Sapienza Delivery

Alberto Cotumaccio 1852040
Giovanni Montobbio 1845035
Vincenzo Romito 2002967

July 2022 - Advanced Software Engineering

1 Introduction

The aim of our project was to create a food delivery system using the WebRatio platform. The users who can interact with the system are tree: Customer, Restaurant and Rider. Even administrators who are responsible for performing system operations.

- **Customers**: They can search for restaurants currently open in the same city to place orders, by selecting one of them and adding the items chosen from the menu. For this to be successful, the customer must have enough money in his account, which he can top up at any time. The total price of the order is given by the sum of the items chosen with an additional tax, defined by the restaurant in question. If the order is processed correctly then it can be accepted by the restaurant, and in that case it is delivered to the customer by an available rider. Finally, he has the opportunity to give feedback on the services offered by the restaurant.
- **Restaurants**: Each one has a menu that can be updated with new items and prices. They receive the orders placed by the customers of the system

and will have the possibility to decide whether to reject them for some reason, or accept them and consequently wait for a rider to bring it at the customer's home.

- **Riders**: have the possibility to keep track of orders that have been accepted by restaurants in their city, so that they can be taken in charge (one at a time) and delivered to the customer's home. Each of them can define slots in which to work, consequently a rider can only take an order if he is in working hours.

2 Conceptual analysis

2.1 System requirements

1. Authenticated user:
   - 1.1 Username
   - 1.2 Email
   - 1.3 Password (hash)
   - 1.4 Phone number
   - 1.5 City
   - 1.6 Balance. Depending on the type of user there is a different logic, only customers can update it
   - 1.7 The type of user, it can be CUSTOMER or RIDER or RESTAURANT (disjunction and completeness)
2. Customer:
   - 2.1 Name
   - 2.2 Surname
   - 2.3 Delivery address
   - 2.4 History of Orders
3. Restaurant:
   - 3.1 Address
   - 3.2 Logo
   - 3.3 Bio
   - 3.4 Rating (given by mean of feedbacks of orders by customers)
   - 3.5 Name
   - 3.6 Delivery fee
   - 3.7 Delivery time
   - 3.8 Slots of work for each day of the week
   - 3.9 Cuisine type
   - 3.10 Minimum spend
   - 3.11 The menu offered to customers, it consists of a set of items. Of each one interests:
     - 3.11.1 name
     - 3.11.2 description
     - 3.11.3 category
     - 3.11.4 price

4. Rider:
    - 4.1 Name
    - 4.2 Surname
    - 4.3 Mean of transport (one among Bike and Car)
    - 4.4 Rating (given by feedback of orders by customers)
    - 4.5 Slots of each day of work when he can deliver
    - 4.6 Orders delivered: able to deliver only one order at the time.
5. Order:
    - 5.1 The total price
    - 5.2 The set of items contained, all of the same restaurant. Of each item we are interested in:
        - 5.2.1 the quantity of it in the order
    - 5.3 an order can be accepted or rejected by the restaurant to which it refers:
        - 5.3.1 It is accepted when the restaurant accepts an order placed by a customer, at this time the rider does not yet exist. Only this type of order can be taken over by the riders, with the aim of making the delivery to the customer. This last type of order may have additional information such as restaurant feedback, provided by the customer who created the order.

2.2 ER domain model

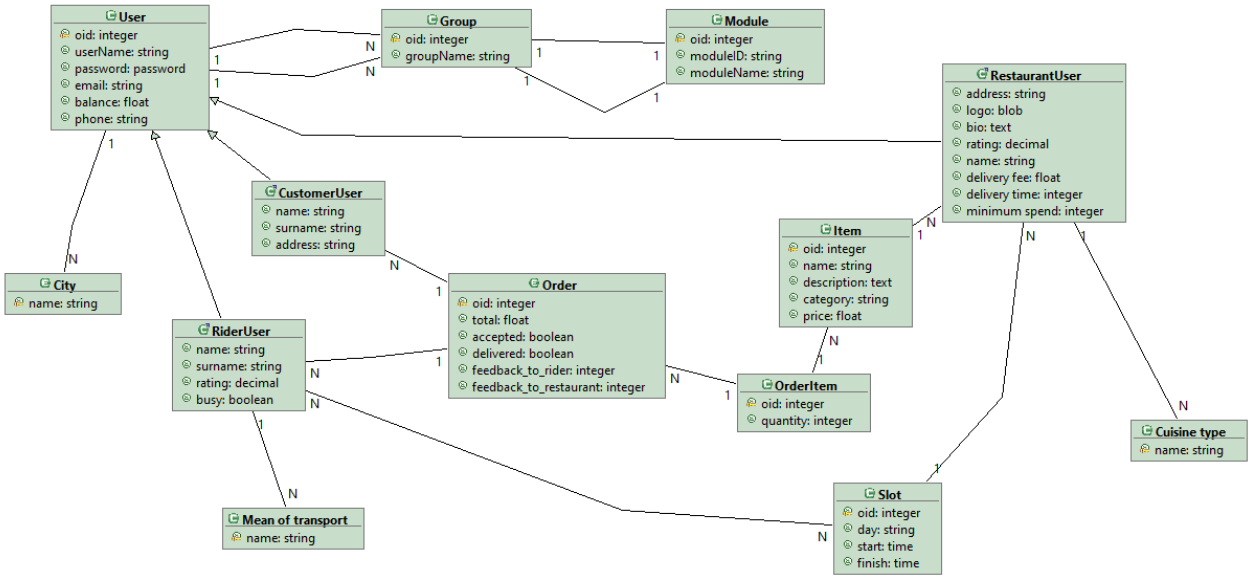


Figure 1: Domain model of our application.

Some data needs to be formatted. For example, the email and telephone of users must respect particular formatting, and are not free strings.

The cities, the types of cuisine and the means of transport must be verified, for this reason the system administrators will define the types.

