Project Report

VIA Software Engineering

2nd semester

Group 8

Stefan Buzu - 331907

Igor Cretu - 331461

Alin Abdallh - 331979

Ahmed Omar - 333467

Supervisor: Henrik Kronborg Pedersen, Ole Ildsgaard Hougaard

Software Engineering

2nd Semester

1 June 2023

Table of content

Abstract

Warehouses are the heart of any industry in Denmark. For this reason, many workers with different backgrounds and nationalities work in this field, thus it is very important that no misunderstandings or unwanted problems happen and they can work without issues.

An owner of a warehouse wants the process of managing the warehouse and the process of using a program for the employees to be eased in order for any misunderstanding issues to disappear, moreover to create a better work environment for everyone involved.

As a result, the project had the purpose of easing the tasks for both the owner and the employees, who often expierience issues at work. Finally, the project is a client-server system implemented in Java which is able to be accessed both by the employees and the owner, while also displaying the data in a much simpler way, as well as having the option to change languages. The graphic interface arranges all the tasks of the owner and employees in a manner, that makes it much easier to access and change the language in order to better understand the program and avoid the language barrier for the employees.

The system follows all the requirements received from the owner and the documentation makes it easier to update and improve in the future.

1 Introduction

The warehouse industry plays a major role in the society we live in, thus the project aims to solve some issues identified by the people who actively work in this field. The main problems that are present in every company that runs a warehouse system in Denmark are the language barrier, as well as the technology that is being used. Currently, all the systems that are being used are operating in Danish, moreover, the technology used, in order to complete the warehouse tasks differ from company to company, nevertheless, they all tend to consume more time than it would take by using more improved technologies. The project aims to deliver a product that will solve these problems, moreover, that will save time and as a result provide more revenue for the stakeholders that choose to use it.

In the process of developing the project, the team has defined some delimitations that need to be stated, in order to provide a clear view of the final product. (See Project Description)

In the following section, there will be presented all the requirements, that were defined in the beginning of the project, as well as all the appropriate diagrams, that will showcase the functionalities of the final product.

2 Analysis

The analysis will showcase all of the requirements that were received from the owner as well as the use cases and the diagrams that are relevant. According to the name, the purpose of the analysis is to analyze the project through different diagrams, in order to view different aspects of it. In the following sections, each part of the Analysis will be covered, and moreover explained.

2.1 Requirements

The project has two types of requirements : functional and non-functional. Together they provide an overview of owner's expectations of the final product. In the end, the delivered product, will be able to fulfil the majority of the requirements. The functional requirments will focus on how the users will interact with the system, moreover what should it do in terms of functionality. On the other hand, the non-functional requirements, refer to the behaviour of the system in aspects that go beyond the functionalities.

2.2 Functional Requirements

1. As an Owner, I want to be able to register new employees in my system.
2. As an Owner I want to be able to assign different orders to my employees.
3. As an Owner I want to have a log-in system so my employees can log-in/log-out when they start/finish working.
4. As an Owner, I want to manage the information about my employees, so it is always updated.
5. As an Owner I want to be able to manage the orders that I get so they are always updated.
6. As an Owner I want to be able to remove employees from my system, in case they want to leave or I have to fire them.
7. As Owner, I want to keep track of which employee has a certain order.
8. As Owner, I want to be able to see how many boxes a certain employee has left from their order.
9. As Owner, I want to be able to store the data about the employees/orders in a database.

