Technical University of Cluj-Napoca

Software Design 2020-2021

Assignment 1

Hunor Debreczeni
Group 30433

Contents

1 Requirements 3

1.1 Assignment specification 3

1.2 Functional requirements 3

1.3 Non-functional requirements 3

2 Use case model 3

2.1 Customer 3

2.2 Administrator 4

3 System architectural design 5

4 UML Sequence diagrams 6

4.1 Login Sequence 6

4.2 Register sequence 6

4.3 Find all the Items 7

4.4 Sequence for making a new Order 8

5 Class design 9

6 Data Model 10

7 System testing 11

8 Bibliography 18

1 REQUIREMENTS

1.1 ASSIGNMENT SPECIFICATION

Use JAVA/C# API to design and implement a food delivery application. The application should have two types of users (a regular user represented by the customer and an administrator user) which have to provide a username and a password in order to use the application.

1.2 FUNCTIONAL REQUIREMENTS

There should be two types of users, the customer and the administrator.

The customer should be able to add/ read/ update/ delete his own information (name, identity card number, personal numerical code, address, etc.) only. He/she may also create/read/update/delete information from the shopping cart (items, amount of money, date and hour, delivery address, etc.). He/she may also change the payment type and see his/her historical data.

The administrator should be able to overlook the whole application by managing the users and the items from inside the application. He/she may also modify the user's data if needed and may also generate reports for orders. Customers should be able to be marked as loyal by the administrator.

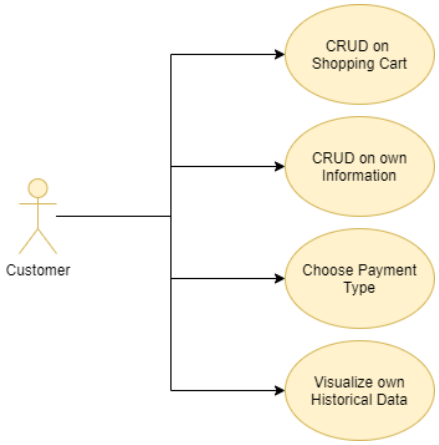
1.3 NON-FUNCTIONAL REQUIREMENTS

The information about the users/ items/ orders will be stored in a database. The Layers architectural pattern will be used to organize the whole application. A data mapper should be also used to map data from the database, to objects in the application.

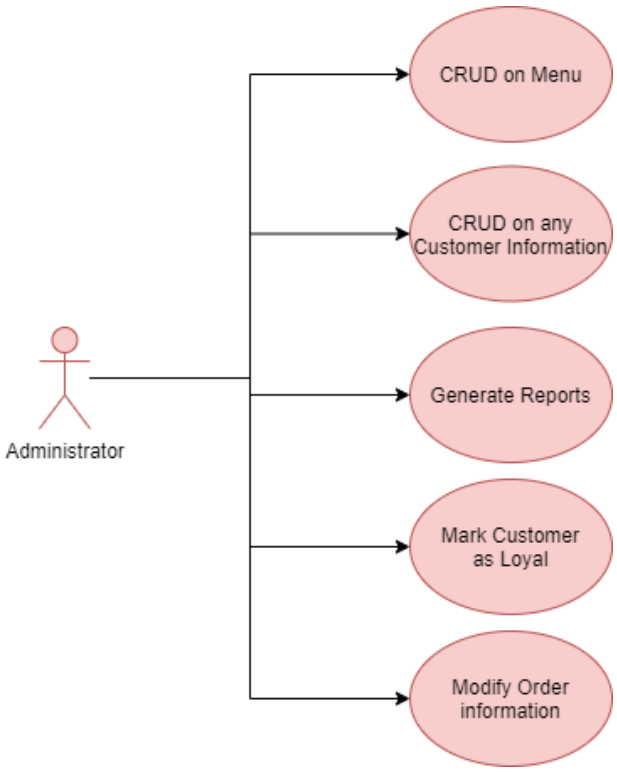
The inputs from the users will be validated and verified before inserting them into the database.

2 USE CASE MODEL

2.1 CUSTOMER



2.2 ADMINISTRATOR



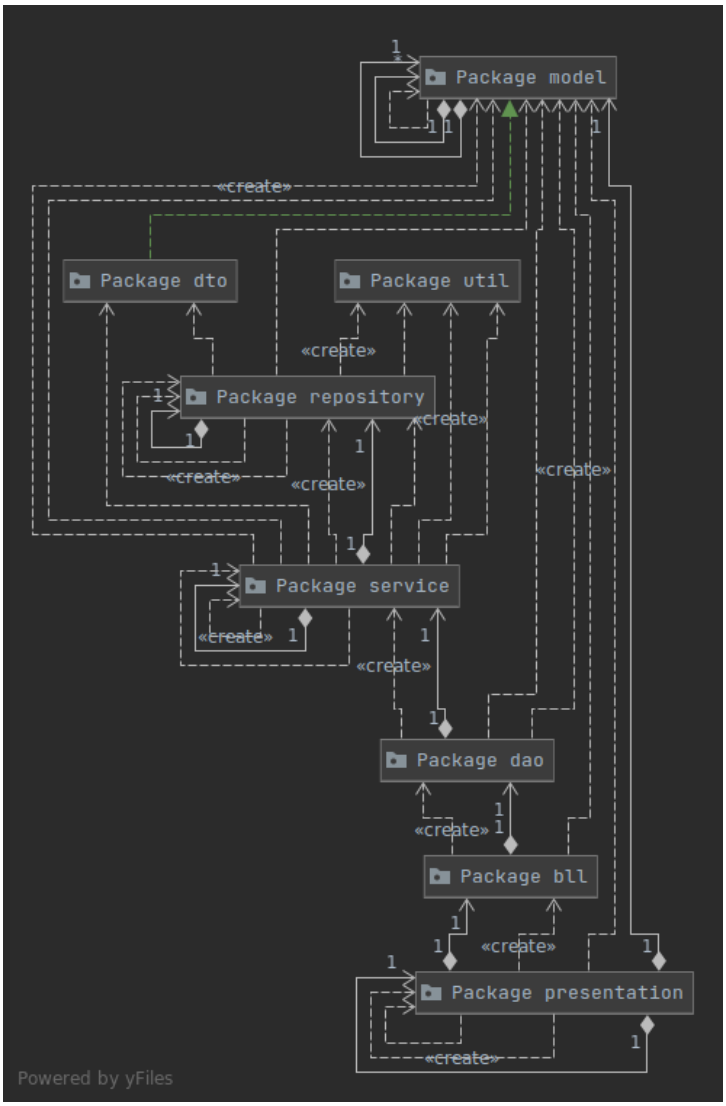
3 SYSTEM ARCHITECTURAL DESIGN

The system has been split up and structured in multiple layers. Each layer is performing a specific role.

The applications base package is the **presentation** layer, which is responsible for the UI part. The communication between the end-user and the application. It instantiates an object from the BLL (Business Level Logic) package.

For each action that comes from the presentation package, the **BLL** is responsible to check/ verify and pass it onwards. It contains the basic logic of the application.

The next layer is the **DAO** (Data Access Object). It starts processing the data as it is meant to be. We can include into this layer, the **Service** and **Repository** layers also. They are all three responsible for the processing/ fetching of the data. The Service and Repository layers are responsible for translating the data between the application and the database. Although, the Repository layer is the one that handles the transactions of the raw data between the application and the DB.

