CSCI318 - Task 3

Mark Cai Yee Lee 6143611

CSCI318 - Task 3

This project implements a microservice which enables ordering of products via REST and Apache Kafka.

Pre-requisites

Framework

- Spring Boot
- Spring Cloud Streams

Dependencies

- Spring Web
- Spring Data JPA
- H2
- Cloud Stream
- Spring for Apache Kafka
- Spring for Apache Kafka Streams

Building

```
cd csci318-project/customer-service && mvn compile && mvn package
cd ../ordering-service && mvn compile && mvn package
cd ../product-service && mvn compile && mvn package
cd ../bi-service && mvn compile && mvn package
```

Running

First, Apache Kafka is required to run the application. In this case, the binary download from `https://kafka.apache.org/downloads` is used.

This version is used `https://archive.apache.org/dist/kafka/2.8.1/kafka_2.13-2.8.1.tgz`.

Next, extract the downloaded tar file and start Apache Kafka. In separate terminal emulators,

```
# Command One
cd kafka_2.13-2.8.0
./bin/zookeeper-server-start.sh config/zookeeper.properties

# In another terminal
cd kafka_2.13-2.8.0
/bin/kafka-server-start.sh config/server.properties
```

```
java -jar customer-service-0.0.1-SNAPSHOT.jar
java -jar product-service-0.0.1-SNAPSHOT.jar
java -jar ordering-service-0.0.1-SNAPSHOT.jar
java -jar bi-service-0.0.1-SNAPSHOT.jar
```

Note: - Customer Service runs on Port 8082 - Product Service runs on Port 8080 - Ordering Service runs on Port 8081 - BI Service runs on Port 8083 - Apache Kafka runs on Port 9092

Structure

customer-service

```
- api
  - ContactController.java
  - ContactExceptionController.java
  - CustomerController.java
  - CustomerExceptionController.java
- model
  - Contact.java
  - ContactNotFoundException.java
  - Customer.java
  - CustomerNotFoundException.java
- repository
  - ContactRepository.java
  - CustomerRepository.java
- service
  - ContactService.java
  - CustomerService.java
- CustomerServiceApplication.java
- CustomerRunner.java
```

product-service

```
- api
  - ProductController.java
  - ProductExceptionController.java
  - ProductDetailController.java
  - ProductDetailExceptionController.java
- model
  - Product.java
  - ProductDetail.java
  - ProductNotFoundException.java
  - ProductDetailNotFoundException.java
- repository
  - ProductDetailRepository.java
  - ProductRepository.java
- service
  - ProductDetailService.java
  - ProductService.java
- ProductServiceApplication.java
- ProductRunner.java
```

ordering-service

```
- api
  - OrderController.java
  - OrderExceptionController.java
- model
  - Order.java
  - OrderEvent.java
  - OrderNotFoundException.java
- repository
  - OrderEventRepository.java
```

```
  - OrderRepository.java
- service
  - OrderEventHandler.java
  - OrderService.java
- OrderingServiceApplication.java
- OrderRunner.java
```

bi-service

```
- api
   - CustomerOrderQueryController.java
- model
   - OrderEvent.java
- service
   - OrderInteractiveQuery.java
   - OrderStreamProcessing.java
- BIServiceApplication
```

Description of the project

Structure

The onlineordering application is made up of four separate subdomains. In this case, each subdomain represents a key problem of the application. In each subdomain is multiple bounded context and database for each business solution. Thus, the application is modular and can be developed in parallel with multiple teams without issues.

The application follow the N-Tier Architecture which is split into API Layer, Service Layer, Data Access Layer and Database. This is valid for each service which represents a subdomain.

The API Layer handles REST Requests and calls Service Functions in the Service Layer. The Service Layer also consists of Domain Models specific to the application. The Service Layer handle function calls and access the Data Access Layer which has repositories specific to the domains models. A repository will then access entries or save data into the database as required.

Distributed Bounded Context Integration

REST

Each application is separate and does not have a shared database. The onlineordering application utilises REST Requests via HTTP to connect applications and facilitate communication. This allows for a clear separation between each subdomain. This is true for `customer-service`, `product-service` and `ordering-service`.