2.3 Non-Functional Requirements

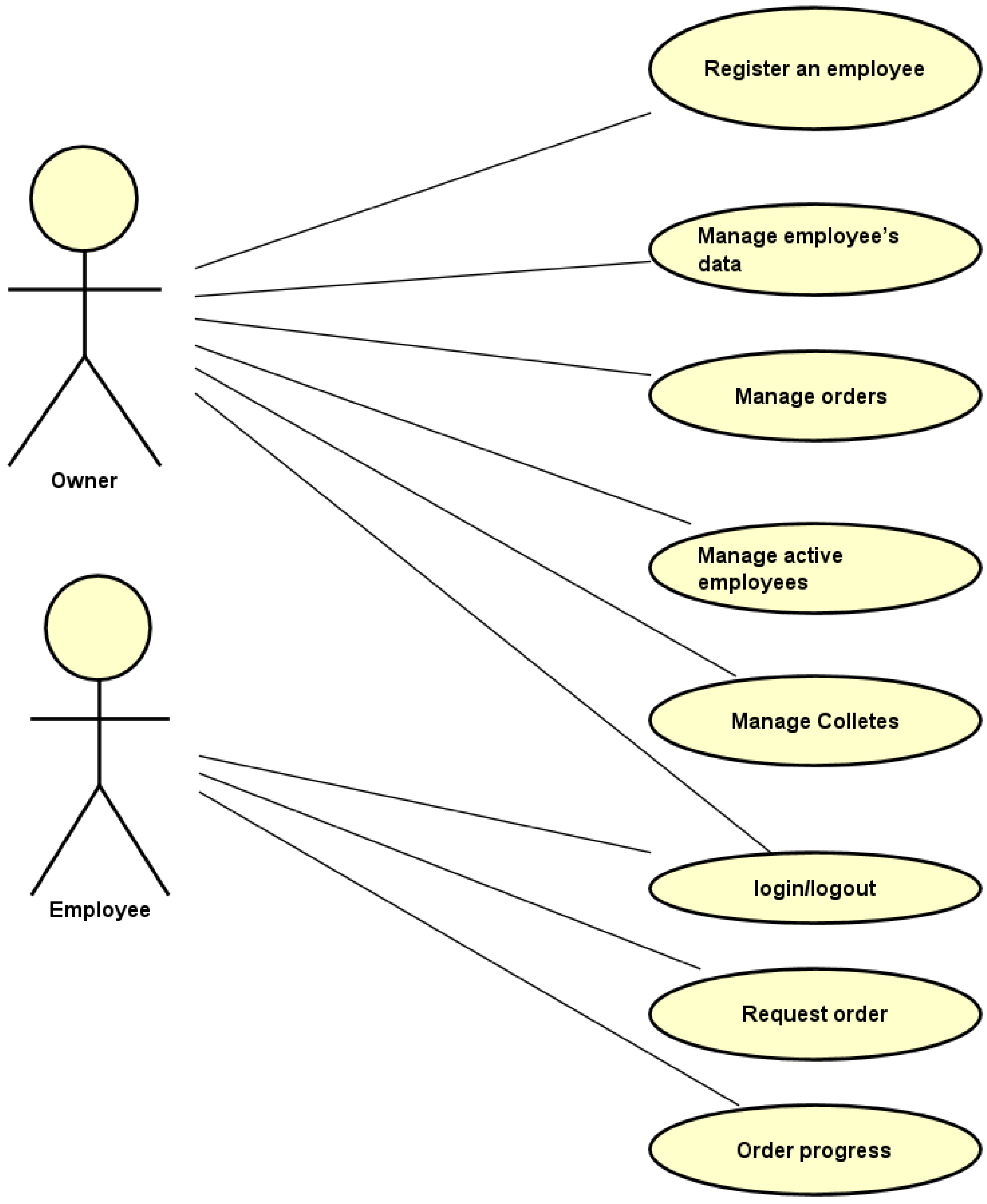
1. As an Owner I want to have my system both in Danish and English, so all my employees have an easy time operating it.

2. As an Owner I want to be able to start my system both on a Local Host, as well as a Local Area Network, so I can operate with the system both in my office and remote.

3. As an Owner I want to have all the steps implemented within the system documented, so I can better understand how it works.

2.4 Use case diagram

From the functional requirements, a use case diagram can simplify all the requirements and showcase them in the form of a diagram with all use cases and users for an easier understanding.



There are the two main actors represented in the use case diagram: the owner and the employee, as well as six main use cases: Register an employee, Manage employee’s data, Manage orders, Manage active employees, Manage Colletes and Login/Logout. The owner can access all six in order to manage the system, meanwhile, the employee can log in in order to see the information regarding the orders which are assigned to him by the owner. Bellow each use case will be presented.

1. **Register an employee**

The first action the owner must be able to do in the warehouse system is register employees. He should be able to register them by their names and the types of employees that they are: truck workers or office workers.

2. **Manage employee's data**

The owner should then be able to manage the data regarding the employees, accordingly to their needs. If a mistake was made, in the process of registering an employee, their data can be edited. On the other hand, if an employee needs to leave the company, their data can also be deleted from the system.

3. **Manage orders**

The warehouse employees have the responsibility to receive orders, and furthermore work with them. Therefore, the owner must be able to create orders by using the information stored in the database. Moreover, they will have the possibility to edit an order if some information is incorrect or needs to be updated, nevertheless, delete them if necessary.

4. **Manage active employees**

When the employees are working, their information should also be displayed in the system, while also having the possibility to assign orders to them or if needed to cancel an order.

5. **Manage Colletes**

The orders are made out of colletes, which can be added into the system or edited if the information of the colletes is stored wrongly. Each collete has a different name, location and size which will also be displayed in the orders.

6. **Login/Logout**

Also, the employees should be able to log into the system, in order to start working on the orders assigned to them. After logging into the system, the employees will be displayed as “active employees”. At the same time, the employees can logout when their shift is done.

7. **Request Order**

The employees should be able to request a new order to be assigned to them when they finish delivering another one. They should be able to press a button which notifies the owner that the employee is ready to be assigned another order.

2.5 Use case description

The use case descriptions show in a deeper way how the use cases work as well as any alternate scenarios that may occur.

In the use case below, one of the most important actions the owner will do is represented. For their warehouse to be able to function, they need to manage their employees’ data. As stated in the previous sections, the information about the employees can be edited if any inconvenience appears, and at the same time, it can be deleted if needed. All the alternative scenarios that may occur are also displayed.

