Mercury Cyclists

Nabeel Badran

Table of Contents

1. Introduction

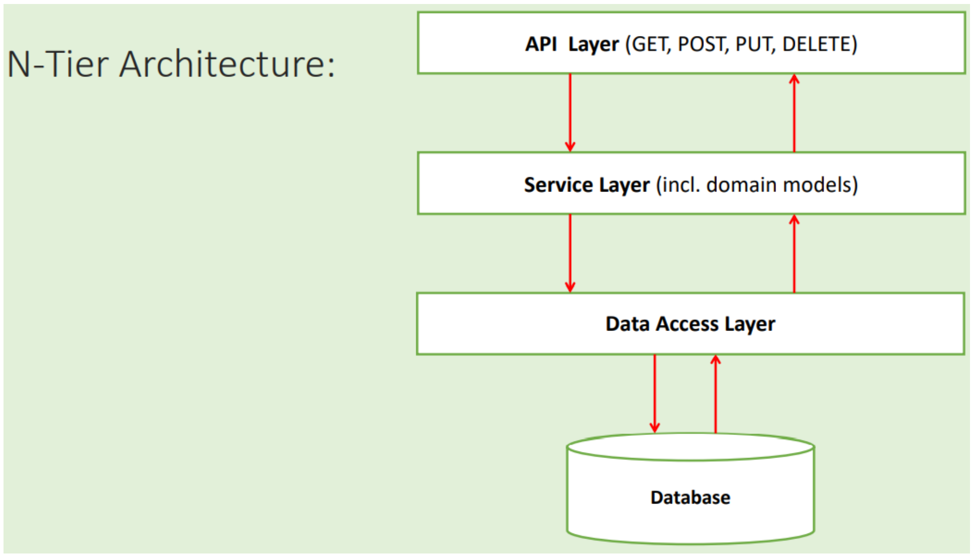
Spring Boot is a tool that makes it easier and faster to develop web applications and microservices which utilise the Java Spring Framework (which is used for making Java applications of enterprise-level quality). Since April 2014, it also consists of auto-configuration or an opinionated approach to configuration. It uses a micro-framework that allows for the making of microservices for websites and mobile apps, such as Amazon, Netflix, Uber, etc.

In our application, we have used this framework because of its simplicity in coding and executing which makes life easier for our development team and the user that will use this application. Our application is in its beta form and will consist of major updates in the future with the estimated completion on the 30th of October 2022.

This document includes a description of the spring project structure, including how the packages align with the multi-tier architecture, the functionality of each java class, examples of input and output and how to build and run the program with all the steps needed. The program has been structured in a way with private variables which is the base of the model, creating optimal security and an ethical program abiding by the Australian guidelines for programming. This has also been linked to an H2 in-memory database which stores the information by getters and setters for later use.

The program uses and implements Kafka through a stream processing-powered microservice system. The streaming platform uses Apache Kafka to handle requests. The program consists of 4 microservices. Each microservice represents a key problem of the application. The producer in our case is the Sale Microservice that publishes the sale event topic and the processor that consumes the topic is the BusinessIntelligence Microservice.
Also, each application of the microservice is separate and does not have a shared database. The program utilises REST Requests via HTTP to connect applications and to facilitate communication between microservices.

2. Description Spring Project’ Structure, Including How The Packages Align With The Multi-tier Architecture, And The Functionality Of Each Java Class


N-Tier Architecture:
API Layer (GET, POST, PUT, DELETE)
Service Layer (incl. domain models)
Data Access Layer
Database

Multi-tier architecture allows for resources to be assigned to specific layers of the system and not be shared, which also makes software easier to manage. Our packages apply this style of architecture by using Spring structures which allow us to represent layers within the system as Java files and classes. Controller classes represent the API layer by receiving GET, POST, PUT and DELETE requests from the user and calling the appropriate functions from the Service class, which calls the appropriate function from the Repository package, before the result travels back up this chain to the user.

2.1. Packages And Their Classes

Our group worked on Part A objections of the project using simple and facile implementation, where we only created two packages, named supplier and contact and included in each package all the classes and interfaces related to every package name separately, so that it is very clear for all members of the group or any future viewers/programmers, with putting in consideration the N-tier architecture representation and concept. The packages are expected to increase in future iterations depending on future Part B and Part C specifications. Description of each package and detailed elaboration of every class and interface found in each package that follows the Multi-tier architecture concept for the program can be found below:

2.2 Procurement Service

2.2.1. Supplier

Supplier class is the Entity class of the supplier package and it contains all the information needed and related to supplier, such as supplierId and supplierName, with all the setters and getters to access data and get needed information, class constructors such as no arguments constructor and all arguments constructor, and a toString() function that allows string representation of supplier details when it is needed to a controller or service.

SupplierController class serves as the API Layer in the Supplier package, and it has all the resources for the API having the @RestController annotation and the @RequestMapping annotation that defines the path of supplier when on localhost. The SupplierController class constructor has @Autowired annotation that acts as a dependency injection so that the studentService object can be instantiated and then injected into the SupplierController constructor, and that enables the mapping annotations to efficiently work. @GetMapping annotation used in this class that gets what we need from the server. It maps the HTTP GET requests onto specific handler-implemented methods. It also has (path = "{supplierId}") that returns supplier with given supplierId. @PostMapping annotation is used in this class to handle HTTP POST type of request methods matched with a given URL expression. @PutMapping annotation is used in SupplierController class for mapping HTTP PUT requests onto specific handler methods. We have also included the @DeleteMapping annotation that maps the HTTP DELETE requests onto specific handler methods of a Spring controller to delete resources and data using the Long supplierId which is the identifier that lets the system know which supplier to remove, all done using deleteSupplier method.

SupplierService class serves as the class that contains all of the business logic needed for managing suppliers. As mentioned above, the API Layer should talk to the service layer to get some data, and the Service Layer should also talk to the Data Access Layer to get data so that there is a connection between all of them. This class has @Service annotation that is used with classes that need to provide some business functionalities with more semantics and more readability. @Service annotation also means that the SupplierService class is a spring bean so that it can be instantiated in other classes like StudentController.

SupplierRepository interface that extends JpaRepository<"Class" that is "Supplier", "ID" that is Long> acts as the Data Access Layer in the multi-tier architecture. This interface, also called Data Access Object, is a design pattern that provides us with simplified access to the stored data and isolates the business layer from the persistence layer. This interface has @Repository annotation because the repository is responsible for data access. The SupplierRepository contains three methods at the moment, each performing different tasks. FindSupplierBySupplierName is one method that takes the supplier name in its parameter and if found from the list of suppliers stored in the H2 database, it returns the supplier. The method has @Query that has SQL query that selects the student that has the same name as the passed name. FindBySupplierId and FindSupplierBySupplierBase all do the same job as the first method, but we thought they should be important to have in cases that require finding suppliers by id or by the base.

2.2.2. Contact

Contact class is the Entity class of the contact package and it contains all the information needed and related to supplierContact, such as name, phone, and id, with all the setters and getters that act as accessors and mutators, constructors such as no arguments constructor and all arguments constructor for the initializing objects of the contact class, and a toString() function that allows string representation of supplierContact details when it is needed to a controller or service.

ContactController class acts as the API Layer in the contact package. This class is responsible for processing incoming RESTful API requests through HTTP, acting as the coordinator between the View and the Model. It is annotated by @RestController and it basically, and in simple terms, controls the flow of data into the model object, then it retrieves necessary data and returns needed responses. @GetMapping annotation is used twice in this ContactController class iteration. It maps the HTTP GET requests onto specific handler-implemented methods. It also has (path = "{Id}") that returns contact with the given id. getContact(Long id) and getContacts() methods use the @GetMapping annotation. To get a contact from the list of contacts, findAllById(id) method is implemented to get the customer with the id provided in the URL. findAll() method is also implemented, and it returns the list of contacts in the database. @PostMapping annotation is used, as it is needed to allow retrieval of data after the application layer has been loaded. createContact(contact) uses the @PostMapping annotation, and it takes a parameter @RequestBody for an object of contact that then is saved into the contactRepository. @PutMapping annotation is used to retrieve and update contacts. Method updateContact takes as a parameter @RequestBody an object of class Contact, as well as the @PathVariable that is the id. Lastly, we have included a deleteContact method that takes id as a @PathVariable in its parameter. It has the @DeleteMapping annotation that maps the HTTP DELETE requests for it to delete the required contact from contactRepository that matches the same ID given.

ContactService class is considered the service class of contact package. It includes the @Service annotation and it is the class that allows us to add and develop business functionalities. getContact(Long) and getContacts() both return a List<Contact> by using findAll() and findAllById(id) methods that retrieve the needed entities from the database. createContact(Contact) method saves contacts in the database, updateContact(Contact, Long) updates contacts in the database, and deleteContact(Long) deletes contacts from the database by deleteById(id) method to delete the contact from the saved contacts in the contact repository.

ContactRepository interface that extends JpaRepsitory<"Class", "ID">, <"Contact", Long> in this case, acts as the Data Access Layer in the multi-tier architecture. It has @Repository annotation that is used to indicate that the class is capable of providing a mechanism for the storage of data, retrieval of data, update, delete, and search operations on objects. In the contact package, and in this iteration, we didn't need to have any code in the interface itself, but we instantiated an object of the ContactRepository in the ContactController class that has @Autowired annotation, that is a Spring API feature for encapsulation initialization of a private object of the service java file and used the instantiated object efficiently in the ContactController class to find and save contacts.

2.3. Inventory Service

2.3.1. Part

Part class is the Entity class of the part package and it contains all the information needed and related to partName, description, supplerId, quantity, and product with all the setters and getters that act as accessors and mutators, constructors such as no argument constructor and all arguments constructor for the initialising objects of the part class, and a toString() function that allows string representation of part details when it is needed to a controller or service.

PartController class serves as the API Layer in the Part package, and is responsible for processing incoming RESTful API requests through HTTP, acting as the coordinator between the View and the Model. It is annotated by @RestController, @ResponseBody, and @RequestMapping key path being "/part". @GetMapping is used to get a part of the list of parts, look up parts by parttId, and validate a part. @PostMapping annotation is implemented in the createPart() method, returning the part. @PutMapping annotation is also used to update part with a new part and add supplier to part.

PartConfig class utilises function restTemplate() from package RestTemplate.

PartRepository interface extends JpaRepository<Part, Long> where "Part" is the class, and "Long" is the ID that acts as the Data Access Layer in the multi-tier architecture. This interface has @Repository annotation that is used to indicate that the class is capable of providing mechanism for the storage of data, retrieval of data, update, delete, and search operations on objects. In the part package with this iteration, there was no need for additional code in the interface itself but instead, instantiated an object of the PartRepository in the PartController class.

2.3.2. Product

Product class is the Entity class of the product package and it contains all the information needed and related to productName, price, comment, stockQuantity, and parts with all the setters and getters that act as accessors and mutators, constructors such as no argument constructor and all arguments constructor for the initialising objects of the product class, and a toString() function that allows string representation of product details when it is needed to a controller or service.

ProductController class serves as the API Layer in the Product package, and is responsible for processing incoming RESTful API requests through HTTP, acting as the coordinator between the View and the Model. It is annotated by @RestController, and @RequestMapping key path being “/product”. @GetMapping is used to get a product of the list of product, look up products by productId, and validate a product. @PostMapping annotation is implemented in the createProduct() method, returning the product. @PutMapping annotation is also used to update product with a new product and add part to product.

ProductRepository interface extends JpaRepository<Product,Long> where “Product” is the class, and “Long” is the ID that acts as the Data Access Layer in the multi-tier architecture. @EnableJpaRepositories ensures the enabled usage of JpaRepositories packages. This interface has @Repository annotation that is used to indicate that the class is capable of providing mechanism for the storage of data, retrieval of data, update, delete, and search operations on objects. In the product package with this iteration, there was no need for additional code in the interface itself but instead, instantiated an object of the ProductRepository in the ProductController class.

2.4 Sales Service

2.4.1. Store

Store class contains data about the ID, address, manager and name of the store, as well as methods to retrieve all this information.

StoreController class allows us to create, update and delete stores, as well as obtain a list of all stores, add a sale to a store’s records and look up how many sales a certain store is associated with.

StoreRepository interface stores the information of the Store class which includes the store ID, store address and manager and of course the name of the store, its also capable of viewing the stores that are newly created, updated or deleted.

2.4.2. Sale

Sale class is one of the main entities of the SalesService and it contains all the information needed and related to saleId, quantity, dateAndTime, and productId with all the setters and getters that act as accessors and mutators, constructors such as no argument constructor and all arguments constructor for the initialising objects of the sale class, and a toString() function that allows string representation of a sale details when it is needed to a controller or service.

SaleController class acts as the API Layer in the sale package and is responsible for processing incoming RESTful API requests through HTTP, acting as the coordinator between the View and the Model. It is annotated by @RestController and has @RequestMapping key path being "/sale". @GetMapping is used to get a sale or the list of sales, look up products by sale id, and validate a sale. @PostMapping annotation is implemented in the createSale() method and for createBackOrderSale for Online and InStore Sales. @PutMapping annotation is also used to update a sale and to add a product to sale. Lastly, @DeleteMapping annotation is implemented to delete a sale.

SaleRepository interface extends JpaRepository<Sale, Long> from the spring framework where "Sale" is the entity, and "Long" is the ID that acts as the Data Access Layer in the multi-tier architecture. It is annotated by @Repository. Mostly in the program, the repositories only extend the JpaRepository and do not contain any methods in them as the default functions, such as save() and findById(), are used.

2.4.3. InStoreSale

InStoreSale class is one of the child classes of the parent entity Sale. It extends the datafields and functions from the parent entity Sale and it has riecieptNo and an object of Store. The relationship between it and Store is @ManyToOne where many InStoreSales can be related to one Store.

InStoreSaleRepository interface is just like the parent entity Sale where it has a repository that acts as the database layer to store InStoreSales and it also extends the JpaRepository from the spring framework.

2.4.4. OnlineSale

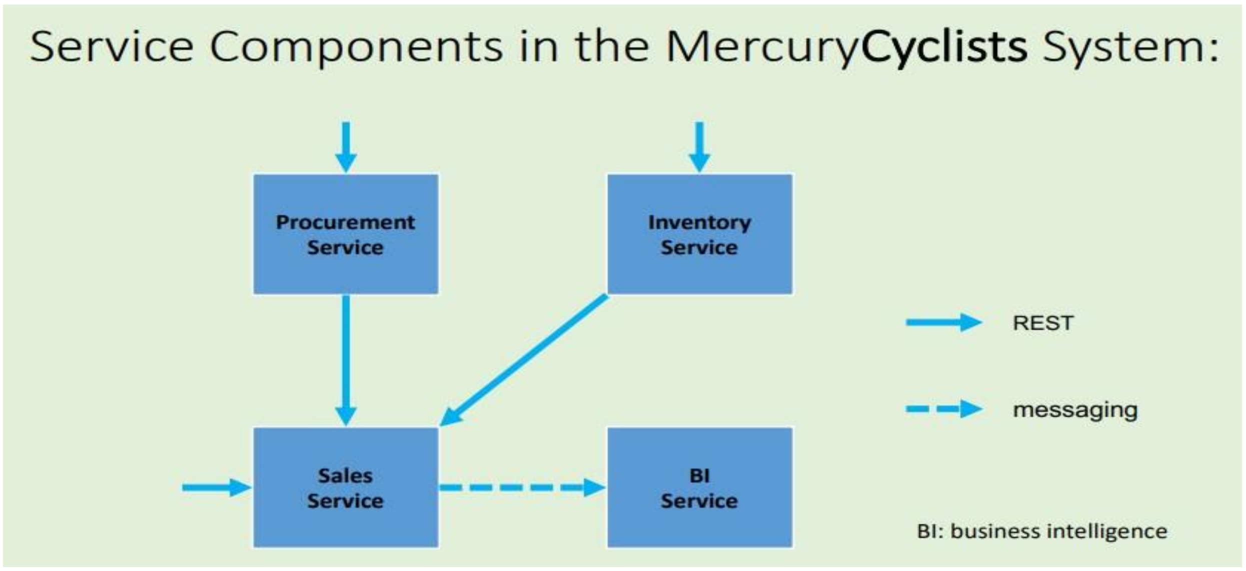
OnlineSale class is the other child class of the parent entity Sale. It contains all the data and methods for an online sale, including the address of the customer, their name, the sale's ID number, the quantity of the sale, the date and time of the sale, and the ID of the product ordered.

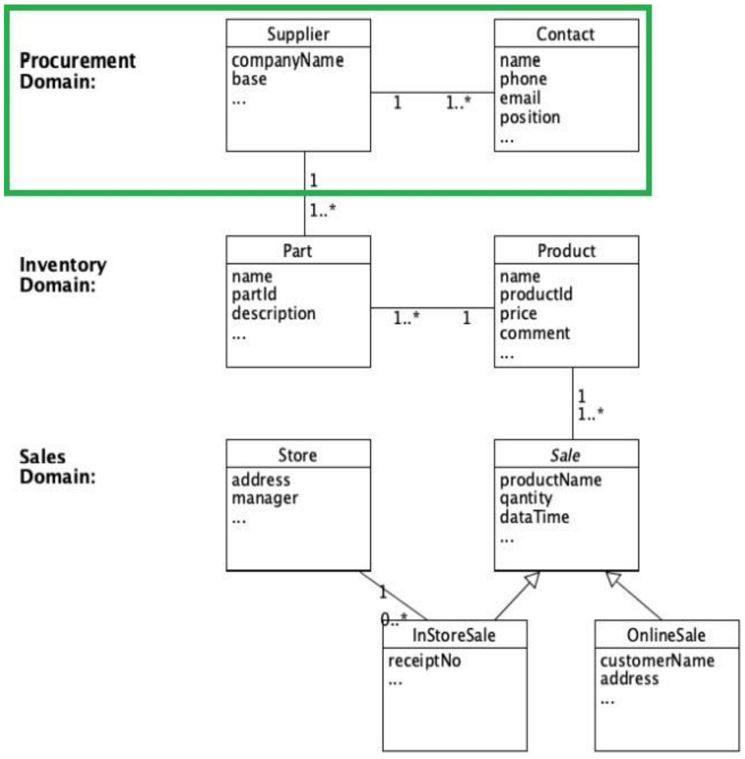
OnlineSaleRepository interface is also a database for the OnlineSale class where the above data mentioned such as the address of the customer, their name, etc, of an online sale is stored to be used later on. Just like the rest of the repositories, it extends the JpaRepository.

3. Domain Driven Design principles implemented

Domain Driven Design (DDD) is the principle of software development that is centred on the domain of people using it. In simple terms, it is the concept of using code to solve business issues.

Part A


Service Components in the MercuryCyclists System:
Procurement Service
Inventory Service
Sales Service
BI Service
REST
messaging
BI: business intelligence

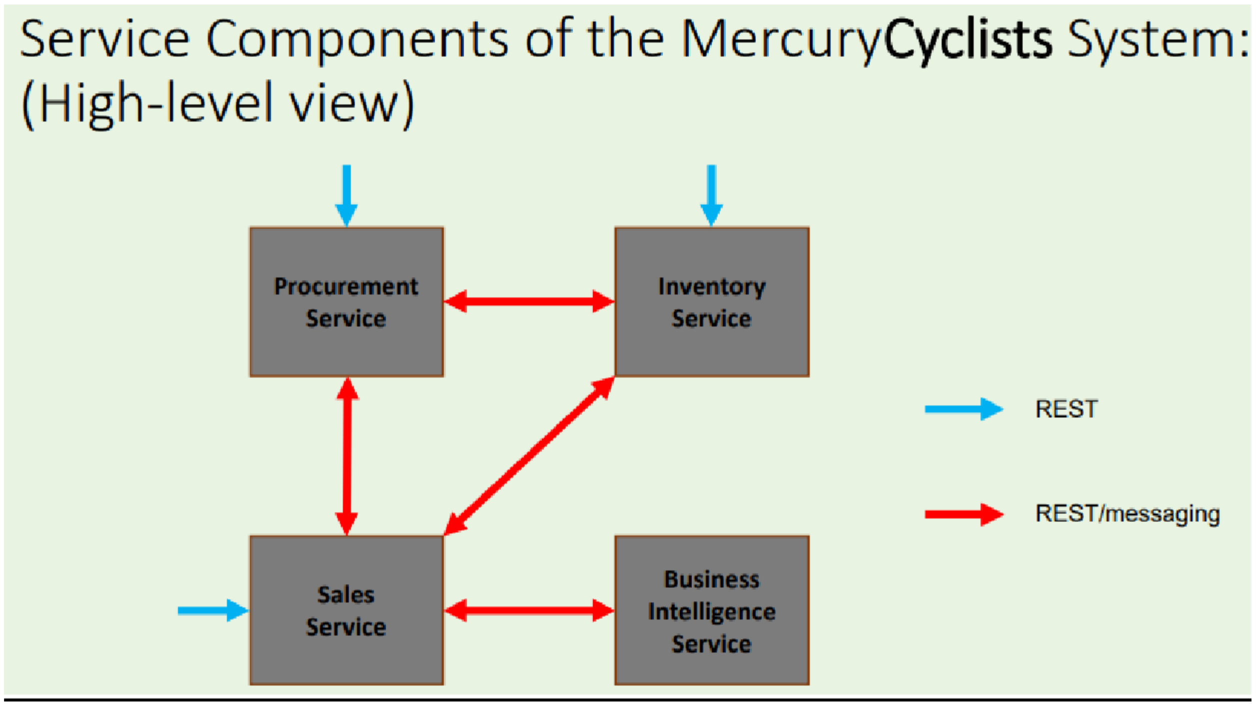


In the above iteration (Part A of the project), the focus is on the Procurement Domain and Procurement Service of the MercuryCyclists system that includes Supplier and Contact (Supplier Contact).

