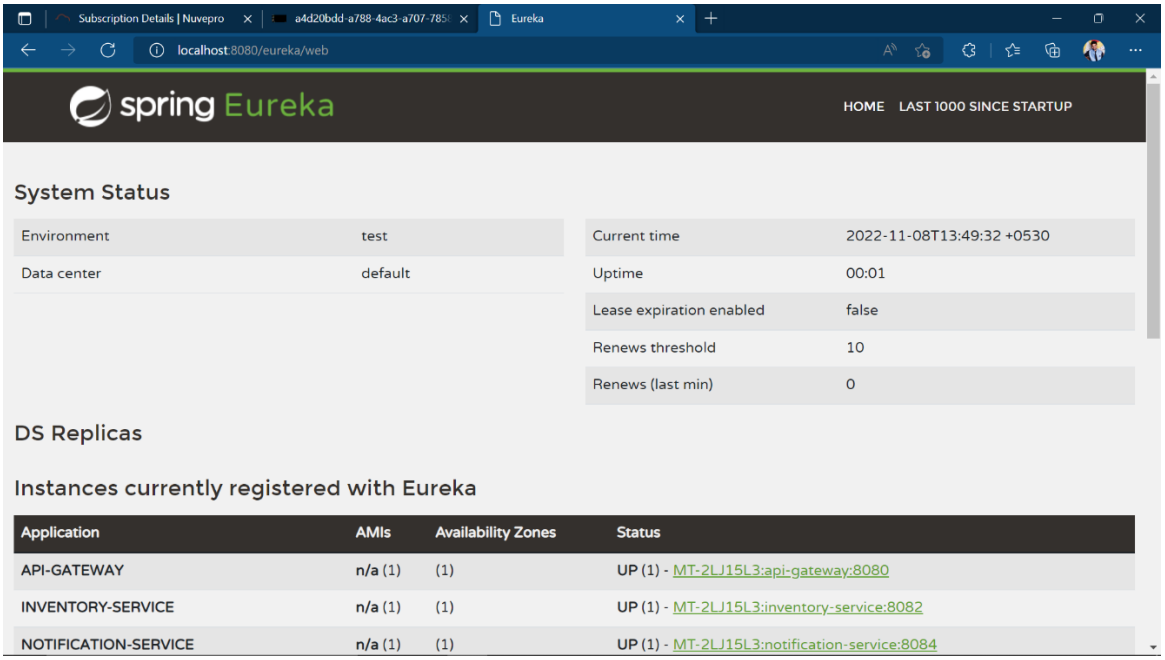
ORDER MY FOOD APPLICATION

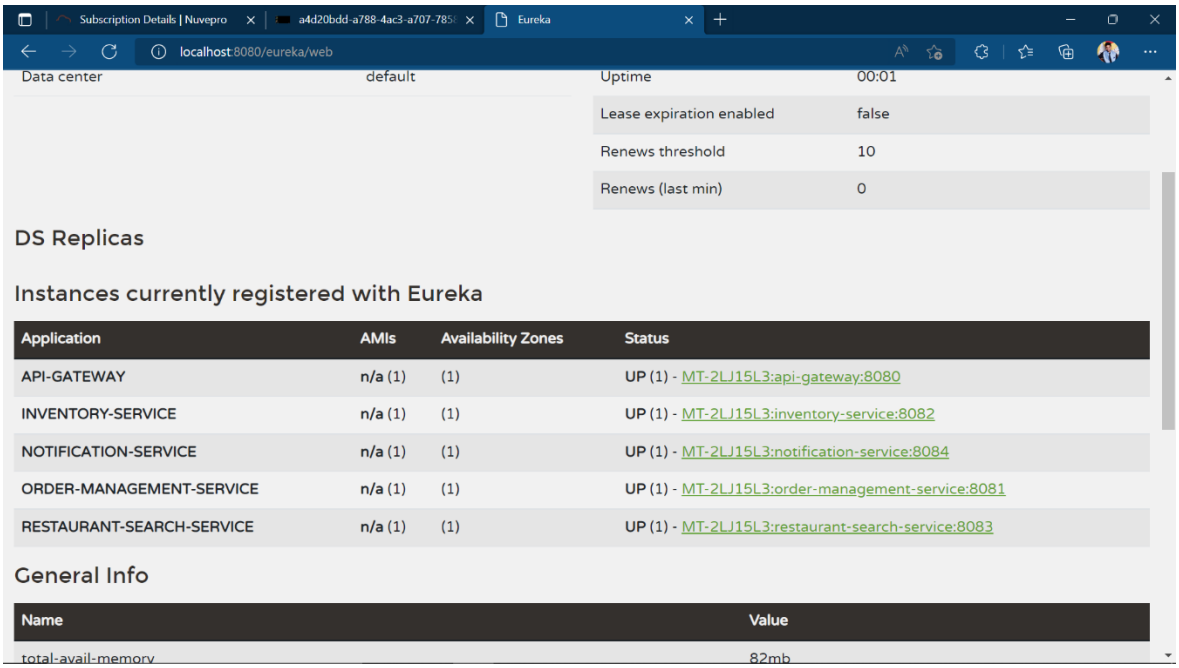
I have created majorly three microservices named

- Restaurant Search Service
- Order Management Service
- Inventory Service

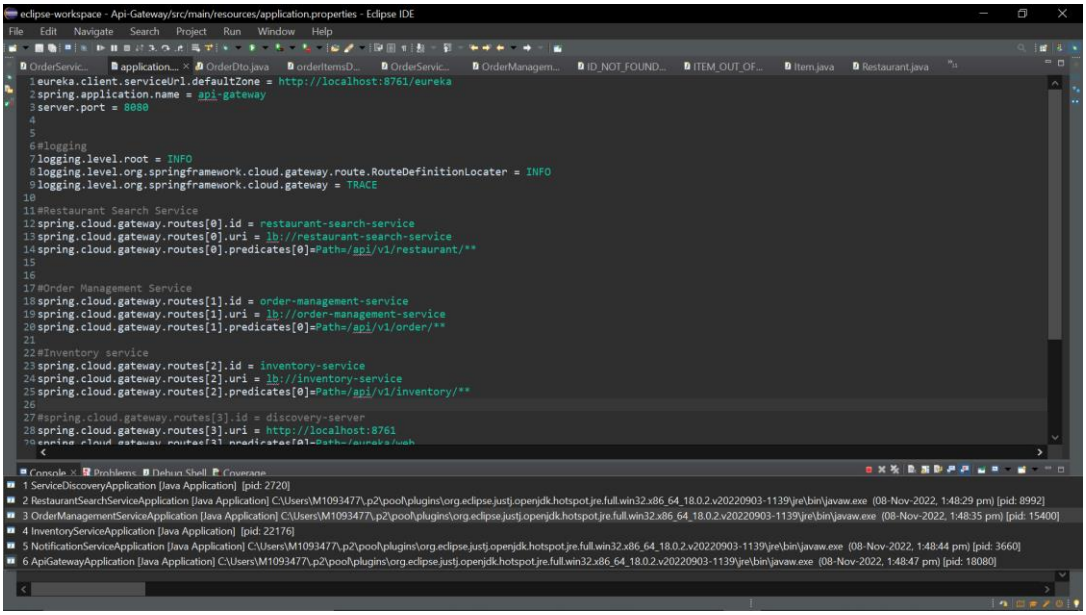
Other supporting services are **Service discovery (Discovery Server Eureka), API-Gateway, Notification- Service**.

Firing all microservices and verify whether all services are discovered by Discovery Server.

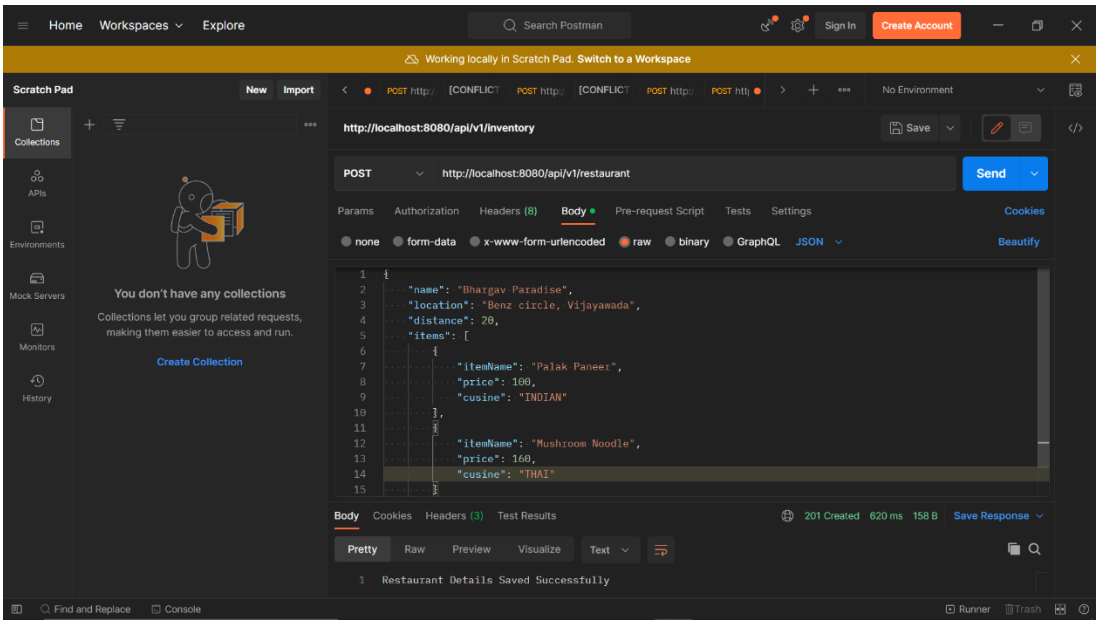


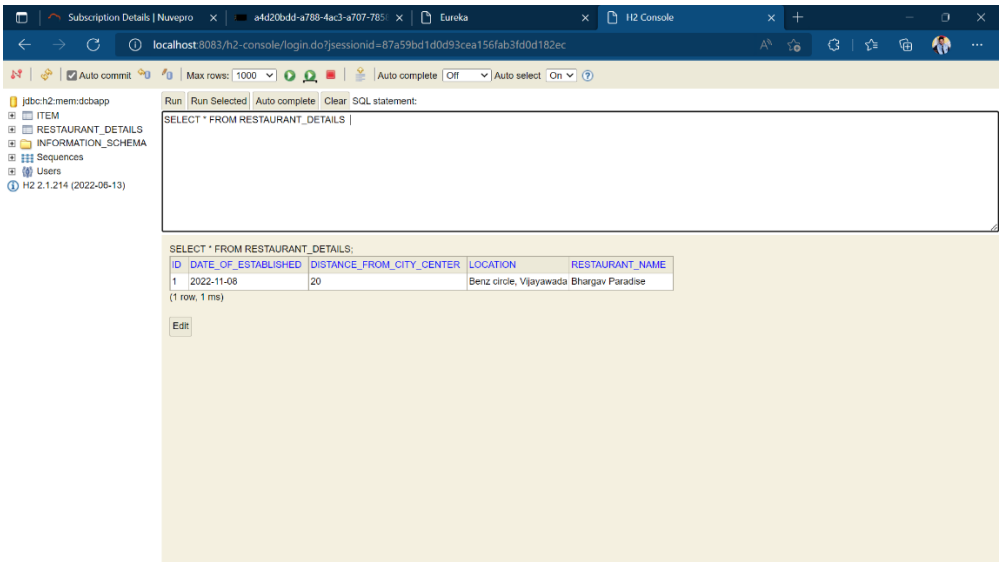


We can see that all services are upon running and can be seen in Eureka discovery server. In the next picture we can see all services are running in eclipse. Api-gateway configuration can be seen to avoid ports because in spring cloud dynamic ports exists such that port will be changing regularly so we need to adopt to it. It also helps us to interact with other microservices without the usage of port.

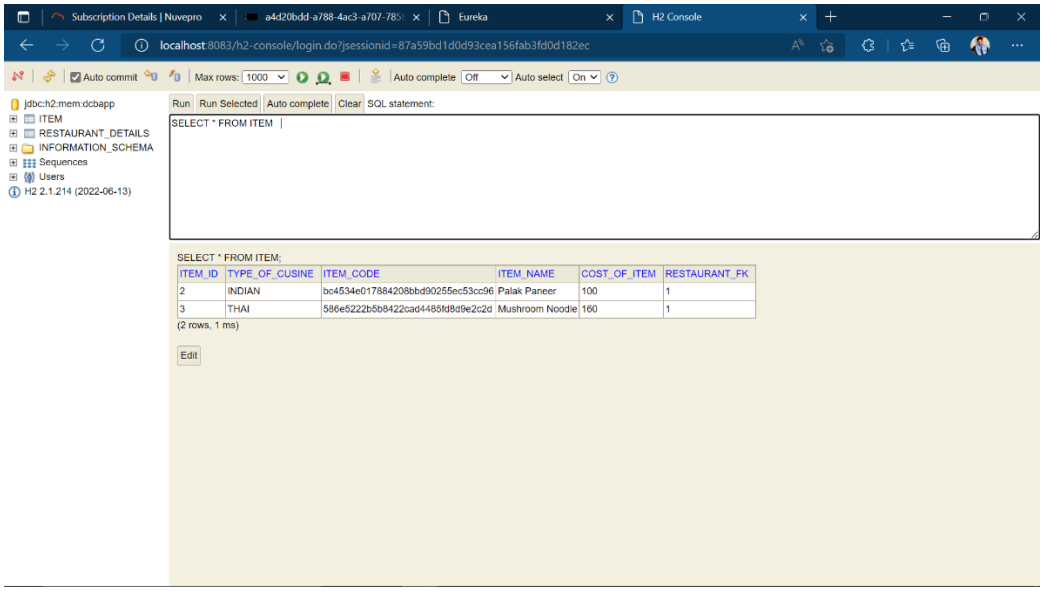


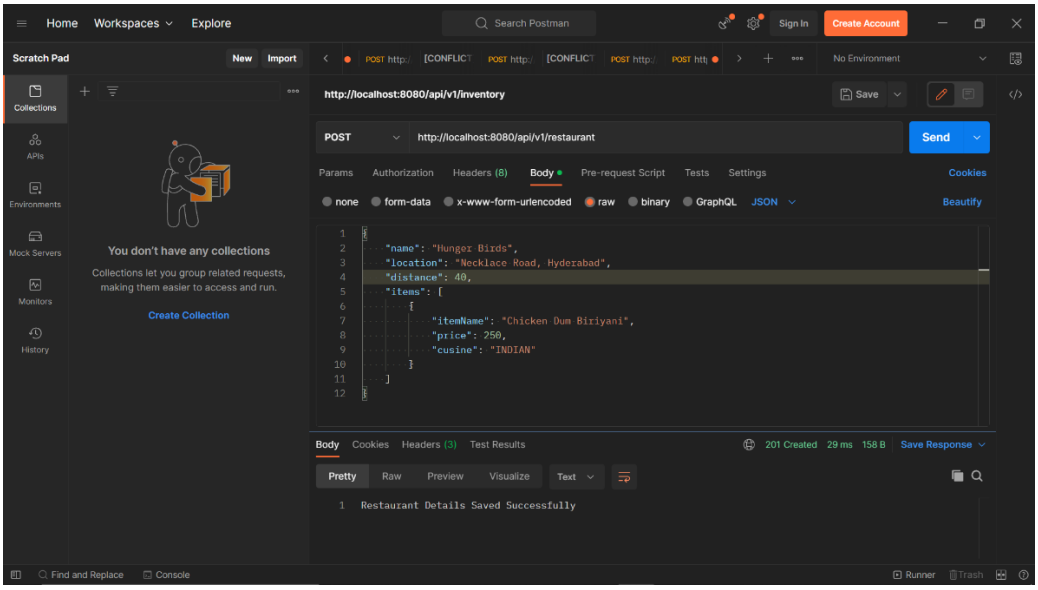
- RESTAURANT – SEARCH – SERVICES walkthrough:
  Adding the restaurant details to the database using postman client. (Here we observe port as 8080 which is api-gateway port which can be used to all microservices)