We must also verify the truthfulness of some external constraints, which express the logic of our application, but which they cannot be defined in the diagram:

- all the items chosen by a customer within an order must all belong to the same restaurant.
- A customer cannot place orders for restaurants in a city different than his/her own. The same goes for the riders, but at the same time they can't be busy with another order.
- An order that has been rejected by the restaurant cannot have feedbacks.
- The total price of a certain order can never be less than the minimum spend of the restaurant to which it refers.

2.3 UML diagram of Use-Cases

Here we model a UML diagram of the Use-Cases with the operations that each actor in the system can do

- unauthenticated users who have the possibility to register by choosing an account type and entering the information requested for the latter.
- Authenticated users: everyone can perform authentication operations, such as login and logout from the application. In particular, the functionalities of these are divided as follows:
  - Restaurants can manage and modify their information on the daily opening slots and on the menu offered to customers. They are also able to carry out operations on pending orders received, accepting or rejecting them.
  - Customers can top up their balance sheets, place orders, and give feedback on orders that have arrived home.
  - The riders are in charge of taking pending orders and taking them to the home of the customer creator of the order.

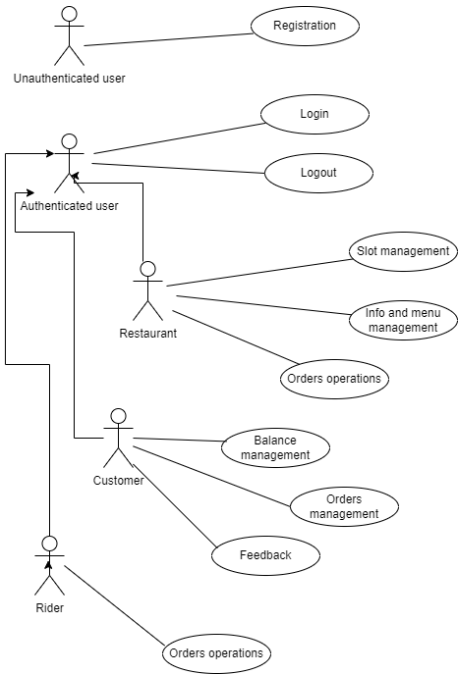


Figure 2: Use case diagram.

3 Technologies and choice of the DBMS

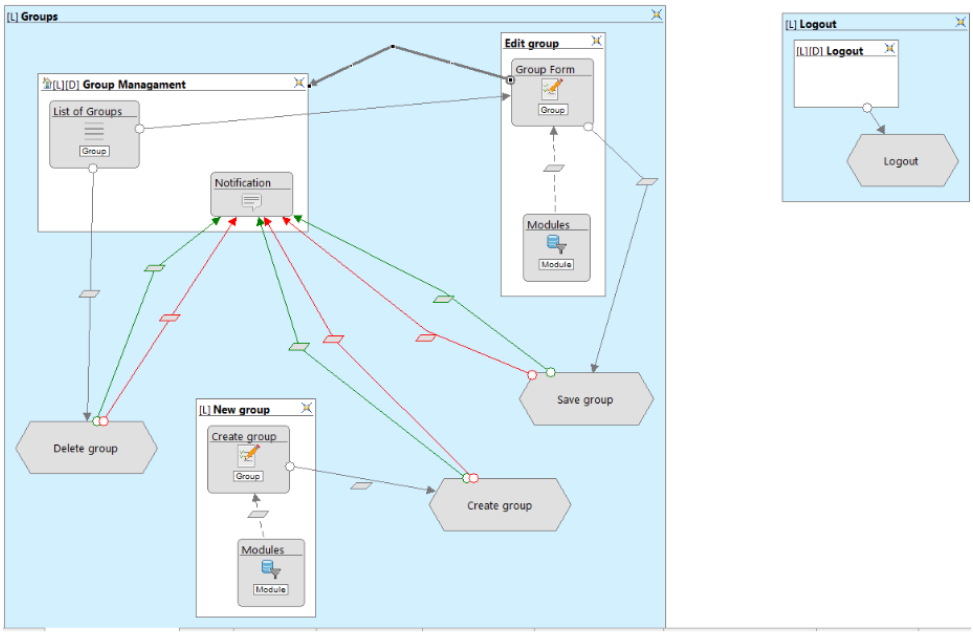
As for the technologies used, we worked with the WebRatio platform, and in particular the version 7.2.18 that we downloaded thanks to the plan offered in the area reserved for students. We connected the project to a MySQL database, a very flexible DBMS suitable for our type of project. In particular we use it and make it work thanks to the XAMPP control panel.

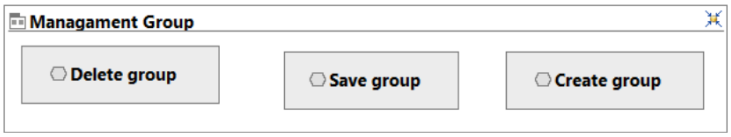
4 Application Design

In this phase we are going over the main views and modules we developed. Some modules are self-explanatory and quite simple, their implementation will not be shown in this report.

