Self-ordering point of sales

Project report

Students:

Marwan Summakieh (285805 ICT)

Rokas Barasa (285047 ICT)

Supervisors:

N/A

Character count: 35674 characters

Software Technology Engineering

2nd Semester

2020-07-13

Table of content

List of figures and tables

Abstract (Marwan)

The project aims to create an improved ordering experience in small to medium sized restaurants by giving the customers the ability to order something by themselves without workers. Effectively making the ordering experience better for both parties by improving efficiency and minimizing interaction.

The resulting system is a generic food ordering interface that aids the customer in making and order and its customization. While the receptionist can manage the menu that the customers see and manage incoming orders. An additional user, the order screen watches order completion and making.

The system that is developed is client/server system that uses JAVA in both tiers. Connection is handled through RMI. The client uses JAVAFX for GUI and MVVM for client architecture. The server has a JDBC connection to a relational MYSQL database. Observer pattern is utilized to get up to date information to the clients.

A total of 21 out of 21 user stories have been implemented and tested using black box testing. There were 42 tests done, all of them passed

1 Introduction (Marwan)

Self-ordering systems are gaining more popularity, because of what they can put on the table for the owners.

The reason behind it is it helps eliminate many of the common mistakes and misunderstanding that can happen with direct contact between employees and customers (LAUV team, u.d.).

Additionally, many customers would like to place their orders themselves to avoid being placed on hold or dealing with busy receptionists.

There is also an monetary advantage to self-ordering systems, it boosts sales without increasing the number of employees (Prange, u.d.).

Therefore, a simple system was needed to help small businesses mange and organize their workplace. An example of that is a small restaurant that has a small number of workers. This workplace cannot afford having its employees spending too much time with customers. As a way of organizing, an ordering screen can be introduced, for customers to use where everything on it is easily managed and tracked.

For receptionists they do not have to do things manually or spend more time with physical contact with customers. As for management, it is simpler to handle all the items in the menu and track orders.

The system helps the restaurants to do all their duties accurately and more efficiently.

However, system deployment will have certain delimitations:

1. Printing of receipt for the customer.
2. Credit card payment (As with printing. Only involves back end).
3. Cash payment
4. Customer information holding or sales statistics tracking.
5. Direct communication of customer and chef.
6. System admin or store chain manager.
7. Multiple stores
8. Food delivery.
9. Employee management (Schedules, removing, editing).

10. Refunds or canceling orders.
11. Employees besides receptionists.
12. Security measures
13. Tracking of supplies, expenses, and income.

The development process is going to be done using scrum with multiple sprints each sprint holds the phases mentioned below:

- Analysis
- Design
- Implementation
- testing

2 Analysis

2.1 Requirements

The system has three actors:

- **Customer –** Actor who will use the system to make orders.
- **Receptionist –** Actor who will watch, complete orders, and manage the menu
- **Order screen –** A secondary actor for customer that will track orders in progress and done.

2.2 User stories (Rokas - Marwan)

As a **user,** I want to …

1. have a menu of different system users, so that I can choose which user I want to use.

As an **order screen user**, I want to …

2. have two separate lists for completed and incomplete orders, so that I can differentiate between them.

As a **customer**, I want to …

3. have a menu of items I can order, so that I can choose what items I want to buy.
4. have a cart to which I can add items to, so that I can buy more than one item at a time.
5. have functionality to view the cart, so that I can see what I added to my order.
6. have functionality to reset the cart, so that I can remove all items I added.
7. have functionality to remove items while in the cart, so that I can remove specific items form the order.
8. have an option to customize items I added to cart, so that I can change the item to my preference
9. be able to make order, so that my order can be fulfilled.
10. be able to pay for my order by card, so that I can have a way to pay for the goods I will receive.

11. see an id on the screen after I pay for my order, so that I know which order is mine.

As a **receptionist**, I want to …

12. login, so that there are no other users who use my receptionist account.
13. have a list that displays current menu items shown to customers, so that I can manage what I put on the menu.
14. have functionality to add an item to the menu list, so that the customer can order new items.
15. have an order list that shows incomplete orders, so that I can see what I need to do next.
16. be able to complete an order, so that the customer can see the order on the order screen.
17. have functionality to delete items from the menu, so that I can remove unwanted menu items.
18. be able to specify ingredients when creating an item, so that the customer sees ingredients in the item.
19. be able to specify an item to be customizable, so that items which have multiple ingredients can be customized by customers.
20. divide the items by specifying a group name, so that the menu is organized.
21. have functionality to print order info, so that the order can be shared among workers outside of the system.

2.3 Non-Functional Requirements (Rokas)

1. The system must be a Client-server system
2. The system must use either RMI or sockets for networking
3. The system must have multiple user types
4. The team must use Unified process and Scrum when developing the system.
5. The system must have a database.
6. The order lists in order screen user must display only ids of orders, so that orders do not clutter the list with excessive information.

7. Have a limit on how many completed orders are shown in the order screen user, so that the completed orders do not overpopulate the list.
8. There must be different colors for unfinished and finished order lists in order screen user, so that the lists can be told apart immediately.
9. Have the menu reset after the customer completes an order, so that another customer can use the menu.

2.4 Use case diagram (Rokas - Marwan)

The use case diagram shows how different actors would behave in the system and what actions they could do. (Larman, 1997)

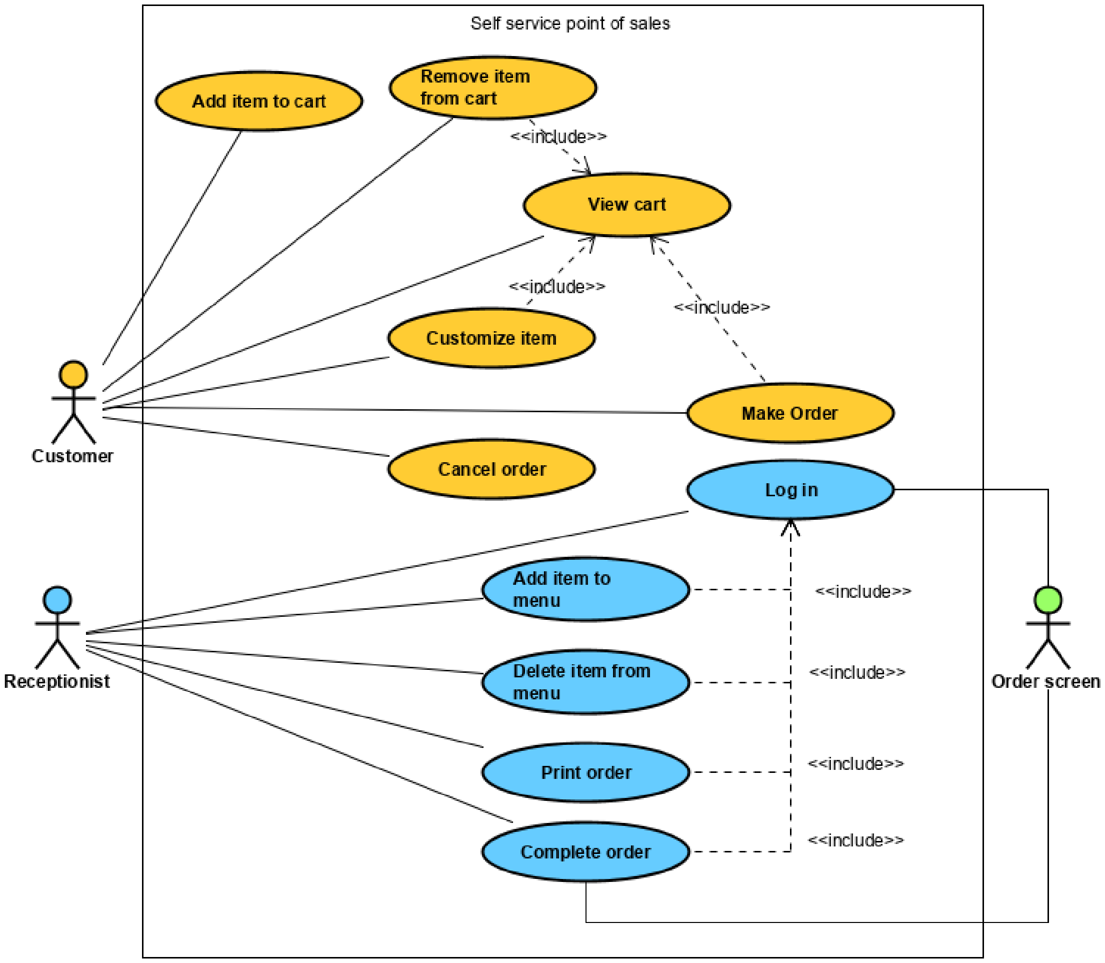
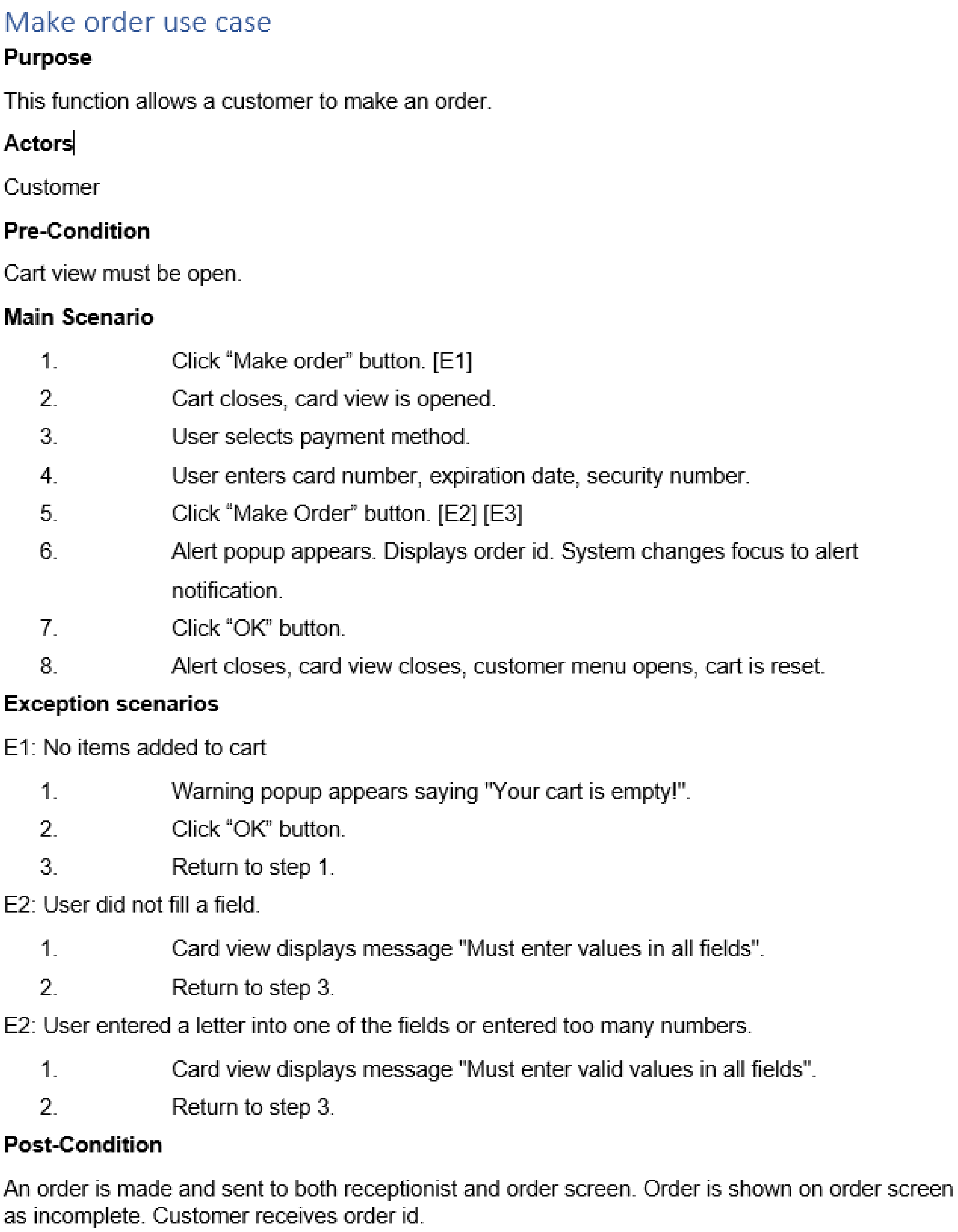


Figure 2.4-1 Use case diagram

2.5 Use case descriptions (Rokas)

Use case descriptions offer a in depth explanation of a scenario from the use case diagram. It showcases all the steps an actor must take to fulfil the use case purpose.



Make order use case

Purpose

This function allows a customer to make an order.

Actors

Customer

Pre-Condition

Cart view must be open.

Main Scenario

1. Click “Make order” button. [E1]
2. Cart closes, card view is opened.
3. User selects payment method.
4. User enters card number, expiration date, security number.
5. Click “Make Order” button. [E2] [E3]
6. Alert popup appears. Displays order id. System changes focus to alert notification.
7. Click “OK” button.
8. Alert closes, card view closes, customer menu opens, cart is reset.

Exception scenarios

E1: No items added to cart

1. Warning popup appears saying "Your cart is empty!".
2. Click “OK” button.
3. Return to step 1.

E2: User did not fill a field.

1. Card view displays message "Must enter values in all fields".
2. Return to step 3.

E2: User entered a letter into one of the fields or entered too many numbers.

1. Card view displays message "Must enter valid values in all fields".
2. Return to step 3.

Post-Condition

An order is made and sent to both receptionist and order screen. Order is shown on order screen as incomplete. Customer receives order id.

Figure 2.5-1 Use case description(Make order use case)

For the rest of the descriptions see appendix – C use case description

2.6 Domain model (Rokas- Marwan)

The domain model is useful for understanding what goes into a domain by illustrating noteworthy concepts and how they interact with each other. (Larman, 1997)

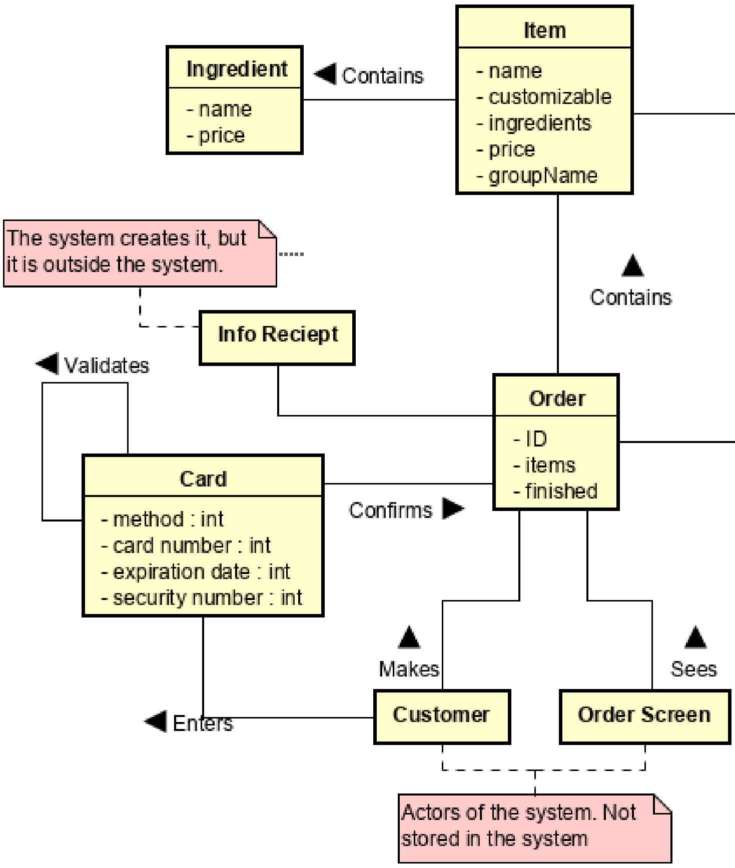


Figure 2.6-1 Domain model

Check appendix – B astah diagrams

3 Design

3.1 MVVM (Rokas)

MVVM was used in the client tier to organize components that are used in the user interface. This architectural pattern divides the client into layers of functionality. This makes the client easier to manage and implement.

These layers are:

- **View** - Manages the buttons and actions
- **ViewModel -** Formats the input from view. Checks and validates user input.
- **Model -** Holds information that the client uses. Responsible for the logic in client

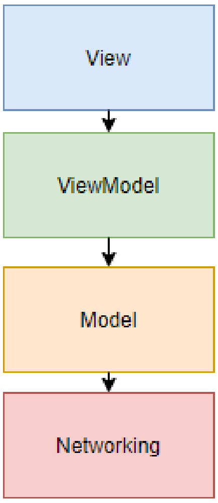


Figure 3.1-1 MVVM layers

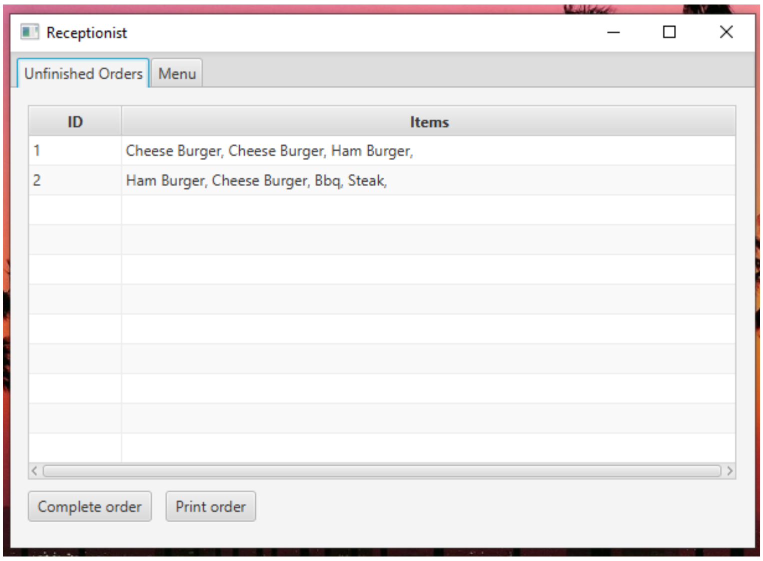
All object in the MVVM pattern are lazily instantiated, meaning that until a functionality is called, for example the customer cart, it is not taking up any resources in the system. A networking layer is added to these MVVM layers to act as communication with the server.

MVVM was used instead of MVC because of its benefits in reusability, loose coupling, and functionality separation. The MVVM structure can be seen in the class diagram in Appendix B - Astah Diagrams.

3.2 GUI design choices (Rokas)

The GUI is made using JavaFX. The GUI is split into three very different users. On When the user opens the system, he is greeted by a user selection screen with buttons that lead to different users and their functionality.

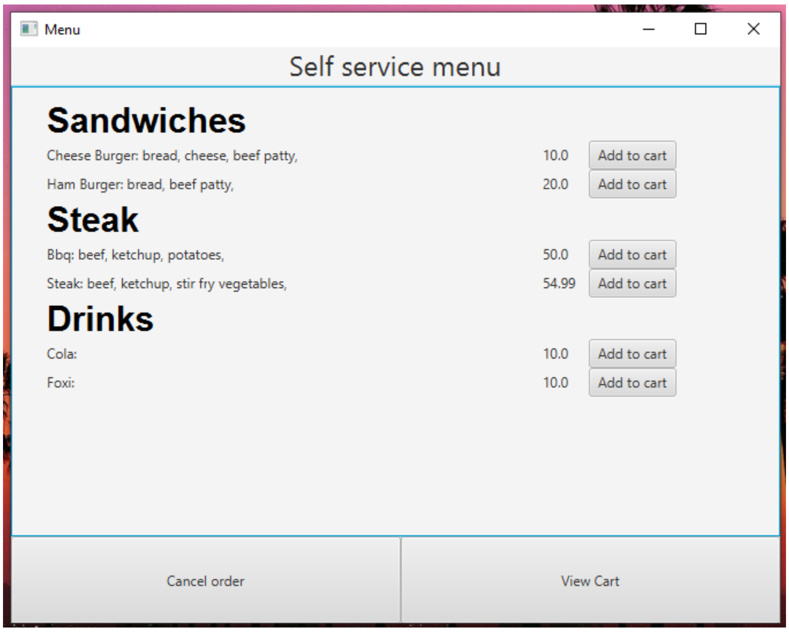
Only one of the users has login functionality, that is the receptionist. The login appears after selecting receptionist user and is the gateway to the rest of the receptionist's functionality. Completing the login gives the receptionist two lists. One is a list that is constantly updated from the server and holds all incomplete orders in the system. The other list is the menu for the system. The receptionist can perform various actions on these lists, such as completing order.



ID	Items
1	Cheese Burger, Cheese Burger, Ham Burger,
2	Ham Burger, Cheese Burger, Bbq, Steak,

Figure 3.2-1 Receptionist view - Unfinished orders

The customer UI accessed in the same way as the receptionist, by clicking a button to choose the customer. The customer interface main page is the menu, the user does not have to login because the user is reset every time, he makes an order and because the details of the user are of no importance to this system. The customers main functionality is to make an order with other functionality that specifies things in that order.



Self service menu

Sandwiches

Cheese Burger: bread, cheese, beef patty,	10.0	Add to cart
Ham Burger: bread, beef patty,	20.0	Add to cart

Steak

Bbq: beef, ketchup, potatoes,	50.0	Add to cart
Steak: beef, ketchup, stir fry vegetables,	54.99	Add to cart

Drinks

Cola:	10.0	Add to cart
Foxi:	10.0	Add to cart

Figure 3.2-2 Customer view - Menu

The order screen GUI responds to actions form the receptionist and client and is just an observer without any buttons.