Messaging

The bi-service uses messaging via Apache Kafka to communicate with the `ordering-service`. In the case for this project, the `ordering-service` is a producer and publishes to the `order-event-topic`. Next, the `bi-service` is a processor which consumes the topic `order-event-topic` and publishes to `quantity-summary-topic`. The types of data for each topic is presented in the table below.

Topic	Type
`order-event-topic`	`<String, OrderEvent>`
`quantity-summary-topic`	`<String, Double>`

BI Service

API Layer

This service only has one controller. This controller handles both interactive query REST requests.

- `findCustomerProductList()`
  - Returns a List with String types from `OrderInteractiveQuery.findCustomerByOrderProduct()`
  - Requires specifying the customerId in the http request.
- `getCustomerTotalOrderValue()`
  - Returns a long from `OrderInteractiveQuery.getCustomerTotalOrderValue()`
  - Requires specifying the customerId in the http request.

Service Layer

BI Service has one model, `OrderEvent`. This model stores the information for processing messages and interactive query.

This service does not have a data access layer as data is not stored in database but obtained by subscribing to the `order-event-topic`.

Order Service

API Layer

OrderController

The OrderController handles several REST Requests. The functions implemented are `addOrder`, `findCustomerByOrderId` and `findProductByOrderId`.

- `addOrder()`
  - Returns a Response Entity from `orderService.addOrder()`.
  - Requires a JSON body with the following keys
    - supplier
    - productName
    - quantity
- `findCustomerByOrderId()`
  - Returns a Response Entity from `orderService.findCustomerByOrderId()`.
- `findProductById()`
  - Returns a Response Entity from `orderService.findProductById()`.
  - Returns a Response String which consist of Customer Address, Customer Contact, Unit Price, Unit(s) Ordered if the Customer is valid and Product is in stock.
  - Returns a Response String which consist of Customer Address, Customer Contact and “Not Enough Stock” if the Customer is valid but there is no stock available.

Service Layer

The Order Domain has two models, Order and OrderEvent.

The OrderEvent stores the following variables: - productName - orderQuantity - stockQuantity - productPrice

OrderService

The `addOrder()` function requires a customerId and a JSON Object which makes up the Order Object.

This service uses RestTemplate to make REST Requests. The first REST Request checks if a customer with the customerId is present. The second checks if the product with the productName stored in the Order Object is in stock. This service publishes a OrderEvent if the customerId is valid and the product is in stock or has sufficient stock.

The `findCustomerByOrderId()` function takes a orderId and uses REST Requests to find the customer details associated with the order with the orderId. This function returns the response from `customer-service` from calling a GET Request with the customerId associated with the order with orderId.

The `findProductById()` function takes a orderId and uses REST Requests to find the details of the product in the order. This function calls GET and returns the response provided from `product-service` with the productName associated with the order with orderId.

OrderEventHandler

This service handles events published which consists of a OrderEvent. The `handle()` function accepts an OrderEvent Object. This service also uses RestTemplate which makes a PUT request to update the stock on the Product Application. In addition, this service sends a OrderEvent Object to the `order-event-topic` topic whenever a valid order is placed.

Data Access Layer

The data access layer for this application saves the previous order events.

Product Service

API Layer

ProductController

This Class has the following functionality:

- addProduct()
  - Returns product and accepts a JSON body which takes in the following keys:
    - productCategory
    - productName
    - price
    - stockQuantity
- findAllProduct()
  - Returns all products stored in the database.
- findProductById()
  - Returns the product with the “{id}” specified in the request.
- deleteProductById()
  - Deletes the product with the “{id}” specified in the request.
- updateStock()
  - Updates the stock of the product by name and accepts the JSON body with the key:
    - stockQuantity
- updateProductById()
  - Updates the product with the new product specified in the JSON body.
- addProductDetail()
  - Links the product with the product detail using two ids.