4.1 Administration view and operations

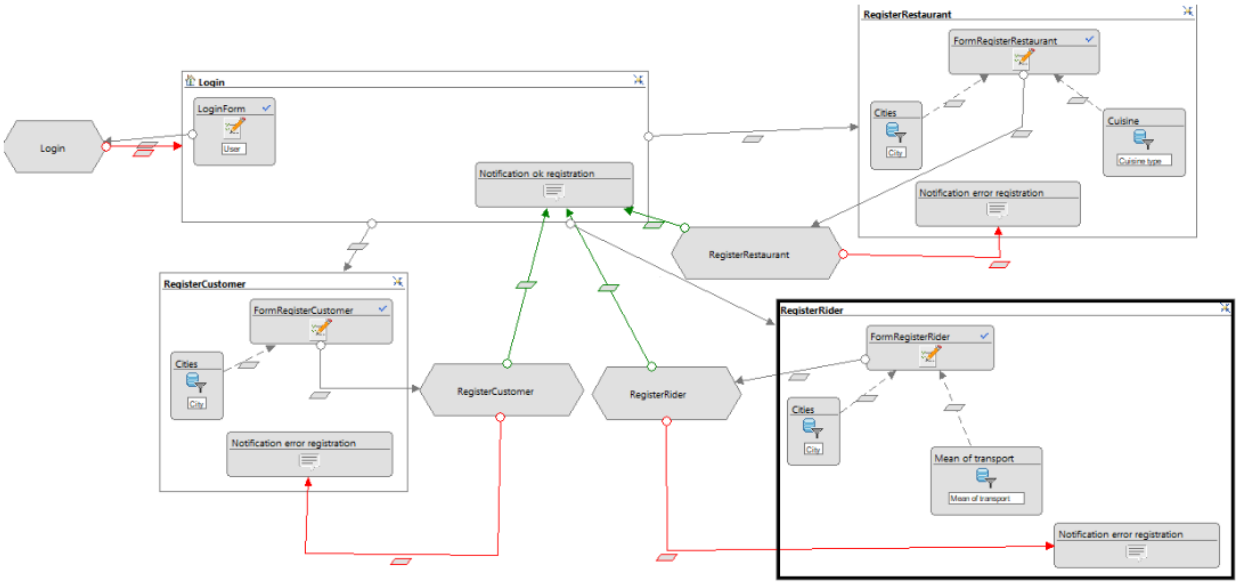
The page used to manage groups will allow the administrator to create a new group, to modify or delete an existing group, and to connect groups to protected modules. In this case, a connection between a group and a module indicates that all the users belonging to that specific group have access to that specific module. In our application we have three types of users and therefore three groups: **customers**, **riders**, **restaurants**. Each group will have a dedicated module.

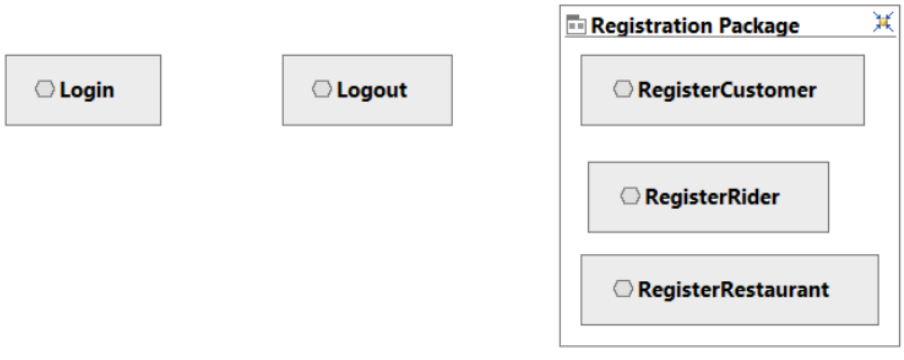




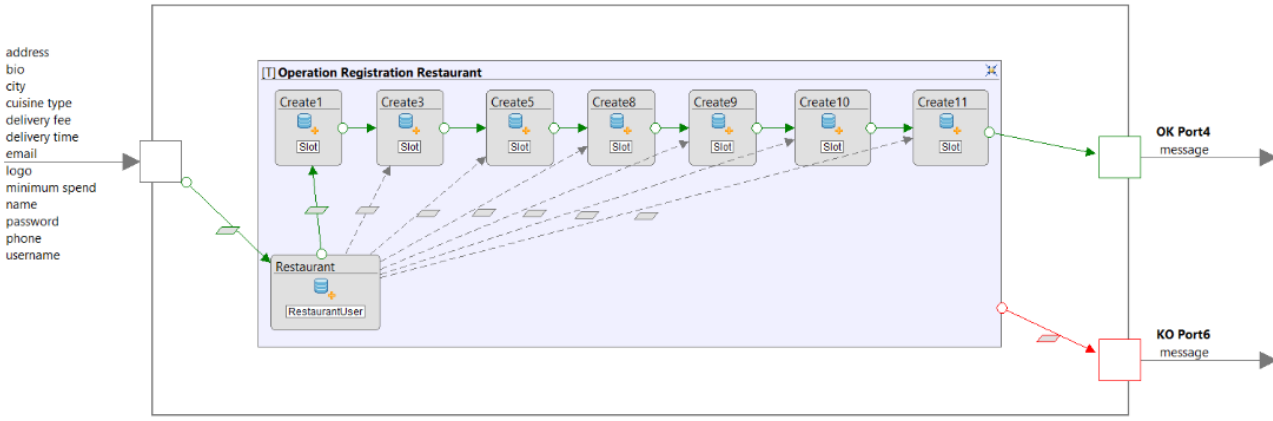
4.2 Authenticated user view and operations

Here we will give the opportunity to the new user to choose the registration that suits him the best by choosing a registration as a **rider** if he wants to deliver orders and getting paid for it, **restaurant** if he wants to register the restaurant in our system and starting delivering food, or **customer** if he would like to order some food from one of our partner restaurants that will be delivered by our fleet of riders. Once registered within a certain role, the user is now capable to **login** into our system and it will be automatically redirected to the specific group landing page.