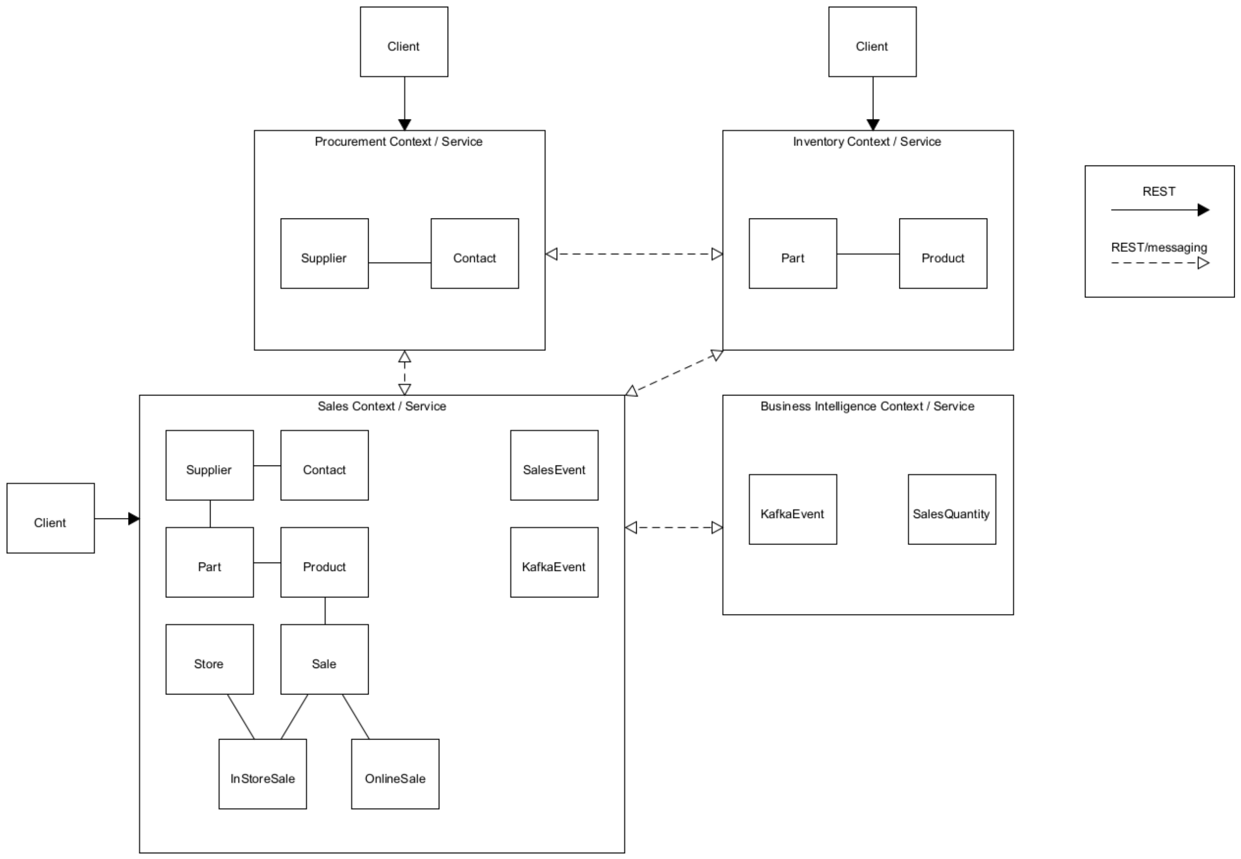
Our implementation, according to Part A specifications, included the Procurement Domain which consists of the Supplier sub-domain. The supplier sub-domain includes two domain entities that are Supplier and Contact that have a relationship and work with each other and can depend on each other in some cases. The relationship between Supplier and Contact entities is implemented as follows:

The relationship from Supplier to Contact is defined as **@OneToMany** where one supplier can have contact details. In comparison, the relationship from Contact to Supplier is defined as **@ManyToOne** where many contact details can be related to one supplier.

Parts B & C


Service Components of the MercuryCyclists System: (High-level view)
Procurement Service
Inventory Service
Sales Service
Business Intelligence Service
REST
REST/messaging

<u>DDD Diagram</u>



The domain design is an online sales application. There are four subdomains, these include;

- **Procurement Domain**
- **Inventory Domain**
- **Sales Domain**
- **Business Intelligence (BI) Domain**

Each subdomain consists of domain objects and/or entities that represent and solve the subdomains which are;

- **Procurement Domain**
  - Supplier
  - Contact
- **Inventory Domain**
  - Part
  - Product
- **Sales Domain**
  - Store
  - Sale

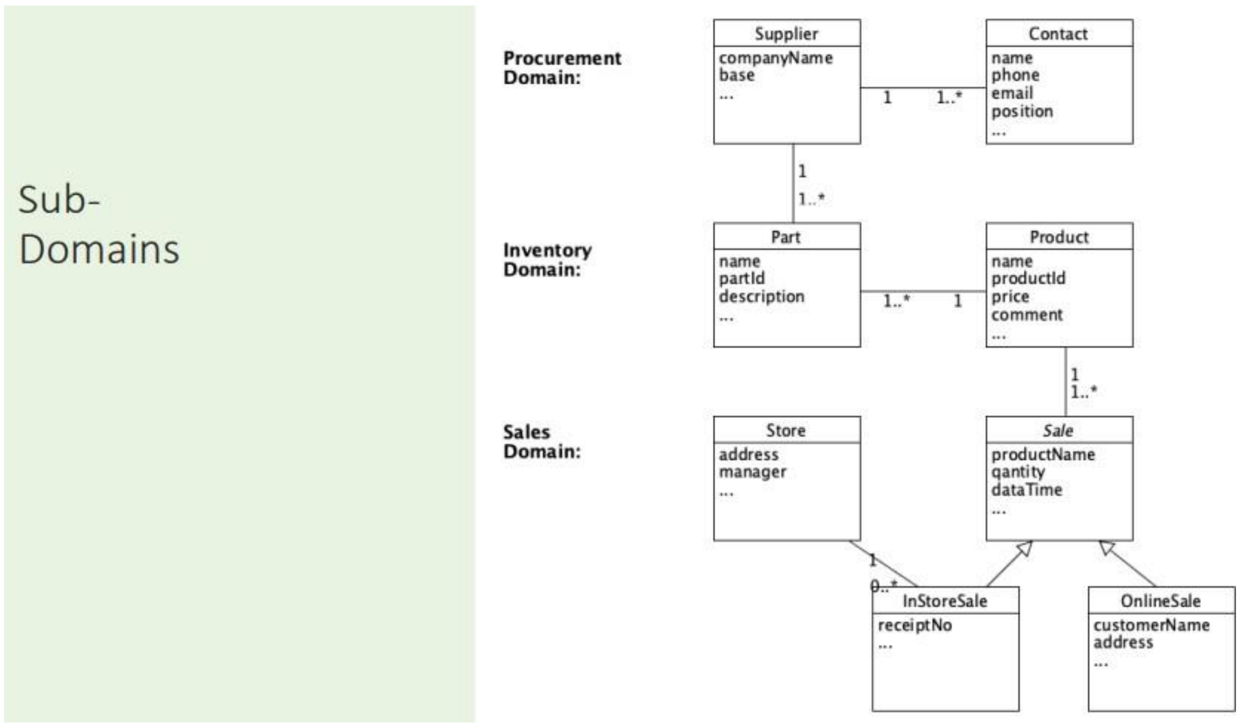
    - InStoreSale
    - OnlineSale
- **Business Intelligence (BI) Domain**
    - SaleEvent

Entities are implemented through the classes: **Supplier, Contact, BackOrderSale, jsonWrapper, Part, Product, BackOrder, Store, Sale, InStoreSale, OnlineSale,** and **SaleEvent.** These classes have global identities and keep these identities throughout the whole system. Their identities are defined by generating an identification number (ID) for each individual object instantiated from these classes.

Value object is implemented through the classes of **SalesEvent, SalesQuantity,** and **KafkaEvent** which holds final temporary variables based on the transactions. When a sale has been made through SalesService, it creates an event that takes the data from the user and stores it temporarily within an instantiated SalesEvent object. The event is handled by SalesService which notifies ProcurementService to obtain the message of updating the stock quantity of the ordered item with the provided quantity value. This is considered a value object which represents no relation to identity as they are defined by their attributes. A new SalesEvent object is created when new values are added from the variables of the class. The object represents an event for SalesService to handle and communicate with ProcurementService in receiving messages to update the stock quantity of the part.

SalesQuantity is similar to **SalesEvent** and **KafkaEvent** value events that contain final temporary values using encapsulation. When a user requests an event through the ‘service’ and ‘control’ layer in the **BIService**, a ‘SalesQuantity’ object is temporarily holding these values.

Services are implemented through the modules **ProcurementService, InventoryService, SalesService,** SaleInteractiveQuery, and SaleStreamProcessing for the **BusinessIntelligenceService (BIService).** These services are used to represent a particular domain and perform domain operations. It contains a variety of operations that can include; creating a sale, updating a part, and many more. These services are responsible for receiving inputs from externals of domains and returning the result of the action.



Aggregates are implemented through the use of relationships between entities. This process refers to the combination of entities to form a single meaningful entity. This can be demonstrated through the multiple functions of the system that include:

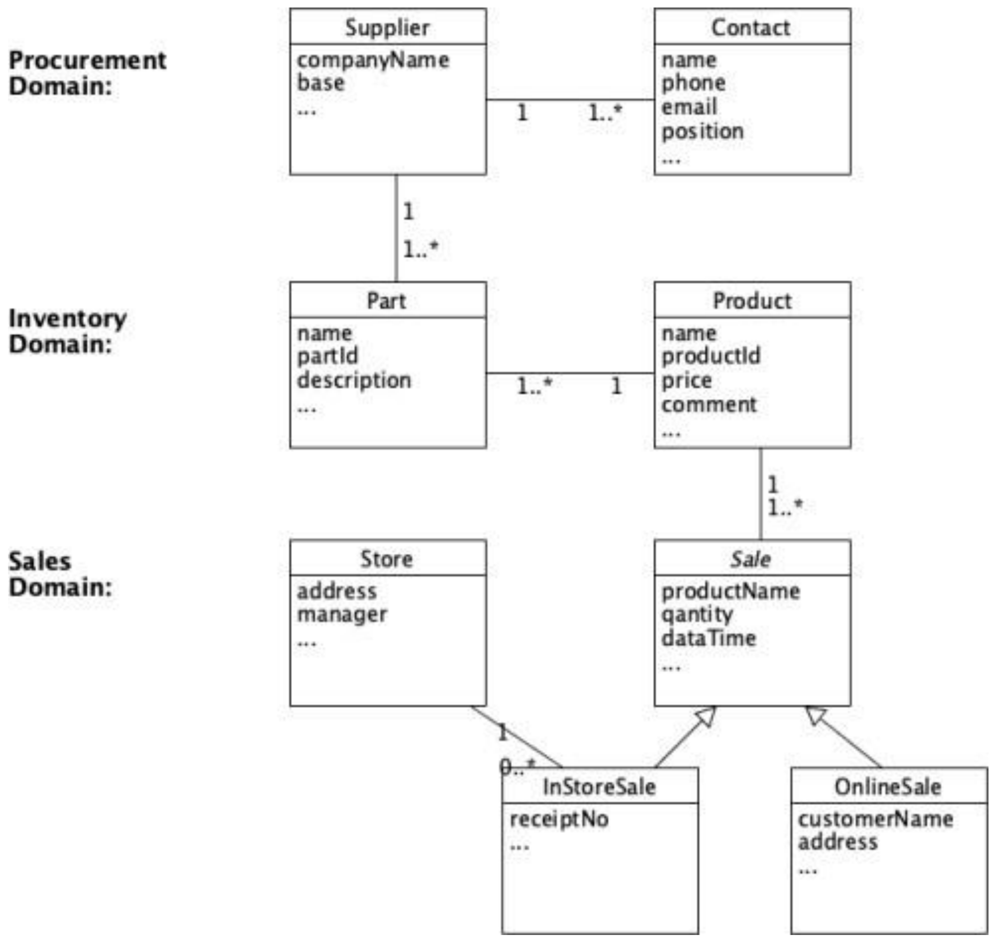
- When a sale is created, you can search the basic information of a part or a user by the sale and etc.
- When a supplier is searched, the supplier data may consist of their existing sales and contacts.
- When a part is searched, it also consists of the part detail information.

The relationships that have been formed to create the aggregation between the entities are:

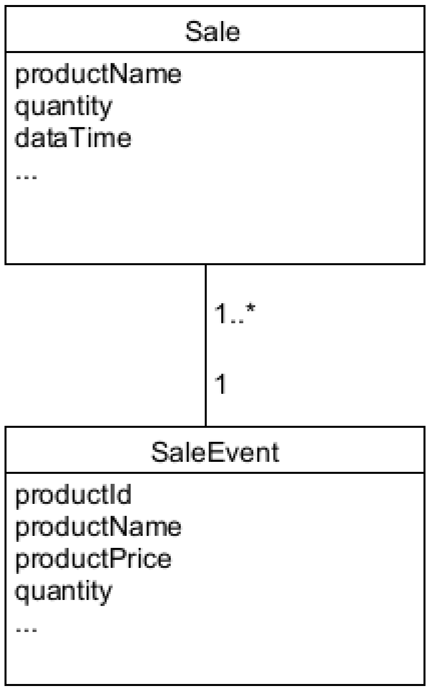
- One-to-many (@OneToMany)
  - Supplier → Contact
  - Supplier → Part
  - Product → Sale
- Many-to-one (@ManyToOne)
  - Part → Product
  - Sale → SaleEvent
- One-or-many (@OneToMany)
  - Store → InStoreSale

Repositories are containers for aggregates and entities. All aggregates that are stored into a repository have been able to be retrieved at a later use. Repositories are implemented through the interfaces of; **PartRepository, ProductRepository, BackOrderSaleRepository, ContactRepository, SupplierRepository, InStoreSaleRepository, OnlineSaleRepository, SaleRepository,** and **StoreRepository**. Each repository uses the extend keyword to inherit the properties of the **JpaRepository**.

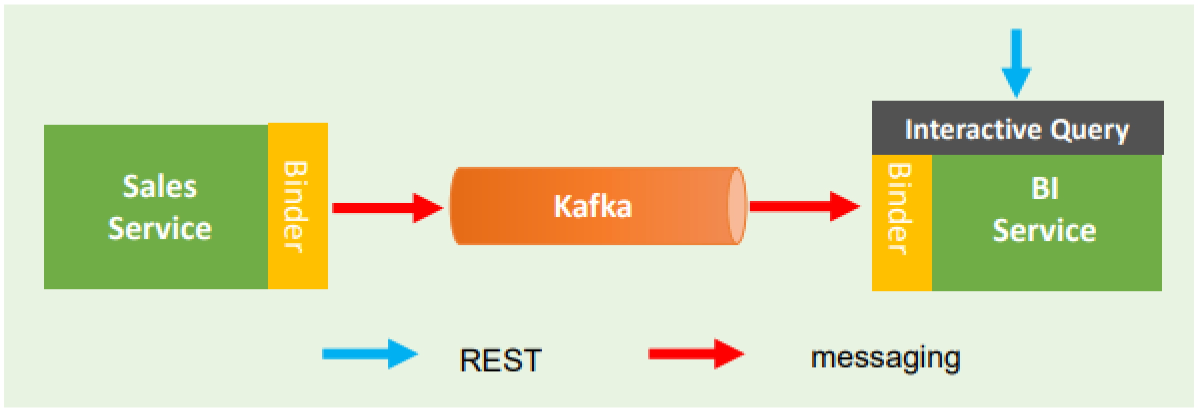
The domain model is used for the complexities of a problem domain. The concepts in the domain are encapsulated as objects that contain both data and behaviour.



Business Intelligence Domain:



3.1. Kafka Architecture

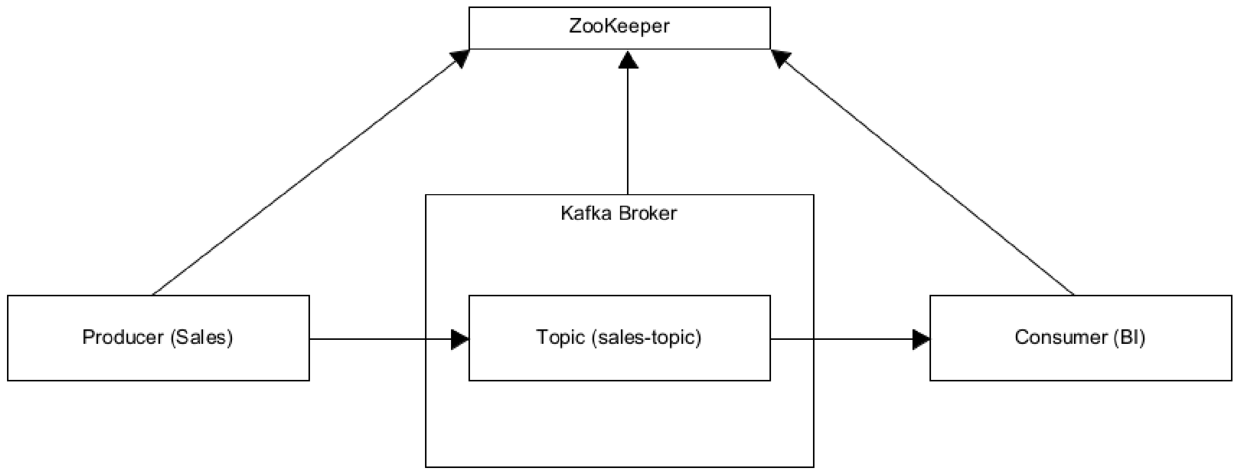


Kafka is a hybrid architecture messenger that implements a database through a process solution. The main goal of the implementation of the system is that it streams real-time data, which can build databases and provide units of distribution storage points for occurrences that can be used by the systems. This allows the systems to resolve issues such as unifying complications when the application becomes increasingly known.

Kafka includes producers, consumers, and a cluster. A cluster requires a number of brokers, serving as servers to store topics that are received by the producers which can be demonstrated in the diagram above in the process from the SalesService. The brokers are responsible for sending data to storage and assigning identification counteracts when a consumer endorses a topic to the BIService. The brokers are responsible for receiving consumer fetch requests as demonstrated in the BIService classes.

When consumers use the file methods, it will return a response to the consumer through the fetch requests. When the consumers send these requests, the broker’s topic is broken down into particular topic partitions where the application is a product object and quantity. These partitions act as separate commit logs.

Kafka Architecture Diagram



The producers in this architecture are often in other applications. In this application, it is a separate part of the microservice, “SalesService” sends data through to the cluster where the cluster performs its process. Consumers in this architecture are recipient applications that endorse a certain topic within the cluster and brokers. In this case, the consumer is the BIService which was mentioned before endorsing topics through requests from the BIService classes.

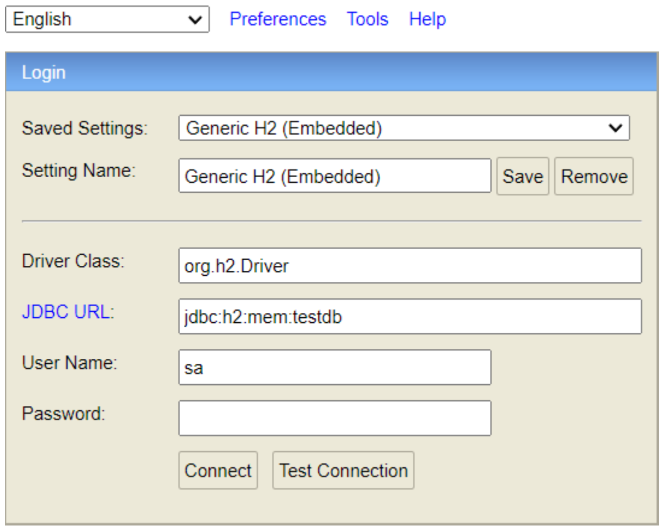
Topics are the data storage in the Kafka brokers which can be broken down into partitions that hold particular information pushed from the producers. In earlier mentions, this data is received from the SalesService boundary of the application process.

The sales producer, sales topic in the Kafka broker, and the BI consumer are utilised in the zookeeper.