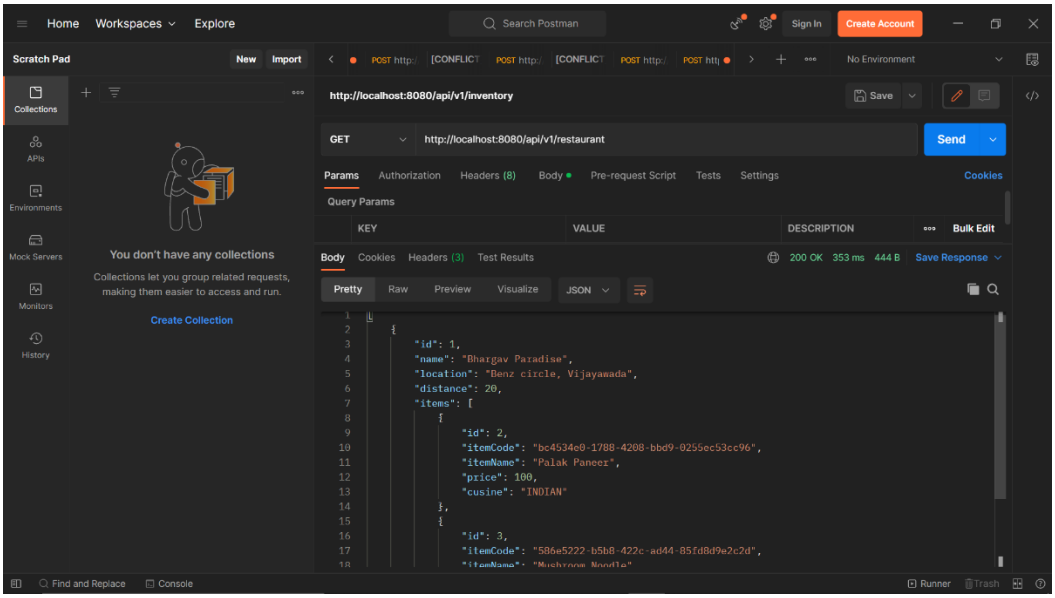


Saved details can be seen in H2 console which is an in-memory database. The actual MySql database is also configured for each microservices but using h2 for easier implementation.

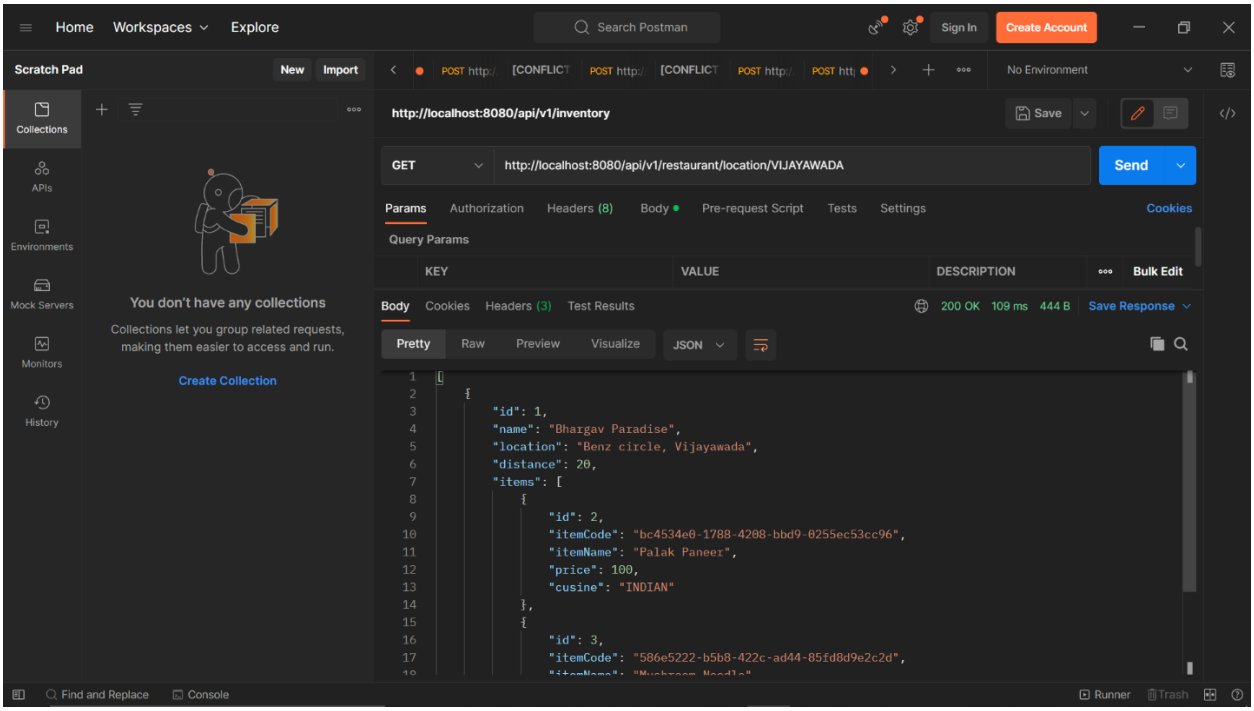




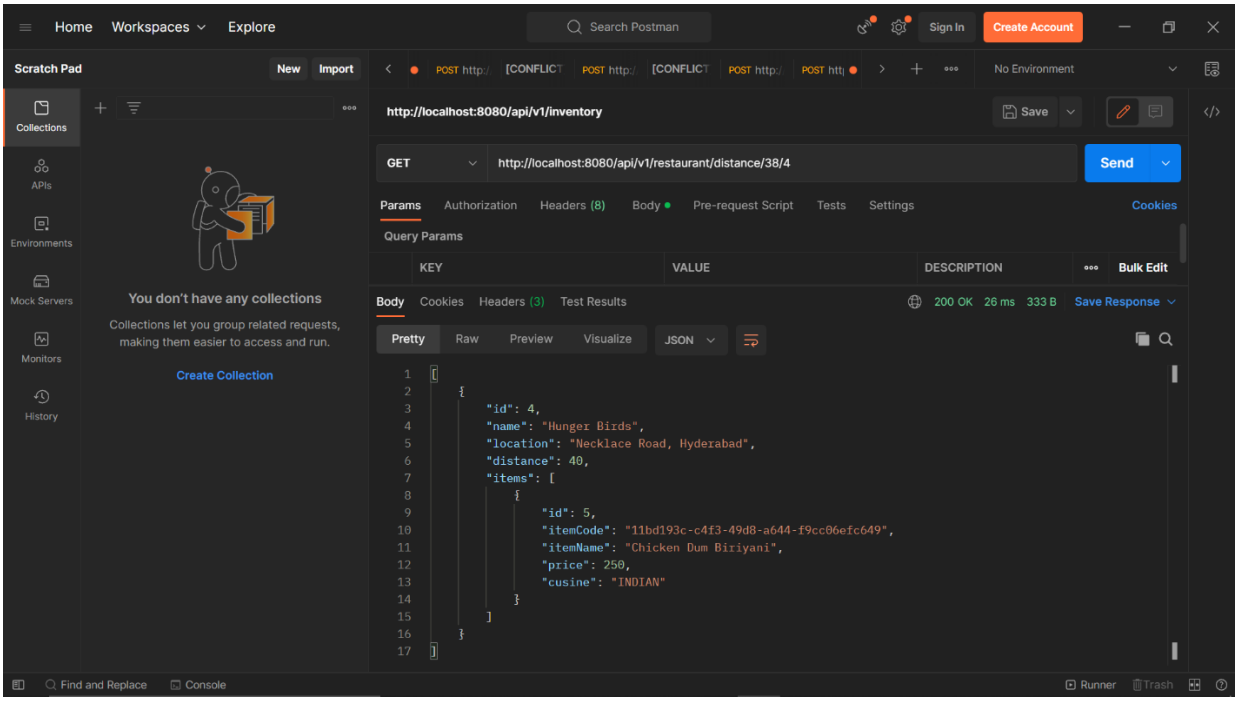
Using a GET method to retrieve all restaurants from the database.



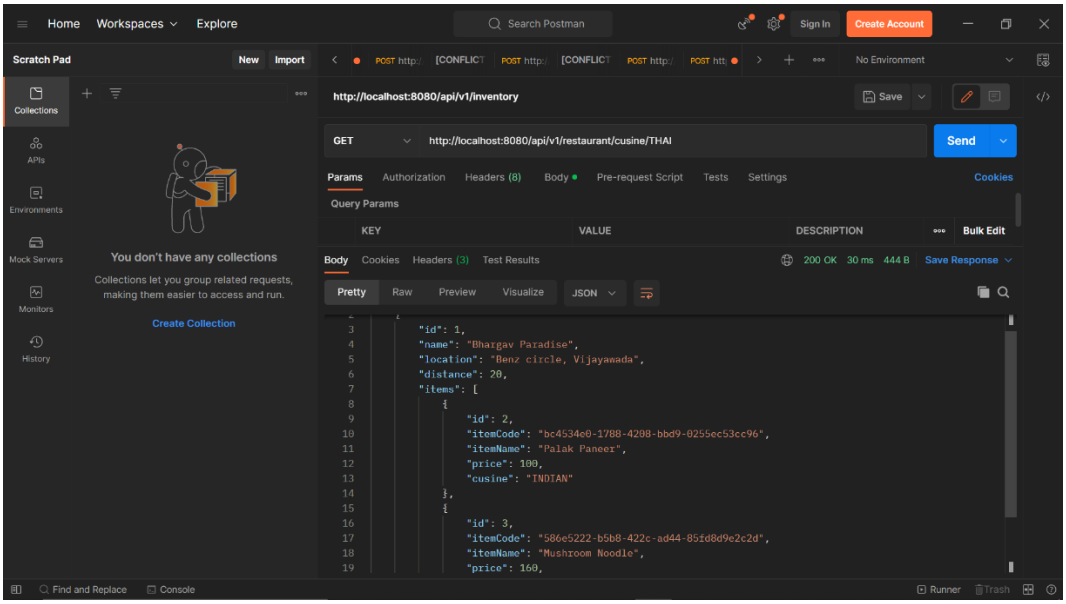
Retrieving restaurants with given location (Here the provided location is case-insensitive such that we can provide location in any case). It returns list of restaurants.



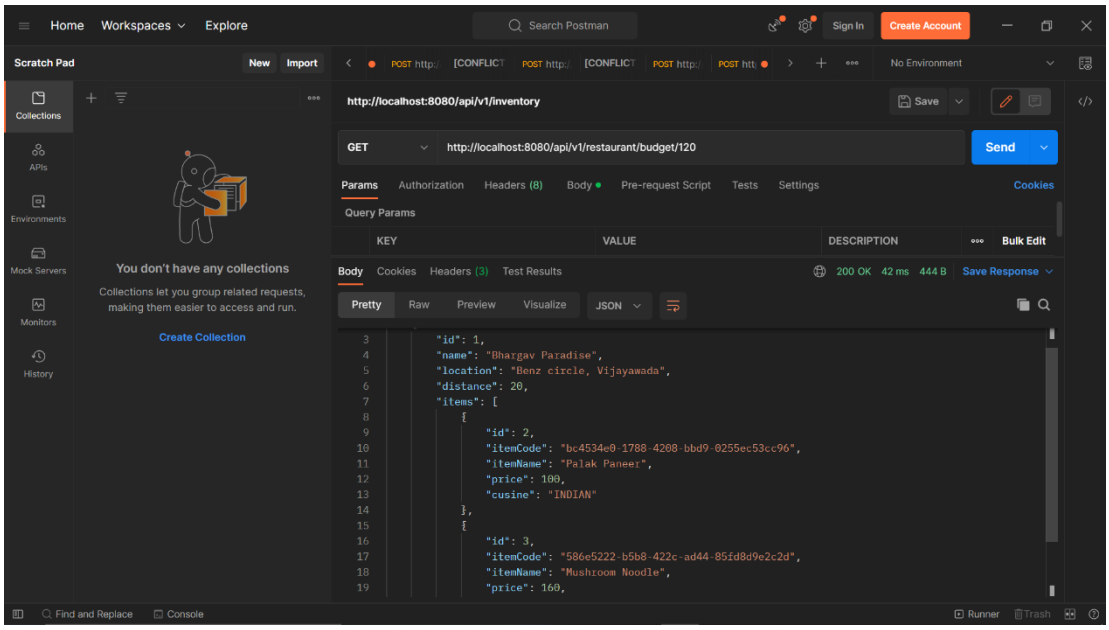
Retrieving restaurant based on distance (Here we are proving the distance from the city you are located and the distance under you want the restaurants). It returns list of restaurants.



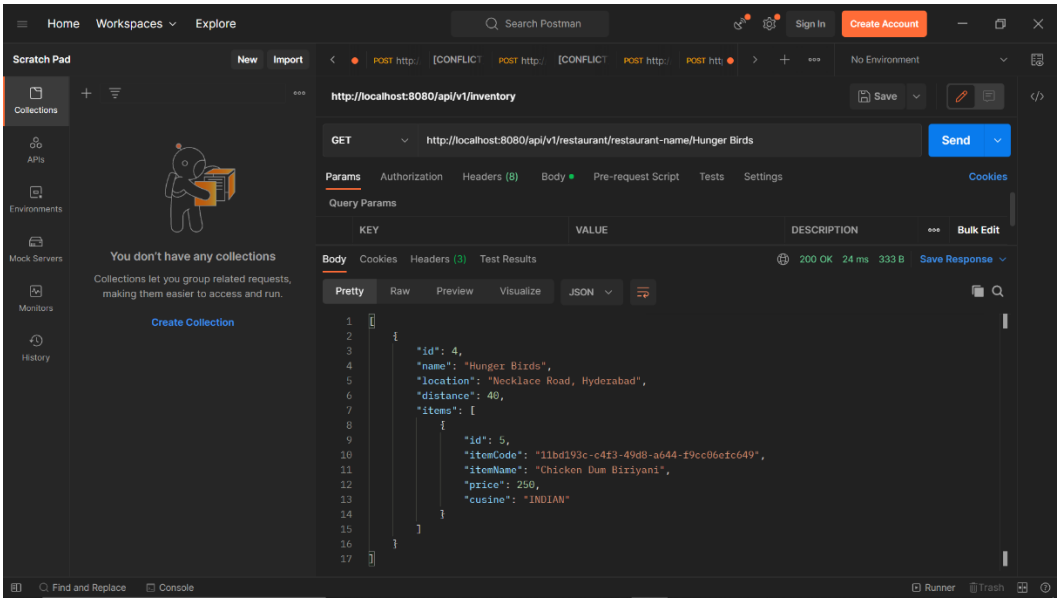
The picture below shows retrieving the restaurant details based on Cusine they provided. Here we provided THAI as cusine then it retrives all the restaurant that provided THAI food as cusine.



