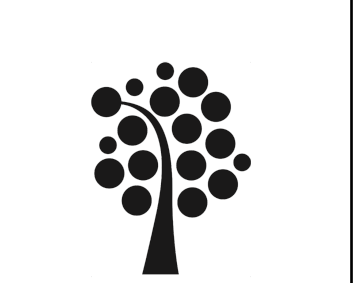

4DV609

Project Development
- individual report

November 11, 2018

Semester: Autumn 2018
Author: Sarpreet Singh Buttar
Email: sb223ce@student.lnu.se

1 Introduction

The report presents the design of two microservices, named as user and cart, that I implemented during the development of SmartShopping application. The implementation of SmartShopping application is available on GitHub, `https://github.com/prasannjeet/SmartShopping`

2 User Microservice

User microservice is responsible to manage the users of SmartShopping application. It is implemented according to the command query responsibility segregation (CQRS) pattern. CQRS divides the user microservice in to command and query services, see figure 2.1.

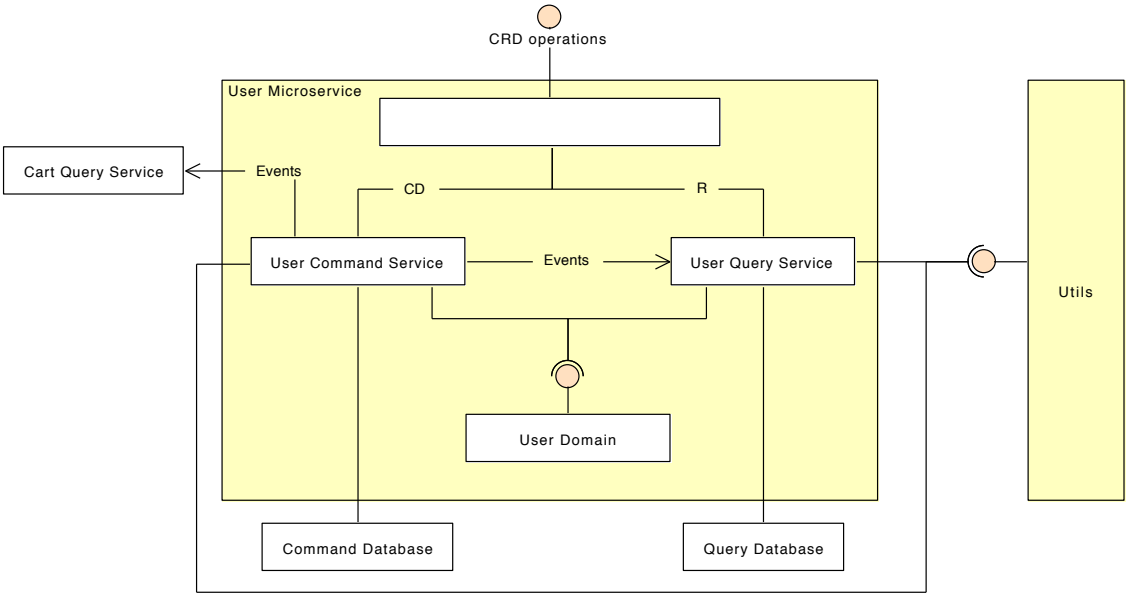

Figure 2.1: An overview of the user microservice

The user microservice provides CRD, i.e., create a user (C), retrieve a user; retrieve users (R), and delete a user (D), operations to the end user. The user command service implements the C and D operations, whereas the user query service implements the R operation. Both of these services have its own database, and the user query service synchronizes with the user command service by subscribing its events that are published whenever the user command service updates its database. The cart query service is a part of the cart microservice, and also subscribes to the events of the user command service. It creates an empty cart when a user is created, and deletes the cart when the user is deleted. The domain entities of the user microservice are placed inside the user domain. The exception handling logic is implemented inside the utils module (a standalone module) and is used by all the microservices of SmartShopping application.

The CQRS pattern has been used because it:

1. provides an efficient way to implementation queries in microservice architecture
2. enables querying the data in an event sourcing-based application

2.1 Class Diagrams

This section presents the class diagrams of the user domain, user command service, user query service, and utils module.

2.1.1 User Domain

User domain has one model named as user that has id, name, and password properties. In addition, the user domain also contains all the events that are published by the user command service and subscribed by the user query service and cart query service. It also has a repository to save the user model in to the query database. Figure 2.2 shows the class diagram of the user domain.

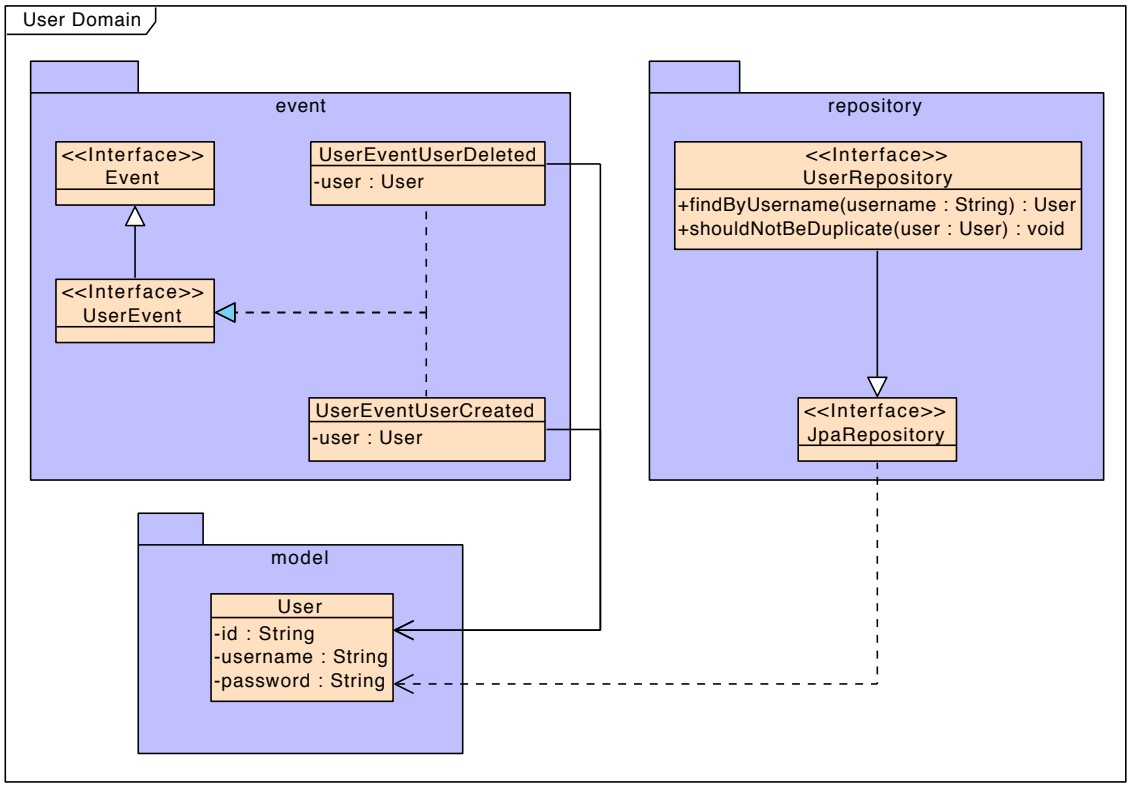

Figure 2.2: Class diagram of the user domain

2.1.2 User Command Service

User command service contains all the commands that are used to create or update a user aggregate. A user aggregate is created or updated when a C or D operation is executed. Controller uses commandservice to respond the http request made by the end user. The commandservice uses an aggregate repository to create or update the user aggregate. An event is always published after

creating or updating the user aggregate. Figure 2.3 shows the class diagram of the user command service.