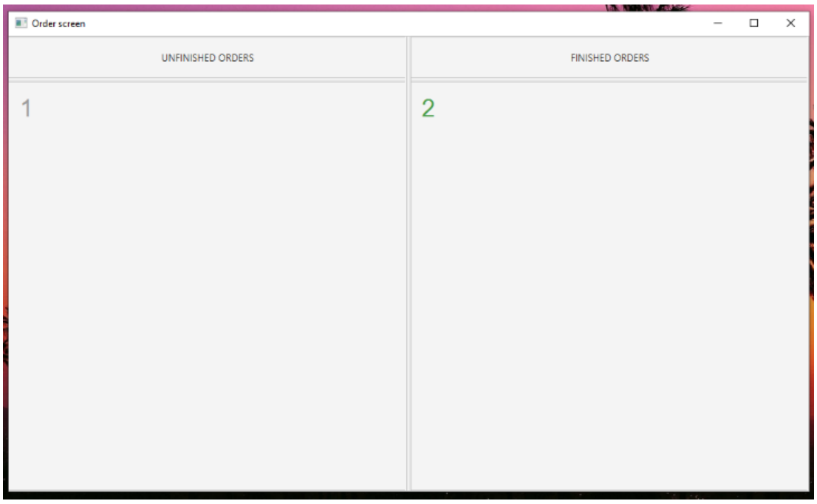


Figure 3.2-3 Order screen view

3.3 RMI (Rokas)

RMI was primarily chosen as it does not need a visible thread to keep both the server and the client working independently. As the system was analyzed, it was concluded that using sockets to get information from the server would require excessive use of the observer pattern. The alternative RMI offered a less complicated way of doing this, using observer pattern only where it is needed, in system functions that depend on updates from the server. Choosing RMI reduces implementation time and in that way gives more time implement and solve problems in other parts of the system.

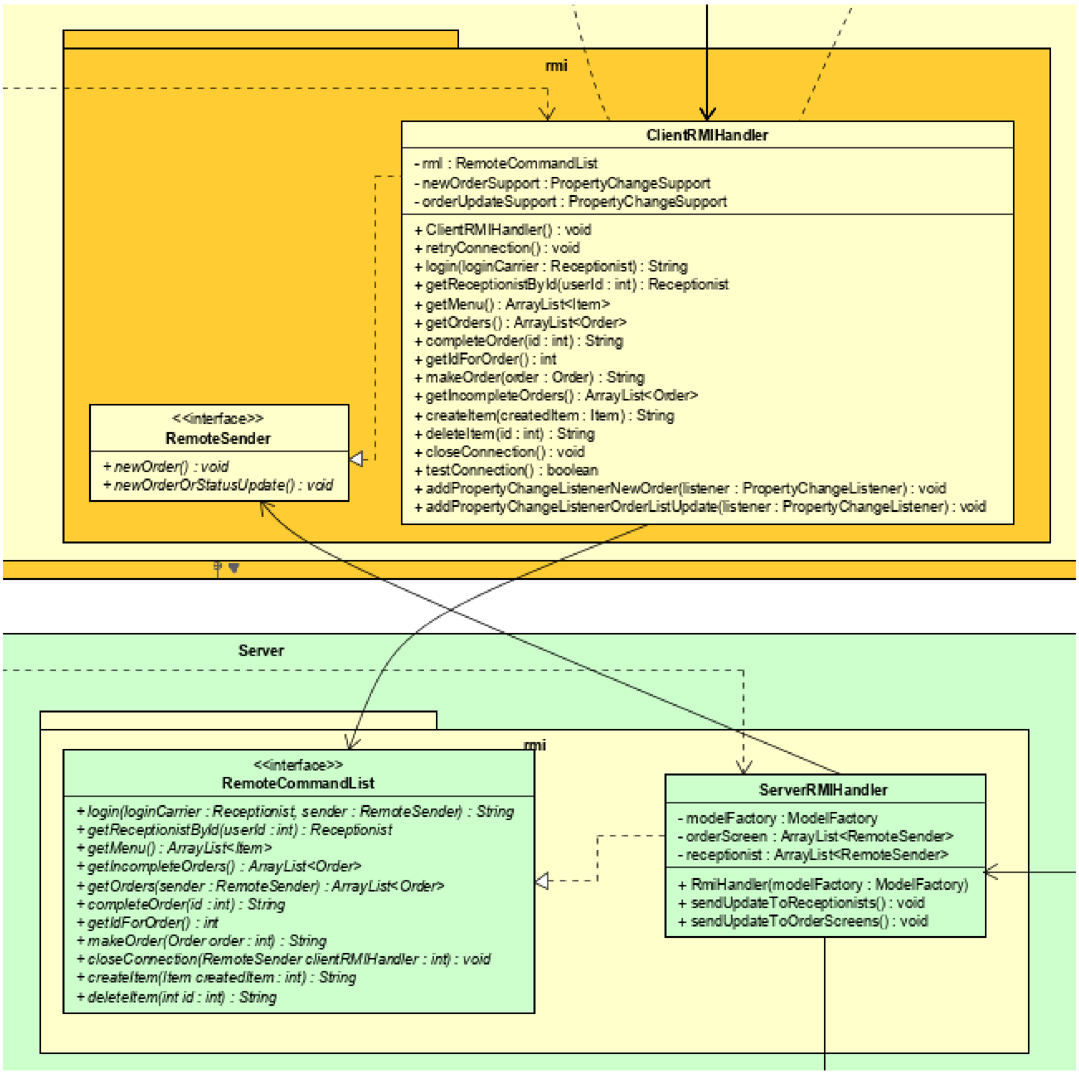

rmi
ClientRMIHandler
- rml : RemoteCommandList
- newOrderSupport : PropertyChangeSupport
- orderUpdateSupport : PropertyChangeSupport
+ ClientRMIHandler() : void
+ retryConnection() : void
+ login(loginCarrier : Receptionist) : String
+ getReceptionistById(userId : int) : Receptionist
+ getMenu() : ArrayList<Item>
+ getOrders() : ArrayList<Order>
+ completeOrder(id : int) : String
+ getIdForOrder() : int
+ makeOrder(order : Order) : String
+ getIncompleteOrders() : ArrayList<Order>
+ createItem(createdItem : Item) : String
+ deleteItem(id : int) : String
+ closeConnection() : void
+ testConnection() : boolean
+ addPropertyChangeListenerNewOrder(listener : PropertyChangeListener) : void
+ addPropertyChangeListenerOrderListUpdate(listener : PropertyChangeListener) : void
<<interface>>
RemoteSender
+ newOrder() : void
+ newOrderOrStatusUpdate() : void
Server
rmi
<<interface>>
RemoteCommandList
+ login(loginCarrier : Receptionist, sender : RemoteSender) : String
+ getReceptionistById(userId : int) : Receptionist
+ getMenu() : ArrayList< Item>
+ getIncompleteOrders() : ArrayList<Order>
+ getOrders(sender : RemoteSender) : ArrayList< Order>
+ completeOrder(id : int) : String
+ getIdForOrder() : int
+ makeOrder(Order order : int) : String
+ closeConnection(RemoteSender clientRMIHandler : int) : void
+ createItem(Item createdItem : int) : String
+ deleteItem(int id : int) : String
ServerRMIHandler
- modelFactory : ModelFactory
- orderScreen : ArrayList<RemoteSender>
- receptionist : ArrayList<RemoteSender>
+ RmiHandler(modelFactory : ModelFactory)
+ sendUpdateToReceptionists() : void
+ sendUpdateToOrderScreens() : void

Figure 3.3-1 RMI between Client and Server

3.4 Observer pattern (Rokas)

The observer pattern is needed primarily when having functionality that needs constant updating from the server. These are the order screen user functionality and the incomplete order list in the receptionist user. These lists need to show a live feed on what they are displaying to keep them relevant to the system.

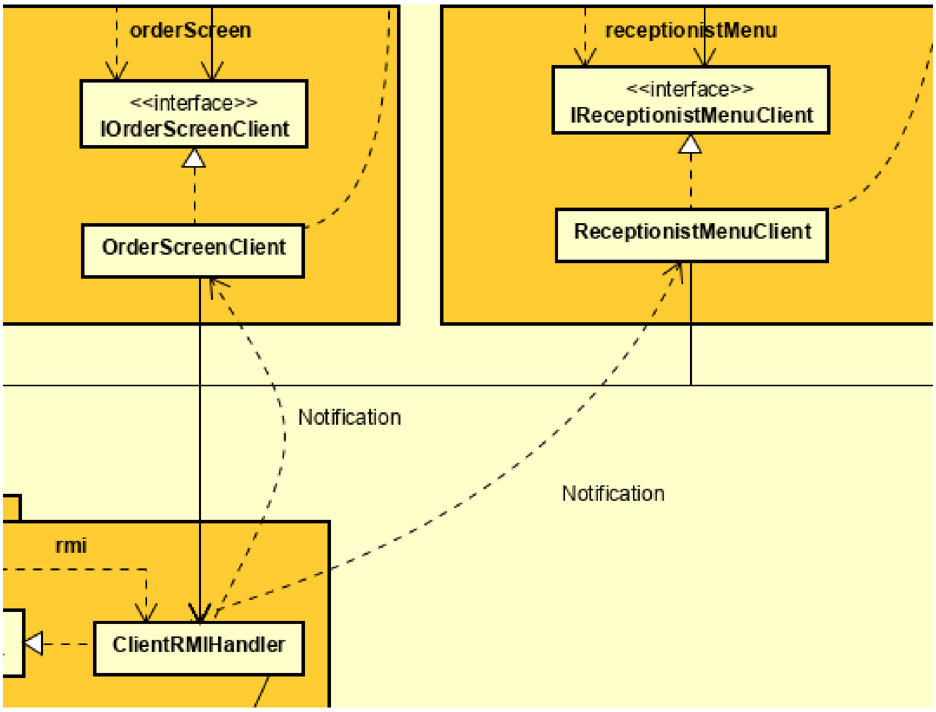


Figure 3.4-1 Observer pattern

3.5 Printing functionality (Rokas)

The printing functionality is a technology that was used to take care of the printing of orders for receptionist. This is an essential part of the system as it lets the back end of the kitchen, the workers that are there besides the receptionist, know what to make. This is taken care of by the java abstract toolkit which has a fully-fledged print tool.

3.6 Database (Marwan)

A relational database is used for storing information. that is used frequently in an organized way making sure to keep the relations between different entities while editing them. The database Accessed by prepared statements from the server using information passed from the client. For this MySQL was chosen as it is easy to use through its workbench editor that lets its users have a general look over the data.

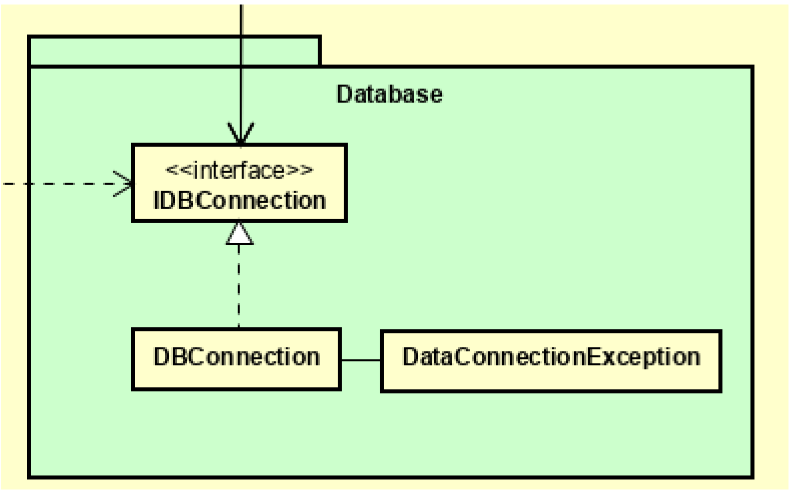


Figure 3.6-1 Database connection

3.7 ER diagram (Marwan)

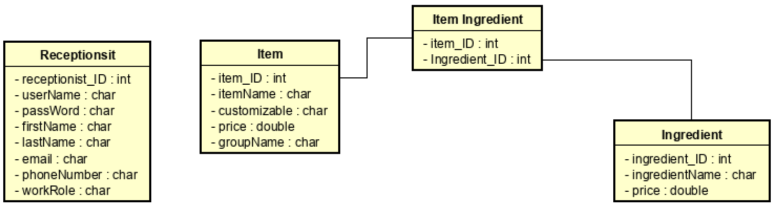


Figure 3.7-1 ER diagram

3.8 DAO (Marwan)

Data access object were used to divide functionality by entity. It allows the system to manage and track different types of objects passing to and from the database, making it easy to track where the errors are made.

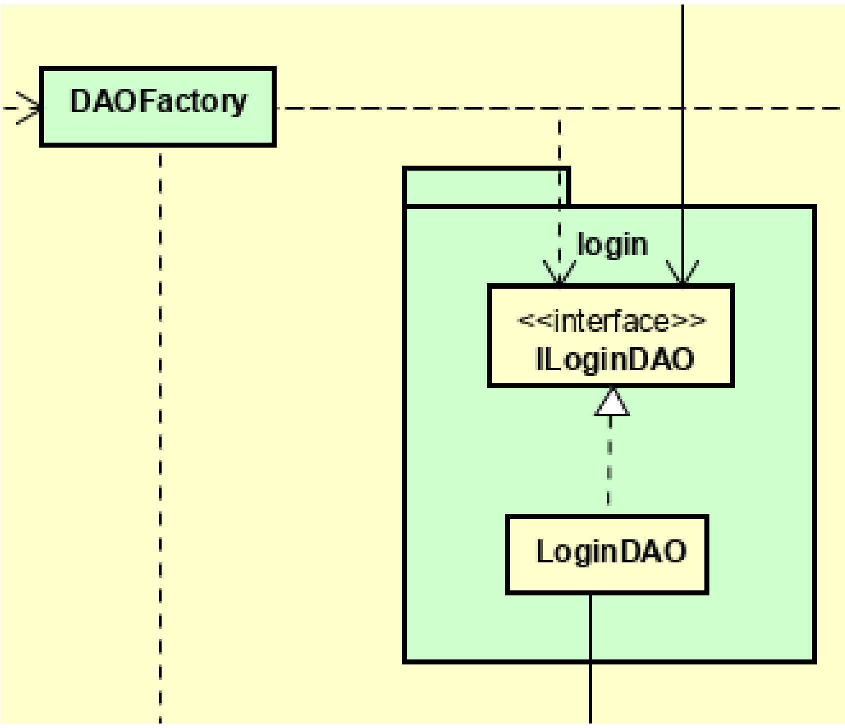


Figure 3.8-1 DAO - login

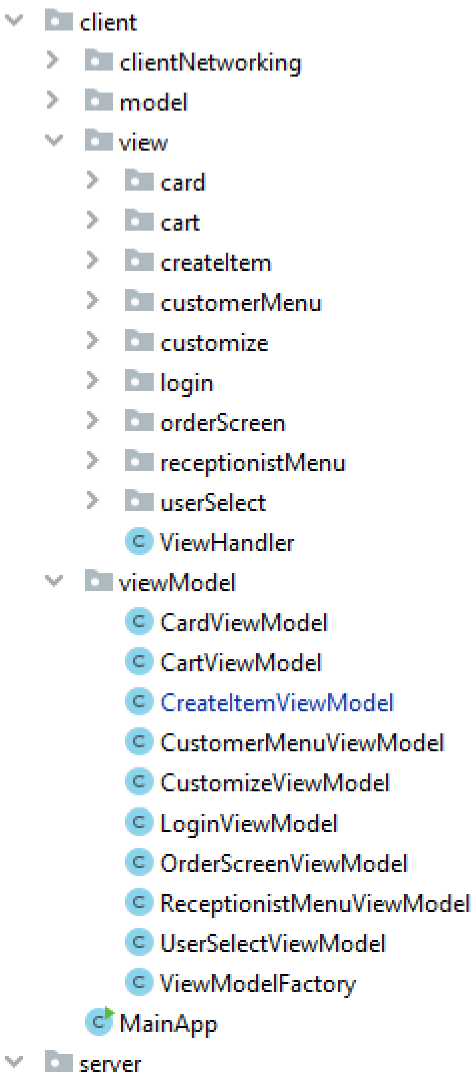
For the full design check appendix – B astah diagrams

4 Implementation (Rokas)

The functionality of making an order is the most important in the system therefore the implementation section will be based on it to show how it goes through the system. MVVM architectural pattern was used to structure the code as said in the section 3.1 (MVVM).

Accessed classes of the make order functionality shown in ascending order:

- CartController – View – Tier 1
- CardController – View – Tier 1
- CardViewModel – ViewModel – Tier 1
- CardModel – Model – Tier 1
- CustomerMenuModel – Tier 1
- CardClient – ClientNetworking – Tier 1
- ClientRMIHandler – ClientNetworking - Tier 1
- RMIHandler – RMI – Tier 2
- OrderModel – Model – Tier2



```
client
  clientNetworking
  model
  view
    card
    cart
    createItem
    customerMenu
    customize
    login
    orderScreen
    receptionistMenu
    userSelect
    ViewHandler
  viewModel
    CardViewModel
    CartViewModel
    CreateItemViewModel
    CustomerMenuViewModel
    CustomizeViewModel
    LoginViewModel
    OrderScreenViewModel
    ReceptionistMenuViewModel
    UserSelectViewModel
    ViewModelFactory
  MainApp
server
```

Figure 3.8-2 Implementation - packages

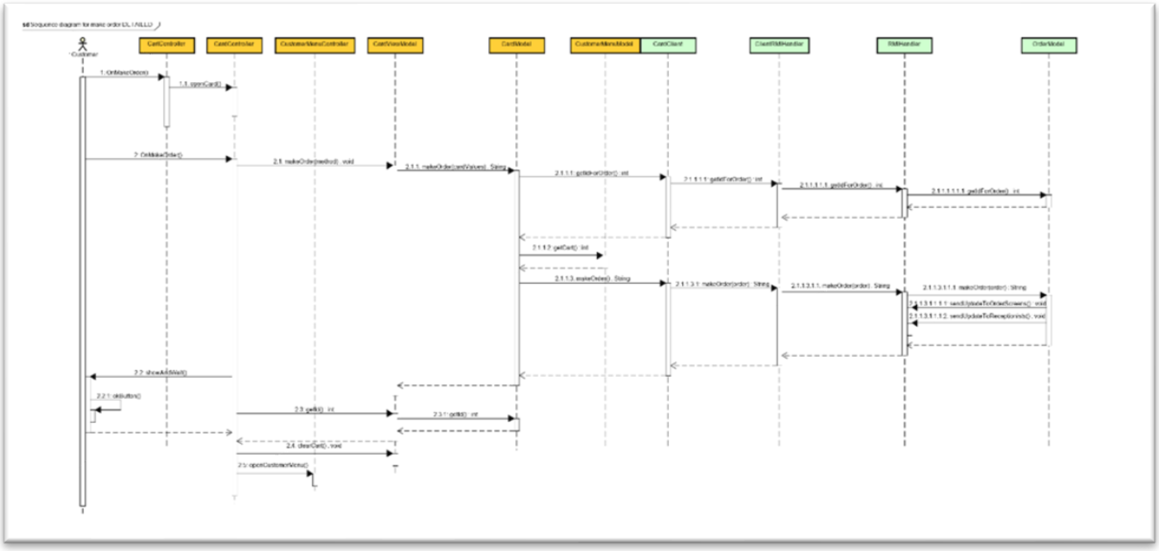
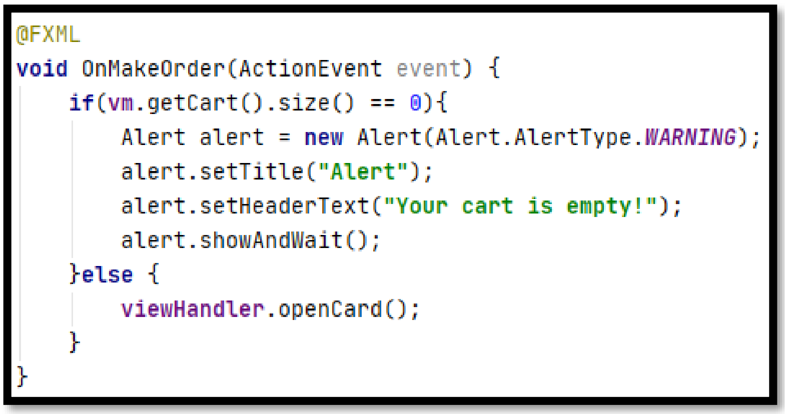


Figure 3.8-3 Sequence diagram - make order overview only

The sequence diagram can be found in Appendix B - Astah Diagrams

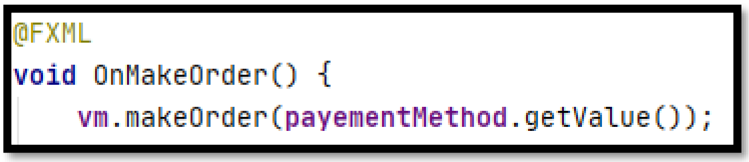
The functionality of make order starts at the cart when the customer presses the “Make order” button. The system checks If the customer does not have any items added, if that is true a popup alert appears saying “Your cart is empty!”. If the cart has items in it the view switches to payment by card view where the user must enter his card information.



```
@FXML
void OnMakeOrder(ActionEvent event) {
    if(vm.getCart().size() == 0){
        Alert alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle("Alert");
        alert.setHeaderText("Your cart is empty!");
        alert.showAndWait();
    }else {
        viewHandler.openCard();
    }
}
```