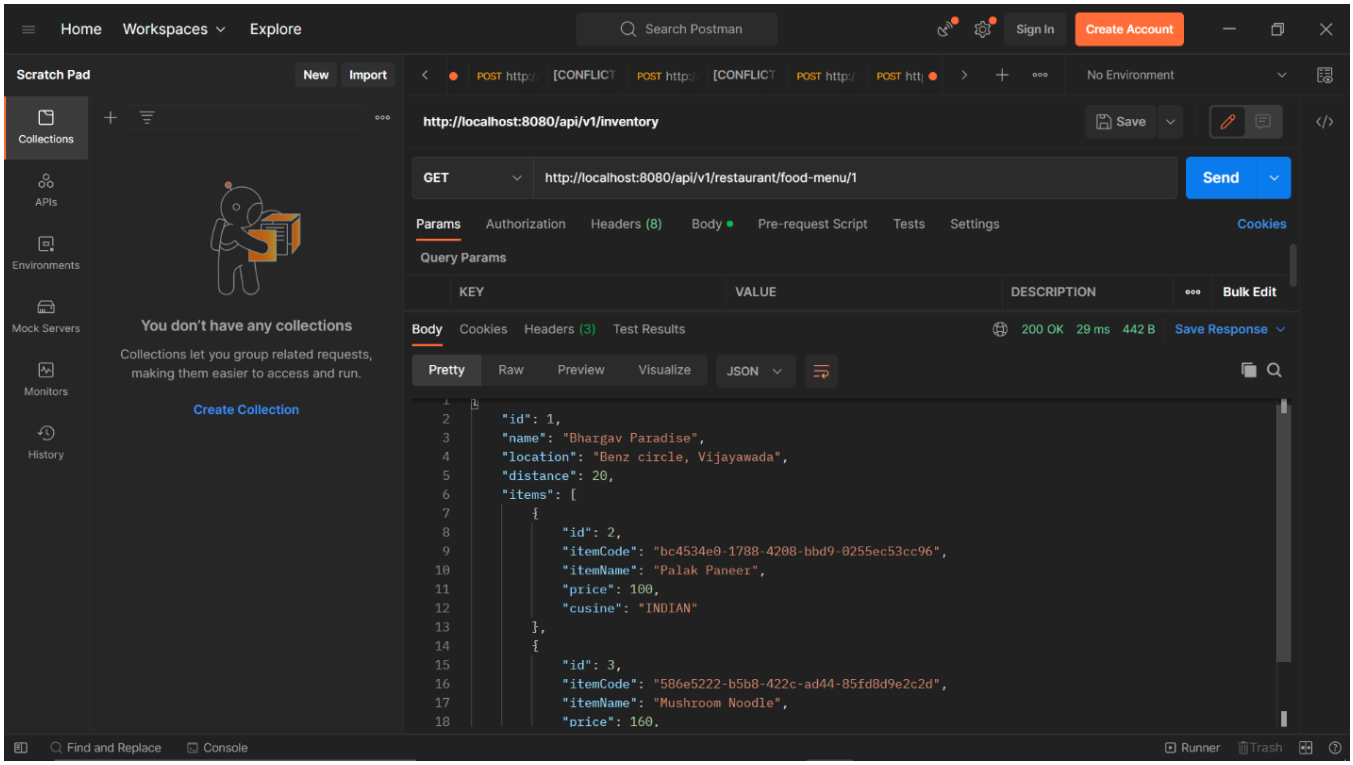
Retrieving the restaurants based on your budget. We pass the budget and it checks all the restaurant for minimum price retrieves all the restaurant that are in our provided budget.



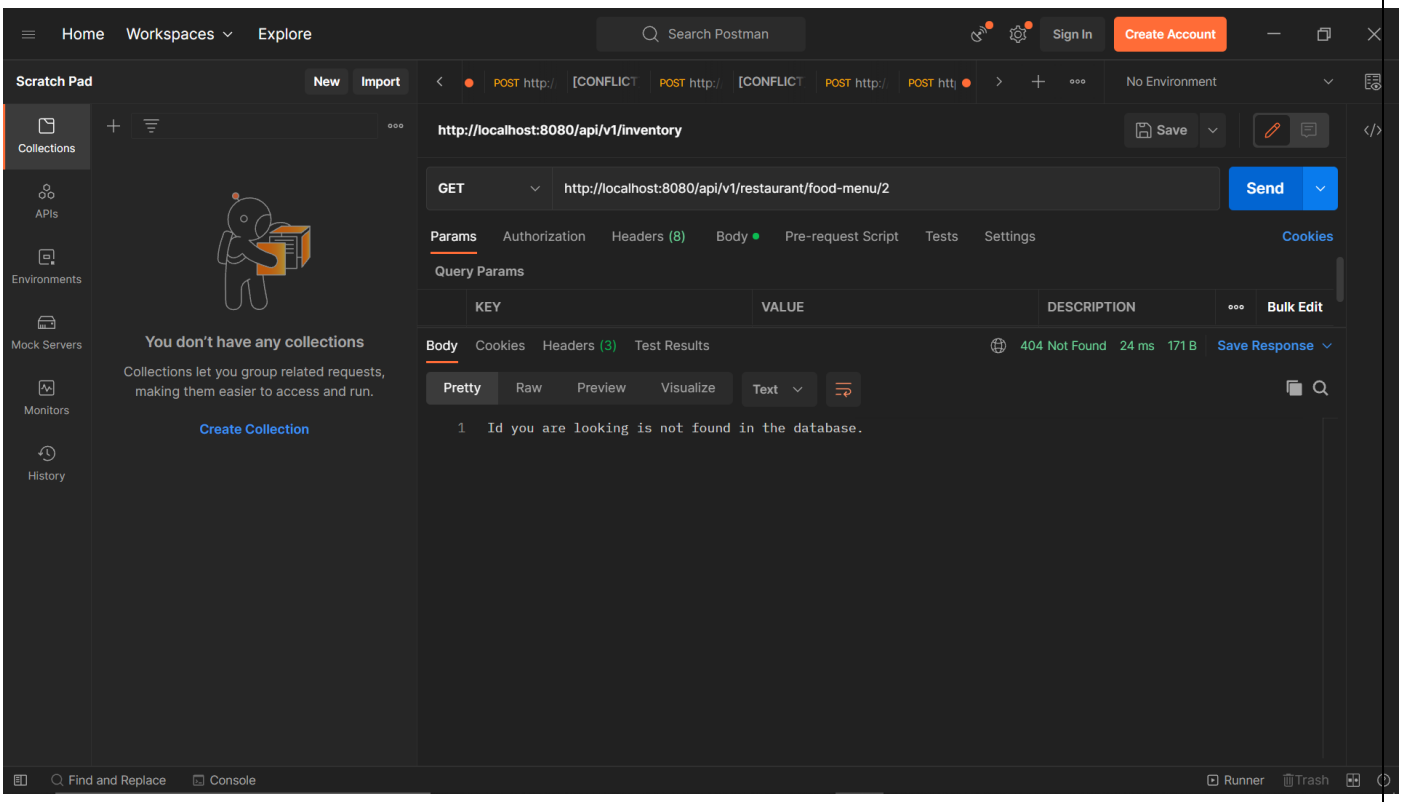
Retrieving the restaurant based on the name provided.



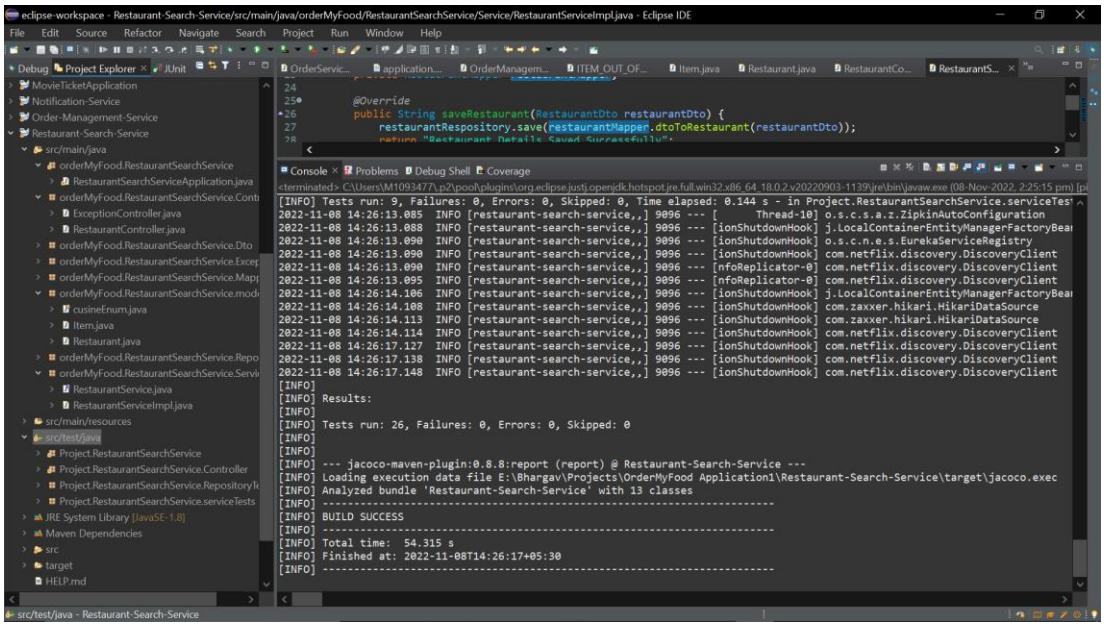
Viewing the food menu based on the id provide for the restaurant



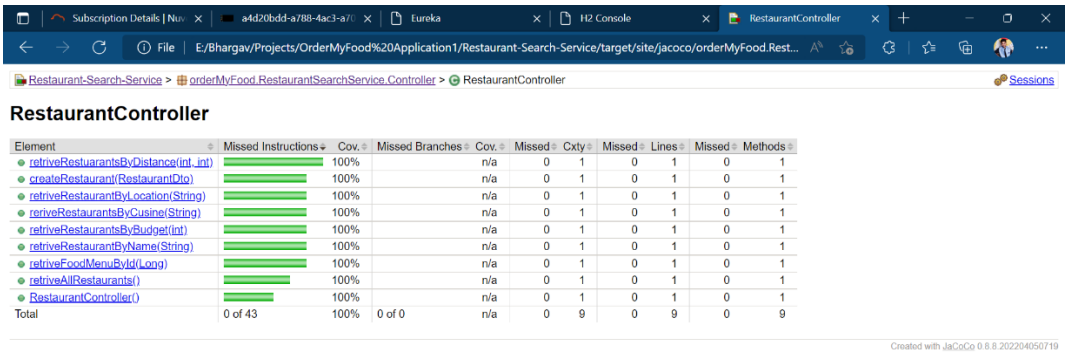
If we provided the id which is not in the database it throws 404 and return customized message.



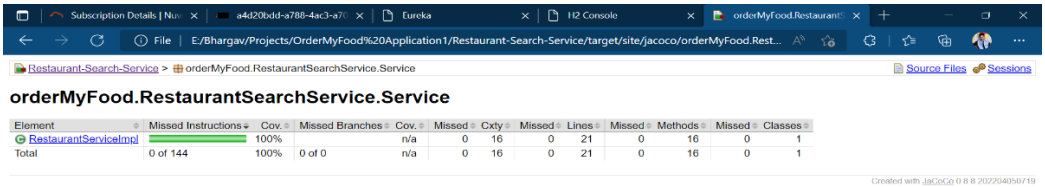
Running the maven-test command to generate the Jacoco code coverage report.



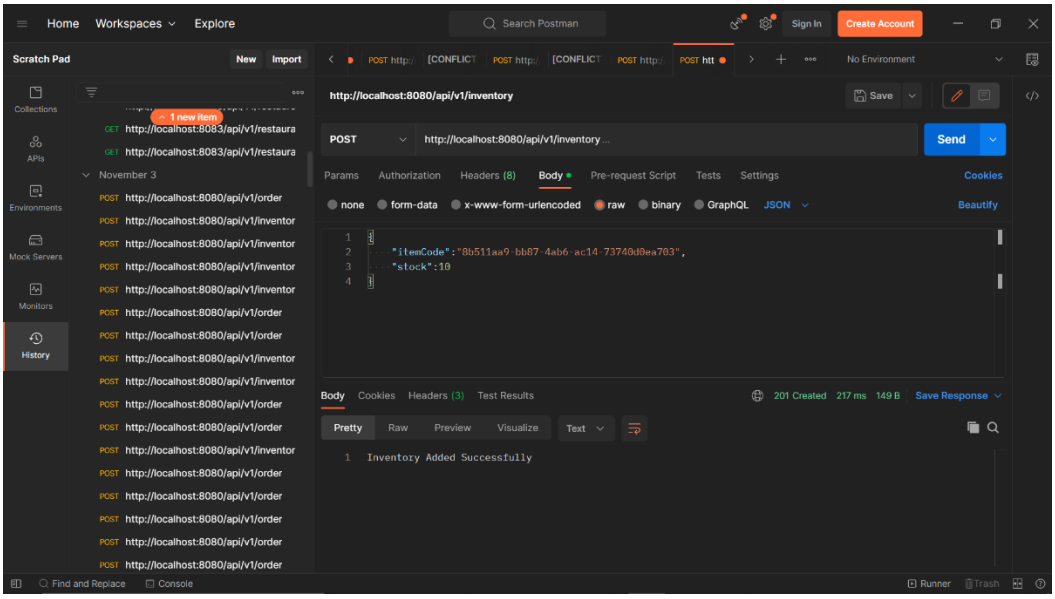
Jacoco code coverage report for Restaurant – service - Controller