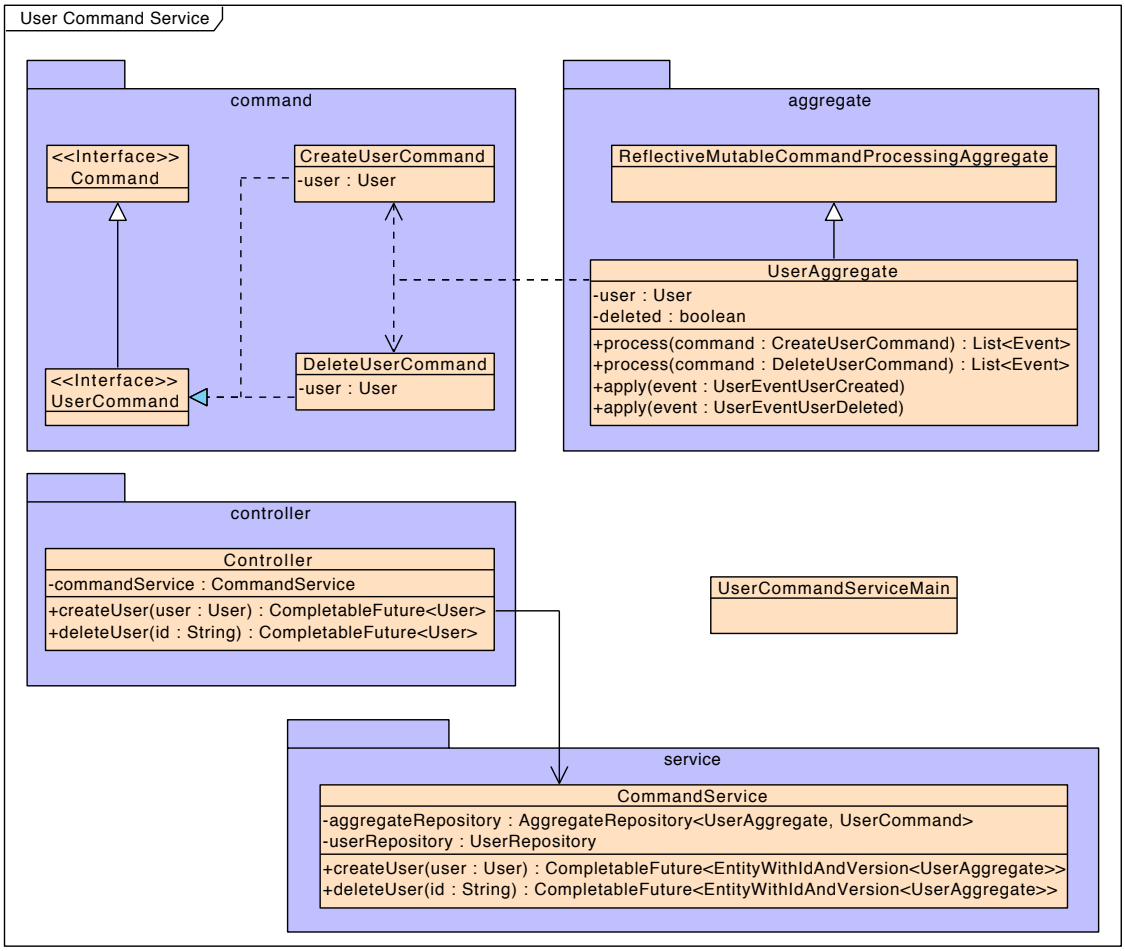

Figure 2.3: Class diagram of the user command service

2.1.3 User Query Service

User query service contains a controller that uses a query service to respond the http request made by the end user. In addition, it also has a subscriber that subscribes the events published by the user command service. These events keep the user query service's database up to date. Figure 2.4 shows the class diagram of the user query service.

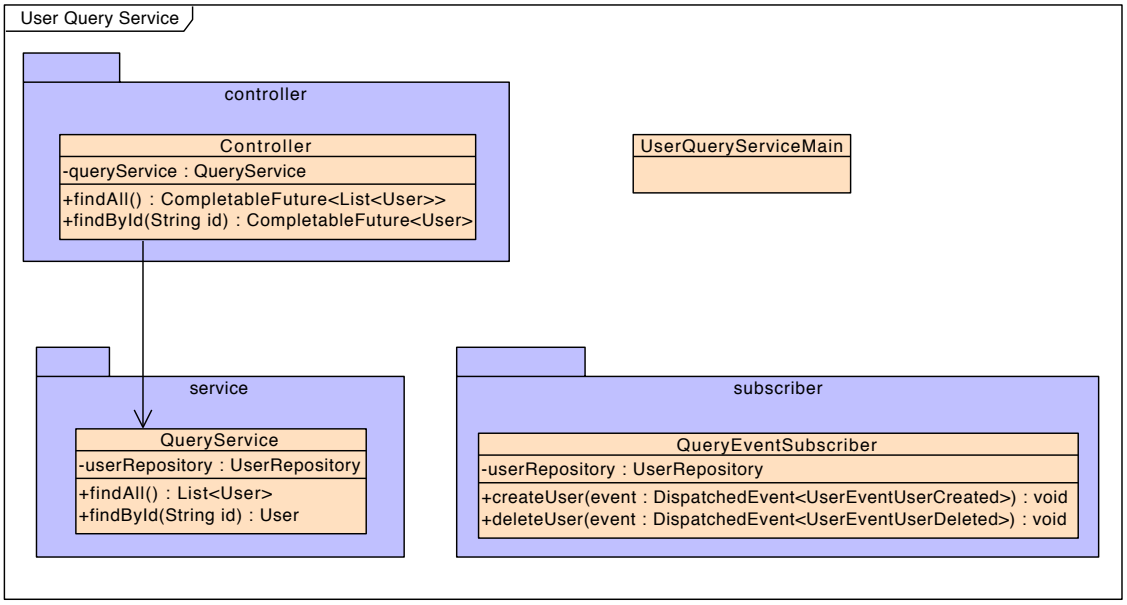

Figure 2.4: Class diagram of the user query service

2.1.4 Utils Module

Utils is an independent module and is not part of any micoservice of SmartShopping application. However, it is used by all the microservices of SmartShopping application to handle the exceptions such as entity not found, bad request, etc. This module enables the cross-origin resource sharing (CORS) that allow us to make http requests from one local port to another. In addition, if some exception occurs while executing the operations, an http exception handler catches it and provides a suitable feedback to the end user. Figure 2.5 shows the class diagram of the utils module.

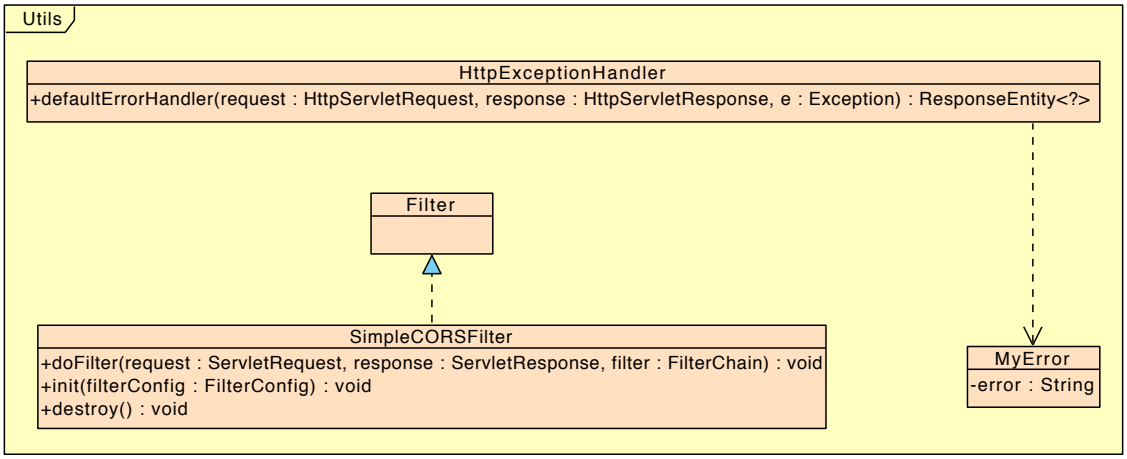

Figure 2.5: Class diagram of the utils moodule

2.2 Public APIs

This section presents the public API's of the user microservice.