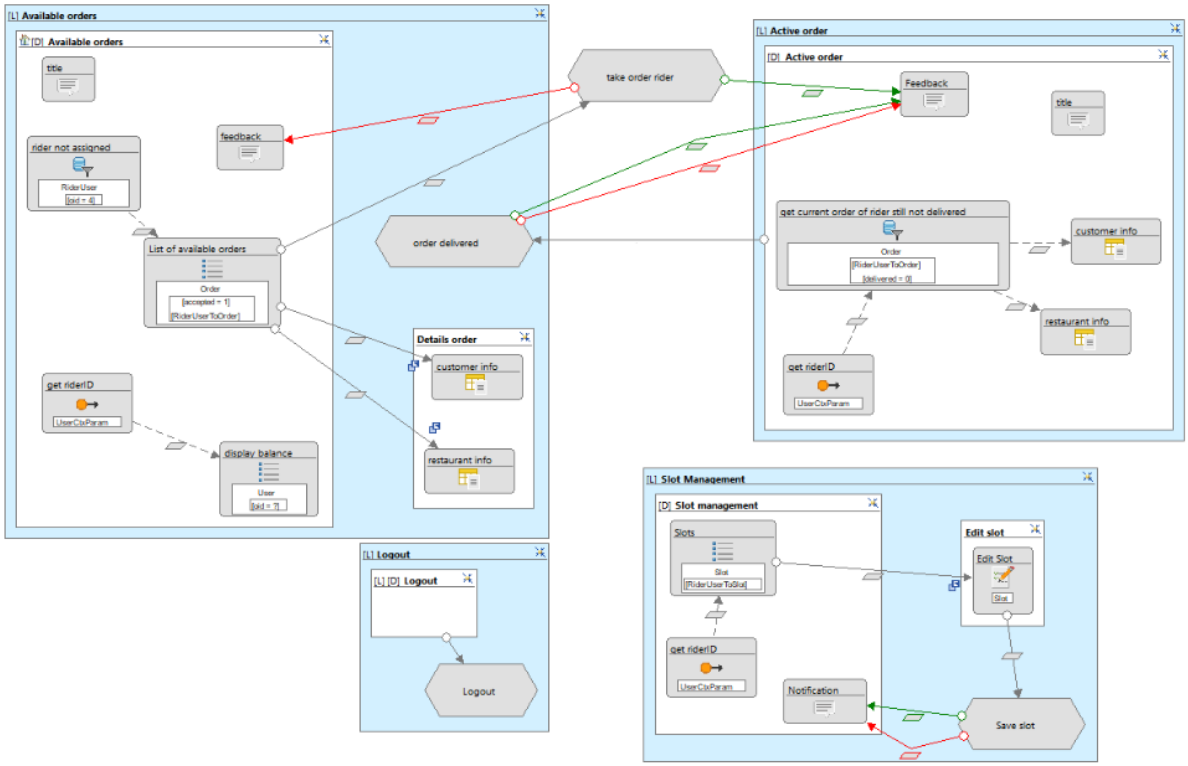
Regarding restaurant and rider user registration we are also creating some working time slots that these users will fill later in the registration process. Take for example the restaurant registration module.

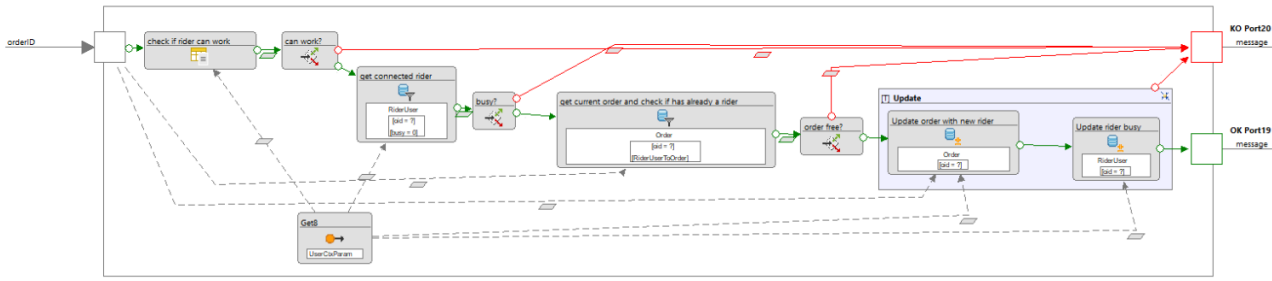


During the registration step, each module will take care of assigning the registering user to his/her desired group.

4.3 Rider view and operations

First thing a rider should do here once logged is to fill up the working schedule hours using the **slot management** tab we designed. Once he/her has done so he/her can start looking at **orders** that are ready to be shipped that don't have an assigned rider just yet. The rider can now have an overview of their taken order in the **active order** tab, looking at the restaurant and customer info including the pick up and delivery address. Once the order has been delivered, the system will transfer 10percent of the total order to the rider.





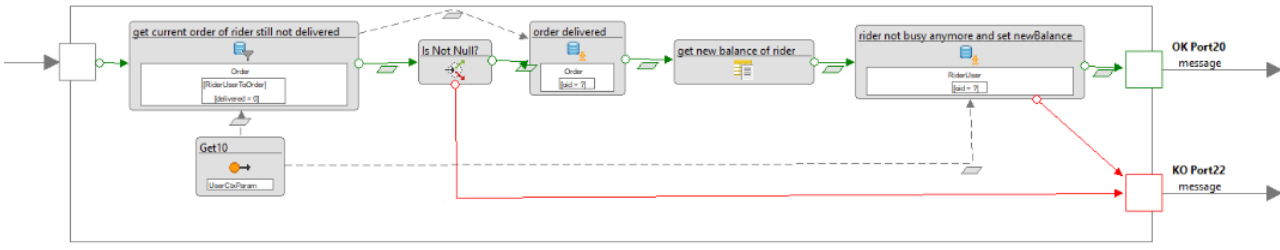
The take order action implements the logic of a rider accepting a ready order. What we do here is to check:

- if the rider is respecting his/her working hours schedule

```
SELECT r.user_oid
FROM   rideruser r,
       slot s,
       rideruser_slot rs
WHERE  rs.rideruser_oid = r.user_oid
       AND r.user_oid = :rider
       AND rs.slot_oid = s.oid
       AND s.day = Dayname(CURRENT_DATE)
       AND s.start <= CURRENT_TIME
       AND s.finish >= CURRENT_TIME;
```

- if the rider is not busy with another delivery (can't take multiple orders at once)
- if the order that they are trying to take on has already been accepted by another rider but it still shows because the logged rider did not refresh the page

If all the above applies, we allow the rider to take the order.