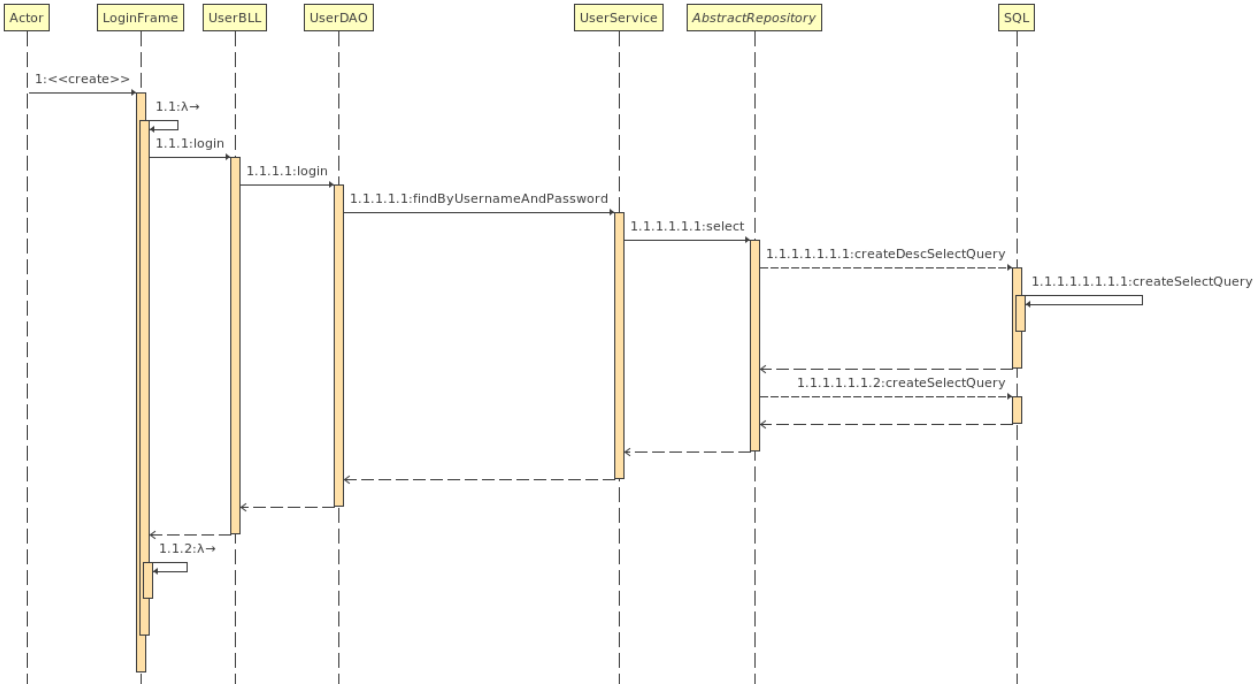


The **Model** and **DTO** packages are the objects that are passed between the layers. The DTO is used only between the service and the repository, to translate them easily.

4 UML SEQUENCE DIAGRAMS

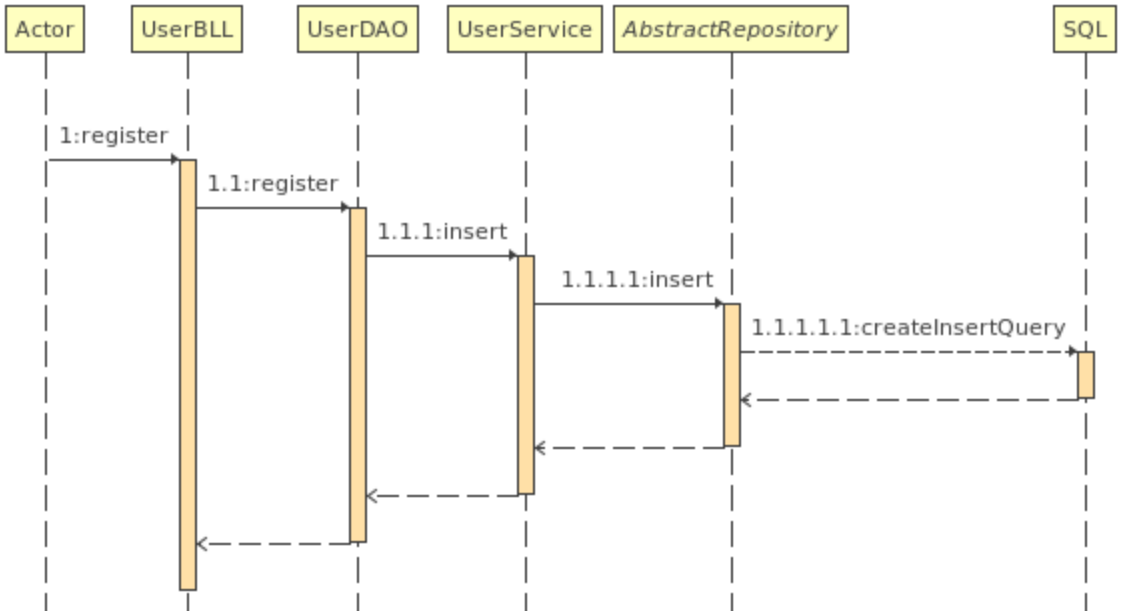
4.1 LOGIN SEQUENCE

The following diagram shows the flow for the login procedure as it passes through the layers.



4.2 REGISTER SEQUENCE

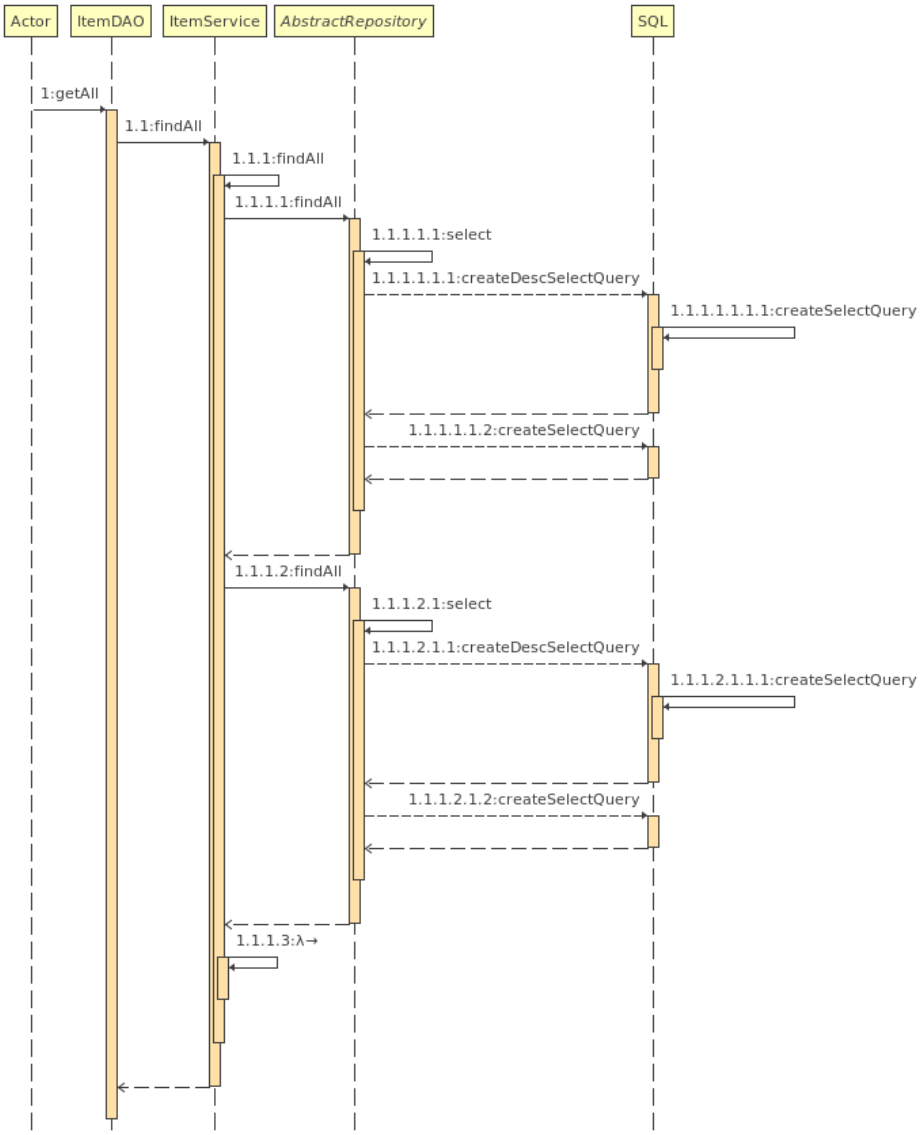
This sequence shows that the registration sequence passes through the BLL, DAO, Service and only after that reaches the Repository.