3.2. Use Cases Of The Application With Examples Of Input And Output (i.e., REST requests)

H2-CONSOLE INPUTS:

InventoryService: http://localhost:8080/h2-console/
ProcurementService: http://localhost:8081/h2-console/
SalesService: http://localhost:8082/h2-console/
BusinessService: http://localhost:8083/h2-console/

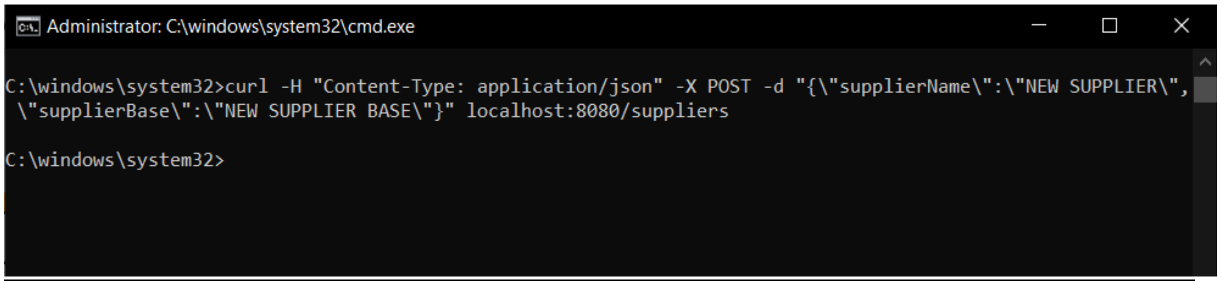


Saved Settings: Generic H2 (Embedded)
Setting Name: Generic H2 (Embedded)
Driver Class: org.h2.Driver
JDBC URL: jdbc:h2:mem:testdb
User Name: sa
Password:

CREATE/UPDATE SUPPLIER:

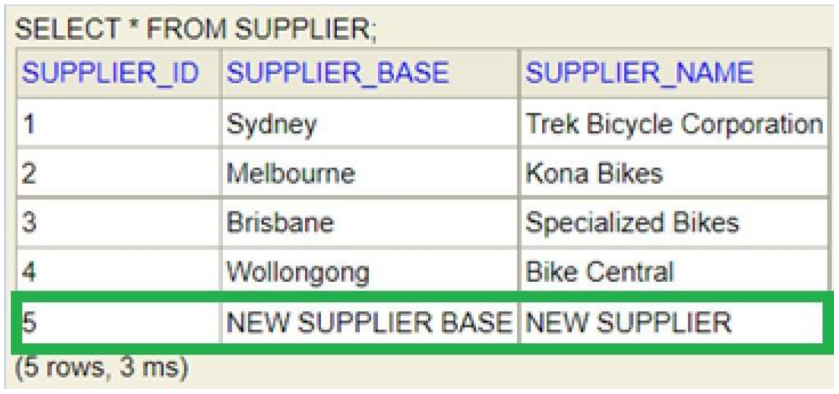
1	Create/update supplier	Procurement

CREATE SUPPLIER - POST REQUEST



```
C:\windows\system32>curl -H "Content-Type: application/json" -X POST -d "{\"supplierName\":\"NEW SUPPLIER\",
 \"supplierBase\":\"NEW SUPPLIER BASE\"}" localhost:8080/suppliers

C:\windows\system32>
```

curl -H "Content-Type: application/json" -X POST -d "{\"supplierName\":\"NEW SUPPLIER\", \"supplierBase\":\"NEW SUPPLIER BASE\"}" localhost:8080/suppliers



SELECT * FROM SUPPLIER;

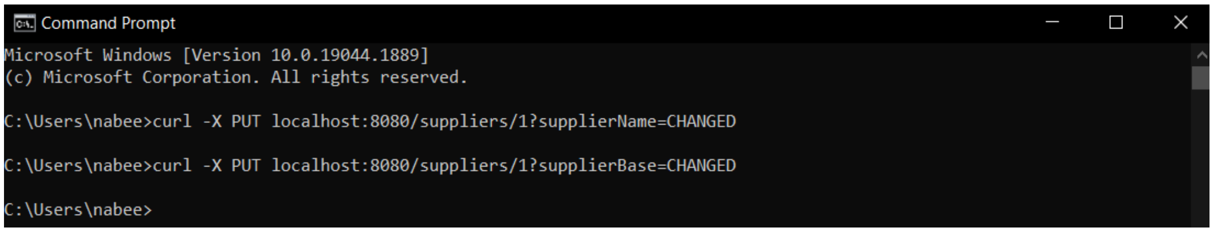
SUPPLIER_ID	SUPPLIER_BASE	SUPPLIER_NAME
1	Sydney	Trek Bicycle Corporation
2	Melbourne	Kona Bikes
3	Brisbane	Specialized Bikes
4	Wollongong	Bike Central
5	NEW SUPPLIER BASE	NEW SUPPLIER

(5 rows, 3 ms)

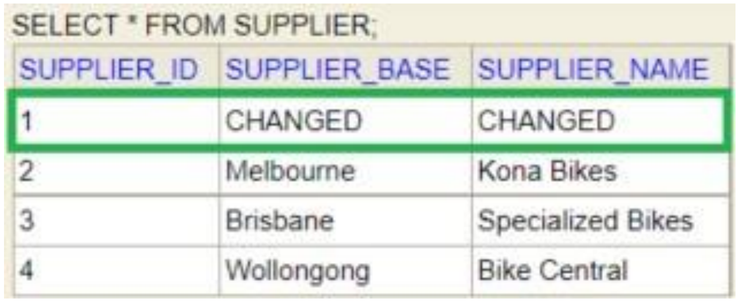
UPDATE SUPPLIER - PUT REQUEST

There are multiple ways that can be used for updating supplier. One is updating supplierName, and other is updating supplierBase.

```
curl -X PUT localhost:8080/suppliers/1?supplierName=CHANGED
curl -X PUT localhost:8080/suppliers/1?supplierBase=CHANGED
```



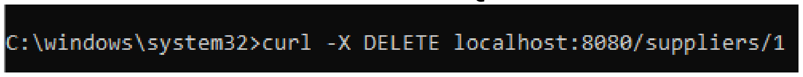
```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nabee>curl -X PUT localhost:8080/suppliers/1?supplierName=CHANGED

C:\Users\nabee>curl -X PUT localhost:8080/suppliers/1?supplierBase=CHANGED

C:\Users\nabee>
```



SELECT * FROM SUPPLIER;

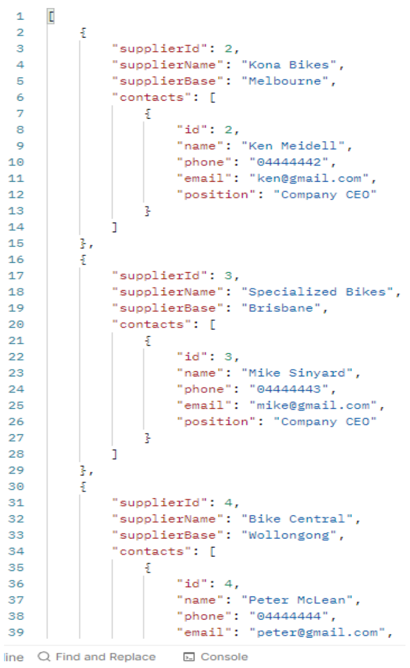
SUPPLIER_ID	SUPPLIER_BASE	SUPPLIER_NAME
1	CHANGED	CHANGED
2	Melbourne	Kona Bikes
3	Brisbane	Specialized Bikes
4	Wollongong	Bike Central

DELETE SUPPLIER - DELETE REQUEST



```
C:\windows\system32>curl -X DELETE localhost:8080/suppliers/1
```

On POSTMAN Software ->

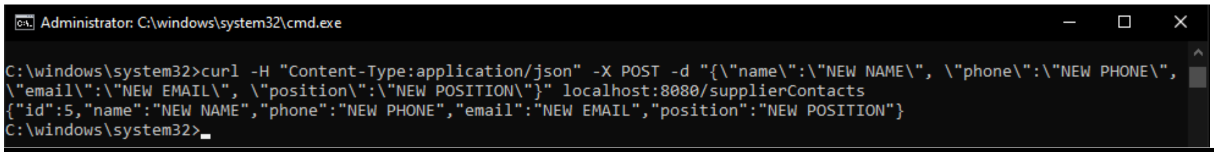


```
[
    {
        "supplierId": 2,
        "supplierName": "Kona Bikes",
        "supplierBase": "Melbourne",
        "contacts": [
            {
                "id": 2,
                "name": "Ken Meidell",
                "phone": "04444442",
                "email": "ken@gmail.com",
                "position": "Company CEO"
            }
        ]
    },
    {
        "supplierId": 3,
        "supplierName": "Specialized Bikes",
        "supplierBase": "Brisbane",
        "contacts": [
            {
                "id": 3,
                "name": "Mike Sinyard",
                "phone": "04444443",
                "email": "mike@gmail.com",
                "position": "Company CEO"
            }
        ]
    },
    {
        "supplierId": 4,
        "supplierName": "Bike Central",
        "supplierBase": "Wollongong",
        "contacts": [
            {
                "id": 4,
                "name": "Peter McLean",
                "phone": "04444444",
                "email": "peter@gmail.com",
```

CREATE/UPDATE SUPPLIER CONTACT:

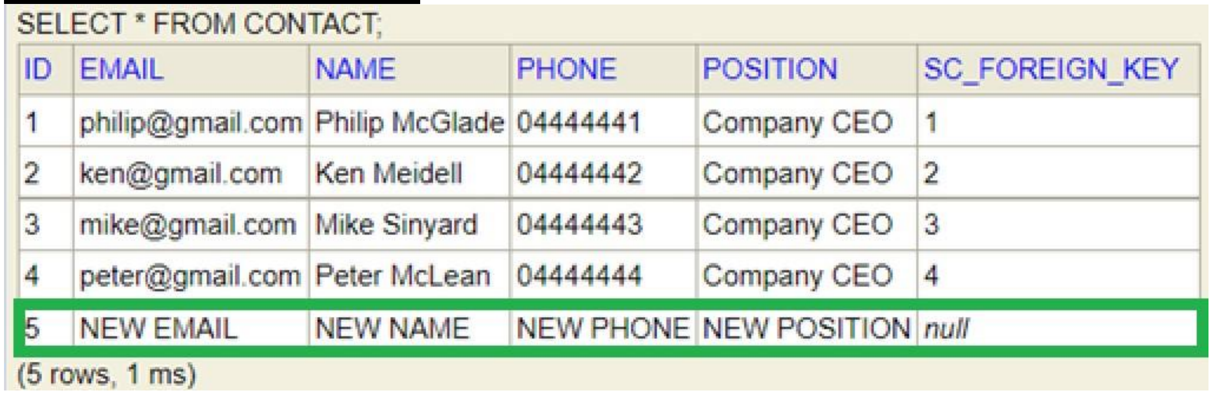
2	Create/update supplier contact	Procurement

CREATE CONTACT - POST REQUEST



```
C:\windows\system32>curl -H "Content-Type:application/json" -X POST -d "{\"name\":\"NEW NAME\", \"phone\":\"NEW PHONE\",
\"email\":\"NEW EMAIL\", \"position\":\"NEW POSITION\"}" localhost:8080/supplierContacts
{"id":5,"name":"NEW NAME","phone":"NEW PHONE","email":"NEW EMAIL","position":"NEW POSITION"}
C:\windows\system32>_
```

curl -H "Content-Type:application/json" -X POST -d "{\"name\":\"NEW NAME\", \"phone\":\"NEW PHONE\", \"email\":\"NEW EMAIL\", \"position\":\"NEW POSITION\"}" localhost:8080/supplierContacts

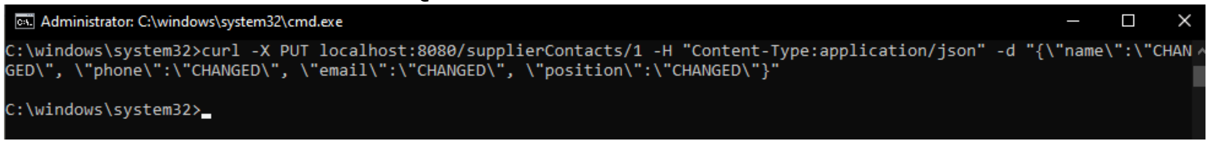


SELECT * FROM CONTACT;

ID	EMAIL	NAME	PHONE	POSITION	SC_FOREIGN_KEY
1	philip@gmail.com	Philip McGlade	04444441	Company CEO	1
2	ken@gmail.com	Ken Meidell	04444442	Company CEO	2
3	mike@gmail.com	Mike Sinyard	04444443	Company CEO	3
4	peter@gmail.com	Peter McLean	04444444	Company CEO	4
5	NEW EMAIL	NEW NAME	NEW PHONE	NEW POSITION	*null*

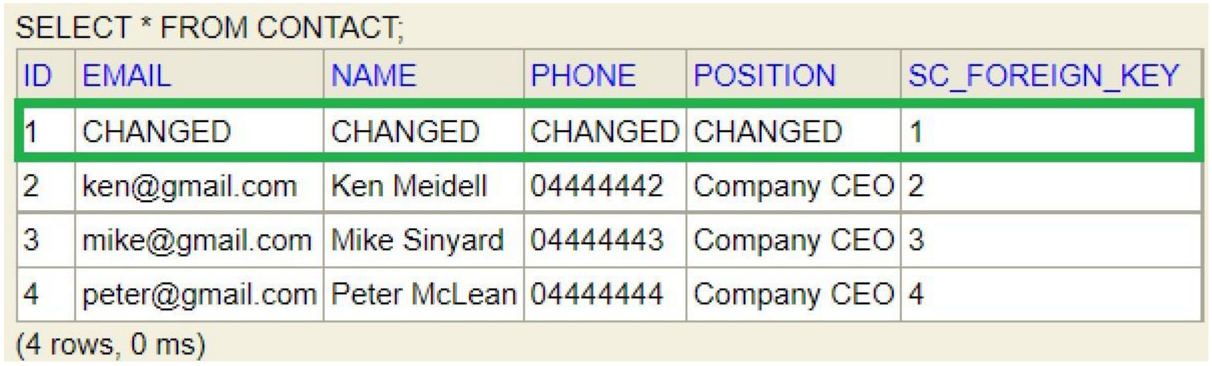
(5 rows, 1 ms)

UPDATE CONTACT - PUT REQUEST



```
C:\windows\system32>curl -X PUT localhost:8080/supplierContacts/1 -H "Content-Type:application/json" -d "{\"name\":\"CHAN
GED\", \"phone\":\"CHANGED\", \"email\":\"CHANGED\", \"position\":\"CHANGED\"}"

C:\windows\system32>_
```

```
curl -X PUT localhost:8080/supplierContacts/1 -H "Content-Type:application/json" -d
"{\"name\":\"CHANGED\", \"phone\":\"CHANGED\", \"email\":\"CHANGED\",
\"position\":\"CHANGED\"}"
```



SELECT * FROM CONTACT;

ID	EMAIL	NAME	PHONE	POSITION	SC_FOREIGN_KEY
1	CHANGED	CHANGED	CHANGED	CHANGED	1
2	ken@gmail.com	Ken Meidell	04444442	Company CEO	2
3	mike@gmail.com	Mike Sinyard	04444443	Company CEO	3
4	peter@gmail.com	Peter McLean	04444444	Company CEO	4

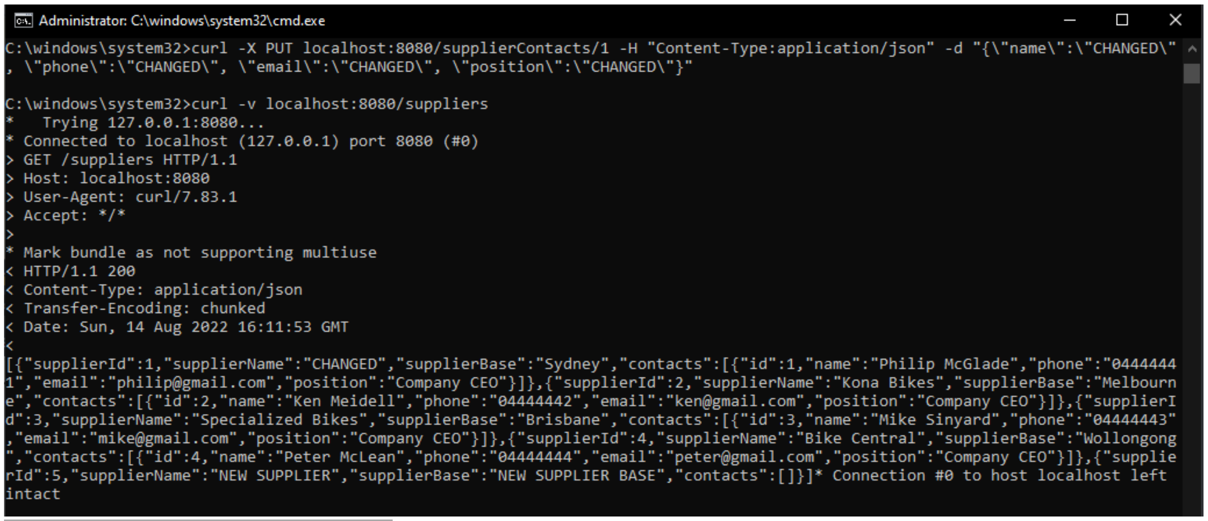
(4 rows, 0 ms)

LOOK UP SUPPLIER BASIC INFO AND CONTACT:

3	Look up supplier basic info and contact	Procurement

GET REQUEST – Look up all suppliers

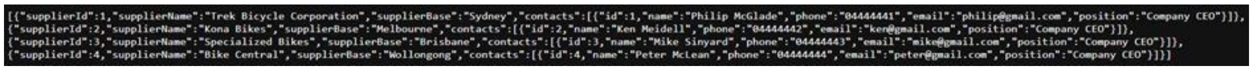
```
curl -v localhost:8080/suppliers
```



Administrator: C:\windows\system32\cmd.exe

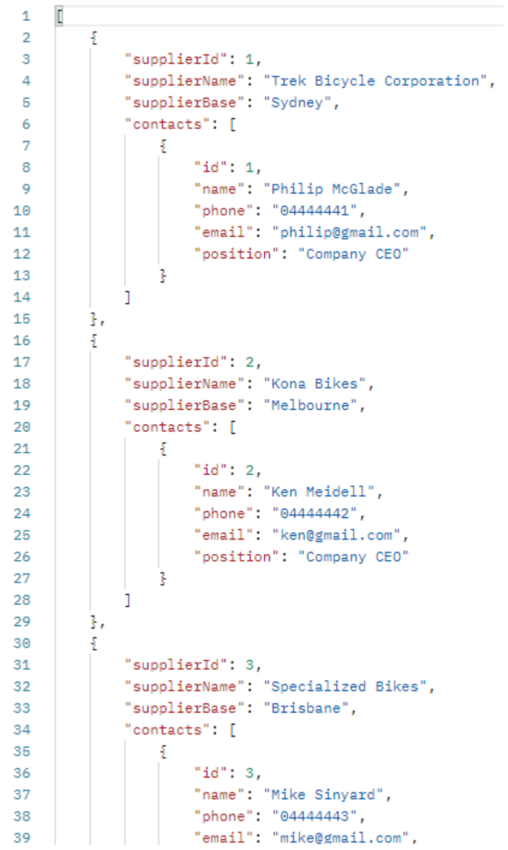
```
C:\windows\system32>curl -X PUT localhost:8080/supplierContacts/1 -H "Content-Type:application/json" -d "{\"name\":\"CHANGED\"
, \"phone\":\"CHANGED\", \"email\":\"CHANGED\", \"position\":\"CHANGED\"}"

C:\windows\system32>curl -v localhost:8080/suppliers
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /suppliers HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Sun, 14 Aug 2022 16:11:53 GMT
<
[{"supplierId":1,"supplierName":"CHANGED","supplierBase":"Sydney","contacts":[{"id":1,"name":"Philip McGlade","phone":"0444444
1","email":"philip@gmail.com","position":"Company CEO"}]},{"supplierId":2,"supplierName":"Kona Bikes","supplierBase":"Melbourn
e","contacts":[{"id":2,"name":"Ken Meidell","phone":"04444442","email":"ken@gmail.com","position":"Company CEO"}]},{"supplierI
d":3,"supplierName":"Specialized Bikes","supplierBase":"Brisbane","contacts":[{"id":3,"name":"Mike Sinyard","phone":"04444443"
,"email":"mike@gmail.com","position":"Company CEO"}]},{"supplierId":4,"supplierName":"Bike Central","supplierBase":"Wollongong
","contacts":[{"id":4,"name":"Peter McLean","phone":"04444444","email":"peter@gmail.com","position":"Company CEO"}]},{"supplie
rId":5,"supplierName":"NEW SUPPLIER","supplierBase":"NEW SUPPLIER BASE","contacts":[]}]* Connection #0 to host localhost left
intact
```

http://localhost:8080/suppliers



```
[{"supplierId":1,"supplierName":"Trek Bicycle Corporation","supplierBase":"Sydney","contacts":[{"id":1,"name":"Philip McGlade","phone":"04444441","email":"philip@gmail.com","position":"Company CEO"}]},
{"supplierId":2,"supplierName":"Kona Bikes","supplierBase":"Melbourne","contacts":[{"id":2,"name":"Ken Meidell","phone":"04444442","email":"ken@gmail.com","position":"Company CEO"}]},
{"supplierId":3,"supplierName":"Specialized Bikes","supplierBase":"Brisbane","contacts":[{"id":3,"name":"Mike Sinyard","phone":"04444443","email":"mike@gmail.com","position":"Company CEO"}]},
{"supplierId":4,"supplierName":"Bike Central","supplierBase":"Wollongong","contacts":[{"id":4,"name":"Peter McLean","phone":"04444444","email":"peter@gmail.com","position":"Company CEO"}]}]
```

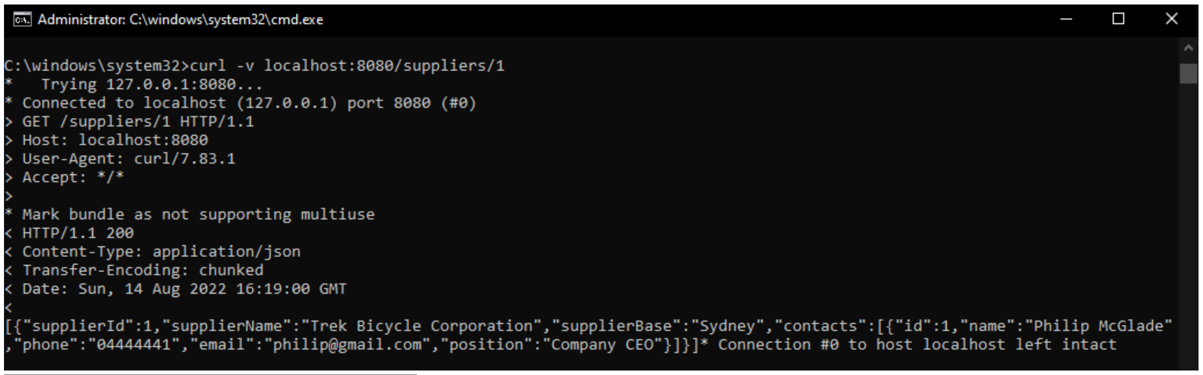
On POSTMAN Software ->



```
[
    {
        "supplierId": 1,
        "supplierName": "Trek Bicycle Corporation",
        "supplierBase": "Sydney",
        "contacts": [
            {
                "id": 1,
                "name": "Philip McGlade",
                "phone": "04444441",
                "email": "philip@gmail.com",
                "position": "Company CEO"
            }
        ]
    },
    {
        "supplierId": 2,
        "supplierName": "Kona Bikes",
        "supplierBase": "Melbourne",
        "contacts": [
            {
                "id": 2,
                "name": "Ken Meidell",
                "phone": "04444442",
                "email": "ken@gmail.com",
                "position": "Company CEO"
            }
        ]
    },
    {
        "supplierId": 3,
        "supplierName": "Specialized Bikes",
        "supplierBase": "Brisbane",
        "contacts": [
            {
                "id": 3,
                "name": "Mike Sinyard",
                "phone": "04444443",
                "email": "mike@gmail.com",
```