Use Case	Manage employee's data
Summary	Display and edit information about the employees.
Actor	Owner
Precondition	The registration of the employees has already been done.
Postcondition	The information about the employees is updated.
Base sequence	1. System displays a list of all the existing employees in the database, along with their name and type of worker (truck worker/picker/office worker). [ALT1][ALT2] 2. Select an employee. 3. System displays the employee data. a. Name b. Type 4. If SHOW, then end the use case, IF EDIT then go to Scenario A. EDIT (step A.1), IF REMOVE then go to Scenario B. REMOVE (step B.1)

Scenario A. Edit	1. Enter/edit data. 2. Approve the data changed. 3. System validates data. [ALT3]

	4. System updates and displays the data.

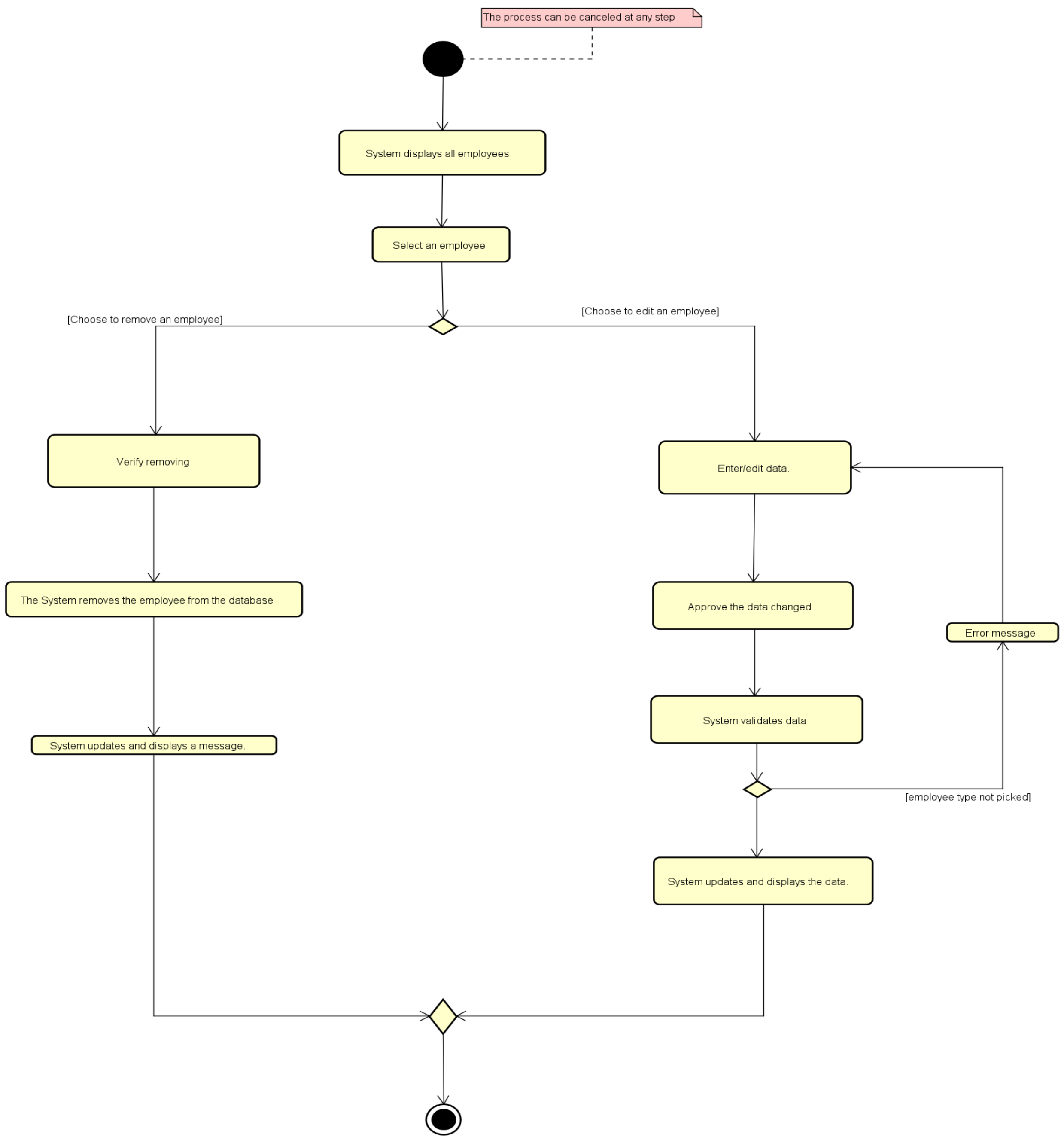
Scenario B. Remove	1. Verify removing. (Do you really want to delete message) 2. The System removes the employee from the database. [ALT4] 3. System updates and displays a message.

Use case ends.