Create a user

- **URL** `/users`
- **Method:** `POST`
- **Headers:** `Content-Type: application/json`
- **Body:**

```
{
    "username": "bob",
    "password": "pas111"
}
```

- **Response:**

```
{
    "id": "ksnc123n212"
    "username": "bob",
    "password": "pas111"
}
```

Retrieve users

- **URL** `/users`
- **Method:** `GET`
- **Response:**

```
[
    {
        "id": "ksnc123n212"
        "username": "bob",
        "password": "pas111"
    },
    ...
]
```

Retrieve a user

- **URL** `/users/{userId}`
- **Method:** `GET`

- **Response:**

```
{
    "id": "ksnc123n212"
    "username": "bob",
    "password": "pas111"
}
```

Delete a user

- **URL** `/users/{userId}`
- **Method:** `DELETE`

- **Response:**

```
{
    "id": "ksnc123n212"
    "username": "bob",
    "password": "pas111"
}
```

2.3 Activity Diagrams

This section shows the high level flow of the SmartShopping application when the user microservice's public APIs are called.

2.3.1 Create a user

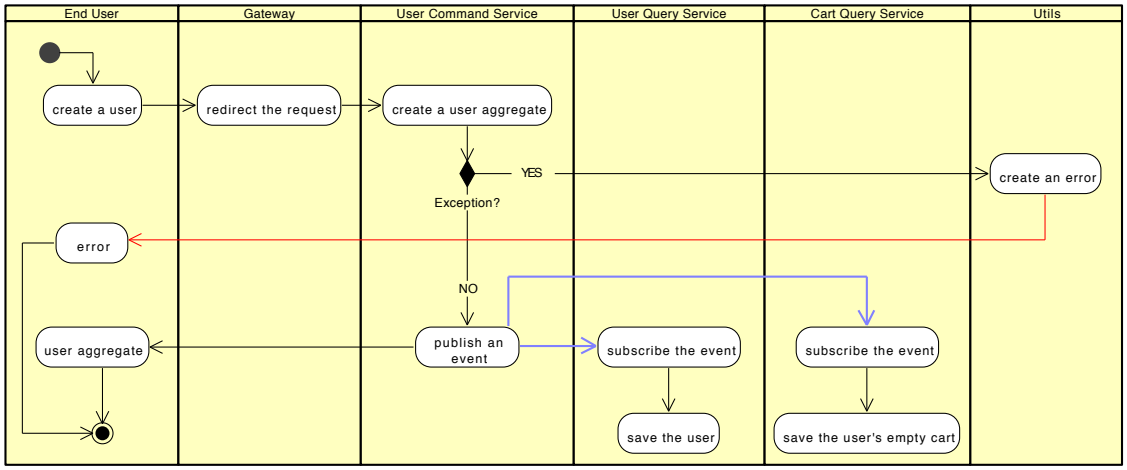

Figure 2.6: Activity diagram for creating a user

2.3.2 Retrieve users

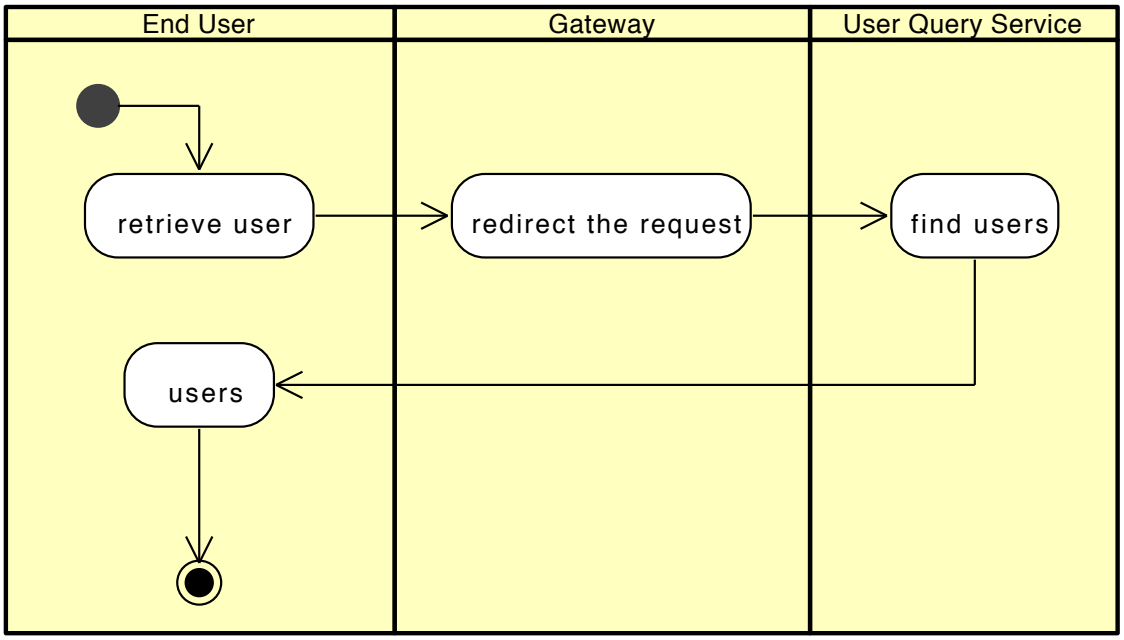

Figure 2.7: Activity diagram for retrieving users

2.3.3 Retrieve a user

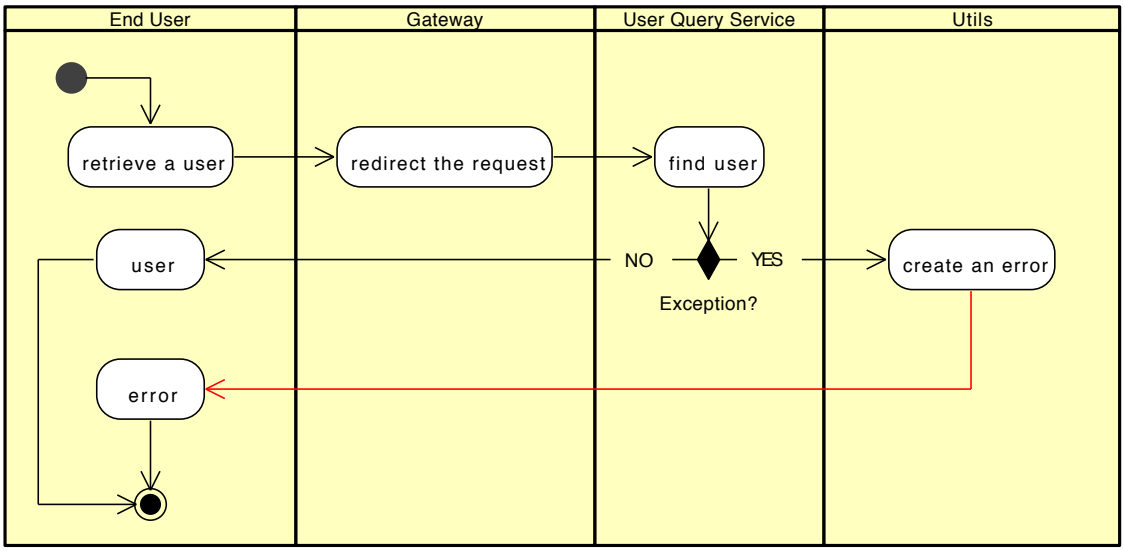

Figure 2.8: Activity diagram for retrieving a user

2.3.4 Delete a user

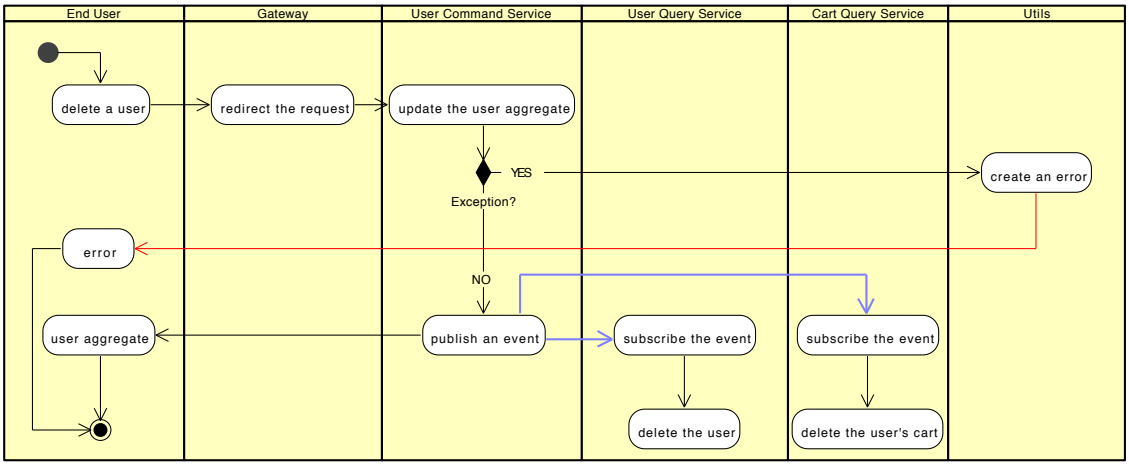

Figure 2.9: Activity diagram for deleting a user

3 Cart Microservice

Cart microservice is also implemented in the same as the user microservice. It also follows the CQRS pattern for the same reasons that has been mentioned above. Figure 3.10 shows an overview of the cart microservice.

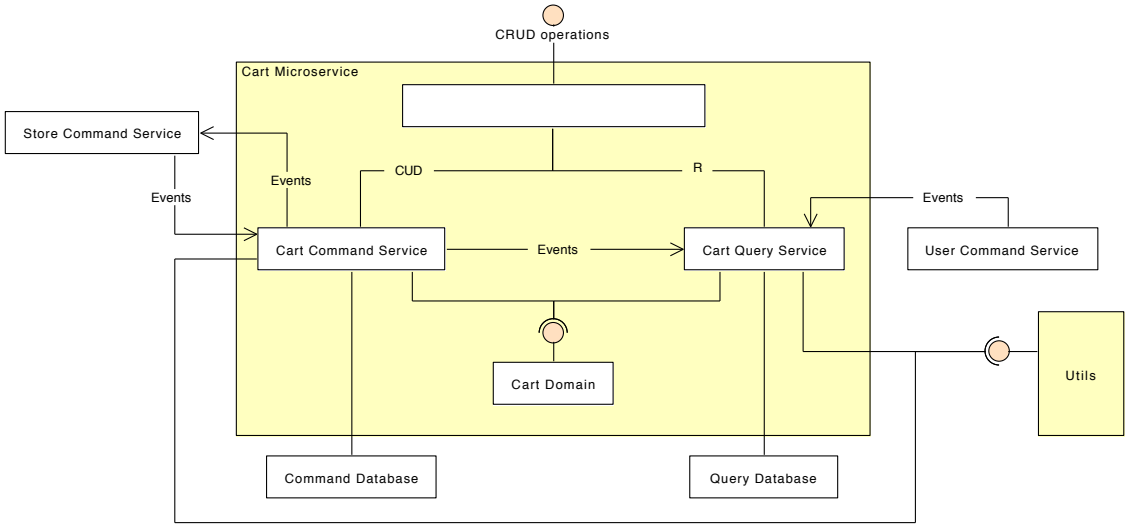

Figure 3.10: An overview of the cart microservice

The cart microservice provides CRUD, i.e., add a product in cart (C), update product quantity in cart; sort cart by stores distance; sort cart by products price (U), delete product from cart (D), retrieve carts; retrieve a cart (R), operations to the end user. The cart command service implements the C, U and D operations, whereas the cart query service implements the R operation. Both of these services have its own database, and the cart query service synchronizes with the cart command service by subscribing its events that are published whenever the cart command service updates its database. As mentioned before, the cart query service also subscribes the events from the user command service to keep the