductEventsHandler using constructor-based dependency injection.
18. Add two find methods in ProductRepository interface:

```
ProductEntity findByProductId(String productId);
ProductEntity findByProductIdOrTitle(String productId, String title);
```

19. The ProductEventsHandler class should have one @EventHandler method that handles the ProductCreatedEvent and persists product details into the "read" database.
20. To persist product details into the database, create a new JPA Entity class called ProductEntity inside com.mphasis.core.data package. Annotate the ProductEntity class with:

```
@Data
@Entity
@Table(name = "products")
```

and make the ProductEntity class have the following fields:

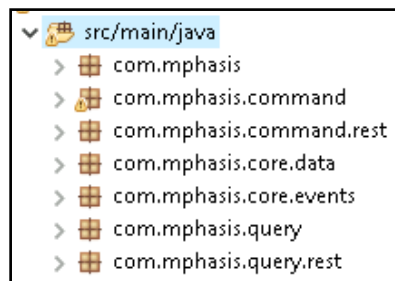
```
@Id
@Column(unique = true)
private String productId;
@Column(unique = true)
private String title;
private BigDecimal price;
private Integer quantity;
```

CQRS, Querying Data:

Further, we will have one Controller class for Command API and one Controller class for Query API which will help you to split the microservices if required tomorrow.

21. Refactor Command API REST Controller:

- Rename ProductController to ProductCommandController for better visualization.
- Rename the Package com.mphasis.controller to com.mphasis.command.rest.
- Rename the Package com.mphasis.core to com.mphasis.core.event.
- So, total we will have 3 main package – command, core, and query.



22. Create the ProductsQueryController class inside com.mphasis.query.rest package.

23. The method that accepts the HTTP Get request, should have List<ProductRestModel> as a response body.

24. The ProductRestModel annotated with @Data and should have the following fields:

```
private String productId;  
private String title;  
private BigDecimal price;  
private Integer quantity;
```

25. This controller class should use the **Axon's QueryGateway** to dispatch an instance of FindProductQuery. As we use the gateway's query() method to issue a point-to-point query. Because we are specifying ResponseTypes.multipleInstancesOf(ProductRestModel.class), Axon knows we only want to talk to query handlers whose return type is a collection of ProductRestModel objects.

26. Create a new @Component class called ProductsQueryHandler inside com.mphasis.query package.

27. Refer the JPA Repository called ProductRepository and inject it into ProductsQueryHandler using constructor-based dependency injection.

28. The ProductsQueryHandler class should have one **@QueryHandler** method that handles the FindProductsQuery and fetch all the product details from the "read" database.

Run and make it work:

29. Let's comment below methods as we don't require them now.

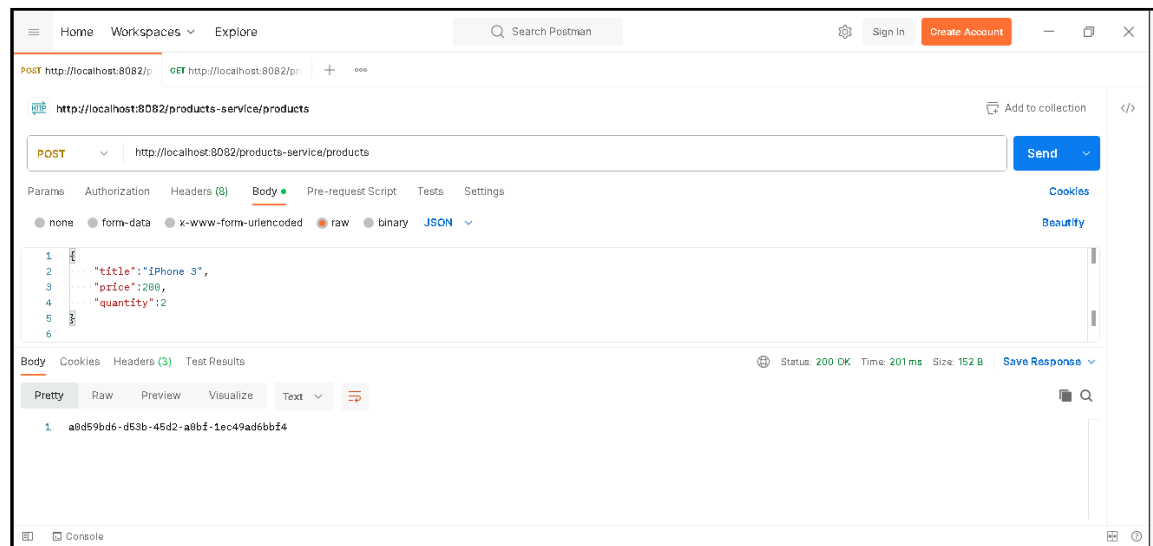


```
44     }
45     return returnValue;
46 }
47 /*
48 @GetMapping
49 public String getProduct() {
50     return " HTTP GET Handled: " + env.getProperty("local.server.port");
51 }
52
53 @PutMapping
54 public String updateProduct() {
55     return " HTTP PUT Handled";
56 }
57
58 @DeleteMapping
59 public String deleteProduct() {
60     return " HTTP DELETE Handled";
61 }
62 */
63 }
64
```

30. Run the Axon Server using Docker command.

31. Start the Discovery Server (Eureka Server), Product Service, and ApiGateway.

32. Send a POST request to Create Product.



POST http://localhost:8082/products-service/products

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary JSON

```
1 {
2   "title": "iPhone 3",
3   "price": 200,
4   "quantity": 2
5 }
6
```

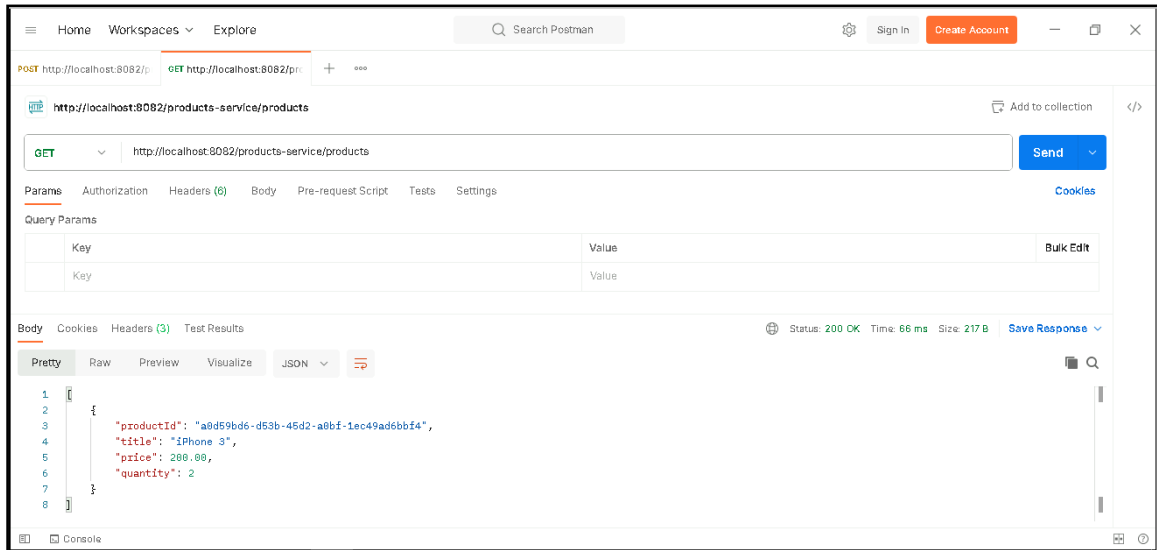
Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize Text

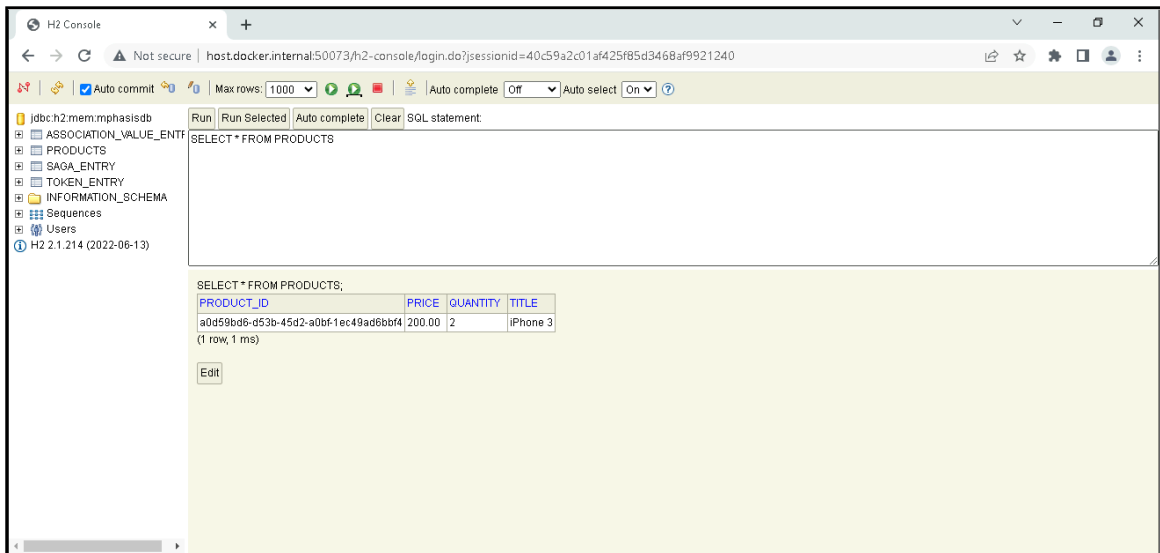
1 a0d59bd6-d53b-45d2-a0bf-1ec49ad6bbf4

Status: 200 OK Time: 201 ms Size: 152 B Save Response

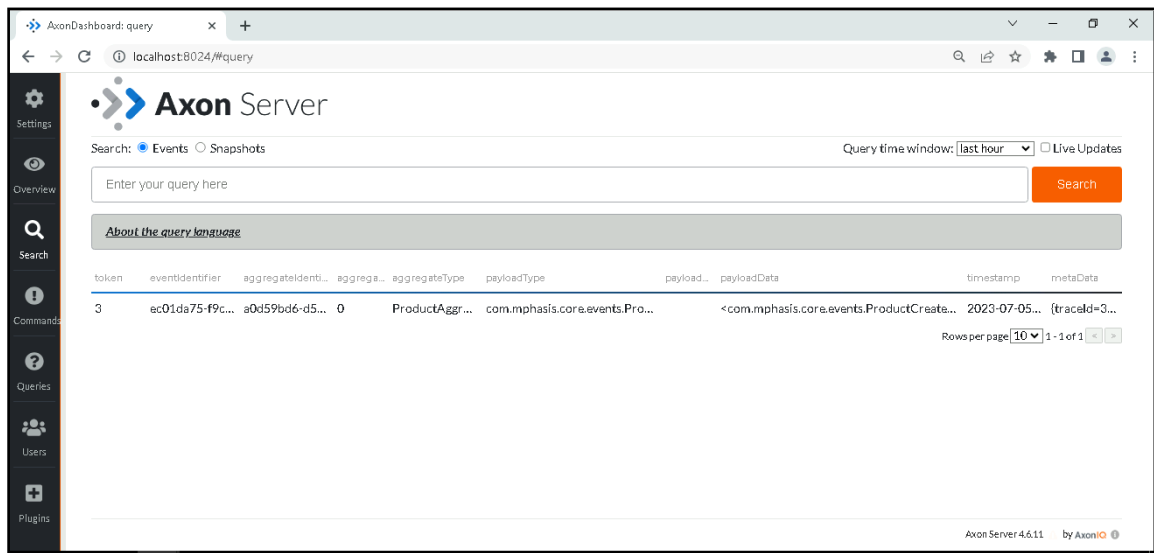
33. Send a GET request to Query the Products.



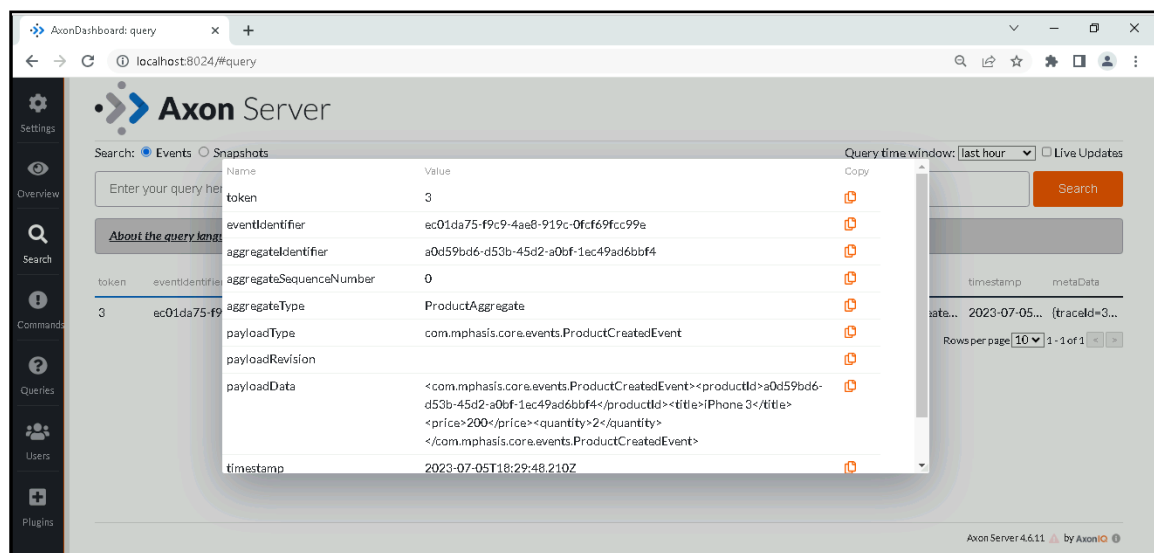
34. Using the /h2-console connect to the Products database and make sure that the product details are stored there as well.



35. Check the Event Store in the Axon server and make sure that the ProductCreatedEvent gets persisted,



36. Let's review the individual ProductCreatedEvent description in the Axon Server.



Problem Statement 8: Validate Request Body, Bean Validation in Product Microservice

When it comes to validating user input, Spring Boot provides strong support for this **common**, yet critical, task straight out of the box.

Although Spring Boot supports seamless integration with custom validators, the **de-facto standard for performing validation is Hibernate Validator**, the Bean Validation framework's reference implementation.

In this problem statement, we'll look at how to **validate domain objects in Spring Boot**.

Technology stack:

- Spring Boot Starter Validation

Steps for Spring Boot Starter Validation:

1. Refer the **ProductService** updated in the problem statement – 7.
2. Ensure the Spring Boot Starter Validation is added to ProductService/pom.xml file.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

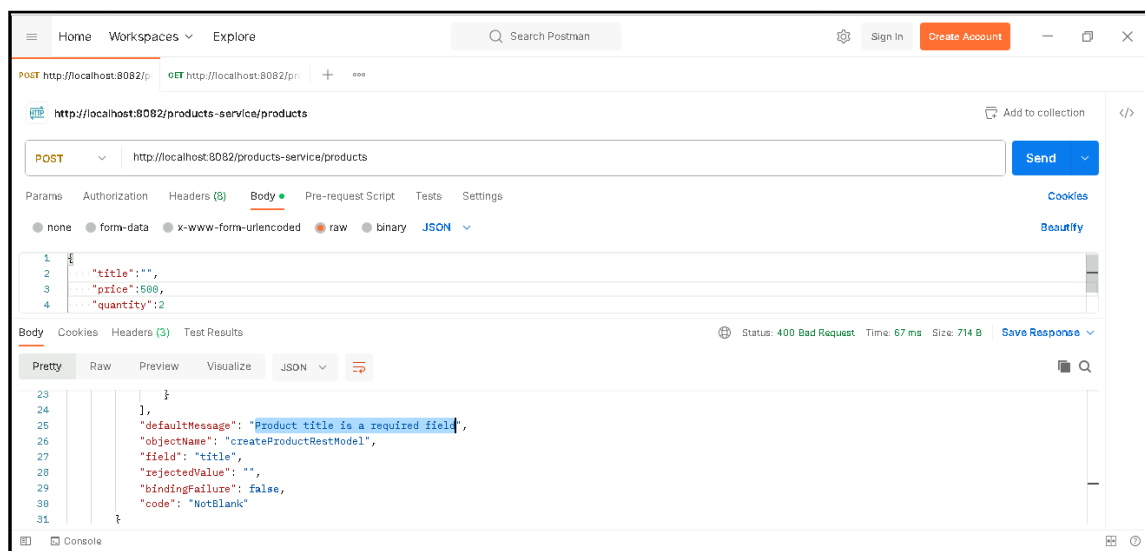
3. Add the below error properties to application.properties:

```
application.properties
1
2 server.port=0
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4 spring.application.name=products-service
5 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7 spring.datasource.url=jdbc:h2:mem:mphasisdb
8 spring.datasource.driver-class-name=org.h2.Driver
9 spring.datasource.username=sa
10 spring.datasource.password=password
11 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
12
13 #Accessing the H2 Console
14 spring.h2.console.enabled=true
15 spring.h2.console.path=/h2-console
16 spring.h2.console.settings.web-allow-others=true
17
18 server.error.include-message=always
19 server.error.include-binding-errors=always
20
```

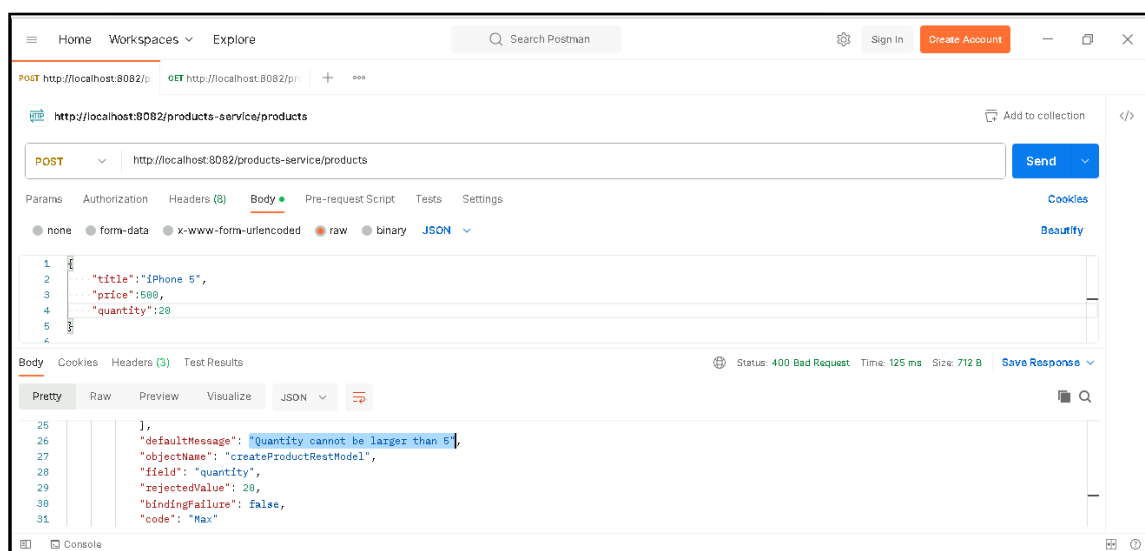

4. Apply the common validation annotation on the **CreateProductRestModel** class.
 - Use **@NotBlank** to say that a title field must not be the empty string.
 - Use **@Min** to say that the price is a numerical field is only valid when its value is above 1.
 - Use **@Min** and **@Max** to say that the quantity is a numerical field is only valid when its value is above 1 and below 5.
5. Adding the **@Valid** annotation will trigger validation of the request body on ProductCommandController handler methods.

Run and make it work:

6. Run the Axon Server using Docker command.
7. Start the Discovery Server (Eureka Server), Product Service, and ApiGateway.
8. Try sending a POST request with title as blank.



9. Try sending a POST request with Quantity greater than 5.



Problem Statement 9: Apply Command Validation using Message Dispatch Interceptor

Message Dispatch Interceptor is invoked when a message is dispatch from Command Gateway to Command Bus. We can create a Message Dispatch Interceptor to intercept immediately where they are dispatched on a Command Bus. You can use Message Dispatch Interceptor to perform additional logging, command validation, change a command message by adding META-DATA, and block the command by throwing an Exception.