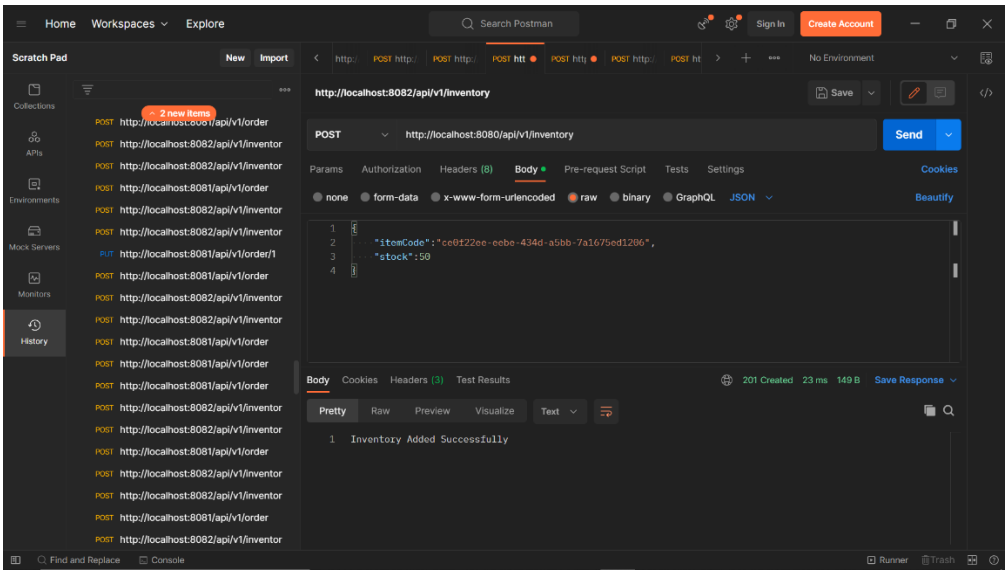


Jacoco Service layer report

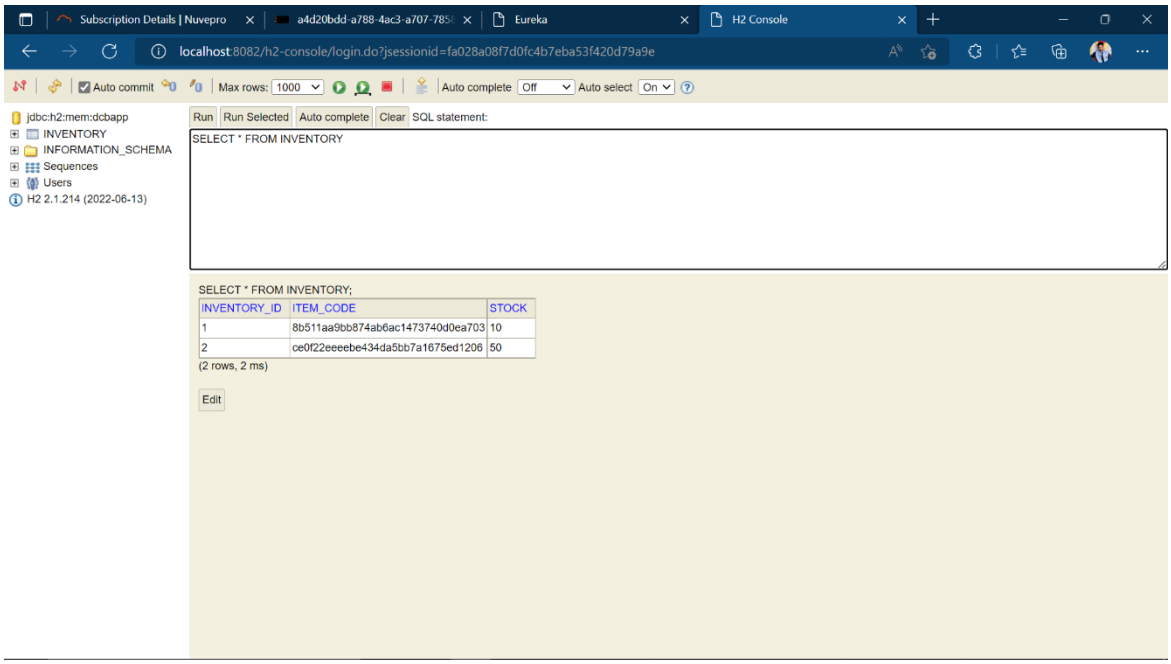


- INVENTORY – SERVICE – Walkthrough:
  Adding inventory into the database

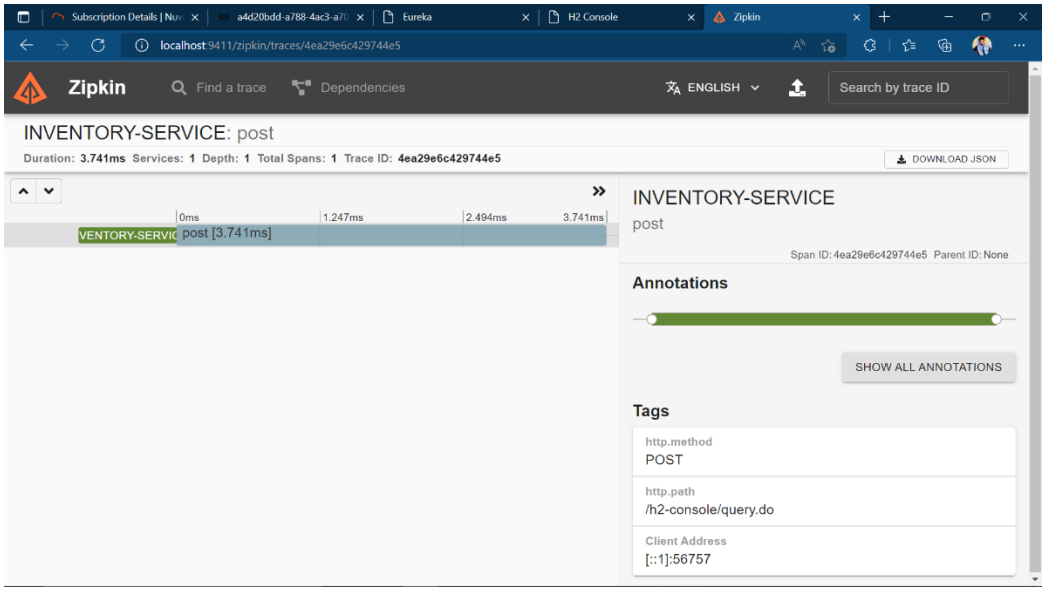




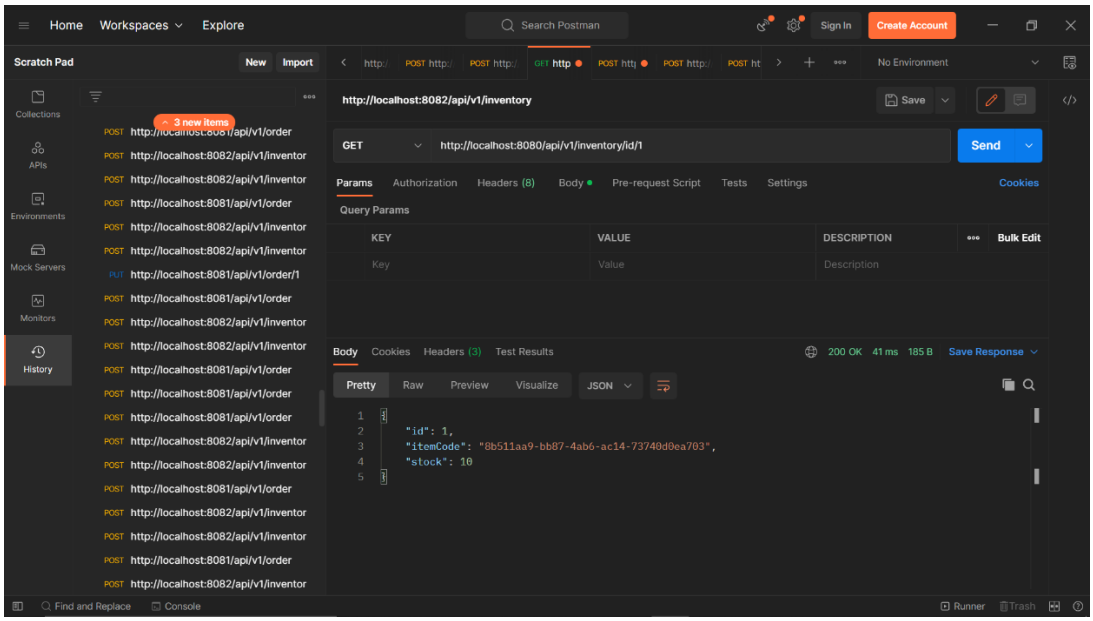
Checking the inserted data in H2 console



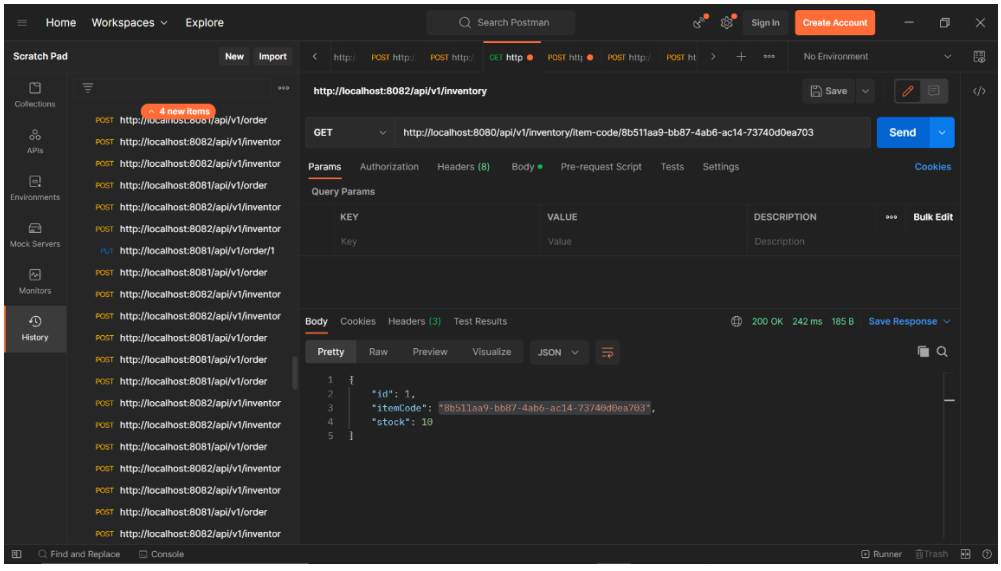
Checking the Spans and traces in Zipkin for Inventory service. We observe a post request that is recorded.



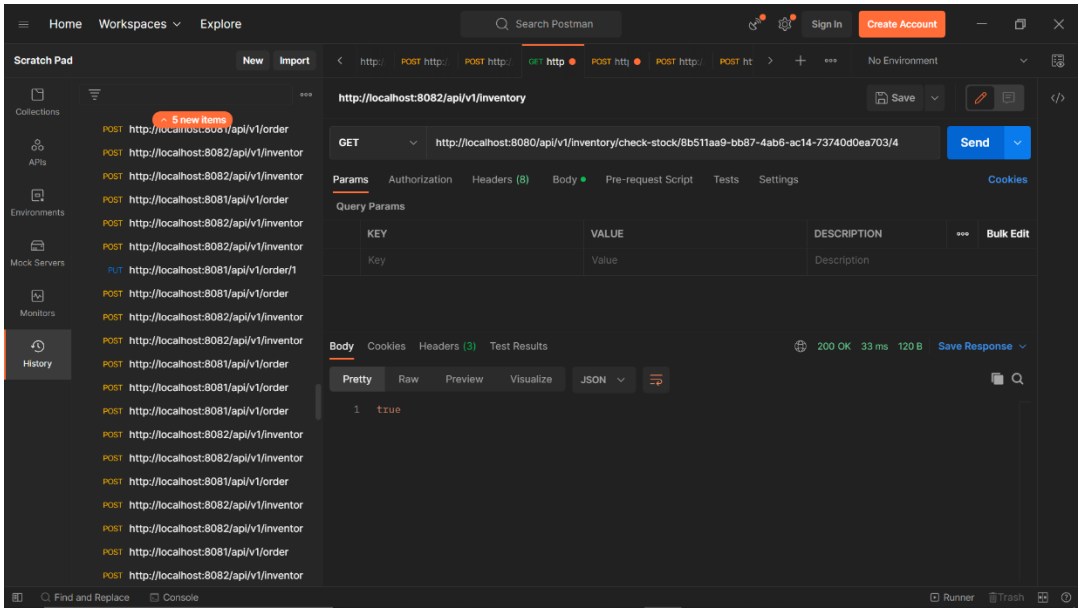
Retrieving the inventory using id



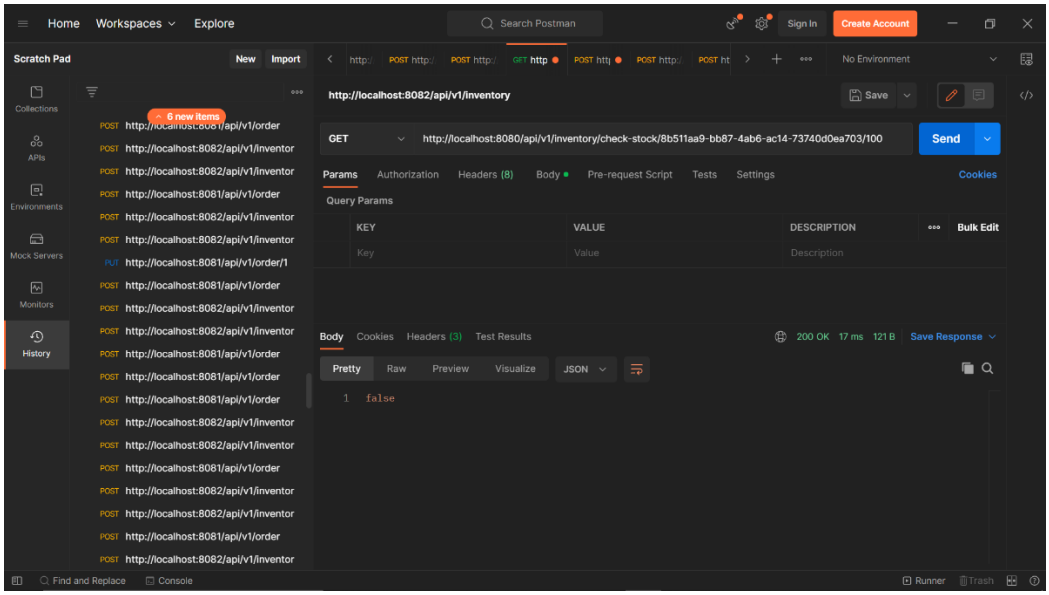
Retrieving the inventory using item-code



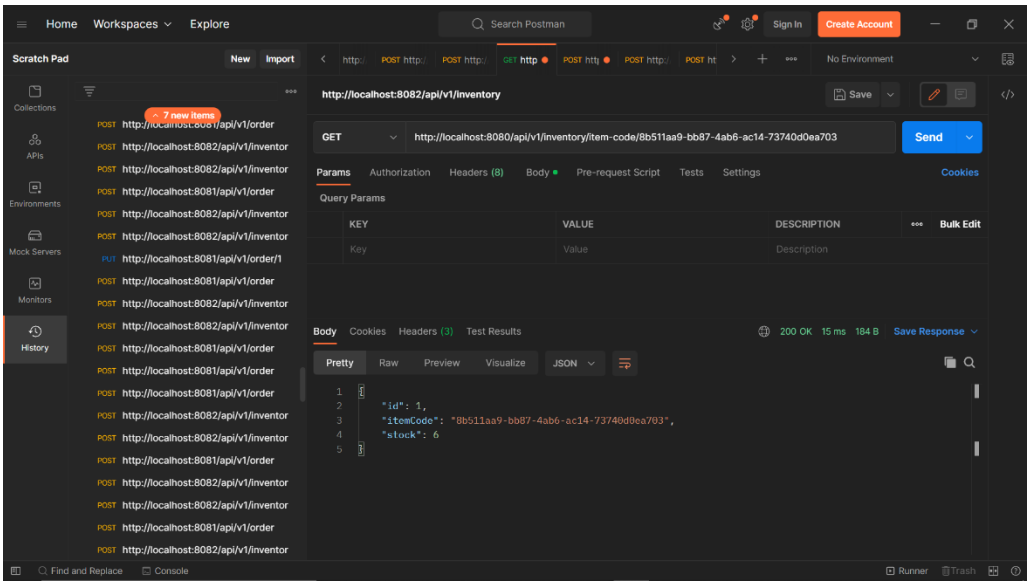
Checking the item is in stock for the provided quantity if yes returns true else false.