Alternate sequence	[*ALT0] The process can be cancelled in all steps with user interaction. Use case ends. [ALT1] Filter by Selected value · Enter selected value to filter by (name, type, efficiency) · System shows an ordered list with the selected filter · Go to step 2 [ALT3] Validation · If type is not picker/truck worker or office worker, then system displays a message that the data is invalid. Go to step 4.1.
Note	This use case covers requirements 4,6,9

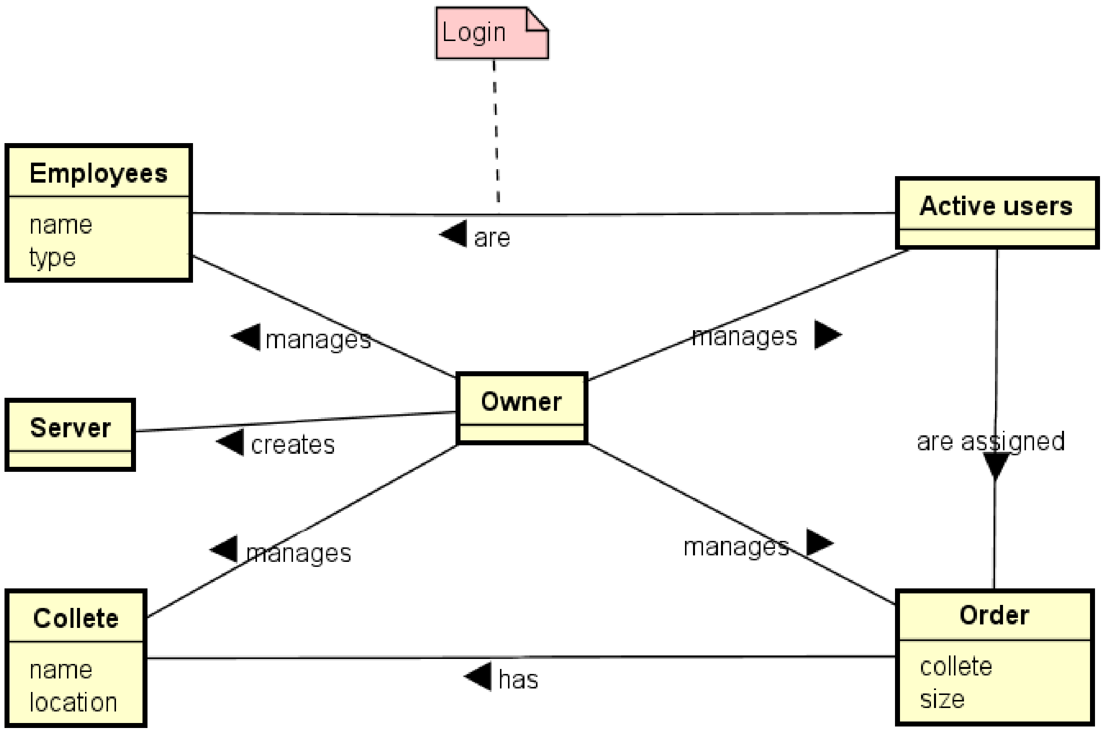
2.6 Activity diagram

The activity diagram is an illustrative explanation of the process described in the use case descriptions. In the activity diagram, it is represented a step-by-step explanation of each use case. In the diagram below, the processes of editing and removing an employee are detailed to ensure that the owner understands how to use the program correctly. There are also illustrated the alternative sequences that may appear and what would happen in each situation.


The process can be canceled at any step
System displays all employees
Select an employee
[Choose to remove an employee]
[Choose to edit an employee]
Verify removing
Enter/edit data.
The System removes the employee from the database
Approve the data changed.
Error message
System validates data
System updates and displays a message.
[employee type not picked]
System updates and displays the data.