Figure 3.8-4 alerting customer when cart is empty

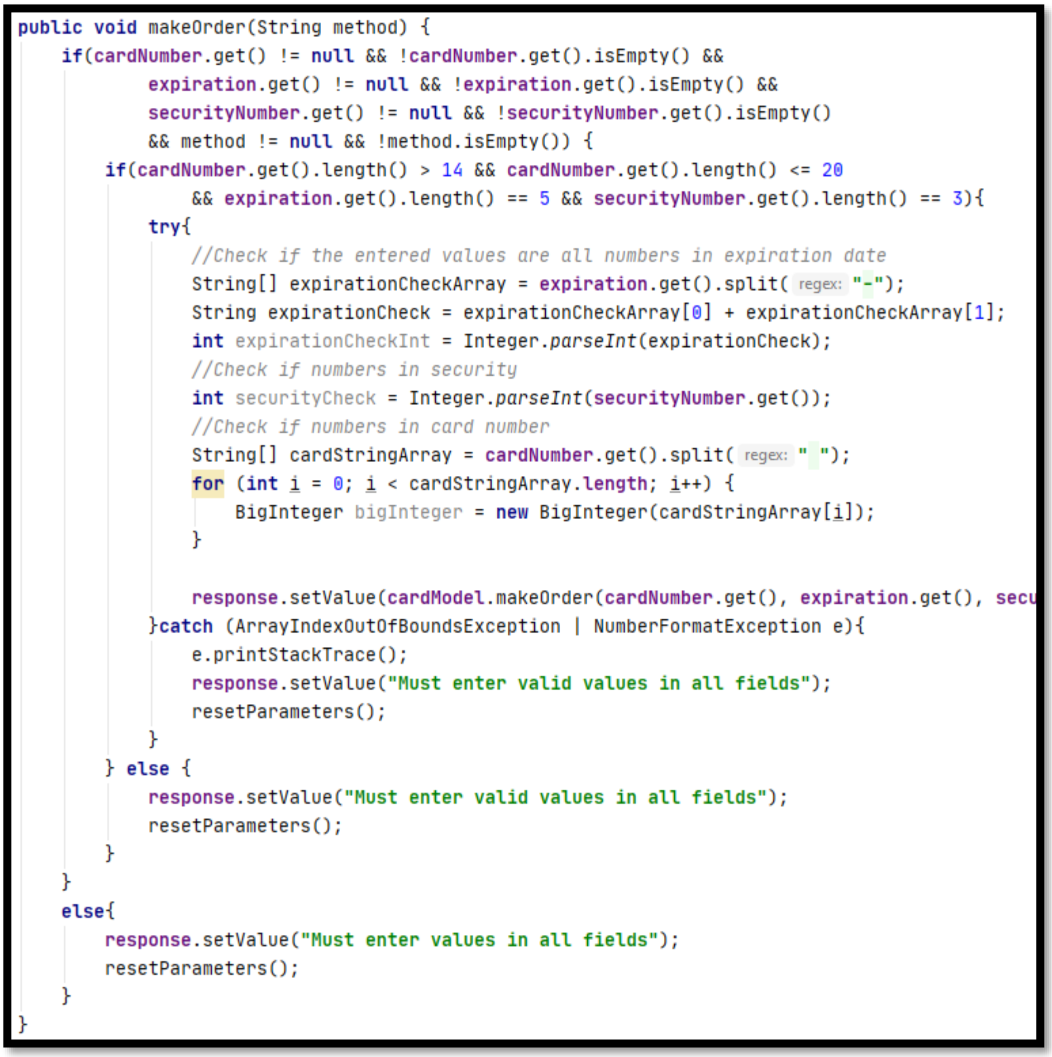
The customer enters card number, expiration date, security number and selects a payment method. Then the customer presses the make order button again . The handler method sends out a payment method to the view model. This method has other functionality for when after making order is completed.



```
@FXML
void OnMakeOrder() {
    vm.makeOrder(payementMethod.getValue());
```

Figure 3.8-5 CardController Class in client

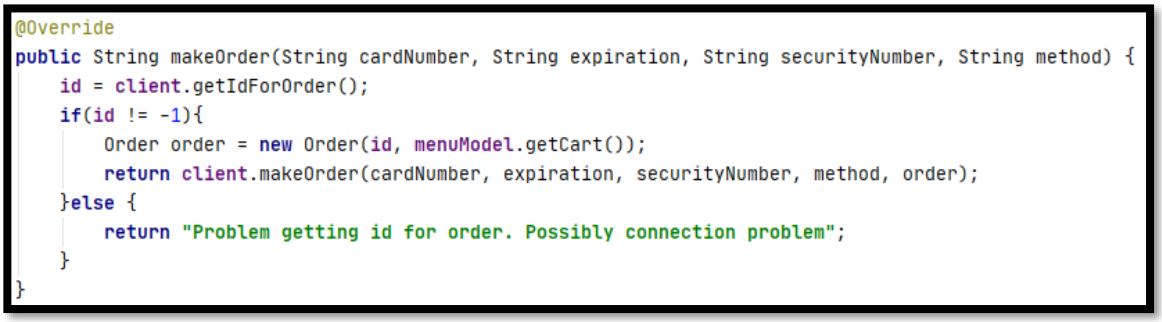
The card view model performs user input validation. First it checks if any of the inputs are missing, then it checks if the input is of the required length and finally checks if the input is all numbers by trying to convert them to integers. If any of these checks fails, the user sees a response from the system and entered card information is reset. The third phase check deliberately performs parsing to integer that causes an exception if the value being parsed is not an integer. If the checks pass all the details of the card are passed to the card model which constructs the order.



```
public void makeOrder(String method) {
    if(cardNumber.get() != null && !cardNumber.get().isEmpty() &&
            expiration.get() != null && !expiration.get().isEmpty() &&
            securityNumber.get() != null && !securityNumber.get().isEmpty()
            && method != null && !method.isEmpty()) {
        if(cardNumber.get().length() > 14 && cardNumber.get().length() <= 20
                && expiration.get().length() == 5 && securityNumber.get().length() == 3){
            try{
                //Check if the entered values are all numbers in expiration date
                String[] expirationCheckArray = expiration.get().split( regex: "-");
                String expirationCheck = expirationCheckArray[0] + expirationCheckArray[1];
                int expirationCheckInt = Integer.parseInt(expirationCheck);
                //Check if numbers in security
                int securityCheck = Integer.parseInt(securityNumber.get());
                //Check if numbers in card number
                String[] cardStringArray = cardNumber.get().split( regex: " ");
                for (int i = 0; i < cardStringArray.length; i++) {
                    BigInteger bigInteger = new BigInteger(cardStringArray[i]);
                }

                response.setValue(cardModel.makeOrder(cardNumber.get(), expiration.get(), secu
            }catch (ArrayIndexOutOfBoundsException | NumberFormatException e){
                e.printStackTrace();
                response.setValue("Must enter valid values in all fields");
                resetParameters();
            }
        } else {
            response.setValue("Must enter valid values in all fields");
            resetParameters();
        }
    }
    else{
        response.setValue("Must enter values in all fields");
        resetParameters();
    }
}
```

Figure 3.8-6 CardViewModel Class in Client

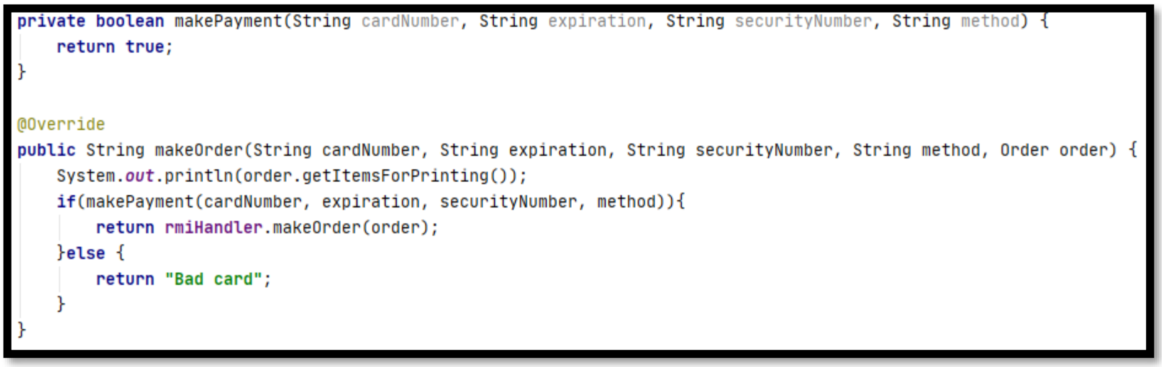
The client cannot set an id of the order itself; this would risk collisions and cause confusion for the customers and the receptionists. The id is instead retrieved from the server where a count is kept of the current available id. If the client fails to retrieve an id from the server, it sends back -1 which the model uses to determine if it is ok to continue. The customer menu model is accessed to obtain the cart to which the customer added items. Together the id and the cart array list are used to construct the order.



```
@Override
public String makeOrder(String cardNumber, String expiration, String securityNumber, String method) {
    id = client.getIdForOrder();
    if(id != -1){
        Order order = new Order(id, menuModel.getCart());
        return client.makeOrder(cardNumber, expiration, securityNumber, method, order);
    }else {
        return "Problem getting id for order. Possibly connection problem";
    }
}
```

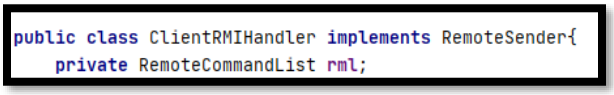
Figure 3.8-7 CardModel Class in Client

The card client checks if the transaction works using the details of the card. The back end of the card payment functionality was delimited therefore it is not implemented. The card check always returns true.



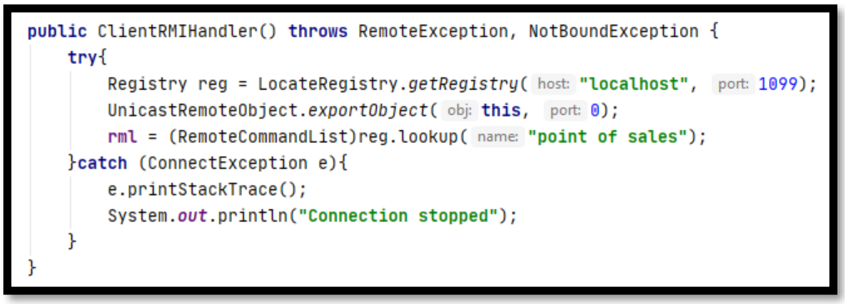
```
private boolean makePayment(String cardNumber, String expiration, String securityNumber, String method) {
    return true;
}

@Override
public String makeOrder(String cardNumber, String expiration, String securityNumber, String method, Order order) {
    System.out.println(order.getItemsForPrinting());
    if(makePayment(cardNumber, expiration, securityNumber, method)){
        return rmiHandler.makeOrder(order);
    }else {
        return "Bad card";
    }
}
```

A link to the server is established using RMI. RMI hides the connection details, threading and makes it easy to create new functionality. The classes look as if they were accessing just another method on a class that has an interface. The client side of the RMI finds a registry named local host on port 1099, which in this case is the same computer. The RMI handler sends a copy of itself and obtains an interface to communicate with a registry point named “point of sales”.



```
public class ClientRMIHandler implements RemoteSender{
    private RemoteCommandList rml;
```

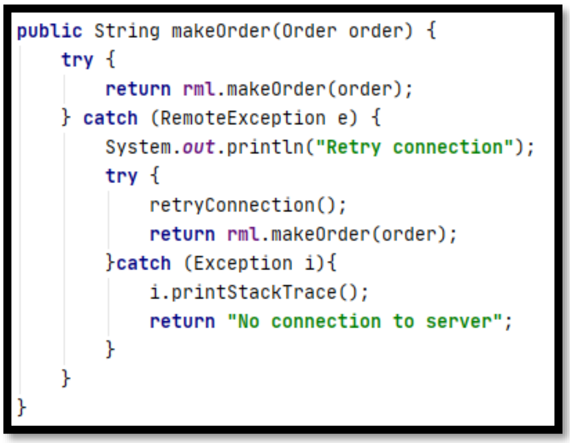
Figure 3.8-8 ClientRmiHandler Class (RMI Client side)



```
public ClientRMIHandler() throws RemoteException, NotBoundException {
    try{
        Registry reg = LocateRegistry.getRegistry( host: "localhost",  port: 1099);
        UnicastRemoteObject.exportObject( obj: this,  port: 0);
        rml = (RemoteCommandList)reg.lookup( name: "point of sales");
    }catch (ConnectException e){
        e.printStackTrace();
        System.out.println("Connection stopped");
    }
}
```

Figure 3.8-9 ClientRmiHandler Class constructor

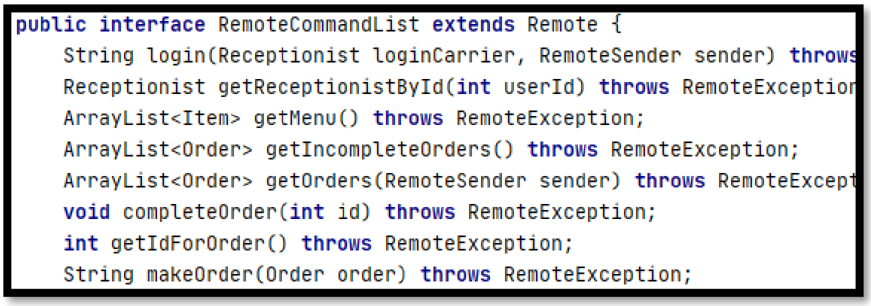
The connection to the server can be severed. If this happens the system will try to reconnect. If it fails, the system will stop trying to connect and display an error message that connection cannot be established. If the user retries the functionality when the server is online the functionality will work again without restarting the system.



```
public String makeOrder(Order order) {
    try {
        return rml.makeOrder(order);
    } catch (RemoteException e) {
        System.out.println("Retry connection");
        try {
            retryConnection();
            return rml.makeOrder(order);
        }catch (Exception i){
            i.printStackTrace();
            return "No connection to server";
        }
    }
}
```

Figure 3.8-10 CLientRMIHandler Class in Client

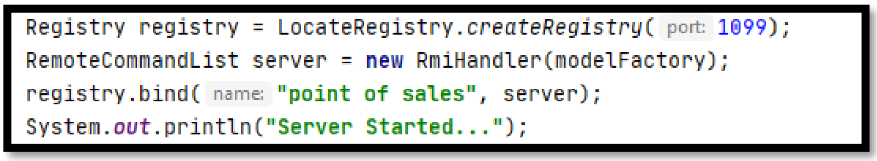
The Remote Command List was obtained by the client when it connected to the server and is used to call the methods on the server.



```
public interface RemoteCommandList extends Remote {
    String login(Receptionist loginCarrier, RemoteSender sender) throws
    Receptionist getReceptionistById(int userId) throws RemoteExceptio
    ArrayList<Item> getMenu() throws RemoteException;
    ArrayList<Order> getIncompleteOrders() throws RemoteException;
    ArrayList<Order> getOrders(RemoteSender sender) throws RemoteExcept
    void completeOrder(int id) throws RemoteException;
    int getIdForOrder() throws RemoteException;
    String makeOrder(Order order) throws RemoteException;
```

Figure 3.8-11 RemoteComandList Interface is server

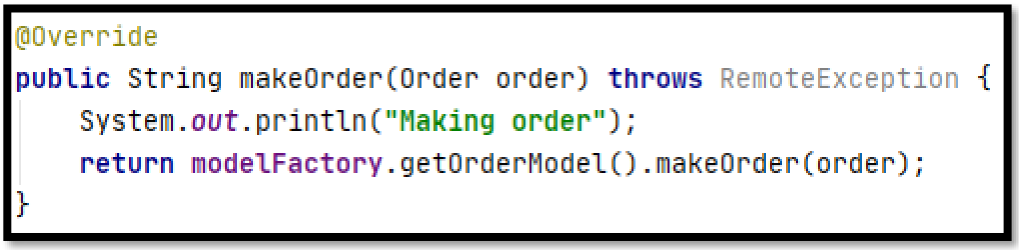
The server creates the registry and binds its RMI handler to it.



```
Registry registry = LocateRegistry.createRegistry( port: 1099);
RemoteCommandList server = new RmiHandler(modelFactory);
registry.bind( name: "point of sales", server);
System.out.println("Server Started...");
```

Figure 3.8-12 JavaServer Class in Server

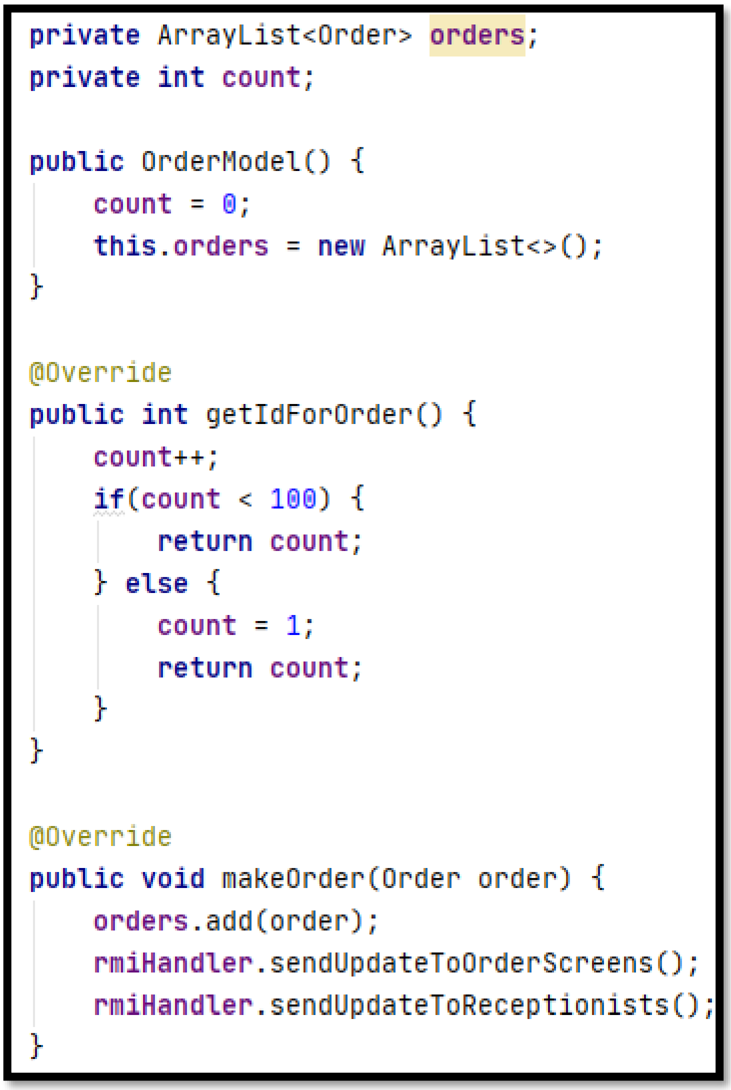
The rmi handler mehods determine where the request will go.



```
@Override
public String makeOrder(Order order) throws RemoteException {
    System.out.println("Making order");
    return modelFactory.getOrderModel().makeOrder(order);
}
```

Figure 3.8-13 RmiHandler Class in Server

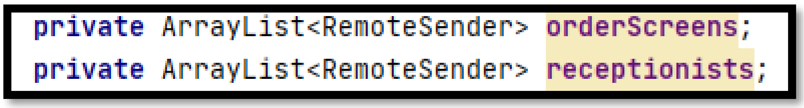
All the orders are held in the order model. This is done because there is currently no need to save these orders in the database. Saving these orders in the database would also cause slowdown for the system. The order gets added to the list on the server and updates are sent to each receptionist and order screen that is currently online.



```
private ArrayList<Order> orders;
private int count;

public OrderModel() {
    count = 0;
    this.orders = new ArrayList<>();
}

@Override
public int getIdForOrder() {
    count++;
    if(count < 100) {
        return count;
    } else {
        count = 1;
        return count;
    }
}

@Override
public void makeOrder(Order order) {
    orders.add(order);
    rmiHandler.sendUpdateToOrderScreens();
    rmiHandler.sendUpdateToReceptionists();
}
```