Look up supplier by ID

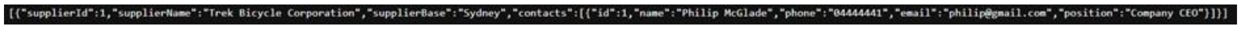
curl -v localhost:8080/suppliers/1



Administrator: C:\windows\system32\cmd.exe

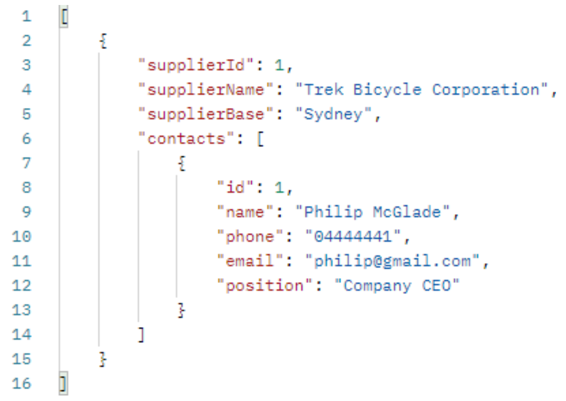
```
C:\windows\system32>curl -v localhost:8080/suppliers/1
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /suppliers/1 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Sun, 14 Aug 2022 16:19:00 GMT
<
[{"supplierId":1,"supplierName":"Trek Bicycle Corporation","supplierBase":"Sydney","contacts":[{"id":1,"name":"Philip McGlade"
,"phone":"04444441","email":"philip@gmail.com","position":"Company CEO"}]}]* Connection #0 to host localhost left intact
```

http://localhost:8080/suppliers/1



```
[{"supplierId":1,"supplierName":"Trek Bicycle Corporation","supplierBase":"Sydney","contacts":[{"id":1,"name":"Philip McGlade","phone":"04444441","email":"philip@gmail.com","position":"Company CEO"}]}]
```

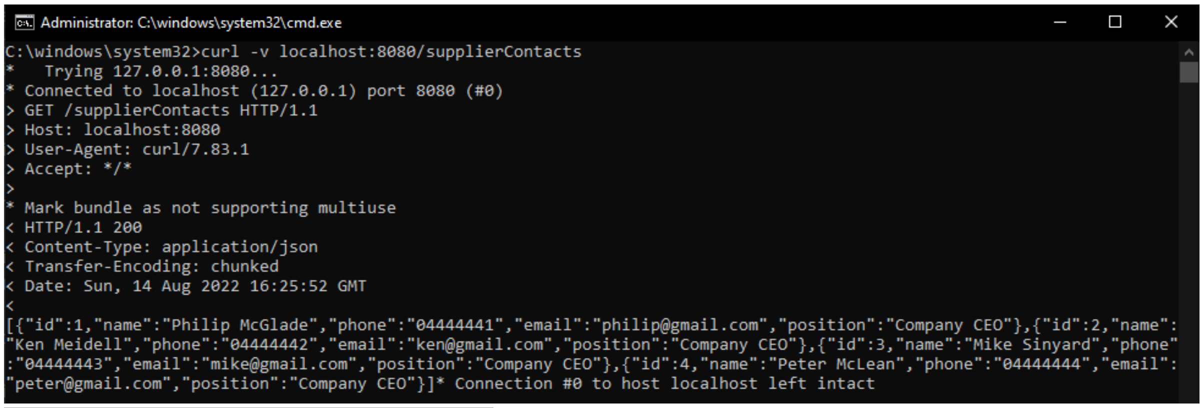
On POSTMAN Software ->



```
[
    {
        "supplierId": 1,
        "supplierName": "Trek Bicycle Corporation",
        "supplierBase": "Sydney",
        "contacts": [
            {
                "id": 1,
                "name": "Philip McGlade",
                "phone": "04444441",
                "email": "philip@gmail.com",
                "position": "Company CEO"
            }
        ]
    }
]
```

Look up all contacts

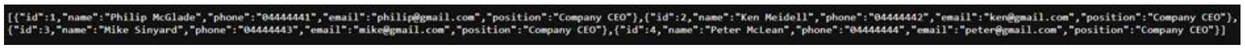
curl -v localhost:8080/supplierContacts



Administrator: C:\windows\system32\cmd.exe

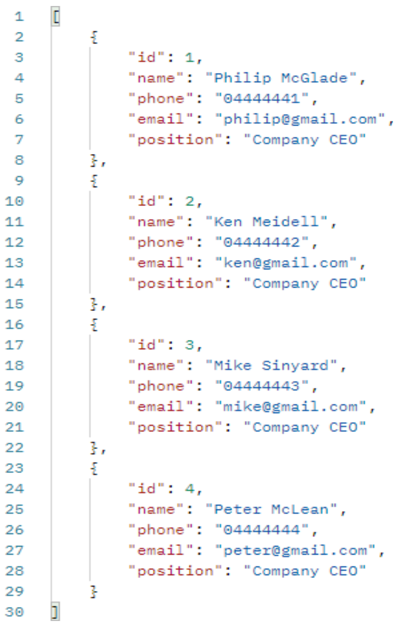
```
C:\windows\system32>curl -v localhost:8080/supplierContacts
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /supplierContacts HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Sun, 14 Aug 2022 16:25:52 GMT
<
[{"id":1,"name":"Philip McGlade","phone":"04444441","email":"philip@gmail.com","position":"Company CEO"},{"id":2,"name":
"Ken Meidell","phone":"04444442","email":"ken@gmail.com","position":"Company CEO"},{"id":3,"name":"Mike Sinyard","phone"
:"04444443","email":"mike@gmail.com","position":"Company CEO"},{"id":4,"name":"Peter McLean","phone":"04444444","email":
"peter@gmail.com","position":"Company CEO"}]* Connection #0 to host localhost left intact
```

http://localhost:8080/supplierContacts



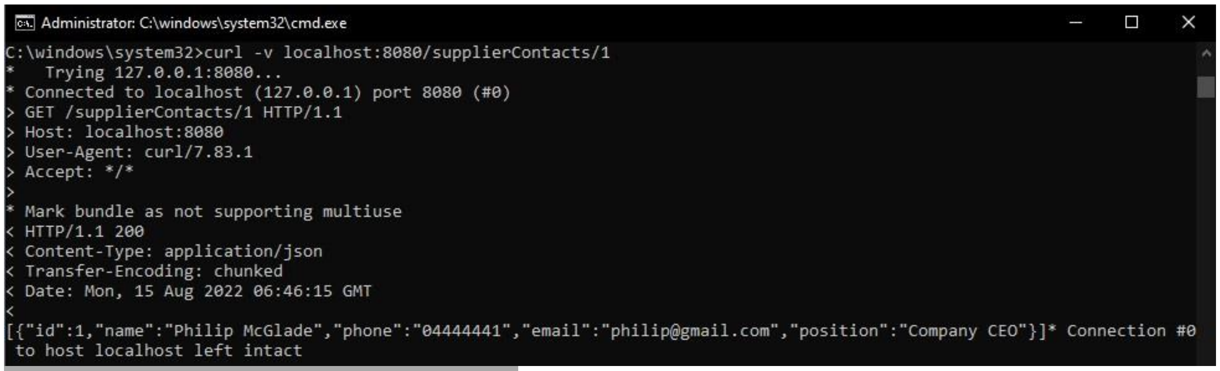
```
[{"id":1,"name":"Philip McGlade","phone":"04444441","email":"philip@gmail.com","position":"Company CEO"},{"id":2,"name":"Ken Meidell","phone":"04444442","email":"ken@gmail.com","position":"Company CEO"},
{"id":3,"name":"Mike Sinyard","phone":"04444443","email":"mike@gmail.com","position":"Company CEO"},{"id":4,"name":"Peter McLean","phone":"04444444","email":"peter@gmail.com","position":"Company CEO"}]
```

On POSTMAN Software ->



```
[
    {
        "id": 1,
        "name": "Philip McGlade",
        "phone": "04444441",
        "email": "philip@gmail.com",
        "position": "Company CEO"
    },
    {
        "id": 2,
        "name": "Ken Meidell",
        "phone": "04444442",
        "email": "ken@gmail.com",
        "position": "Company CEO"
    },
    {
        "id": 3,
        "name": "Mike Sinyard",
        "phone": "04444443",
        "email": "mike@gmail.com",
        "position": "Company CEO"
    },
    {
        "id": 4,
        "name": "Peter McLean",
        "phone": "04444444",
        "email": "peter@gmail.com",
        "position": "Company CEO"
    }
]
```

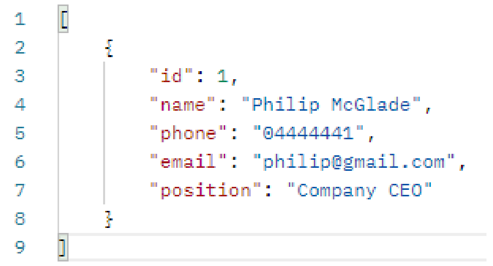
curl -v localhost:8080/supplierContacts/1



```
C:\windows\system32>curl -v localhost:8080/supplierContacts/1
*   Trying 127.0.0.1:8080...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET /supplierContacts/1 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Mon, 15 Aug 2022 06:46:15 GMT
<
[{"id":1,"name":"Philip McGlade","phone":"04444441","email":"philip@gmail.com","position":"Company CEO"}]* Connection #0
 to host localhost left intact
```

http://localhost:8080/supplierContacts/1

On POSTMAN Software ->



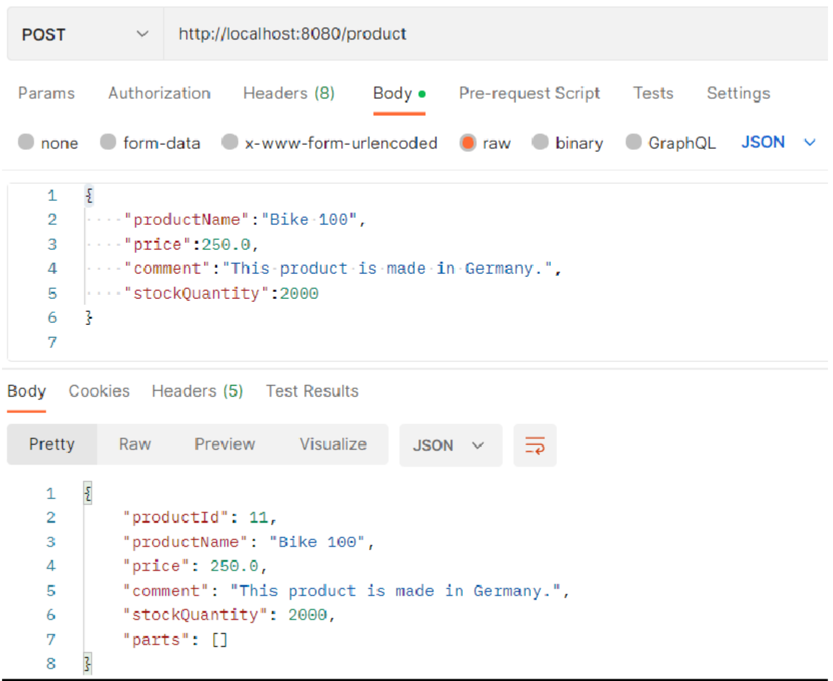
```
[
    {
        "id": 1,
        "name": "Philip McGlade",
        "phone": "04444441",
        "email": "philip@gmail.com",
        "position": "Company CEO"
    }
]
```

CREATE/UPDATE PART/PRODUCT:

4	Create/update part/product	Inventory*

On POSTMAN Software ->

Create Product

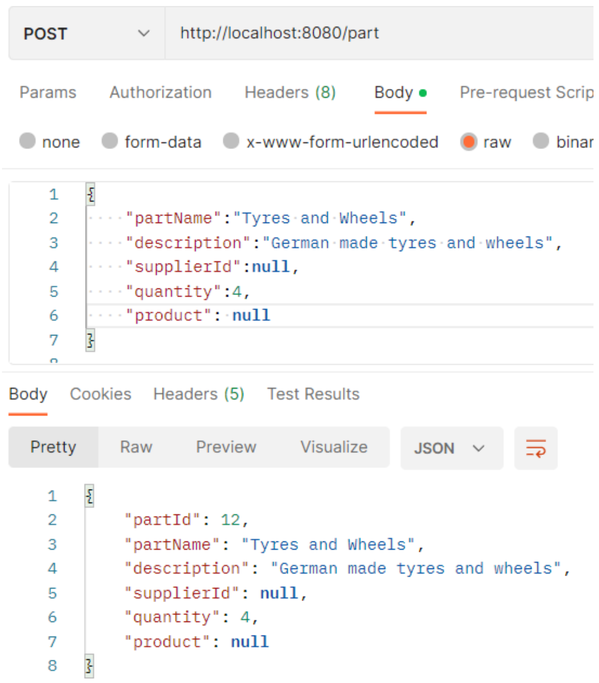


http://localhost:8080/product

```
{
    "productName":"Bike 100",
    "price":250.0,
    "comment":"This product is made in Germany.",
    "stockQuantity":2000
}
```

```
{
    "productId": 11,
    "productName": "Bike 100",
    "price": 250.0,
    "comment": "This product is made in Germany.",
    "stockQuantity": 2000,
    "parts": []
}
```

Create Part



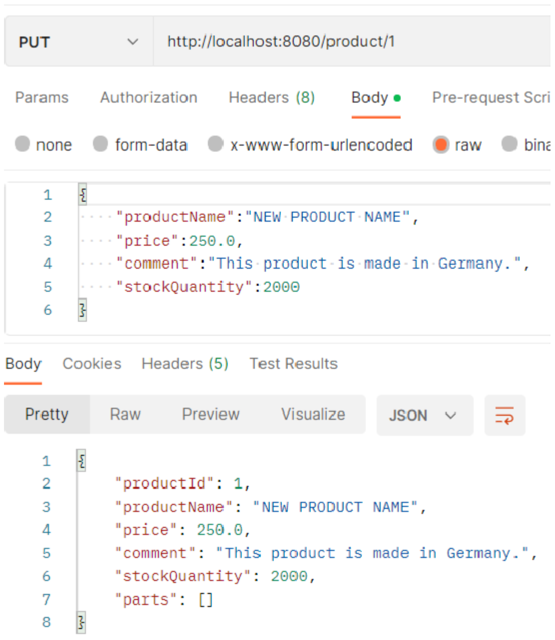
http://localhost:8080/part

```
{
    "partName":"Tyres and Wheels",
    "description":"German made tyres and wheels",
    "supplierId":null,
    "quantity":4,
    "product": null
}
```

```
{
    "partId": 12,
    "partName": "Tyres and Wheels",
    "description": "German made tyres and wheels",
    "supplierId": null,
    "quantity": 4,
    "product": null
}
```

Update Product

http://localhost:8080/product/1



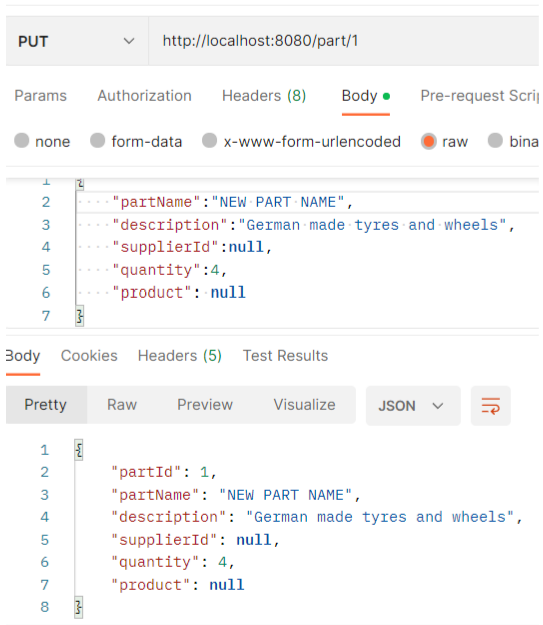
PUT http://localhost:8080/product/1

```
{
    "productName":"NEW PRODUCT NAME",
    "price":250.0,
    "comment":"This product is made in Germany.",
    "stockQuantity":2000
}
```

```
{
    "productId": 1,
    "productName": "NEW PRODUCT NAME",
    "price": 250.0,
    "comment": "This product is made in Germany.",
    "stockQuantity": 2000,
    "parts": []
}
```

Update Part

http://localhost:8080/part/1



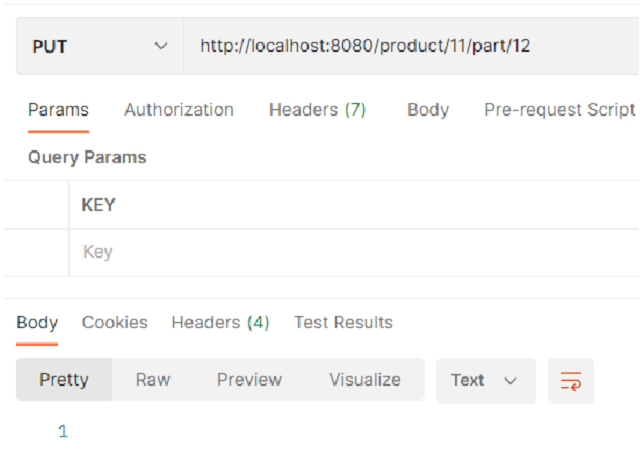
PUT http://localhost:8080/part/1

```
    "partName":"NEW PART NAME",
    "description":"German made tyres and wheels",
    "supplierId":null,
    "quantity":4,
    "product": null
}
```

```
{
    "partId": 1,
    "partName": "NEW PART NAME",
    "description": "German made tyres and wheels",
    "supplierId": null,
    "quantity": 4,
    "product": null
}
```

Put Part to Product

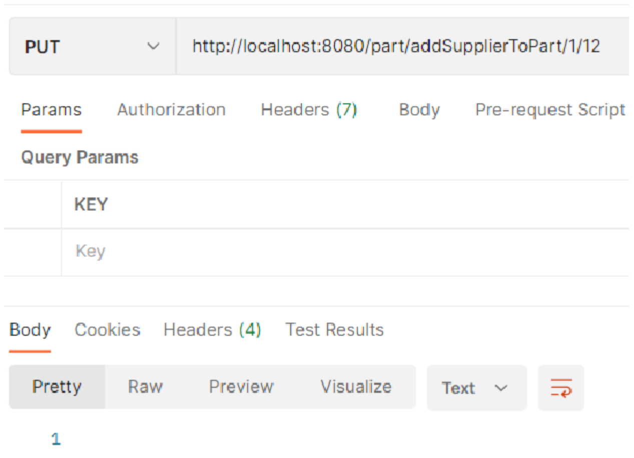
http://localhost:8080/product/11/part/12


PUT http://localhost:8080/product/11/part/12

```
1
```

Put Supplier to Part

http://localhost:8080/part/addSupplierToPart/1/12


PUT http://localhost:8080/part/addSupplierToPart/1/12

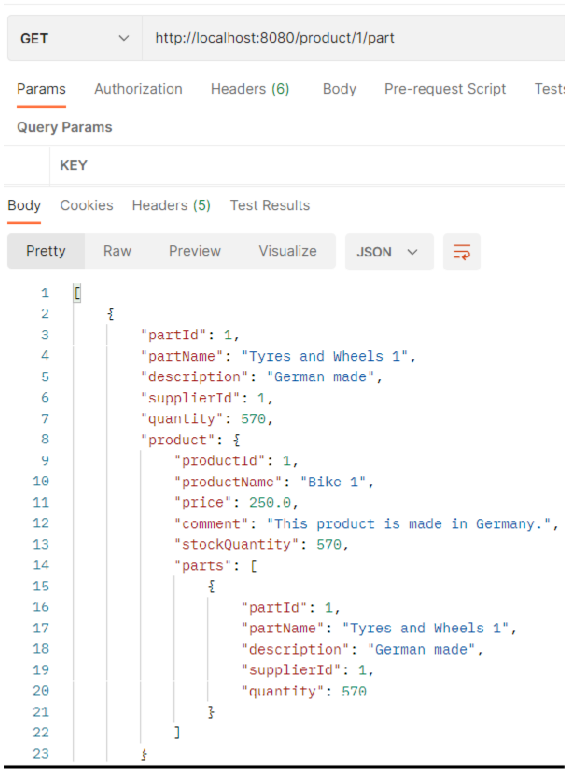
```
1
```

LOOK UP ALL PARTS BY PRODUCT:

5	Look up all parts by product	Inventory*

On POSTMAN Software ->

http://localhost:8080/product/1/part



http://localhost:8080/product/1/part

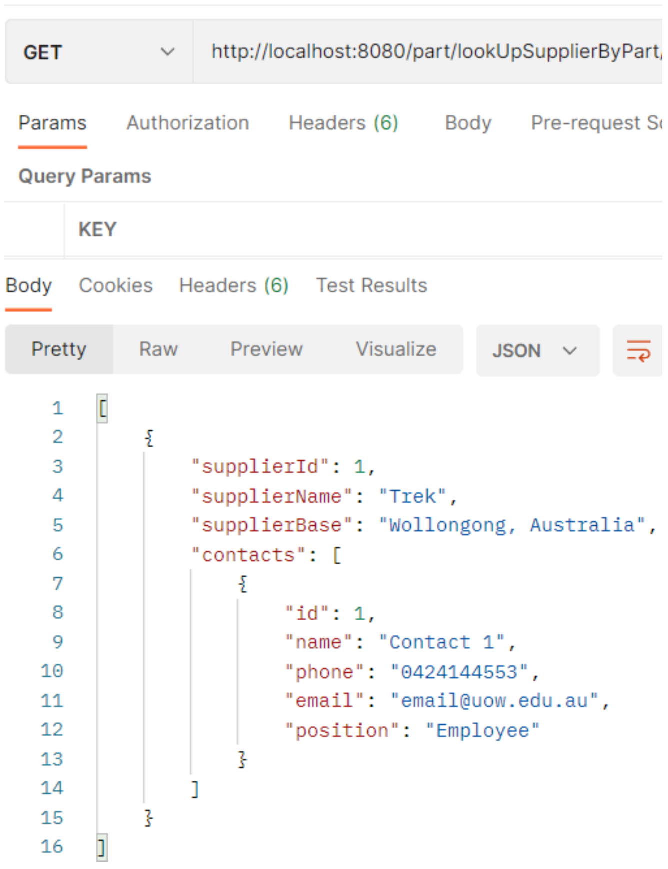
```
[
    {
        "partId": 1,
        "partName": "Tyres and Wheels 1",
        "description": "German made",
        "supplierId": 1,
        "quantity": 570,
        "product": {
            "productId": 1,
            "productName": "Bike 1",
            "price": 250.0,
            "comment": "This product is made in Germany.",
            "stockQuantity": 570,
            "parts": [
                {
                    "partId": 1,
                    "partName": "Tyres and Wheels 1",
                    "description": "German made",
                    "supplierId": 1,
                    "quantity": 570
                }
            ]
        }
```