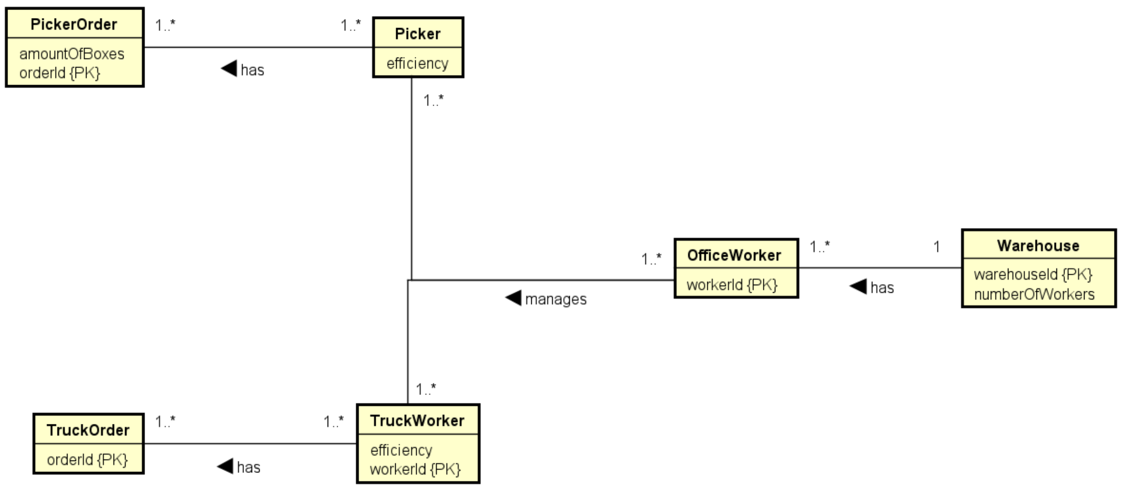
2.7 Domain Model

The domain model is a simplified view of how different classes within the program will connect to each other. As represented below, the owner is the one that starts the server, moreover manages all the data stored in the database, thus the owner is the main actor within the system. Each employee needs to log-in, in order to start operating with the orders that are made of different colletes, while also appearing "active" while doing so. It is important to be noted that only the "active employees" can operate with the orders.



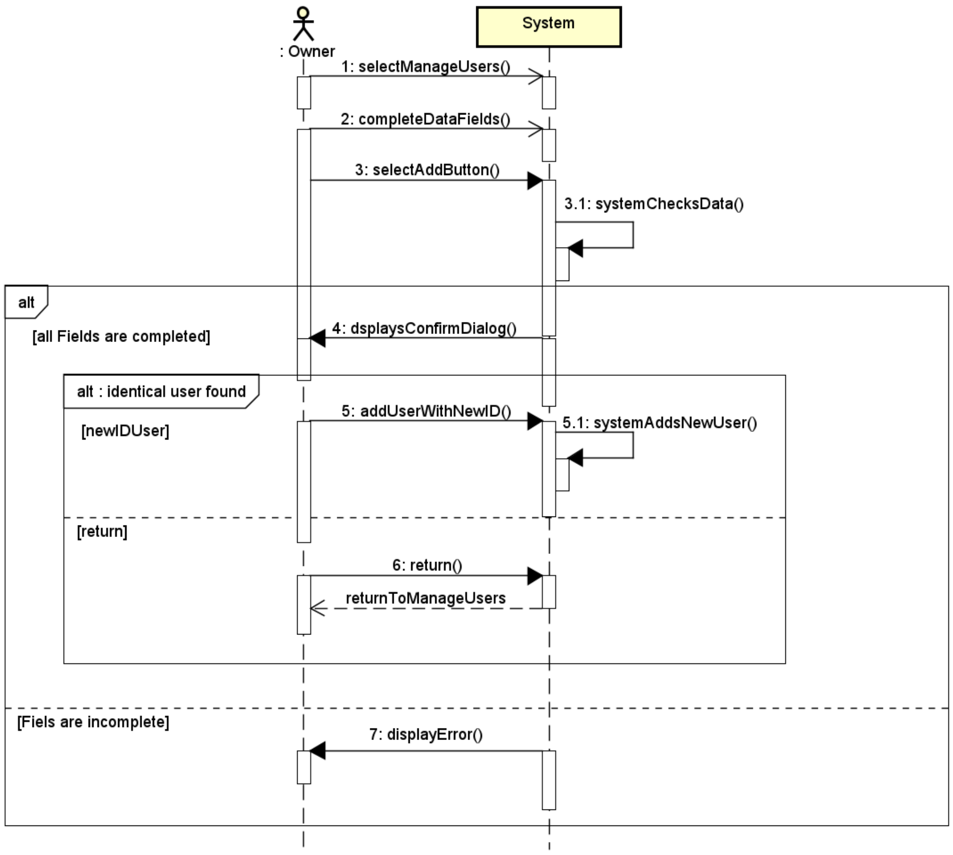
2.8 ER diagram

The ER diagram is a summary of all the data that will be stored in the database. It’s the most basic diagram which ensures the owner’s understanding of the database. According to the ER diagram, the warehouse has *OfficeWorker* that manages *Picker* and *TruckWorker.* Each one of them has a *workerId.* The *Picker* and the *TruckWorker* have their own type of orders, that ultimately also have an *orderId*, in order to be able to identify them.



2.9 System sequence diagram

In order to have a better understanding of the client-server communication, the system sequence diagram was made. This diagram, showcases the register employee use case, along with all the alternate scenarios that may occur during its execution. Another element present in this diagram is the actor who interacts with the system.



3 Design

The Design is another key component of the project, it provides the necessary diagrams that back up the code for the database, as well as all the classes present within the system.

The Class Diagram is the diagram that includes all the classes from the system, as well as the structure that was being used while developing the project.

The Sequence Diagram will show how the main actors interact with the system, furthermore, the steps they need to do, in order to achieve a certain goal in the end.

Finally, the EER, GR and Relational Schema are three diagrams that are used for a better understanding of a database.

3.1 Class diagram

The class diagram is a visual representation of all the classes and interfaces that are used in the code, as well as all the relations between them.