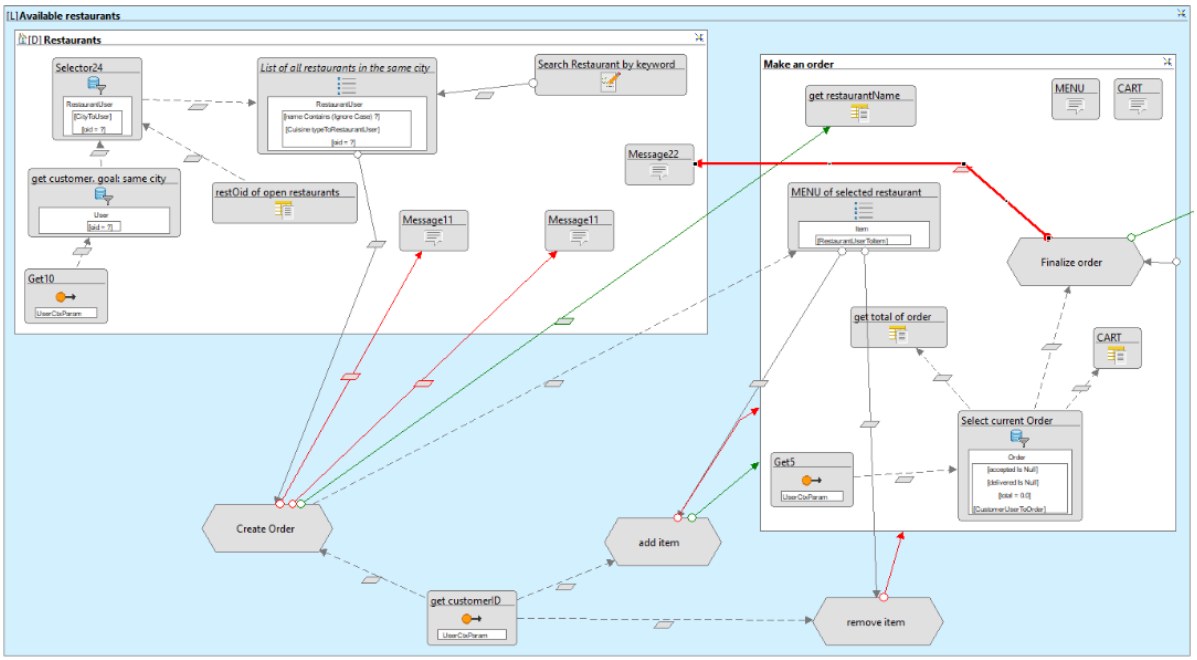
The deliver order action implements the logic of a rider delivering the order to the customer and getting paid for it by out application. What we do

here is to make sure that the current order of the rider has not been delivered yet, if so, we allow the rider to deliver the order and we add to his balance 10percent of the total of the order.

```
SELECT u.balance + ( o.total / 10 )
FROM   sapienza_delivery.order o,
       sapienza_delivery.USER u,
       rideruser ru
WHERE  o.rideruser_oid = ru.user_oid
       AND ru.user_oid = u.oid
       AND o.oid = :orderID;
```

4.4 Customer view and operations

As soon as the customers logs in they are presented with a list of **available restaurants** in the same city that are open in that specific moment (query similar to the one used for riders previously). For each restaurant, the customers will be presented with a series of features that will help them with the choice. These are: name of the restaurant, **rating** based on feedback by previous customers, delivery fee, delivery time, minimum amount to spend, cuisine type etc. More importantly we provide to the customer the option to **search** a restaurant by name and by cuisine type.



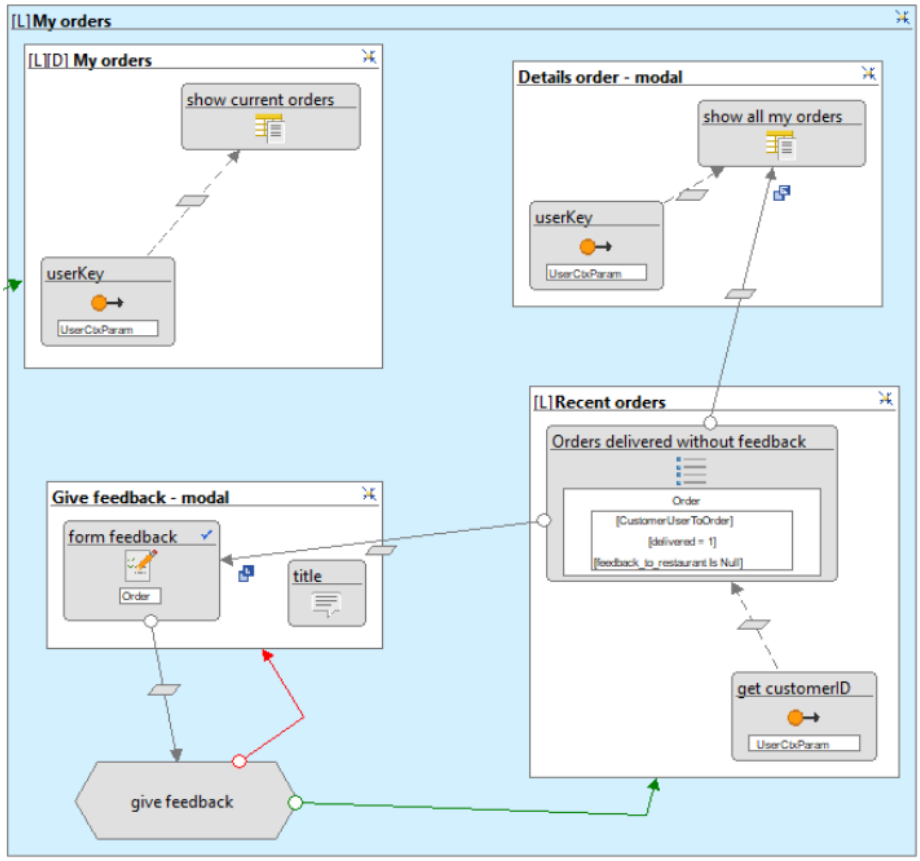
Once selected the desired restaurant, the customer can now examine the **menu** with all the items and their costs. The customer can add/remove items from the chart by simply clicking " + " or " - " next to each item. The chart is visible on the right side of the screen, reporting the chosen items, their quantity, their partial price

```
SELECT item.NAME            AS NAME,
       quantity             AS quantity,
       ( quantity * price ) AS price
FROM   orderitem,
       item
WHERE  order_oid = :ordine
       AND orderitem.item_oid = item.oid
```

and the total, taking into consideration also the delivery fee.

```
SELECT Sum(quantity*price)
       + restaurantuser.delivery_fee AS total
FROM   orderitem,
       item,
       restaurantuser
WHERE  orderitem.order_oid = :ordine
       AND orderitem.item_oid = item.oid
       AND item.restaurantuser_oid = restaurantuser.user_oid
```

If the customer is not happy with the order can decide to leave the page and start a new one, from the same, or from another restaurant.