LOOK UP SUPPLIER BY PART:

6	Look up supplier by part	Procurement*

On POSTMAN Software ->

http://localhost:8080/part/lookUpSupplierByPart/1



```
[
    {
        "supplierId": 1,
        "supplierName": "Trek",
        "supplierBase": "Wollongong, Australia",
        "contacts": [
            {
                "id": 1,
                "name": "Contact 1",
                "phone": "0424144553",
                "email": "email@uow.edu.au",
                "position": "Employee"
            }
        ]
    }
]
```

CREATE SALE:

7	Create sale	Sales*

Note: The Create Sale use case has been modified to follow Part C specifications. Continuous order REST requests are sent to the Sales Service every 2 seconds to make POST requests for InStoreSale and OnlineSale in a loop as shown in the code below:

```
//every 2 seconds a random inStore/online order is created
try {
    while (!Thread.currentThread().isInterrupted()) {
        log.info("------------ ONLINE ORDER-------------");
        createRandomOrder();
        Thread.sleep(2000);
        log.info("------------ IN-STORE ORDER-------------");
        createRandomInStoreOrder();
        Thread.sleep(2000);
    }
} catch (InterruptedException ignored) {
}
```

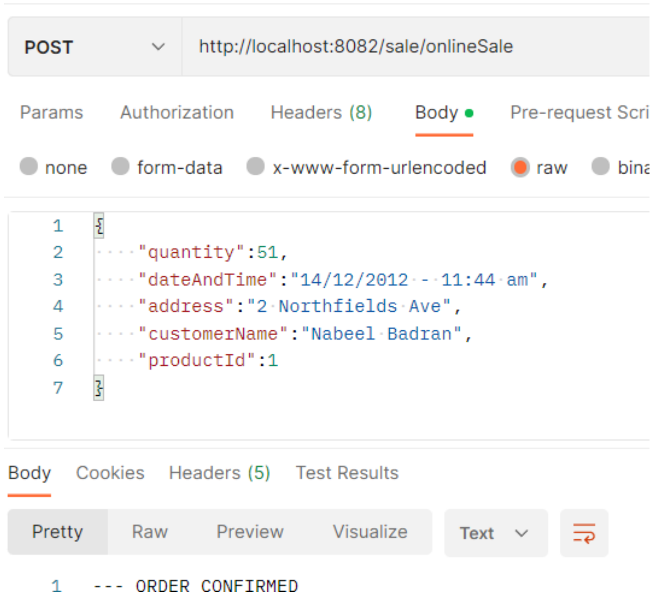
Example of one of the confirmed orders is shown below:

Console Output ->

```
2022-10-30 17:20:16.062  INFO 520 --- [           main] group09.SalesService.OrderLoader         :
------------ ONLINE ORDER ------------
Product ID is VALID, we are now checking quantity of both PRODUCT and PART
RESPONSE FROM INVENTORY ----( true ) —
PRODUCT NAME = Bike 7
PRODUCT PRICE = 250.0
--- ORDER CONFIRMED
```

CREATE ONLINE SALE MANUALLY USING POSTMAN:

http://localhost:8082/sale/onlineSale



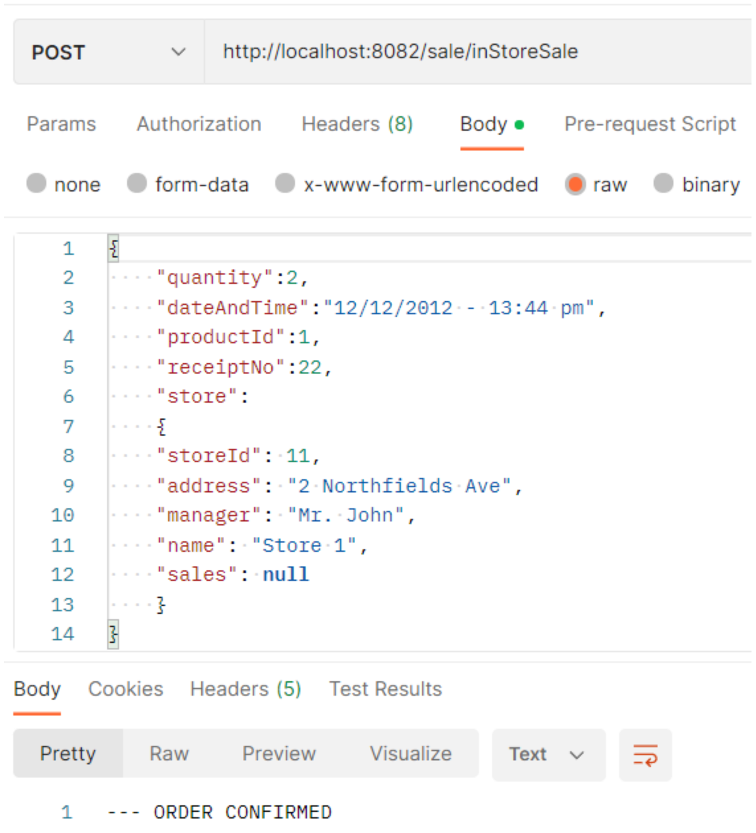
POST http://localhost:8082/sale/onlineSale

```
{
    "quantity":51,
    "dateAndTime":"14/12/2012 - 11:44 am",
    "address":"2 Northfields Ave",
    "customerName":"Nabeel Badran",
    "productId":1
}
```

```
--- ORDER CONFIRMED
```

CREATE IN STORE SALE MANUALLY USING POSTMAN:

http://localhost:8082/sale/inStoreSale



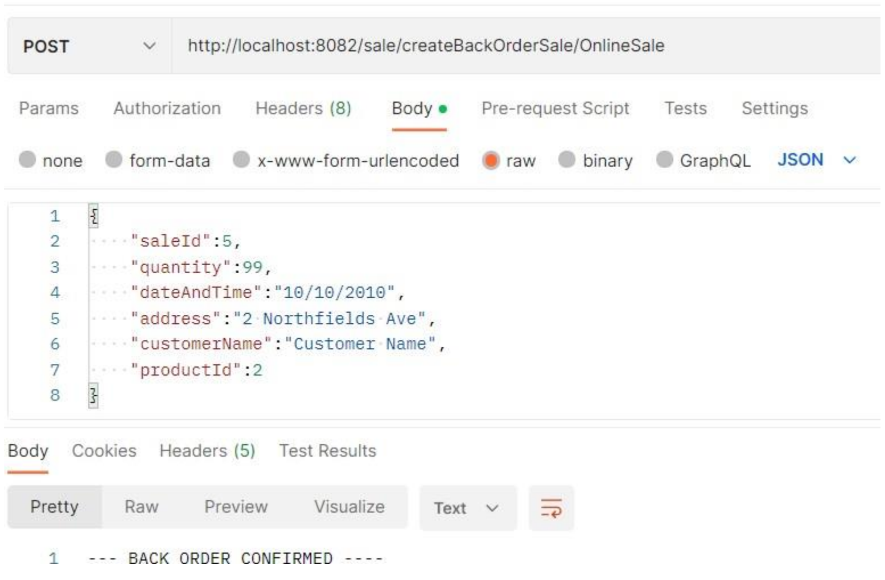
http://localhost:8082/sale/inStoreSale

```
{
    "quantity":2,
    "dateAndTime":"12/12/2012 - 13:44 pm",
    "productId":1,
    "receiptNo":22,
    "store":
    {
    "storeId": 11,
    "address": "2 Northfields Ave",
    "manager": "Mr. John",
    "name": "Store 1",
    "sales": null
    }
}
```

```
--- ORDER CONFIRMED
```

EXAMPLE OF CREATE BACK ORDER SALE:

http://localhost:8082/sale/createBackOrderSale/OnlineSale



http://localhost:8082/sale/createBackOrderSale/OnlineSale

```
{
    "saleId":5,
    "quantity":99,
    "dateAndTime":"10/10/2010",
    "address":"2 Northfields Ave",
    "customerName":"Customer Name",
    "productId":2
}
```

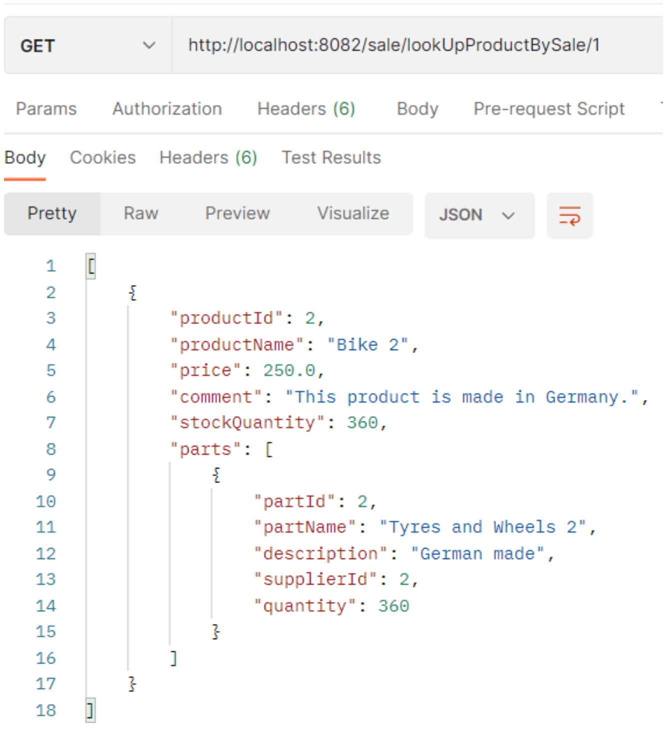
```
--- BACK ORDER CONFIRMED ----
```

LOOK UP PRODUCTS INFO BY SALE:

8	Look up products info by sale	Sales*

On POSTMAN Software ->

http://localhost:8082/sale/lookUpProductBySale/1


GET http://localhost:8082/sale/lookUpProductBySale/1

Params Authorization Headers (6) Body Pre-request Script

Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize JSON

```
[
    {
        "productId": 2,
        "productName": "Bike 2",
        "price": 250.0,
        "comment": "This product is made in Germany.",
        "stockQuantity": 360,
        "parts": [
            {
                "partId": 2,
                "partName": "Tyres and Wheels 2",
                "description": "German made",
                "supplierId": 2,
                "quantity": 360
            }
        ]
    }
]
```

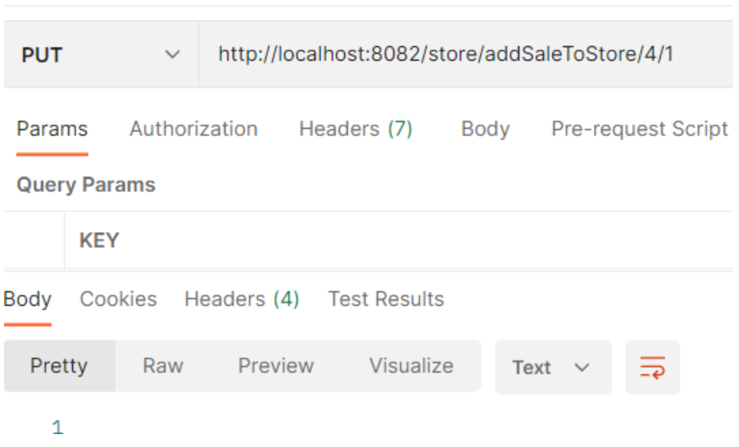
LOOK UP SALES BY STORE:

9	Look up sales by store	Sales*

On POSTMAN Software ->

<u>Add Sale to Store Manually</u>

http://localhost:8082/store/addSaleToStore/4/1


PUT http://localhost:8082/store/addSaleToStore/4/1

Params Authorization Headers (7) Body Pre-request Script

Query Params

KEY

Body Cookies Headers (4) Test Results

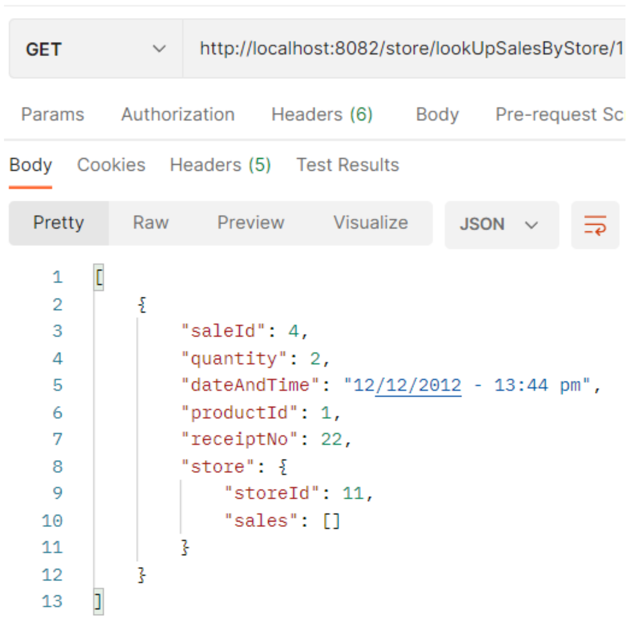
Pretty Raw Preview Visualize Text

```
1
```

Look up Sales by Store

Note: for this use case, it is best to comment out the loop that creates orders in OrderLoader.java in SalesService and do it manually.

http://localhost:8082/store/lookUpSalesByStore/1



http://localhost:8082/store/lookUpSalesByStore/1

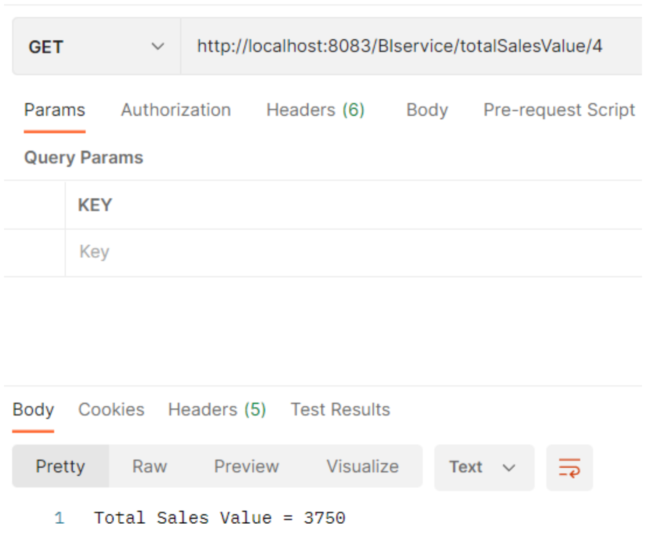
```
[
    {
        "saleId": 4,
        "quantity": 2,
        "dateAndTime": "12/12/2012 - 13:44 pm",
        "productId": 1,
        "receiptNo": 22,
        "store": {
            "storeId": 11,
            "sales": []
        }
    }
]
```

LOOK UP THE TOTAL SALES VALUE PER PRODUCT AT REAL TIME:

10	Look up the total sales value per product at real time	BI

On POSTMAN Software ->

http://localhost:8083/BIservice/totalSalesValue/4



http://localhost:8083/BIservice/totalSalesValue/4

```
Total Sales Value = 3750
```

3.4. The Design Of The Services

The design of the sales, procurement, inventory business intelligence service are based on the following use cases:

- Create/update supplier
- Create/update supplier contact
- Look up supplier basic info and contact
- Create/update part/product
- Look up all parts by product
- Look up supplier by part
- Create sale
- Look up products info by sale
- Look up sales by store
- Look up the total sales value per product at real time

The design for the procurement, inventory and sales and business services are comprised of the following:

1. Controllers which have special instances and many kinds of mapping which are used for the presentation layer. They employ REST at the presentation layer to allow other applications to interact with the services which in other words means to access the entity information and methods we want in the microservice
2. Entities that represent concepts in our domains are represented by their identities instead of their attributes which in code may be done by overriding functions. These entities use getter and setter methods that can be used within the package. These entries also use JSON data which we can test on the API Platform, 'Postman', to see whether these data entries work and are successful. After selecting Building and Running the program of the microservice, on Postman, we can choose 'POST', then we can then select option 'Body' then 'Raw' with 'JSON'. After this, we then enter our JSON data in accordance with the microservice we are testing with and then click 'SEND'. If the override 'String toString()' returns, the entries are working.

3. Domain Services which contain domain logic and concepts that aren’t available in value objects or entities which represent behaviour and hence no identity or state.

4. Value objects are represented by their known characteristics, these are determined by immutable attributes which in the code are final data types, value objects can also be overridden toString methods, the reason for this is the flat denormalization by creating a persistence format.

5. Factories where we have many constructors in an individual class for several classes, these constructors may be empty or contain assignments/logic which can be hidden from client code

6. Repositories are used to store and retrieve information from a database with simple or complex queries. These repositories ensure that entities are saved entirely. They also allow us, developers, to focus on the model instead of the framework.

7. Apache Kafka is an event streaming platform which means streaming in real-time, in the sense of Kafka, most businesses use this since it is scalable, fault-tolerant, and reliable where it allows for data pipelines, streaming analytics, data integration, mission-critical applications and more [6]. A couple of features of the many that there are, are publish, subscribe, store, process and analyse events. We managed to run and set up Kafka successfully but didn't manage to program the use cases with it.

8. Domain events are events in the domain that are in the interest of the business. this is where event handlers are implemented and where Apache Kafka is used

9. Event sourcing allows for multiple events to be stored in a database where replies or queries are flexible in terms of entities

10. Aggregates are defined by the relationships between entities, for example, one-to-many, one-to-one, many-to-many and many-to-one relationships. This allows for the combinations of entities and to meet invariants within the domain model. The use of aggregate patterns guarantees consistency.

11. Infrastructural integration which allows the necessary services to communicate between applications. This can be achieved by a shared database, file system, HTTP or messaging and also the application of REST functions.

As demonstrated above, our design conforms to the domain-driven design patterns because it uses a combination of tactical and strategic patterns for the software engineering design principles.

3.5. Implementation Of The Services

The implementations of the services are done in the following way:

For the **SalesService**:

We have controllers SaleController.java and StoreController.java where REST and Postman functions are handled. This includes the creation, updating, and deletion of sales and stores, as well as more specific search requests related to the inner workings of each entity. We also have SaleConfig.java to return new RestTemplates.

Furthermore, entity Sale.java sets up the base attributes for sale in the system, while entities InStoreSale.java and OnlineSale.java use the “extends” keyword to employ the inheritance stipulated in the diagram.

In addition, we have two (2) services for the domain which are SalesService.java and StoreService.java and they consist of immutable value objects: SalesService.java has a SaleRepository object, StoreService.java has a StoreRepository object, and both have a RestTemplate object.

The classes used above are contained in SaleRepository.java and StoreRepository.java. These classes both extend the imported class JpaRepository, where SalesRepository.java uses the term JpaRepository<Sale, Long> and StoreRepository.java uses JpaRepository<Store, Long>. We also have the SalesServiceApplication class, which runs the entire service.

The class contained in InStoreSaleRepository.java extend the imported class JpaRepository where InStoreSaleRepository.java uses the term JpaRepository<InStoreSale, Long>. The class is contained in OnlineSaleRepository.java and extends the imported class JpaRepository where OnlineStoreRepository.java uses the term JpaRepository<OnlineSale, Long>.

Here we have one (1) aggregate where the root is a foreign key storeId in the class Store which has a one-to-many association annotated by @OneToMany between the class Store and class Sale and allows for the joining of columns

Now for Infrastructural integration HTTP request is used on an API Platform called ‘Postman’. The platform allows us to put entries into the database associated with the system. This also allows us to test and see whether the program and data are working and performing as we expect. This will allow us to see if there are any problems in the programs that need attention in resolving those issues.

For the **InventoryService**:

We have controllers PartConfig.java, PartController.java, and ProductController.java which also use many kinds/ instances of mapping and special instances of controllers to handle the functions for REST and postman.

Furthermore, entities Part.java, Product.java, and InventoryServiceApplication.java represent the concept in the inventory domain and have the same idea as the rest of the services but with regard to this specific microservice logic.

We have entities for BackOrder.java and jsonWrapper.java. BackOrder.java will include the backOrderId, supplierId, partId, and quantityNeeded that can be returned back to the InventoryService.

In addition, we have two (2) services for the domain which are PartService.java and ProductService.java and they consist of immutable value objects: partService an attribute of the class PartService in the PartController class in the PartController.java, productService from the ProductService class in the ProductController class for the ProductController.java and we have partRepository from the PartRepository class, restTemplate from the RestTemplate class in class PartService for the PartRepository.java.

On the other hand, we have the ProductRepository.java and PartRepository.java repositories that contain the entities mentioned above by extending the imported class JpaRepository<Product, Long> with ProductRepository class and JpaRepository<Part, Long> extended from PartRepository. We also have the InventoryServiceApplication class, which runs the entire service.

Class Part in part.java has a many-to-one association using the @ManyToOne annotation and a key that allows for the joining of another class called product_product_id while class Product in product.java has a one-to-many association annotated by @OneToMany mapped by the attribute product class that makes for the relationship between the class Part and product.

Now for Infrastructural integration, we have used HTTP where we have the Inventory microservice (Part in specific) which uses it to communicate with the Inventory microservice to validate suppliers. if the supplier is valid in the supplier repository, then we can add a part to the supplier as shown in the screenshot.

For the **ProcurementService**:

We have controllers ContactController.java, SupplierConfig.java, and SupplierController.java which do the same handled functions as the inventory and sales service but with regards to the procurement logic.

Furthermore, entities Contact.java, and Supplier.java with a one-to-many association with Contact.java, and ProcurementServiceApplication.java work the same as the rest of the entities in the other services but also with regards to their own domain concept

In addition, we have two (2) services for the domain which are ContactService.java, SupplierService.java and they consist of immutable value objects: contactRepository an attribute of the class ContactRepository in the ContactService class in the ContactService.java, supplierRepository from the SupplierRepository class, contactRepository from the

ContactRepository class and restTemplate from the RestTemplate class in the SupplierService class for the SupplierService.java.