Figure 3.8-14 OrderModel Class in Server

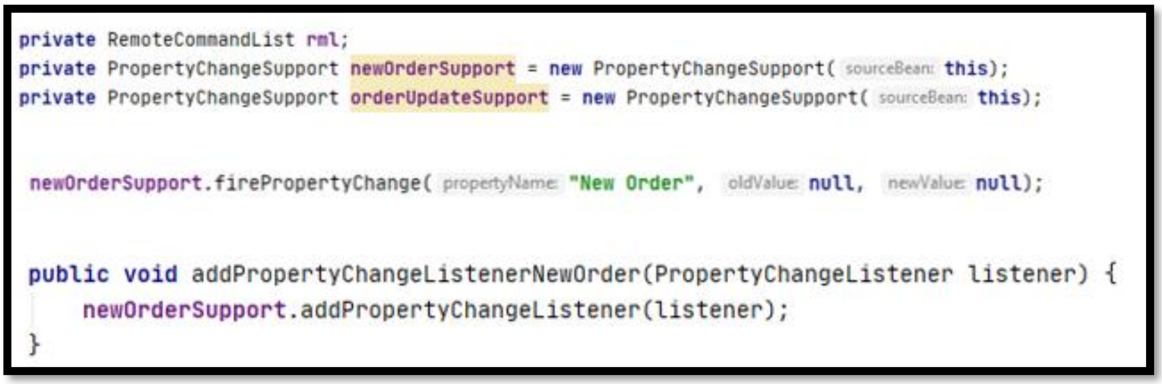
The receptionist and the order screen user get added to the server RMI handler lists to keep track of who to send updates to. Call back to the client can only happen if the clients send their interface using which you can access them. This works much like when the client connects to the server except the sending of the interface is done for specific methods in the client socket handler. To update these users the server RMI handler cycles trough every user that he has received and calls a method on the interface it received. The listed users are removed when the order screen or receptionist users close their systems.



```
private ArrayList<RemoteSender> orderScreens;
private ArrayList<RemoteSender> receptionists;
```

Figure 3.8-15 RmiHandler Class in Server

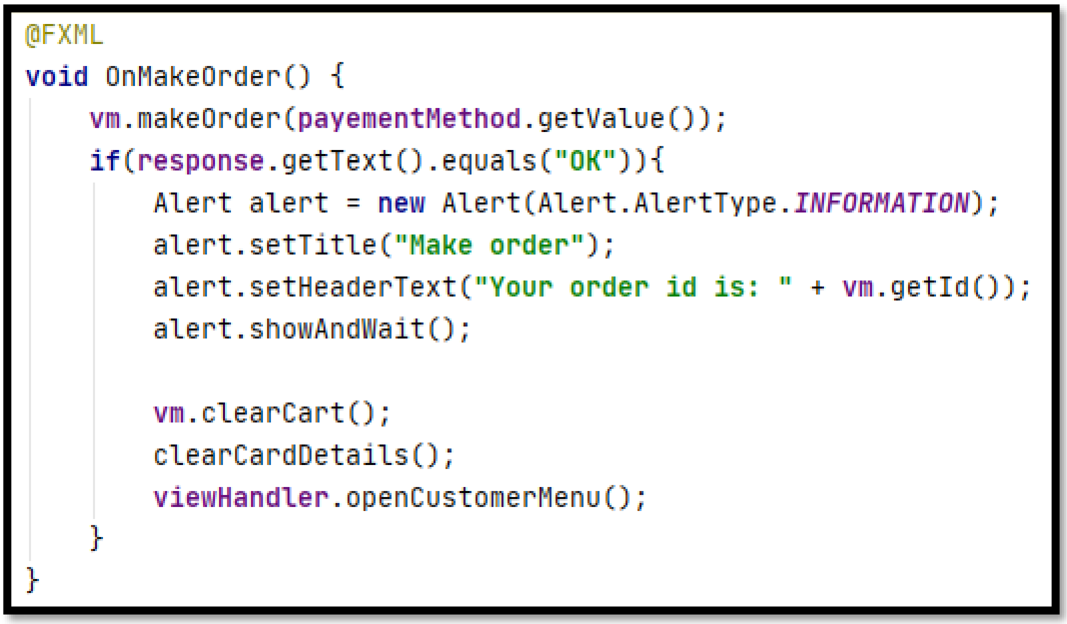
The users after receiving the update method call will start jumping up the MVVM architectural pattern using the observer pattern. This is done because the classes cannot see what is calling them and therefore cannot call any methods to which the GUI would respond. Once the observers jump to the controller an update method is called that simply retrieves the orders again from the server, in this way keeping what is show up to date.



```
private RemoteCommandList rml;
private PropertyChangeSupport newOrderSupport = new PropertyChangeSupport( sourceBean: this);
private PropertyChangeSupport orderUpdateSupport = new PropertyChangeSupport( sourceBean: this);


 newOrderSupport.firePropertyChange( propertyName: "New Order",  oldValue: null,  newValue: null);


 public void addPropertyChangeListenerNewOrder(PropertyChangeListener listener) {
     newOrderSupport.addPropertyChangeListener(listener);
 }
```

Figure 3.8-16 ClientRMIHandlerClass in Client

After the making of order is done the controller checks if the making of order was successful. This tells the controller if it ok to continue. The id is displayed for the user to see in an alert. When the user presses ok button on the alert the cart and the card are cleared, and the user sees the customer menu. Another customer can now use the system.



```
@FXML
void OnMakeOrder() {
    vm.makeOrder(payementMethod.getValue());
    if(response.getText().equals("OK")){
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Make order");
        alert.setHeaderText("Your order id is: " + vm.getId());
        alert.showAndWait();

        vm.clearCart();
        clearCardDetails();
        viewHandler.openCustomerMenu();
    }
}
```

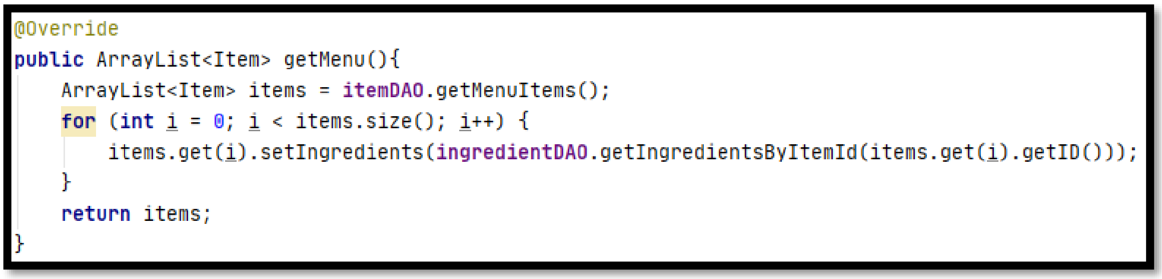
Figure 3.8-17 CardController Class in Client

Accessed classes of the get menu functionality shown in ascending order:

- MenuModel – Model – Tier 2
- ItemDAO – Persistance – Tier 2
- IngredientDAO – Persistance – Tier 2
- DBConnection – Persistance – Tier 2

The focus will be on the classes in tier two.

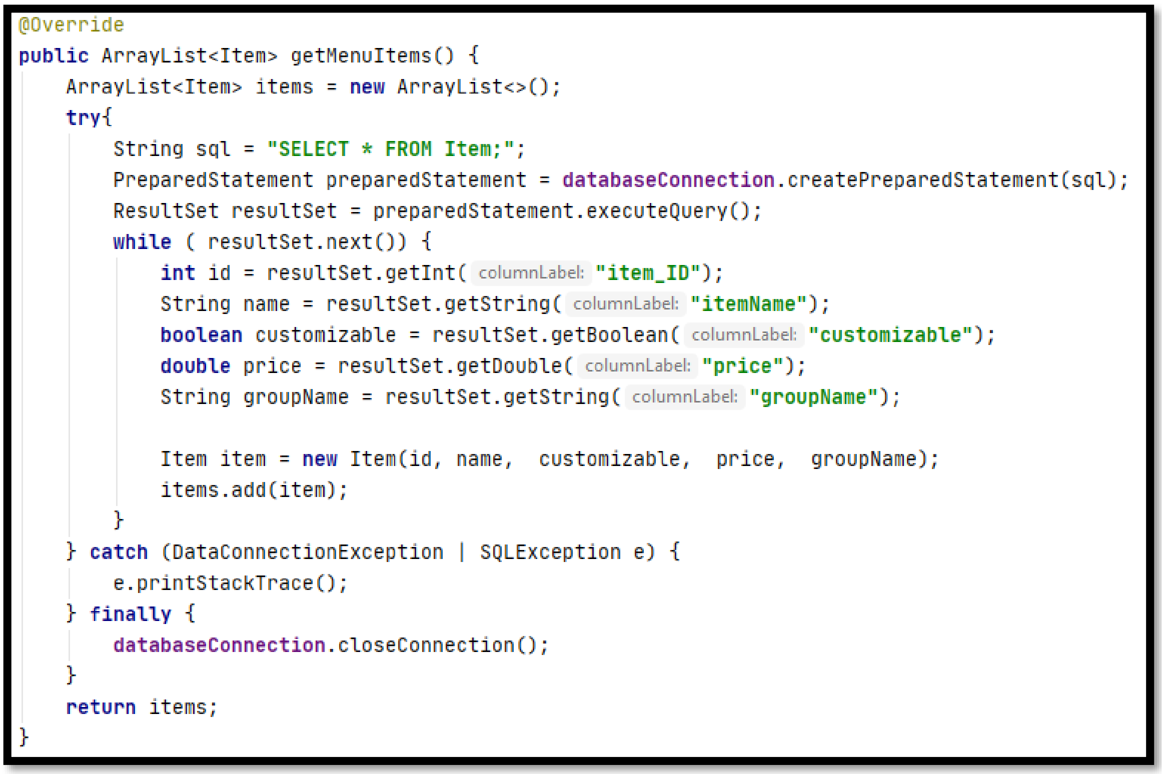
The method in the menu model first gets all the items stored in the database. These items are what goes into the menu. This does not however retrieve the ingredients of the item. For this a loop cycles trough every item and accesses the database again to get the ingredients that are linked to the item.



```java
@Override
public ArrayList<Item> getMenu(){
    ArrayList<Item> items = itemDAO.getMenuItems();
    for (int i = 0; i < items.size(); i++) {
        items.get(i).setIngredients(ingredientDAO.getIngredientsByItemId(items.get(i).getID()));
    }
    return items;
}
```

Figure 3.8-18 MenuModel Class in Server

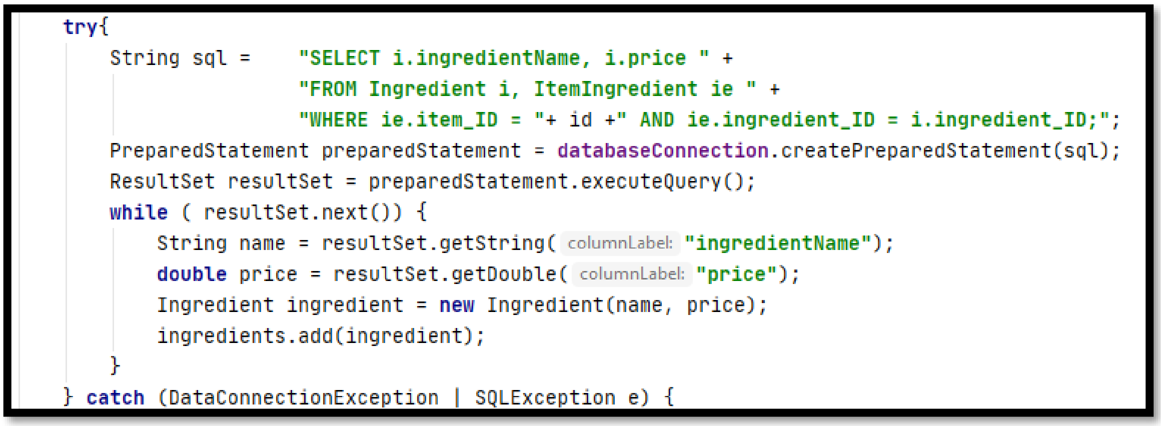
All these objects are obtained from the Data Access Objects which as explained in 3.8 (DAO) help organise entity access. The "getMenuItems" method accesses the database using a prepared statement which selects all items that are in the table. Each result is looped trough and created into an Item and added to an array list that is returned to the menu model. Before returning the array list it closes the database connection.



```java
@Override
public ArrayList<Item> getMenuItems() {
    ArrayList<Item> items = new ArrayList<>();
    try{
        String sql = "SELECT * FROM Item;";
        PreparedStatement preparedStatement = databaseConnection.createPreparedStatement(sql);
        ResultSet resultSet = preparedStatement.executeQuery();
        while ( resultSet.next()) {
            int id = resultSet.getInt( columnLabel: "item_ID");
            String name = resultSet.getString( columnLabel: "itemName");
            boolean customizable = resultSet.getBoolean( columnLabel: "customizable");
            double price = resultSet.getDouble( columnLabel: "price");
            String groupName = resultSet.getString( columnLabel: "groupName");

            Item item = new Item(id, name,  customizable,  price,  groupName);
            items.add(item);
        }
    } catch (DataConnectionException | SQLException e) {
        e.printStackTrace();
    } finally {
        databaseConnection.closeConnection();
    }
    return items;
}
```

Figure 3.8-19 ItemDAO Class in Server

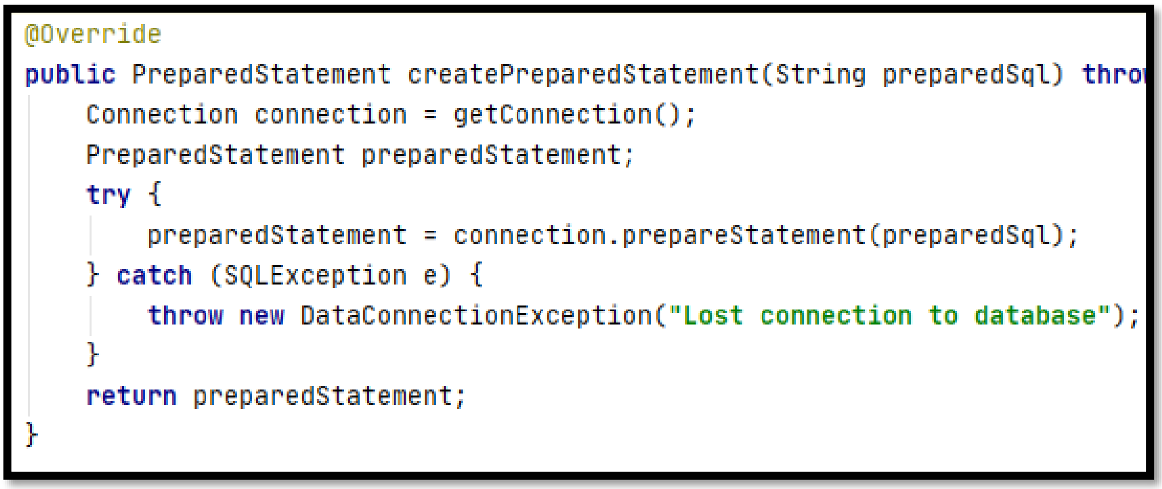
Ingredients are retrieved for each item. For this the procedure is the same as with the items except for the prepared statement and the class that is created are different. The prepared statement selects ingredients which have an id associated with the current item id in a table called ItemIngredient. The ItemIngredient entity ties the Item and the ingredient together and holds the id of the item and the id of the ingredient, both foreign keys. The retrieved ingredients are added to a list and returned to the menu model.



```
try{
    String sql =    "SELECT i.ingredientName, i.price " +
                    "FROM Ingredient i, ItemIngredient ie " +
                    "WHERE ie.item_ID = "+ id +" AND ie.ingredient_ID = i.ingredient_ID;";
    PreparedStatement preparedStatement = databaseConnection.createPreparedStatement(sql);
    ResultSet resultSet = preparedStatement.executeQuery();
    while ( resultSet.next()) {
        String name = resultSet.getString( columnLabel: "ingredientName");
        double price = resultSet.getDouble( columnLabel: "price");
        Ingredient ingredient = new Ingredient(name, price);
        ingredients.add(ingredient);
    }
} catch (DataConnectionException | SQLException e) {
```

Figure 3.8-20 IngredientDAO Class in Server

The DBConnection holds the database connection and provides functionality like the mentioned prepared statements.



```
@Override
public PreparedStatement createPreparedStatement(String preparedSql) thro
    Connection connection = getConnection();
    PreparedStatement preparedStatement;
    try {
        preparedStatement = connection.prepareStatement(preparedSql);
    } catch (SQLException e) {
        throw new DataConnectionException("Lost connection to database");
    }
    return preparedStatement;
}
```

Figure 3.8-21 PreparedStatment method implemented by DBConnection class

`

The full source code is attached with the file. Check appendix – E

5 Testing

To list the user story as finished, it must successfully pass all the relevant tests, that ensure the reason behind it, for instance:

“As a customer, I want to have a menu of items I can order, so that I can choose what items I want to buy”

The only way that the user story behind this requirement would be marked finished is if the customer can in fact get to the menu and choose items to buy, otherwise the user story would be marked unfinished hence the requirement is not met.

The same would go for all the rest of the requirements.

The tests for the system were done using black box testing. The system would take input like the ones a regular user would use. The tests would be executed without going through the code.

5.1 Blackbox testing

Some of the test cases that the system went through:

5.1.1 Test case: adding item that is non customizable (Success)

Pre-conditions:
System must be connected to the network. Receptionist must be logged in. Menu tab must be opened

Step	Action	Reaction	Test Data
1	Clicks “Create new menu item” button	Create item view opens. Receptionist menu closes	
2	User enters item name, price, group name, ingredients.		Name = “Salmon sandwich”

			Price = 10.99
			Group name = "Sandwich"
			Ingredients = "salmon, bread"

3	User selects if item is customizable.		False is selected
	User clicks "Create item" button.	Create item view is closed, receptionist menu is opened	

Post-conditions

The menu is updated with the newest item

Test status

Works as intended

5.1.2 Test case: adding item that is customizable (Success)

Pre-conditions:

System must be connected to the network. Receptionist must be logged in. Menu tab must be opened

Step	Action	Reaction	Test Data
1	Clicks "Create new menu item" button	Create item view opens. Receptionist menu closes	

2	User enters item name, price, group name, ingredients.		Name = "Salmon sandwich"
			Price = 10.99
			Group name = "Sandwich"
			Ingredients = "salmon =5.99, bread =1.40"

3	User selects if item is customizable.		True is selected

	User clicks “Create item” button.	Create item view is closed, receptionist menu is opened	

Post-conditions

The menu is updated with the newest item

Test status

Works as intended

5.1.3 Test case: adding item with missing fields (Fail)

Pre-conditions:

System must be connected to the network. Receptionist must be logged in. Menu tab must be opened.

Step	Action	Reaction	Test Data
1	Clicks “Create new menu item” button	Create item view opens. Receptionist menu closes	

2	User enters item name, price, group name, ingredients.		Name = “Salmon sandwich”
			Price = blank
			Group name = “Sandwich”
			Ingredients = “salmon =5.99, bread =1.40”

3	User selects if item is customizable.		False is selected
4	User clicks “Create item” button.	The system displays message "Must enter all values".	

Post-conditions

The user has to enter fields he has not filled in.

Test status

Works as intended

5.1.4 Test case: adding item with correct details but no connection (Fail)

Pre-conditions:

System must be connected to the network. Receptionist must be logged in. Menu tab must be opened.

Step	Action	Reaction	Test Data
1	Clicks “Create new menu item” button	Create item view opens. Receptionist menu closes	

2	User enters item name, price, group name, ingredients.		Name = “Salmon sandwich”
			Price = 10.99
			Group name = “Sandwich”
			Ingredients = “salmon =5.99, bread =1.40”

3	User selects if item is customizable.		False is selected
4	User clicks “Create item” button.	The system displays message "Failed to connect".	

Post-conditions

System displays error message.

Test status

Works as intended

5.1.5 Test case: Start making order with items in cart (Success)

Pre-conditions:

Cart must be opened.

Step	Action	Reaction	Test Data

1	Click ”Make order” button	Cart closes, card view is opened.	

Post-conditions

Card view is opened. Customer can continue to make order.

Test status

Works as intended

5.1.6 Test case: Start making order with no items in cart (Fail)

Pre-conditions:

Cart must be opened.

Step	Action	Reaction	Test Data
1	Click ”Make order” button	Warning popup appears saying "Your cart is empty!".	
	Click “OK” button.		

Post-conditions

Nothing happens, user must have items in cart.

Test status

Works as intended

5.1.7 Test case: Remove item from cart (Success)

Pre-conditions:

Cart must be opened, items must be added to cart.

Step	Action	Reaction	Test Data
1	Click “Remove” button that is next to every item in the cart.	Item disappears from cart.	

Post-conditions

The item is removed from cart

Test status

Works as intended

5.1.8 Test case: Go back to menu from cart (Success)

Pre-conditions:

Cart must be opened, connection to server must be established.

Step	Action	Reaction	Test Data
1	Click "Back to menu" button.	Cart is closed, customer menu is opened	

Post-conditions

The view switches to customer menu

Test status

Works as intended

5.1.9 Test case: Go back to menu from cart with no connection (Fail)

Pre-conditions:

Cart must be opened.

Step	Action	Reaction	Test Data
1	Click "Back to menu" button.	Alert popup appears. Saying “No connection to server”	

Post-conditions

System displays alert with error response.

Test status

Works as intended

5.1.10 Test case: Customer resetting cart (Success)

Pre-conditions:

Customer menu must be opened.

Step	Action	Reaction	Test Data
1	Click “Cancel order” button	Cart is emptied	

Post-conditions

Any items that were added to cart are deleted. If no items were added nothing changes.

Test status

Works as intended

The rest of test cases can be found in appendix F – use case testing.

6 Results and Discussion (Marwan)

The system provided the customers and employees an easier way of doing the ordering procedure, it made it easy for the receptionists to work on the menu by helping them remove and add items while specifying if they can be customized by the customer to meet their expectations. And finish the orders that the customers make by using the print function to share the order with the people working on it and not using the system directly The customers can buy items from the menu and they customize the ones that are set to be customizable by the receptionist. They can also wait and track the orders as they are being done through the order screen that will notify them if the order is finished.