4.3 FIND ALL THE ITEMS

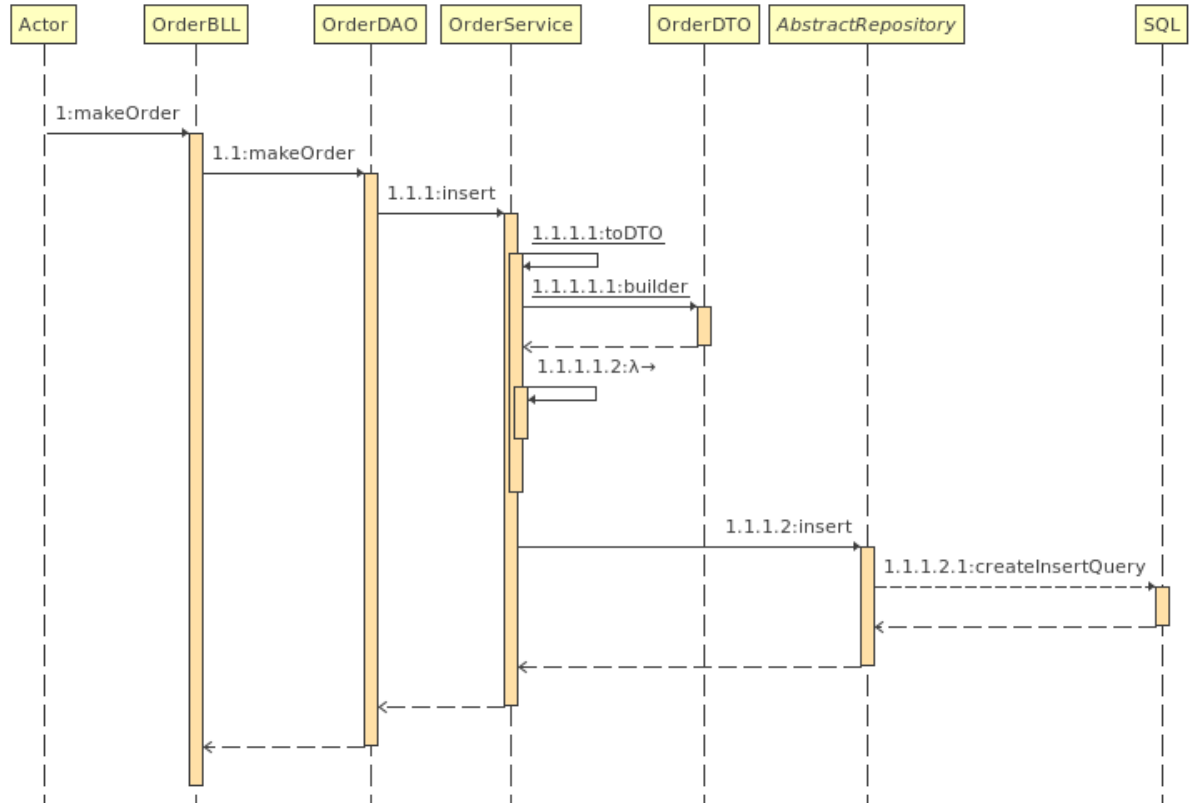
In the following diagram, we can see that the items may be found after different criteria:

- In ascending order
- In descending order
- Only those, which haven't been soft-deleted
- Every item



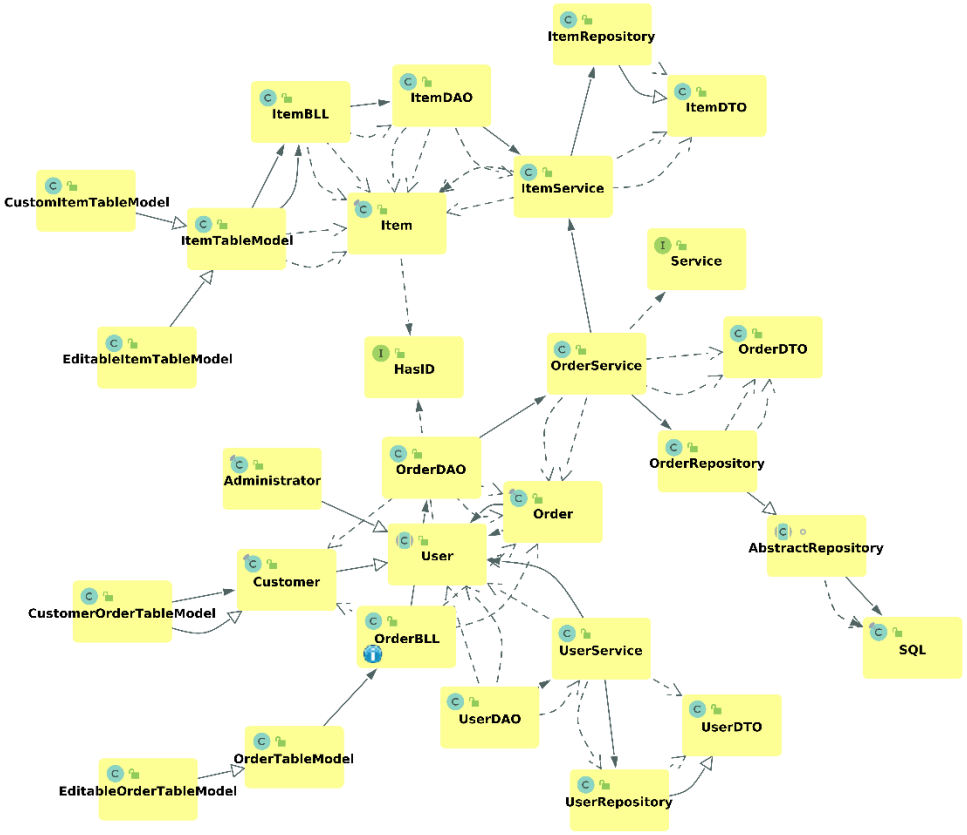
4.4 SEQUENCE FOR MAKING A NEW ORDER

The next sequence depicts, the making of a new order, and the transformation of it with the Data Transfer Object also.