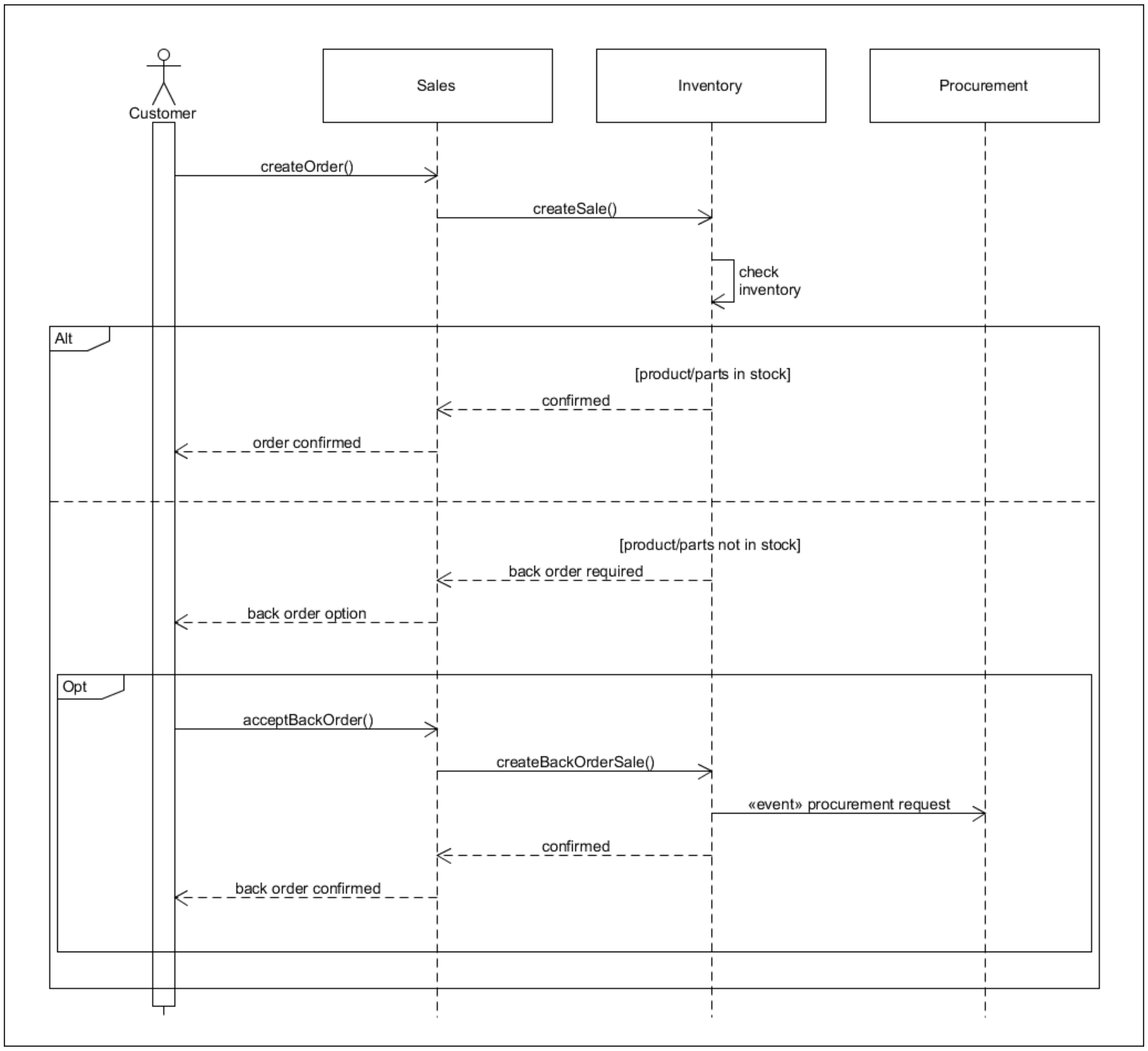
On the other hand, we have the ContactRepository.java, and SupplierRepository.java repositories that contain the entities mentioned above by extending the imported class JpaRepository<Supplier, Long> with SupplierRepository class and JpaRepository<Contact, Long> extended from ContactRepository.

In the procurement service, we have only one (1) aggregate in the class Supplier which has a one-to-many association denoted by @OneToMany with the target entity being the contact class and a foreign key supplierId allowing the Contact class to relate to the Supplier class.

The procurement microservice receives a procurement request from the inventory team. This happens when a customer orders a product and neither the product nor parts of the product are in stock, therefore customers accept a backorder option and the inventory team sends it to the procurement team as a procurement request. Procurement Microservice has a ‘BackOrderSale’ entity with its repository to save all of the back orders received from the inventory team.

This was conducted by our team using GitHub and Discord. On discord, we could download the file to our own computers and then work on specific services, sharing our work and advice over the Discord server before finally integrating the finished product back. Through GitHub, we cloned the main branch repository into our own branch, we could then modify the files on each individual’s end, and upload, modify, and edit the files that could be considered for uploading into the main branch for everyone to get a working version of the program files.

Our program follows the following sequence diagram:



- **The communications between the Sales and Inventory services are via REST;**
- **The communications between the ~~Sales~~ Inventory and Procurement services are via messaging.**

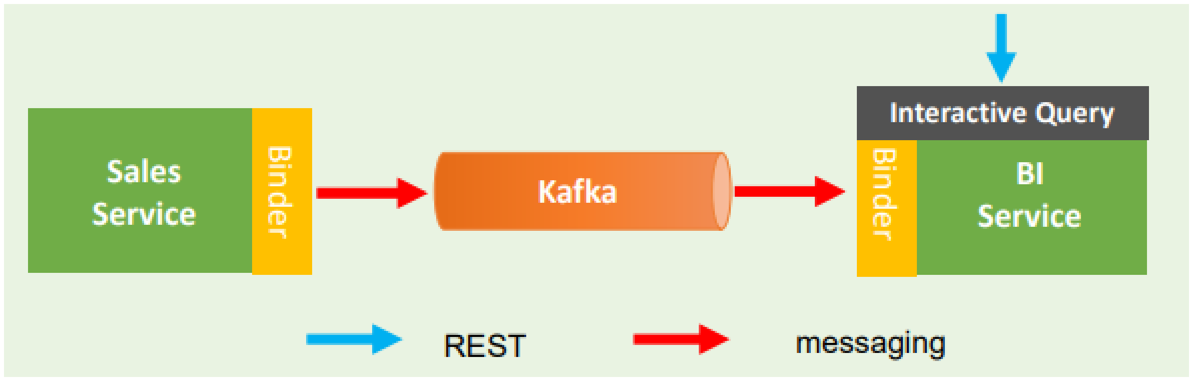
The Customer first starts by creating an order towards SalesService. Next, the SalesService will create a sale towards InventoryService. The InventoryService checks its inventory to see if its product/parts are or are in stock. This will be confirmed back to the SalesService. If the product/part is in stock, the SalesService will confirm the order back to the Customer.

The InventoryService will check if the product/part is not in stock. If the product/part is not in stock, a backorder is required from InventoryService to SalesService. The SalesService will give a backorder option to the Customer.

If the Customer accepts the backorder, the backorder will be sent from the Customer to the SalesService. The SalesService will create the backorder sale to the InventoryService. The InventoryService will create an event for the procurement request in the ProcurementService. The InventoryService will confirm the back order to the SalesService. Lastly, the SalesService will confirm the backorder with the Customer.

For the **BusinessIntelligenceService**:

The Business Intelligence (BI) Service looks into the business performance, more specifically the sales data from the online and physical stores.



The following diagram shows the architecture of the SalesService, BIService and Kafka.

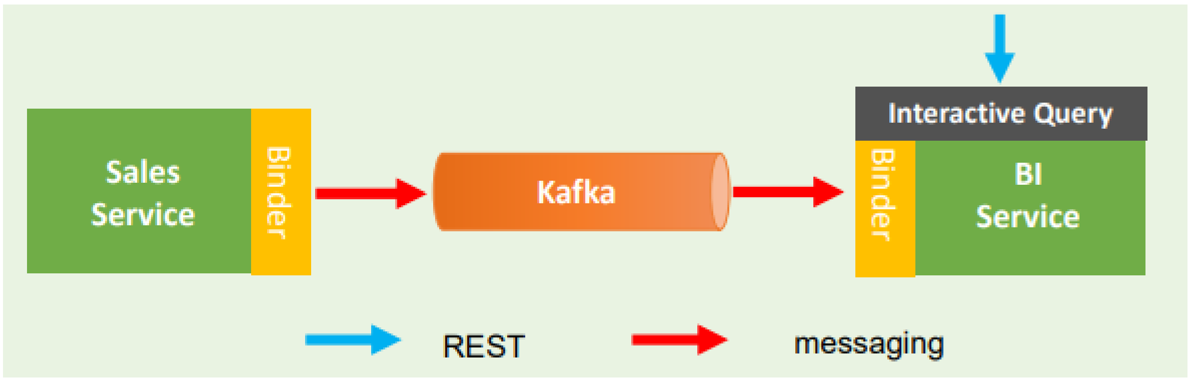
The BI Service consumes a SalesEvent stream and implements use case 10. Use Case 10 can be expressed as the following SQL-like statement:

```
SELECT SaleEvent.ProductName,
        SUM(SaleEvent.Quantity * SaleEvent.ProductPrice)
        GROUP BY SalesEvent.ProductName
```

The SQL-like statement refers to the BIService where the selected ProductName from the SaleEvent and the selected sum of the Quantity from SaleEvent multiplied by the ProductPrice from SaleEvent are grouped by the ProductName from the SalesEvent.
The implemented code of the above SQL statement is the following:

```
public String  getProductTotalSalesValue (Long productId) {
    Long totalSalesValue = 0L;
    KeyValueIterator<String, SaleEvent> all = productSaleEventStore().all();
    while (all.hasNext()) {
        SaleEvent saleEvent = all.next().value;
        Long table_productId = saleEvent.getProductId();
        long quantity = saleEvent.getQuantity();
        long price = saleEvent.getProductPrice();
        if (table_productId == (productId)) {
            totalSalesValue += (quantity * price);
        }
    }
    String result = "Total Sales Value = " + totalSalesValue.toString();
    return result;
}
```

Use case 10 is implemented as an *interactive query* with a REST API endpoint.



The controller SaleQueryController.java controls the Business Intelligence (BI) Service. The method @GetMapping “/totalSalesValue/{productId}” returns the sales value of a product in real-time. The method @GetMapping “/productNames” returns all the product names that have been ordered.

The entity SalesEvent.java returns the productId, productName, productPrice, and quantity. This utilises the attribute of ProductName for use case 10 that consumes a SaleEvent stream while implementing use case 10.

The service in BI includes SaleInteractiveQuery.java which creates and gathers the SaleEvent, productId, quantity, and ProductPrice. This also retrieves ProductNames and Store ID numbers. The service SaleStreamProcessing.java displays the state of the store products. This will return the productId, quantity, ProductPrice, and ProductName.

These controllers, entities, and services are utilised in the BusinessServiceApplication.java which runs the entire Business Intelligence Service.

4. Procedure To Set Up The Platform, And build And Run The Application

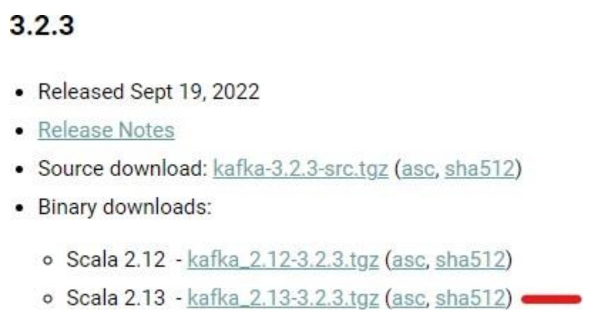
Setting Up Apache Kafka

4.1. How to Install Apache Kafka on Windows

4.1.1. Setting Up The Platform For Project

To operate kafka in the project you must download and run the server. To run the kafka and zookeeper servers the following commands can be used in your terminal:

Step 1: Download Apache Kafka, as mentioned above, from its official site: https://kafka.apache.org/downloads



3.2.3

- Released Sept 19, 2022
- Release Notes
- Source download: kafka-3.2.3-src.tgz (asc, sha512)
- Binary downloads:
  - Scala 2.12 - kafka_2.12-3.2.3.tgz (asc, sha512)
  - Scala 2.13 - kafka_2.13-3.2.3.tgz (asc, sha512)

Step 2: Extract tgz via cmd or from the available tool to a location of your choice:

```
tar -xvzf kafka_2.12-2.4.1.tgz
```

Step 3:

Open terminal and do the following:

Navigate to the folder where you saved the downloaded kafka file

Windows

```
C:\kafka\bin\windows\zookeeper-server-start.bat C:\kafka\config\zookeeper.properties
```

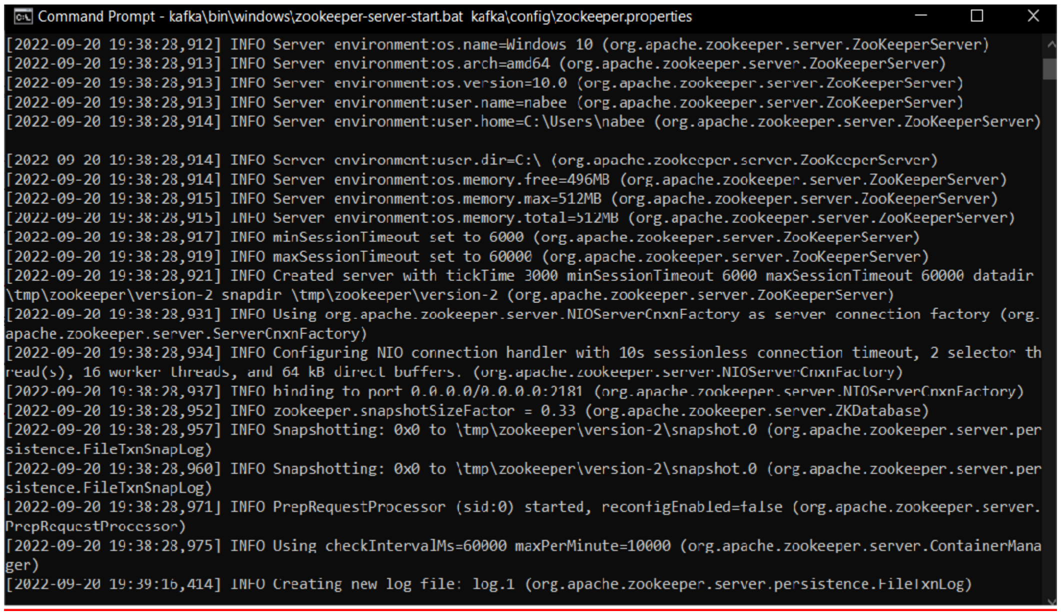
```
C:\kafka\bin\windows\kafka-server-start.bat C:\kafka\config\server.properties
```

Linux and MacOS

```
./bin/zookeeper-server-start.sh
```

./bin/kafka-server-start.sh

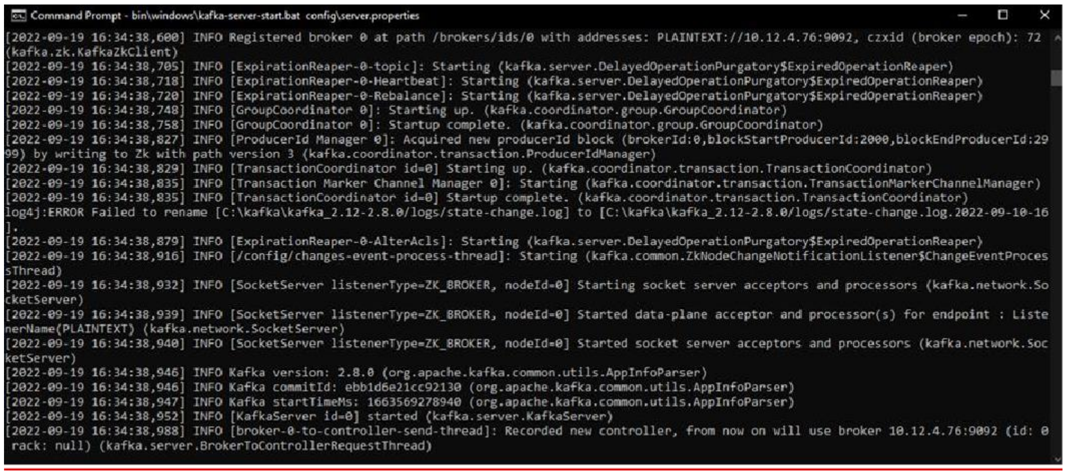
What you should see after running zookeeper:



Command Prompt - kafka\bin\windows\zookeeper-server-start.bat kafka\config\zookeeper.properties

```
[2022-09-20 19:38:28,912] INFO Server environment:os.name=Windows 10 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,913] INFO Server environment:os.arch=amd64 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,913] INFO Server environment:os.version=10.0 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,913] INFO Server environment:user.name=nabee (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,914] INFO Server environment:user.home=C:\Users\nabee (org.apache.zookeeper.server.ZooKeeperServer)

[2022 09 20 19:38:28,914] INFO Server environment:user.dir=C:\ (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,914] INFO Server environment:os.memory.free=496MB (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,915] INFO Server environment:os.memory.max=512MB (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,915] INFO Server environment:os.memory.total=512MB (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,917] INFO minSessionTimeout set to 6000 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,919] INFO maxSessionTimeout set to 60000 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,921] INFO Created server with tickTime 3000 minSessionTimeout 6000 maxSessionTimeout 60000 datadir
\tmp\zookeeper\version-2 snapdir \tmp\zookeeper\version-2 (org.apache.zookeeper.server.ZooKeeperServer)
[2022-09-20 19:38:28,931] INFO Using org.apache.zookeeper.server.NIOServerCnxnFactory as server connection factory (org.
apache.zookeeper.server.ServerCnxnFactory)
[2022-09-20 19:38:28,934] INFO Configuring NIO connection handler with 10s sessionless connection timeout, 2 selector th
read(s), 16 worker threads, and 64 kB direct buffers. (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2022-09-20 19:38:28,937] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
[2022-09-20 19:38:28,952] INFO zookeeper.snapshotSizeFactor = 0.33 (org.apache.zookeeper.server.ZKDatabase)
[2022-09-20 19:38:28,957] INFO Snapshotting: 0x0 to \tmp\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.per
sistence.FileTxnSnapLog)
[2022-09-20 19:38:28,960] INFO Snapshotting: 0x0 to \tmp\zookeeper\version-2\snapshot.0 (org.apache.zookeeper.server.per
sistence.FileTxnSnapLog)
[2022-09-20 19:38:28,971] INFO PrepRequestProcessor (sid:0) started, reconfigEnabled=false (org.apache.zookeeper.server.
PrepRequestProcessor)
[2022-09-20 19:38:28,975] INFO Using checkIntervalMs=60000 maxPerMinute=10000 (org.apache.zookeeper.server.ContainerMana
ger)
[2022-09-20 19:39:16,414] INFO Creating new log file: log.1 (org.apache.zookeeper.server.persistence.FileTxnLog)
```

What you should see after running kafka:



Command Prompt - bin\windows\kafka-server-start.bat config\server.properties

```
[2022-09-19 16:34:38,600] INFO Registered broker 0 at path /brokers/ids/0 with addresses: PLAINTEXT://10.12.4.76:9092, czxid (broker epoch): 72
(kafka.zk.KafkaZkClient)
[2022-09-19 16:34:38,705] INFO [ExpirationReaper-0-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-09-19 16:34:38,718] INFO [ExpirationReaper-0-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-09-19 16:34:38,720] INFO [ExpirationReaper-0-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-09-19 16:34:38,748] INFO [GroupCoordinator 0]: Starting up. (kafka.coordinator.group.GroupCoordinator)
[2022-09-19 16:34:38,758] INFO [GroupCoordinator 0]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
[2022-09-19 16:34:38,827] INFO [ProducerId Manager 0]: Acquired new producerId block (brokerId:0,blockStartProducerId:2000,blockEndProducerId:29
99) by writing to Zk with path version 3 (kafka.coordinator.transaction.ProducerIdManager)
[2022-09-19 16:34:38,829] INFO [TransactionCoordinator id=0] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
[2022-09-19 16:34:38,835] INFO [Transaction Marker Channel Manager 0]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2022-09-19 16:34:38,835] INFO [TransactionCoordinator id=0] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
log4j:ERROR Failed to rename [C:\kafka\kafka_2.12-2.8.0/logs/state-change.log] to [C:\kafka\kafka_2.12-2.8.0/logs/state-change.log.2022-09-10-16
].
[2022-09-19 16:34:38,879] INFO [ExpirationReaper-0-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
[2022-09-19 16:34:38,916] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProces
sThread)
[2022-09-19 16:34:38,932] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Starting socket server acceptors and processors (kafka.network.So
cketServer)
[2022-09-19 16:34:38,939] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started data-plane acceptor and processor(s) for endpoint : Liste
nerName(PLAINTEXT) (kafka.network.SocketServer)
[2022-09-19 16:34:38,940] INFO [SocketServer listenerType=ZK_BROKER, nodeId=0] Started socket server acceptors and processors (kafka.network.Soc
ketServer)
[2022-09-19 16:34:38,946] INFO Kafka version: 2.8.0 (org.apache.kafka.common.utils.AppInfoParser)
[2022-09-19 16:34:38,946] INFO Kafka commitId: ebb1d6e21cc92130 (org.apache.kafka.common.utils.AppInfoParser)
[2022-09-19 16:34:38,947] INFO Kafka startTimeMs: 1663569278940 (org.apache.kafka.common.utils.AppInfoParser)
[2022-09-19 16:34:38,952] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
[2022-09-19 16:34:38,988] INFO [broker-0-to-controller-send-thread]: Recorded new controller, from now on will use broker 10.12.4.76:9092 (id: 0
 rack: null) (kafka.server.BrokerToControllerRequestThread)
```

*You can see on the last line “**from now on will use broker on localhost:9092**”*, so we have Kafka up and running.