Steps for implementing MessageDispatchInterceptor:

1. Refer the **ProductService** updated in the problem statement – 8.
2. Create a new `com.mphasis.command.interceptor` package.
3. Create a new class `CreateProductCommandInterceptor` which implements **MessageDispatchInterceptor** and override the **handle** method.
4. Let apply similar validation on the `CreateProductCommand` object used in `ProductAggregate` class previously.
 - Price cannot be less than or equal to zero.
 - Title cannot be empty.
5. Register the `CreateProductCommandInterceptor` in the `Application` class.

```
@Autowired
public void registerCreateProductCommandInterceptor(
    ApplicationContext context, CommandBus commandBus) {

    commandBus.registerDispatchInterceptor(context.getBean(CreateProductCommandInterceptor.class));
}
```

Run and make it work:

6. Let's comment the `@NotBlank` annotation for demonstration purpose.

```
@Data
public class CreateProductRestModel {

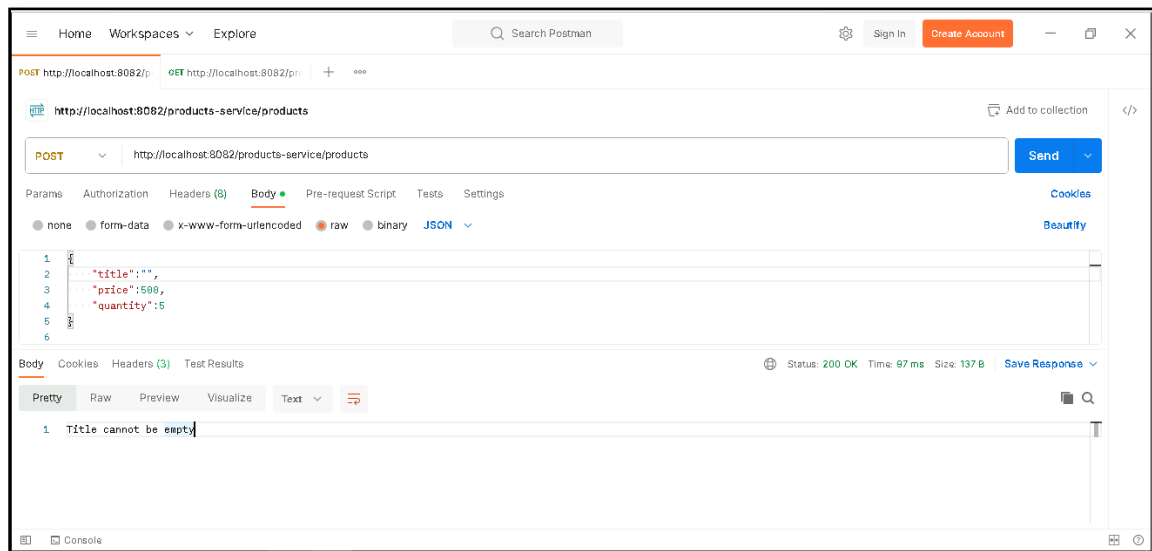
    // @NotBlank(message = "Product title is a required field")
    private String title;

    @Min(value=1, message = "Price cannot be lower than 1")
    private BigDecimal price;

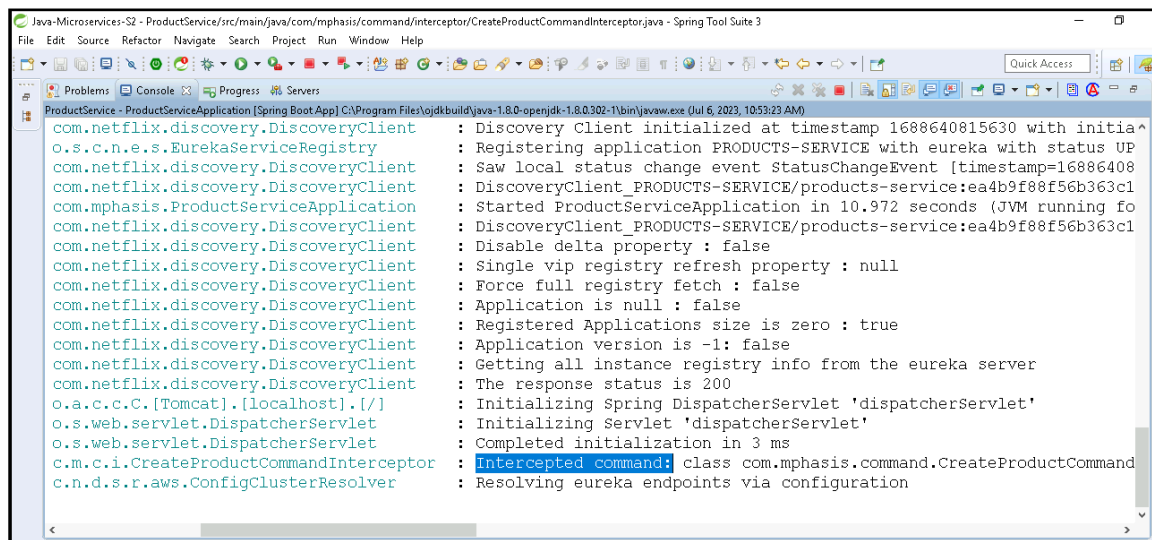
    @Min(value=1, message = "Quantity cannot be lower than 1")
    @Max(value=5, message = "Quantity cannot be larger than 5")
    private Integer quantity;
}
```

7. Run the Axon Server using Docker command.
8. Start the Discovery Server (Eureka Server), Product Service, and ApiGateway.

9. Send a POST request to Create Product with title as Empty.



10. Let's review the logs in the Console.



11. Let's uncomment the `@NotBlank` annotation.

Problem Statement 10: Set Based Consistency Validation in CQRS and Event Sourcing Application

A very common question asked by developers:

- How to check if record already exists in a database table?
- How to check if Product already exists?
- As the communication between Command API and Query API is via Messaging Architecture.
- How can Command API quickly check the record already exists for the Persistent Event in the Event Store?
- How can I validate a uniqueness constraint throughout the entire application using **CQRS** and **Event Sourcing**, while dealing with eventual consistency?

This issue is not related to Axon Server, but rather to the CQRS design pattern.

In this problem statement, we will discuss a solution that you can use if you are developing your application with the Axon Framework.

Steps for implementing Set Based Consistency Validation:

1. Refer the **ProductService** updated in the problem statement – 9.
2. Create a new `@Component` class called `ProductLookupEventsHandler` inside `com.mphasis.command` package and should be annotated with `@ProcessingGroup("product-group")`.
3. Create a new JPA Repository called `ProductLookupRepository` inside `com.mphasis.core.data` package and inject it into `ProductLookupEventsHandler` using constructor-based dependency injection.
4. Add a `find` method in `ProductLookupRepository` interface:

```
ProductLookupEntity findByProductIdOrTitle(String productId, String title);
```

5. The `ProductLookupEventsHandler` class should have one `@EventHandler` method that handles the `ProductCreatedEvent` and persists product lookup details into the "read" database.
6. To persist lookup details into the database, create a new JPA Entity class called `ProductLookupEntity` in `com.mphasis.core.data` package. Annotate the `ProductLookupEntity` class with:

```
@Entity
@Table(name = "productlookup")
@Data
@NoArgsConstructor
@AllArgsConstructor
```

and make the `ProductLookupEntity` class have the following fields:

```
@Id
public String productId;
@Column(unique = true)
private String title;
```

7. How do we query this lookup table before the command handler processes the command?
8. We use `Message Dispatch Interceptor`, which we have already created. The command will be intercepted by the `Message Dispatch Interceptor` before it is handled by the command handler method. It will use the JPA repository to query the lookup table and if the record already exists, the command will be blocked.

9. Update the CreateProductCommandInterceptor class and inject the ProductLookupRepository using constructor-based dependency injection.
10. Let's remove the if conditions of Price & Title from the CreateProductCommandInterceptor class.
11. In the handle method, check whether the ProductLookupEntity exists using the productId and title.
12. If exists, throw IllegalStateException with the below message:

```
throw new IllegalStateException(  
    String.format("Product with productId %s or title %s already exist",  
        createProductCommand.getProductId(),  
        createProductCommand.getTitle()  
    );
```

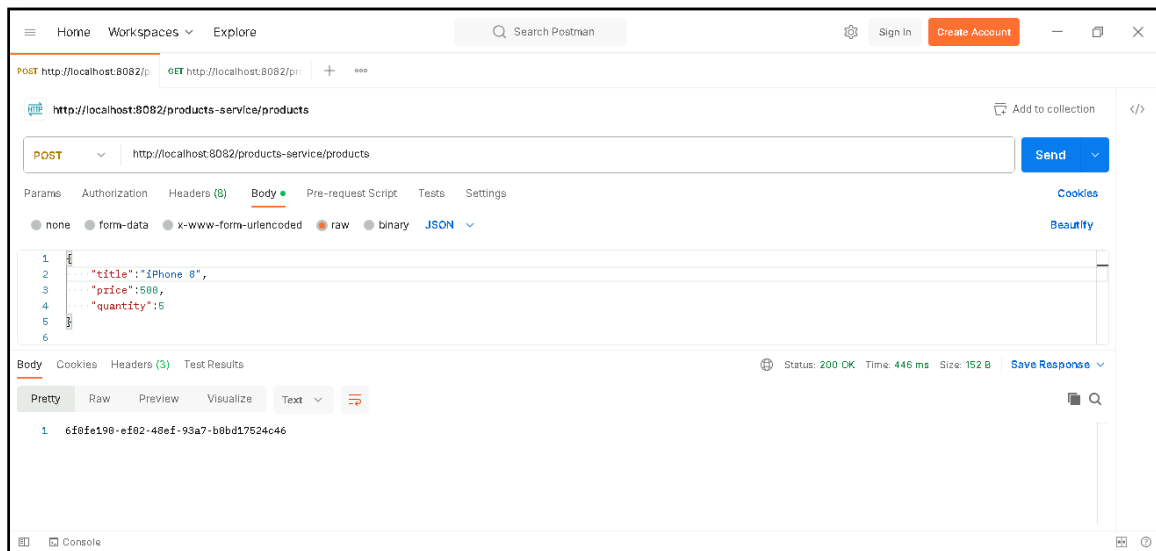
13. Verify the CreateProductCommandInterceptor is registered in the Application class.
14. Add the **processing-group** property inside the application.properties file:

```
application.properties  
2 server.port=0  
3 eureka.client.service-url.defaultZone=http://localhost:8761/eureka  
4 spring.application.name=products-service  
5 eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}  
6  
7 spring.datasource.url=jdbc:h2:mem:mphasisdb  
8 spring.datasource.driver-class-name=org.h2.Driver  
9 spring.datasource.username=sa  
10 spring.datasource.password=password  
11 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
12  
13 #Accessing the H2 Console  
14 spring.h2.console.enabled=true  
15 spring.h2.console.path=/h2-console  
16 spring.h2.console.settings.web-allow-others=true  
17  
18 server.error.include-message=always  
19 server.error.include-binding-errors=always  
20  
21 axon.eventhandling.processors.product-group.mode=subscribing
```

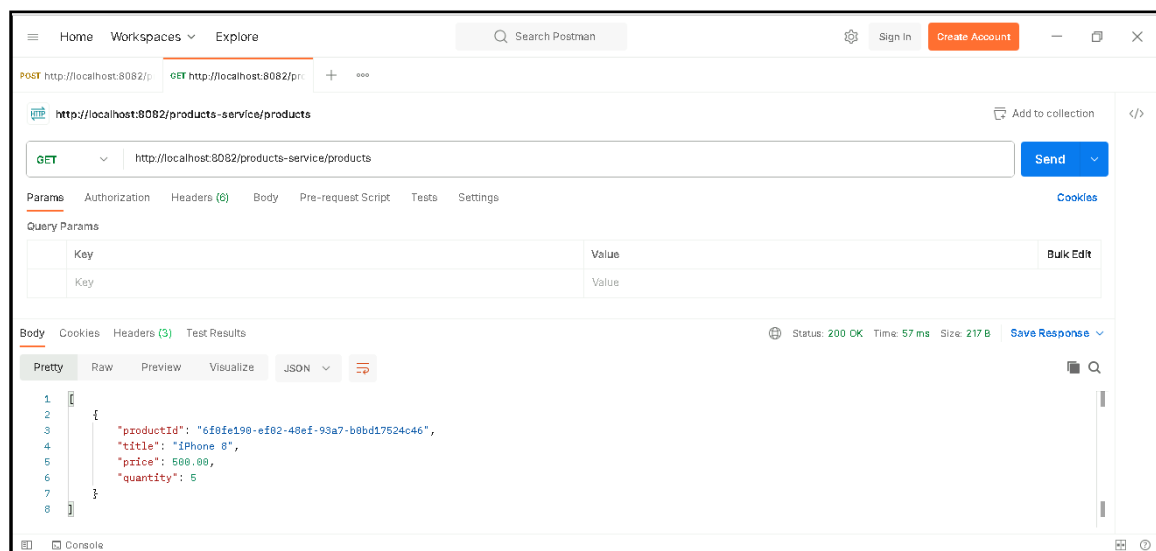
Run and make it work:

15. Run the Axon Server using Docker command.
16. Start the Discovery Server (Eureka Server), Product Service, and ApiGateway.

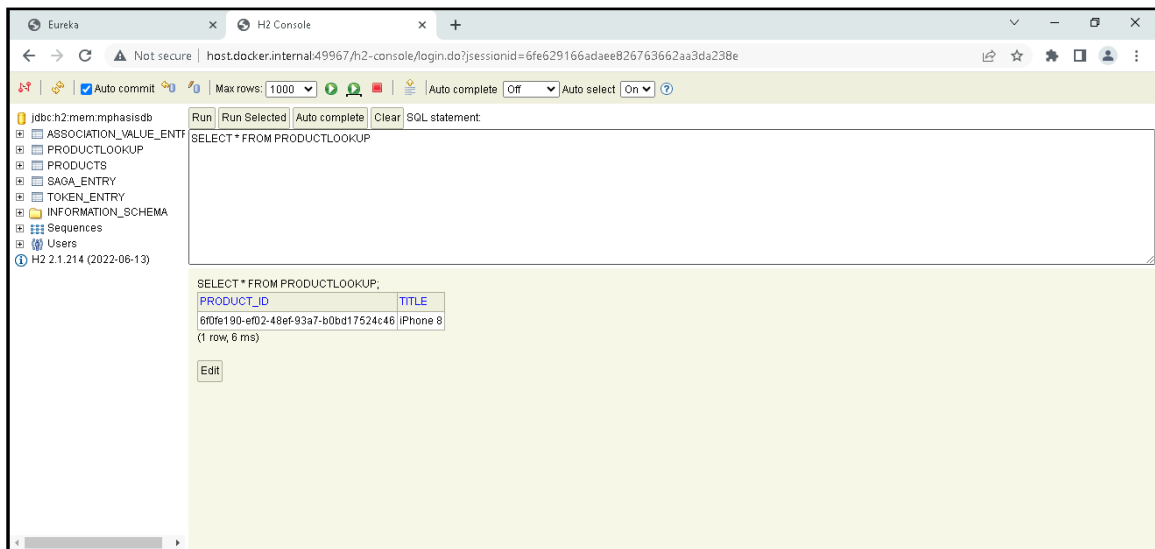
17. Send a POST request to Create Product.



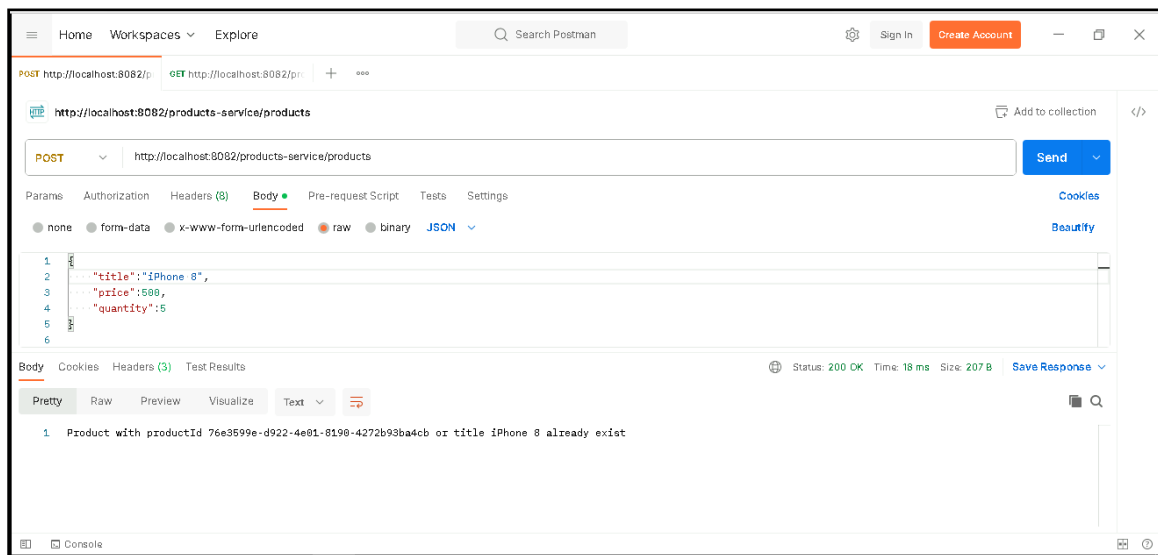
18. Send a GET request to Query the Products.



19. Let's review the ProductLookup table data.



20. Send a POST request to Create Product with Same JSON.



Problem Statement 11: Handling Errors and Rollback Transaction with Axon

In this problem statement, we will discuss how to handle an error in the event handling method. How to rollback changes made in various event handler methods.

This is very helpful when you have multiple error handler methods, and you want to undo changes made in all error handlers that are in the **same processing group**.

Steps for implementing Handling Errors and Rollback Transaction:

1. Refer the **ProductService** updated in the problem statement – 10.
2. Create a centralized ProductServiceErrorHandler class in the com.mphasis.core.errorhandling package and should be annotated with **@ControllerAdvice** annotation.
3. In this case, we will use **@ExceptionHandler** to handle the IllegalStateException and other Exceptions, which will return the ResponseEntity set with a ErrorMessage object (current date and message) and the status code INTERNAL_SERVER_ERROR.
4. Create a ErrorMessage class in the com.mphasis.core.errorhandling package and annotated with:

@Data

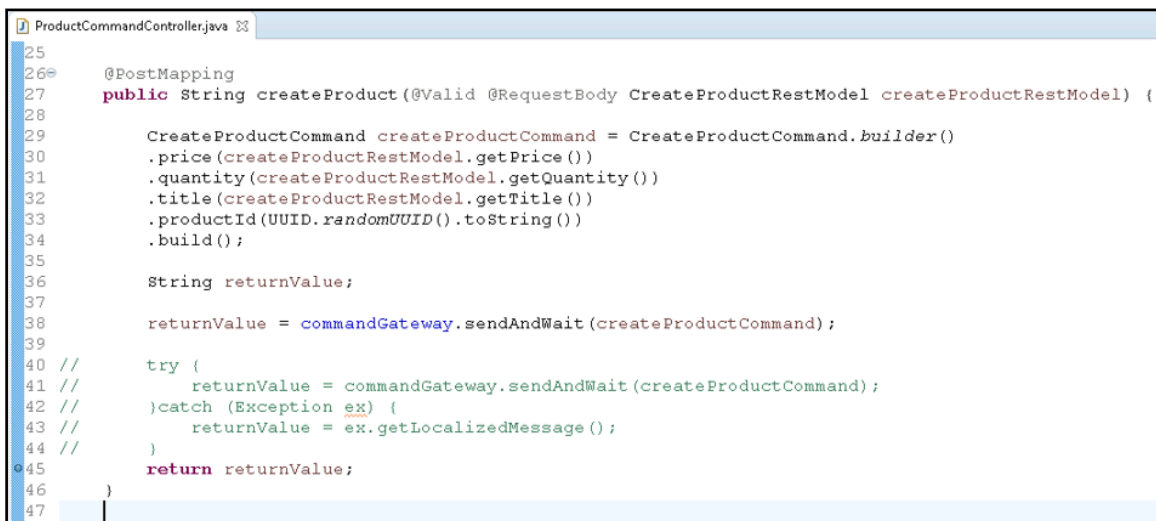
@AllArgsConstructor

And have the following fields:

private final java.util.Date timestamp;

private final String message;

5. Comment the try catch block in the ProductCommandController class.

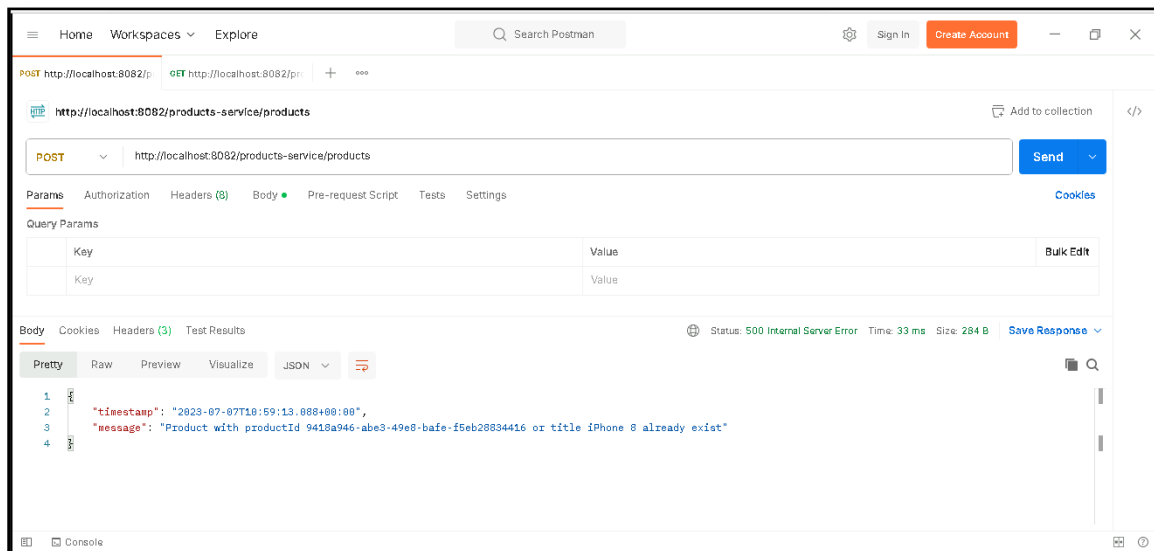


```
25
26 @PostMapping
27 public String createProduct(@Valid @RequestBody CreateProductRestModel createProductRestModel) {
28
29     CreateProductCommand createProductCommand = CreateProductCommand.builder()
30         .price(createProductRestModel.getPrice())
31         .quantity(createProductRestModel.getQuantity())
32         .title(createProductRestModel.getTitle())
33         .productId(UUID.randomUUID().toString())
34         .build();
35
36     String returnValue;
37
38     returnValue = commandGateway.sendAndWait(createProductCommand);
39
40     // try {
41     //     returnValue = commandGateway.sendAndWait(createProductCommand);
42     // } catch (Exception ex) {
43     //     returnValue = ex.getLocalizedMessage();
44     // }
45     return returnValue;
46 }
47
```

Run and make it work:

6. Run the Axon Server using Docker command.
7. Start the Discovery Server (Eureka Server), Product Service, and ApiGateway.