5 CLASS DESIGN

Even if the application is on separate layers, the classes also work in parallel mostly. Only the Order class uses another types.



It's easily distinguishable, that the Item is working separately and even the BLL, the DAO, the Table Models and the Repository is also separate from the application. The relation which connects it together with the other parts are the services. The Order Service uses the Item Service, because an Order may have multiple items, and we have a one-to-many relationship. Upon data fetch, the Order Service must also get each item for an order.

This class diagram doesn't contain the static classes and the user interface classes, which appear in the presentation layer.

We can observe, that a lot of activity appears around the User class and the Order class. It's because we need to know the user in almost any scenario and the whole application is made to process orders.

Although, it's still observable, that the layered architecture is present, and objects are accessed from top to down, through each layer, without skipping one, or without accessing more than one layers. The only package that is accessed from multiple layers, is the model one, which contains only basic information about the main objects of the whole application (Users, Items and Orders).

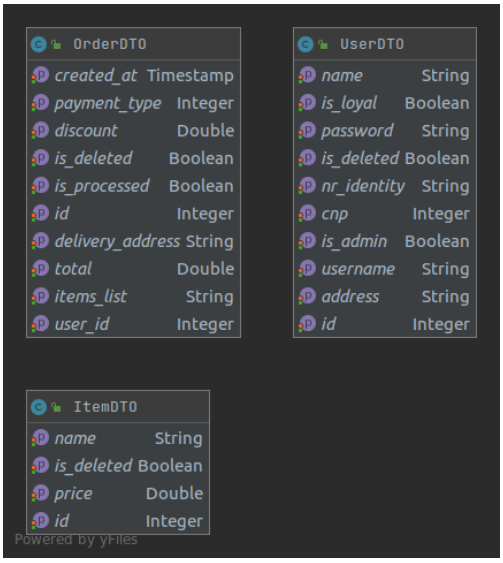
The Abstract Repository class contains the general transactions to the database. It's reached with reflection upon the DTO object. This makes it very easy to add new objects into the DB and/ or to modify them, by deleting or adding new fields and maybe even modifying their type. The only must is that the name of the field must coincide with the column/field name from the DB. The field names from the DTO classes are accessed (even if private) and their getter/setter methods to access its values.

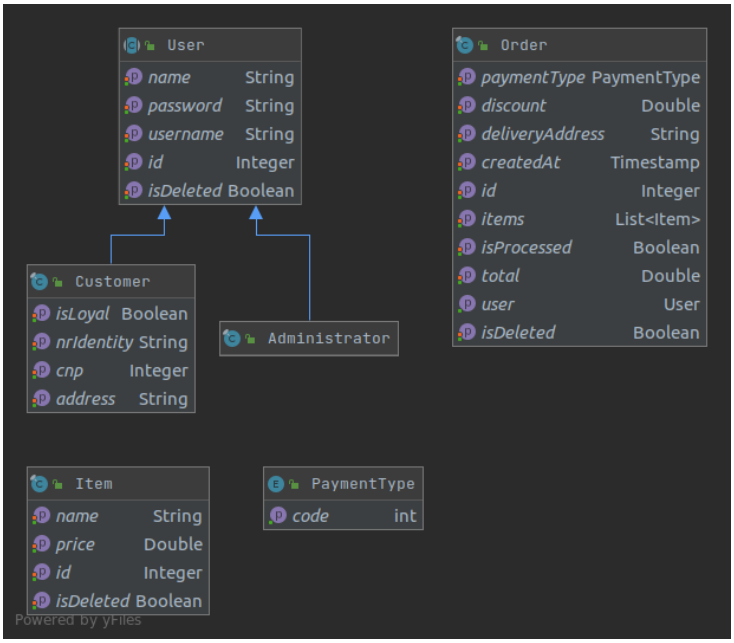
The SQL class is responsible for creating the SQL queries, for a given reference names and with the help of a given reference values.

The Table Models take an important role in the UI, they contain all the data that is displayed in the front-end. Upon action from the end-user, they are responsible for setting in motion the whole back-end flow of the application, by reaching out to the BLL-s.

6 DATA MODEL

The application consists of three main models. The User, the Order and the Item. We can also take as a model the Payment Type, which is an Enum used for the Order, to differentiate the payment type for a user.



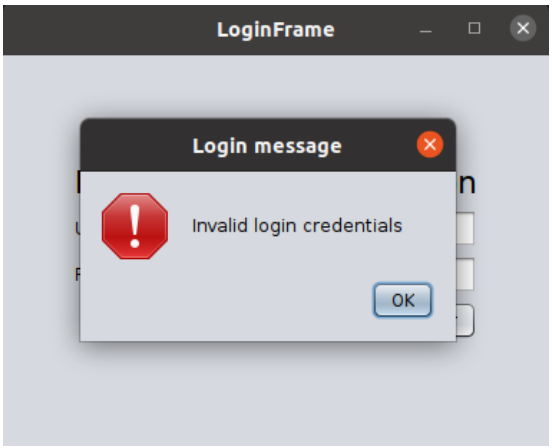


The User class is an abstract class, and a super class for the Customer and Administrator. It helps us to use the same type of object but with different behavior. The Customer contains basic information, like the identity number, the personal numeric code and the address of the given user.

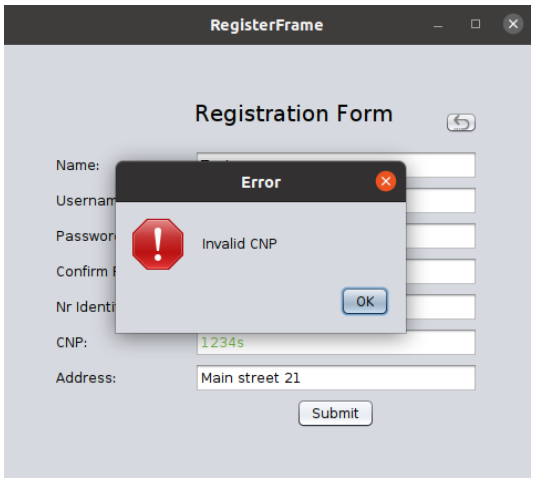
The Data Transfer Objects are in a different package, but still act as models. They model the data from the Database, while the other type of objects model the data from the application. A good use for this can be seen at the UserDTO, which contains a Boolean *is_admin*. It differentiates the Customer from the Administrator. Upon data fetch from the DB, the UserService knows which object should be instantiated.

7 SYSTEM TESTING

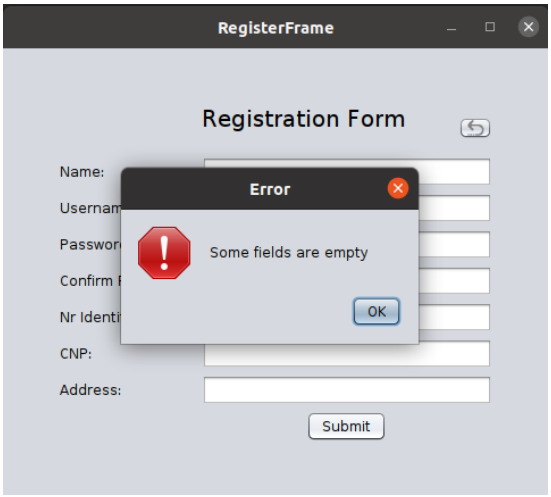
For testing the System, I went with the manual testing approach, doing the task and verifying if the result is the expected one.