ProductDetailController

The class has the following functions:

- addProductDetail()
  - Returns the product detail and accepts a JSON body with the following keys:
    - description
    - comment
- findAllProductDetail()
  - Returns all product detail items stored in the database.
- findProductDetailById()
  - Returns the specified product detail with "{id}" specified in the request.
- updateProductDetailById()
  - Returns the new product detail. Updates the product detail with the parameters in the JSON body.
- deleteProductDetailById()
  - Deletes the product detail with the "{id}" from the database.

Service Layer

In the Product Domain there are two models, Product and Product Detail. The Services for these models respectively implements the functionality specified in the section above.

Data Access Layer

The repository interfaces extends the JpaRepository. The project uses the default functions. ProductRepository has a function `findByProductName()` which enables search for product by name.

Customer Service

This service handles requests that are in the Customer Domain.

API Layer

There are two classes in the API Layer, ContactController and CustomerController. ContactController accepts REST Requests with the path set to '/contact'. This class handles REST Requests and has the following functionality:

ContactController

- getAllContact()
  - Returns all the contacts stored in the database.
- getContactById()
  - Returns the contact stored in the database which matches the "{id}" specified in the request.
- addContact()
  - Returns the new contact after accepting a JSON body which takes these following keys:
    - name
    - phone
    - email
    - position
- deleteContactById()
  - Deletes the contact specified by "{id}" in the database.
- updateContactById()
  - Accepts a JSON body which is similar to addContact() and updates the contact in the database which matches the specified "{id}". The function returns the updated contact.

CustomerController

The CustomerController class accepts REST requests with the path '/customer' and has the following functionality:

- getAllCustomer()
  - Returns all the customers stored in the database.
- addCustomer()
  - Returns the new customer after accepting a JSON body which takes in the following keys:
    - companyName
    - address
    - country
- getCustomerById()
  - Returns the customer which matches the "{id}" specified.
- getCustomerByCompanyName()
  - Requires the path '/customer/company' and the search param 'name'.
  - Returns the customer which matches the search parameter.
- updateCustomer()
  - Accepts a JSON body similar to addCustomer() and updates the customer in the database which matches the "{id}" specified in the path.
- addCustomerContact()
  - Links an existing customer with a contact which exists in the database.
  - Requires specifying with '/customer/{id}/contact/{contactId}'
  - Returns the updated customer.

Service Layer

In the Customer Domain, two models created for this task, Customer and Contact.

A Customer model has the following variables:

- companyName
- address
- country
- contact

A Contact model has the following variables:

- name
- phone
- email
- position
- customer

Data Access Layer

The repositories for this task extends on the JpaRepository from the Spring framework. Thus, the default function such as save(), findById() and delete() are used.

The CustomerRepository has an extra definition which enables the application to search the database when provided the search term. This is implemented as the function findByCompanyName() and returns a List of the matches.

Database

The project currently uses the H2 in memory database.

Messaging Demonstration

CustomerRunner

A CustomerRunner is implemented in customer-service. This component implements CommandLineRunner which will run when the application is started. In this case, five customers and five contacts will be posted and linked using REST Requests.

The Customers are as follows:


```
[
    {
        "id": 6,
        "companyName": "Microsoft",
        "address": "Redmond",
        "country": "USA",
        "contact": {
            "id": 1,
            "name": "Bill Gates",
            "phone": "450000000",
            "email": "billgates@gmail.com",
            "position": "CEO"
        }
    },
    {
        "id": 7,
        "companyName": "Apple",
        "address": "Palo Alto",
        "country": "USA",
        "contact": {
            "id": 2,
            "name": "Steve Jobs",
            "phone": "450000001",
            "email": "stevejobs@gmail.com",
            "position": "CEO"
        }
    },
    {
        "id": 8,
        "companyName": "Apple",
        "address": "Apple Park",
        "country": "USA",
        "contact": {
            "id": 3,
            "name": "Tim Cook",
            "phone": "450000002",
            "email": "timcook@gmail.com",
            "position": "CEO"
        }
    },
    {
        "id": 9,
        "companyName": "Mojang",
        "address": "Stockholm",
        "country": "Sweden",
```

```
        "contact": {
            "id": 4,
            "name": "Steve Steve",
            "phone": "450000003",
            "email": "stevesteve@gmail.com",
            "position": "CEO"
        }
    },
    {
        "id": 10,
        "companyName": "Nickelodeon",
        "address": "New York",
        "country": "USA",
        "contact": {
            "id": 5,
            "name": "John Doe",
            "phone": "450000004",
            "email": "johndoe@gmail.com",
            "position": "CEO"
        }
    }
]
```