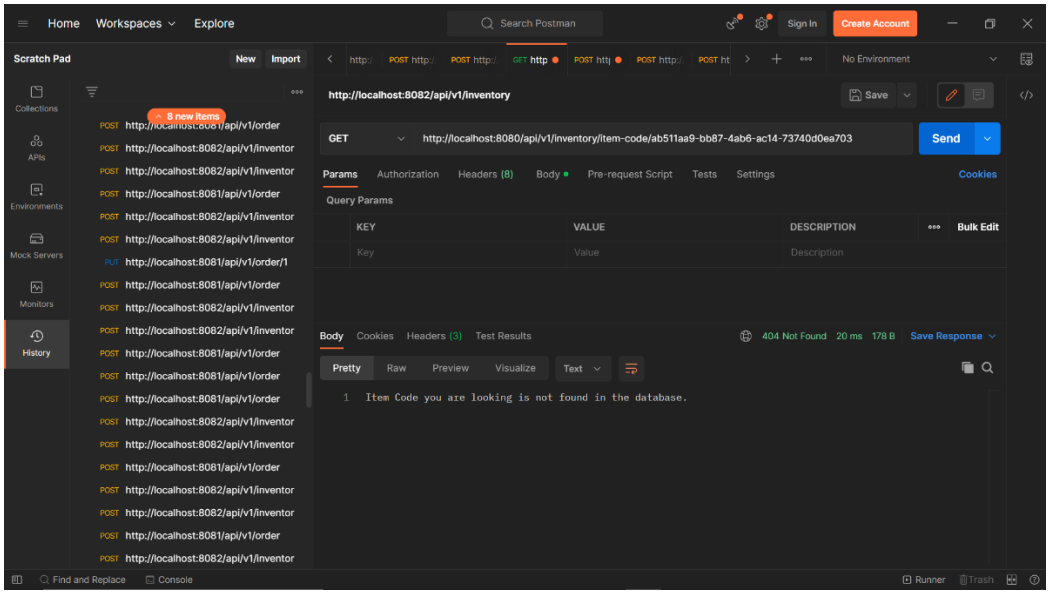
If item quantity required is more than the items have then it returns false.



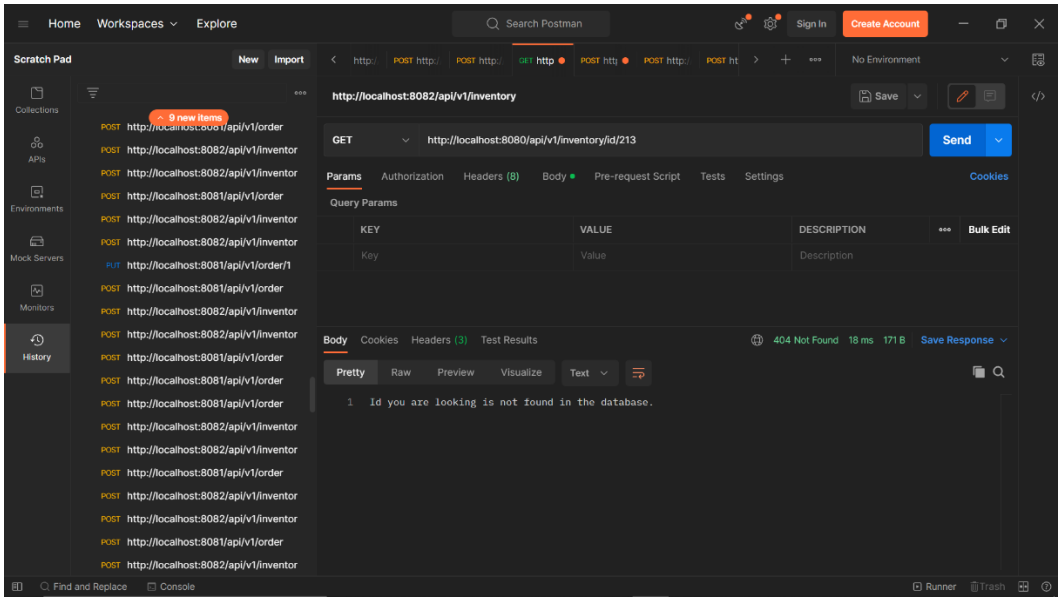
If item is in stock it updates the stock (here from 10 to 6)



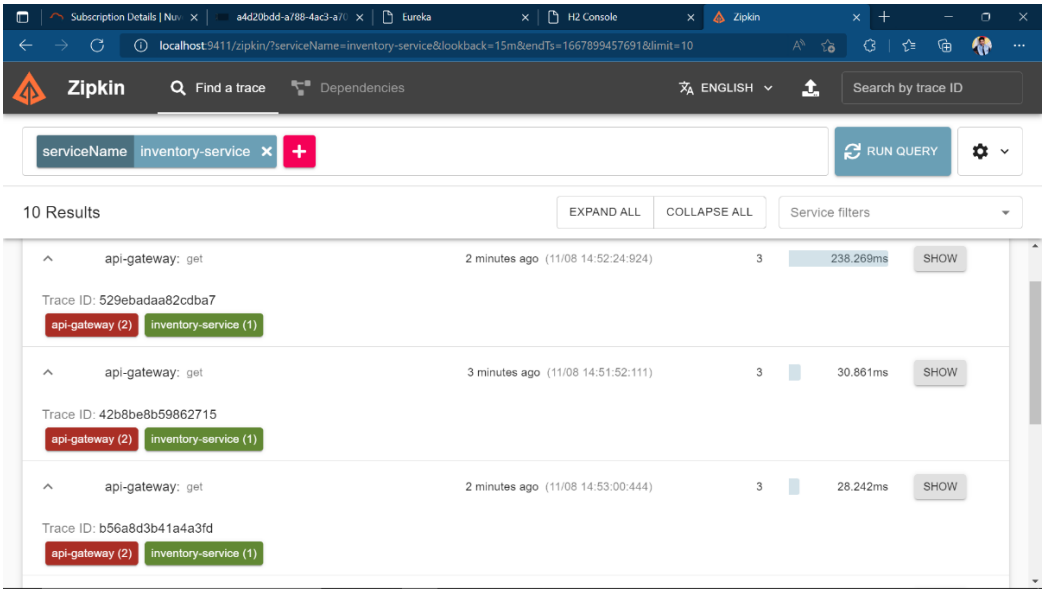
If you entered an invalid item code then it throws 404 and returns customized error message



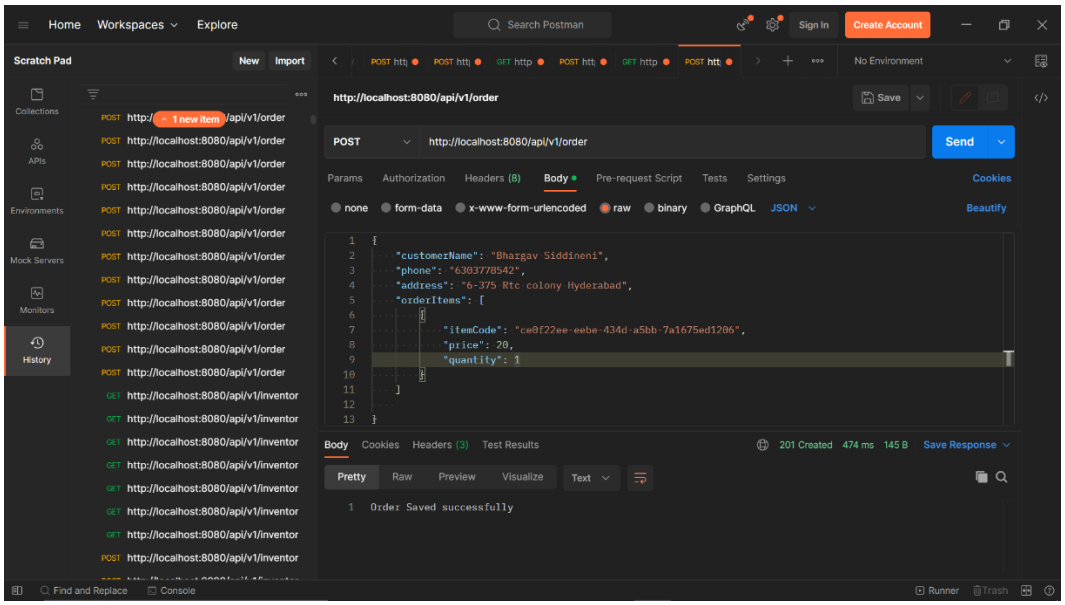
While searching based on id, if we provide wrong id then it throws 404 and returns customized error message



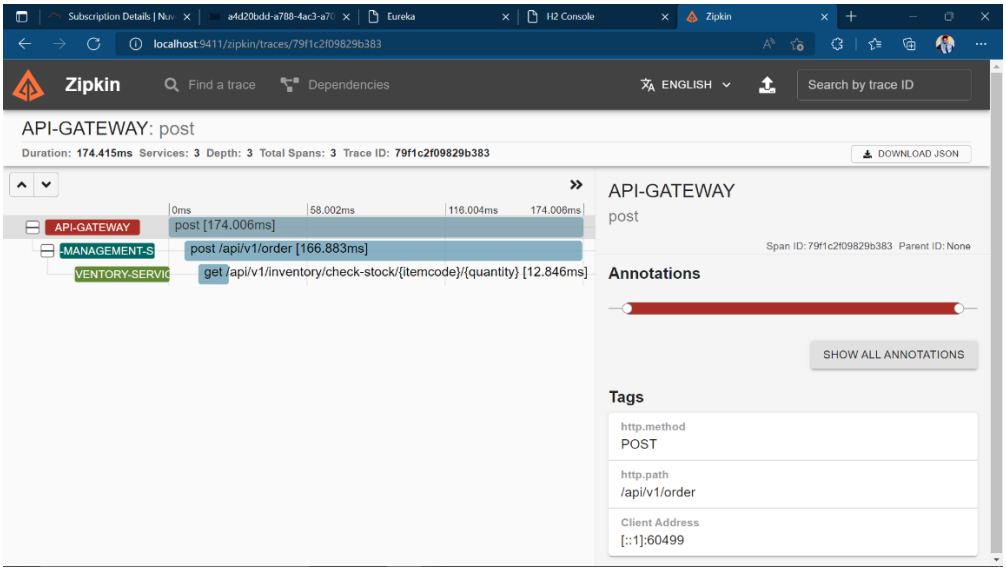
We can see the tracing of api in zipkin in the below picture.



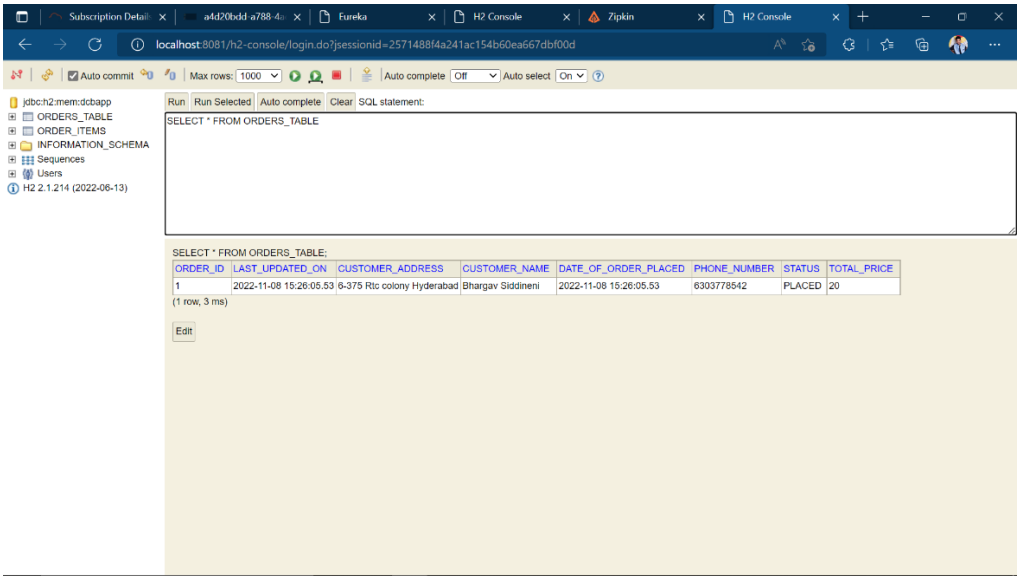
- ORDER -MANAGEMENT – SERVICE walkthrough:
  Placing the order which saved the data to database. It calls inventory service and checks for the stocks using rest template.

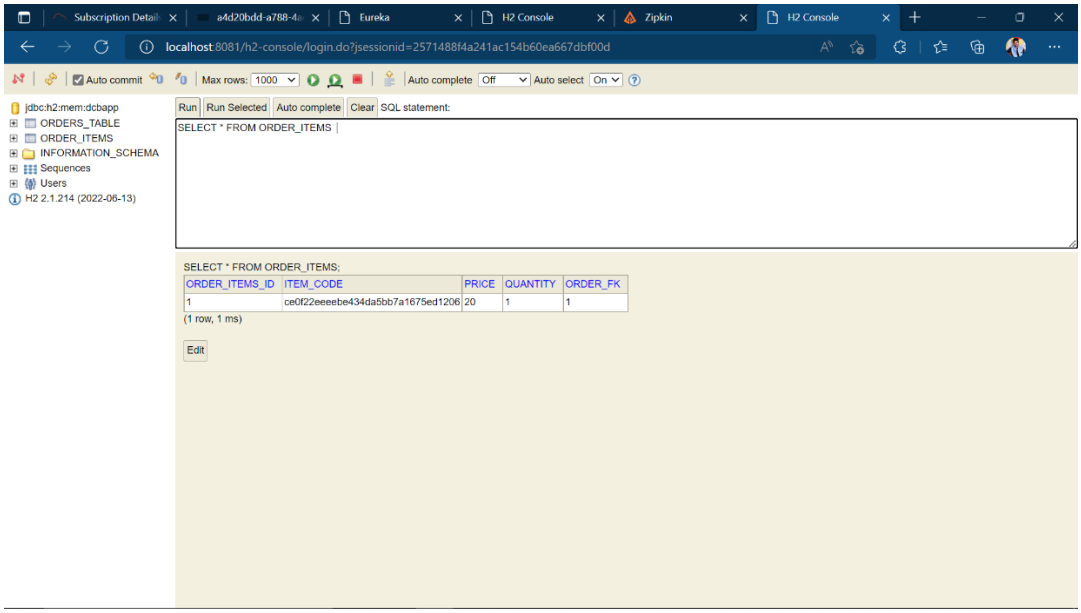


We can observe the tracing of post request which involves two other microservices.