6.1 Finished user stories

	User Stories	Tests passed
1	As a user, I want to have a menu of different system users, so that I can choose which user I want to use.	5/5

2	As an order screen user, I want to have two separate lists for completed and incomplete orders, so that I can differentiate between them.	Is covered by user story 10
3	As a customer, I want to have a menu of items I can order, so that I can choose what items I want to buy.	Taken care of in user story 1
4	As a customer, I want to have a cart to which I can add items to, so that I can buy more than one item at a time.	1/1
5	As a customer I want to have functionality to view the cart, so that I can see what I added to my order.	3/3
6	As a customer, I want to have functionality to reset the cart, so that I can remove all items I added.	1/1
7	As a customer, I want to have functionality to remove items while in the cart, so that I can remove specific items form the order.	1/1
8	As a customer, I want to have an option to customize items I added to cart, so that I can change the item to my preference	4/4
9	As a customer, I want to be able to make order, so that my order can be fulfilled.	2/2
10	As a customer I want to be able to pay for my order by card, so that I can have a way to pay for the goods I will receive.	5/5
11	As a customer, I want to see an id on the screen after I pay for my order, so that I know which order is mine.	Is covered by user story 10
12	As a receptionist, I want to login, so that there are no other users who use my receptionist account.	8/8
13	As a receptionist, I want to have a list that displays current menu items shown to customers, so that I can manage what I put on the menu.	1/1
14	As a receptionist, I want to have functionality to add an item to the menu list, so that the customer can order new items.	4/4
15	As a receptionist, I want to have an order list that shows incomplete orders, so that I can see what I need to do next.	Is covered by user story 12

16	As a receptionist, I want to be able to complete an order, so that the customer can see the order on the order screen.	2/2
17	As a receptionist, I want to have functionality to delete items from the menu, so that I can remove unwanted menu items.	2/2
18	As a receptionist, I want to be able to specify ingredients when creating an item, so that the customer sees ingredients in the item.	Is covered by user story 14
19	As a receptionist, I want to be able to specify an item to be customizable, so that items which have multiple ingredients can be customized by customers.	2/2
20	As a receptionist, I want to divide the items by specifying a group name, so that the menu is organized.	Is covered by user story 14
21	As a receptionist, I want to have functionality to print order info, so that the order can be shared among workers outside of the system.	1/1

7 Conclusions (Rokas)

The goal of the project was to create a system that would help smaller stores make the process of ordering faster and more precise reducing the queues and that way improving the revenue of the store.

The analysis sections 21 user stories were created to guide trough what each user of the system could be capable of doing. Further diagrams were created to clarify such as the use case diagram, which visualized what each user could do and the domain model which gave specific domain concepts to focus on that helped go into the design of the system. To accompany the use case diagram, use case descriptions were made to give an in-depth description of how the users would act in each scenario.

The design section shows of the architecture of the system, the GUI design choices. various patters that were used and technologies, particularly the MVVM architectural pattern and the JDBC database access points.

The implementation section shows of the code of the most important functionality of the system the make order along with a supplementary showcase of the get menu functionality.

The testing section shows what testing methodology was used, in particular black box testing. Examples of test cases are shown.

The results section shows what was accomplished compared to our main goal of the project and whether it was a success. Also, a list of all user stories that were completed is shown.

8 Project future (Rokas)

The project user stories have all been finished. There are features that could be improved or changed. There are places in the implementation that are messy and could use some cleaning up to improve future system expandability and maintenance. The menu could include pictures looking at it from the customer side which would improve customer experience greatly. The system could also use a possible redesign to make it look nicer. The order printing functionality could be heavily improved as it is very basic now.

The payment methods that are available right now could be implemented in the back end and expanded.

The system could be made more generic to serve as total ordering system instead of only food. This would mean the point of sales system could be used in any shop.

There is a potential of the system being upgraded to help manage things such as finances. It could track how much was made from sales of products and what was the profit margin. This would mean a more complex item creation method would be needed.

The system could track storage of ingredients at shop. Letting a manager receive ingredients and each order slowly take them out of the system. Notifications would be sent for low stock.

The system could have employee management functionality or even branch tracking functionality. This would mean the kitchen back end could be implemented sending orders directly to stationed worker screens. The kitchen backend would replace the order printing functionality.

The system could be expanded to let the customers do shopping online for deliveries or ordering from a distance. Could even be a mobile app for the client. If this were to happen the system networking should be switched for an API connection to reduce stress on the server.

The system is not very expandable at its present state. As it is meant for small to medium businesses that do not have multiple branches.

9 Sources of information

Larman, C., 1997. *Applying UML and patterns.* s.l.:s.n.

LAUV team, u.d. *self-ordering systems.* [Online]
Available at: https://lavu.com/be-more-efficient-self-ordering-restaurant-technology/
[Senest hentet eller vist den 10 08 2020].

Prange, J., u.d. *Why Self Ordering Kiosks are Becoming the Secret Weapon for Successful Restaurants.* [Online]
Available at: https://www.touchbistro.com/blog/why-self-ordering-kiosks-are-becoming-the-secret-weapon-for-successful-restaurants/
[Senest hentet eller vist den 10 7 2020].

VIA Engineering, in preparation. *Confidential Student Reports,* s.l.: s.n.

10 Appendices

- Appendix A - Project Description
- Appendix B - Astah Diagrams
  - Class diagram with functions
  - Class diagram without functions
  - Domain model
  - ER diagram
  - Sequence diagram for making order
  - USeCase Diagram
- Appendix C - Use Case Description
- Appendix D - User Manual
  - Self-ordering point of sales - user manual
- Appendix E - Source Code
- Appendix F - Use Case Testing

Process Report

Marwan Summakieh (285805 ICT)

Rokas Barasa (285047 ICT)

Character count: 38372 characters

Software engineering ICT

2nd semester

13-08-2020

Table of content

1 Introduction

in this report we describe how the group was formed, worked, and ended during the repeating period of the 2nd semester project of the ICT course at VIA University collage.

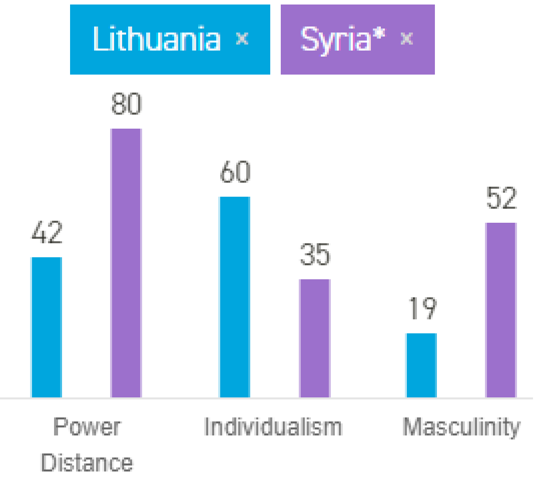
The project covered many aspects of the knowledge we acquired during the semester and some personal experiences that we collected during the 3rd semester.

The project period took place in summer 2020 June-August where we had to fit an entire semester worth of work in a month and half.

2 Group Description (Marwan)

Rokas	20	Lithuanian	Self-taught programmer with previous university experience with coding. High sense of responsibility, diligent worker, both inside and outside of university
Marwan	24	Syrian	Started as a self-taught JS programmer, tries to see the big picture, and find the best approach to achieve the group goals. Sometimes this might create a state of ignorance to the group's capabilities. Focusses on the harmony of work and a healthy work environment

2.1 Culture/Hofstede



Our group has two different nationalities: each with their own customs, traditions, mindset, perceptions of power, and ways of doing things. We come from different cultures and it goes without saying that the culture clash can impact the group dynamic

As part of our SSE (Study Skills for Engineering students) course from the 1st semester, we have been familiarized with Geert Hofstede and his cultural dimensions theory. By using his online platform for comparing the countries in our group we got the following results as we choose three - “Power distance”, “Individualism” and “Masculinity” as the most impactful traits for our group work and we will prospect each one below:

2.1.1 Power Distance

However, Lithuania show tendencies to prefer equality and a decentralization of power and decision-making preference for teamwork and an open management style. Of all of those three countries Syria has the biggest score of them all which means that people accept a hierarchical order in which everybody has a place, and which needs no further justification.

In our group Power Distance expresses attitude towards inequalities and hierarchies, societies with lower score believe in minimalizing inequalities, while those with a higher score accept their role in the society. It was not a major contributing factor in the group.

2.1.2 Individualism

Although Syria scores a low 35 on individualism, it is seldomly apparent in our group. The group tends to lean towards a higher score of individualism, which correlates to the higher Hofstede scores of Ireland (74) and Lithuania (60) which accentuates the aversion of being controlled and told what to do.

This becomes apparent in our group when the members are expected to take care of themselves and take responsibility for their assigned tasks, though it is still encouraged to ask for help when someone feels stuck on specific task. Project and technical talk are the most prevalent in our group during assigned project work time, as small talk is usually kept at a minimum during those times.

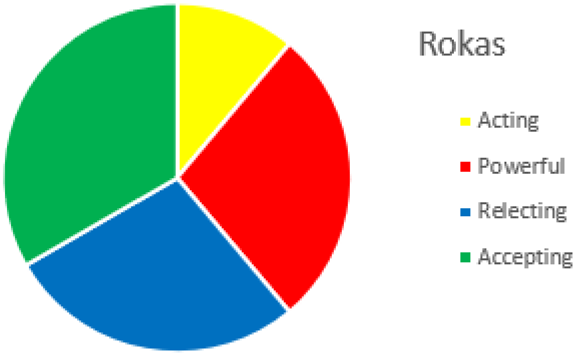
2.1.3 Masculinity

As for the Hofstede definition of masculinity, two countries of our group are above of the category of “feminine” cultures. It indicates that the society are driven by competition, achievement, and success, with success being defined by the best in field. Looking into our group it is visible that people who are from these countries puts a lot of effort to keeping workload on track, making tasks on time and seek for the best. Unlike Syria, Lithuanians are from a “feminine” culture which means more taking care of others and tending to keep a low profile, and usually communicate with a soft and diplomatic voice in order not to offend anyone.

2.2 E-stimate / Personal profiles

In order to assess our qualities shortcomings, openings and dangers we chose to break down an individual (personal) and a group analysis, based on a behavioral test evaluation (which was part of our SSE course in the first semester), comprising in deciding our conduct propensities. In this way we got an overview about how our group is performing and what kind of improvements we need to make to establish a flawless workflow. Our group consists of members who have different cultural and educational background; therefore, communication and dividing tasks accordingly is essential. In this way we are exchanging knowledge and all the members are involved in the process report. All of us are self-motivated and hard-working. We are meeting every day for more than a few hours to see how things are coming together, even when the schedule is tight. For a better interaction and backup plan, we are all communicating via Facebook group and we are sharing all the files and documents through Microsoft Teams and GitHub.

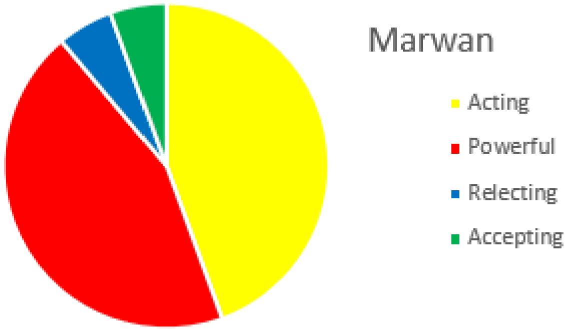
2.2.1 Rokas Barasa



My chart demonstrates me accurately. I cooperate with the group very well and when it is needed, I take charge. I am open to new ideas if they seem realizable. I am very stubborn when It comes to big risky decisions if I know the person suggesting them does not have experience.

- **Strengths**: Detailed orientated work, revising completion criteria, and forming ideas for dealing with problems, assertive decision making when seen to be required for the greater good of the group. Detailed work, problem solving, global thinking, group management.
- **Weaknesses**: Fear of failing, stubbornness, taking control of ideas.
- **Opportunities**: Working in areas where I have little to no previous experience. Letting go of control.
- **Threats**: Perfectionism, micromanagement, panic if the team is not performing.

2.2.2 Marwan Summakieh



My chart demonstrates how I was a year a go although some of the characteristics are accurate a lot of them have changed throughout the year, even though I am still very acting I am not as powerful this part has been reduced significantly and the accepting part was increased, that was because of an attempt to make working with other people smoother and healthier.

- **Strengths:** thinking outside of the box, accepting criticism, determined but also patient and lenient, can control panicking situations.
- **Weaknesses:** reluctant when asking for help.
- **Opportunities:** manages the flow of the group, and acts as an adaptor to ensure harmonious work.
- **Threats:** work stops because of reluctant to asking for help, can't control myself when patience runs out.

3 Project Initiation (Rokas)

As this is our second attempt for this project, we came to a clearer understanding on how we should tackle the problems that faced us.

When we started thinking about the project our supervisors gave us some pointers to what the system would need for this project. These requirements were

- Client-server system (RMI/Sockets)
- Database
- Multiuser types
- Use UP and Scrum

Using these requirements, we came up with two projects that are very similar to each other.

3.1 Food ordering system, mobile version

The idea was derived from another idea in the second semester. The original idea was a pizza store management system though up by Rokas. It was very flawed; it was too big and would not have been a good project. It was constrained, made more specific and that became the mobile food ordering system. It was very simple, only two different actors, receptionist, and customer.

The system would have features for the customer such as paying by card, order completion notifications and order customization. The receptionist would be able to manage orders and the menu.

3.2 Food ordering system, in store point of sale.

The point of sale food ordering system idea was made by Rokas and was derived as an alternative to the mobile version. Intended as a secondary idea for the project. The system would have been more complex. Having three users instead of two and being more focused on the actual location, like a restaurant or a fast food place. It was much like the first idea. It had customizable items and two of the users doing the same functionality as in the mobile version. What made this idea interesting was the addition of the third user, the order screen which showed completed and incomplete orders.

3.3 Chosen project

A vote was made, and the second idea won 3 to 1. The point of sale version of the system seemed more interesting to everyone in the group. The only one who did not want to choose this idea was Rokas, who tough the system would be too big at its core for the team's experience to handle.

Everyone had a different understanding of how the system would be. One of the criticisms of the idea was that the idea did not consider how a kitchen works in the back end. The product owner decided to heavily constrain the idea to make It only focus on the ordering of food and the order completion and to have kitchen backend taken care of by being able to print orders. This made sense to all team members.

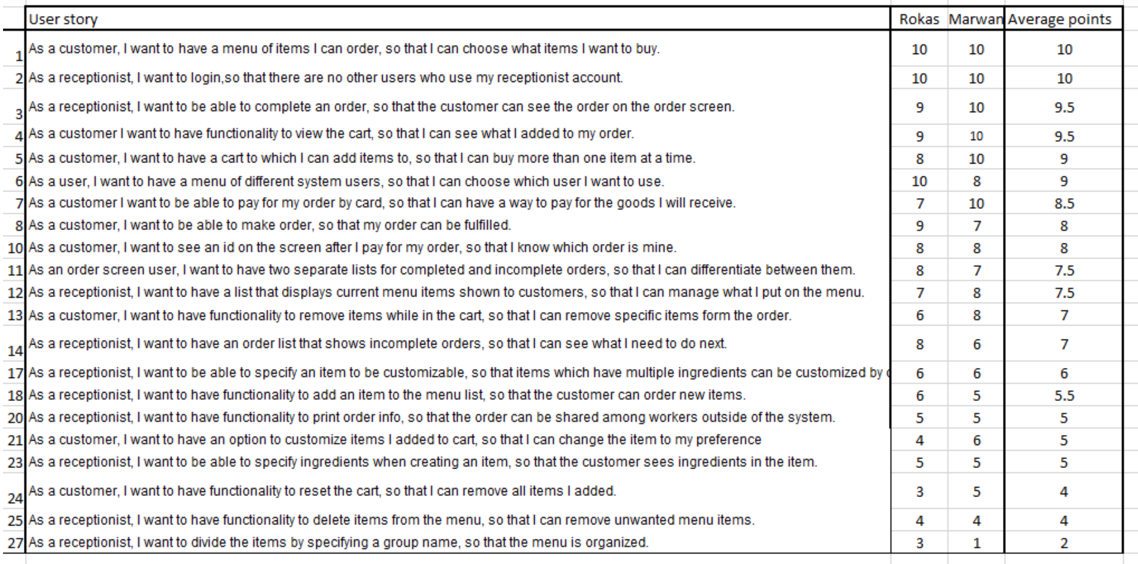
4 Project Description (Rokas)

Before starting the project description, we discussed about our goals from this project along with our strengths and weaknesses. After a quick one-week vacation everyone got back to working on the description. This part of the project was essential for figuring out if the project idea was viable and determining the scope of the project. Team members were poking holes at our idea, trying to make it more realistic. Research was done into inner workings of the kitchen, advantages of point of sales systems and the effect of customer lines on profit. The result was an idea that had less pitfalls than the starting idea. Although there was a problem, due to having too much time to work with there was lack of knowing when to finish the project description. That meant the members were sitting on it for too long.

The first draft of the requirements was made by Rokas who came up with the idea, the rest of the team gave feedback and filled in gaps. Domain model and use case diagram tasks were given to members other than the product owner to give them a better understanding of the system we would be making. The product owner gave feedback on their representation of the system in these diagrams. Throughout this process the team was missing deadlines it had set. The team was having problems with Hugh’s and Smilte’s productivity which put stress on the remaining members who waited for these members to do something before the sprints began.

5 Project Execution (Marwan - little bit Rokas)

Before starting the scrum process, we made a backlog that showed most important features to the system. Ten being the highest rating and most important, one being the lowest rating and least important to the system. Each member rated each user story separately. An average score was made using all the ratings from the team members, unfortunately only two members only rated them.



	User story	Rokas	Marwan	Average points
1	As a customer, I want to have a menu of items I can order, so that I can choose what items I want to buy.	10	10	10
2	As a receptionist, I want to login,so that there are no other users who use my receptionist account.	10	10	10
3	As a receptionist, I want to be able to complete an order, so that the customer can see the order on the order screen.	9	10	9.5
4	As a customer I want to have functionality to view the cart, so that I can see what I added to my order.	9	10	9.5
5	As a customer, I want to have a cart to which I can add items to, so that I can buy more than one item at a time.	8	10	9
6	As a user, I want to have a menu of different system users, so that I can choose which user I want to use.	10	8	9
7	As a customer I want to be able to pay for my order by card, so that I can have a way to pay for the goods I will receive.	7	10	8.5
8	As a customer, I want to be able to make order, so that my order can be fulfilled.	9	7	8
10	As a customer, I want to see an id on the screen after I pay for my order, so that I know which order is mine.	8	8	8
11	As an order screen user, I want to have two separate lists for completed and incomplete orders, so that I can differentiate between them.	8	7	7.5
12	As a receptionist, I want to have a list that displays current menu items shown to customers, so that I can manage what I put on the menu.	7	8	7.5
13	As a customer, I want to have functionality to remove items while in the cart, so that I can remove specific items form the order.	6	8	7
14	As a receptionist, I want to have an order list that shows incomplete orders, so that I can see what I need to do next.	8	6	7
17	As a receptionist, I want to be able to specify an item to be customizable, so that items which have multiple ingredients can be customized by	6	6	6
18	As a receptionist, I want to have functionality to add an item to the menu list, so that the customer can order new items.	6	5	5.5
20	As a receptionist, I want to have functionality to print order info, so that the order can be shared among workers outside of the system.	5	5	5
21	As a customer, I want to have an option to customize items I added to cart, so that I can change the item to my preference	4	6	5
23	As a receptionist, I want to be able to specify ingredients when creating an item, so that the customer sees ingredients in the item.	5	5	5
24	As a customer, I want to have functionality to reset the cart, so that I can remove all items I added.	3	5	4
25	As a receptionist, I want to have functionality to delete items from the menu, so that I can remove unwanted menu items.	4	4	4
27	As a receptionist, I want to divide the items by specifying a group name, so that the menu is organized.	3	1	2

Check appendix – C for the full backlog

This list was updated multiple times in the sprint period and is not the same as the backlog before starting sprints. The backlog was recalculated multiple times as after each sprint we gained more insight into the importance of some user stories. The user stories were ordered by score given by each team member that was then averaged. The final choice of which user story to do was still based on what the product owner deemed important to the system.

Before the sprint period some issues arose. The team decided to split and remove two team members from the team. This decision was made because these team members were slowing down progress significantly by not communicating, not following deadlines, not contributing to the analysis section of the project. We took this issue very seriously because we worked with one of these people before, they did not seem to have improved their abilities or solved their weak points from previous projects. The split happened at the date 19-07-2020 16:00. The *Microsoft Teams* group we had made was archived so that everyone could still access it but not edit it. Both teams had access to the files that we made as a team up until the split. There were no other requirements specified on group splitting in the group contract.

5.1 Tools

5.1.1 GitHub

For version control we used GitHub which we are very familiar with. We did not have any problems with it as we used to in the previous semesters.

5.1.2 Jira

For planning the scrum process the team used Jira to help keep track of the scrum process, it was useful to keep track of sprints, issues to be solved, efforts of team members, and burndown charts.

5.1.3 Discord

Discord was the main communication method because of its reliability and smoothest screen sharing feature compared to other apps like skype and Facebook messenger

5.1.4 Teams

Teams was used to create an official group so we can use SharePoint feature, that allowed us to edit and save files that needed to be accessed and modified by all team members.