4 Checking if credentials are okay

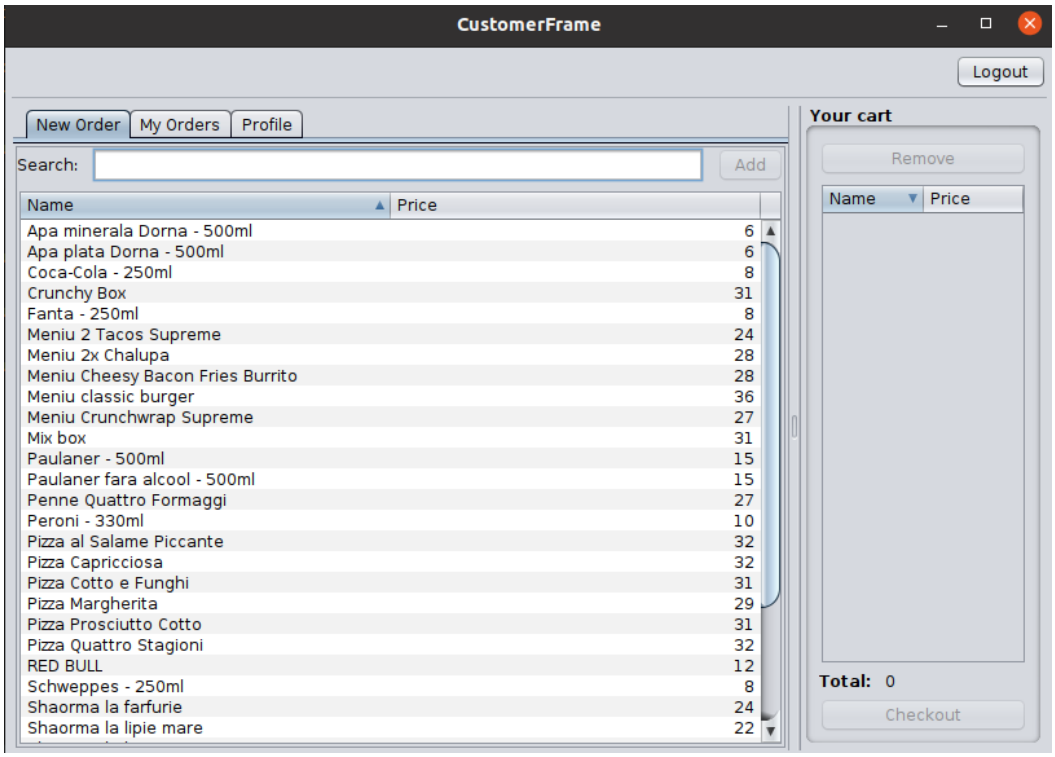


3 Checking if CNP is entered correctly

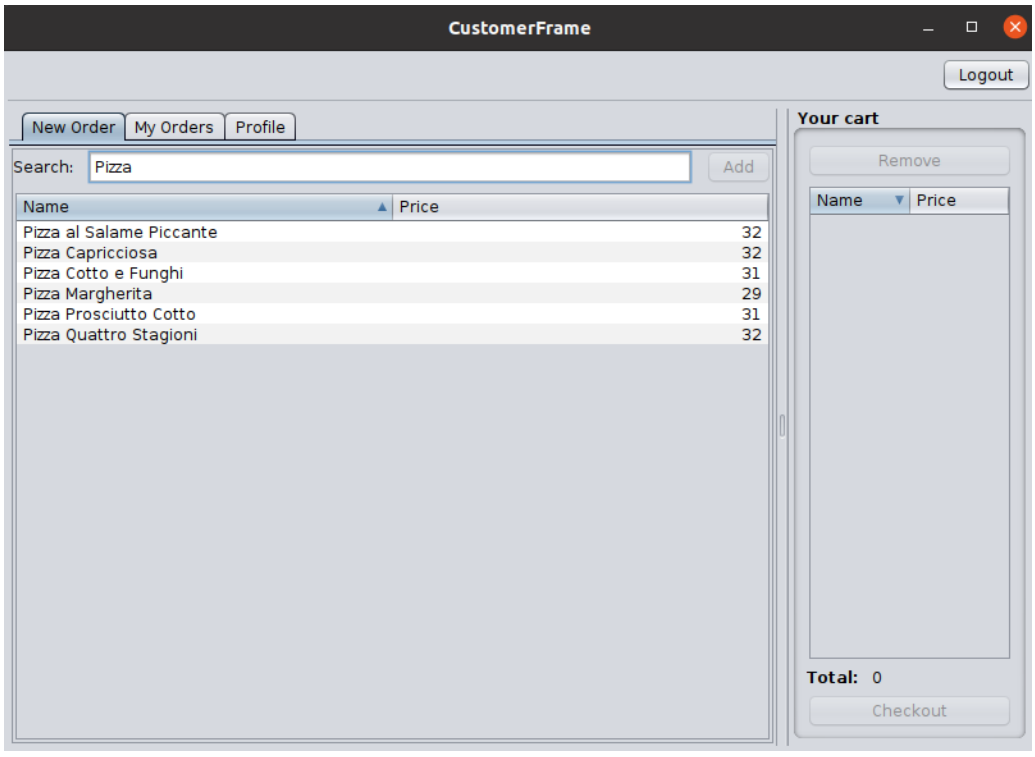


2Checking if every field is filled

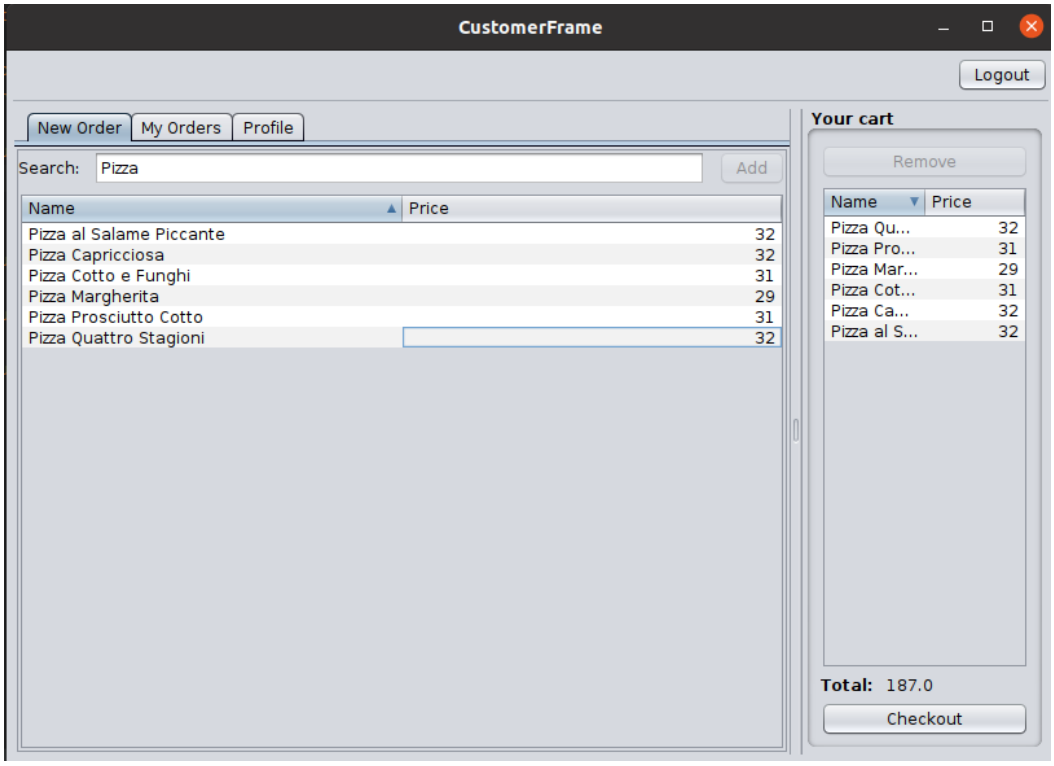
1 Checking if the passwords match



