Ryerson University

Department of Electrical & Computer Engineering
Faculty of Engineering & Architectural Science

Course Title:	Software Design Architecture
Course Number:	COE692
Semester/Year:	W2022

Instructor:	Dr. Faezeh Ensan

Assignment/Lab Number:	Lab 5
Assignment/Lab Title:	Lab 5 Part 1

Submission Date:	TBD
Due Date:	TBD

Last Name:	First Name:	Student ID:	Section:	*Signature:
Faisal	Mahir	500896206	01	MF
Patel	Dhaval	500901575	01	DP

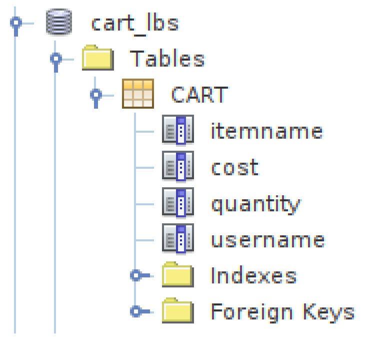
COE692 Lab 5 Part 1

Dependant Microservice Description:

There are two microservices that communicate asynchronously which are the searchItem and addItem microservices. The searchItem microservice has a global table of all the items in the store along with other information such as the quantity of each item and the cost. The addItem microservice can only add an item if the item is in stock. Hence the addItem microservice needs to know if the item is in stock. Whenever the customer searches for an item the searchItem microservice sends a message to the consumers. The addItem microservice is subscribed to the search_item_channel where it can receive and listen to these messages.

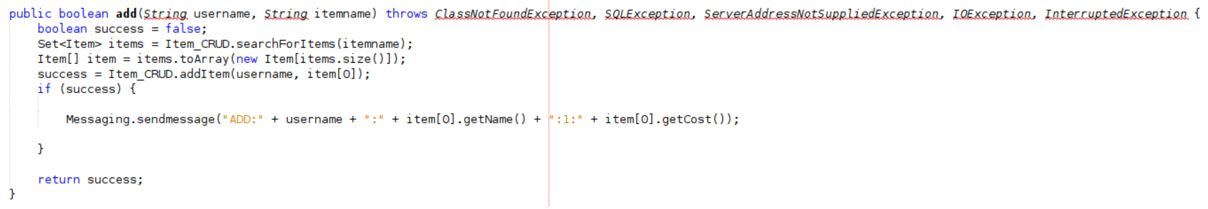
Database Description:

The addItem database has one table called cart. It stores information about the itemname, quantity, cost, and username. Whenever a customer adds an item to cart it creates a new entry in this table.



The searchItem database has two tables item and cart. The cart table stores the same information as above since it is used to keep track of local cart information for the user. The item table is a global table that contains all the items that the store needs and provides. Whenever a customer searches an item a message is sent to the search_item_channel. The message format is “ADD:itemname : quantity : cost : username“.

Sending Message:



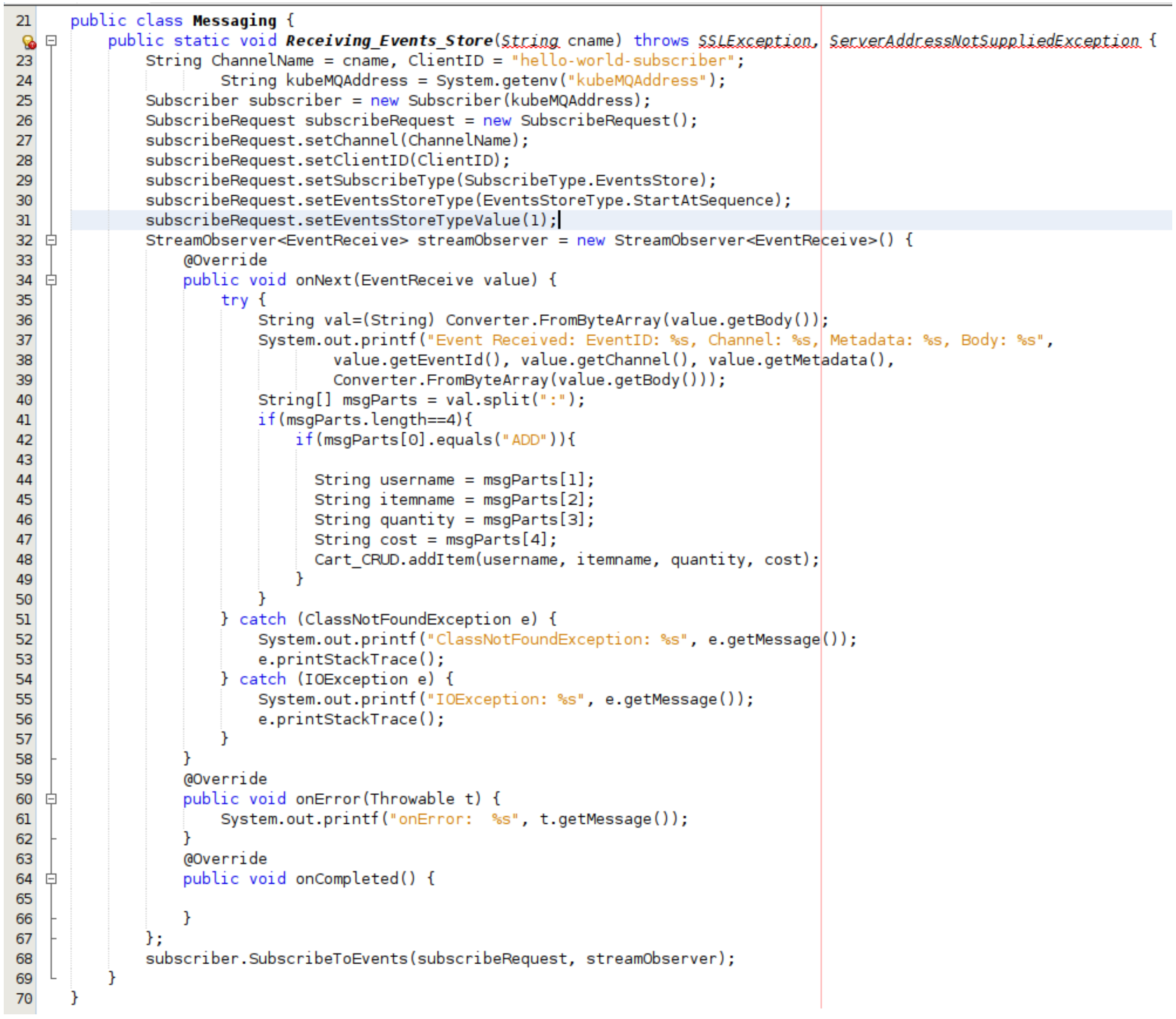
```java
public boolean add(String username, String itemname) throws ClassNotFoundException, SQLException, ServerAddressNotSuppliedException, IOException, InterruptedException {
    boolean success = false;
    Set<Item> items = Item_CRUD.searchForItems(itemname);
    Item[] item = items.toArray(new Item[items.size()]);
    success = Item_CRUD.addItem(username, item[0]);
    if (success) {

        Messaging.sendmessage("ADD:" + username + ":" + item[0].getName() + ":1:" + item[0].getCost());

    }

    return success;
}
```

Receiving Message:



```java
public class Messaging {
    public static void Receiving_Events_Store(String cname) throws SSLException, ServerAddressNotSuppliedException {
        String ChannelName = cname, ClientID = "hello-world-subscriber";
                String kubeMQAddress = System.getenv("kubeMQAddress");
        Subscriber subscriber = new Subscriber(kubeMQAddress);
        SubscribeRequest subscribeRequest = new SubscribeRequest();
        subscribeRequest.setChannel(ChannelName);
        subscribeRequest.setClientID(ClientID);
        subscribeRequest.setSubscribeType(SubscribeType.EventsStore);
        subscribeRequest.setEventsStoreType(EventsStoreType.StartAtSequence);
        subscribeRequest.setEventsStoreTypeValue(1);
        StreamObserver<EventReceive> streamObserver = new StreamObserver<EventReceive>() {
            @Override
            public void onNext(EventReceive value) {
                try {
                    String val=(String) Converter.FromByteArray(value.getBody());
                    System.out.printf("Event Received: EventID: %s, Channel: %s, Metadata: %s, Body: %s",
                            value.getEventId(), value.getChannel(), value.getMetadata(),
                            Converter.FromByteArray(value.getBody()));
                    String[] msgParts = val.split(":");
                    if(msgParts.length==4){
                        if(msgParts[0].equals("ADD")){

                          String username = msgParts[1];
                          String itemname = msgParts[2];
                          String quantity = msgParts[3];
                          String cost = msgParts[4];
                          Cart_CRUD.addItem(username, itemname, quantity, cost);
                        }
                    }
                } catch (ClassNotFoundException e) {
                    System.out.printf("ClassNotFoundException: %s", e.getMessage());
                    e.printStackTrace();
                } catch (IOException e) {
                    System.out.printf("IOException: %s", e.getMessage());
                    e.printStackTrace();
                }
            }
            @Override
            public void onError(Throwable t) {
                System.out.printf("onError:  %s", t.getMessage());
            }
            @Override
            public void onCompleted() {

            }
        };
        subscriber.SubscribeToEvents(subscribeRequest, streamObserver);
    }
}
```