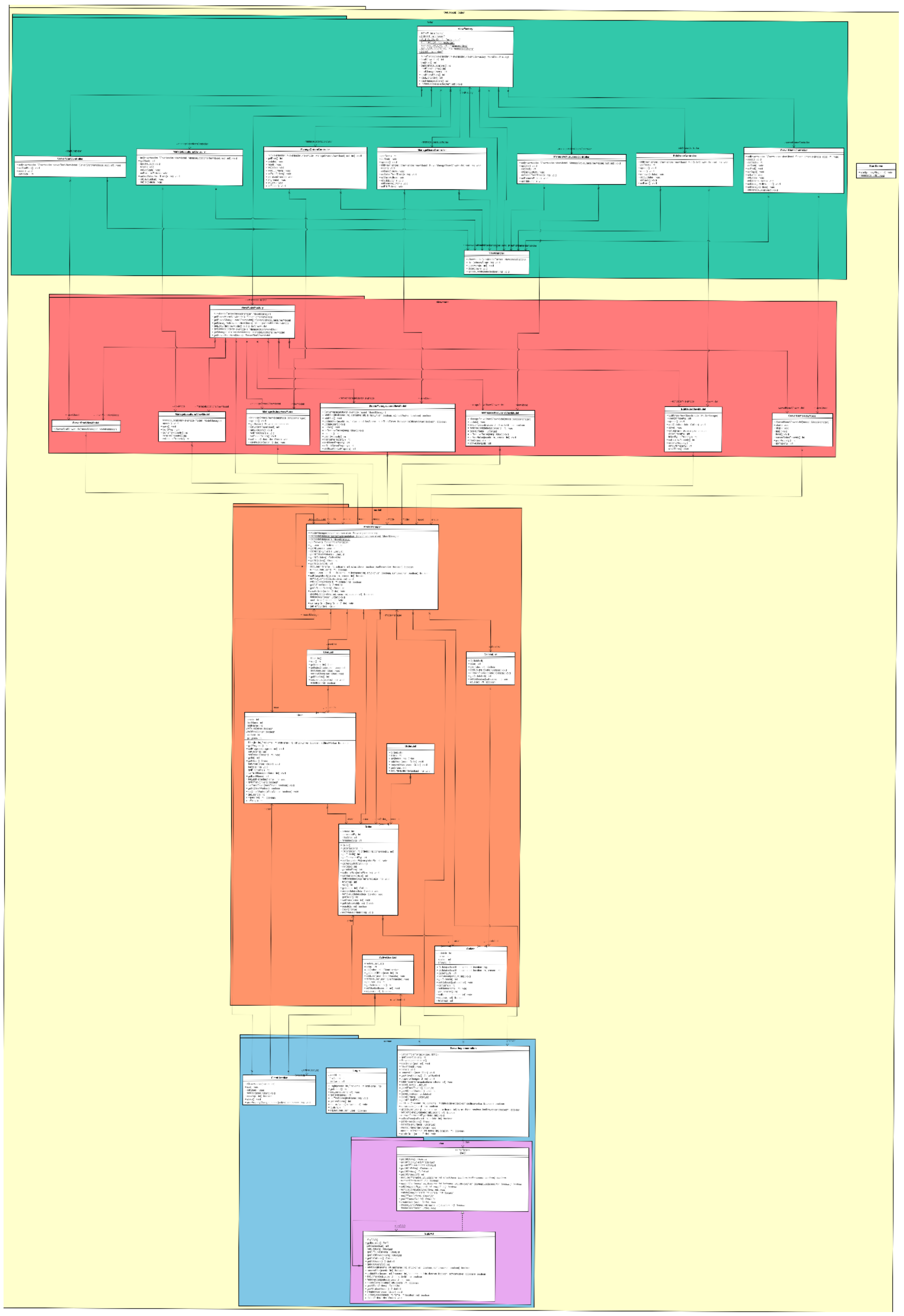
In the first figure below, it is illustrated the part of the class diagram which is responsible for the *ServerApplication*. It has four packages: model, view, viewmodel, and server as well as a StartServer class outside the packages which will enable the owner to start the server from their computer.

The server package contains all the classes that are responsible for the client to be able to log in such as *ServerImplementation*, *Login* and *ClientHandler*.