- Send a POST request to Create Product – twice.



- Further going we will handle an Exception that is thrown by Command Handler method in the Aggregate class.

```
@CommandHandler
public ProductAggregate(CreateProductCommand createProductCommand) throws Exception {
    // Validate Create Product Command

    if (createProductCommand.getPrice().compareTo(BigDecimal.ZERO) <= 0) {
        throw new IllegalArgumentException("Price cannot be less or equal than zero");
    }

    if (createProductCommand.getTitle() == null || createProductCommand.getTitle().isEmpty()) {
        throw new IllegalArgumentException("Title cannot be empty");
    }

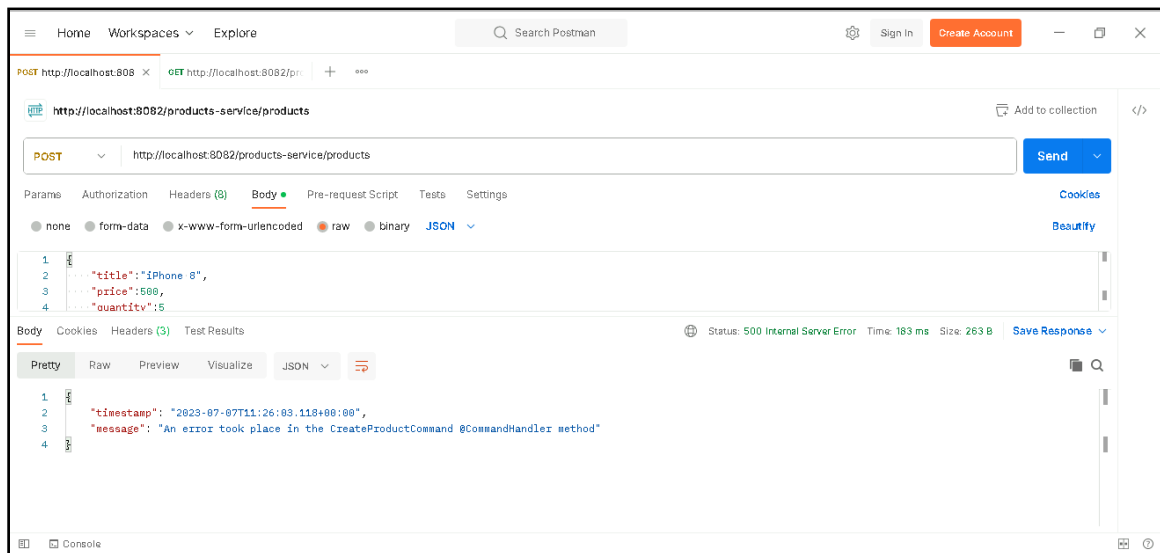
    ProductCreatedEvent productCreatedEvent = new ProductCreatedEvent();
    BeanUtils.copyProperties(createProductCommand, productCreatedEvent);

    AggregateLifecycle.apply(productCreatedEvent);

    if(true)
        throw new Exception("An error took place in the CreateProductCommand @CommandHandler method");
}
```

- When an Error is thrown from the command handler/query handler method then the Axon Framework wrap this Error into **CommandExecutionException/QueryExecutionException**.
- In the ProductServiceErrorHandler, let's add an **@ExceptionHandler** to handle the CommandExecutionException, which will return the ResponseEntity set with a ErrorMessage object (current date and message) and the status code INTERNAL_SERVER_ERROR.
- Restart the Discovery Server (Eureka Server), Product Service, and ApiGateway.

13. Send a POST request to Create Product.



14. Further going the exception can occur in the Event Handler method that handle the ProductCreatedEvent in the ProductEventsHandler class.
15. There are different ways to handle the exception either by using try-catch block or use `@ExceptionHandler` annotation.
16. In this case, we will use **`@ExceptionHandler`** to handle the `IllegalStateException` and other Exceptions in ProductEventsHandler class, rethrow the exception.
17. If you look to roll back the transaction and do the database changes made by EventHandler method, you need to either handle the Exception in your EventHandler method or handle it and rethrow it again. So that it can propagate up the flow.
18. This Exception can be handled by another ExceptionHandler and be propagated further up the flow which will eventually rollback the entire transaction and none of the changes to the database or Event Store will be made.
19. This is possible if your **processing group** is configured to use **subscribing event processors**.
20. Let create a class ProductsServiceEventsErrorHandler which implements **ListenerInvocationErrorHandler** interface and overrides the **onError** method. Here we will rethrow the exception.
21. Register the ProductsServiceEventsErrorHandler in the Application class.

```
@Autowired
public void configure(EventProcessingConfigurer configurer) {
    configurer.registerListenerInvocationErrorHandler("product-group",
        config -> new ProductsServiceEventsErrorHandler());
}
```

22. Cut the below two lines from the ProductAggregate:

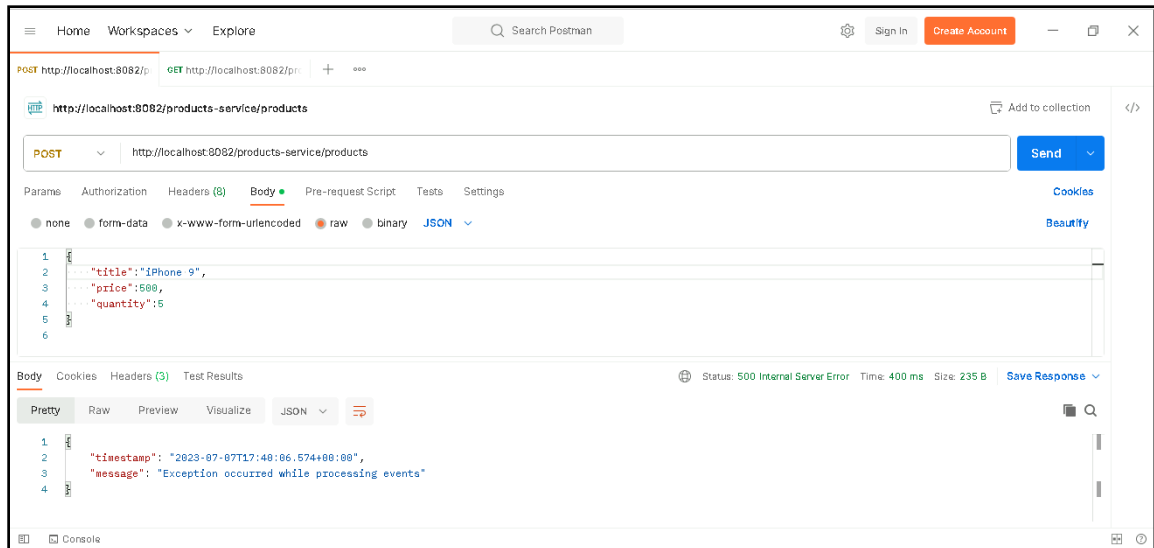
```
if (true)
    throw new Exception("An error took place in the CreateProductCommand @CommandHandler method");
```

23. Paste it in ProductEventsHandler and modify the message:

```
ProductEventsHandler.java
26         throw exception;
27     }
28
29     @ExceptionHandler(resultType = IllegalArgumentException.class)
30     public void handle(IllegalArgumentException exception) {
31         // log error message
32     }
33
34     @EventHandler
35     public void on(ProductCreatedEvent event) throws Exception {
36
37         ProductEntity productEntity = new ProductEntity();
38         BeanUtils.copyProperties(event, productEntity);
39         try {
40             productRepository.save(productEntity);
41         } catch (IllegalArgumentException ex) {
42             ex.printStackTrace();
43         }
44
45         if(true)
46             throw new Exception("Forcing exception in the Event Handler class");
47     }
48 }
49 }
```

24. Restart the Discovery Server (Eureka Server), Product Service, and ApiGateway.

25. Send a POST request to Create Product.



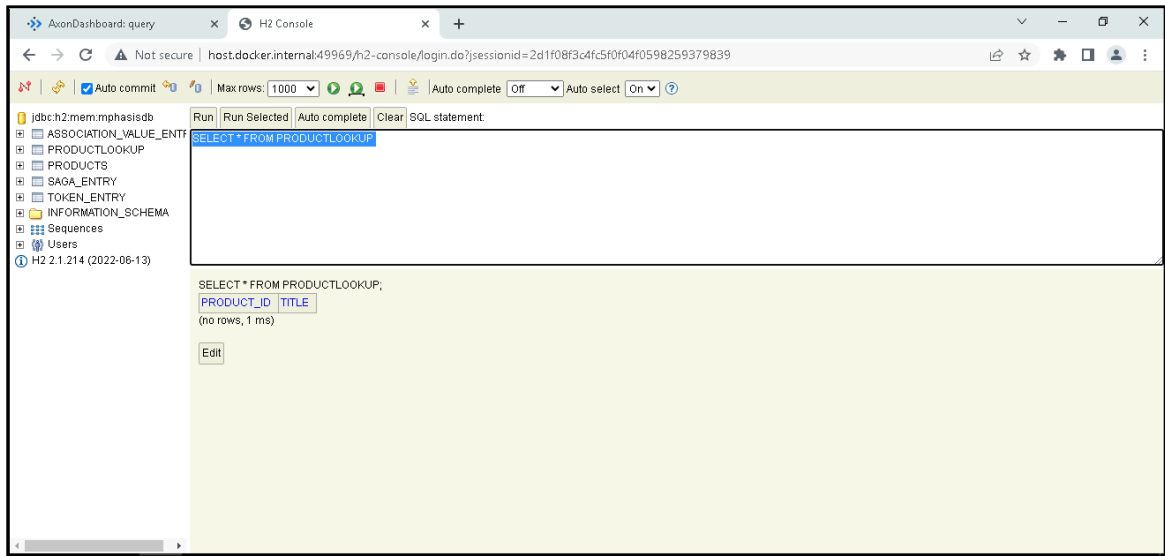
Postman interface showing a POST request to `http://localhost:8082/products-service/products`. The request body is a JSON object:

```
{
  "title": "iPhone 9",
  "price": 500,
  "quantity": 5
}
```

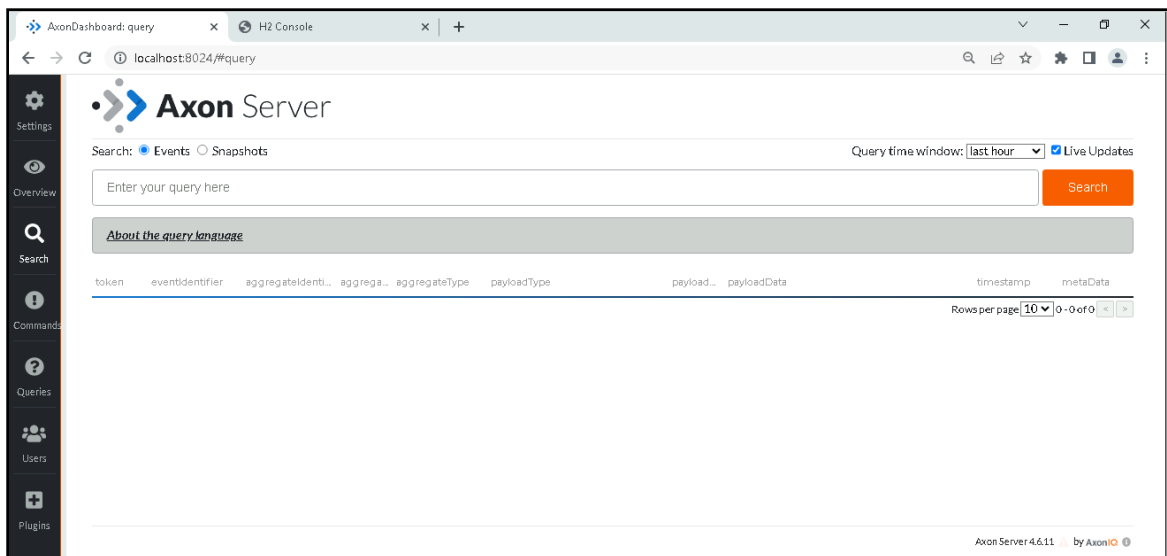
The response is a 500 Internal Server Error with the following JSON body:

```
{
  "timestamp": "2023-07-07T17:40:06.574+00:00",
  "message": "Exception occurred while processing events"
}
```

26. Let's go to browser and query the ProductLookup table. Still, we don't have any record. That's good the transaction was rollback, and the Entity did not persist.



27. Now, let's go and lookup in Event Store and see if we have records here.



Problem Statement 12: Implementing the CQRS & Event Sourcing Design Pattern in Orders Microservice

Similarly, to the Product Microservice, we will create the Orders Microservices with CQRS and Event Sourcing through the Axon Framework.

Technology stack:

- Spring Web
- Spring Data JPA
- H2 Database
- Spring Cloud Eureka Client
- Lombok
- Axon Spring Boot Starter
- Google Guava
- Spring Boot Starter Validation

Steps for implementing CQRS and Event Sourcing using Axon Server:

1. Create a new Spring Boot Project:
 - Create a new Spring Boot project using either Spring Initializer Tool(<https://start.spring.io>) or using your development environment.
 - Call this new project "OrdersService".
2. Add the Spring Web, Spring Data JPA, H2 Database, Spring Cloud Eureka Client, Lombok, Axon Spring Boot Starter, Google Guava, and Spring Boot Starter Validation dependencies in pom.xml.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.axonframework</groupId>
  <artifactId>axon-spring-boot-starter</artifactId>
  <version>4.5.8</version>
</dependency>

<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>30.1-jre</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

- Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
- Create a new OrdersCommandController class with a request mapping "/orders" and one method that accepts the HTTP POST request.
- The method that accepts the HTTP Post request, should accept the OrderCreateRest JSON payload as a request body.

```
{
  "productId":"f241af45-4854-43f4-95bc-ab54da338a29",
  "quantity":1,
  "addressId":"afbb5881-a872-4d13-993c-faeb8350eea5"
}
```

- Apply the common validation annotation on the OrderCreateRest class:
Use @NotBlank to say that a productId field must not be the empty string.
Use @Min and @Max to say that the quantity is a numerical field is only valid when its value is above 1 and below 5.
Use @NotBlank to say that an addressId field must not be the empty string.
- Trigger the validation of the request body on ProductCommandController handler methods.
- This controller class should use the **Axon's CommandGateway** and publish the CreateOrderCommand.
- The CreateOrderCommand annotated with @Data, @Builder and should have the following fields:

```
public final String orderId;
private final String userId;
private final String productId;
private final int quantity;
private final String addressId;
private final OrderStatus orderStatus;
```

Where:

- orderId - is a randomly generated value. For example, `UUID.randomUUID().toString()` and annotated with **@TargetAggregateIdentifier**.
- userId - is a static hard-coded value: `27b95829-4f3f-4ddf-8983-151ba010e35b`. At this moment there is no user registration, authentication, and authorization implemented, so we will hard code the value of userId for now.
- orderStatus - is an Enum with the following content:

```
public enum OrderStatus {  
    CREATED, APPROVED, REJECTED  
}
```
- After sending the `CreateOrderCommand`, use **Axon's QueryGateway** instance to invoke **query** method which is used publish the `FindOrderQuery` with orderId and use `ResponseTypes.instanceOf(OrderSummary)` to get the `OrderSummary` and return as a response body.
- The `OrderSummary` annotated with `@Value` and should have the following fields.

```
public final String orderId;  
private final OrderStatus orderStatus;  
private final String message
```

10. Create a new class called `OrderAggregate` and make it handle the `CreateOrderCommand` using **@CommandHandler** and publish the `OrderCreatedEvent`.

11. The `OrderCreatedEvent` class annotated with `@Data` and should have the following fields:

```
private String orderId;  
private String productId;  
private String userId;  
private int quantity;  
private String addressId;  
private OrderStatus orderStatus;
```

12. The `OrderAggregate` class should also have an **@EventSourcingHandler** method that sets values for all fields in the `OrderAggregate`.

13. Create a new `@Component` class called `OrderEventsHandler` annotated with **@ProcessingGroup("order-group")** inside `com.mphasis.query` package.

14. Create a new JPA Repository called `OrdersRepository` inside `com.mphasis.core.data` package and inject it into `OrderEventsHandler` using constructor-based dependency injection.

15. Add a find method in `OrdersRepository` interface:

```
OrderEntity findById(String orderId);
```

16. The `OrderEventHandler` class should have one **@EventHandler** method that handles the `OrderCreatedEvent` and persists order details into the "read" database.

17. To persist order details into the database, create a new JPA Entity class called OrderEntity inside com.mphasis.core.data package. Annotate the OrderEntity class with:

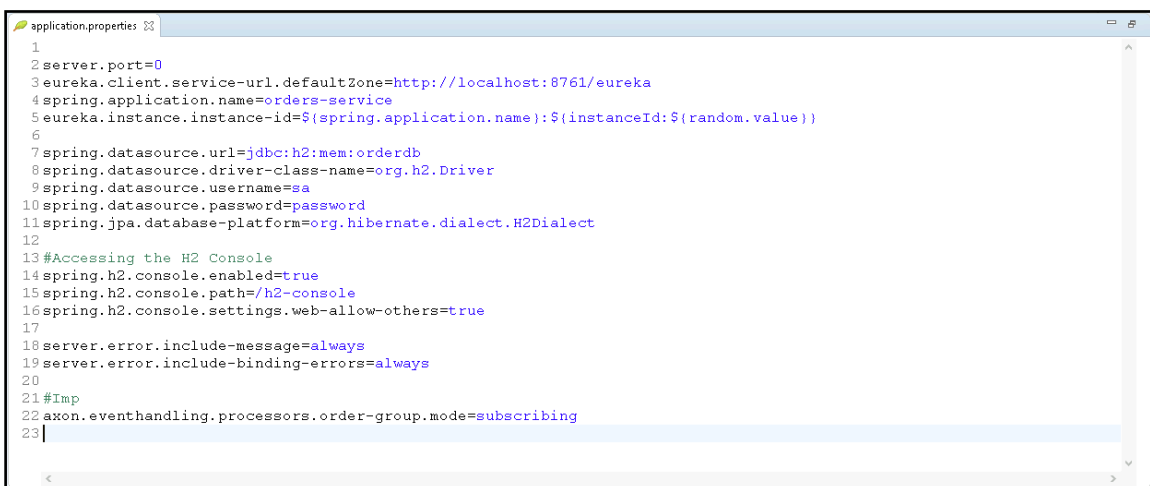
```
@Data
@Entity
@Table(name = "orders")
```

and make the OrderEntity class have the following fields:

```
@Id
@Column(unique = true)
public String orderId;
private String productId;
private String userId;
private int quantity;
private String addressId;
```

```
@Enumerated(EnumType.STRING)
private OrderStatus orderStatus;
```

18. Create a new @Component class called OrderQueriesHandler.
19. Create a new JPA Repository called OrdersRepository and inject it into OrderQueriesHandler using constructor-based dependency injection.
20. The OrderQueriesHandler class should have one @QueryHandler method that handles the FindOrderQuery and fetch the order summary from the "read" database.
21. Register with Eureka: Make OrdersService microservice register with Eureka as a Client.
22. Since each Microservice should store data in its own database, configure this Microservice to work with a new database called "orderdb".
23. Add the below properties to application.properties:

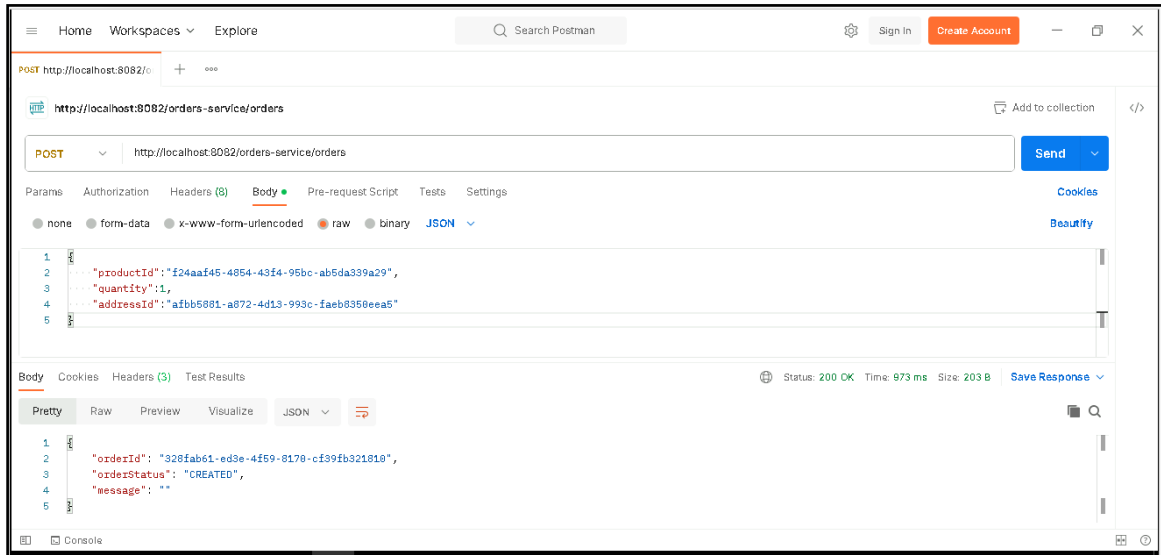
A screenshot of a code editor showing the contents of an application.properties file. The file contains 23 lines of configuration for a Spring Boot application, including server port, Eureka client settings, database connection details for H2, and Axon event handling configuration.

```
1
2server.port=0
3eureka.client.service-url.defaultZone=http://localhost:8761/eureka
4spring.application.name=orders-service
5eureka.instance.instance-id=${spring.application.name}:${instanceId:${random.value}}
6
7spring.datasource.url=jdbc:h2:mem:orderdb
8spring.datasource.driver-class-name=org.h2.Driver
9spring.datasource.username=sa
10spring.datasource.password=password
11spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
12
13#Accessing the H2 Console
14spring.h2.console.enabled=true
15spring.h2.console.path=/h2-console
16spring.h2.console.settings.web-allow-others=true
17
18server.error.include-message=always
19server.error.include-binding-errors=always
20
21#Imp
22axon.eventhandling.processors.order-group.mode=subscribing
23|
```


Run and make it work:

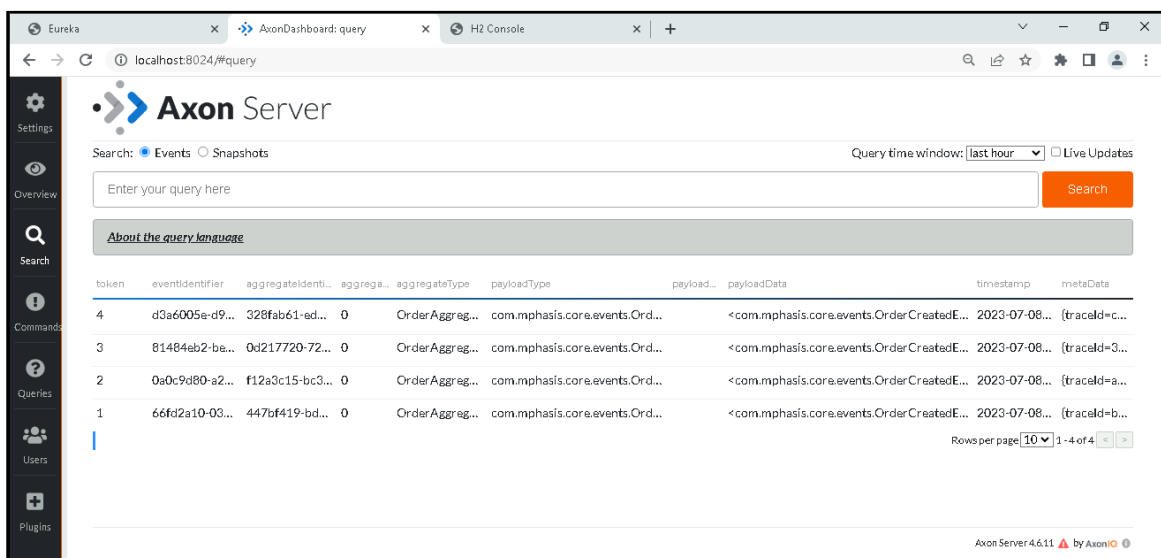
24. Run your OrdersService microservice and make it work. Send a request with the following JSON and make sure it gets successfully stored in the read database. Since you have annotated the OrderEntity class with @Table(name = "orders"), the database table name will be "orders".