query database up to date. On the other side, the cart command service subscribes to the events of the store command service that belongs to the store microservice. The goal of this subscription is to retrieve the price of the products to sort the demanded cart when needed. In addition, the store command service also subscribes to the events of the cart command service. This subscription awakes the store command service so that it can provide the products price to the cart command service. The domain entities of the cart microservice are placed inside the cart domain. As mentioned earlier, the exception handling logic is implemented inside the utils module (a standalone module) and is used by all the microservices of SmartShopping application.

3.1 Class Diagrams

This section presents the class diagrams of the cart domain, cart command service, and cart query service.

3.1.1 Cart Domain

Cart domain has two models named as cart and product. The attribute of the cart model is user id, whereas, id, barcode, name, quantity, has weight, and user id are the attributes of the product model. In addition, the user domain also contains all the events that are published by the cart command service and subscribed by the cart query service and store command service. Repositories are also added to save the models in to the query database. The cart domain also has various dao classes that represent different variation of its models. These classes are used inside the events so that necessary data can be easily transferred. Similarly, these are also used to provide well structure response to the end user. Figure 3.11 shows the class diagram of the cart domain.

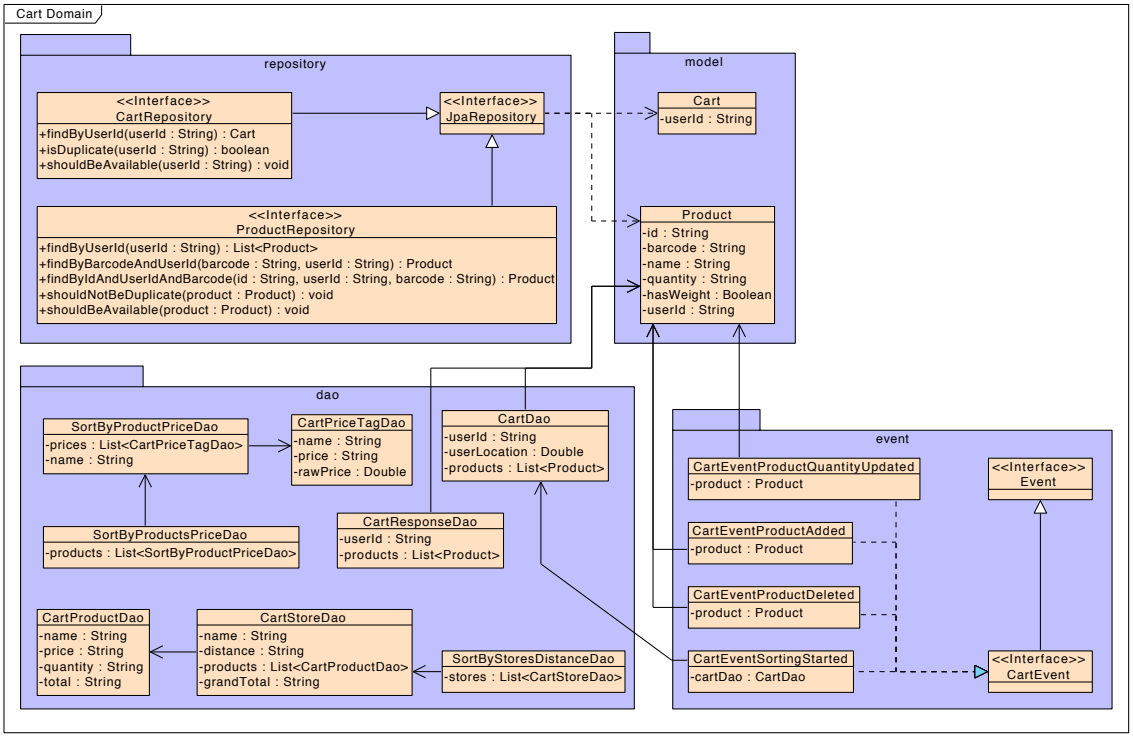

Figure 3.11: Class diagram of the cart domain

3.1.2 Cart Command Service

Cart command service contains all the commands that are used to create or update a cart aggregate. A cart aggregate is created or updated when a C, U, or D operation is executed. Controller uses commandservice to respond the http request made by the end user. The commandservice uses an aggregate repository to create or update the cart aggregate. An event is always published after creating or updating the cart aggregate. In addition, it also has a subscriber that subscribes the events published by the store command service. These events help the cart command service to sort the demanded cart. Figure 3.12 shows the class diagram of the cart command service.