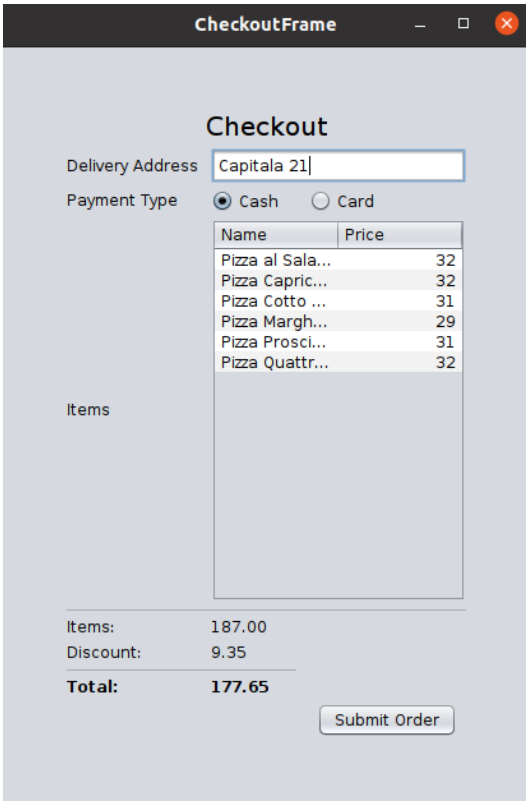
5 Checking if the customer dashboard starts up



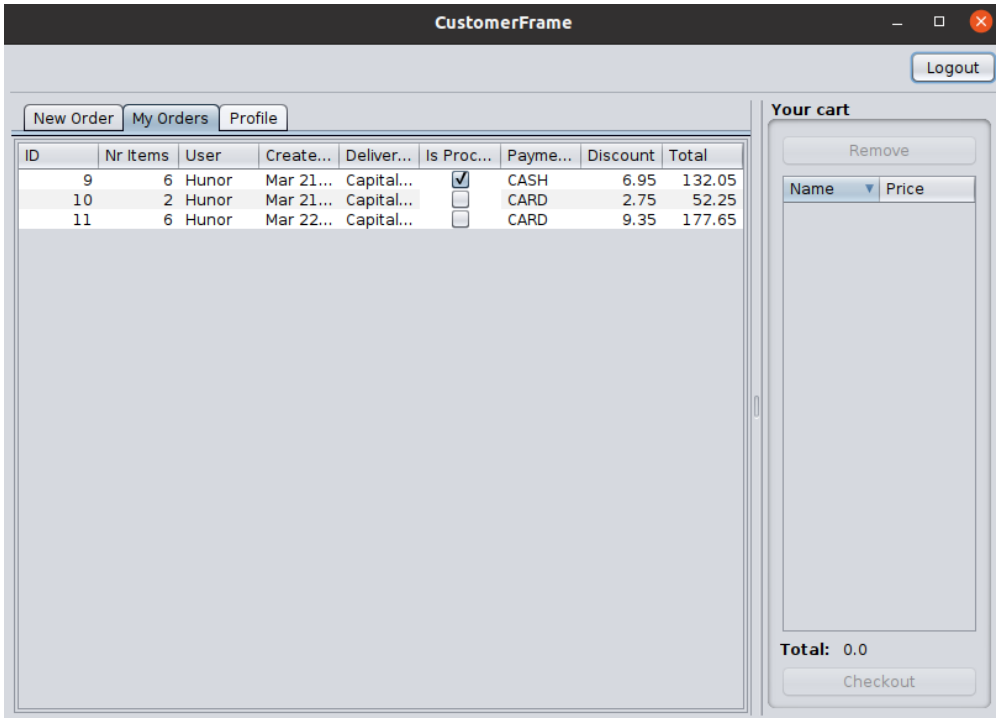
6 Checking if the filter works correctly



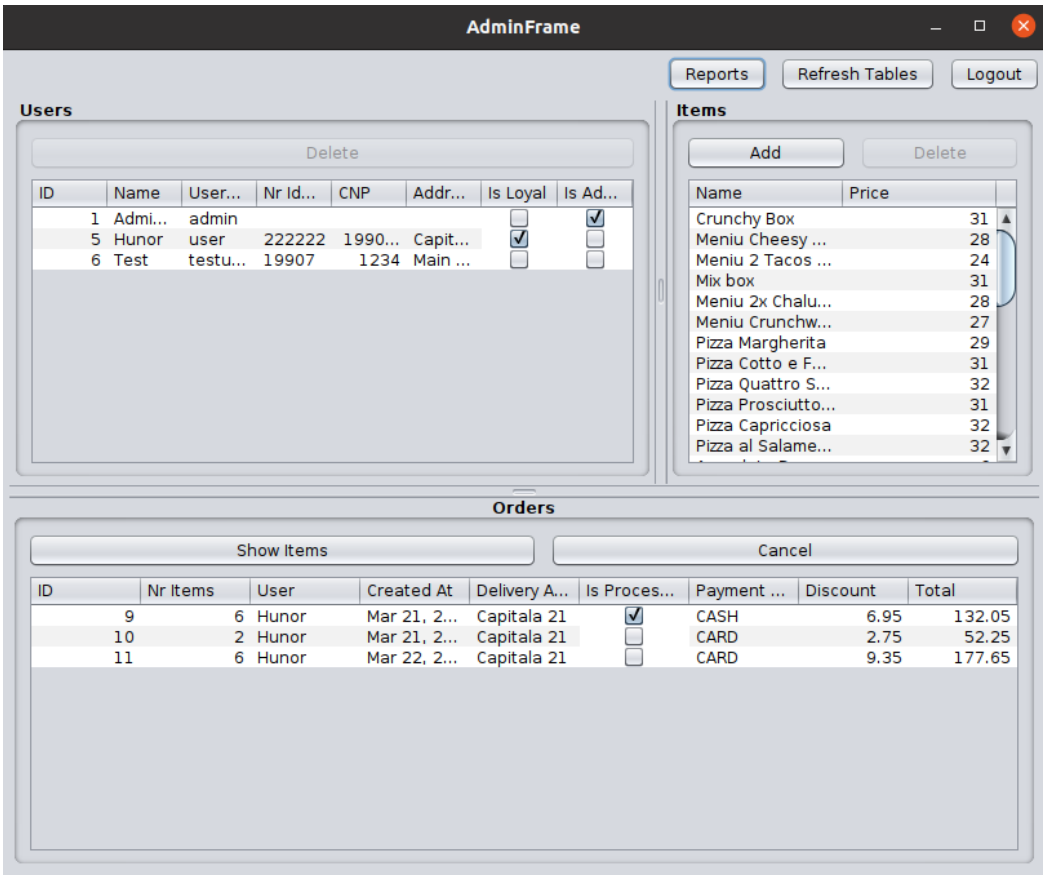
7 Checking if the item add works correctly