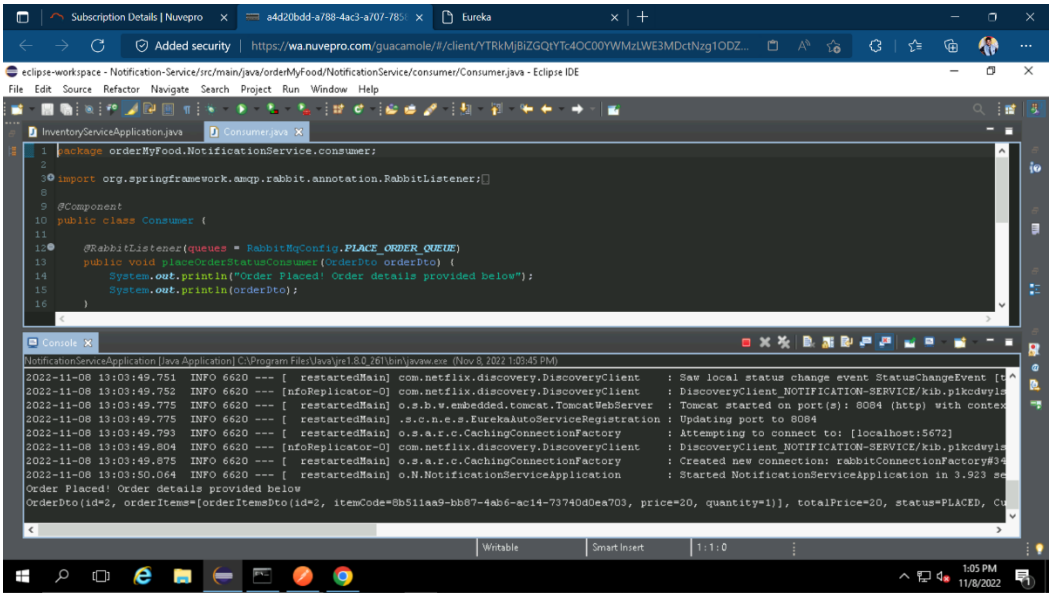


We can see the order details in the database.

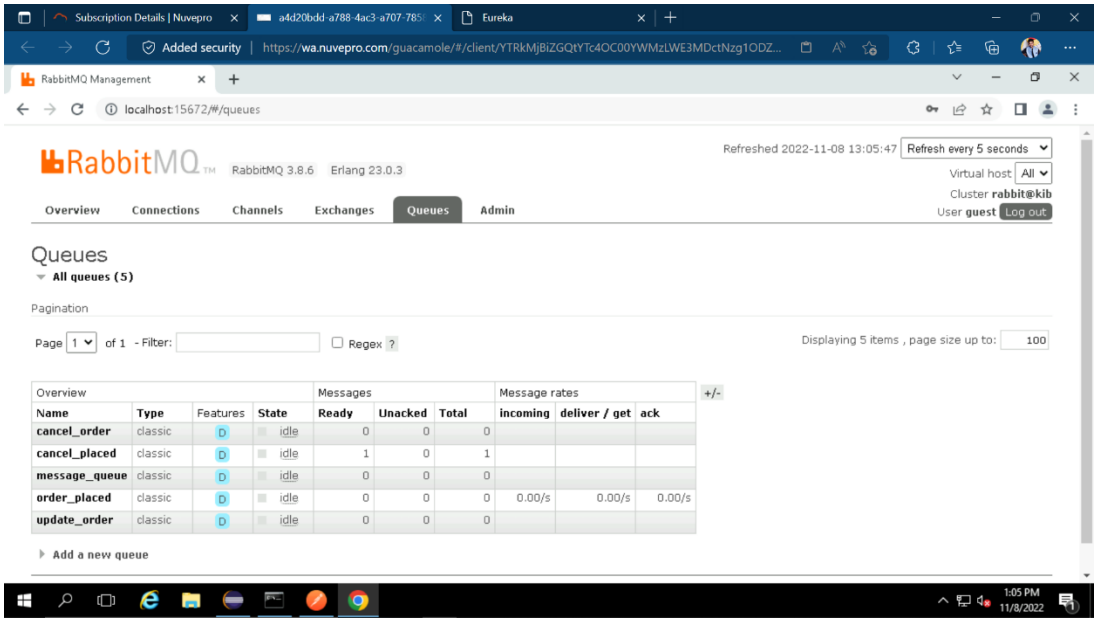




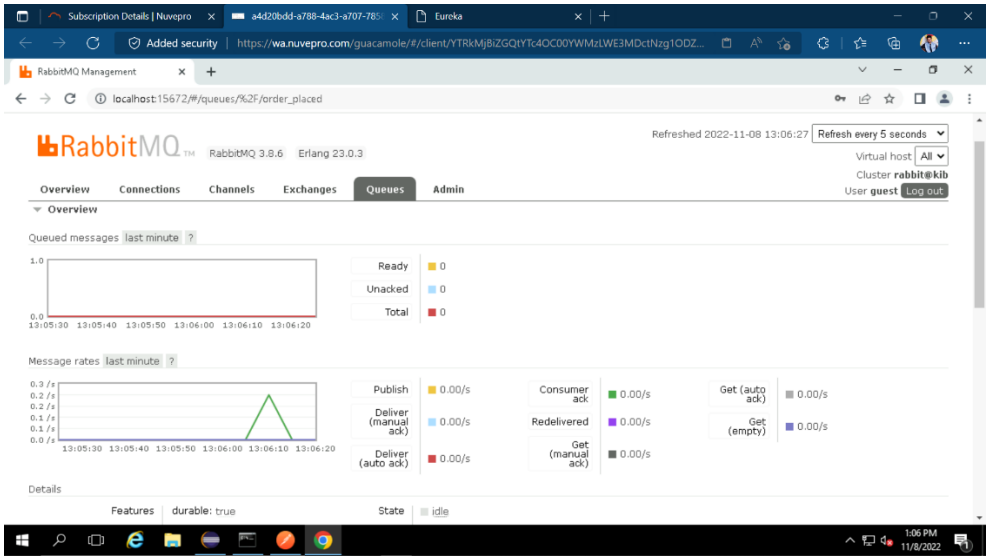
After order is placed using the rabbit Template from Rabbit MQ broker we send a notification along with order details to Notification service through PLACE_ORDER_QUEUE.



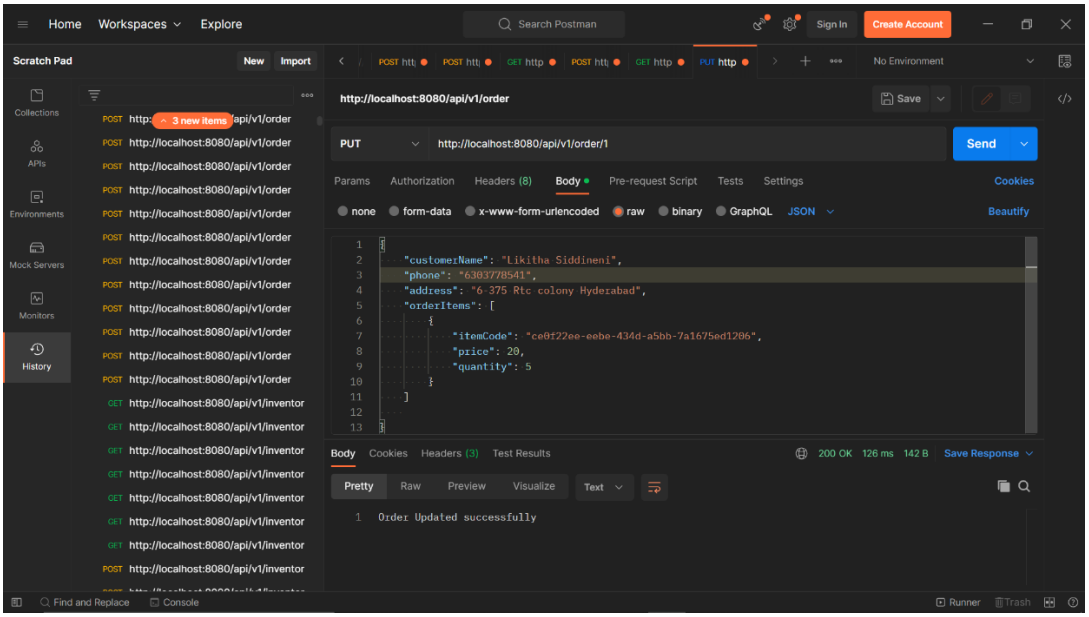
All queues that are created can be seen in Rabbit Mq UI at port 15672.



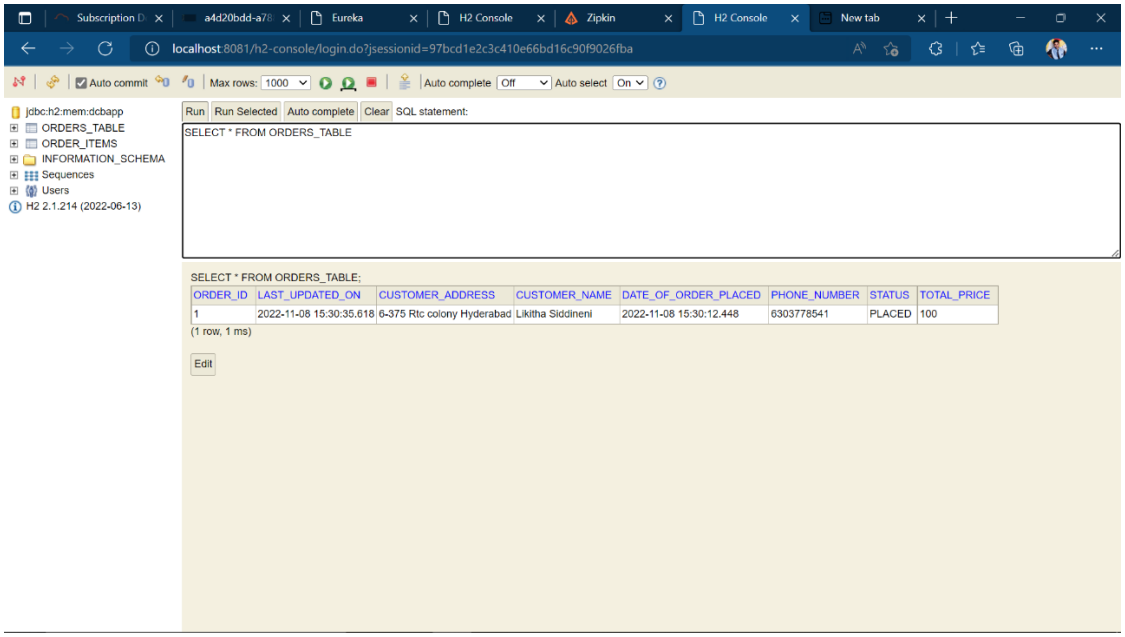
We can see graphical the message transferred from order service to notification service when an order is placed.



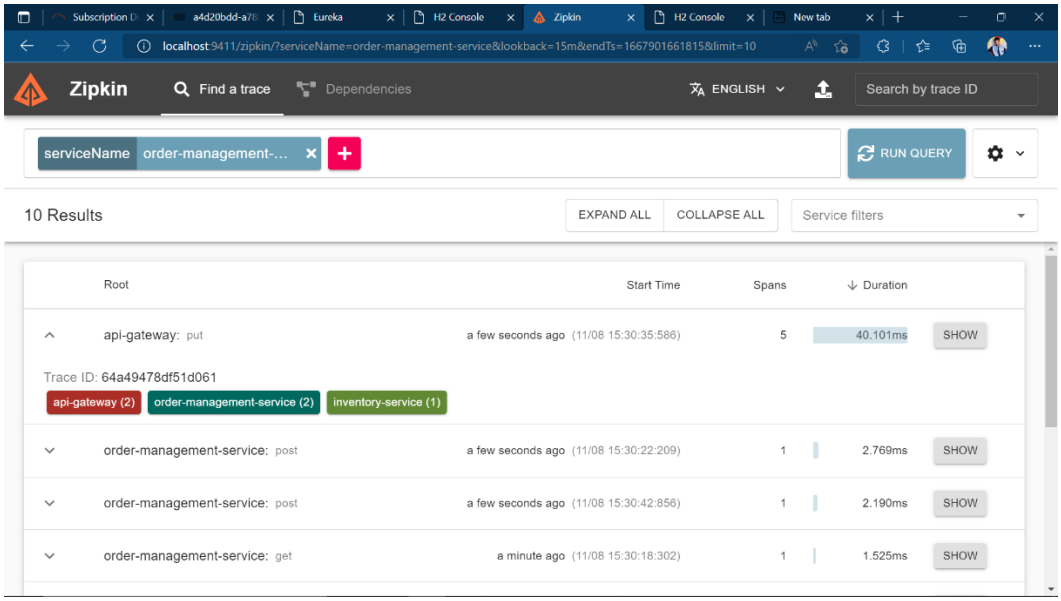
Updating the data that is present in the database.



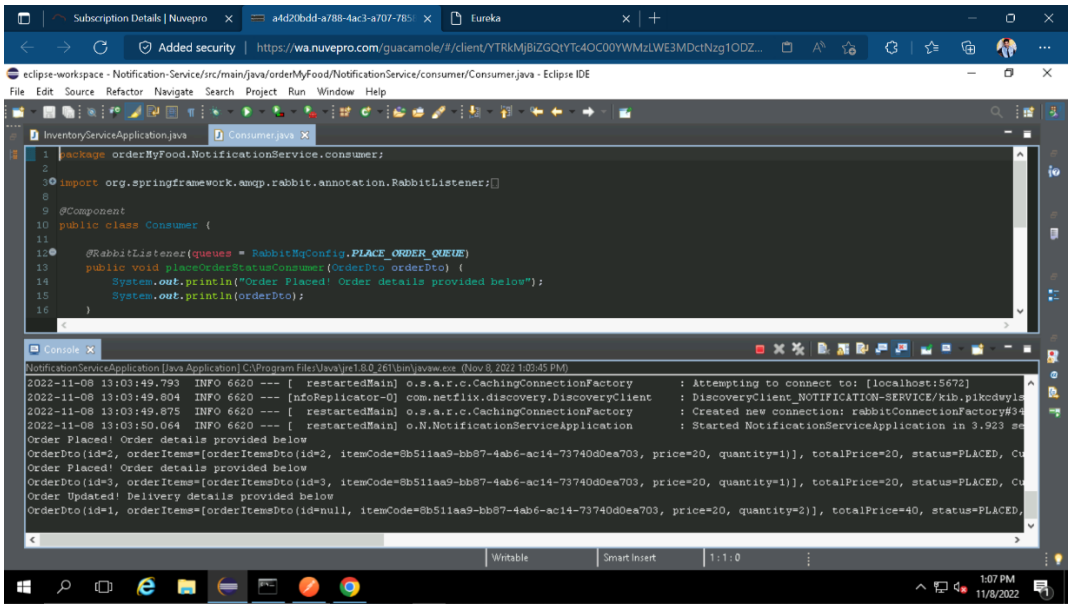
Checking the updated data in the database



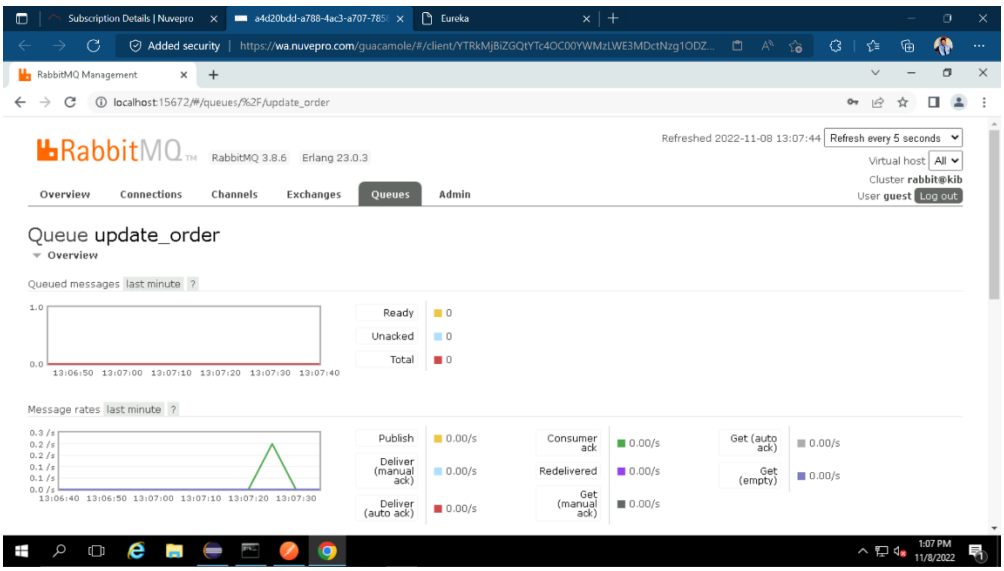
Checking the trace after executing put request in zipkin.