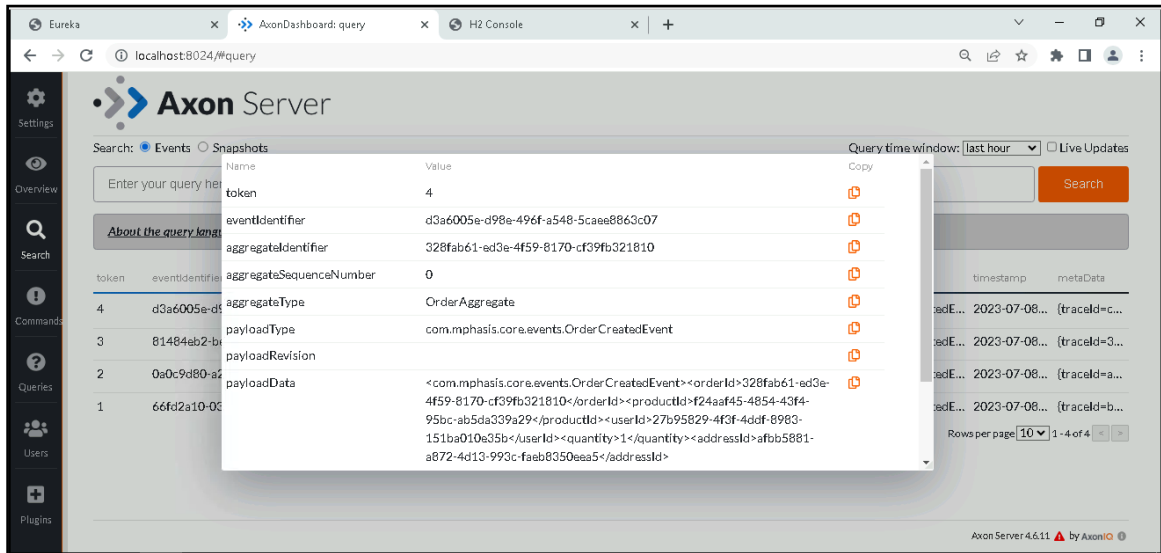
```
{
  "productId": "f241af45-4854-43f4-95bc-ab54da338a29",
  "quantity": 1,
  "addressId": "afbb5881-a872-4d13-993c-faeb8350eea5"
}
```



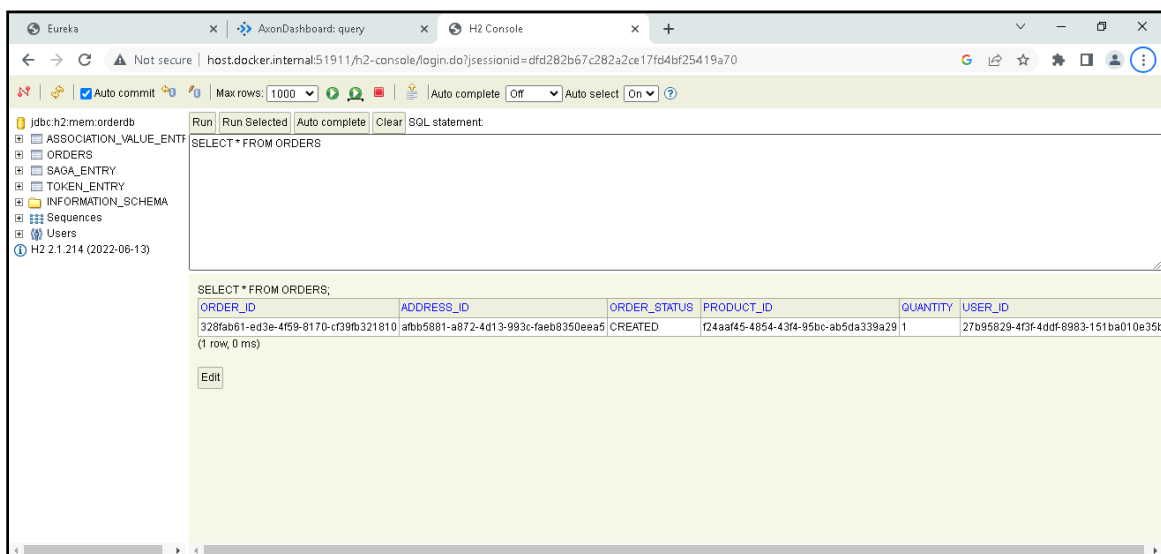
Verify results:

25. Check the Event Store in the Axon server and make sure that the OrderCreatedEvent gets persisted,





26. Using the /h2-console connect to the orderdb database and make sure that the order details are stored there as well.



Problem Statement 13: Orchestration based Saga – Reserve Product in Stock

In this problem statement, we will work on the create order flow. So, I will add the order saga into my orders microservice.

Now, the flow will begin when order aggregate, received the create order command, and publishes an order created event. This is the beginning of the flow, so I will make my saga class handle the order created event and use it as the beginning of the saga flow. The order saga will then publish reserve product command, and once processed by the products microservice, the saga will handle the product reserved event. Saga will continue the flow and publish the process payment command once the product has been reserved. And once the payment is processed, it'll be the saga component that will handle the payment processed event.

So, our saga class will be an event-handling component that will manage the create order flow by handling events, and publishing commands to complete the flow. And if one of the steps on the flow fails, it will be this Saga class that manages the flow of compensating operations to rollback changes done in this flow.

The following Saga Class Structure will be implemented:

```
@Saga
public class OrderSaga {

    @Autowired
    private transient CommandGateway commandGateway;

    @StartSaga
    @SagaEventHandler(associationProperty = "orderId")
    public void handle(OrderCreatedEvent orderCreatedEvent) {
        //
    }

    @SagaEventHandler(associationProperty = "productId")
    public void handle(ProductReservedEvent productReservedEvent) {
        //
    }

    @SagaEventHandler(associationProperty = "paymentId")
    public void handle(PaymentProcessedEvent paymentProcessedEvent) {
        //
    }

    @EndSaga
    @SagaEventHandler(associationProperty = "orderId")
    public void handle(OrderApprovedEvent orderApprovedEvent) {
        //
    }
}
```

Steps for implementing Orchestration based Saga:

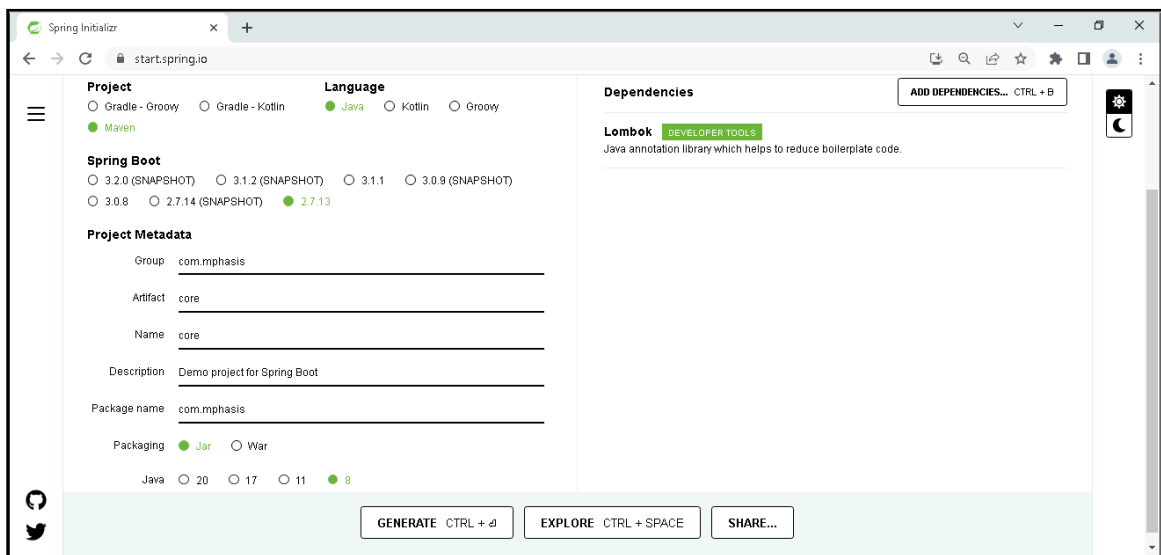
1. Refer the **ProductService** updated in the problem statement – 11.
2. Refer the **OrdersService** created in the problem statement – 12.
3. Create a new class OrderSaga annotated with **@Saga** in the com.mphasis.saga package. Also, Autowire the **Axon's CommandGateway** instance inside the OrderSaga class.
4. Start the Saga using **@StartSaga** annotation and create a handler for OrderCreatedEvent using **@SagaEventHandler** annotation with the **associatedProperty** "orderId".

- The OrderCreatedEvent class annotated with @Data, @NoArgsConstructor, @AllArgsConstructor and should have the following fields:

```
private String orderId;  
private String productId;  
private String userId;  
private int quantity;  
private String addressId;  
private OrderStatus orderStatus;
```

- Create a new Project for **Core** using the **Spring Initializr** (start.spring.io) with the Lombok starter.

Refer the below screenshots:



- Select the Lombok dependencies.
- Click on GENERATE button or CTRL + Enter to create the project structure.
- Let's import the Core maven project in STS.
- Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
- Add **axon-spring-boot-starter** dependency in pom.xml:

```
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <optional>true</optional>  
</dependency>  
  
<dependency>  
  <groupId>org.axonframework</groupId>  
  <artifactId>axon-spring-boot-starter</artifactId>  
  <version>4.5.8</version>  
</dependency>
```

12. Delete the build tag from the pom.xml file.
13. Delete the CoreApplication class and CoreApplicationTests class.
14. So now we can use this project as a dependency to another project.
15. Adding Core project as a dependency to OrdersService/pom.xml and ProductService/pom.xml files:

```
<dependency>
  <groupId>com.mphasis</groupId>
  <artifactId>core</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

16. Now you can fetch the classes available in core project.
17. Create a new ReserveProductCommand class in com.mphasis.core.commands package inside the Core project.
18. The ReserveProductCommand annotated with @Data, @Builder and should have the following fields:
private final String productId;
private final int quantity;
private final String orderId;
private final String userId;
Where:
productId – Annotated with **@TargetAggregateIdentifier** annotation.
19. ReserveProductCommand from the core module is now available to OrdersService.
20. Inside the OrderSaga - handler method of OrderCreatedEvent, we will create the instance of ReserveProductCommand and Publish it.
21. Now let's handle the ReserveProductCommand inside the ProductService – ProductAggregate class using the **@CommandHandler** annotation.
22. Now to check this command will be successful executed, we need to check if the current quantity of the Product is not less than the requested quantity.
23. Using LOGGER.info, let's log the orderId and productId on the console.
24. If there is enough quantity of this product in stock, then we will create and publish the ProductReservedEvent which will be handled by the OrderSaga class.
25. The ProductReservedEvent class annotated with @Data, @Builder and should have the following fields:
private final String productId;
private final int quantity;
private final String orderId;
private final String userId;
26. Let's go to our ProductService – ProductAggregate class and will publish the ProductReservedEvent inside the CommandHandler method.
27. Also create an **@EventSourcingHandler** method which is handling the ProductReservedEvent and updating the quantity (subtraction action).
28. To make our read database up to date with the changes that we have just made to the product quantity, we will need to handle the ProductReservedEvent and update the read database as well.

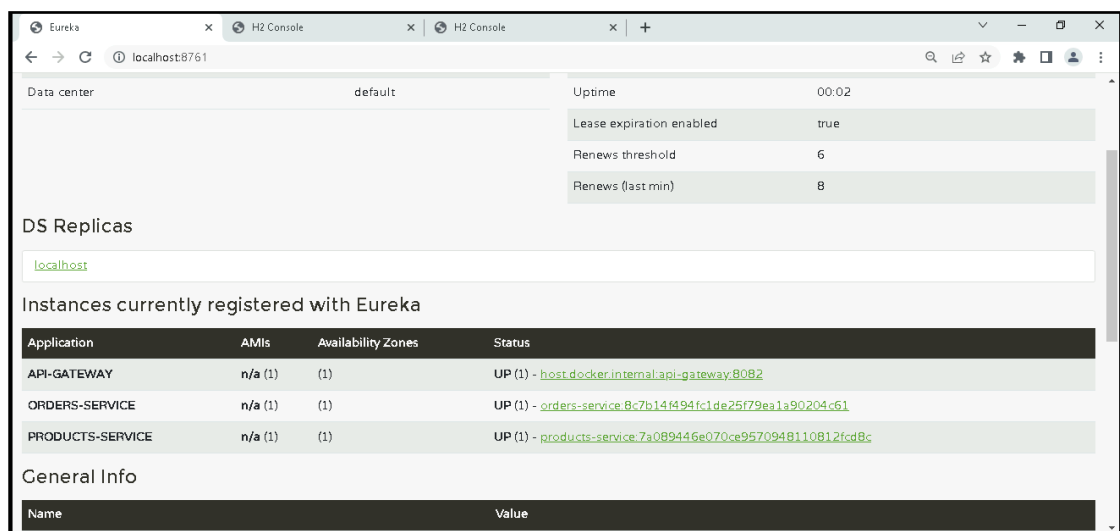
29. In ProductService – ProductEventsHandler class, we need to add one more **@EventHandler** method for the ProductReservedEvent and update the Products projection.
30. After updating, let's log the orderId and productId on the console.
31. Finally, we will Handle the ProductReservedEvent in OrdersService – OrderSaga, by creating a new handler method with the same **associationProperty** i.e., orderId.

```
@SagaEventHandler(associationProperty = "orderId")
public void handle(ProductReservedEvent productReservedEvent) {
    // Process user payment

    LOGGER.info("ProductReservedEvent is called for productId: " + productReservedEvent.getProductId() +
        " and orderId: " + productReservedEvent.getOrderId());
}
```

Run and make it work:

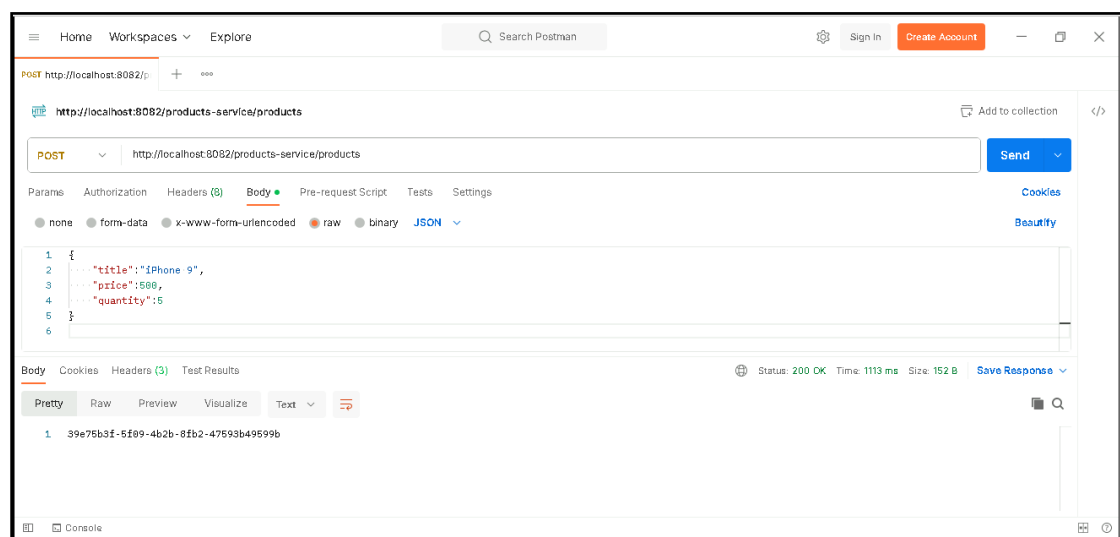
32. Run the Axon Server using Docker command.
33. Start the Discovery Server (Eureka Server), Product Service, Orders Service, and ApiGateway.
34. Verify the Eureka Server: <http://localhost:8761/>



The screenshot shows the Eureka Server UI. The top navigation bar includes 'Data center' (default), 'Uptime' (00:02), 'Lease expiration enabled' (true), 'Renews threshold' (6), and 'Renews (last min)' (8). Below this, the 'DS Replicas' section shows 'localhost'. The 'Instances currently registered with Eureka' section displays a table with columns: Application, AMIs, Availability Zones, and Status. The table lists three instances: API-GATEWAY, ORDERS-SERVICE, and PRODUCTS-SERVICE, all with status 'UP (1)'. The 'General Info' section is partially visible at the bottom.

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - host.docker.internal:api-gateway:8082
ORDERS-SERVICE	n/a (1)	(1)	UP (1) - orders-service:8c7b14f494fc1de25f79ea1a90204c61
PRODUCTS-SERVICE	n/a (1)	(1)	UP (1) - products-service:7a089446e070ce9570948110812fcd8c

35. Send a POST request to Create Product.



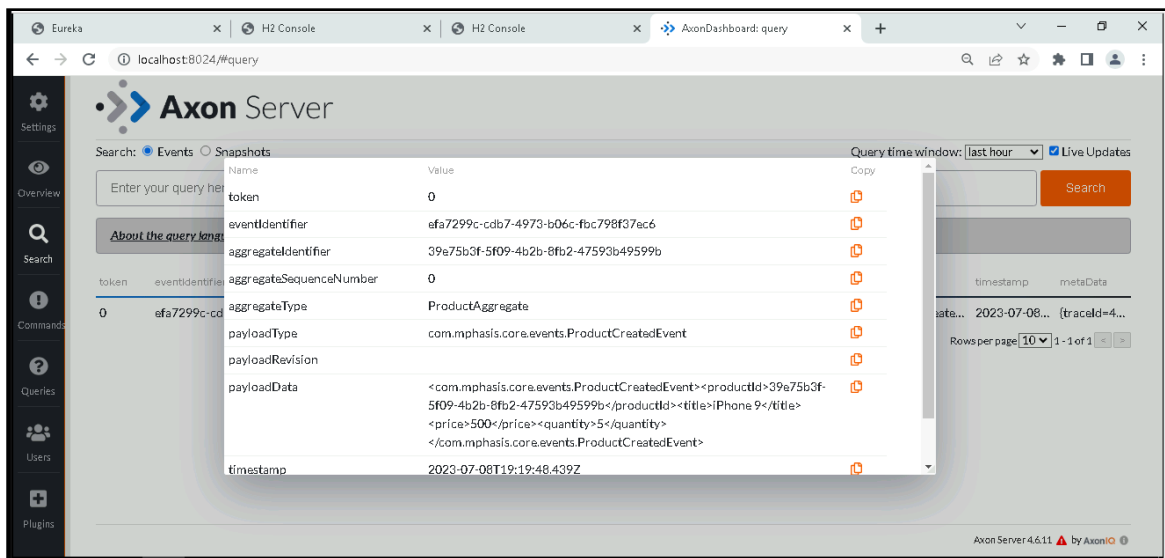
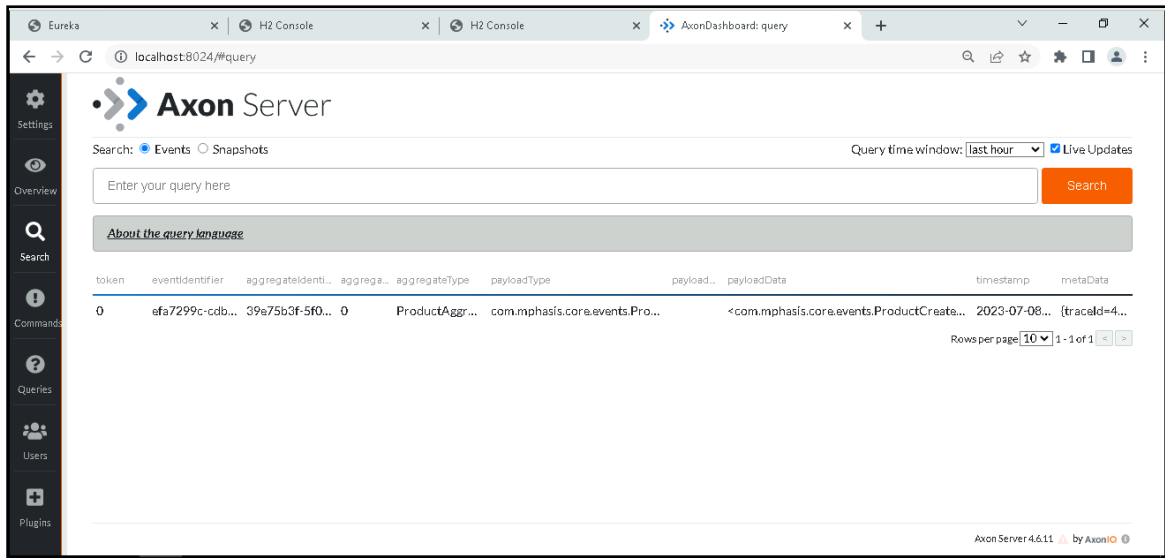
The screenshot shows a REST client interface. The URL bar displays 'http://localhost:8082/products-service/products'. The request method is 'POST'. The request body is a JSON object:

```
{  "title": "iPhone 9",  "price": 568,  "quantity": 5}
```

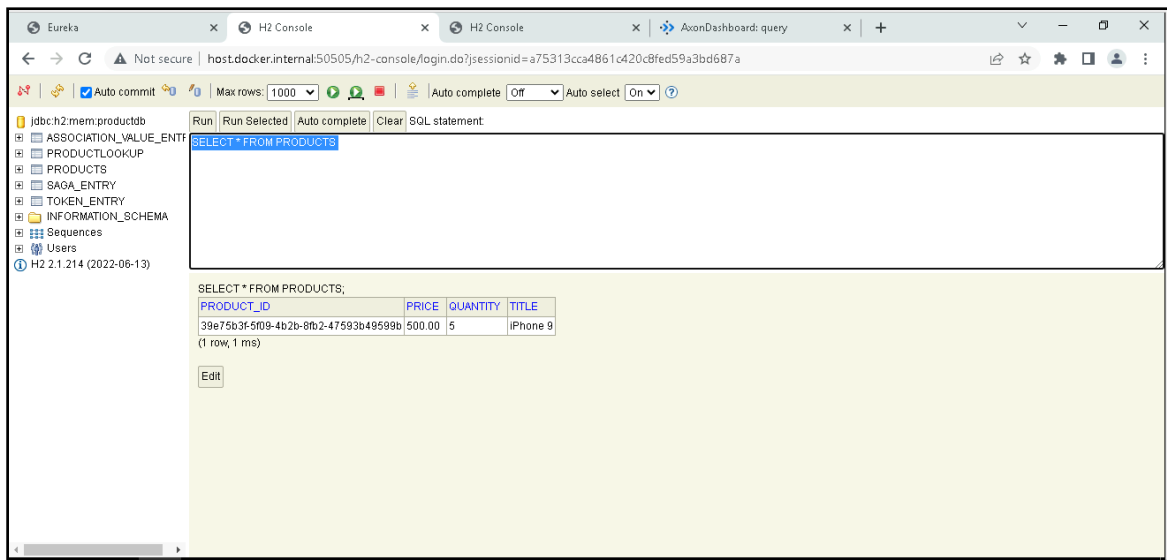
. The response status is '200 OK' with a time of '1113 ms' and a size of '152 B'. The response body is a long alphanumeric string: '39e75b3f-5f89-4b2b-8fb2-47593b49599b'.

Verify results:

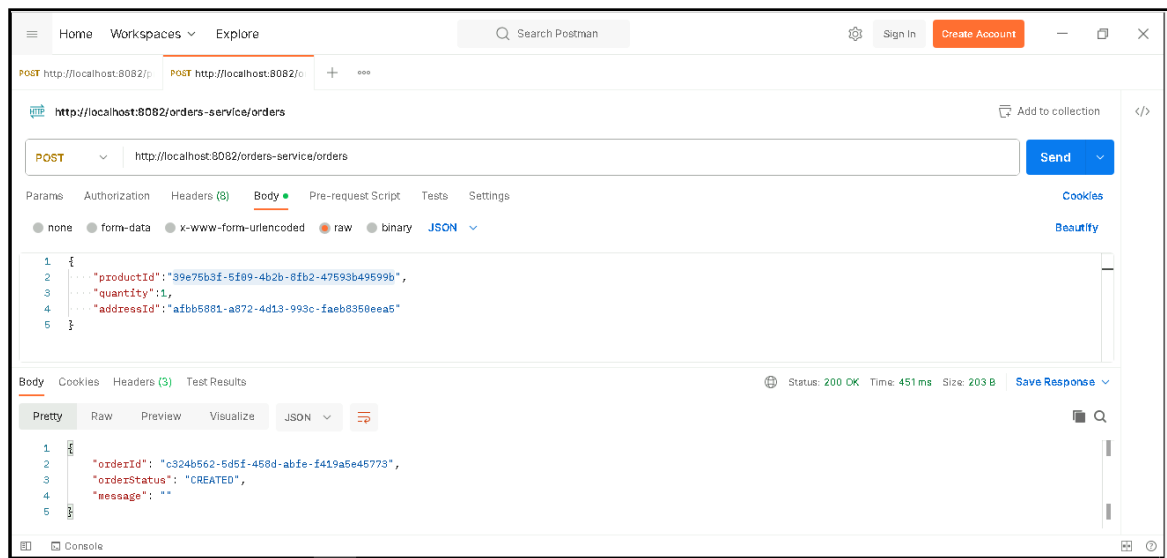
36. Check the Event Store in the Axon server and make sure that the ProductCreatedEvent gets persisted,



37. Using the /h2-console connect to the productdb database and make sure that the product details are stored there as well.

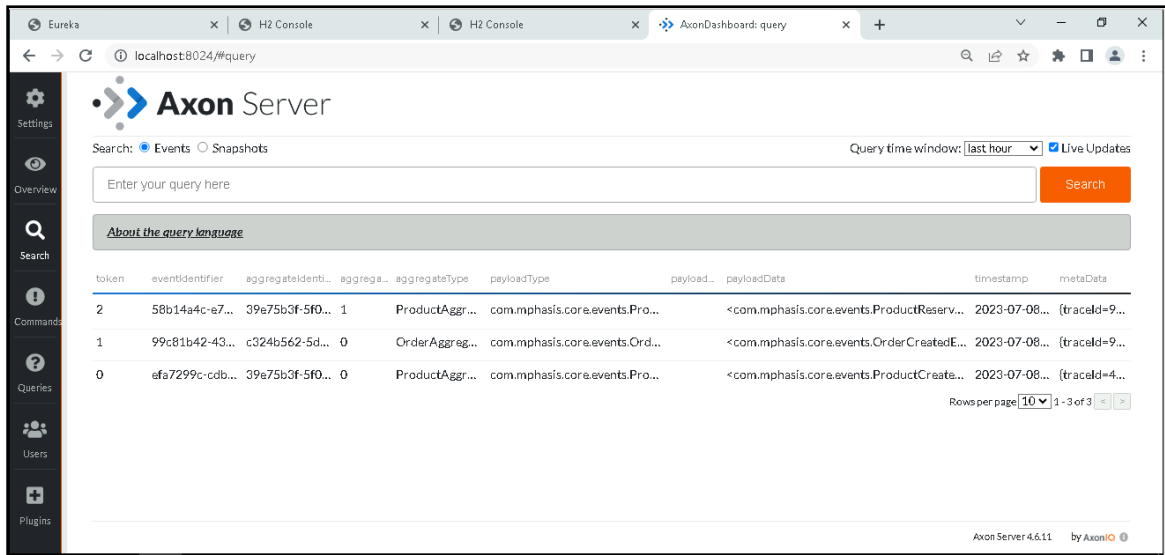


38. Copy the productId, use it in /orders-service/orders. Send a POST request to Create Order.

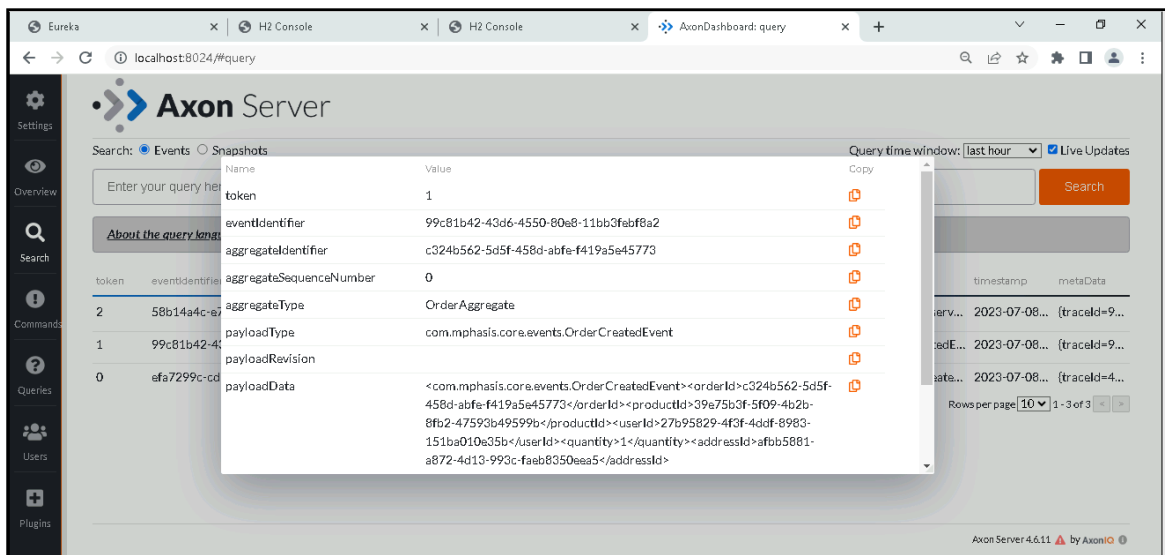


Verify results:

39. Check the Event Store in the Axon server and make sure that the OrderCreatedEvent gets persisted,

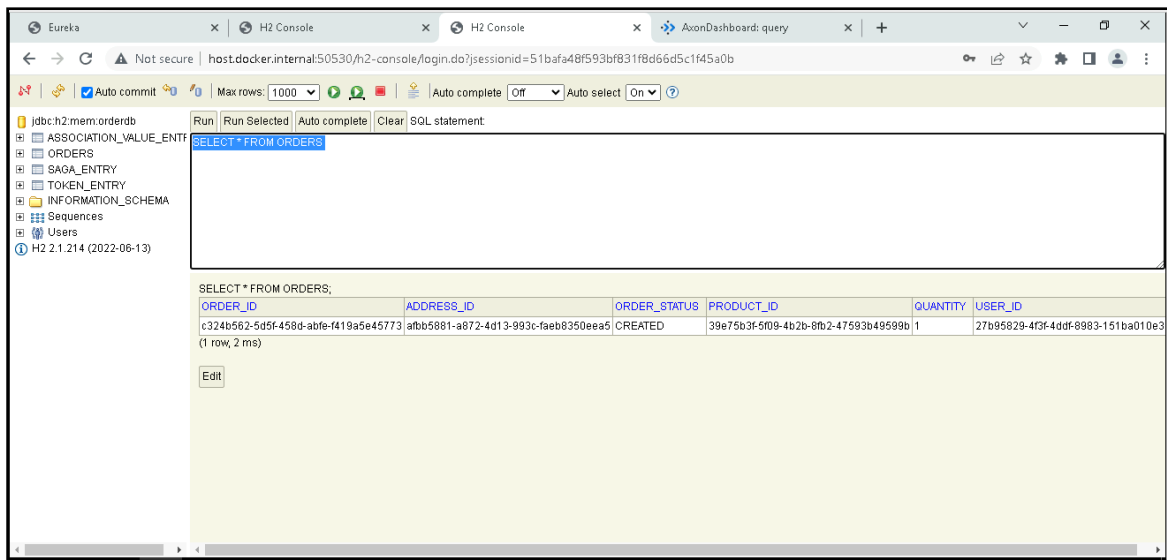


token	eventIdIdentifier	aggregateIdentifier	aggregateType	payloadType	payloadData	timestamp	metaData
2	58b14a4c-e7...	39e75b3f-5f0...	1	ProductAggr...	com.mphasis.core.events.Pro...	<com.mphasis.core.events.ProductReserv...	2023-07-08... (traceld=9...
1	99c81b42-43...	c324b562-5d...	0	OrderAggreg...	com.mphasis.core.events.Ord...	<com.mphasis.core.events.OrderCreatedE...	2023-07-08... (traceld=9...
0	efa7299c-cdb...	39e75b3f-5f0...	0	ProductAggr...	com.mphasis.core.events.Pro...	<com.mphasis.core.events.ProductCreate...	2023-07-08... (traceld=4...



Name	Value	Copy
token	1	
eventIdIdentifier	99c81b42-43d6-4550-80a8-11bb3fabf8a2	
aggregateIdentifier	c324b562-5d5f-458d-abfe-f419a5e45773	
aggregateSequenceNumber	0	
aggregateType	Order Aggregate	
payloadType	com.mphasis.core.events.OrderCreatedEvent	
payloadRevision		
payloadData	<com.mphasis.core.events.OrderCreatedEvent><orderId>c324b562-5d5f-458d-abfe-f419a5e45773</orderId><productId>39e75b3f-5f09-4b2b-8fb2-47593b49599b</productId><userId>27b95829-4f3f-4ddf-8983-151ba010e35b</userId><quantity>1</quantity><addressId>afbb5881-a872-4d13-993c-faeb8350eea5</addressId>	

40. Using the /h2-console connect to the orderdb database and make sure that the order details are stored there as well.

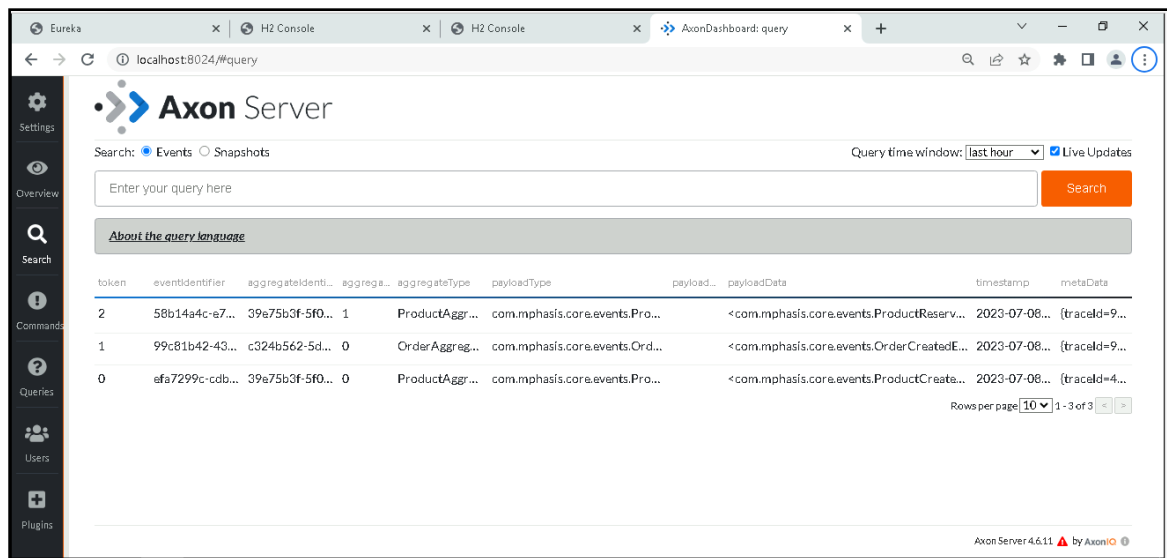


The screenshot shows the H2 Console interface in a web browser. The SQL statement `SELECT * FROM ORDERS;` has been executed successfully. The result set contains one row with the following data:

ORDER_ID	ADDRESS_ID	ORDER_STATUS	PRODUCT_ID	QUANTITY	USER_ID
c324b562-5d5f-458d-abfe-f419a5e45773	atbb5881-a872-4d13-993c-faeb8350ee45	CREATED	39e75b3f-5f09-4b2b-8fb2-47593b49599b	1	27b95829-4f3f-4ddf-8983-151ba010e3

The console also shows the database schema on the left, including tables like `ASSOCIATION_VALUE_ENTRY`, `ORDERS`, `SAGA_ENTRY`, `TOKEN_ENTRY`, and `INFORMATION_SCHEMA`.

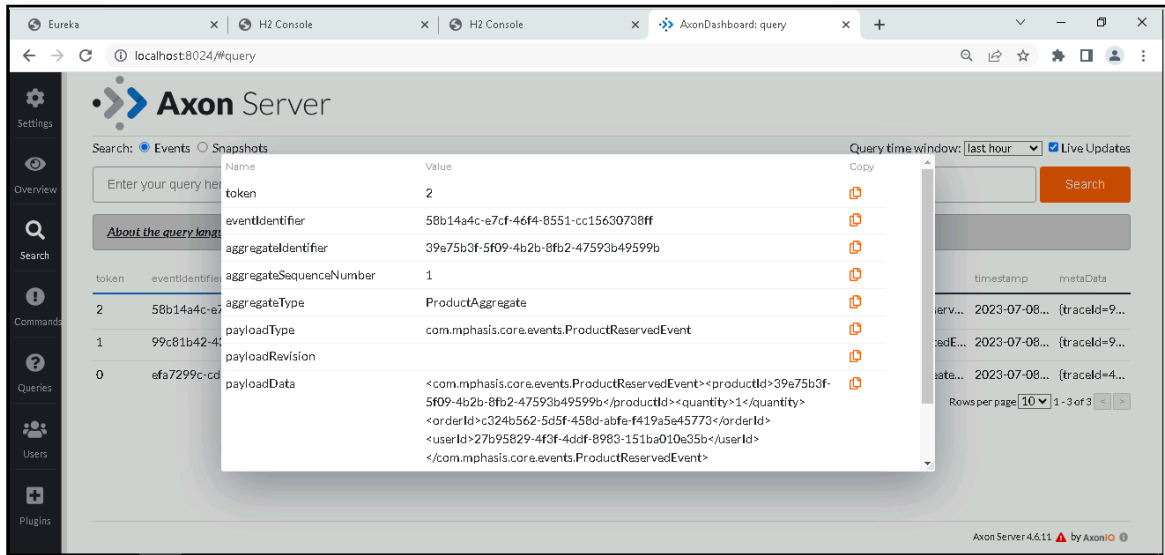
41. You can also find ProductReservedEvent Payload in Axon Server:



The screenshot shows the Axon Server web interface. A query has been executed, and the results are displayed in a table. The table includes columns for token, event identifier, aggregate identifier, aggregate type, payload type, payload data, timestamp, and meta data. The results show three events:

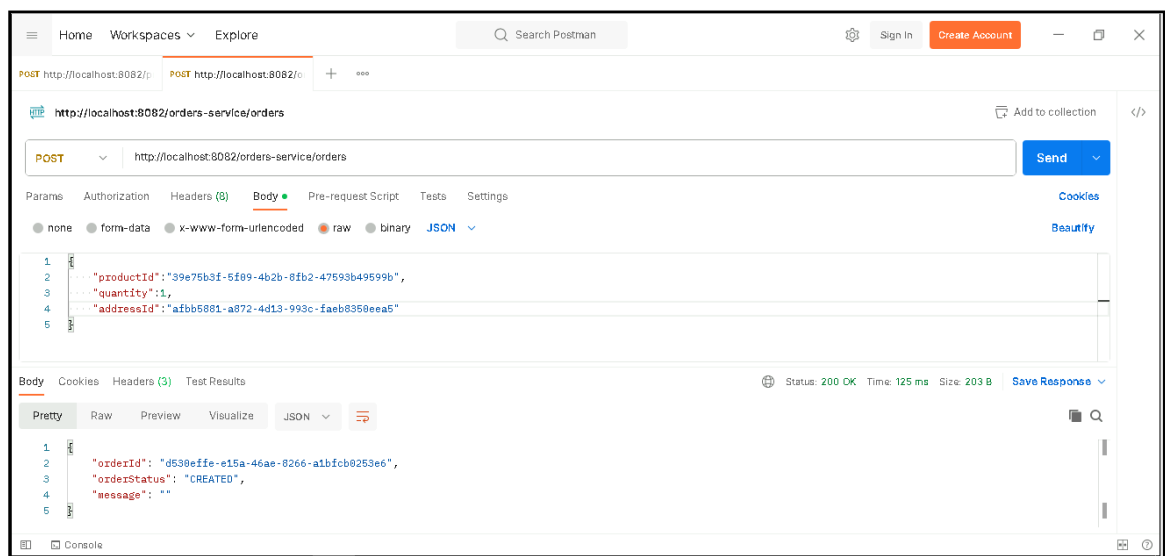
token	eventIdentifier	aggregateIdentifier	aggregateType	payloadType	payloadData	timestamp	metaData
2	58b14a4c-e7...	39e75b3f-5f0...	1	ProductAggr...	com.mphasis.core.events.Pro...	<com.mphasis.core.events.ProductReserv...	2023-07-06... (traceId=9...
1	99c81b42-43...	c324b562-5d...	0	OrderAggreg...	com.mphasis.core.events.Ord...	<com.mphasis.core.events.OrderCreatedE...	2023-07-06... (traceId=9...
0	efa7299c-cdb...	39e75b3f-5f0...	0	ProductAggr...	com.mphasis.core.events.Pro...	<com.mphasis.core.events.ProductCreate...	2023-07-06... (traceId=4...

The interface also includes a search bar, a query time window selector (set to 'last hour'), and a 'Live Updates' checkbox.



42. You can review the logs on the console of each Microservice.

43. Try placing one more order with the same productId. You will find new order with orderId is CREATED.



Problem Statement 14: Orchestration based Saga – Fetch Payment Details

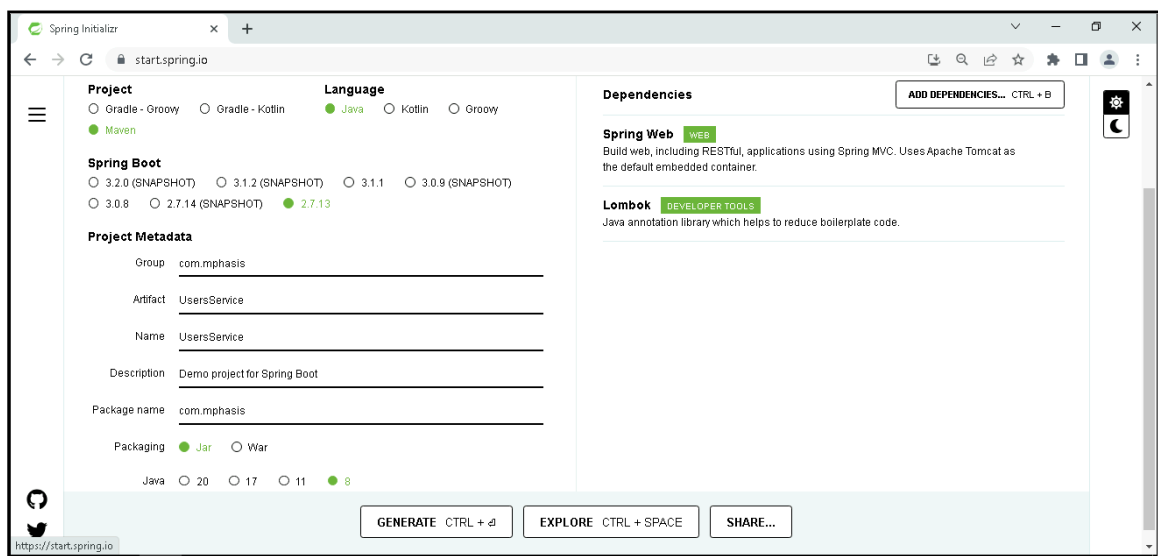
In this problem statement, we will develop a `UserService` that `OrderSaga` will utilize to fetch the user's payment details from another Microservice.

Technology stack:

- Spring Web
- Axon Spring Boot Starter
- Google Guava
- Core (A project with shared classes)

Implementation approach for Orchestration based Saga:

1. Refer the **OrdersService** updated in the problem statement – 13.
2. Refer the **UsersService** created in the problem statement – 13.
3. Create a new Spring Boot Project:
 - Create a new Spring Boot project using either Spring Initializer Tool(<https://start.spring.io>) or using your development environment.
 - Call this new project " `UserService` ".



4. Add the Spring Web, Lombok, Axon Spring Boot Starter, Google Guava, and Core dependencies in pom.xml.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.axonframework</groupId>
    <artifactId>axon-spring-boot-starter</artifactId>
    <version>4.5.8</version>
</dependency>
```

```
<dependency>
    <groupId>com.mphasis</groupId>
    <artifactId>core</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
```

```
<dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>30.1-jre</version>
</dependency>
```

5. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
6. In the Core project, create a new PaymentDetails class with the following fields.

```
private final String name;
private final String cardNumber;
private final int validUntilMonth;
private final int validUntilYear;
private final String cvv;
```

Annotate this class with @Data and @Builder Lombok annotations and place this class into com.mphasis.core.model package.

7. In the Core project, create a new User class with the following fields.

```
private final String firstName;  
private final String lastName;  
private final String userId;  
private final PaymentDetails paymentDetails;
```

Annotate this class with `@Data` and `@Builder` Lombok annotations and place this class into `com.mphasis.core.model` package.

8. In the Core project, create a `FetchUserPaymentDetailsQuery` class with a single instance property for `userId` and place this class into a `com.mphasis.core.query` package.

```
private String userId;
```

Annotate this class with `@Data` and `@AllArgsConstructor` Lombok annotations.

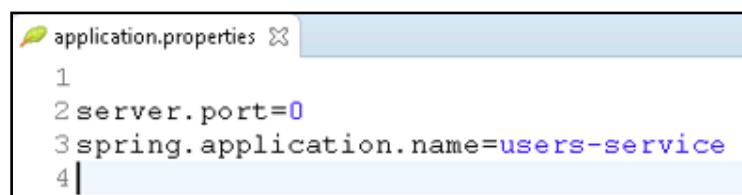
9. In the UsersService project in `com.mphasis.query` package, create a new `UserEventsHandler` class annotated with `@Component` annotation. In this class create a single method annotated with **`@QueryHandler`**. Make this method accept `FetchUserPaymentDetailsQuery` as a method argument and return an instance of a `User` object with hard-coded details. For example,

```
PaymentDetails paymentDetails = PaymentDetails.builder()  
.cardNumber("123Card")  
.cvv("123")  
.name("Manpreet Singh Bindra")  
.validUntilMonth(12)  
.validUntilYear(2030)  
.build();  
  
User userRest = User.builder()  
.firstName("Manpreet Singh")  
.lastName("Bindra")  
.userId(query.getUserId())  
.paymentDetails(paymentDetails)  
.build();
```

10. Create the `UsersQueryController` class in `com.mphasis.query.rest` package.

- The method that accepts the HTTP Get request, should have `User` as a response body and `userId` as a parameter with the URI `"/users/{userId}/payment-details"`.
- This controller class should use the **Axon's QueryGateway** to dispatch an instance of `FetchUserPaymentDetailsQuery`. As we use the gateway's `query()` method to issue a point-to-point query. Because we are specifying `ResponseTypes.instancesOf(User.class)`, Axon knows we only want to talk to query handlers whose return type is a `User` object.

11. Add the property to `application.properties` file:



```
application.properties ✕  
1  
2server.port=0  
3spring.application.name=users-service  
4|
```

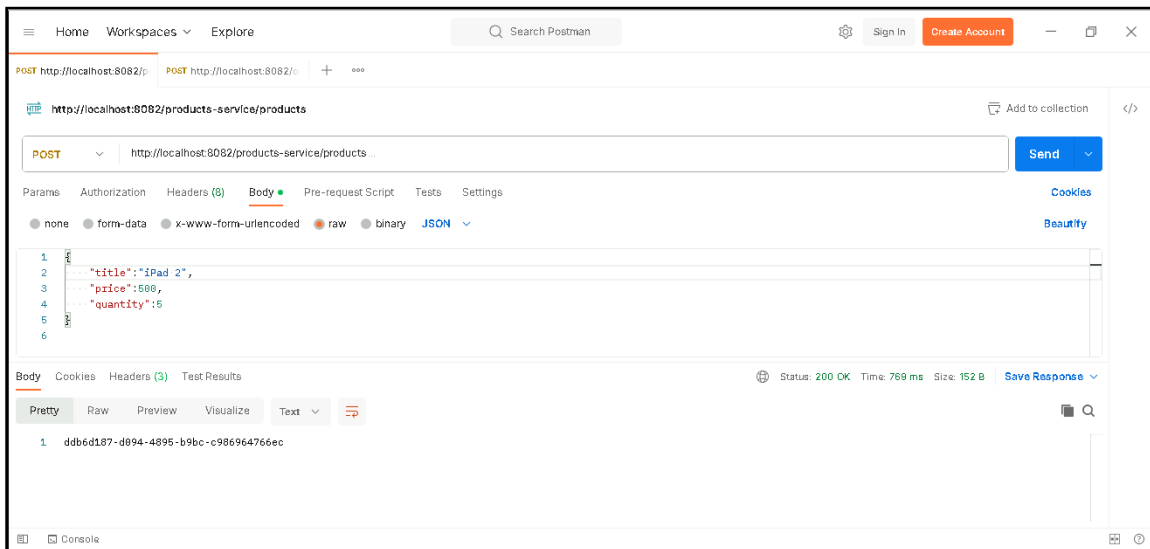
12. Finally, let's go to the `OrdersService – OrderSaga` class, should have one **`@SagaEventHandler`** method that handles the `ProductReservedEvent` and fetch the user payment details from the **"read"** database.

13. The OrderSaga class should use the **Axon's QueryGateway** to dispatch an instance of FetchUserPaymentDetailsQuery.

```
OrderSaga.java
60 @SagaEventHandler(associationProperty = "orderId")
61 public void handle(ProductReservedEvent productReservedEvent) {
62     // Process user payment
63     LOGGER.info("ProductReservedEvent is called for productId: " + productReservedEvent.getProductId() +
64         " and orderId: " + productReservedEvent.getOrderId());
65
66     FetchUserPaymentDetailsQuery fetchUserPaymentDetailsQuery =
67         new FetchUserPaymentDetailsQuery(productReservedEvent.getUserId());
68
69     User userPaymentDetails = null;
70     try {
71         userPaymentDetails = queryGateway.query(fetchUserPaymentDetailsQuery, ResponseTypes.instanceOf(User.class));
72     } catch (Exception ex) {
73         LOGGER.error(ex.getMessage());
74         //Start compensating transaction
75         return;
76     }
77
78     if (userPaymentDetails == null) {
79         //Start compensating transaction
80         return;
81     }
82
83     LOGGER.info("Successfully fetched user payment details for user " + userPaymentDetails.getFirstName());
84 }
```

Run and make it work:

14. Run the Axon Server using Docker command.
15. Start the Discovery Server (Eureka Server), Product Service, OrderService, UserService, and ApiGateway.
16. Send a POST request to Create Product.



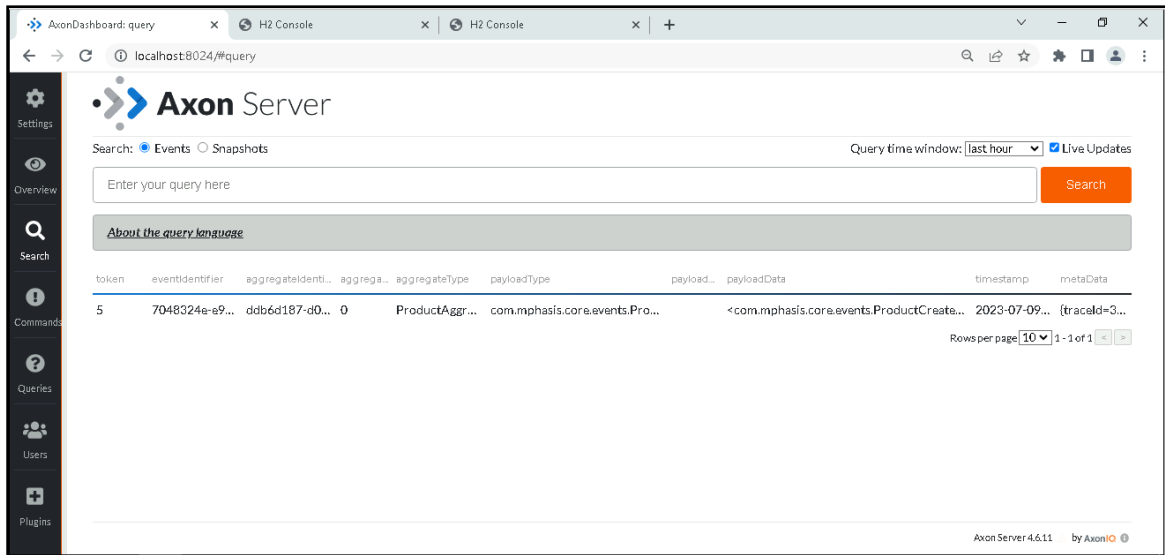
Postman interface showing a successful POST request to `http://localhost:8082/products-service/products`. The request body is a JSON object:

```
{
  "title": "iPad 2",
  "price": 500,
  "quantity": 5
}
```

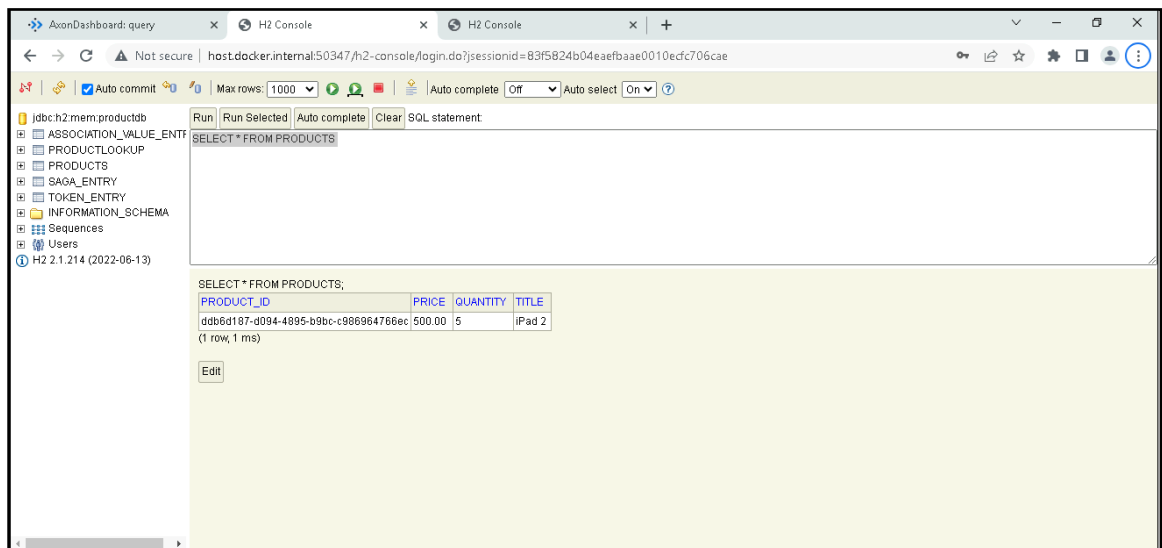
The response status is 200 OK, and the response body is a UUID: `ddb6d187-d094-4895-b9bc-c986964766ec`.

Verify results:

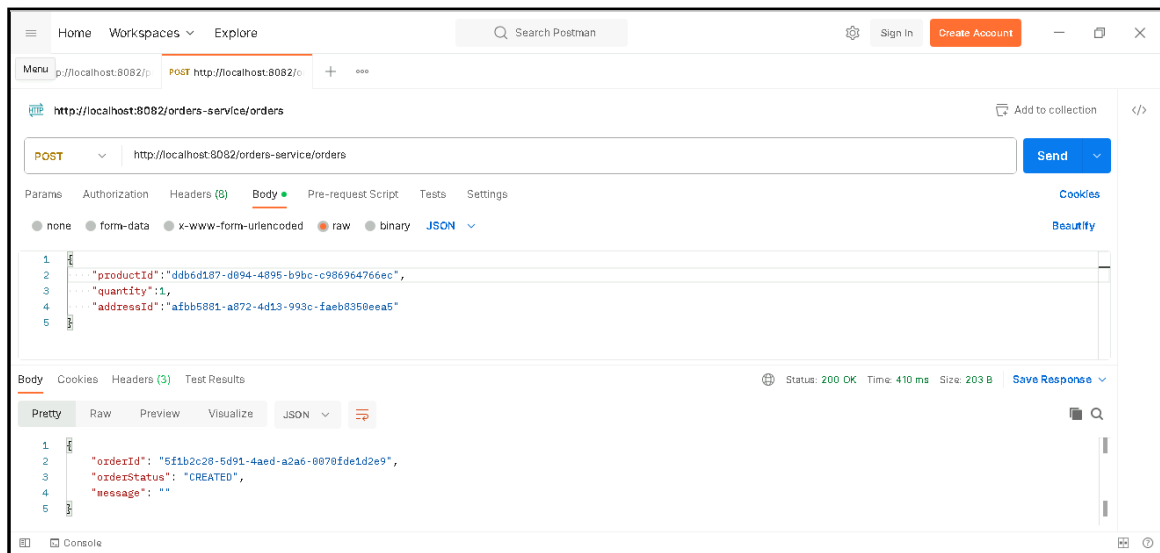
17. Check the Event Store in the Axon server and make sure that the ProductCreatedEvent gets persisted,



18. Using the /h2-console connect to the productdb database and make sure that the product details are stored there as well.

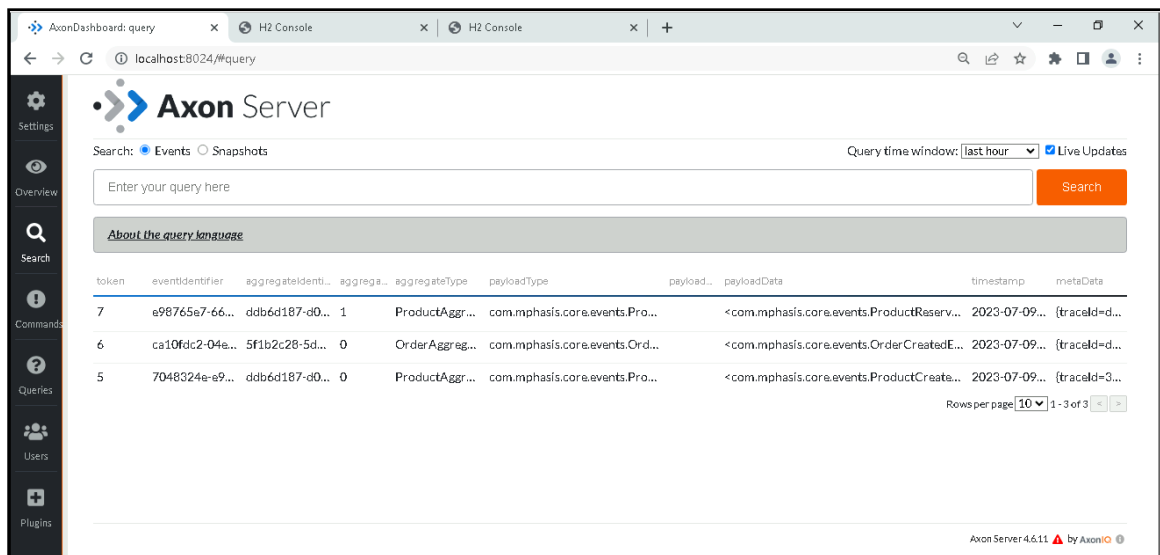


19. Copy the productId, use it in /orders-service/orders. Send a POST request to Create Order.

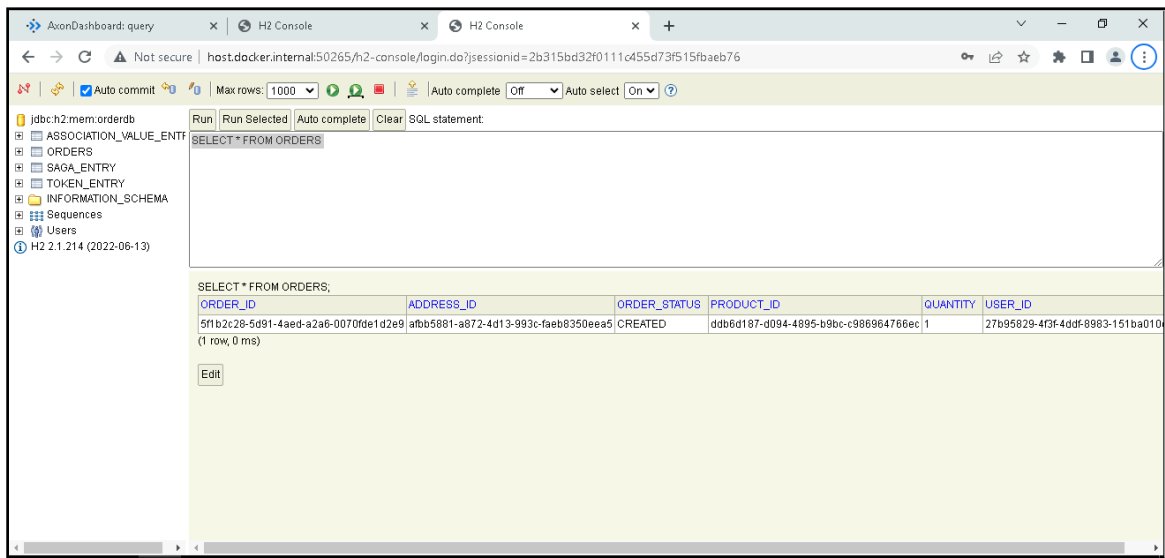


Verify results:

20. Check the Event Store in the Axon server and make sure that the OrderCreatedEvent gets persisted,



21. Using the /h2-console connect to the orderdb database and make sure that the order details are stored there as well.



22. Check the successful log message on the OrdersService console.

```
com.mphasis.saga.OrderSaga : Successfully fetched user payment details for user Manpreet Singh
```

Problem Statement 15: Orchestration based Saga – Process User Payment

As the next step in our Saga flow, we will need to process user payment. If the payment is successful, the order will be approved. If payment is not successful, we will need to initiate a compensating transaction to reject the order.