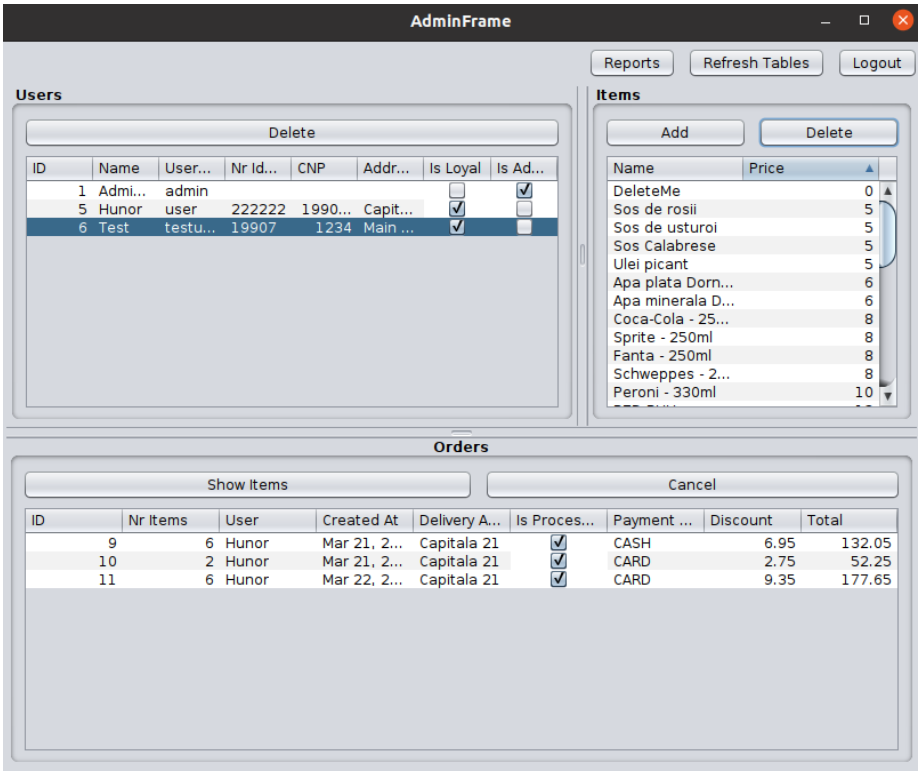
8 Checking if the cart contains the elements and the values are correctly calculated



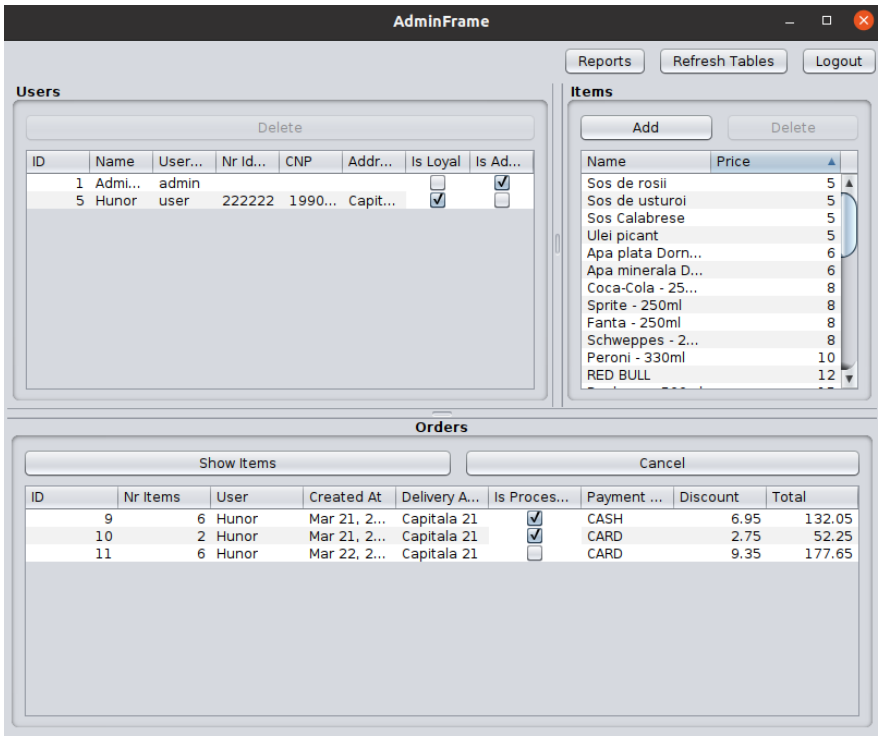
9 Checking if the cart is empty after successful order and the past orders tab contains the new order (6 items, Mar 22.)



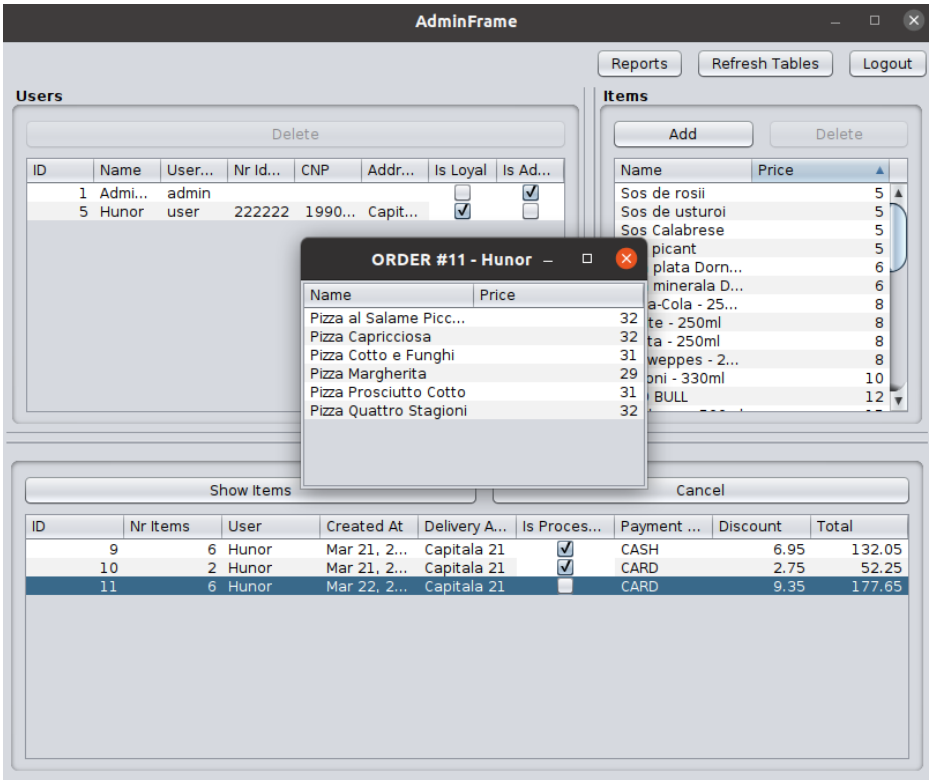
10 Checking if the admin dasbhoard starts up correctly and the data is loaded