ProductRunner

Products and ProductDetails are POST using REST similarly to CustomerRunner. This component posts five products and product details.

The Products and Product Details are shown below.


```
[
    {
        "id": 1,
        "productCategory": "Confectionery",
        "productName": "Chocolate Cookie",
        "price": 2,
        "stockQuantity": 10000,
        "productDetail": {
            "id": 6,
            "description": "Quite a Chocolate-ish Chocolate cookie.",
            "comment": "Gluten Free"
        }
    },
    {
        "id": 2,
        "productCategory": "Confectionery",
        "productName": "Vanilla Cookie",
        "price": 2,
        "stockQuantity": 10000,
        "productDetail": {
            "id": 7,
            "description": "Quite a Vanilla-ish Vanilla cookie.",
            "comment": "Not Gluten Free"
        }
    },
```

```
    {
        "id": 3,
        "productCategory": "Confectionery",
        "productName": "Caramel Cookie",
        "price": 2,
        "stockQuantity": 10000,
        "productDetail": {
            "id": 8,
            "description": "Quite a Caramel-ish Caramel cookie.",
            "comment": "Gluten Free"
        }
    },
    {
        "id": 4,
        "productCategory": "Confectionery",
        "productName": "White Chocolate Cookie",
        "price": 2,
        "stockQuantity": 10000,
        "productDetail": {
            "id": 9,
            "description": "Quite a White Chocolate-ish White Chocolate cookie.",
            "comment": "Not Gluten Free"
        }
    },
    {
        "id": 5,
        "productCategory": "Confectionery",
        "productName": "Plain Cookie",
        "price": 2,
        "stockQuantity": 10000,
        "productDetail": {
            "id": 10,
            "description": "Quite a Plain-ish Plain cookie.",
            "comment": "Not Gluten Free"
        }
    }
]
```


OrderRunner

This component uses CommandLineRunner to make REST Requests to make orders every 5 seconds. A random productName is selected from the available productNames. In addition, a random customerId is also selected from the available customerIds. A random quantity is selected from a range of 1 to 10.

Note that in this case customerIds range from 6 to 10 inclusive.

An example order request is as follows.

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Chocolate Cookie",
        "quantity" : "10"
    }' http://localhost:8081/order/customer?id=6
```

Example of the order request is shown below.

```
[
    {
        "supplier": "Coles",
        "productName": "Chocolate Cookie",
        "quantity": 10,
        "customerId": 6
    }
]
```

Next, another two example order request is made.

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Caramel Cookie",
        "quantity" : "2"
    }' http://localhost:8081/order/customer?id=6
```

Example of the order request is shown below.

```
[
    {
        "supplier": "Coles",
        "productName": "Caramel Cookie",
        "quantity": 2,
        "customerId": 6
    }
]
```

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Chocolate Cookie",
        "quantity" : "2"
    }' http://localhost:8081/order/customer?id=6
```

Example of the order request is shown below.

```
[
    {
        "supplier": "Coles",
        "productName": "Chocolate Cookie",
        "quantity": 2,
        "customerId": 6
    }
]
```

Messages

The message shown below will be available on the command console.