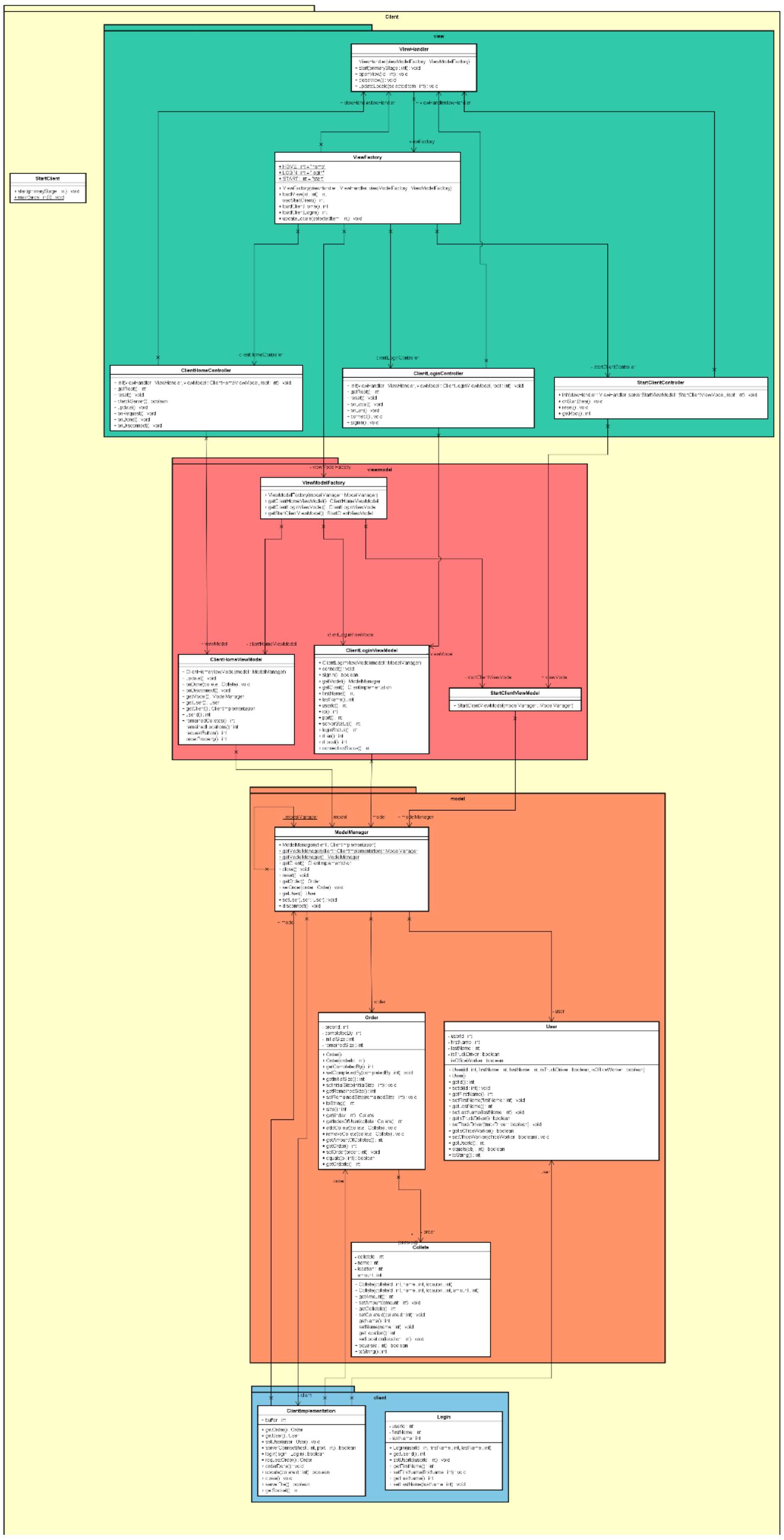
The model package includes classes, where there are represented different items such as *Order*, *Collete* and *User* and classes in which there are methods to manage the information stored in the other classes by using ArrayLists such as *ColleteList*, *OrderList*, *UserList* and *ActiveUserList*.

The classes within the view package are the controllers, which are responsible for handling the graphic user interfaces (GUI) as well as the *ViewFactory* and *ViewHandler* and the classes that are in the viewmodel package provide data and operations in order to manage the information and communicates with the controllers to update the *view* and to perform the required actions.

Moreover, in the making of the code, the SOLID principles were used, in order to build code that is easy to understand, change and test.