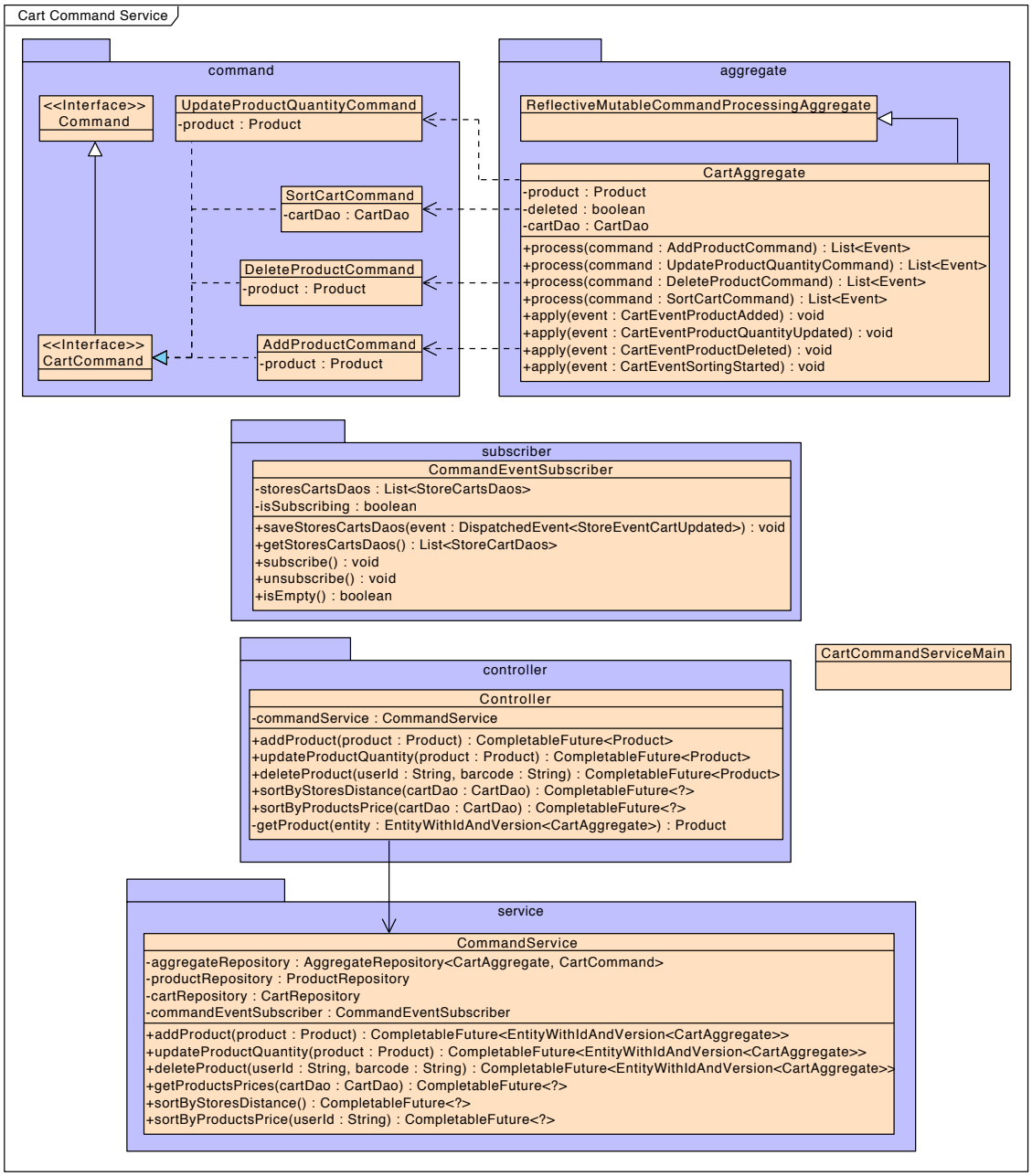

Figure 3.12: Class diagram of the cart command service

3.1.3 Cart Query Service

Cart query service contains a controller that uses a query service to respond the http request made by the end user. In addition, it also has a subscriber that subscribes the events published by the user command service as well cart command service. These events keep the cart query service's database up to date. Figure 3.13 shows the class diagram of the user query service.