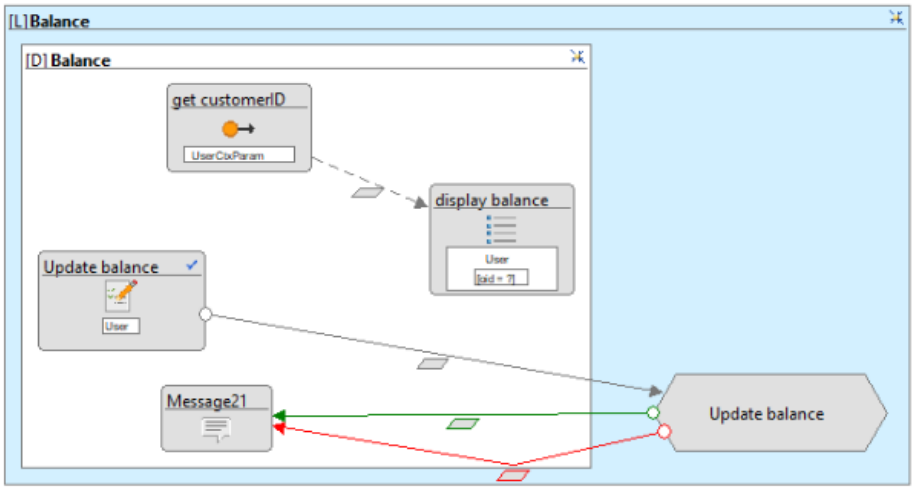


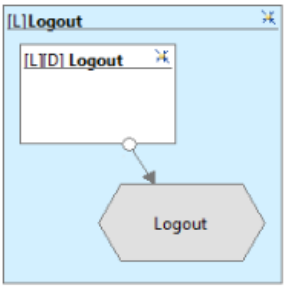
However if the customers decide to proceed with the **order** and to finalize it, through the section "my orders" they will be able to check on the status of the order. The page will show some basic information about the order (restaurant name, total, phone number) and some more advanced insights such as:

- if the order has been accepted, pending, or refused by the restaurant
- once accepted, if a rider has been assigned or not
- delivery time

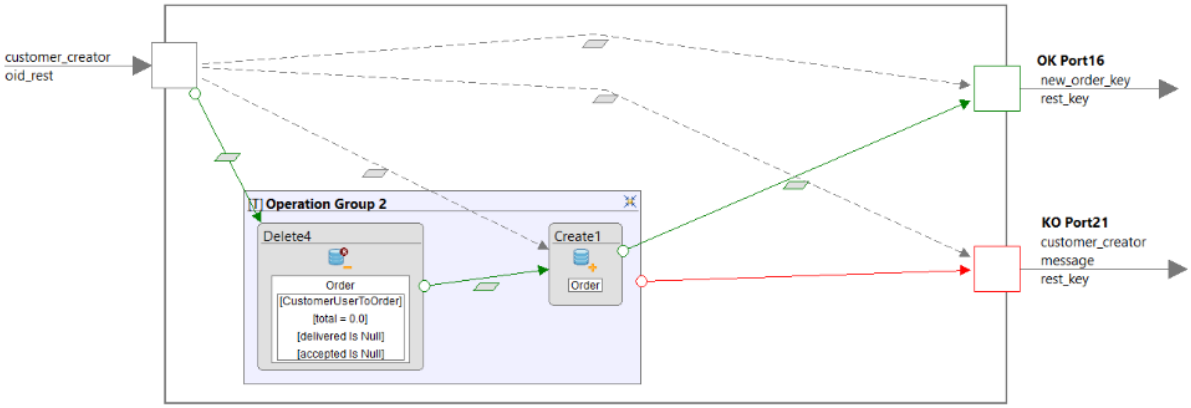
```
SELECT DISTINCT o.oid,
                r.NAME,
                o.total,
                o.accepted,
                r.delivery_time,
                rider.surname,
                ur.phone
FROM   restaurantuser r,
       sapienza_delivery.order o,
       item i,
       orderitem oi,
       rideruser rider,
       sapienza_delivery.USER u,
       sapienza_delivery.USER ur
WHERE  o.customeruser_oid = :userKey
       AND r.user_oid = i.restaurantuser_oid
       AND oi.item_oid = i.oid
       AND oi.order_oid = o.oid
       AND o.rideruser_oid = rider.user_oid
       AND rider.user_oid = u.oid
       AND r.user_oid = ur.oid
       AND u.oid <> ur.oid
       AND ( o.delivered <> 1
             OR o.delivered IS NULL )
```

To finalize an order, the customers need to have money on their **balance**. To do so they access the balance page. Here they can add funds to their account and check the current balance.