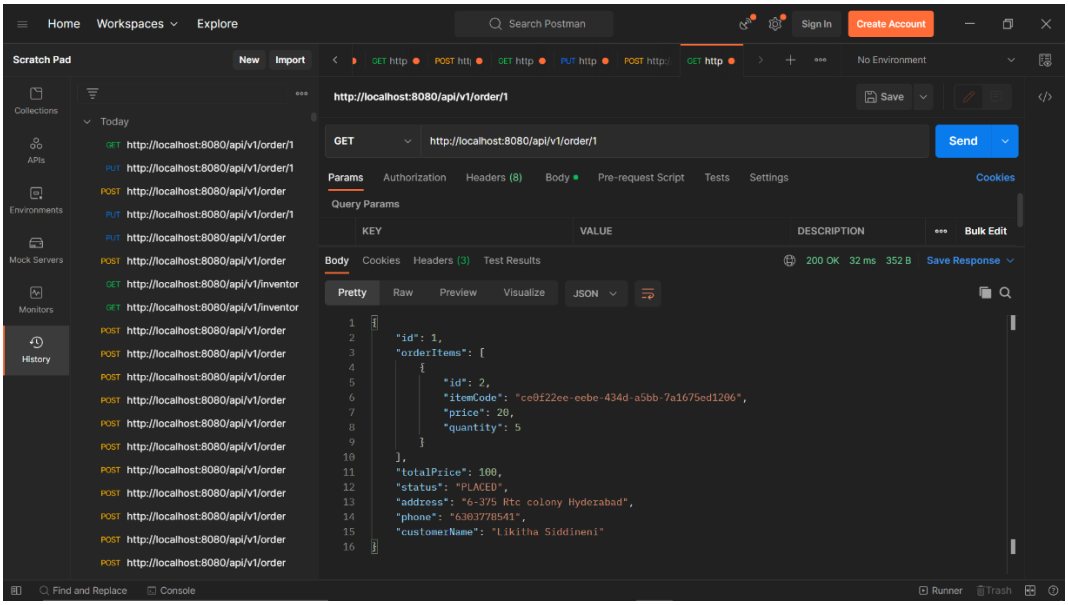
A notification is sent to Notification service from order Service regarding the update of order along with the updated order details using rabbit MQ broker. We can see it in the console.



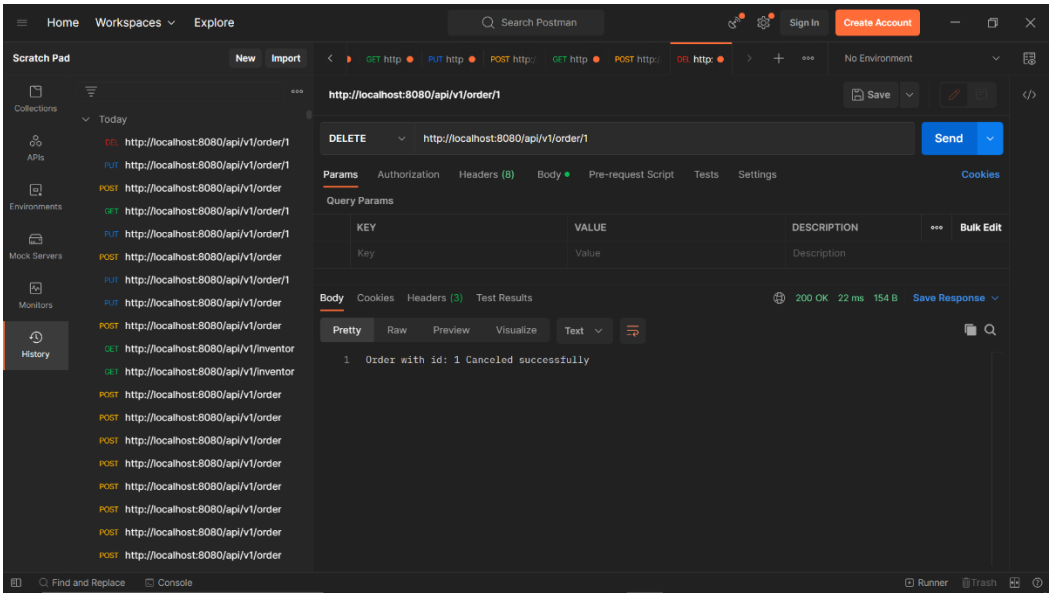
We can see graphical the message and time of transfer in Rabbit MQ UI regarding update order.



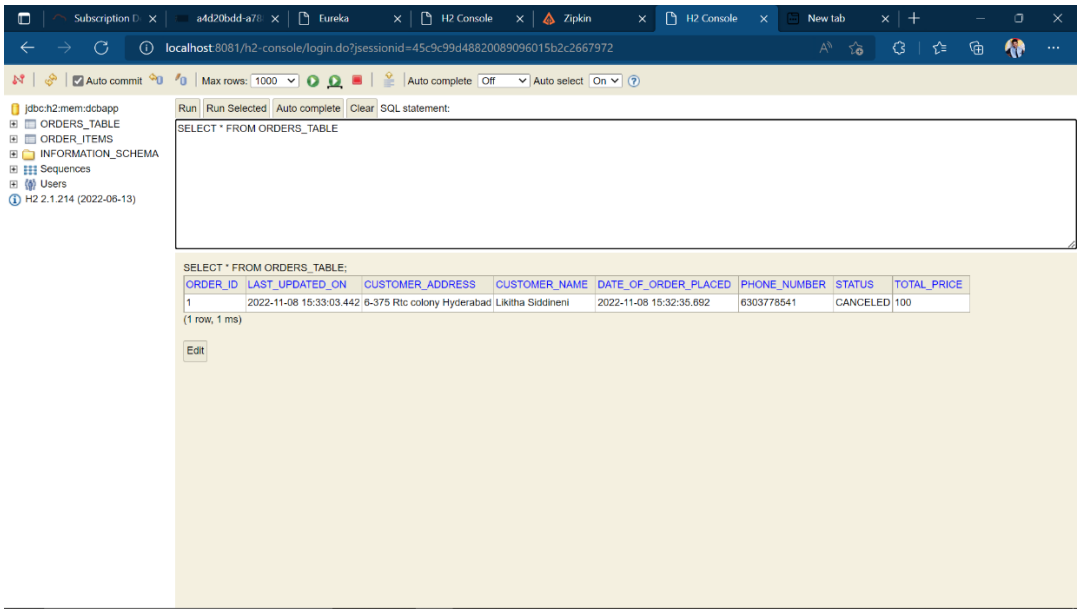
Retrieving the order based on id



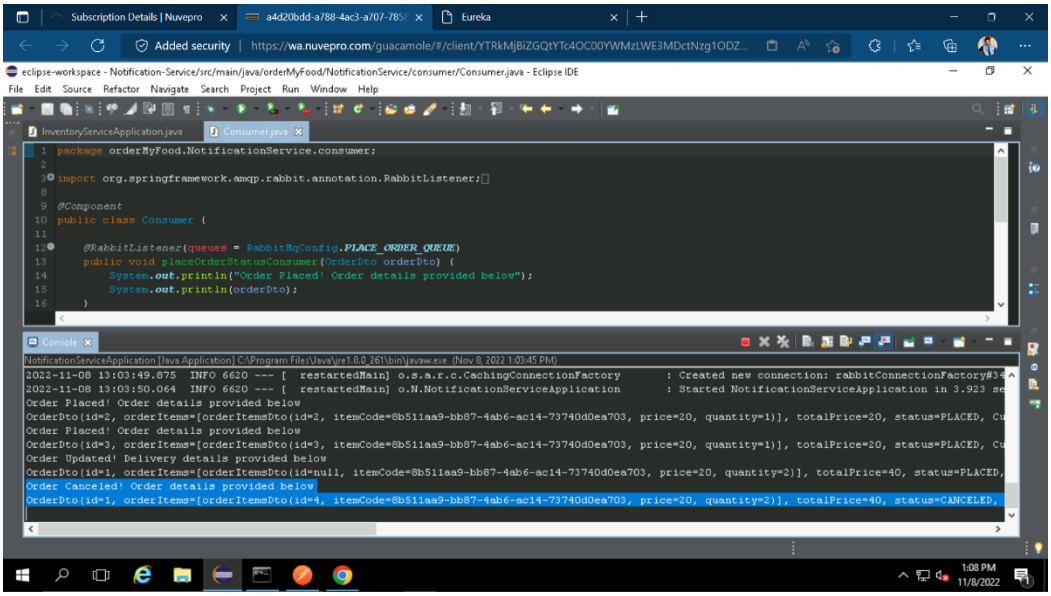
Cancelling an order



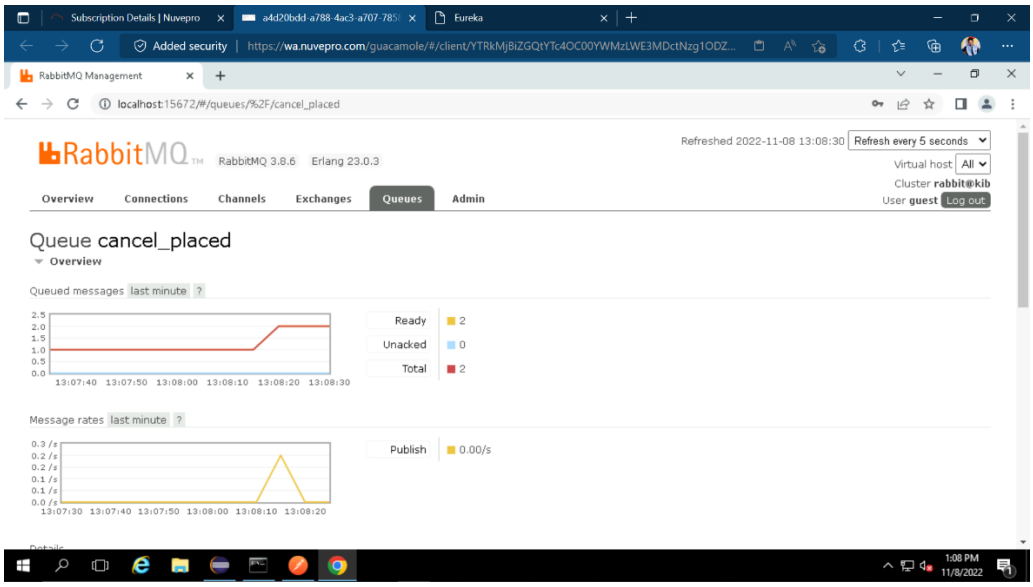
Checking the modified status. It is changed to Canceld.



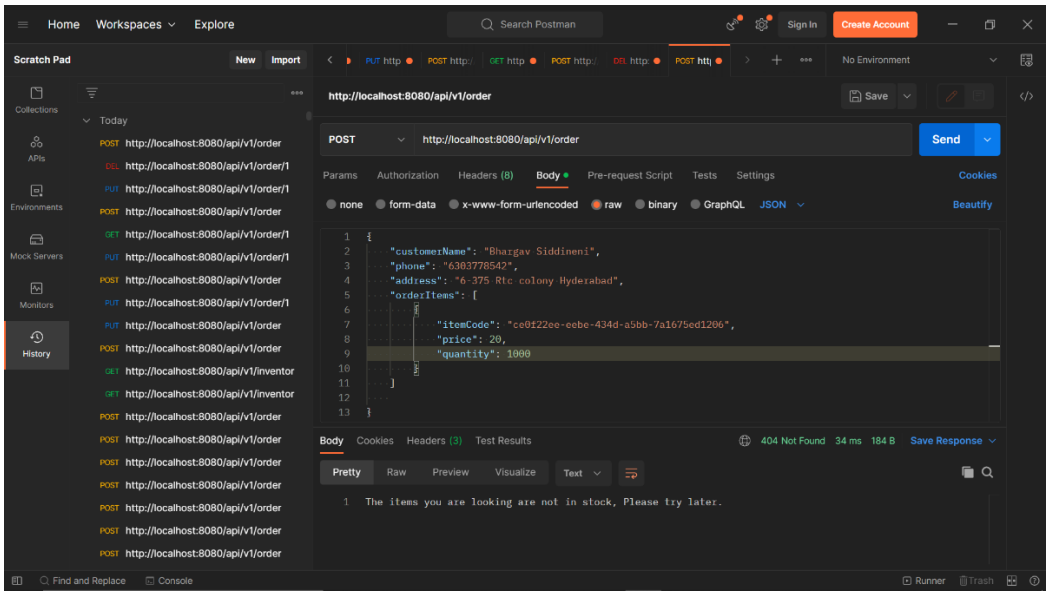
A notification is sent to Notification service from order service regarding the cancellation of the order along with the order details using rabbit mq. We can see the message in the console



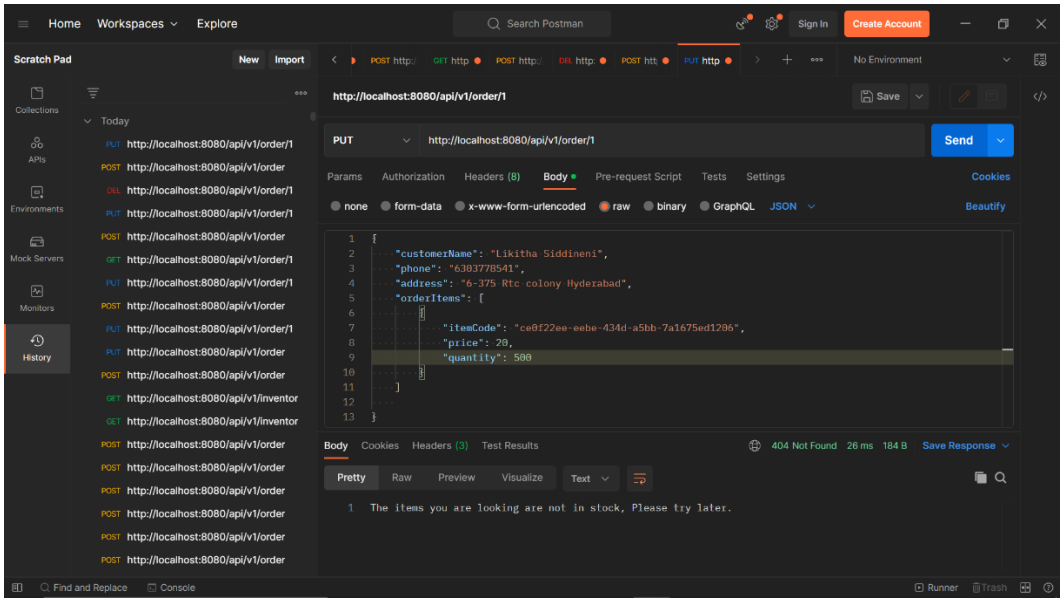
Graphical analysis of the time rate of the message that is sent to the notification service in rabbitMq UI.