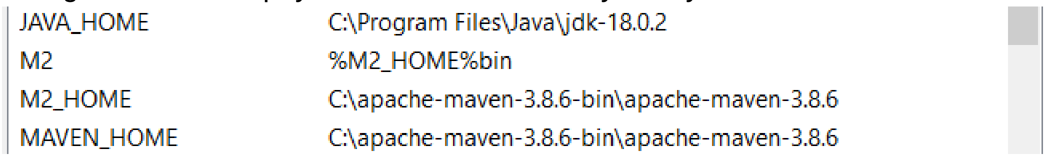
In this project, Inventory microservice acts as the kafka producer and connects to the kafka broker that is running on localhost:9092, and Procurement microservice acts as the kafka consumer. Also, the Sales microservice acts as a producer and connects to the kafka broker to send the SaleEvent topic to the BI microservice that is the processor that consumes the topic.

4.1.2. Building And Running The Application

Note: Please check that you have Apache Maven installed on your computer and do the following if you didn't do it before in order to enable maven, and in order for the next options to run successfully:

1. Go to Environment Variables on your computer -> Under Advanced -> Click Environment Variables -> Under System Variables -> Click New -> **Variable name:** JAVA_HOME & **Variable value:** the path of your JDK -> Click Ok
2. Click New -> **Variable name** M2_HOME & **Variable value:** the path of the installed maven file on your computer -> Click Ok
3. Click New -> **Variable name:** M2 & **Variable value:** %M2_HOME%\bin -> Click Ok
4. Click on Path -> Edit -> New -> **Variable value:** %M2_HOME%\bin
5. Click on Path -> Edit -> New -> **Variable value:** the path of your JDK
6. Click Ok -> Click Ok

Doing the above step, you should have this in your System Variables ->



JAVA_HOME	C:\Program Files\Java\jdk-18.0.2
M2	%M2_HOME%bin
M2_HOME	C:\apache-maven-3.8.6-bin\apache-maven-3.8.6
MAVEN_HOME	C:\apache-maven-3.8.6-bin\apache-maven-3.8.6

Order of running the microservices –

1. **Procurement Microservice**
2. **Inventory Microservice**
3. **Sales Microservice**
4. **BusinessIntelligence Microservice**

Running the application through IDE (i.e IntelliJ) –

1. Please note that the application runs on JDK1.8
2. Please note that Apache Maven is installed on your computer, if not, please follow the above instructions.
3. Please note that Apache Kafka should be running in order to run the application.
4. Open the project in the IDE: File -> Open -> Select the project that is MercuryCyclistsCSCI318 -> Ok
5. Select **ProcurementService** -> Go to its main -> Run
6. Select **InventoryService** -> Go to its main -> Run
7. Select **SalesService** -> Go to its main -> Run
8. Select **BusinessIntelligenceService** -> Go to its main -> Run
9. (Optional) You can go to Services -> Right click as shown below -> Run
10. All 4 microservices are successfully running on their own different ports. You can see them interacting with each other and you can do REST API requests.

Running the application directly through CMD using maven –

1. Please note that the application runs on JDK1.8
2. Please note that Apache Maven is installed on your computer, if not, please follow the above instructions.

3. Please note that Apache Kafka should be running in order to run the application.
4. Open a new CMD (Windows Command Prompt)
5. Navigate to the folder root of **ProcurementService** in the program folder MercuryCyclistsCSCI318
6. Type mvnw spring-boot:run
7. Open a new CMD (Windows Command Prompt)
8. Navigate to the folder root of **InventoryService** in the program folder MercuryCyclistsCSCI318
9. Type mvnw spring-boot:run
10. Open a new CMD (Windows Command Prompt)
11. Navigate to the folder root of **SalesService** in the program folder MercuryCyclistsCSCI318
12. Type mvnw spring-boot:run
13. Open a new CMD (Windows Command Prompt)
14. Navigate to the folder root of **BusinessIntelligenceService** in the program folder MercuryCyclistsCSCI318
15. Type mvnw spring-boot:run
16. Application is successfully running, now you can see how the microservices are interacting with each other, and you can also do REST API requests.

Building the JAR file (2 ways) –

Way 1: On CMD (Windows Command Prompt):

1. Open CMD (Windows Command Prompt)
2. Navigate to the folder root of the project -> cd C:\Users\nabee\Downloads\MercuryCyclistsCSCI318 (in our case)
3. Type mvnw package
4. JAR file is officially made in target directory

Way 2: On IDE (IntelliJ):

1. Open IDE (IntelliJ)
2. File -> Open -> Look for the project folder (MercuryCyclistsCSCI318 in our case) -> Open
3. View -> Tool Windows -> Maven
4. Expand the project -> Expand Lifecycle -> double-click on package
5. Maven will compile the package
6. JAR file is officially made in target directory

Running the JAR file on CMD –

1. Open a new CMD (Windows Command Prompt)
2. Navigate to the folder root of the project -> cd C:\Users\nabee\Downloads\318 (in our case)
3. Type mvnw package
4. Navigate to folder root of target in the project -> cd C:\Users\nabee\Downloads\MercuryCyclistsCSCI318\target
5. Type java -jar MercuryCyclistsCSCI318-0.0.1-SNAPSHOT.jar
6. Application is successfully running, now you can do REST API requests.

Running the application directly through CMD using maven –

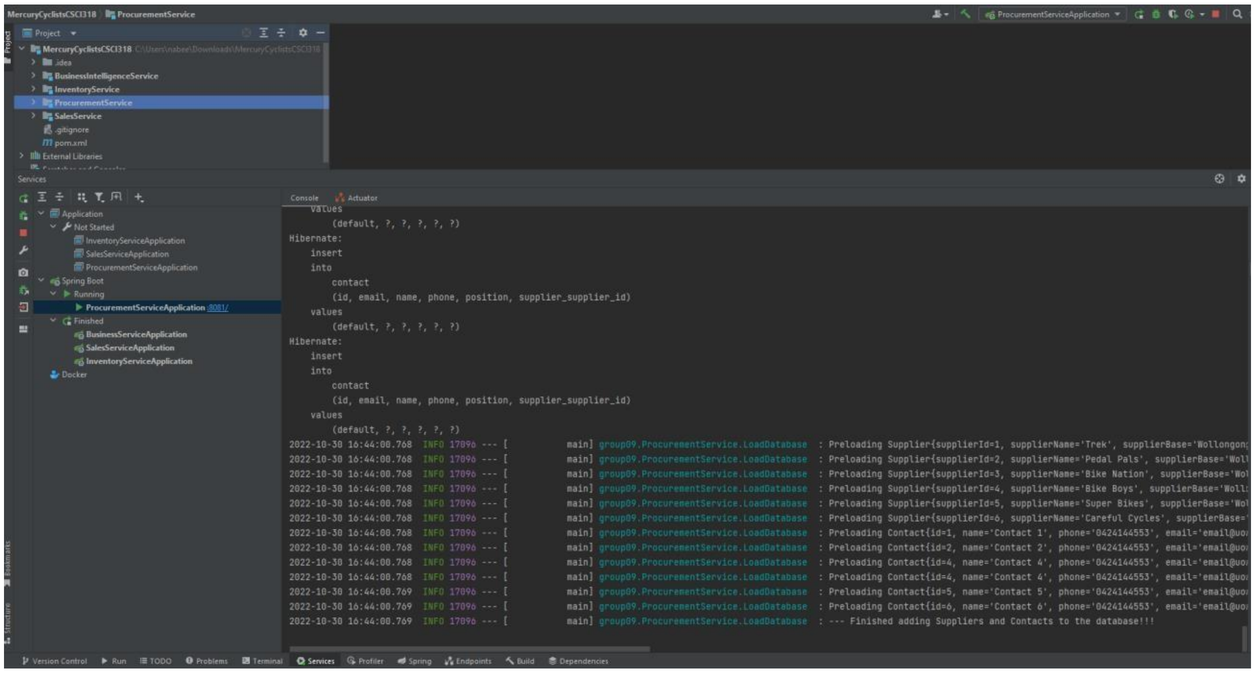
1. Open a new CMD (Windows Command Prompt)

2. **2.** Navigate to the folder root of the project -> cd C:\Users\nabee\Downloads\MercuryCyclistsCSCI318 (in our case)
3. **3.** Type mvnw spring-boot:run
4. **4.** Application is successfully running, now you can do REST API requests.

Screenshots showing the program running successfully using the exact steps above –

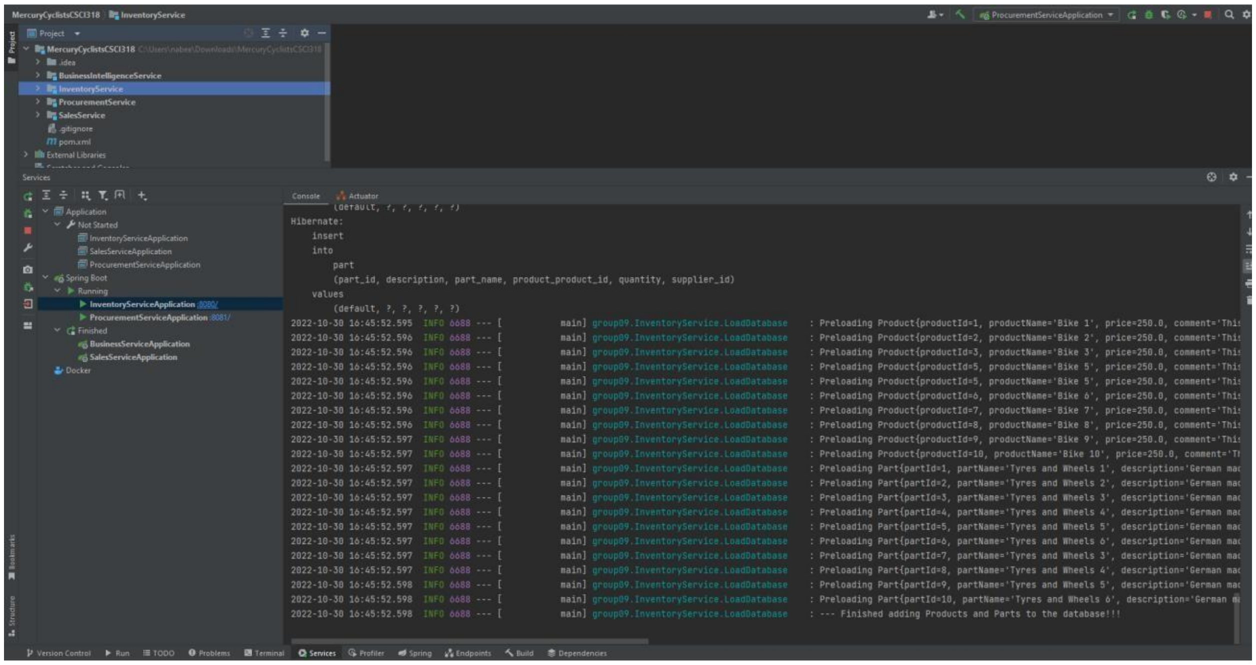
Running the application through IDE (i.e IntelliJ) screenshots –

1. **Procurement Microservice**



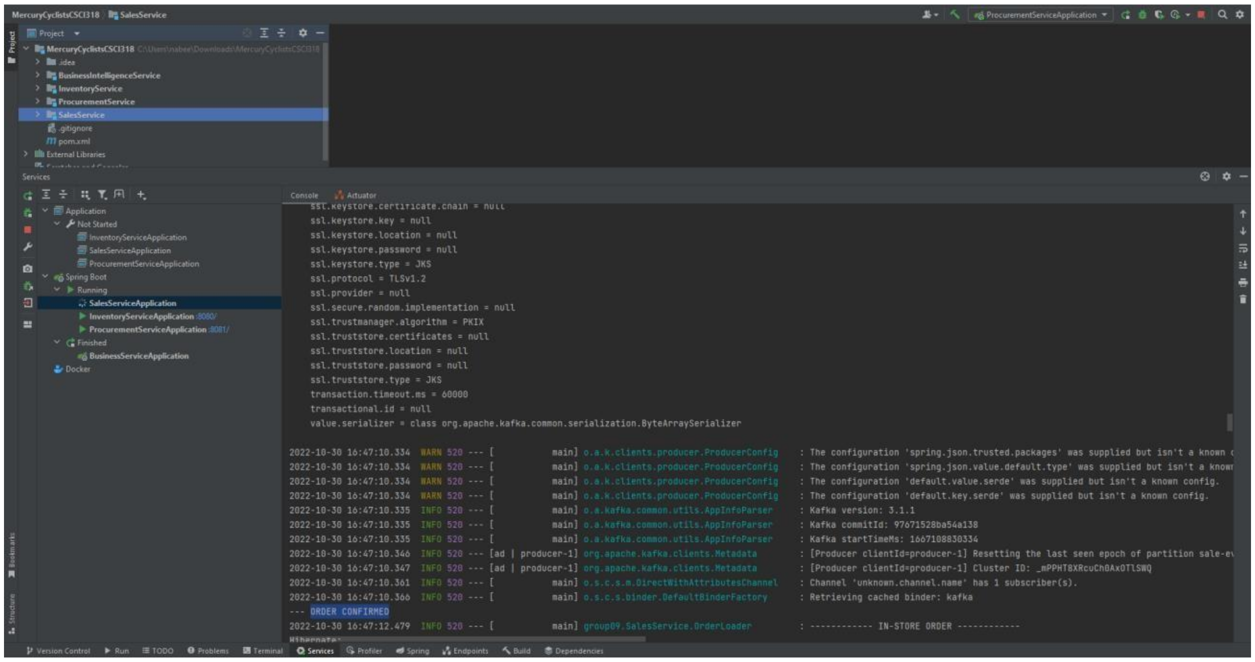
```
    values
        (default, ?, ?, ?, ?, ?)
Hibernate: 
    insert 
    into
        contact
        (id, email, name, phone, position, supplier_supplier_id) 
    values
        (default, ?, ?, ?, ?, ?)
Hibernate: 
    insert 
    into
        contact
        (id, email, name, phone, position, supplier_supplier_id) 
    values
        (default, ?, ?, ?, ?, ?)
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=1, supplierName='Trek', supplierBase='Wollongon
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=2, supplierName='Pedal Pals', supplierBase='Woll
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=3, supplierName='Bike Nation', supplierBase='Wol
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=4, supplierName='Bike Boys', supplierBase='Woll
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=5, supplierName='Super Bikes', supplierBase='Wol
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=6, supplierName='Careful Cycles', supplierBase='
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=1, name='Contact 1', phone='0424144553', email='email@uo
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=2, name='Contact 2', phone='0424144553', email='email@uo
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=4, name='Contact 4', phone='0424144553', email='email@uo
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=4, name='Contact 4', phone='0424144553', email='email@uo
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=5, name='Contact 5', phone='0424144553', email='email@uo
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=6, name='Contact 6', phone='0424144553', email='email@uo
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : --- Finished adding Suppliers and Contacts to the database!!!
```

2. **Inventory Microservice**



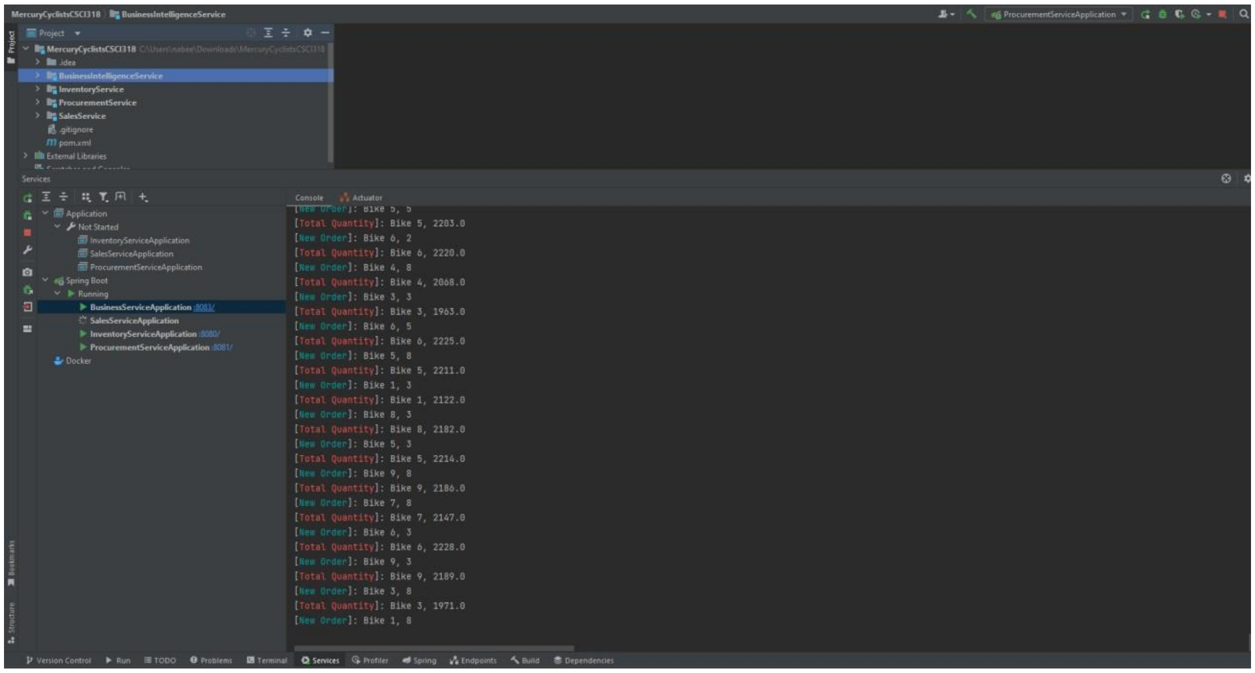
```
        (default, ?, ?, ?, ?, ?)
Hibernate: 
    insert 
    into
        part
        (part_id, description, part_name, product_product_id, quantity, supplier_id) 
    values
        (default, ?, ?, ?, ?, ?)
2022-10-30 16:45:52.595  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=1, productName='Bike 1', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=2, productName='Bike 2', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=3, productName='Bike 3', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=5, productName='Bike 5', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=5, productName='Bike 5', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=6, productName='Bike 6', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=7, productName='Bike 7', price=250.0, comment='Thi
2022-10-30 16:45:52.596  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=8, productName='Bike 8', price=250.0, comment='Thi
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=9, productName='Bike 9', price=250.0, comment='Thi
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Product{productId=10, productName='Bike 10', price=250.0, comment='T
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=1, partName='Tyres and Wheels 1', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=2, partName='Tyres and Wheels 2', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=3, partName='Tyres and Wheels 3', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=4, partName='Tyres and Wheels 4', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=5, partName='Tyres and Wheels 5', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=6, partName='Tyres and Wheels 6', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=7, partName='Tyres and Wheels 3', description='German ma
2022-10-30 16:45:52.597  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=8, partName='Tyres and Wheels 4', description='German ma
2022-10-30 16:45:52.598  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=9, partName='Tyres and Wheels 5', description='German ma
2022-10-30 16:45:52.598  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : Preloading Part{partId=10, partName='Tyres and Wheels 6', description='German 
2022-10-30 16:45:52.598  INFO 6688 --- [           main] group09.InventoryService.LoadDatabase    : --- Finished adding Products and Parts to the database!!!
```

3. Sales Microservice



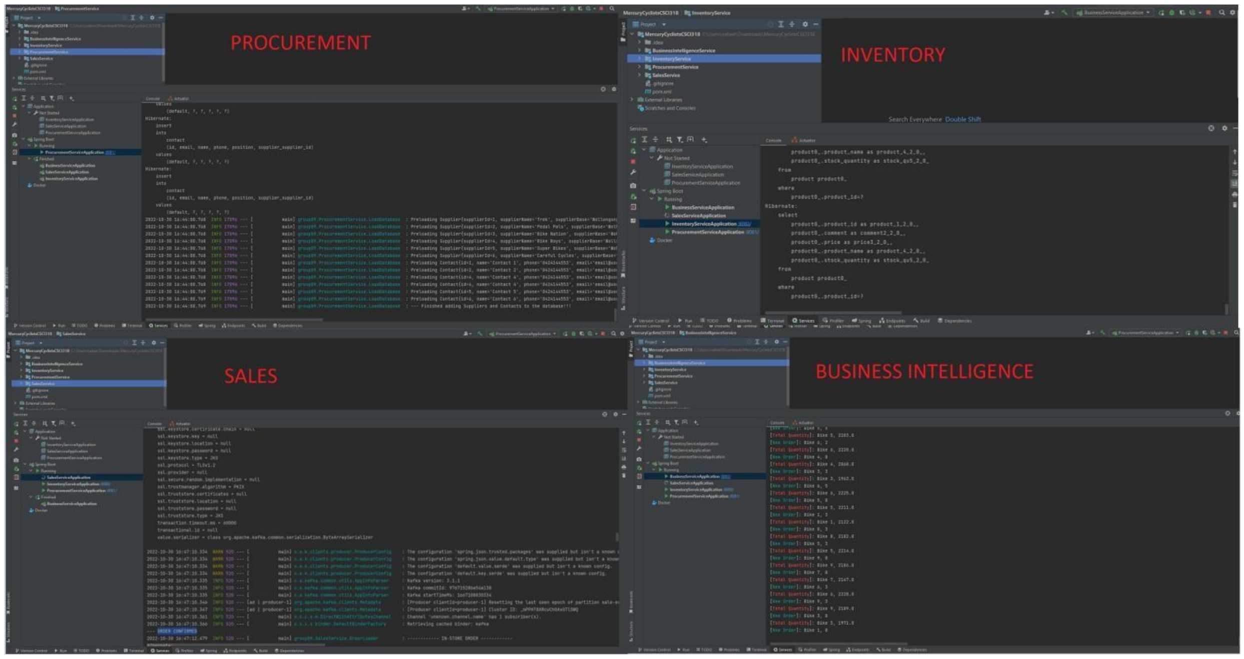
```
    ssl.keystore.certificate.chain = null
    ssl.keystore.key = null
    ssl.keystore.location = null
    ssl.keystore.password = null
    ssl.keystore.type = JKS
    ssl.protocol = TLSv1.2
    ssl.provider = null
    ssl.secure.random.implementation = null
    ssl.trustmanager.algorithm = PKIX
    ssl.truststore.certificates = null
    ssl.truststore.location = null
    ssl.truststore.password = null
    ssl.truststore.type = JKS
    transaction.timeout.ms = 60000
    transactional.id = null
    value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer

2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'spring.json.trusted.packages' was supplied but isn't a known c
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'spring.json.value.default.type' was supplied but isn't a known
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'default.value.serde' was supplied but isn't a known config.
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'default.key.serde' was supplied but isn't a known config.
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka version: 3.1.1
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId: 97671528ba54a138
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka startTimeMs: 1667108830334
2022-10-30 16:47:10.346  INFO 520 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : [Producer clientId=producer-1] Resetting the last seen epoch of partition sale-ev
2022-10-30 16:47:10.347  INFO 520 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : [Producer clientId=producer-1] Cluster ID: _mPPHT8XRcuCh0AxOTlSWQ
2022-10-30 16:47:10.361  INFO 520 --- [           main] o.s.c.s.m.DirectWithAttributesChannel    : Channel 'unknown.channel.name' has 1 subscriber(s).
2022-10-30 16:47:10.366  INFO 520 --- [           main] o.s.c.s.binder.DefaultBinderFactory      : Retrieving cached binder: kafka
--- ORDER CONFIRMED
2022-10-30 16:47:12.479  INFO 520 --- [           main] group09.SalesService.OrderLoader         : ------------ IN-STORE ORDER ------------
Hibernate:
```