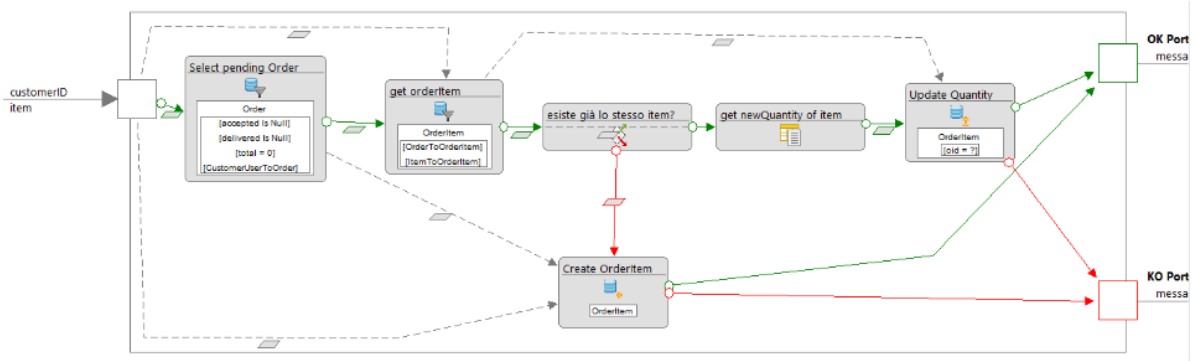




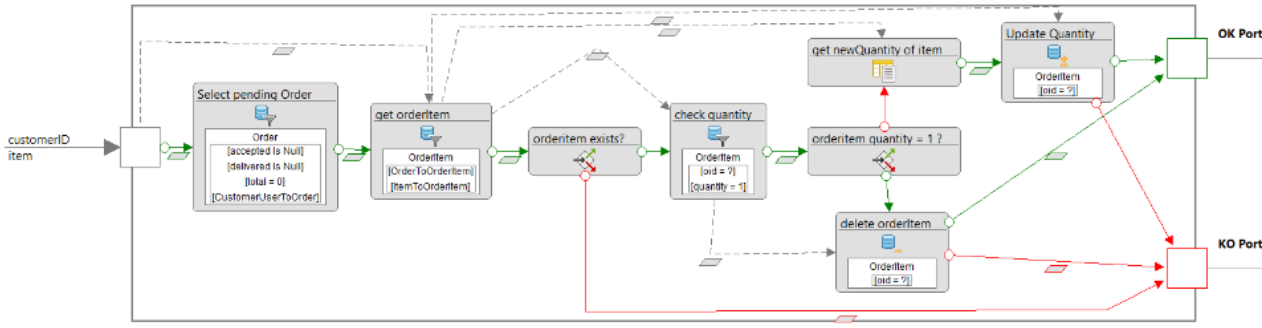
Once the order has been delivered to the customer, the lifespan of the order can be considered almost finished. Now the customer can access the **recent orders** to give feedback (1-5) to the restaurant. This will be averaged with the feedback of other users to compose the restaurant rating.



The **create order** happens as soon as the customer clicks on a restaurant in order to start to add items in their basket. Any existing order process that has not been finalized yet will be removed to make space for the current purchase of the customer.

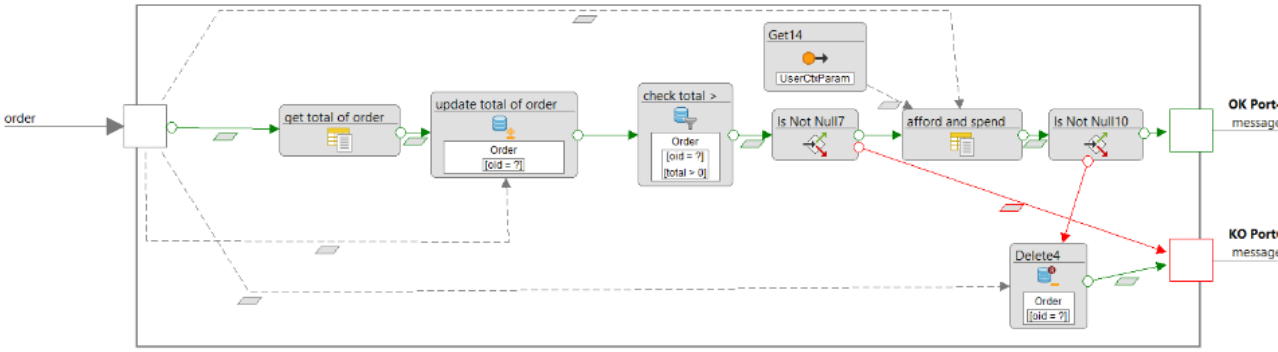


The **add item** module is triggered when the customer tries to increase the quantity of a certain item into their purchase. What the system does here is to check whether that specific item is already present in the order or is its first unit. In the first case it will increase the quantity of the item by one, in the latter it will create a new item in the chart.



The **remove item** module is triggered when the customer tries to decrease the quantity of a certain item into their purchase. What the system does is to check whether the item that the customer is trying to decrease the quantity already exists in the order. Assuming the previous statement is true:

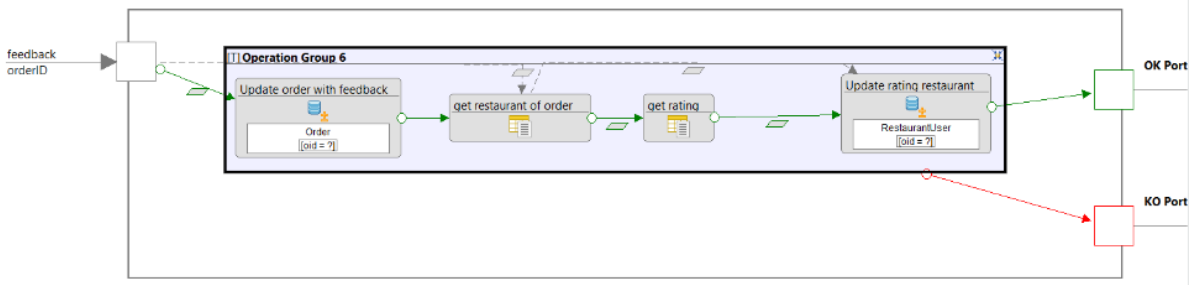
- if the quantity of the item is 1, the item is removed from the order
- if the quantity of the item is 2 or more, the quantity of the item is reduced by 1 quantity



To **finalize the order** the total is computed taking into consideration the number of items, their cost per unit and the delivery fee. The final total of the order can now be updated.

```
SELECT Sum(quantity*price)
       + restaurantuser.delivery_fee AS total
FROM   orderitem,
       item,
       restaurantuser
WHERE  orderitem.order_oid = :ordine
       AND orderitem.item_oid = item.oid
       AND item.restaurantuser_oid = restaurantuser.user_oid
```

The system now checks if the user can afford to finalize the order and if it reached the minimum spend required by the restaurant.