5.2 Sprint 1

5.2.1 Sprint Planning

Date: 20-07-2020 09:36

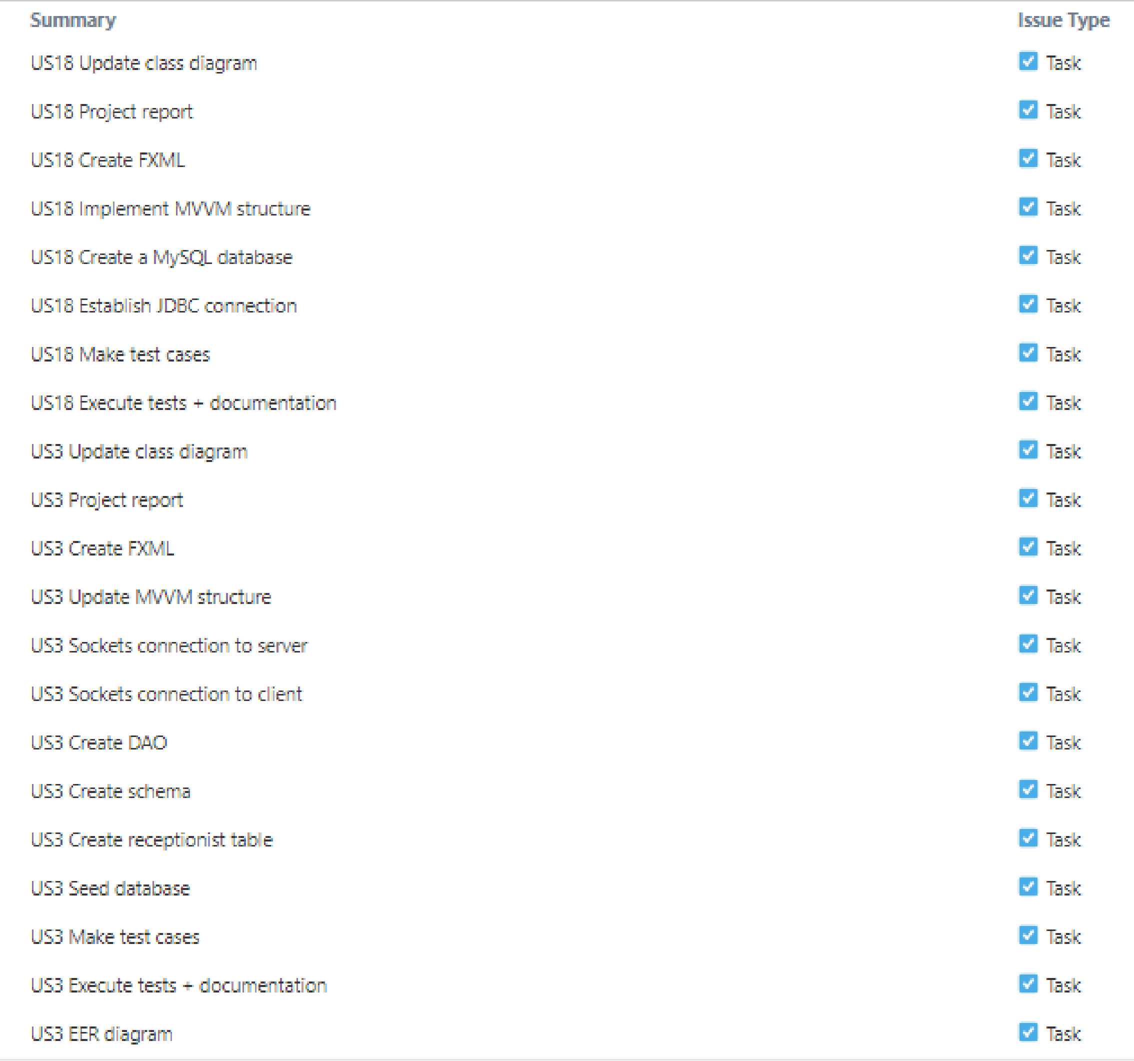
Present: Rokas, Marwan

Minutes:

- Discussed the definition of done: A user story is finished when it is fully tested and documented.
- Discussed what user stories to pick
- Picked user story - As a **user**, I want to have a menu of different system users, so that I can choose which user I want to use.
- Picked user story - As a **receptionist**, I want to login, so that there are no other users who use my receptionist account.
- Broke down user stories into tasks. See appendix D

Sprint backlog:

Who is assigned and hours can be seen in work breakdown structure.



Summary	Issue Type
US18 Update class diagram	☑ Task
US18 Project report	☑ Task
US18 Create FXML	☑ Task
US18 Implement MVVM structure	☑ Task
US18 Create a MySQL database	☑ Task
US18 Establish JDBC connection	☑ Task
US18 Make test cases	☑ Task
US18 Execute tests + documentation	☑ Task
US3 Update class diagram	☑ Task
US3 Project report	☑ Task
US3 Create FXML	☑ Task
US3 Update MVVM structure	☑ Task
US3 Sockets connection to server	☑ Task
US3 Sockets connection to client	☑ Task
US3 Create DAO	☑ Task
US3 Create schema	☑ Task
US3 Create receptionist table	☑ Task
US3 Seed database	☑ Task
US3 Make test cases	☑ Task
US3 Execute tests + documentation	☑ Task
US3 EER diagram	☑ Task

5.2.2 Daily Scrum 2

Date: 21-07-2020 09:00

Present: Rokas, Marwan

Meeting lenght: 5 min

Marwan – Had set up database and GitHub. Today he will do JDBC and start EER diagram.

Rokas – Did MVVM for both user stories, finished all FXML files for this sprint, did class diagram. Going to do test cases for login and continue work on networking. Having a lot of rabbit hole problems with sockets, considering using RMI.

After meeting discussion – Sockets vs RMI, amount of work needed for both.

5.2.3 Daily Scrum 3

Date: 22-07-2020 16:35

Present: Rokas, Marwan

Meeting lenght: 7 min

Marwan – Was working on JDBC, reviewed use cases. No problems. Will finish JDBC, update project report and do RMI layer.

Rokas – Finished network RMI connection, finished login test cases, finished MVVM implementations. Will do project report, class diagram.

After meeting discussion – Rerating user stories, changes to user story wording.

5.2.4 Daily Scrum 4

This daily scrum was skipped because the team members were thinking sprint review and retrospective would be the daily part. They were wrong. This is fixed in the next sprint.

5.2.5 Sprint Review

Date: 23-07-2020 22:30

Present: Rokas, Marwan

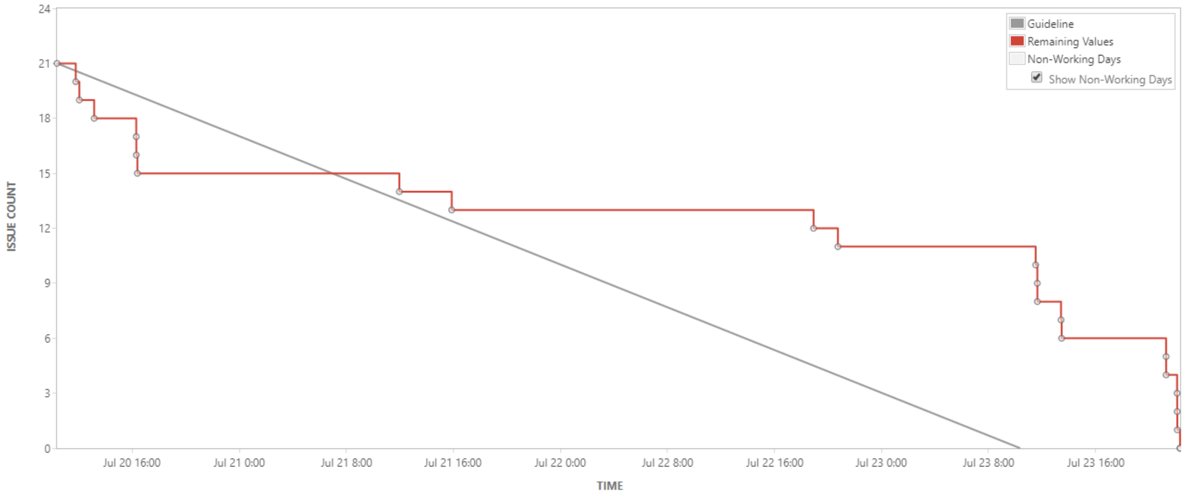
Minutes:

- Considering if the sprint was successful: Success
- What was accomplished: Login and user select functionality
- Burndown chart
- Burndown chart issue

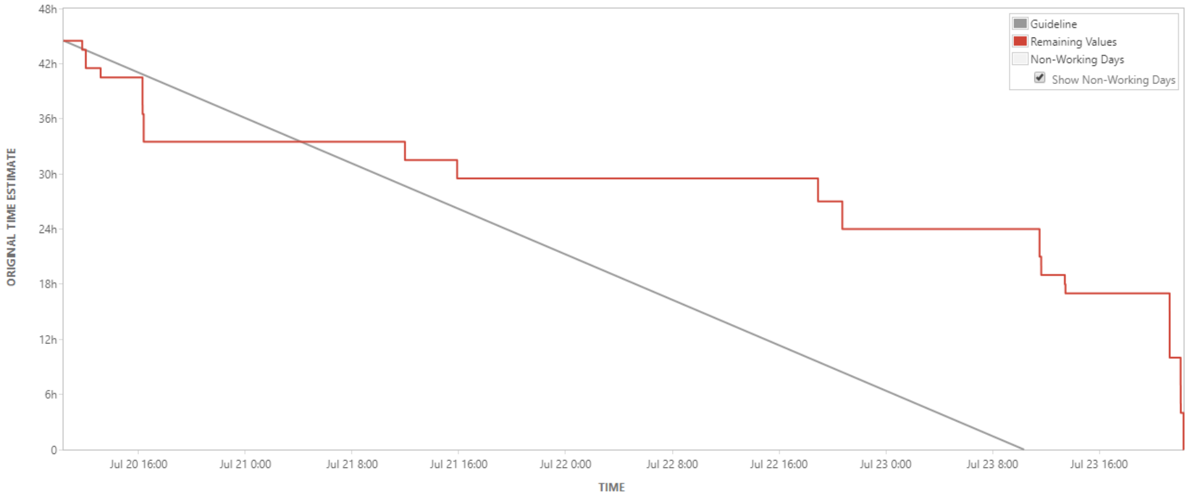
5.2.5.1 Burn down charts

We made a mistake in setting up the end of the sprint. We specified it to end at 12 AM instead of 12 PM by accident. Therefore, it looks like we were working after the sprint has ended.

5.2.5.2 By issue:



5.2.5.3 By hours:



5.2.6 Sprint Retrospective

Date: 23-07-2020 23:00

Present: Rokas, Marwan

Minutes:

- Start doing: Things for Communication. Asking for help when needed. Notifying when something important is done. Start using git desktop for repository instead of IntelliJ.
- Things to continue: Assigning tasks in a way that no one steps on each other’s toes. Continue our way of using work breakdown.
- Things to stop. Thinking how small things are going to affect the whole project rather than user story. Going solo, not asking for help, do not do all the work in the last day.

5.3 Sprint 2

5.3.1 Sprint Planning

Date: 20-07-2020 09:36

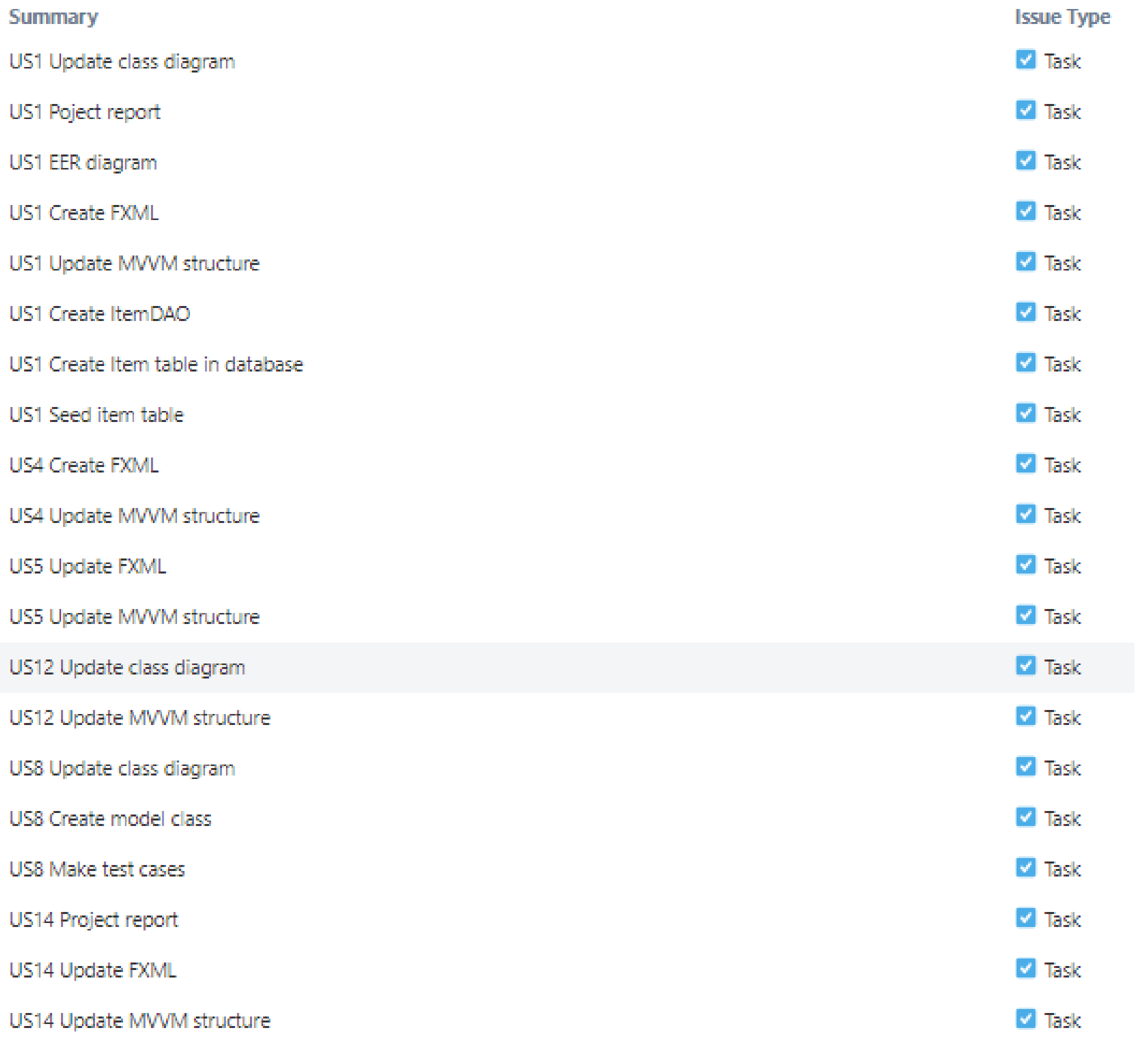
Present: Rokas, Marwan

Minutes:

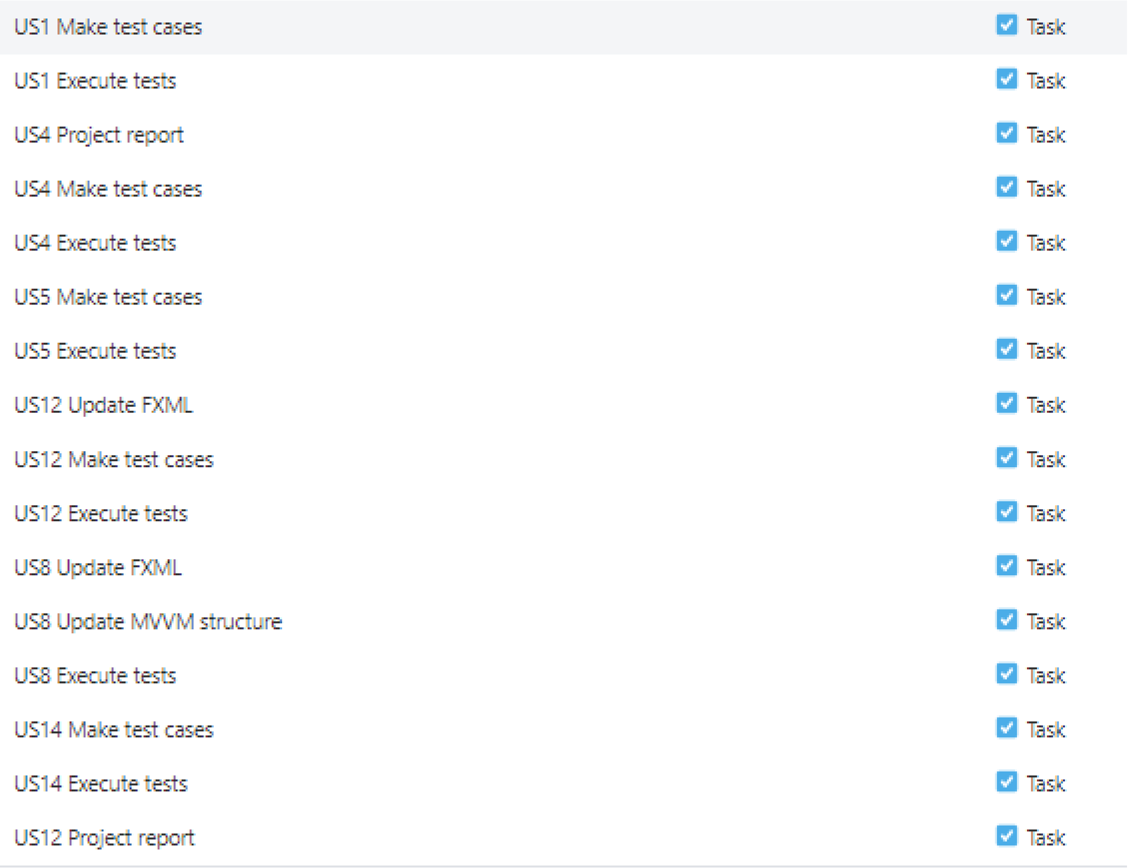
- Discussed what user stories to pick
- Picked user stories 1, 4, 5, 8, 12, 14.
- Broke down user stories into tasks. See appendix D

Sprint backlog:

Who is assigned and hours can be seen in work breakdown structure.



Summary	Issue Type
US1 Update class diagram	☑ Task
US1 Poject report	☑ Task
US1 EER diagram	☑ Task
US1 Create FXML	☑ Task
US1 Update MVVM structure	☑ Task
US1 Create ItemDAO	☑ Task
US1 Create Item table in database	☑ Task
US1 Seed item table	☑ Task
US4 Create FXML	☑ Task
US4 Update MVVM structure	☑ Task
US5 Update FXML	☑ Task
US5 Update MVVM structure	☑ Task
US12 Update class diagram	☑ Task
US12 Update MVVM structure	☑ Task
US8 Update class diagram	☑ Task
US8 Create model class	☑ Task
US8 Make test cases	☑ Task
US14 Project report	☑ Task
US14 Update FXML	☑ Task
US14 Update MVVM structure	☑ Task



US1 Make test cases	☑ Task
US1 Execute tests	☑ Task
US4 Project report	☑ Task
US4 Make test cases	☑ Task
US4 Execute tests	☑ Task
US5 Make test cases	☑ Task
US5 Execute tests	☑ Task
US12 Update FXML	☑ Task
US12 Make test cases	☑ Task
US12 Execute tests	☑ Task
US8 Update FXML	☑ Task
US8 Update MVVM structure	☑ Task
US8 Execute tests	☑ Task
US14 Make test cases	☑ Task
US14 Execute tests	☑ Task
US12 Project report	☑ Task

5.3.2 Daily scrum 1

Meeting information

Location:	Online
Date:	27.7.2020
Time:	11:30
Attendees:	Marwan, Rokas
Duration:	7 min

Daily meeting content

Marwan – Updated EER diagram, created item table in the database, started working on itemDAO. Will make and execute test cases

Rokas – Finished FXML diagram for all user stories for this sprint, finished all MVVM implementation except making order, finished class diagrams for use case 12 and 8. Will start doing test cases for user story 8 and project report for US1 and 14. No problems faced.

After meeting discussion – Third normal form in the database.

5.3.3 Daily scrum 2

Location: Online

Date: 22.7.2020

Time: 15:46

Attendees: Marwan, Rokas

Duration: 8 min

min

Daily meeting content

Marwan – ItemDAO and IngredienDAO, updated EER diagram, created Item table in database and seeded it, problem with getting arraylist into and from the database. Going to finish item DAO completely

Rokas – Test cases. No problems. Going to work on Project report.

After meeting discussion – Rerating user stories. Decided to remove user story 15 from backlog and requirements.

5.3.4 Daily scrum 3

Meeting information

Location:	Online
Date:	29.7.2020
Time:	10:20
Attendees:	Marwan, Rokas
Duration:	5 min

Daily meeting content

Marwan – Finished tables in database and updated the ER diagram, worked on ItemDAO and IngredientDAO, started on the model. Having problems retrieving data from the database and link the items with their appropriate ingredients. Will continue working on the model and project report.

Rokas – Did process report part, description, and execution, worked on project report, no problems. Will continue working on project report.