Because the goal of this problem statement is to learn how SAGA works rather than how to interact with a third-party payment system, the payment Microservice that we will construct will not really transfer user payment credentials to a real payment system. If the payment details are valid, the Payments Microservice will persist the event details to a database table. Otherwise, the payment microservice will return an error.

Technology stack:

- Spring Web
- Spring Data JPA
- H2
- Axon Spring Boot Starter
- Google Guava
- Core (A project with shared classes)

Implementation approach for Orchestration based Saga:

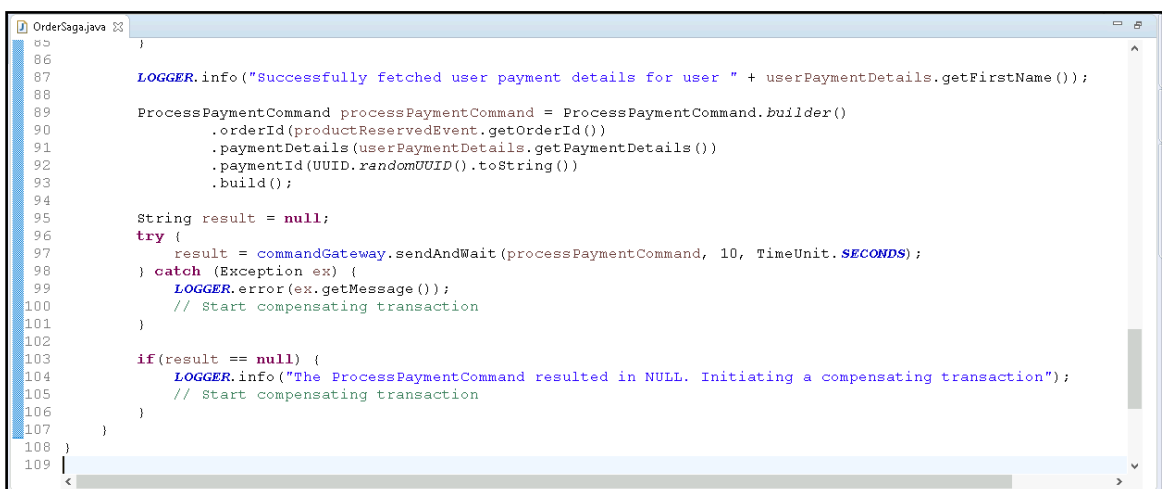
1. Refer the **OrdersService** updated in the problem statement – 14.
2. Refer the **UsersService** updated in the problem statement – 14.
3. In the Core project, create a new **ProcessPaymentCommand** class annotated with **@Data**, **@Builder** in the **com.mphasis.core.commands** package and should have the following fields:

```
private final String paymentId;  
private final String orderId;  
private final PaymentDetails paymentDetails;
```

Where:

paymentId – Annotated with **@TargetAggregateIdentifier** annotation.

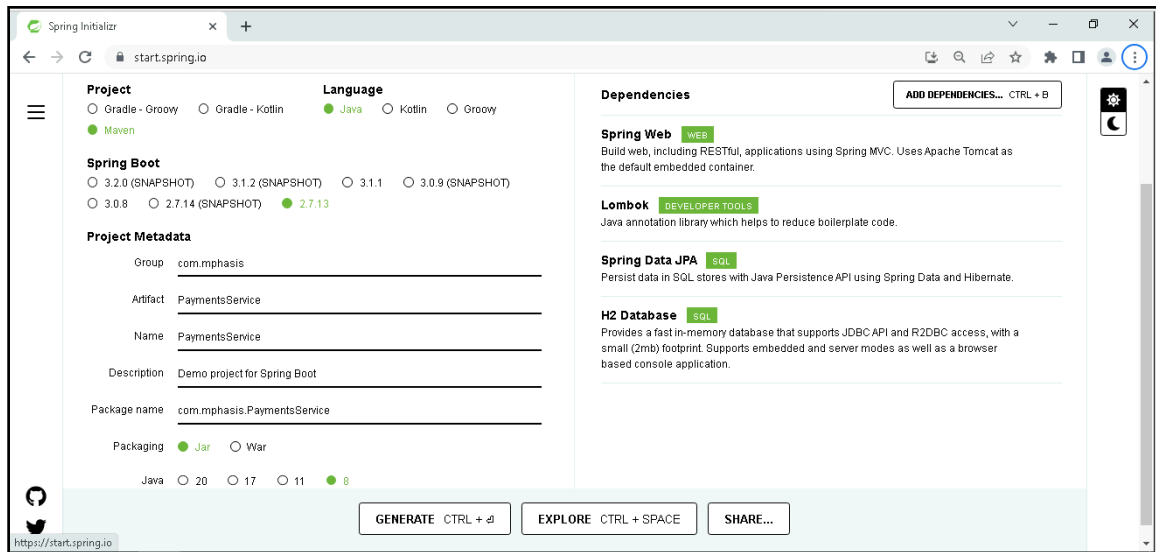
4. Let's go to the **OrdersService – OrderSaga** class, should have one **@SagaEventHandler** method that handles the **ProductReservedEvent** and fetch the user payment details from the "read" database.
5. After that the **OrderSaga** class should use the **Axon's CommandGateway** and publish the **ProcessPaymentCommand**.



```
OrderSaga.java  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
212
```

6. Create a new Spring Boot Project:

- Create a new Spring Boot project using either Spring Initializer Tool(<https://start.spring.io>) or using your development environment.
- Call this new project " PaymentsService".



7. Add the Spring Web, Lombok, Spring Data JPA, H2, Axon Spring Boot Starter, Google Guava, and Core dependencies in pom.xml.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.axonframework</groupId>
  <artifactId>axon-spring-boot-starter</artifactId>
  <version>4.5.8</version>
</dependency>

<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>30.1-jre</version>
</dependency>
```

```
<dependency>
  <groupId>com.mphasis</groupId>
  <artifactId>core</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

8. Will take some time for the project to be imported and the maven dependencies to be downloaded once you click **Finish**.
9. In the PaymentsService, create a new class called PaymentAggregate, make it handle the ProcessPaymentCommand, and publish the PaymentProcessedEvent.
10. In the Core project, create the PaymentProcessedEvent class will have the following fields:
private final String orderId;
private final String paymentId;
Annotate this class with @Value annotations and in the com.mphasis.core.events package.
11. The PaymentAggregate class should also have an **@EventSourcingHandler** method that sets values for all fields in the PaymentAggregate.
12. In the PaymentAggregate class, create a **@CommandHandler** method that validate the ProcessPaymentCommand. If one of the required fields contains an invalid value, then throw an IllegalArgumentException.

```

21 @CommandHandler
22 public PaymentAggregate (ProcessPaymentCommand processPaymentCommand) {
23
24     if (processPaymentCommand.getPaymentDetails() == null) {
25         throw new IllegalArgumentException("Missing payment details");
26     }
27
28     if (processPaymentCommand.getOrderId() == null) {
29         throw new IllegalArgumentException("Missing orderId");
30     }
31
32     if (processPaymentCommand.getPaymentId() == null) {
33         throw new IllegalArgumentException("Missing paymentId");
34     }
35
36     AggregateLifecycle.apply(new PaymentProcessedEvent (processPaymentCommand.getOrderId(),
37                                                         processPaymentCommand.getPaymentId()));
38 }
39

```

13. Create a new `@Component` class called `PaymentEventsHandler` in `com.mphasis.events` package.
14. Create a new JPA Repository called `PaymentsRepository` in `com.mphasis.data` package and inject it into `PaymentEventsHandler` using constructor-based dependency injection.
15. The `PaymentEventsHandler` class should have one `@EventHandler` method that handles the `PaymentProcessedEvent` and persists payment details into the "read" database.
16. To persist payment details into the database, create a new JPA Entity class called `PaymentEntity` in `com.mphasis.data` package. Annotate the `PaymentEntity` class with:


```

@Data
@Entity
@Table(name = "payments")

```

 and make the `PaymentEntity` class have the following fields:


```

@Id
private String paymentId;
@Column
public String orderId;

```
17. Since each Microservice should store data in its own database, configure Payments Microservice to work with a new database called "paymentdb".
18. Add the DB properties to `application.properties` file:

```

application.properties
1
2 server.port=0
3 spring.application.name=payments-service
4
5 spring.datasource.url=jdbc:h2:mem:paymentdb
6 spring.datasource.driver-class-name=org.h2.Driver
7 spring.datasource.username=sa
8 spring.datasource.password=password
9 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
10
11 #Accessing the H2 Console
12 spring.h2.console.enabled=true
13 spring.h2.console.path=/h2-console
14 spring.h2.console.settings.web-allow-others=true
15
16

```

19. Finally, let's go to the OrdersService – OrderSaga class, should have one **@SagaEventHandler** method with the **associationProperty "orderId"** that handles the PaymentProcessedEvent and publish the ApproveOrderCommand.
20. In the OrdersService project, create a new ApproveOrderCommand class annotated with **@Data** and **@AllArgsConstructor** in the com.mphasis.command.commands package and should have the following field:

private final String orderId;

Where:
orderId – Annotated with **@TargetAggregateIdentifier** annotation.
21. In the OrderAggregate class, create a **@CommandHandler** method that will handle the ApproveOrderCommand and publish the OrderApprovedEvent.
22. In the OrdersService project, create a new OrderApprovedEvent class annotated with **@Value** in the com.mphasis.core.events package and should have the following fields:

private final String orderId;
private final OrderStatus orderStatus = OrderStatus.Approved;
23. The OrderAggregate class should have **@EventSourcingHandler** method that will handle the OrderApprovedEvent and set the orderStatus field.
24. Create a new JPA Repository called OrdersRepository inside com.mphasis.core.data package and inject it into OrderEventsHandler using constructor-based dependency injection.
25. Add the find method in OrderRepository interface:

```
OrderEntity findById(String orderId);
```

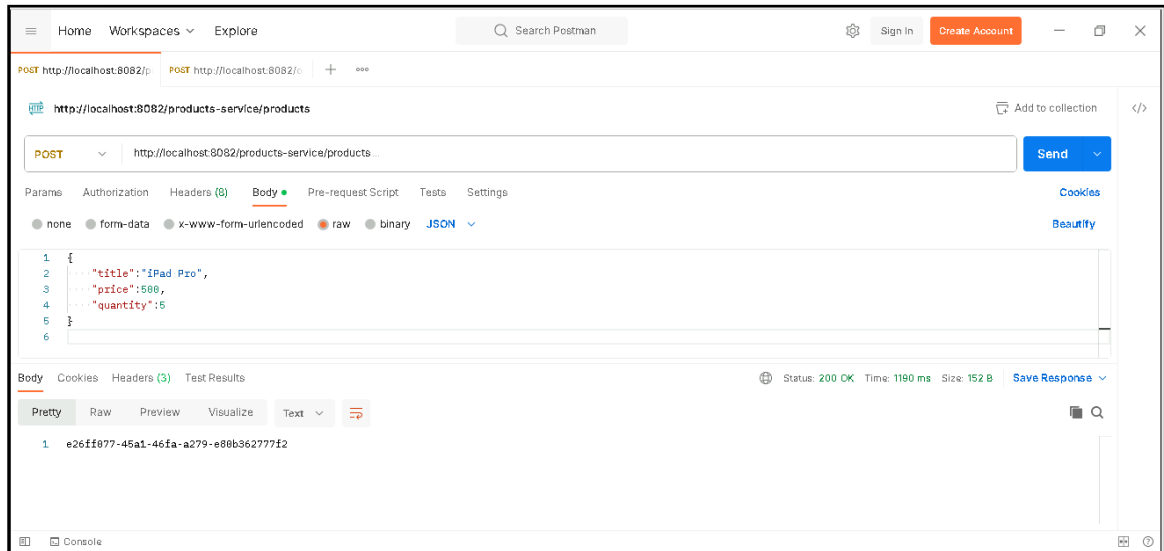
26. The OrderEventsHandler class should have **@EventHandler** method that will handle the OrderApprovedEvent and persist the order details in "read" database.
27. Finally, let's go to the OrdersService – OrderSaga class, should have one **@SagaEventHandler** method with the **associationProperty "orderId"** that handles the OrderApprovedEvent and log the orderId.
28. This method should be annotated with **@EndSaga** which indicates the end of a Saga instance's lifecycle. When event handling completes, the Saga is destroyed and may no longer receive events.

```
@EndSaga
@SagaEventHandler(associationProperty = "orderId")
public void handle(OrderApprovedEvent orderApprovedEvent) {

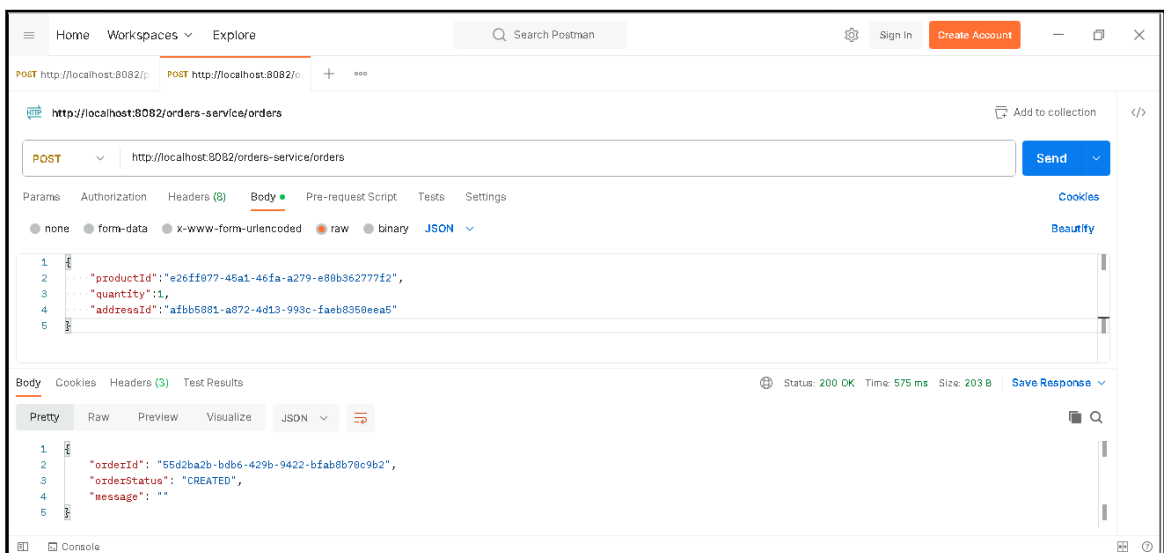
    LOGGER.info("Order is approved. Order Saga is complete for orderId: "
        + orderApprovedEvent.getOrderId());
    // SagaLifecycle.end();
}
```

Run and make it work:

29. Run the Axon Server using Docker command.
30. Start the Discovery Server (Eureka Server), Product Service, OrdersService, UsersService, PaymentsService, and ApiGateway.
31. Send a POST request to Create Product.



32. Copy the productId, use it in `/orders-service/orders`. Send a POST request to Create Order.



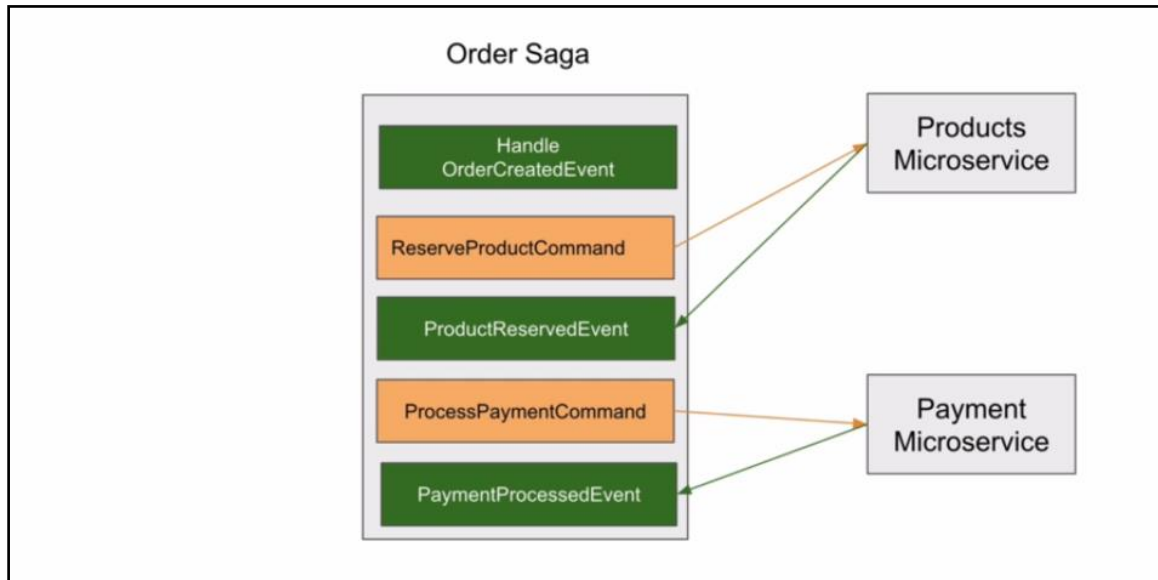
33. Check the Order approved log message on the OrdersService console.

```
com.mphasis.saga.OrderSaga : Order is approved. Order Saga is complete for orderId: 55d2ba2b-bdb6-429b-9422-
```


Problem Statement 16: Saga Compensating Transaction in Microservices

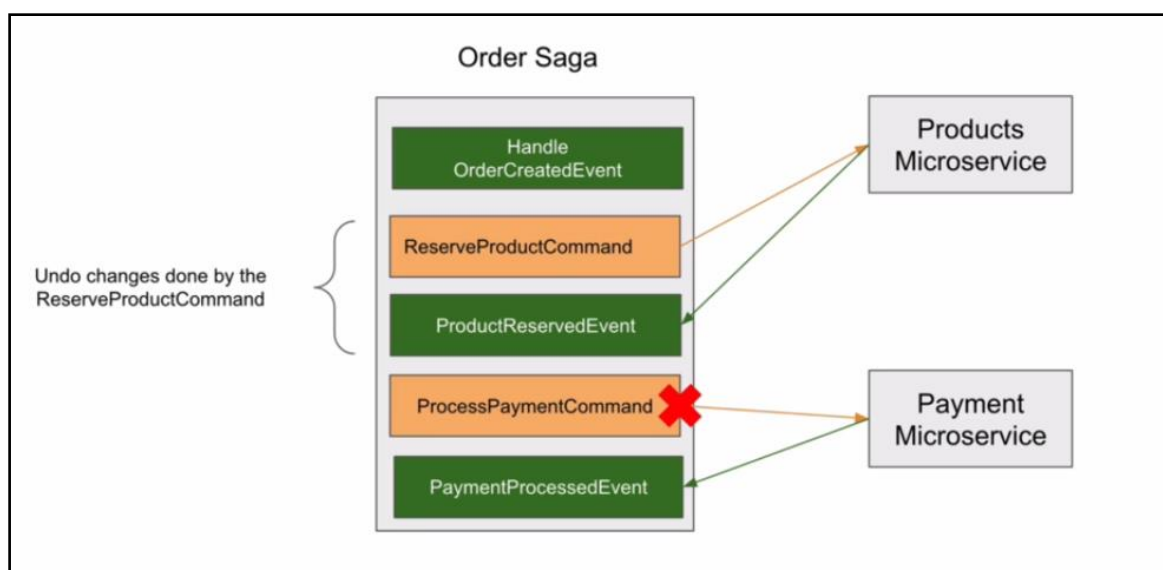
As you already know, Saga is event handling component that maintains a sequence of local transactions. In previous problem statements, we have constructed a Saga that manages a very simple order form. It will handle an event and it will publish a new command to trigger the next Saga local transactions.

Here is a diagram explaining the order flow:



If one of the local transactions fails, Saga will need to run a series of compensated transactions to undo the modifications done by the previous transactions. For example, if the process payment command fails, Saga must undo the subsequent modifying transaction initiated by the reserve product command. If there were more altering transactions, Saga will have to undo them all in reverse order.

Here is a diagram explaining the failed order flow:



Implementation of compensating transaction in our Saga flow:

1. Refer all the **Microservices** created in the previous problem statement.
2. In the Core project, create the CancelProductReservationCommand class annotated with @Data and @Builder in the com.mphasis.core.commands package and should have the following fields:

```
private final String productId;
private final int quantity;
private final String orderId;
private final String userId;
private final String reason;
```

Where:

orderId – Annotated with **@TargetAggregateIdentifier** annotation.