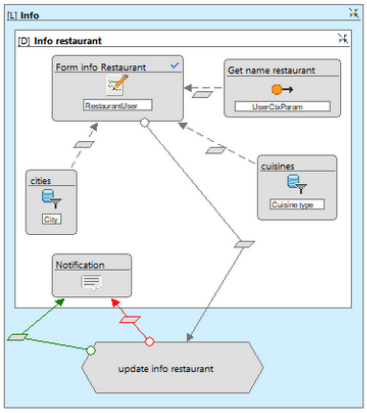


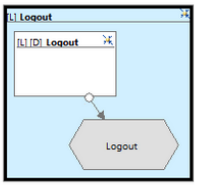
Once the customer gives a feedback to the order, the rating of the restaurant has to be updated taking into consideration all the previous feedback and this last one.

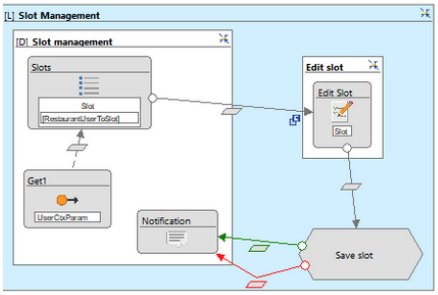
```
SELECT Avg(o.feedback_to_restaurant)
FROM   sapienza_delivery.order o,
       orderitem oi,
       item i,
       restaurantuser r
WHERE  i.restaurantuser_oid = :restaurantID
       AND oi.item_oid = i.oid
       AND oi.order_oid = o.oid;
```

4.5 Restaurant view and operations

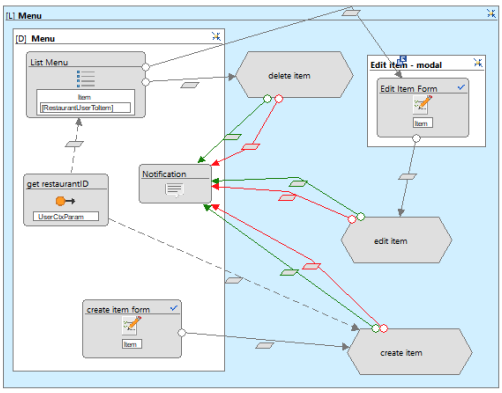
First thing a restaurant should do here once logged is to fill up the working schedule hours using the **slot management** tab we designed. This is important because our system keeps track of opening hours of restaurants, if the restaurant does not insert opening and closing hours, it won't be taken into consideration because it will always be considered as closed.



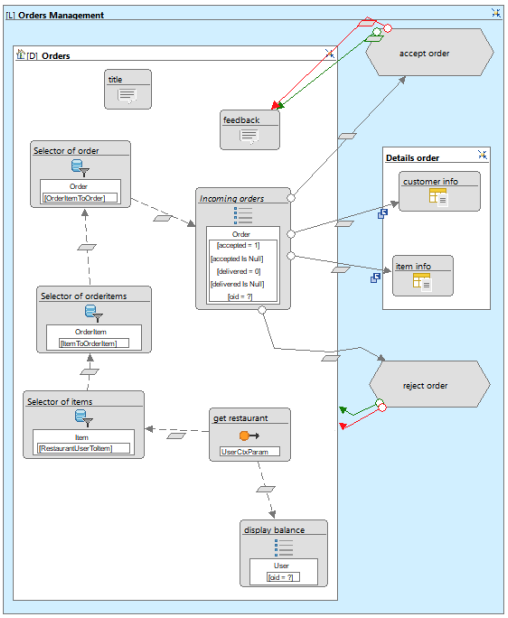




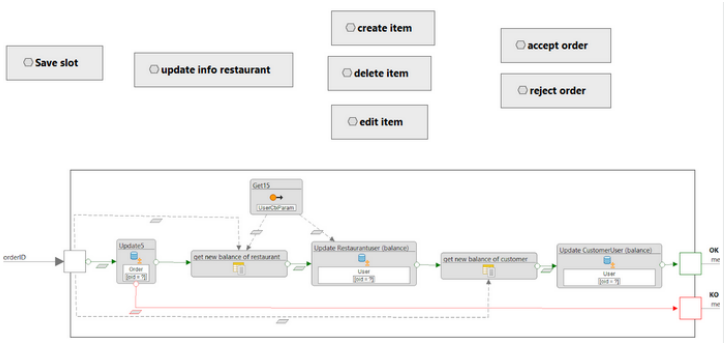
At this point it can keep the registration process going adding more and more **information** like cuisine type, city, logo, bio, delivery fee, minimum spend etc.



Another vital part of the configuration process for a restaurant is the one of adding **products** and **prices** to its menu. An item can be added, edited and deleted. An item correctly added will be shown to the customers when they will try to order from your restaurant.



On the **order management** page the restaurant can keep track of the incoming orders. They can examine information about the customer such as their location and info about the order itself, for example which products have been ordered and in which quantity, for what total price etc. Restaurants have the option to accept or refuse the order. If accepted, the order can be checked all the time by the restaurant on the same page, until it has been delivered by the rider.



Once the restaurant accept the order it is time to trigger the money transfer from the customer to the restaurant.

```
SELECT o.total + u.balance
FROM   sapienza_delivery.order o,
       sapienza_delivery.USER u
WHERE  o.oid = :orderID
       AND u.oid = :restID;
```

```
SELECT u.balance - o.total,
       u.oid
FROM   sapienza_delivery.order o,
       sapienza_delivery.USER u,
       customeruser cu
WHERE  o.customeruser_oid = cu.user_oid
       AND cu.user_oid = u.oid
       AND o.oid = :orderID;
```

Here we also change the status of the order to “accepted”, this makes life easier for the customer and for the restaurant since they both know the status of the order and it also makes the order eligible to be picked by a rider to be delivered.

5 Conclusion

Initially we had some difficulties developing the web project, especially the inability to create a shared repository that was a problem that slowed down the develop consistently. But despite the initial difficulties, with the ongoing progress of the work we realized the comfort of working with such a software and we managed to solve many situations very quickly.

We made available a sample video in order to show some key operations that can be done.