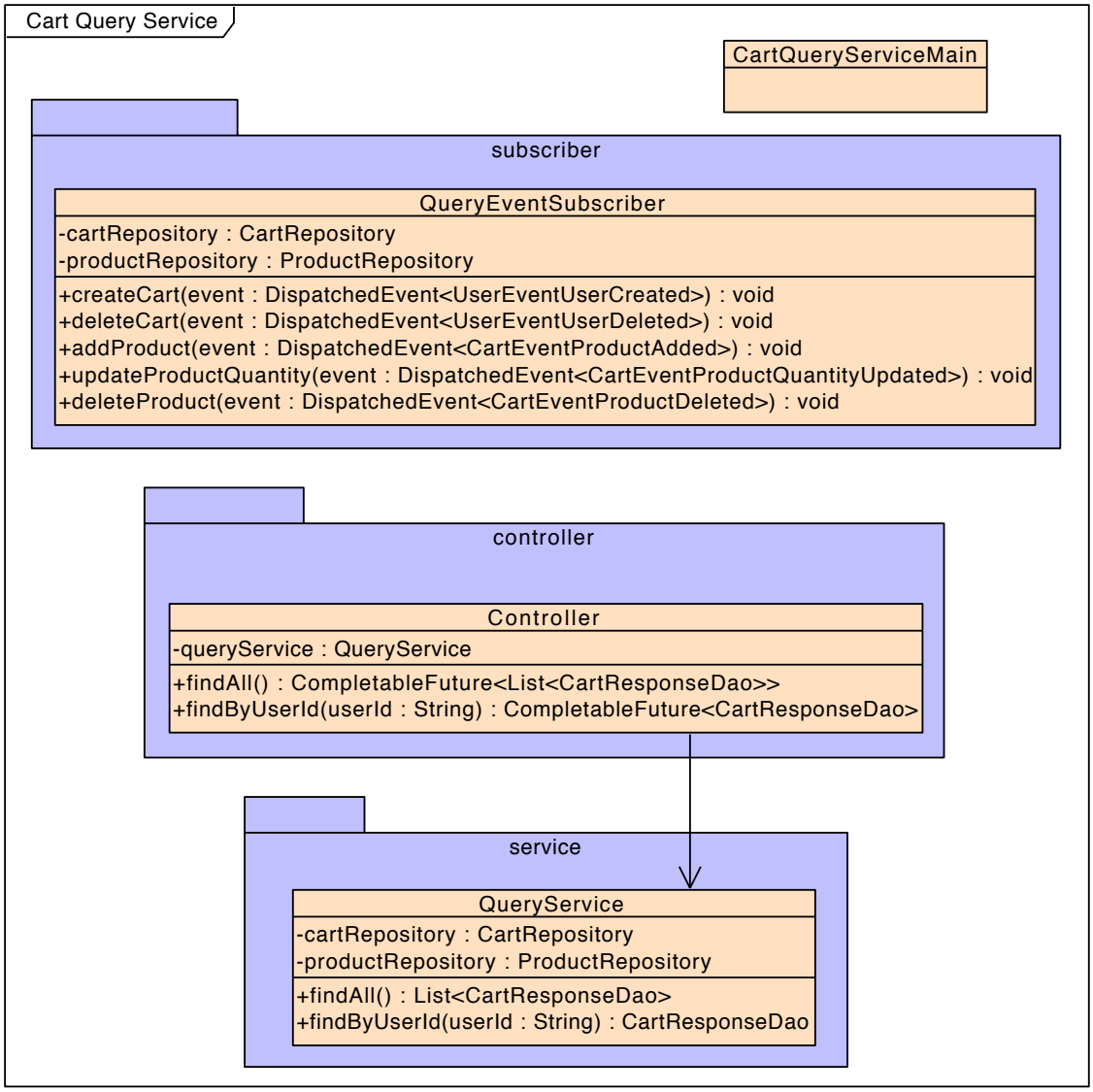

Figure 3.13: Class diagram of the cart query service

3.2 Public API's

This section presents the public API's of the cart microservice.

Retrieve carts

- **URL** `/carts`
- **Method:** `GET`

- **Response:**

```
[
    {
        "userId": "wdnfjn11mke"
        "products": [
            {
                "id": "ufhu121dm",
                "barcode": "214234",
                "name": "Carrot",
                "quantity": "2.4",
                "hasWeight": true,
                "userId": "wdnfjn11mke"
            },
            ...
        ]
    },
    ...
]
```

Retrieve a cart

- **URL** `/carts/{userId}`
- **Method:** `GET`

- **Response:**

```
{
    "userId": "wdnfjn11mke"
    "products": [
        {
            "id": "ufhu121dm",
            "barcode": "214234",
            "name": "Carrot",
            "quantity": "2.4",
```

```
            "hasWeight": true,
            "userId": "wdnfjn11mke"
        },
        ...
    ]
}
```

Add product in cart

- **URL** `/carts/products`
- **Method:** `POST`
- **Headers:** `Content-Type: application/json`
- **Body:**

```
{
    "barcode": "214234",
    "name": "Carrot",
    "quantity": "2.4",
    "hasWeight": true,
    "userId": "wdnfjn11mke"
}
```

- **Response:**

```
{
    "id": "ufhu121dm",
    "barcode": "214234",
    "name": "Carrot",
    "quantity": "2.4",
    "hasWeight": true,
    "userId": "wdnfjn11mke"
}
```

Update product quantity in cart

- **URL** `/carts/products`

- **Method:** `PUT`
- **Headers:** `Content-Type: application/json`
- **Body:**

```
{
    "id": "ufhu121dm",
    "barcode": "214234",
    "name": "Carrot",
    "quantity": "5.9",
    "hasWeight": true,
    "userId": "wdnfjn11mke"
}
```

- **Response:**

```
{
    "id": "ufhu121dm",
    "barcode": "214234",
    "name": "Carrot",
    "quantity": "5.9",
    "hasWeight": true,
    "userId": "wdnfjn11mke"
}
```

Delete product from cart

- **URL** `/carts/{userId}/products/{barcode}`
- **Method:** `DELETE`
- **Response:**

```
{
    "id": "ufhu121dm",
    "barcode": "214234",
    "name": "Carrot",
    "quantity": "5.9",
    "hasWeight": true,
    "userId": "wdnfjn11mke"
```

```
}
```

Sort cart by stores distance

- **URL** `/carts/sort/stores-distance`
- **Method:** `PUT`
- **Headers:** `Content-Type: application/json`
- **Body:**

```
{
    "userId": "wdnfjn11mke",
    "userLocation": "12"
}
```

- **Response:**

```
{
    "stores": [
        {
            "name": "Willys",
            "distance": "1.3km",
            "products": [
                {
                    "name": "Carrot",
                    "price": "2kr/kg",
                    "quantity": "5.9",
                    "total": "11.8kr"
                },
                ..
            ],
            "grandTotal": "302.73kr"
        },
        ...
    ]
}
```

Sort cart by products price

- **URL** `/carts/sort/products-price`
- **Method:** `PUT`
- **Headers:** `Content-Type: application/json`

- **Body:**

```
{
    "userId": "wdnfjn11mke",
    "userLocation": "12"
}
```

- **Response:**

```
{
    "products": [
        {
            "prices": [
                {
                    "name": "Willys",
                    "price": "2kr/kg"
                },
                ...
            ],
            "name": "Carrot"
        },
        ...
    ]
}
```

3.3 Activity Diagrams

This section shows the high level flow of the SmartShopping application when the cart microservice's public APIs are called.