3. In the OrdersService – OrderSaga class, create a new private cancelProductReservation method with the ProductReservedEvent and reason as an argument. Write a code to create an instance of CancelProductReservationCommand and use **Axon's CommandGateway** to publish it.
4. Now publish from 4 places in the OrderSaga class:

```
70= @SagaEventHandler(associationProperty = "orderId")
71 public void handle(ProductReservedEvent productReservedEvent) {
72     // Process user payment
73     LOGGER.info("ProductReservedEvent is called for productId: " + productReservedEvent.getProductId() +
74         " and orderId: " + productReservedEvent.getOrderId());
75
76     FetchUserPaymentDetailsQuery fetchUserPaymentDetailsQuery =
77         new FetchUserPaymentDetailsQuery(productReservedEvent.getUserId());
78
79     User userPaymentDetails = null;
80     try {
81         userPaymentDetails = queryGateway.query(fetchUserPaymentDetailsQuery, ResponseTypes.instanceOf(User.class));
82     } catch (Exception ex) {
83         LOGGER.error(ex.getMessage());
84         //Start compensating transaction
85         cancelProductReservation(productReservedEvent, ex.getMessage());
86         return;
87     }
88
89     if (userPaymentDetails == null) {
90         //Start compensating transaction
91         cancelProductReservation(productReservedEvent, "Could not fetch user payment details");
92         return;
93     }
94 }
```

```
95     LOGGER.info("Successfully fetched user payment details for user " + userPaymentDetails.getFirstName());
96
97     ProcessPaymentCommand processPaymentCommand = ProcessPaymentCommand.builder()
98         .orderId(productReservedEvent.getOrderId())
99         .paymentDetails(userPaymentDetails.getPaymentDetails())
100         .paymentId(UUID.randomUUID().toString())
101         .build();
102
103     String result = null;
104     try {
105         result = commandGateway.sendAndWait(processPaymentCommand, 10, TimeUnit.SECONDS);
106     } catch (Exception ex) {
107         LOGGER.error(ex.getMessage());
108         // Start compensating transaction
109         cancelProductReservation(productReservedEvent, ex.getMessage());
110         return;
111     }
112
113     if (result == null) {
114         LOGGER.info("The ProcessPaymentCommand resulted in NULL. Initiating a compensating transaction");
115         // Start compensating transaction
116         cancelProductReservation(productReservedEvent, "Could not process user payment with provided payment de
117     }
118 }
```

5. In the ProductService – ProductAggregate class, create a **@CommandHandler** method that will handle the CancelProductReservationCommand and publish the ProductReservationCancelledEvent.

6. In the Core project, create the ProductReservationCancelledEvent class will have the following fields:

```
private final String productId;  
private final int quantity;  
private final String orderId;  
private final String userId;  
private final String reason;
```

Annotate this class with `@Data` and `@Builder` annotations and in the `com.mphasis.core.events` package.

7. In the ProductService – ProductAggregate class, also create an **@EventSourcingHandler** method which is handling the ProductReservationCancelledEvent and updating the quantity (addition action).
8. To make our read database up to date with the changes that we have just made to the product quantity, we will need to handle the ProductReservationCancelledEvent and update the read database as well.
9. In ProductService – ProductEventsHandler class, we need to add one more **@EventHandler** method for the ProductReservationCancelledEvent and update the Products projection.
10. Finally, we will Handle the ProductReservationCancelledEvent in OrderService – OrderSaga, by creating a new handler method annotated with **@SagaEventHandler** and **associationProperty** i.e., orderId.

```
@SagaEventHandler(associationProperty = "orderId")  
public void handle(ProductReservationCancelledEvent productReservationCancelledEvent) {  
  
    // Create and send a RejectOrderCommand  
}
```

11. In the OrdersService project, create the RejectOrderCommand class annotated with `@Value` in the `com.mphasis.command.commands` package and should have the following fields:

```
private final String orderId;  
private final String reason;
```

Where:

orderId – Annotated with **@TargetAggregateIdentifier** annotation.

12. Let's go to OrderSaga class inside the ProductReservationCancelledEvent handler method, write a code to create an instance of RejectOrderCommand and use **Axon's CommandGateway** to publish it.

```
OrderSaga.java  
154  
155 @SagaEventHandler(associationProperty = "orderId")  
156 public void handle(ProductReservationCancelledEvent productReservationCancelledEvent) {  
157  
158     // Create and send a RejectOrderCommand  
159     RejectOrderCommand rejectOrderCommand = new RejectOrderCommand(  
160         productReservationCancelledEvent.getOrderId(), productReservationCancelledEvent.getReason());  
161  
162     commandGateway.send(rejectOrderCommand);  
163 }  
164  
165
```

- Also publish the RejectOrderCommand from OrderSaga class inside the OrderCreatedEvent handler method if order is not created.

```

59
60=    commandGateway.send(reserveProductCommand, new CommandCallback<ReserveProductCommand, Object>() {
61
62=    @Override
63        public void onResult (CommandMessage<? extends ReserveProductCommand> commandMessage,
64                             CommandResultMessage<? extends Object> commandResultMessage) {
65
66            if (commandResultMessage.isExceptional()) {
67                // Start a compensating transaction
68                RejectOrderCommand rejectOrderCommand = new RejectOrderCommand(
69                    orderCreatedEvent.getOrderId(), commandResultMessage.exceptionResult().getMessage());
70
71                commandGateway.send(rejectOrderCommand);
72            }
73        }
74    });
75
    
```

- In the OrdersService – OrderAggregate class, create a **@CommandHandler** method that will handle the RejectOrderCommand and publish the OrderRejectedEvent.
- In the OrdersService project, create the OrderRejectedEvent class will have the following fields:
 private final String orderId;
 private final String reason;
 private final OrderStatus orderStatus = OrderStatus.REJECTED;
 Annotate this class with @Value annotations and in the com.mphasis.core.events package.
- In the OrdersService – OrderAggregate class, should also have an **@EventSourcingHandler** method that sets value of the orderStatus field.
- In OrdersService – OrderEventsHandler class should have **@EventHandler** method for the OrderRejectedEvent and persist the order details in “read” database.
- Finally, let’s go to the OrdersService – OrderSaga class, should have one **@SagaEventHandler** method with the **associationProperty “orderId”** that handles the OrderRejectedEvent and log the orderId.
- This method should be annotated with **@EndSaga** which indicates the end of a Saga instance’s lifecycle. When event handling completes, the Saga is destroyed and may no longer receive events.

```

@EndSaga
@sagaEventHandler(associationProperty = "orderId")
public void handle (OrderRejectedEvent orderRejectedEvent) {

    LOGGER.info("Successfully rejected order with id " + orderRejectedEvent.getOrderId());
}
    
```

Run and make it work:

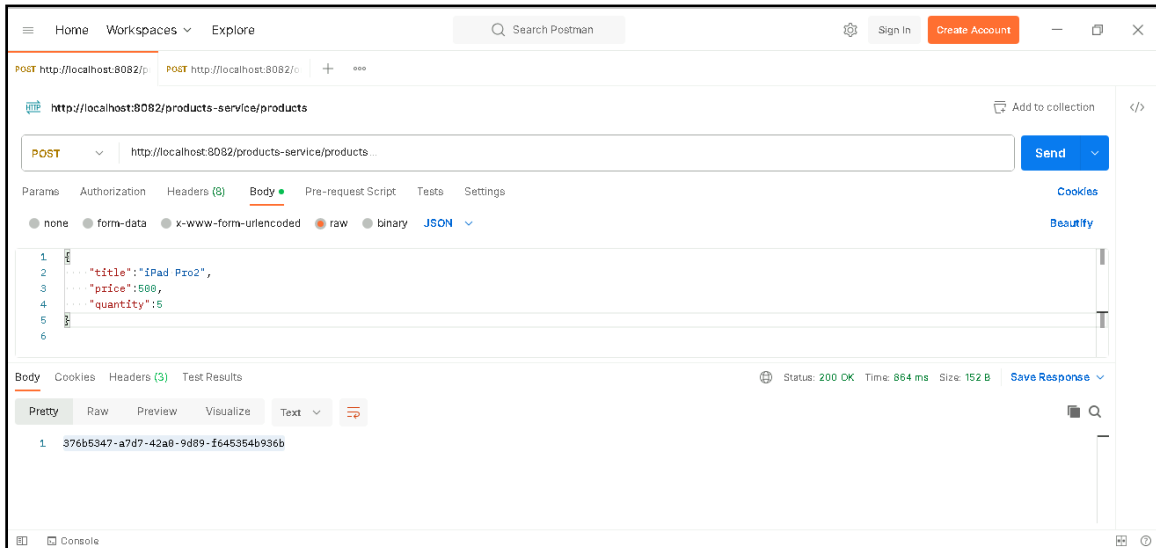
- Run the Axon Server using Docker command.
- Start the Discovery Server (Eureka Server), Product Service, OrdersService, UsersService, PaymentsService and ApiGateway.

22. Add logs to handler in ProductEventsHandler class:

```
54 @EventHandler
55 public void on(ProductReservedEvent productReservedEvent) throws Exception {
56
57     ProductEntity productEntity = productRepository.findByProductId(productReservedEvent.getProductId());
58
59     LOGGER.info("ProductReservedEvent: Current product quantity " + productEntity.getQuantity());
60
61     productEntity.setQuantity(productEntity.getQuantity() - productReservedEvent.getQuantity());
62     productRepository.save(productEntity);
63
64     LOGGER.info("ProductReservedEvent: New product quantity " + productEntity.getQuantity());
65
66     LOGGER.info("ProductReservedEvent is called for productId: " + productReservedEvent.getProductId() +
67         " and orderId: " + productReservedEvent.getOrderId());
68 }
69
```

```
70 @EventHandler
71 public void on(ProductReservationCancelledEvent productReservationCancelledEvent) throws Exception {
72
73     ProductEntity currentlyStoredProduct =
74         productRepository.findByProductId(productReservationCancelledEvent.getProductId());
75
76     LOGGER.info("ProductReservationCancelledEvent: Current product quantity "
77         + productReservationCancelledEvent.getQuantity());
78
79     int newQuantity = currentlyStoredProduct.getQuantity() + productReservationCancelledEvent.getQuantity();
80     currentlyStoredProduct.setQuantity(newQuantity);
81
82     productRepository.save(currentlyStoredProduct);
83
84     LOGGER.info("ProductReservationCancelledEvent: New product quantity "
85         + productReservationCancelledEvent.getQuantity());
86 }
87
88
```

23. Send a POST request to Create Product.

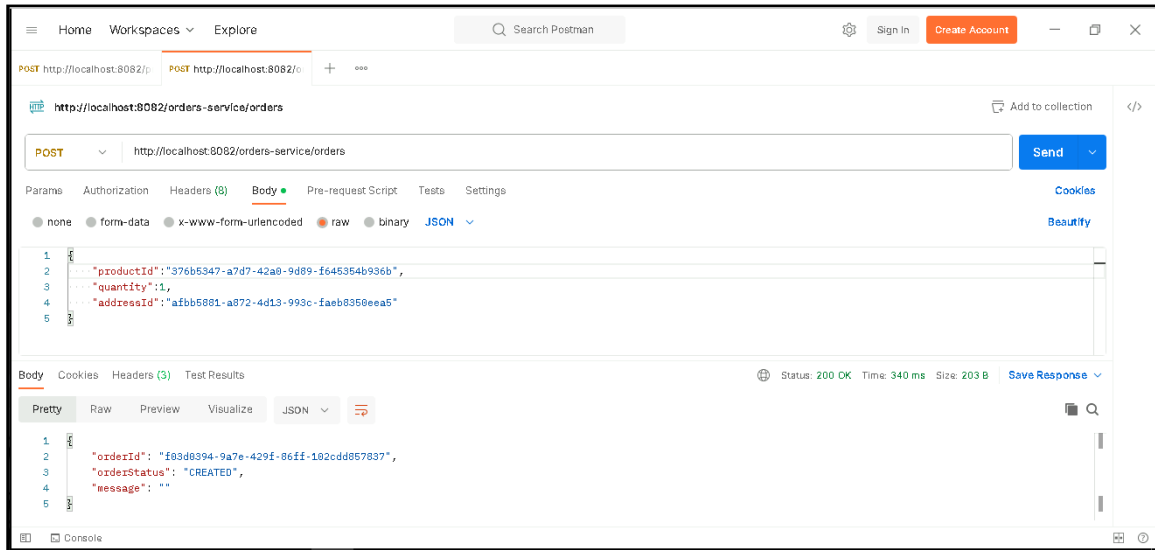


The screenshot shows the Postman interface for a POST request to `http://localhost:8082/products-service/products`. The request body is a JSON object:

```
{
  "title": "iPad Pro2",
  "price": 600,
  "quantity": 5
}
```

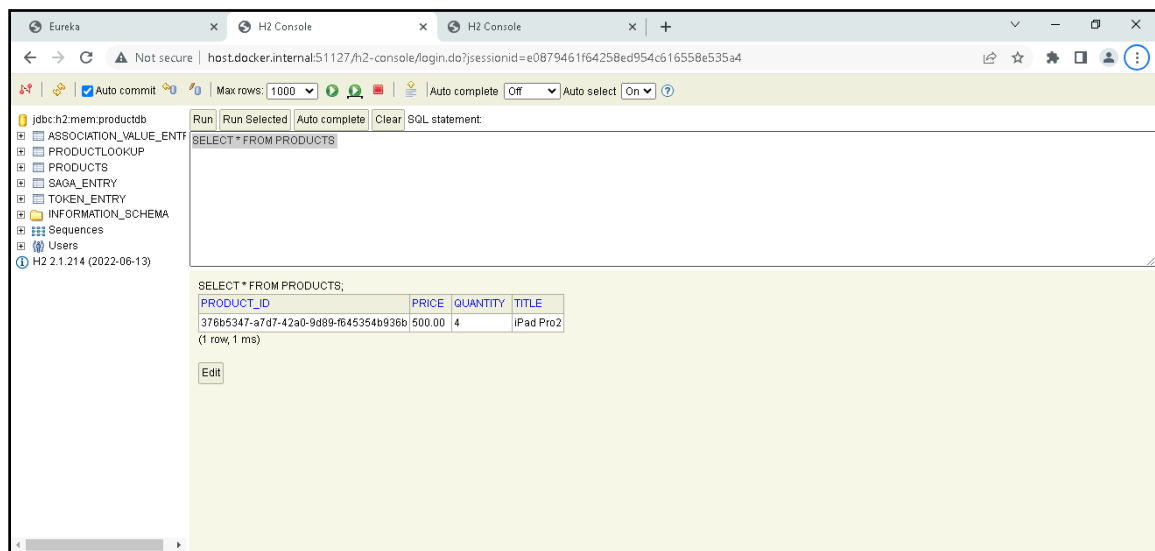
The response status is 200 OK, and the response body is a UUID: `876b5347-a7d7-42a0-9d89-f645354b936b`.

24. Copy the productId, use it in /orders-service/orders. Send a POST request to Create Order.

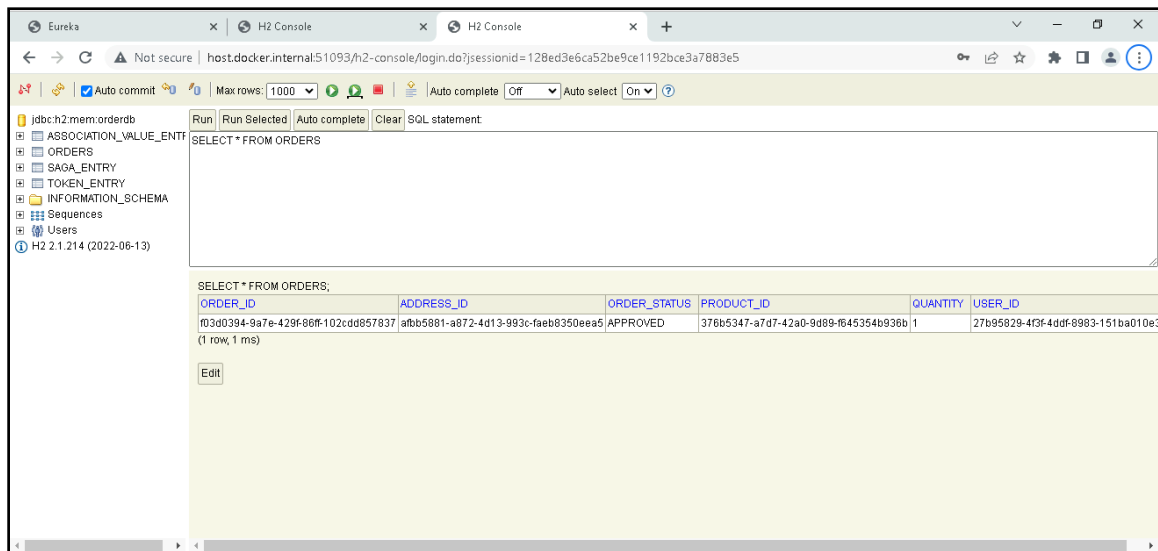


25. Check the Order approved log message on the OrdersService console.

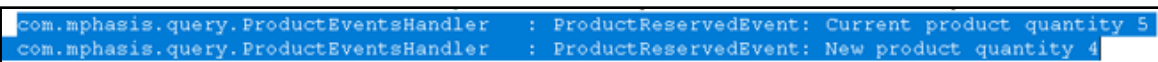
26. Using the /h2-console connect to the Products database and make sure that the product details are stored there as well. Also, check the Quantity after the order is placed successfully.



27. Using the /h2-console connect to the Orders database and make sure that the order details are stored there as well.

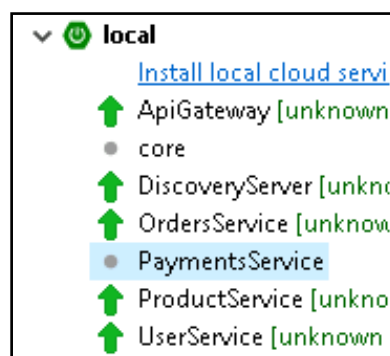


28. Check the Product Quantity logs messages on the ProductService console.



29. We will now Stop PaymentsService and place a new Order. A new Order reduces the number of products in stock by one more item. Because PaymentsService is unavailable and will be unable to accept user payment, it should kick in the **compensating transaction** in our **OrderSaga**, which will undo changes done by the proceeding transactions. Finally, we will notice the Product Quantity in the Product database has been increased once more.

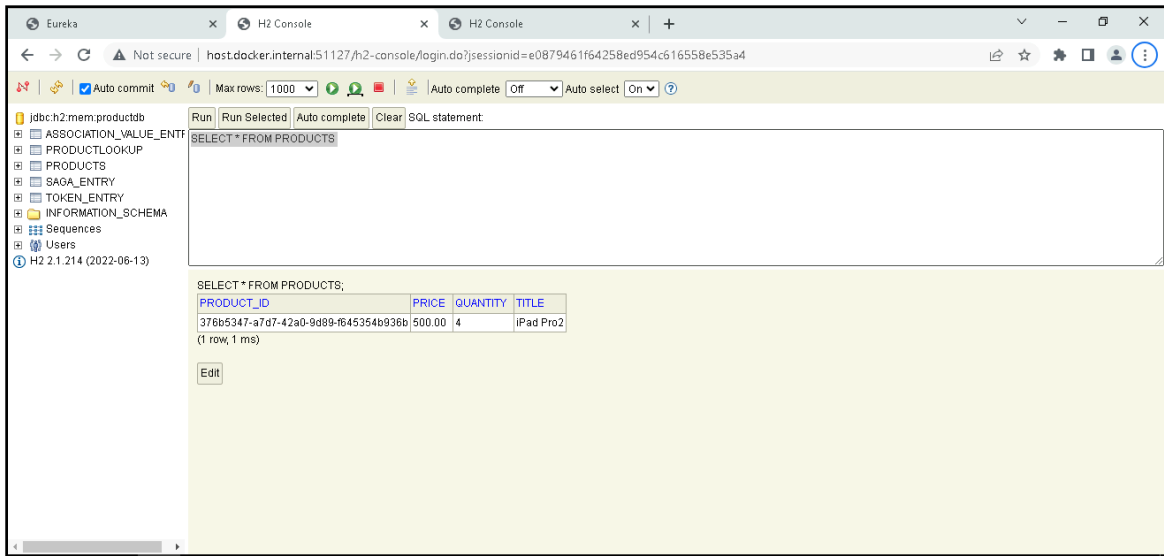
30. Let's stop the PaymentsService:



31. Let's place the order again, to the same productId.
32. Using the /h2-console connect to the Orders database and make sure that the **rejected** order details are stored there as well.

ADDRESS_ID	ORDER_STATUS
afbb5881-a872-4d13-993c-faeb8350eea5	APPROVED
afbb5881-a872-4d13-993c-faeb8350eea5	REJECTED

33. Using the /h2-console connect to the Products database and check the Quantity after the order is Rejected.



The screenshot shows the H2 Console web interface. The browser address bar indicates the URL: `host.docker.internal:51127/h2-console/login.do?jsessionid=e0879461f64258ed954c616558e535a4`. The interface includes a sidebar with a tree view of database schemas, including `jdbc:h2:mem:productdb`, `ASSOCIATION_VALUE_ENTRY`, `PRODUCTLOOKUP`, `PRODUCTS`, `SAGA_ENTRY`, `TOKEN_ENTRY`, `INFORMATION_SCHEMA`, `Sequences`, and `Users`. The main area contains a text input field with the SQL query `SELECT * FROM PRODUCTS`. Below the query, there are buttons for `Run`, `Run Selected`, `Auto complete`, and `Clear`. The results of the query are displayed in a table with columns `PRODUCT_ID`, `PRICE`, `QUANTITY`, and `TITLE`. The table contains one row with the following data: `376b5347-a7d7-42a0-9d89-6645354b936b`, `500.00`, `4`, and `iPad Pro2`. Below the table, it indicates `(1 row, 1 ms)` and an `Edit` button.

PRODUCT_ID	PRICE	QUANTITY	TITLE
376b5347-a7d7-42a0-9d89-6645354b936b	500.00	4	iPad Pro2