After meeting discussion – model in the server has multiple features

5.3.5 Sprint review

Starting Time: 22:57 29.7.2020

Duration: 1h

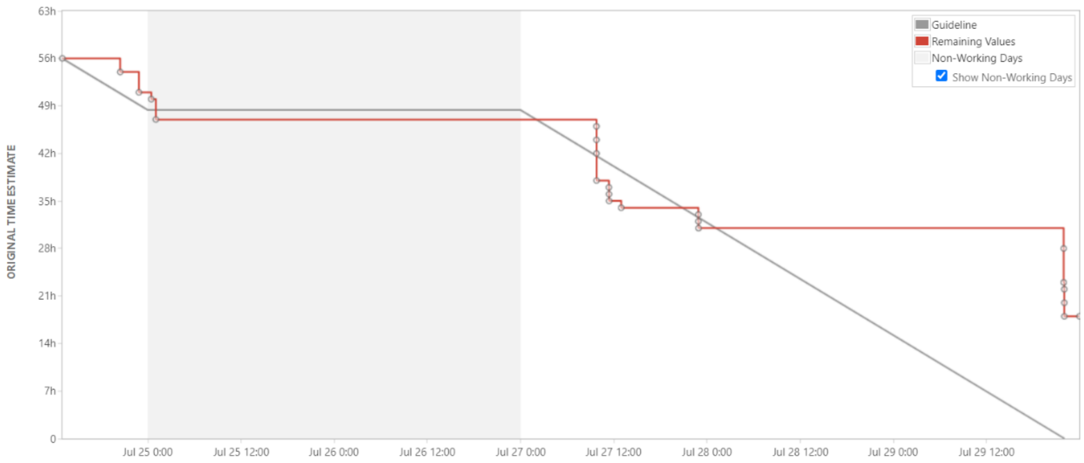
Members: Marwan, Rokas.

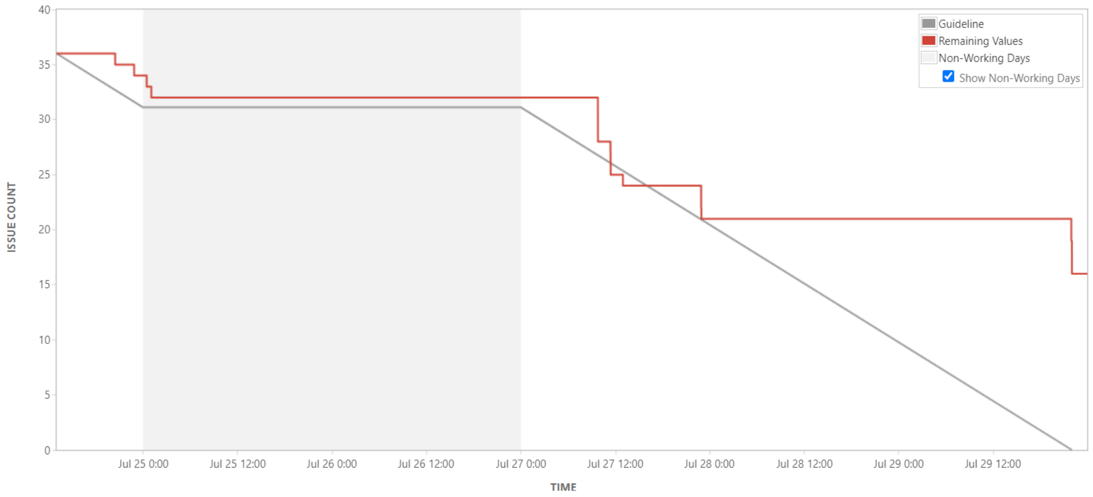
Minutes

- Sprint productivity conclusion: not productive.
- Reasons for the sprint productivity status and the result.

- User stories completion: no user story fully completed.
- All user stories need testing.
- US8 needs back end implementation.
- Sprint 2 was not successful.
- Upgraded our understanding to the system as a result we removed US15: As a customer, I want to have a quantity counter on every item in the cart

5.3.6 Sprint burndown chart





5.3.7 Sprint retrospective

Starting Time: 23:57

Members: Marwan, Rokas.

Duration: 20 min

- Things to continue doing: no conflicts, we did not have communication problems.
- Things we should start doing asking for in depth help. Looking back on the retrospective and learn from it
- Things we should stop doing stop hording tasks.

The second sprint was not as successful as we anticipated

5.4 Sprint 3

5.4.1 Sprint planning:

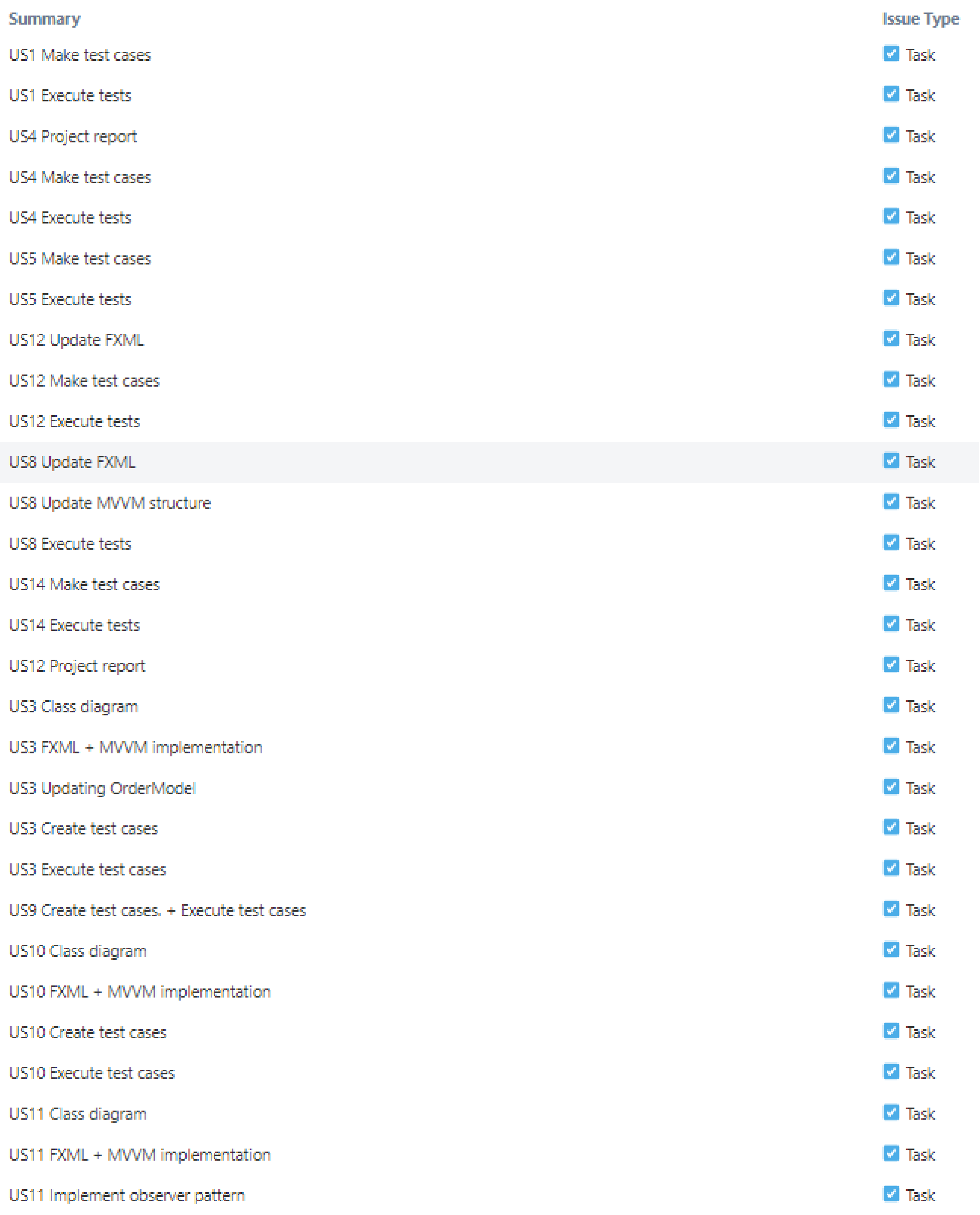
Starting time: 11:41 30/July

Participants: Marwan, Rokas

- Discussed the unfinished user stories
- Why does it make sense to add more user stories? The tasks that were not completed are very simple.
- Estimated the time required for the unfinished tasks.
- Added new user stories to the sprint with time estimates.
- Calculated time assigned to each member then assigned tasks accordingly. See appendix D

Sprint backlog:

Who is assigned and hours can be seen in work breakdown structure.



Summary	Issue Type
US1 Make test cases	☑ Task
US1 Execute tests	☑ Task
US4 Project report	☑ Task
US4 Make test cases	☑ Task
US4 Execute tests	☑ Task
US5 Make test cases	☑ Task
US5 Execute tests	☑ Task
US12 Update FXML	☑ Task
US12 Make test cases	☑ Task
US12 Execute tests	☑ Task
US8 Update FXML	☑ Task
US8 Update MVVM structure	☑ Task
US8 Execute tests	☑ Task
US14 Make test cases	☑ Task
US14 Execute tests	☑ Task
US12 Project report	☑ Task
US3 Class diagram	☑ Task
US3 FXML + MVVM implementation	☑ Task
US3 Updating OrderModel	☑ Task
US3 Create test cases	☑ Task
US3 Execute test cases	☑ Task
US9 Create test cases. + Execute test cases	☑ Task
US10 Class diagram	☑ Task
US10 FXML + MVVM implementation	☑ Task
US10 Create test cases	☑ Task
US10 Execute test cases	☑ Task
US11 Class diagram	☑ Task
US11 FXML + MVVM implementation	☑ Task
US11 Implement observer pattern	☑ Task

US11 Implement call for unfinished orders	☑ Task
US11 Implement call for finished orders	☑ Task
US11 Create test casees	☑ Task
US11 Execute test cases	☑ Task
US13 MVVM implementation	☑ Task
US13 Create test cases	☑ Task
US13 Execute test cases	☑ Task

5.4.2 Daily scrum 1

Location: Online

Date: 27.7.2020

Time: 11:00

Attendees: Marwan, Rokas

Duration: 15 min

Daily meeting content

Marwan – Making test and Execution for US1, US4 and US5. Working on US12 implementing

Rokas – Did FXML + MVVM implementation for US3, US10, US11 and US13, then did class diagram for US3.

Going to do class diagram for US10 and US11. Then create test cases and execute tests for US11 and US13

After meeting discussion – How the model in the server would work.

5.4.3 Daily scrum 2

Meeting information

Location: Online

Date: 03.8.2020

Time: 9:28

Attendees: Marwan, Rokas

Duration: 5 min

Daily meeting content

Marwan – Worked on calling finished and unfinished orders in the server. The OrderModels had some errors. Going to do Test cases and finish calling for finished and unfinished orders

Rokas – Did class diagram for US10 and US11, did Testing for US11 and US13. No problems. Going to Execute tests for US14.

After meeting discussion – Making test cases for US14.

5.4.4 Daily scrum 3

Meeting information

Location: Online

Date: 29.7.2020

Time: 9:00

Attendees: Marwan, Rokas

Duration: 13 min

Daily meeting content

Marwan – Finished test cases US12, 14 and 3, didn't yet finish the OrderModel, There is an error with getting the orders from RmiHandler. Finish tests and OrderModel.

Rokas – Went through process report. Could not do execution because the OrderMaodel was not finished, no problems, finish test execution for US14 if the OrderModel is finished.

After meeting discussion – the group decided to rename a US16 as a customer I want to have a limit on how many completed orders can be shown, so that the completed orders do not overpopulate the list.

5.4.5 Review

Location: online.

Participants: Rokas, Marwan.

Starting Time: 21:25 29.7.2020

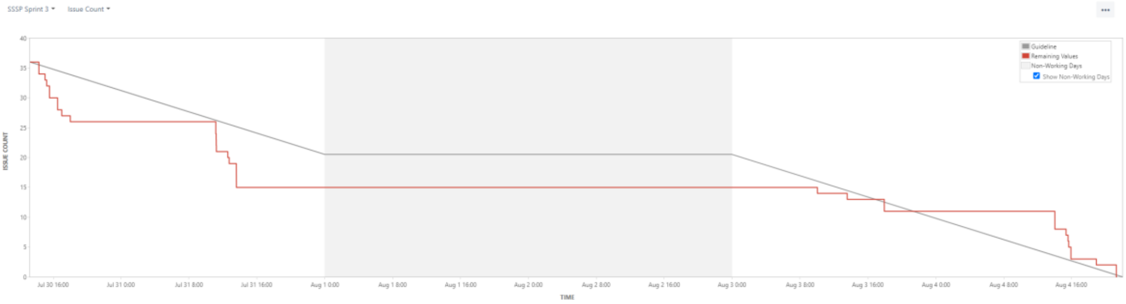
Duration: 30 minutes

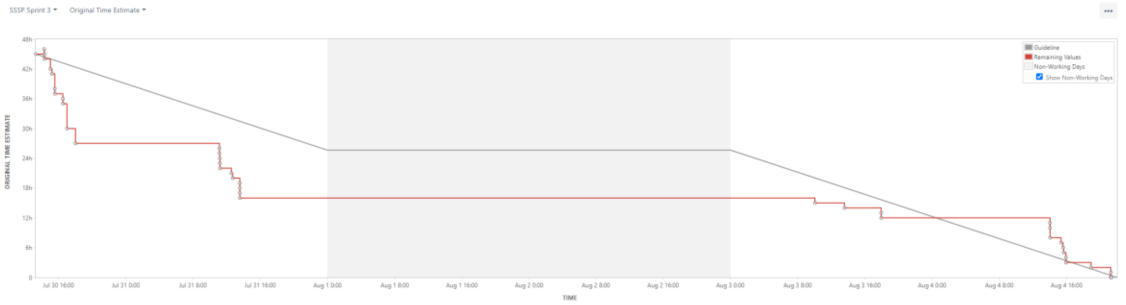
Minutes:

- What we did: order screen, giving IDs to orders when making them, completing orders.
- User stories completion:
  - As a customer, I want to have a menu of items I can order, so that I can choose what items I want to buy
  - As a customer I want to have functionality to view the cart, so that I can see what I added to my order
  - As a customer, I want to have a cart to which I can add items to, so that I can buy more than one item at a time Improved our understanding of the system: renamed user story 16.
  - As a receptionist, I want to have a list that displays current menu items shown to customers, so that I can manage what I put on the menu.

- As a customer, I want to be able to make order, so that my order can be fulfilled
- As a receptionist, I want to have an order list that shows incomplete orders, so that I can see what I need to do next
- As a receptionist, I want to be able to complete an order
- As an order screen user, I want to have both order lists sorted top to bottom by number size
- As a customer, I want to see an id on the screen after I pay for my order
- As an order screen user, I want to have two separate lists for completed and incomplete orders
- As a customer, I want to have functionality to remove items while in the cart

5.4.5.1 Burndown chart





5.4.6 Retrospective

Starting Time: 21:55 29.7.2020

Members: Marwan, Rokas.

Duration: 18 minutes.

Minutes:

- Things to continue doing: Asking for help, communicating.
- Things we should start doing: Asking for help even faster, better comments while committing to branches, commit daily to each personal branch, synchronize efficiency between members.
- Things we should stop doing thinking on how to do a task differently too much, instead of just doing it.

5.5 Sprint 4

5.5.1 Pre-planning:

Minutes:

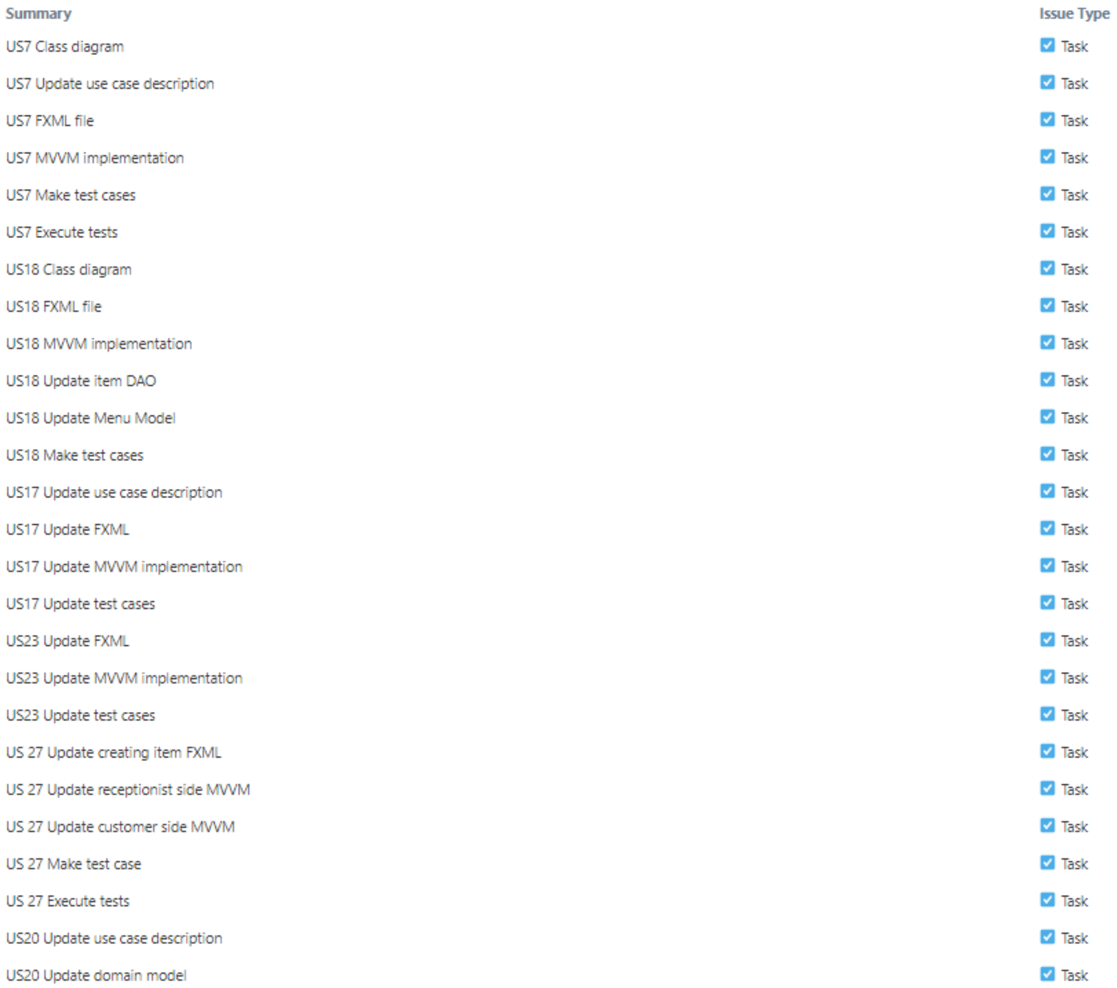
Starting time: 8:20 05/08

Participants: Marwan, Rokas

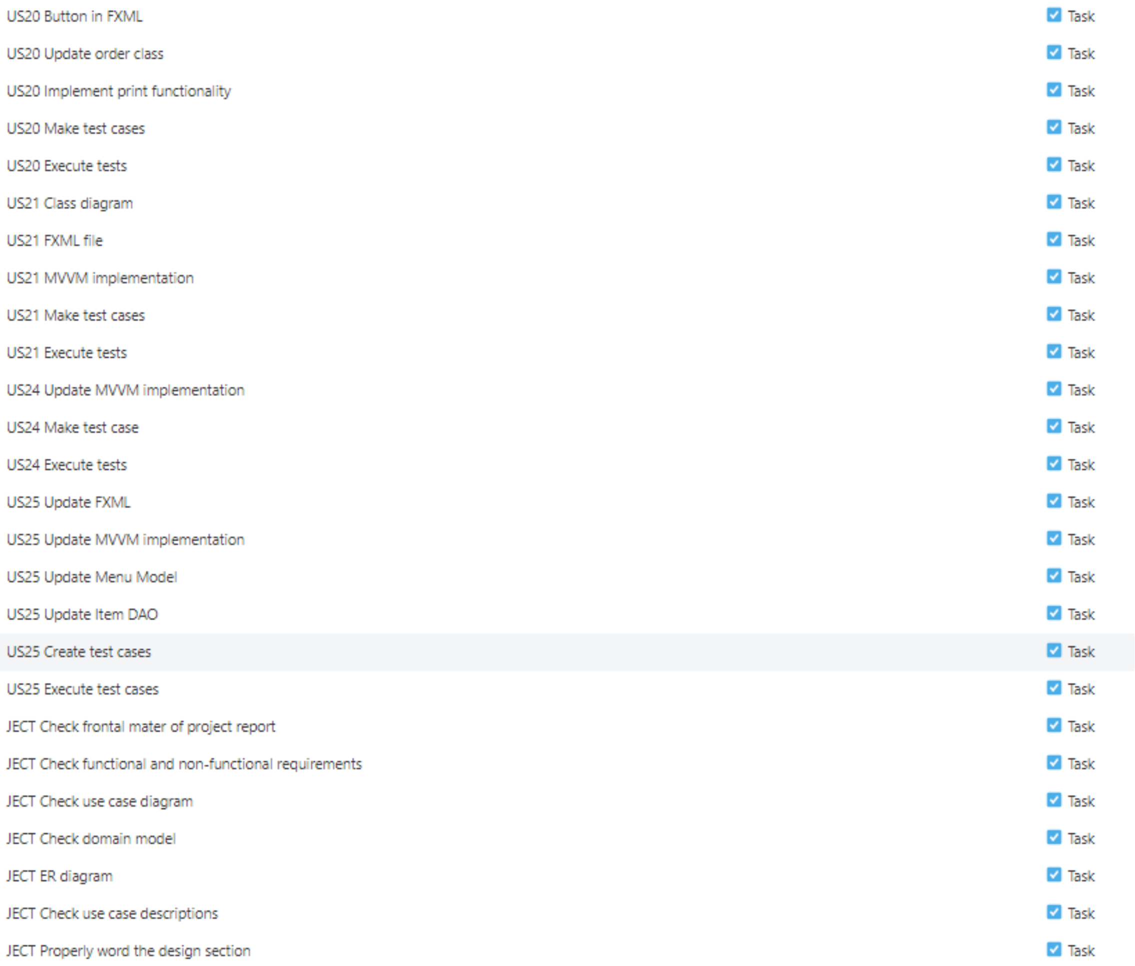
- Went through backlog to pick new user stories
- Discovering that some of the backlog user stories are non-functional requirements
- Contacted super visor for clearer understanding of the non-functional
- Picked user stories
- Broke down the user stories into tasks
- Took a break 10.00 – 10.30
- Later discovered another non-functional requirement from the user stories that we picked
- Calculated time assigned to each member then assigned tasks accordingly. See appendix D