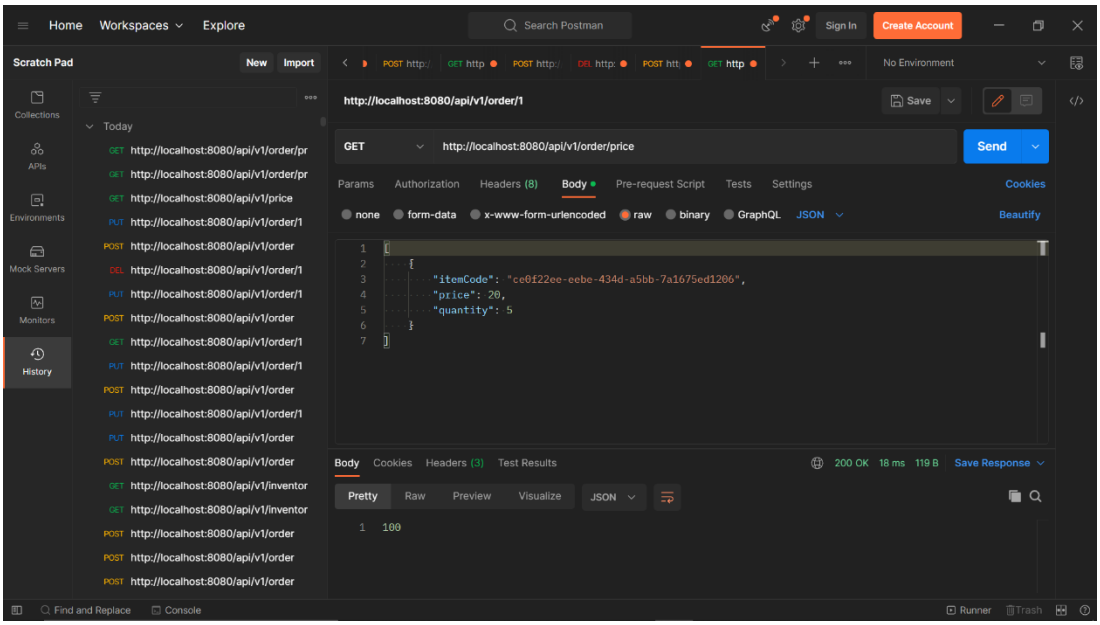
If we are posting order having item quantity more than the actual quantity present it shows error as below.



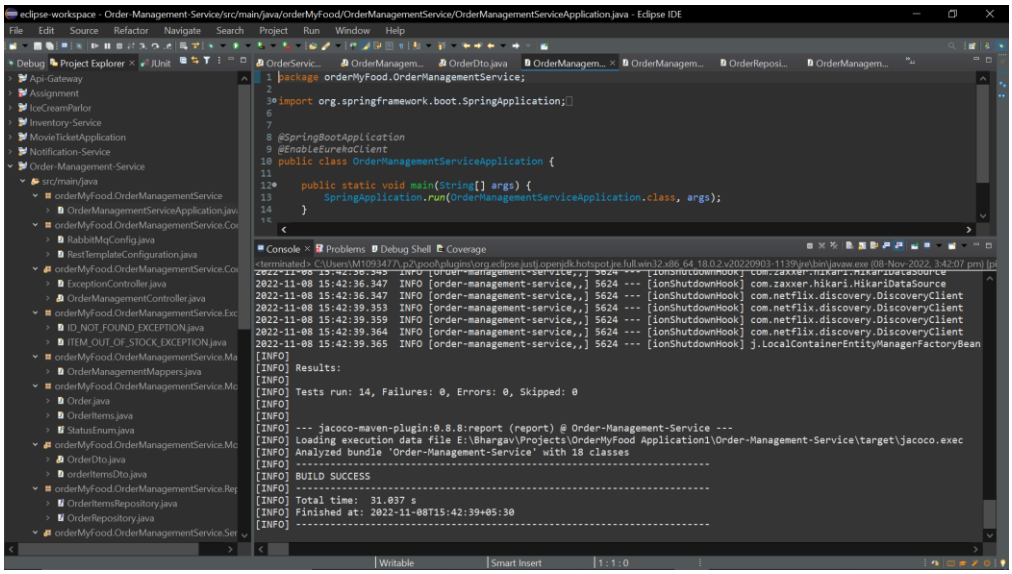
Same we observe in PUT operation



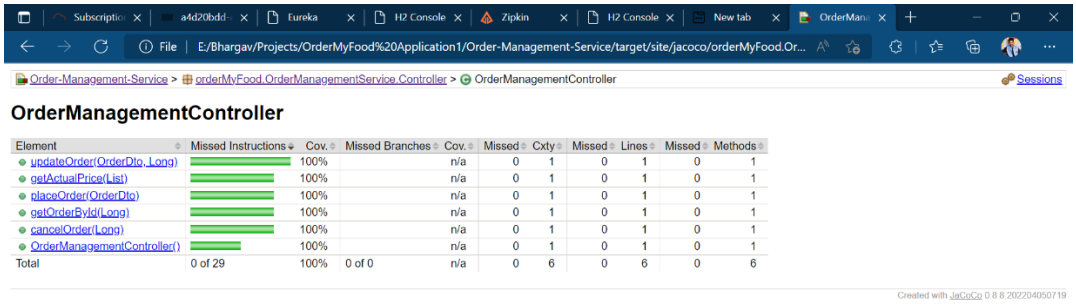
We can check the price of the item by calling to this end point before actual order is placed.



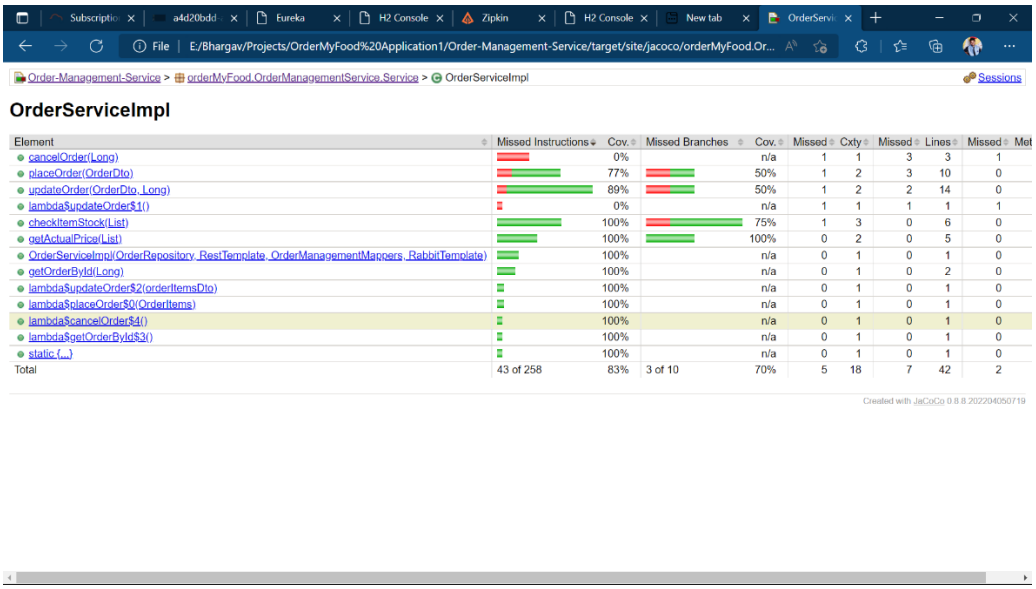
Using maven -test for order management service such that jacoco report is created. We can see unit tests passed



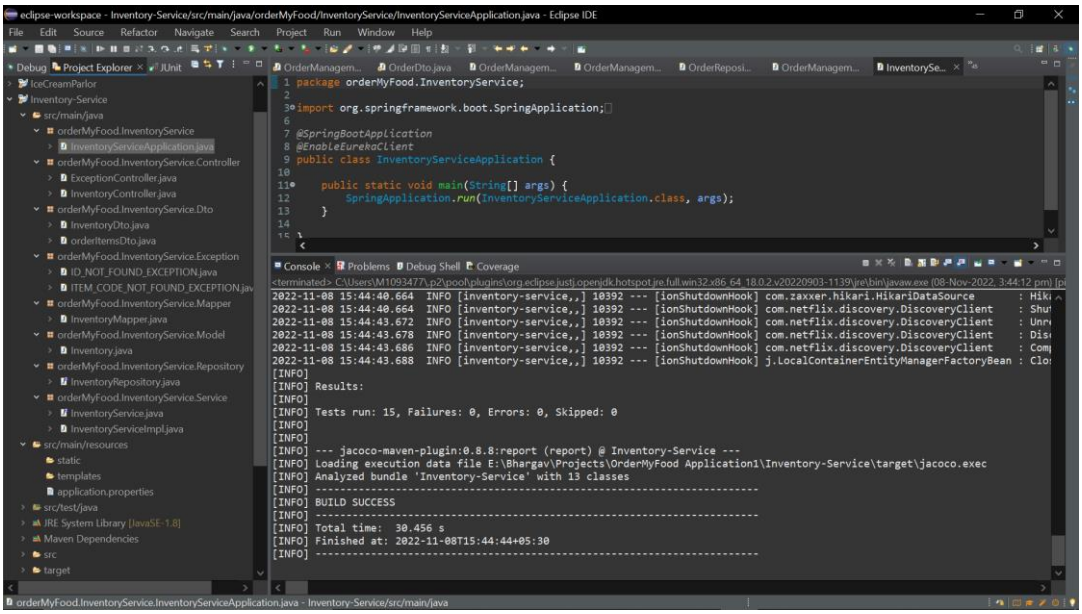
We can observe the order Controller report



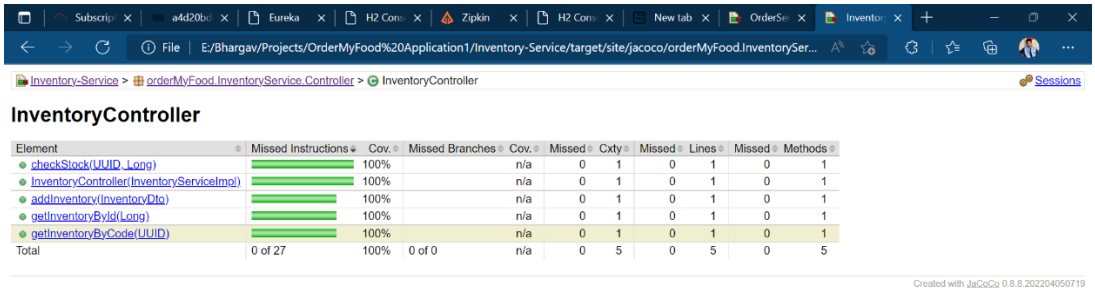
Order Management service layer jacoco report



Inventory Service maven-test report, all unit tests are passed



Inventory Controller Jacoco code coverage report



Inventory-Service > orderMyFood.InventoryService.Controller > InventoryController

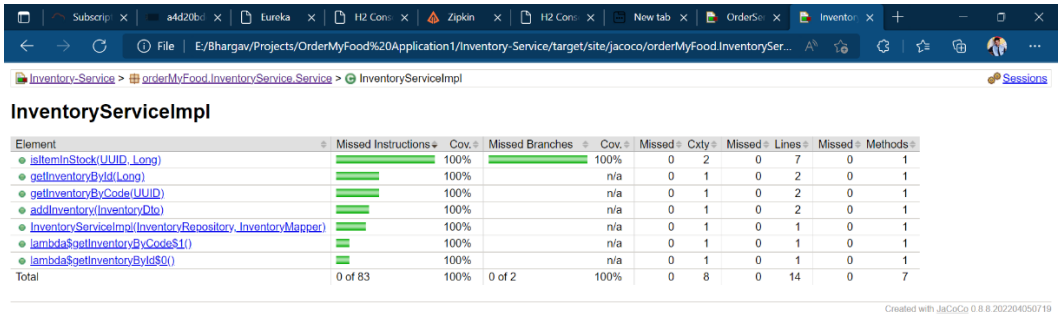
Sessions

InventoryController

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkStock(UUID, Long)		100%		n/a	0	1	0	1	0	1
InventoryController(InventoryServiceImpl)		100%		n/a	0	1	0	1	0	1
addInventory(InventoryDto)		100%		n/a	0	1	0	1	0	1
getInventoryById(Long)		100%		n/a	0	1	0	1	0	1
getInventoryByCode(UUID)		100%		n/a	0	1	0	1	0	1
Total	0 of 27	100%	0 of 0	n/a	0	5	0	5	0	5

Created with JaCoCo 0.8.8.202204050719

Inventory Service layer jacoco code coverage report



Inventory-Service > orderMyFood.InventoryService.Service > InventoryServiceImpl

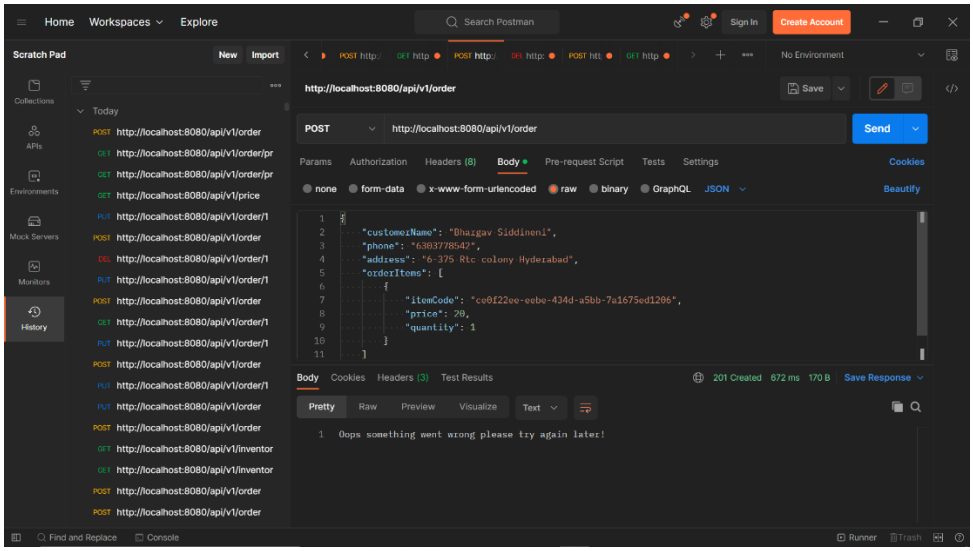
Sessions

InventoryServiceImpl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
isItemInStock(UUID, Long)		100%		100%	0	2	0	7	0	1
getInventoryById(Long)		100%		n/a	0	1	0	2	0	1
getInventoryByCode(UUID)		100%		n/a	0	1	0	2	0	1
addInventory(InventoryDto)		100%		n/a	0	1	0	2	0	1
InventoryServiceImpl(InventoryRepository, InventoryMapper)		100%		n/a	0	1	0	1	0	1
lambda$getInventoryByCode$1()		100%		n/a	0	1	0	1	0	1
lambda$getInventoryById$0()		100%		n/a	0	1	0	1	0	1
Total	0 of 83	100%	0 of 2	100%	0	8	0	14	0	7

Created with JaCoCo 0.8.8.202204050719

Here while placing the order if the inventory service is down then that exception is handled successfully using circuit breaker.



Query Management for Restaurant -search service is done using native query technique