```
[Log]: Chocolate Cookie, 10.0
[Log]: Caramel Cookie, 2.0
[Log]: Chocolate Cookie, 12.0
```

The output above shows that a QuantitySummary of each Product is printed.

Interactive Query

Get the List of Product for a Customer

- Input:

```
curl -X GET http://localhost:8083/customer/6/product-list/
```

- Output:

```
[
    "Chocolate Cookie",
    "Caramel Cookie",
]
```

Get the Total Order Value for a Customer

- Input:

```
curl -X GET http://localhost:8083/customer/6/total-order-value/
```

- Output:

```
48
```

Examples of Input and Output

Customer Service

Request 1: Creating a new customer entry.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "companyName":"Microsoft",
        "address" : "Redmond",
        "country" : "USA"
    }' http://localhost:8082/customer
```

- Output:

```
"id":1,"companyName":"Microsoft","address":"Redmond","country":"USA","contact":null}
```

Request 2: Creating a new contact entry.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "name" : "Bill Gates",
        "phone" : "04555555555",
        "email" : "thisisgates@gmail.com",
        "position" : "CEO"
    }' http://localhost:8082/contact
```

- Output:

```
{"id":2,"name":"Bill Gates","phone":"04555555555","email":"thisisgates@gmail.com",
"position":"CEO"}
```

Request 3: Linking a Customer and a Contact.

- Input:

```
curl -i -X PUT http://localhost:8082/customer/1/contact/2
```

- Output:

```
{"id":1,"companyName":"Microsoft","address":"Redmond","country":"USA",
"contact":{"id":2,"name":"Bill Gates","phone":"04555555555","email":"thisisgates@gmail.com",
"position":"CEO"}}
```

Request 4: Updating a specific contact by ID.

- Input:

```
curl -i -X PUT -H "Content-Type:application/json" -d \
    '{
        "name" : "Tim Cook",
        "phone" : "04555555557",
        "email" : "thisistim@hotmail.com",
        "position" : "CEO",
        "customer" : "Apple"
    }' http://localhost:8082/contact/2
```

- Output:

```
{"id":2,"name":"Tim Cook","phone":"04555555557","email":"thisistim@hotmail.com",
"position":"CEO"}
```

Request 5: Get all customers in the database.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "companyName":"Apple",
        "address" : "California",
        "country" : "USA"
    }' http://localhost:8082/customer

curl -X GET http://localhost:8082/customer
```

- Output:

```
{"id":3,"companyName":"Apple","address":"California","country":"USA","contact":null}
[{"id":1,"companyName":"Microsoft","address":"Redmond","country":"USA",
"contact":{"id":2,"name":"Tim Cook","phone":"04555555557","email":"thisistim@hotmail.com",
"position":"CEO"}},{"id":3,"companyName":"Apple","address":"California",
"country":"USA","contact":null}]
```

Request 6: Get a specific customer in the database.

- Input:

```
curl -X GET http://localhost:8082/customer/3
```

- Output:

```
{"id":3,"companyName":"Apple","address":"California","country":"USA","contact":null}
```

Request 7: Search for a customer by company name.

- Input:

```
curl -X GET http://localhost:8082/customer/company?name=Apple
```

- Output:

```
[{"id":3,"companyName":"Apple","address":"California","country":"USA","contact":null}]
```

Request 8: Get all contacts in the database.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "name" : "Bill Gates",
        "phone" : "04555555555",
        "email" : "thisisgates@gmail.com",
        "position" : "CEO"
    }' http://localhost:8082/contact

curl -X GET http://localhost:8082/contact
```

- Output:

```
{"id":4,"name":"Bill Gates","phone":"04555555555","email":"thisisgates@gmail.com",
"position":"CEO"}[{"id":2,"name":"Tim Cook","phone":"04555555557",
"email":"thisistim@hotmail.com","position":"CEO"},{"id":4,"name":"Bill Gates",
"phone":"04555555555","email":"thisisgates@gmail.com","position":"CEO"}]
```

Request 9: Get a specific contact in the database.

- Input:

```
curl -X GET http://localhost:8082/contact/4
```

- Output:

```
{"id":4,"name":"Bill Gates","phone":"04555555555","email":"thisisgates@gmail.com",
"position":"CEO"}
```

Product Service

Request 1: Post Product Detail.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "description" : "A pretty cool chocolate cookie.",
        "comment" : "Might have peanuts."
    }' http://localhost:8080/product/detail
```

- Output:

```
{"id":1,"description":"A pretty cool chocolate cookie.","comment":"Might have peanuts."}
```

Request 2: Post Product.