Sprint backlog:

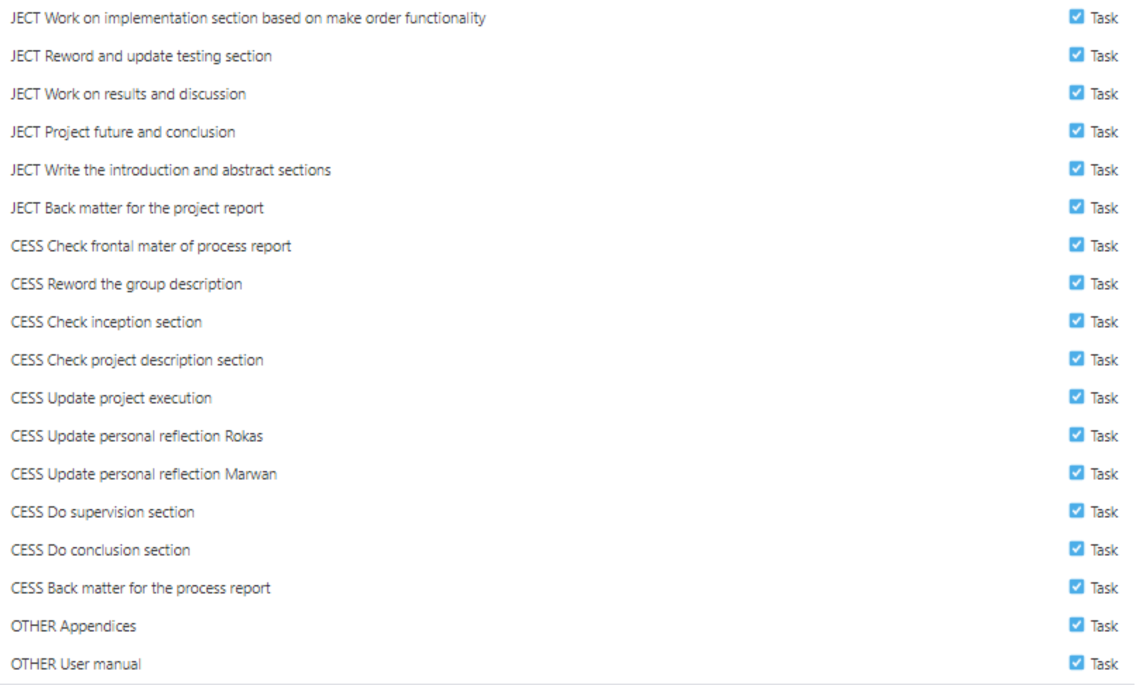
Who is assigned and hours can be seen in work breakdown structure.



Summary	Issue Type
US7 Class diagram	☑ Task
US7 Update use case description	☑ Task
US7 FXML file	☑ Task
US7 MVVM implementation	☑ Task
US7 Make test cases	☑ Task
US7 Execute tests	☑ Task
US18 Class diagram	☑ Task
US18 FXML file	☑ Task
US18 MVVM implementation	☑ Task
US18 Update item DAO	☑ Task
US18 Update Menu Model	☑ Task
US18 Make test cases	☑ Task
US17 Update use case description	☑ Task
US17 Update FXML	☑ Task
US17 Update MVVM implementation	☑ Task
US17 Update test cases	☑ Task
US23 Update FXML	☑ Task
US23 Update MVVM implementation	☑ Task
US23 Update test cases	☑ Task
US 27 Update creating item FXML	☑ Task
US 27 Update receptionist side MVVM	☑ Task
US 27 Update customer side MVVM	☑ Task
US 27 Make test case	☑ Task
US 27 Execute tests	☑ Task
US20 Update use case description	☑ Task
US20 Update domain model	☑ Task



US20 Button in FXML	☑ Task
US20 Update order class	☑ Task
US20 Implement print functionality	☑ Task
US20 Make test cases	☑ Task
US20 Execute tests	☑ Task
US21 Class diagram	☑ Task
US21 FXML file	☑ Task
US21 MVVM implementation	☑ Task
US21 Make test cases	☑ Task
US21 Execute tests	☑ Task
US24 Update MVVM implementation	☑ Task
US24 Make test case	☑ Task
US24 Execute tests	☑ Task
US25 Update FXML	☑ Task
US25 Update MVVM implementation	☑ Task
US25 Update Menu Model	☑ Task
US25 Update Item DAO	☑ Task
US25 Create test cases	☑ Task
US25 Execute test cases	☑ Task
JECT Check frontal mater of project report	☑ Task
JECT Check functional and non-functional requirements	☑ Task
JECT Check use case diagram	☑ Task
JECT Check domain model	☑ Task
JECT ER diagram	☑ Task
JECT Check use case descriptions	☑ Task
JECT Properly word the design section	☑ Task



JECT Work on implementation section based on make order functionality	☑ Task
JECT Reword and update testing section	☑ Task
JECT Work on results and discussion	☑ Task
JECT Project future and conclusion	☑ Task
JECT Write the introduction and abstract sections	☑ Task
JECT Back matter for the project report	☑ Task
CESS Check frontal mater of process report	☑ Task
CESS Reword the group description	☑ Task
CESS Check inception section	☑ Task
CESS Check project description section	☑ Task
CESS Update project execution	☑ Task
CESS Update personal reflection Rokas	☑ Task
CESS Update personal reflection Marwan	☑ Task
CESS Do supervision section	☑ Task
CESS Do conclusion section	☑ Task
CESS Back matter for the process report	☑ Task
OTHER Appendices	☑ Task
OTHER User manual	☑ Task

5.5.2 Daily scrum 1

Meeting information

Location: Online

Date: 06.08.2020

Time: 10:20

Attendees: Marwan, Rokas

Duration: 6 min

Daily meeting content

Marwan – Did ItemDAO. Faced a problem with itemingredientDAO. Going to work on itemingredientDAO and MenuModel and process report.

Rokas – Did all FXML files and MVVM implementations. Faced some bugs with some of the classes that need to be resolved. Going to work on class diagrams, fix the bugs, do tests and work on project report.

After meeting discussion – Personal matters.

5.5.3 Daily scrum 2

Meeting information

Location: Online

Date: 07.8.2020

Time: 8:23

Attendees: Marwan, Rokas

Duration: 7 min

Daily meeting content

Marwan – I made tests for US7 and finished the MenuModel, worked on ItemDAO. Problems with database connection. Going to work on process report and ItemDAO.

Rokas – Fully debugged everything in the client implementation and did test cases for customizing items and class diagrams for US7 US21 and another one, then worked on project report checking Use case diagram. No problems. Going to check requirements use case description checking implementation section in project report and whatever sees fit.

After meeting discussion – removed a nonfunctional requirement: Both order lists in order screen user must be sorted bottom to top by number size, so that I can predict which order is going to be completed next, because we think it’s up to the receptionist to finish which order first.

5.5.4 Daily scrum 3

Meeting information

Location: Online

Date: 10.8.2020

Time: 9:02

Attendees:

Duration: Marwan, Rokas

14 min

Daily meeting content

Marwan – Finished ItemDAO. Had some problems regarding adding items and ingredients to the database. Will make test cases, update domain model, frontal matter of project report, ER diagram, and the rest of the process report.

Rokas – Checked functional and non-functional requirements, worked on use case diagrams, checked all descriptions for use cases. Implementation section of project report is taking way longer than expected. Will Finish implementation section, do project future and conclusion, check project inception section, and project report description section, maybe do the user manual

After meeting discussion – some of the bugs in the implementation when creating a customizable item, it doesn't set a price to the item.

5.5.5 Review

Minutes:

Location: online.

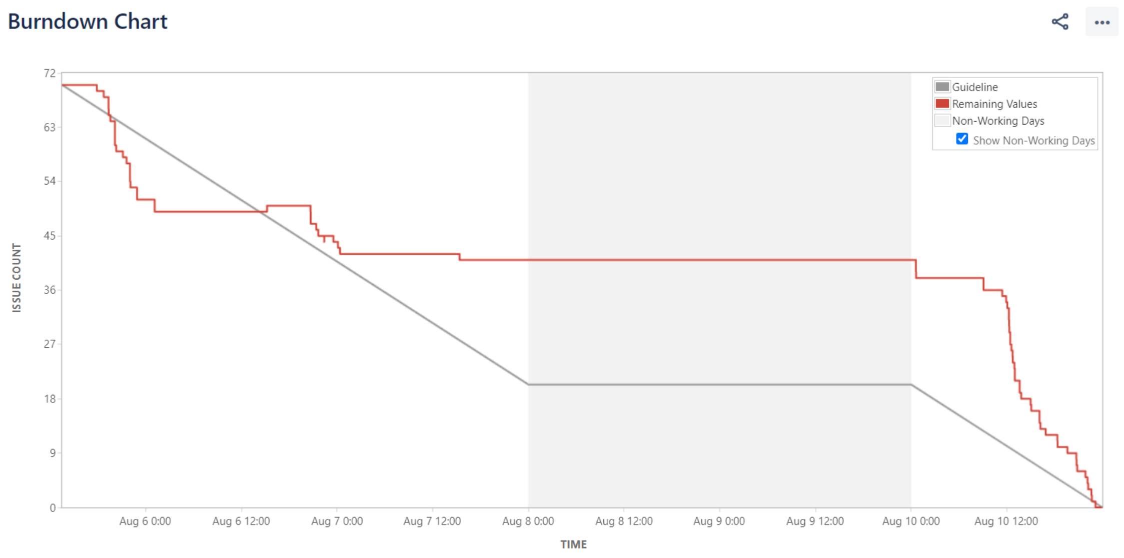
Participants: Rokas, Marwan.

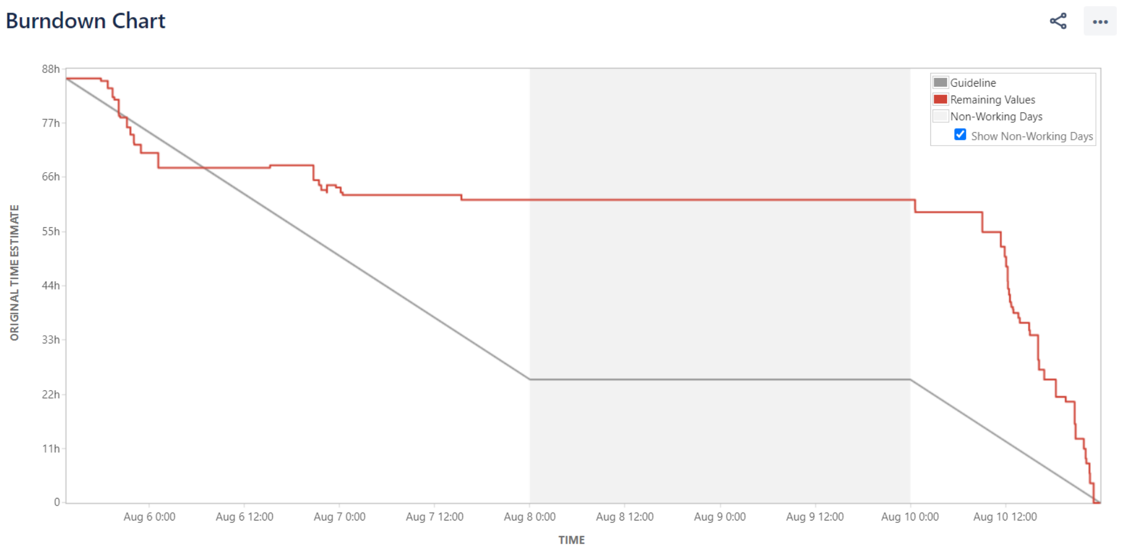
Starting Time: 23:58 10.8.2020

Duration: 18 minutes

- What we did: order screen, creating items in the menu, customizing items in orders.
- User stories all user stories are finished
  - As a customer, I want to be able to pay for my order by card, so that I can have a way to pay for the goods I will receive
  - As a receptionist, I want to have functionality to add an item to the menu list, so that the customer can order new items
  - As a receptionist I want be able to specify an item to be customizable, so that items which have multiple ingredients can be customized by customers.
  - As a receptionist, I want to be able to specify ingredients when creating an item, so that the customer sees ingredients in the item
  - as a receptionist I want to divide the items by specifying a group name so that the menu is organized
  - As a receptionist, I want to have functionality to print order info, so that the order can be shared among workers outside of the system.
  - As a customer, I want to have an option to customize items I added to cart, so that I can change the item to my preference
  - As a customer, I want to have functionality to reset the cart, so that I can remove all items I added.
  - As a receptionist, I want to have functionality to delete items from the menu, so that I can remove unwanted menu items

5.5.5.1 Burndown chart


Burndown Chart


Burndown Chart

5.5.6 retrospective

Starting Time: 00:16 11.8.2020

Members: Marwan, Rokas.

Duration: 9 minutes.

Minutes:

- Things to continue doing: Asking for help, communicating, not stepping on each other's toes, proper management of GitHub branches, updating Jira actively to inform the rest of the group that we are working.
- Things we should start doing: Asking for help even faster, better comments while committing to branches, commit daily to each personal branch, synchronize efficiency between members.
- Things we should stop doing: thinking on how to do a task differently too much, instead of just doing it, picking so many tasks in one sprint.

Check appendix – B for the full information about the scrum process and sprints

6 Personal Reflections

6.1 Rokas Barasa

Starting this project, I had a lot of momentum built up because the end of the third semester project had just happened not too long ago and I was determined to be very efficient and productive. I decided to give another chance to two members I worked with in the project I failed, because I thought they will have learned something new and improved their work, they were Marwan and Hugh. We also let another member Smiltė join the group who I have never worked before but heard very good things about.

I created the ideas, we agreed on it, everyone worked very well on project description and gave great criticism to my idea. In the second week of this phase problems started to occur. Hugh told me that he was very tired of documentation and would not focus on it, I understood that as he will be reluctant to write in the reports.

I made all the requirements and requested feedback form team members. I got feedback from one member Marwan, the others seemed to ignore my messages. The two members did not say they were doing something or showing any self-initiative to move the project, so we tried to give them tasks and aid us. We got promises that they will work on things, but by the end that turned out to be a lie. One that seemed as an insult the most was Smiltė saying she will do use case descriptions until Sunday. Sunday came, the member went to work at her part time job, I checked the use case description document, and nothing was there. I found out that Smiltė decided to work 5 days that week, which she was not forced to do by her contract, and Hugh not having any work but deciding to write up legal documents for things happening in his personal life.

These members seemed totally not prepared to do the project. We gave them time to fix course and we were patient but seeing that the day before the first day came, we have not received feedback on anything we made and the use case descriptions unfinished. Not getting these things done before the sprint was catastrophic in my view for the project, so I decided to stop giving chances and make a split. Before doing this, I talked to Marwan. This decision was met with guilt tripping, trying to take over the project and trying to replace me as leader of the project by the two members. These members gave no good reasons for me to reconsider and instead called me names. I contacted Troels to get some information on how to split properly. Everyone got the same files we made as a team up until that point.

After the split, the analysis went smoothly. Us two were more than capable of finishing up everything needed before sprinting the same day the split happened.

The sprints were a mixed bag. Me and Marwan have very different experience levels and due to that I had to explain a lot of things and take charge in decision making which was absolutely exhausting for me. I mostly took the client part of the implementation and let Marwan take what I think he was best at, the server. I was finishing things very quickly in the because I was interested to do this project. That interest became an obsession to make progress and ended up being a bad for my mental health. Marwan was having trouble keeping up but did a good job following scrum as a scrum master. We did not miss a single meeting and we focused on fixing most of the issues that caused us to fail sep2.

The end of the project was very stressful as there were a lot of things to do and a lot of things to consider. Marwan had a lot of troubles doing documentation as he had less experience and needed my attention which really tested my patience. We decided who is responsible for each part of the project. Gave each other feedback on how we did those parts, but we could not change each other’s work.

Comparing how Marwan was in previous semesters, he has improved tremendously but there is still a lot of things left for him to figure out. He put a lot of work into the project and tried his best and I do not regret working with him.

6.2 Marwan Summakieh

I have mixed feeling about this project, in the beginning of the project I wanted to prove to m self that I am better than I thought I was so I tried my hardest to achieve something, especially coming out of a disaster third semester. Three of the members were already in a group before Rokas, Hugh and me. Rokas and I worked twice at this point, so we were pretty used to working with each other, however me and Hugh never managed to get along even though we tried to go along before, but we needed each other which made us decided to put things aside and work professionally. Smilte on the other hand was a completely new member although she lacked the implementation skills, she supposedly was going to replace it with more documenting time, the idea behind adding her was to make the group diverse.

When the project period started, we had high expectations from the group, but things turned out to be not as expected.

When the project started, I noticed the lack of communication form some of the members, there was an incident where I mentioned them by name to give me updates on their work cause my part was dependent on theirs. But the message was ignored for two days it was the straw that broke the camel’s back we were getting close to a deadline and we simply didn’t see any sign that they are willing to work as hard as the rest of us which made me consider kicking them out of the group, but I was too understanding and couldn’t do it myself.

When they finally answered they gave the worst excuse that anyone could give “I was busy”

We all were busy, but we had to set our priorities, and they clearly didn’t consider the project as high priority. They had to go butt I was willing to give another chance but Rokas did not, and I was glad that he did not the work would have stacked for us to do and we would be unfairly graded.

As a result, we split the group and we formed a new one Rokas and I.

After that incident, the work was very smooth and for the first time, we finished all user stories. And the work did not seem as hard as we thought. We gained experience and personally I gained some of my self-confidence again. Although I faced so many problems during the project period Rokas helped me go through them and filled the gabs in my knowledge.

7 Supervision (Rokas)

This project had no supervisors. We did however contact a teacher that we had in second semester, Troels Mortensen, for some feedback. We received feedback on our project ideas, got help with handling the split of the team and get some information if the technology we used for the project was what was needed. We would have felt more confident having someone assigned to our project officially

8 Conclusions (Rokas)

In conclusion, we completed way more user stories that we expected in this project. Working in a two-person team is hard but preferable to a team that is half dead. This being our third experience with scrum and unified process we started being efficient at it and liking it more compared to the alternative of waterfall. Scrum is easy to understand and very good from a teamwork and efficiency standpoint. It places an importance on team communication by doing daily meeting, sprint planning, sprint review and retrospective all of this keeps the team up to date with each other and there to help if problems start occurring. The tools we used for this project have been working great.

Jira despite being very slow in sprint planning worked perfectly for the task management of the team. It provided an easy way to get multiple useful report diagrams including the burndown chart.

This being the fourth time using GitHub we can say were finally good at it by making a lot of mistakes and learning from them.

We could learn something in the team aspect. A mistake we made was choosing to work with people that do not have any coding experience and praise themselves as being only reports.

We learned the importance of poking holes at our ideas and importance of constraining a project idea using the delimitations section in the project description. This prevented us from being overwhelmed and getting stuck in analysis rabbit holes.

We learned a lot in the analysis, design, implementation, and testing sections. Not giving feedback in the analysis section early on can be catastrophic for the project. This leads to massive amounts of wasted work and costs team members sanity and time. Work could become obsolete if something that does not make sense is fixed, especially if there has been progress made in on that thing. Procrastinating in a sprint provides a bad time to your teammates. Rushing to finish the tasks in the first few days usually does not save time and only makes you a perfectionist for the rest of the sprint.

9 Sources of information

– Hofstede, 2019. *Hofstede-Insight.* [Online]

Available at: https://www.hofstede-insights.com/country-comparison/ireland,lithuania,syria/ [Accessed 10 07 2020].

10 Appendices

- Appendix A - Group Contract
- Appendix B – Minutes
  - Sprint1
    - Sprint1 Day 1
    - Sprint1 Day 2
    - Sprint1 Day 3
    - Sprint1 Day 4
    - Sprint1 review - retrospective
  - Sprint2
    - Sprint2 Day 1
    - Sprint2 Day 2
    - Sprint2 Day 3
    - Sprint2 Day 4
    - Sprint2 review - retrospective
  - Sprint3
    - Sprint3 Day 1
    - Sprint3 Day 2