3.3.1 Retrieve carts

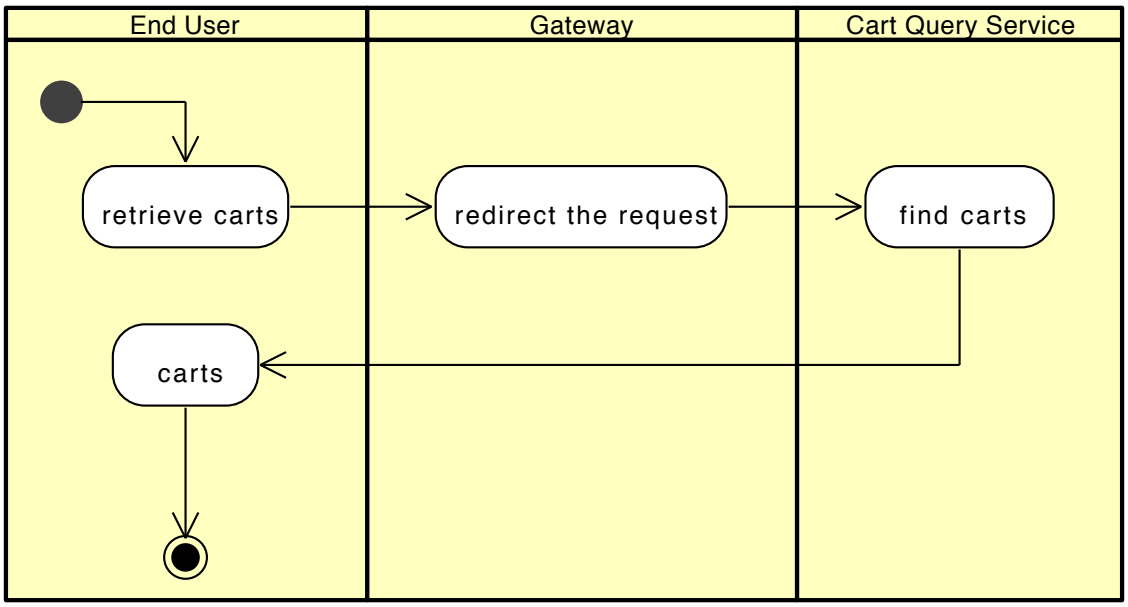

Figure 3.14: Activity diagram for retrieving carts

3.3.2 Retrieve a cart

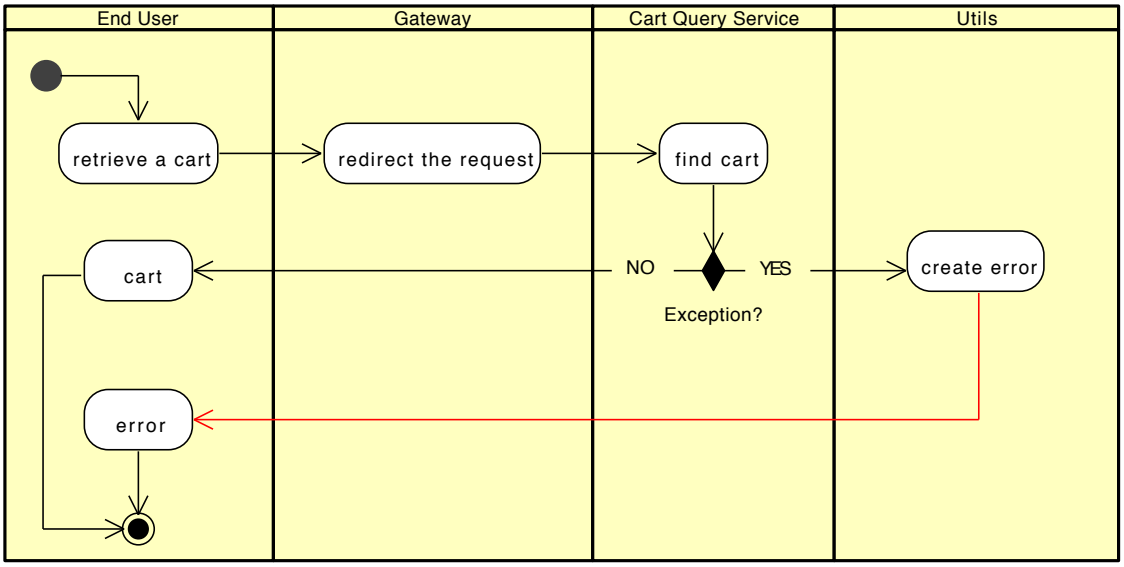

Figure 3.15: Activity diagram for retrieving a cart

3.3.3 Add product in cart

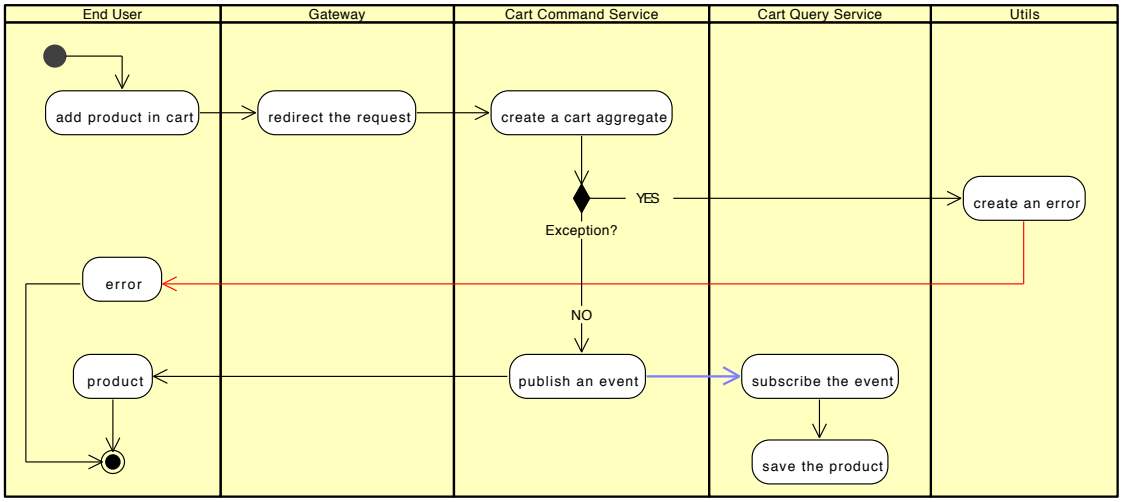

Figure 3.16: Activity diagram for adding product in cart

3.3.4 Update product quantity in cart

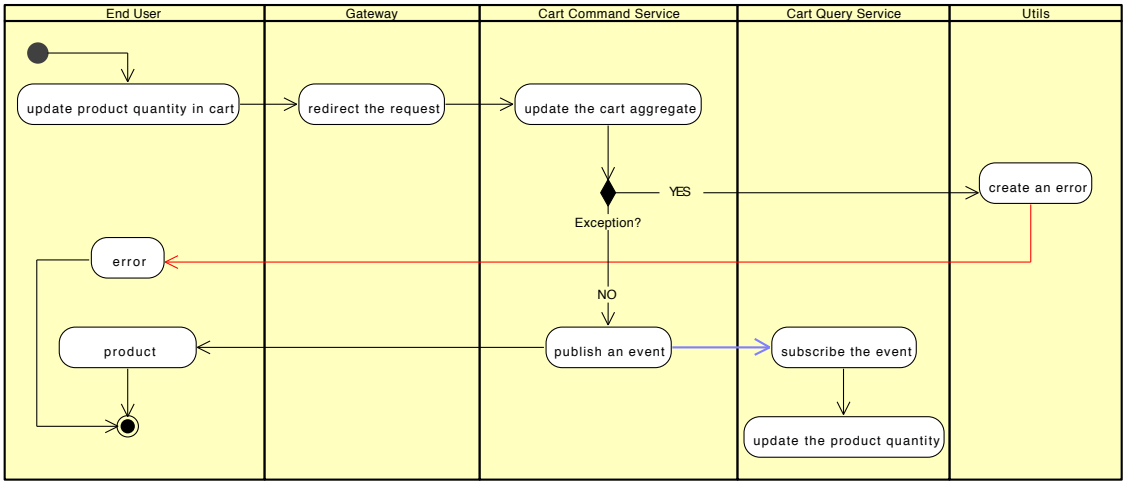

Figure 3.17: Activity diagram for updating product quantity in cart

3.3.5 Delete product from cart

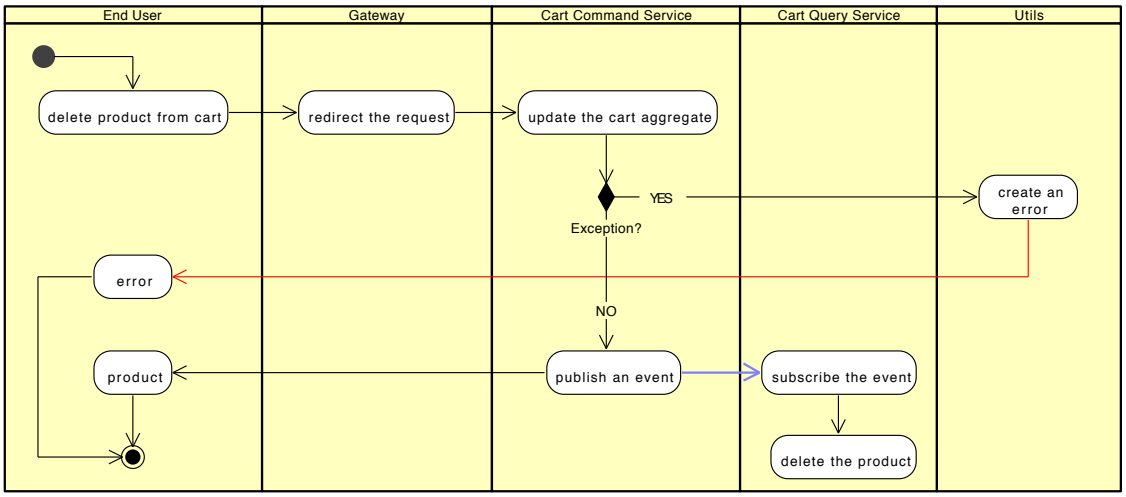

Figure 3.18: Activity diagram for deleting product from cart

3.3.6 Sort cart by stores distance

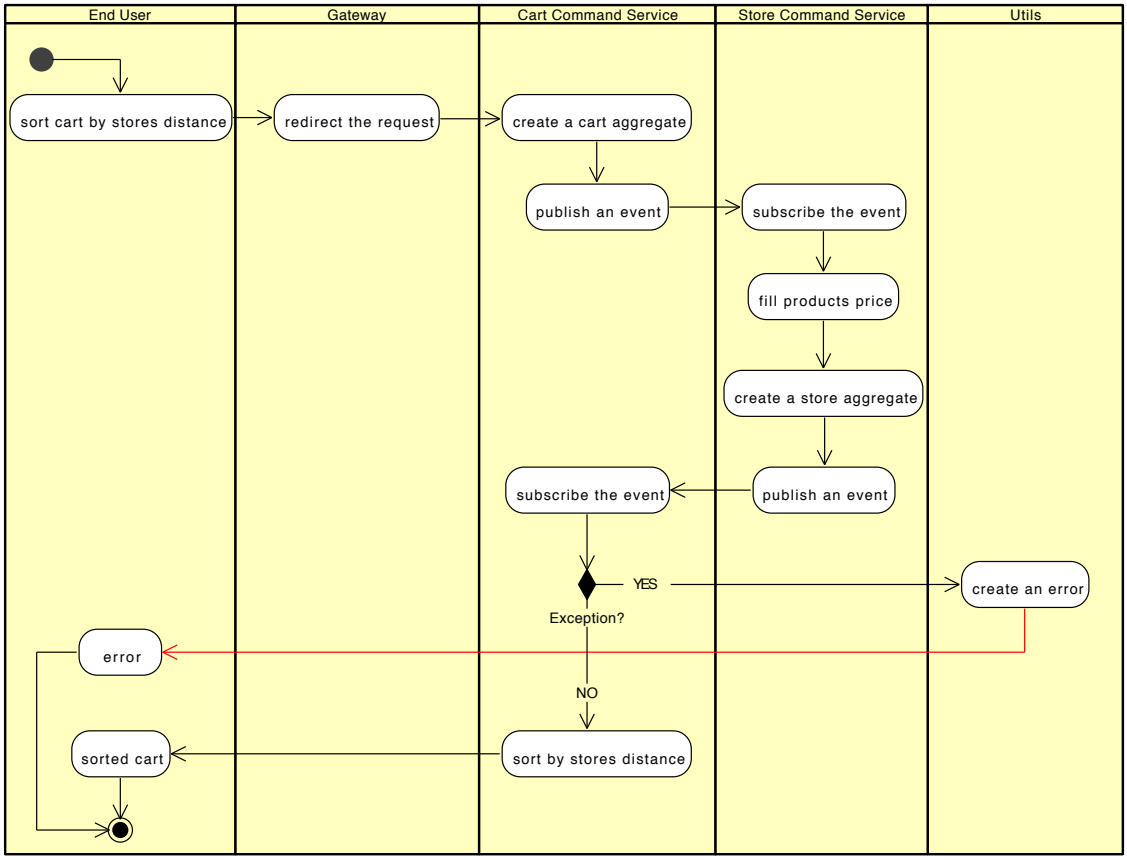

Figure 3.19: Activity diagram for sorting cart by stores distance

3.3.7 Sort cart by products price

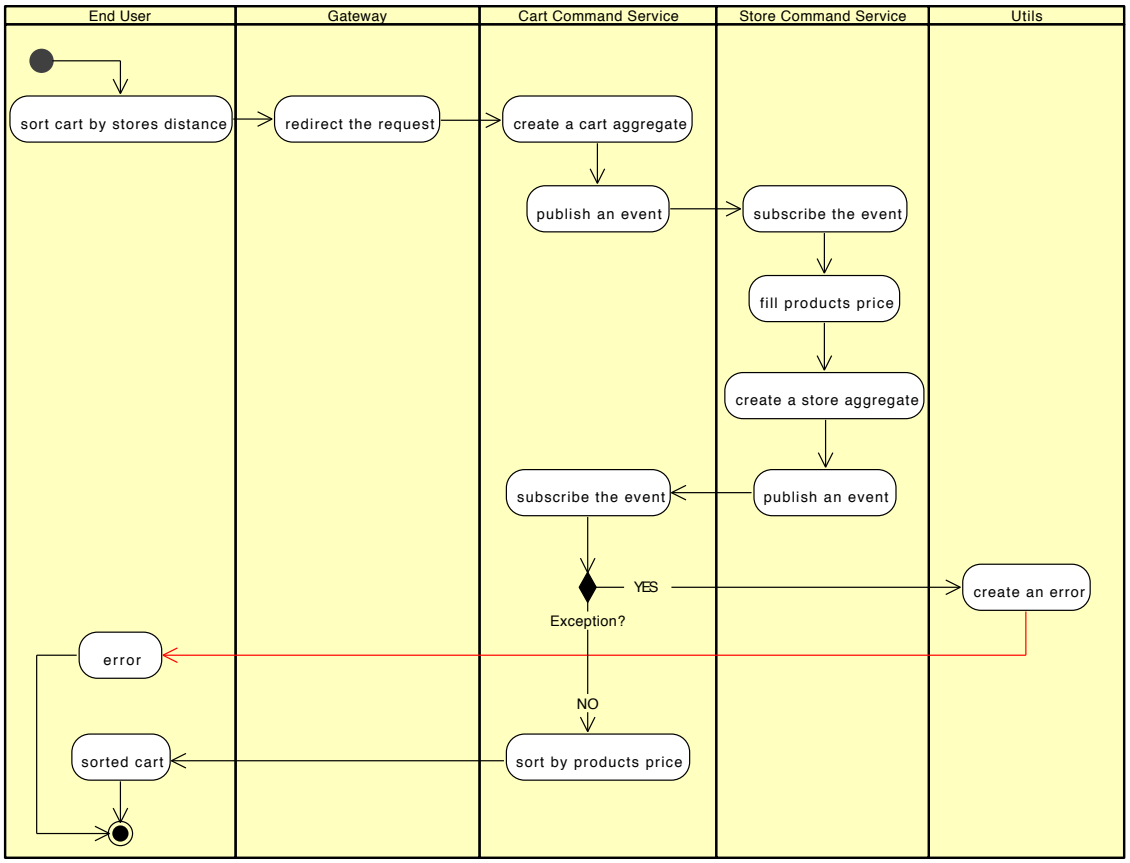

Figure 3.20: Activity diagram for sorting cart by products prices

Time Log

Date	Task	Time
19-10-18	1. Understand and run the eventuate examples	4h 30m
20-10-18	1. Understand docker	4h
21-10-18	1. Set up dev environment with maven for cart micro service	8h
22-10-18	1. Implement cart domain	4h
23-10-18	1. Implement cart query service 2. Implement cart command service (without products because other team members had not written any code)	8h
25-10-18	1. Refactor the code 2. Implement utils that handle exceptions	4h
27-10-18	1. Set up dev environment for all the team members because they could not understand where and how to start	4h
28-10-18	1. Implement add product in cart 2. Implement update product quantity 3. Implement view products in cart 4. Implement delete delete products in cart	10h