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "productCategory" : "Confectionery",
        "productName" : "Triple Chocolate Cookie",
        "price" : "100",
        "stockQuantity" : "5"
    }' http://localhost:8080/product
```

- Output:

```
{"id":2,"productCategory":"Confectionery","price":100,"stockQuantity":5,"productDetail":null,
"name":"Triple Chocolate Cookie"}
```

Request 3: Get Product Detail.

- Input:

```
curl -X GET http://localhost:8080/product/detail/
```

- Output:

```
[{"id":1,"description":"A pretty cool chocolate cookie.","comment":"Might have peanuts."}]
```

Request 4: Get Product

- Input:

```
curl -X GET http://localhost:8080/product/
```

- Output:

```
{"id":2,"productCategory":"Confectionery","price":100,"stockQuantity":5,"productDetail":null,
"name":"Triple Chocolate Cookie"}
```

Request 5: Link Product - Product Detail

- Input:

```
curl -i -X PUT http://localhost:8080/product/2/detail/1
```

- Output:

```
{"id":2,"productCategory":"Confectionery","price":100,"stockQuantity":5,"productDetail":{"id":1,
"description":"A pretty cool chocolate cookie.","comment":"Might have peanuts."},
"name":"Triple Chocolate Cookie"}
```

Request 6: Update Stock

- Input:

```
curl -i -X PUT -H "Content-Type:application/json" -d \
    '{
        "stockQuantity" : "5"
    }' "http://localhost:8080/product/update?name=Triple%20Chocolate%20Cookie"
```

- Output:

{"id":2,"productCategory":"Confectionery","price":100,"stockQuantity":5,"productDetail":{"id":1,
"description":"A pretty cool chocolate cookie.","comment":"Might have peanuts."},
"name":"Triple Chocolate Cookie"}

Order Service

Request 1: Post Order

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Triple Chocolate Cookie",
        "quantity" : "2"
    }' http://localhost:8081/order/customer?id=1
```

- Output:

```
Customer Address : "Redmond"
Customer Contact : "04555555555"
Unit Price : 100
Unit(s) Ordered : 2
```

Request 2: Post Order without Valid Customer

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Triple Chocolate Cookie",
        "quantity" : "2"
    }' http://localhost:8081/order/customer?id=5
```

- Output:

```
Customer with id : 5 is Not Found
```

Request 3: Post Order without Sufficient Stock

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Triple Chocolate Cookie",
        "quantity" : "20"
    }' http://localhost:8081/order/customer?id=1
```

- Output:

```
Customer Address : "Redmond"
Customer Contact : "04555555555"
Not Enough Stock
```

Request 4: Post Order without Valid Product

- Input:

```
curl -X POST -H "Content-Type:application/json" -d \
    '{
        "supplier" : "Coles",
        "productName" : "Better Multiple Chocolate Cookie",
        "quantity" : "2"
    }' http://localhost:8081/order/customer?id=1
```

- Output:

```
Customer Address : "Redmond"
Customer Contact : "04555555555"
Product Not Found
```

Request 5: Find Customer By Order Id

In this case, the OrderEvent is saved to the database before Order. This Order has the id of 2. - Input:

```
curl -X GET http://localhost:8081/order/2/customer
```

- Output

```
{"id":1,"companyName":"Microsoft","address":"Redmond","country":"USA",
"contact":{"id":2,"name":"Bill Gates","phone":"04555555555","email":"thisisgates@gmail.com",
"position":"CEO"}}
```

Request 6: Find Product Details By Order Id

- Input:

```
curl -X GET http://localhost:8081/order/2/product
```

```
{"id":2,"productCategory":"Confectionery","price":100,"stockQuantity":5,"productDetail":{"id":1,
"description":"A pretty cool chocolate cookie.","comment":"Might have peanuts."},
"name":"Triple Chocolate Cookie"}
```