4. BusinessIntelligence Microservice



```
[New Order]: Bike 5, 5
[Total Quantity]: Bike 5, 2203.0
[New Order]: Bike 6, 2
[Total Quantity]: Bike 6, 2220.0
[New Order]: Bike 4, 8
[Total Quantity]: Bike 4, 2068.0
[New Order]: Bike 3, 3
[Total Quantity]: Bike 3, 1963.0
[New Order]: Bike 6, 5
[Total Quantity]: Bike 6, 2225.0
[New Order]: Bike 5, 8
[Total Quantity]: Bike 5, 2211.0
[New Order]: Bike 1, 3
[Total Quantity]: Bike 1, 2122.0
[New Order]: Bike 8, 3
[Total Quantity]: Bike 8, 2182.0
[New Order]: Bike 5, 3
[Total Quantity]: Bike 5, 2214.0
[New Order]: Bike 9, 8
[Total Quantity]: Bike 9, 2186.0
[New Order]: Bike 7, 8
[Total Quantity]: Bike 7, 2147.0
[New Order]: Bike 6, 3
[Total Quantity]: Bike 6, 2228.0
[New Order]: Bike 9, 3
[Total Quantity]: Bike 9, 2189.0
[New Order]: Bike 3, 8
[Total Quantity]: Bike 3, 1971.0
[New Order]: Bike 1, 8
```

Screenshot of all 4 microservices running and interacting with each other:



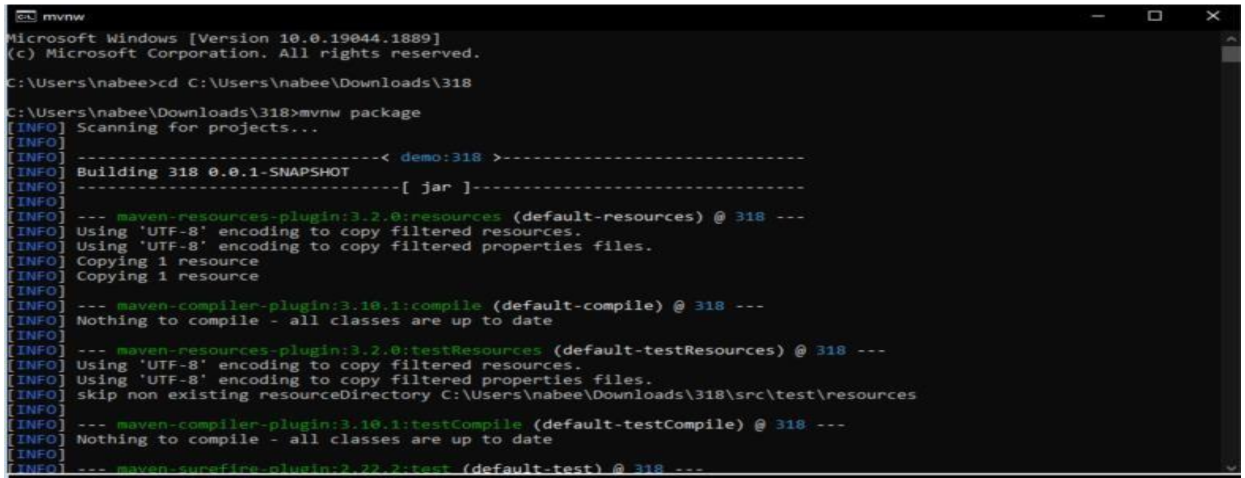
```
    values
        (default, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        contact
        (id, email, name, phone, position, supplier_supplier_id)
    values
        (default, ?, ?, ?, ?, ?)
Hibernate:
    insert
    into
        contact
        (id, email, name, phone, position, supplier_supplier_id)
    values
        (default, ?, ?, ?, ?, ?)
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=1, supplierName='Trek', supplierBase='Wollongon
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=2, supplierName='Pedal Pals', supplierBase='Wol
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=3, supplierName='Bike Nation', supplierBase='Wo
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=4, supplierName='Bike Boys', supplierBase='Woll
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=5, supplierName='Super Bikes', supplierBase='Wo
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Supplier{supplierId=6, supplierName='Careful Cycles', supplierBase='
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=1, name='Contact 1', phone='0424144553', email='email@ex
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=2, name='Contact 2', phone='0424144553', email='email@ex
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=4, name='Contact 4', phone='0424144553', email='email@ex
2022-10-30 16:44:00.768  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=4, name='Contact 4', phone='0424144553', email='email@ex
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=5, name='Contact 5', phone='0424144553', email='email@ex
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : Preloading Contact{id=6, name='Contact 6', phone='0424144553', email='email@ex
2022-10-30 16:44:00.769  INFO 17096 --- [           main] group09.ProcurementService.LoadDatabase  : --- Finished adding Suppliers and Contacts to the database!!!
```

```
            product0_.product_name as product_4_2_0_,
            product0_.stock_quantity as stock_qu5_2_0_
    from
            product product0_
    where
            product0_.product_id=?
Hibernate:
    select
            product0_.product_id as product_1_2_0_,
            product0_.comment as comment2_2_0_,
            product0_.price as price3_2_0_,
            product0_.product_name as product_4_2_0_,
            product0_.stock_quantity as stock_qu5_2_0_
    from
            product product0_
    where
            product0_.product_id=?
```

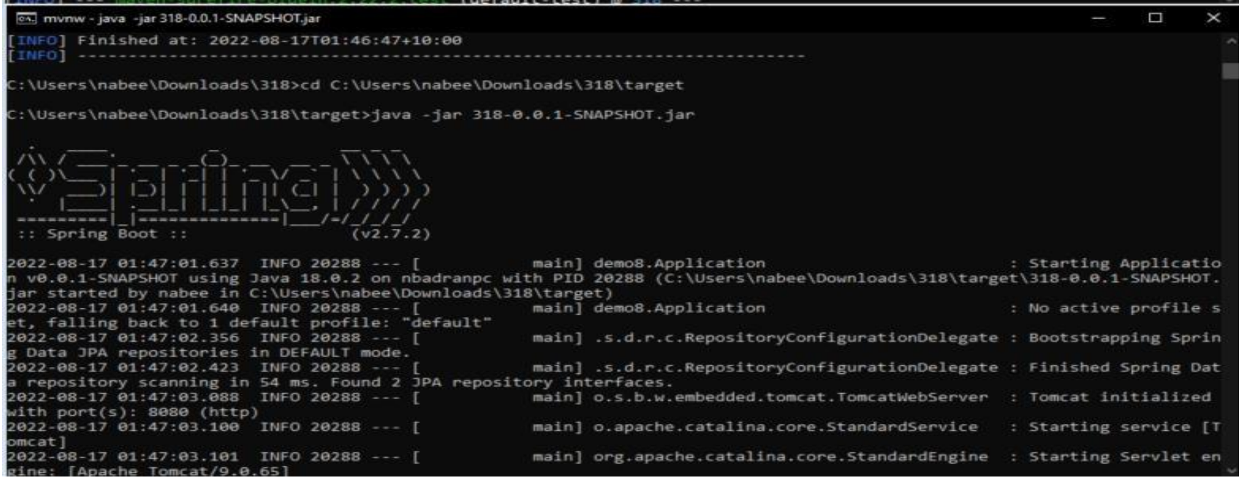
```
    ssl.keystore.certificate.chain = null
    ssl.keystore.key = null
    ssl.keystore.location = null
    ssl.keystore.password = null
    ssl.keystore.type = JKS
    ssl.protocol = TLSv1.2
    ssl.provider = null
    ssl.secure.random.implementation = null
    ssl.trustmanager.algorithm = PKIX
    ssl.truststore.certificates = null
    ssl.truststore.location = null
    ssl.truststore.password = null
    ssl.truststore.type = JKS
    transaction.timeout.ms = 60000
    transactional.id = null
    value.serializer = class org.apache.kafka.common.serialization.ByteArraySerializer

2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'spring.json.trusted.packages' was supplied but isn't a known c
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'spring.json.value.default.type' was supplied but isn't a known
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'default.value.serde' was supplied but isn't a known config.
2022-10-30 16:47:10.334  WARN 520 --- [           main] o.a.k.clients.producer.ProducerConfig    : The configuration 'default.key.serde' was supplied but isn't a known config.
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka version: 3.1.1
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka commitId: 97671528ba54a138
2022-10-30 16:47:10.335  INFO 520 --- [           main] o.a.kafka.common.utils.AppInfoParser     : Kafka startTimeMs: 1667108830334
2022-10-30 16:47:10.346  INFO 520 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : [Producer clientId=producer-1] Resetting the last seen epoch of partition sale-e
2022-10-30 16:47:10.347  INFO 520 --- [ad | producer-1] org.apache.kafka.clients.Metadata        : [Producer clientId=producer-1] Cluster ID: _mPPH7BXRcuCh6Ax0TL0WQ
2022-10-30 16:47:10.361  INFO 520 --- [           main] o.s.i.c.e.DirectWithAttributesChannel    : Channel 'unknown.channel.name' has 1 subscriber(s).
2022-10-30 16:47:10.366  INFO 520 --- [           main] o.s.c.s.binder.DefaultBinderFactory      : Retrieving cached binder: kafka
--- ORDER CONFIRMED
2022-10-30 16:47:12.479  INFO 520 --- [           main] group09.SalesService.OrderLoader         : ------------ IN-STORE ORDER ------------
```

```
[New Order]: Bike 5, 9
[Total Quantity]: Bike 5, 2203.0
[New Order]: Bike 6, 2
[Total Quantity]: Bike 6, 2220.0
[New Order]: Bike 4, 8
[Total Quantity]: Bike 4, 2868.0
[New Order]: Bike 3, 3
[Total Quantity]: Bike 3, 1963.0
[New Order]: Bike 6, 5
[Total Quantity]: Bike 6, 2225.0
[New Order]: Bike 5, 8
[Total Quantity]: Bike 5, 2211.0
[New Order]: Bike 1, 3
[Total Quantity]: Bike 1, 2122.0
[New Order]: Bike 8, 3
[Total Quantity]: Bike 8, 2182.0
[New Order]: Bike 5, 3
[Total Quantity]: Bike 5, 2214.0
[New Order]: Bike 9, 8
[Total Quantity]: Bike 9, 2186.0
[New Order]: Bike 7, 8
[Total Quantity]: Bike 7, 2147.0
[New Order]: Bike 6, 3
[Total Quantity]: Bike 6, 2228.0
[New Order]: Bike 9, 3
[Total Quantity]: Bike 9, 2189.0
[New Order]: Bike 3, 8
[Total Quantity]: Bike 3, 1971.0
[New Order]: Bike 1, 8
```

Running the JAR file on CMD screenshots –

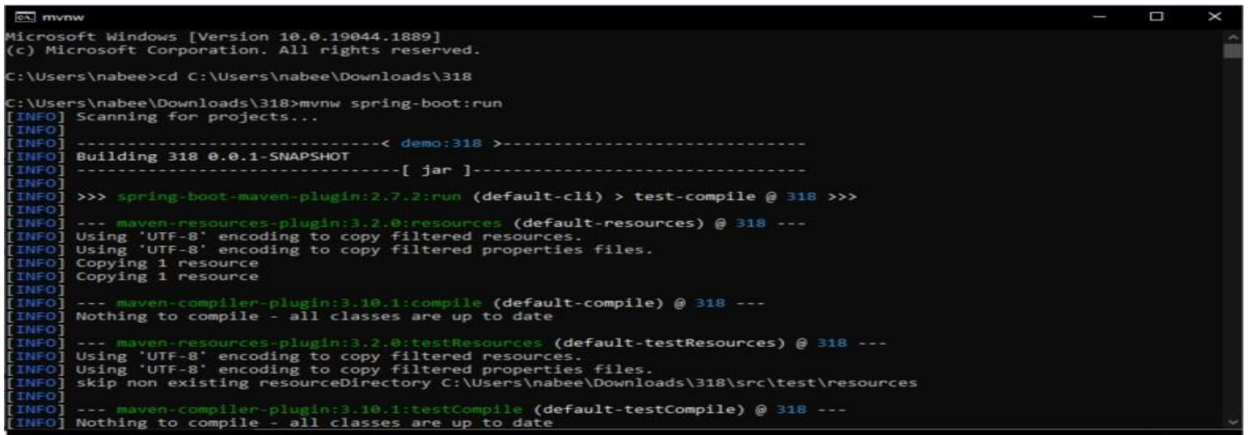


```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nabee>cd C:\Users\nabee\Downloads\318

C:\Users\nabee\Downloads\318>mvnw package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------< demo:318 >------------------------------
[INFO] Building 318 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ 318 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:compile (default-compile) @ 318 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ 318 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\Users\nabee\Downloads\318\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:testCompile (default-testCompile) @ 318 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ 318 ---
```

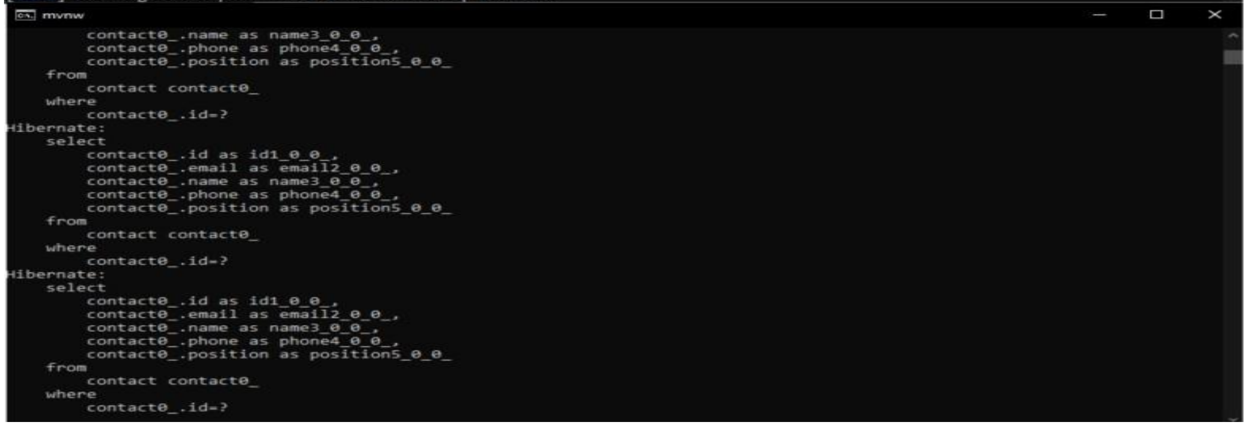


```
[INFO] Finished at: 2022-08-17T01:46:47+10:00
[INFO] ------------------------------------------------------------------------

C:\Users\nabee\Downloads\318>cd C:\Users\nabee\Downloads\318\target

C:\Users\nabee\Downloads\318\target>java -jar 318-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.2)

2022-08-17 01:47:01.637  INFO 20288 --- [           main] demo8.Application                        : Starting Applicatio
n v0.0.1-SNAPSHOT using Java 18.0.2 on nbadranpc with PID 20288 (C:\Users\nabee\Downloads\318\target\318-0.0.1-SNAPSHOT.
jar started by nabee in C:\Users\nabee\Downloads\318\target)
2022-08-17 01:47:01.640  INFO 20288 --- [           main] demo8.Application                        : No active profile s
et, falling back to 1 default profile: "default"
2022-08-17 01:47:02.356  INFO 20288 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Sprin
g Data JPA repositories in DEFAULT mode.
2022-08-17 01:47:02.423  INFO 20288 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Dat
a repository scanning in 54 ms. Found 2 JPA repository interfaces.
2022-08-17 01:47:03.088  INFO 20288 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized
with port(s): 8080 (http)
2022-08-17 01:47:03.100  INFO 20288 --- [           main] o.apache.catalina.core.StandardService   : Starting service [T
omcat]
2022-08-17 01:47:03.101  INFO 20288 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet en
gine: [Apache Tomcat/9.0.65]
```

Running the application directly through CMD using maven screenshots –



```
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\nabee>cd C:\Users\nabee\Downloads\318

C:\Users\nabee\Downloads\318>mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------------< demo:318 >------------------------------
[INFO] Building 318 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.7.2:run (default-cli) > test-compile @ 318 >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ 318 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:compile (default-compile) @ 318 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ 318 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\Users\nabee\Downloads\318\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.10.1:testCompile (default-testCompile) @ 318 ---
[INFO] Nothing to compile - all classes are up to date
```



```
        contact0_.name as name3_0_0_,
        contact0_.phone as phone4_0_0_,
        contact0_.position as position5_0_0_
    from
        contact contact0_
    where
        contact0_.id=?
Hibernate:
    select
        contact0_.id as id1_0_0_,
        contact0_.email as email2_0_0_,
        contact0_.name as name3_0_0_,
        contact0_.phone as phone4_0_0_,
        contact0_.position as position5_0_0_
    from
        contact contact0_
    where
        contact0_.id=?
Hibernate:
    select
        contact0_.id as id1_0_0_,
        contact0_.email as email2_0_0_,
        contact0_.name as name3_0_0_,
        contact0_.phone as phone4_0_0_,
        contact0_.position as position5_0_0_
    from
        contact contact0_
    where
        contact0_.id=?
```

Some Examples of Input Requests Through CMD or Terminal

Note: Please refer to under '*Use Cases Of The Application With Examples Of Input And Output (i.e., REST requests)*' title for screenshot results proof of the input requests that are in the 10 project use cases

GET Requests –

Supplier

GET all suppliers: curl -v localhost:8081/suppliers

GET supplier by supplier id: curl -v localhost:8081/suppliers/1

Contact

GET all contacts: curl -v localhost:8081/supplierContacts

GET contact by contact id: curl -v localhost:8081/supplierContacts/1

Product

GET product by productId: curl -v localhost:8080/product/1

GET all products: curl -v localhost:8080/product

GET parts by productId: curl -v localhost:8080/product/1/part

Validate a product by productId: curl -v localhost:8080/product/productsValidate/1

Validate products parts by productId: curl -v localhost:8080/product/validateProductParts/1

GET product name by productId: curl -v localhost:8080/product/getProductName/1

GET product price by productId: curl -v localhost:8080/product/getProductPrice/1

Procurement request for kafka -> productId/quantity: curl -v localhost:8080/product/procurementRequest/1/50

Sale

GET all sales: curl -v localhost:8082/sale

GET sale by saleId: curl -v localhost:8082/sale/1

Look up products by saleId: curl -v localhost:8082/sale/lookUpProductBySale/1

Validate a sale by saleId: curl -v localhost:8082/sale/saleValidate/1

BusinessIntelligence

GET total sales value by productId: curl -v localhost:8083/BIservice/totalSalesValue/1

GET all productNames: curl -v localhost:8083/BIservice/productNames

PUT Requests –

Supplier

Update supplierName: curl -X PUT localhost:8081/suppliers/1?supplierName=NewSupplierName

Contact

Update contact: curl -X PUT localhost:8081/supplierContacts/1 -H "Content-Type:application/json" -d "{\"name\":\"NewName\", \"phone\":\"newPhone\", \"email\":\"NewEmail\", \"position\":\"NewPosition\"}"

POST Requests –

Supplier

Create supplier: curl -H "Content-Type: application/json" -X POST -d "{\"supplierName\":\"NEW SUPPLIER\", \"supplierBase\":\"NEW SUPPLIER BASE\"}" localhost:8081/suppliers

Contact

Create contact: curl -H "Content-Type:application/json" -X POST -d "{\"name\":\"NEW NAME\", \"phone\":\"NEW PHONE\", \"email\":\"NEW EMAIL\", \"position\":\"NEW POSITION\"}" localhost:8081/supplierContacts

DELETE Requests –

Supplier

Delete supplier: curl -X DELETE localhost:8081/suppliers/1

Contact

Delete contact: curl -X DELETE localhost:8081/supplierContacts/1

Product

Delete product: curl -X DELETE localhost:8080/product/1

Sale

Delete a sale: curl -X DELETE localhost:8082/sale/1

References

[1]. Youtu.be. 2022. *How to Install Apache Kafka on Windows.* [online] Available at: <https://youtu.be/EUzH9khPYgs> [Accessed 20 September 2022].

[2] Goavega.com. 2022. *Install Apache Kafka on Windows*. [online] Available at: <https://www.goavega.com/blog/install-apache-kafka-on-windows/> [Accessed 20 September 2022].

[3] kafka-console-consumer.sh, z. and Singh, G., 2022. *zookeeper is not a recognized option when executing kafka-console-consumer.sh*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/53428903/zookeeper-is-not-a-recognized-option-when-executing-kafka-console-consumer-sh> [Accessed 20 September 2022].

[4] Apache Kafka. 2022. *Apache Kafka*. [online] Available at: <https://kafka.apache.org/quickstart> [Accessed 20 September 2022].

[5] kafka-console-consumer.sh, z. and Singh, G., 2022. *zookeeper is not a recognized option when executing kafka-console-consumer.sh*. [online] Stack Overflow. Available at: <https://stackoverflow.com/a/58371707/10294775> [Accessed 20 September 2022].

[6] Apache Kafka. 2022. *Apache Kafka*. [online] Available at: <https://kafka.apache.org/#:~:text=Apache%20Kafka%20is%20an%20open,%2C%20and%20mission%2Dcritical%20applications> [Accessed 20 September 2022].