29-10-18	1. Implement update cart info 2. Implement delete cart if deleted in product service. 3. Write the API on GitHub	4h
30-10-18	1. **Implement user domain** 2. **Implement user command service** 3. **Implement user query service** 4. Set up new environment so that cart micro service can subscribe the events of user micro service. 5. Implement adding cart via user micro service event.	10h
31-10-18	1. Set up event driven docker environment for all the team member as we moved from HTTP to events approach 2. Design discussion 3. Rough ideas for sorting cart	7h

1-11-18	1. Try if we can response end user http request by triggering the events from gateway to specific micro service 2. Updated user and cart micro services for integrating with gateway and store services	8h
2-11-18	1. Refactor all the micro services 2. Integration testing	8h
3-11-18	1. Solve bug between user and cart micro service 2. Reading eventuate docs to learn how to create channel between two micro services 3. Test a simple channel between two micro services	10h
4-11-18	1. Implement sort by stores distance 2. Implement sort by products price 3. Implement an event channel between cart and store micro service 4. Implement subscriber of store command service	9h 30m
6-11-18	1. Making overall design diagram 2. Help team members with their problems	3h
9-11-18	1. Integration testing of all micro services	3h
10-11-18	1. Write user micro service report 2. Fix minor bug in sorting the cart	5h
11-11-18	1. Write cart micro service report 2. Updating the design diagrams	8h
TOTAL		~122 h