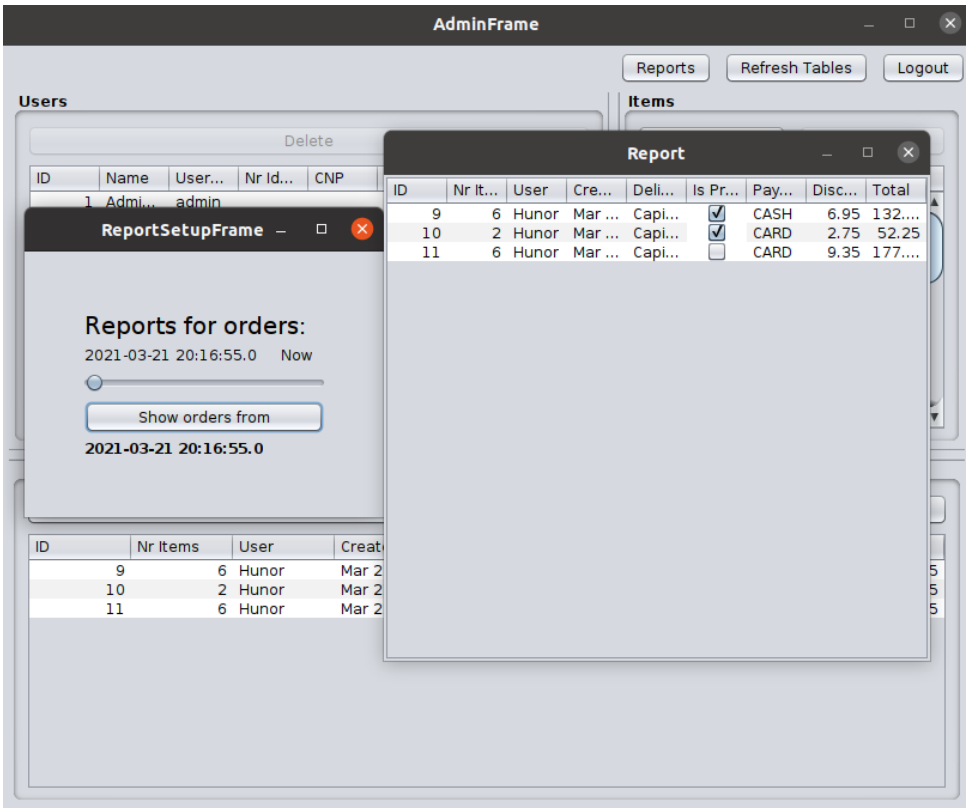
11 Checking if updates can be made (Test user set as loyal and orders set as processed) and Item add is working (Added DeleteMe item) and also sorting



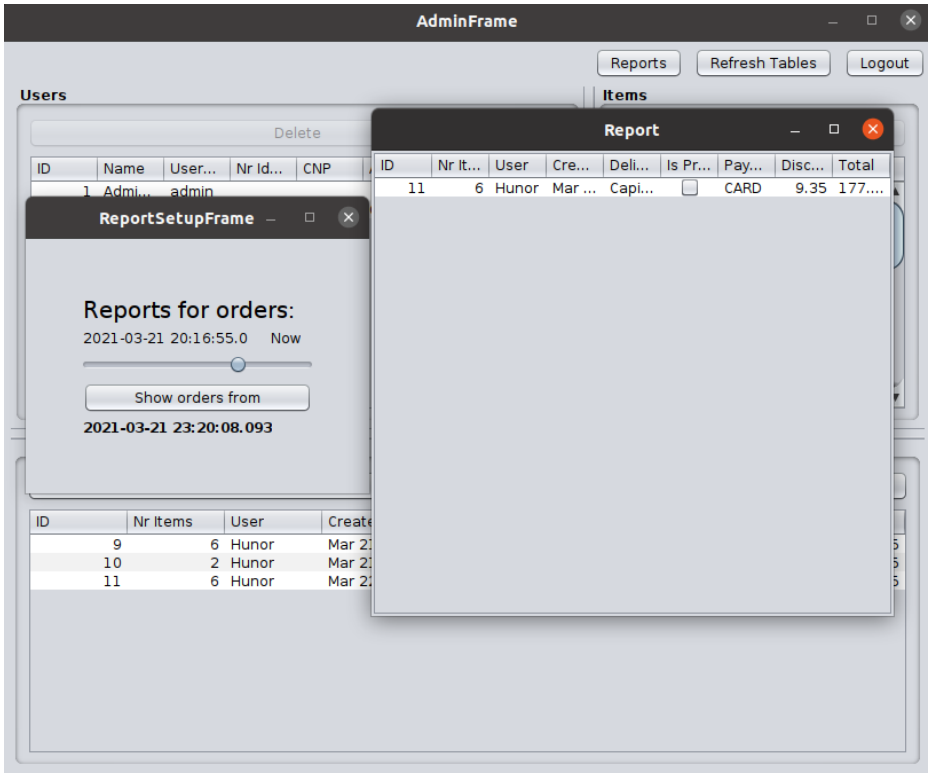
12 Check if delete is working (User and Item)



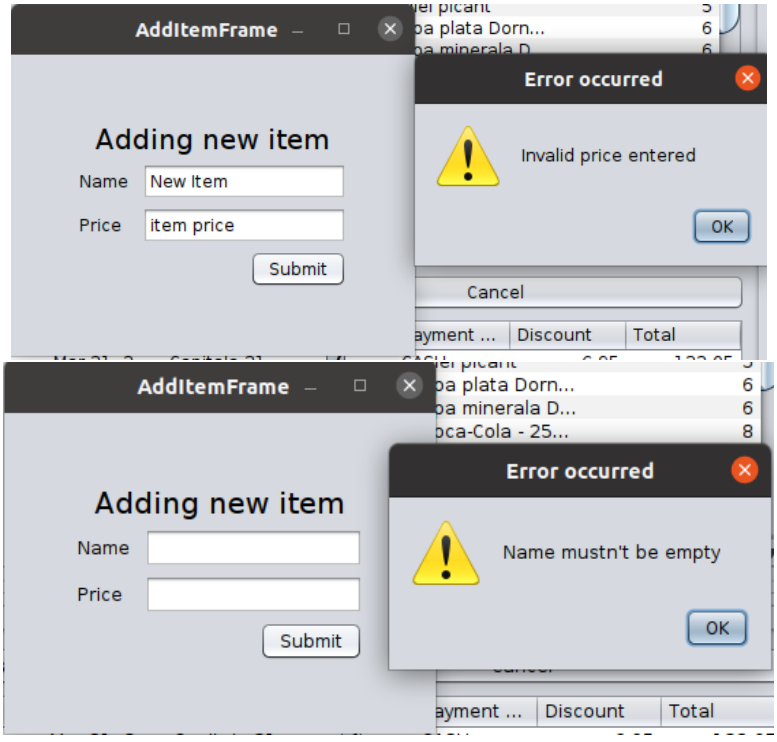
13 Checking if item list is working for orders



14 Checking if Order report works



15 Checking if order report works with another interval



16 Checking if Item validation works

8 BIBLIOGRAPHY

[1]. Layered Architecture Is Good