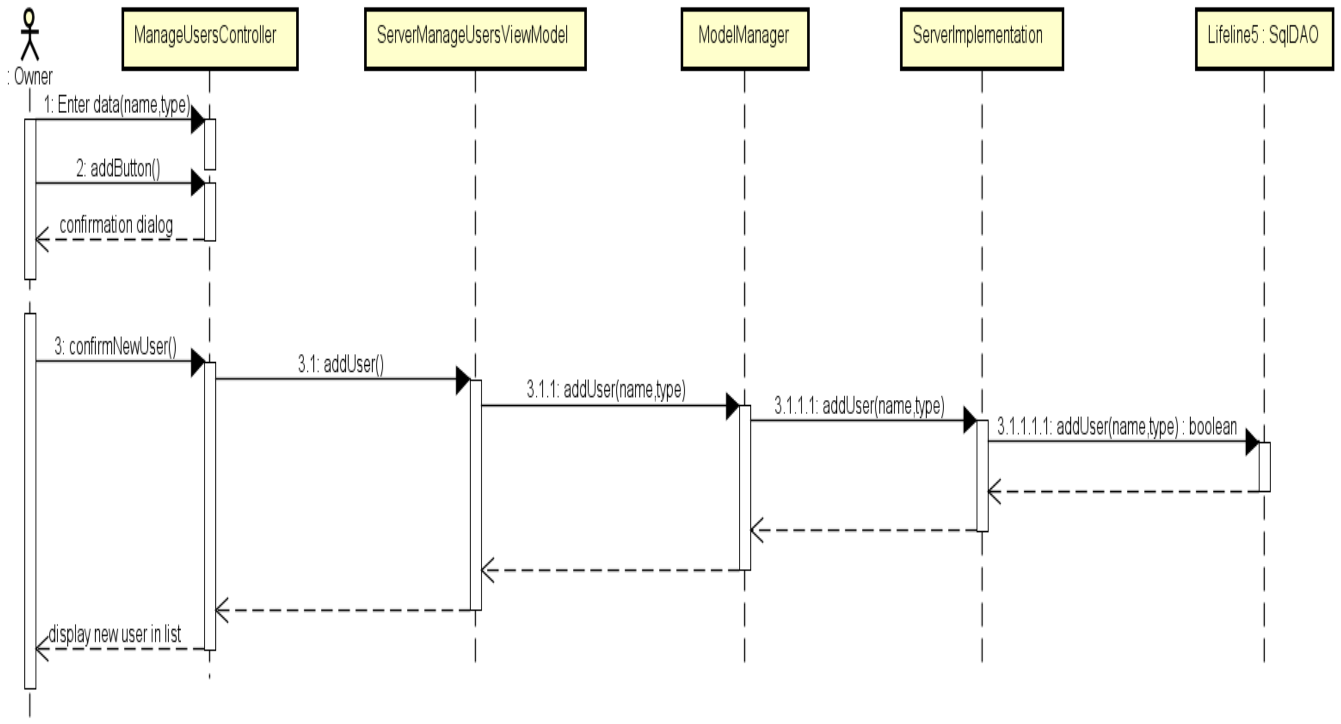


The diagram below shows the part of the diagram responsible for the *Client*. The class *StartClient* initializes the user interface of the client. There are 4 main packages: *model, view, viewmodel* and *client*.
The *model* package contains classes which represent different objects, and a *ModelManager* which manages the data and holds all the information. The classes from the view package and the *viewmodel* package are responsible for handling the user interaction and updating the data from the database, as well as providing the according properties and methods, in order to facilitate that process.


Client
view
ViewHandler
ViewFactory
StartClient
ClientHomeController
ClientLoginController
StartClientController
viewmodel
ViewModelFactory
ClientHomeViewModel
ClientLoginViewModel
StartClientViewModel
model
ModelManager
Order
User
Collate
client
ClientImplementation
Login

3.2 Sequence diagram

The Sequence Diagram shows the order of different events and actors within the system. It illustrates how all these components interact with one another, moreover how they do the necessary actions. The sequence diagram below shows the step by step execution of the process of registering an employee into the system. The diagram clearly shows the use of the MVVM pattern in developing the project.

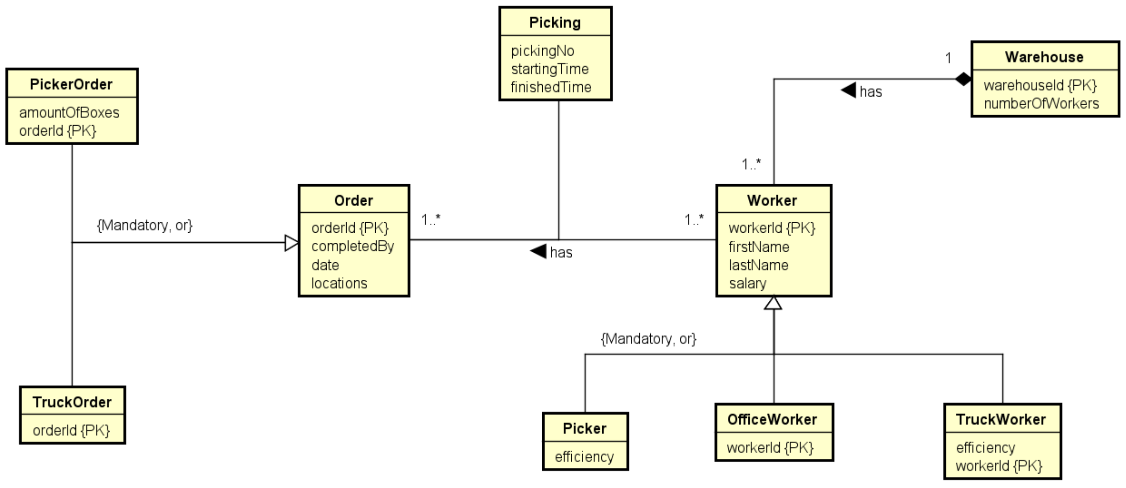


3.3 EER diagram

The Enhanced Entity Diagram (EER) is an extended version of the ER diagram. It contains, alongside everything that was present in the ER diagram, subclasses, and superclasses and also includes composition and aggregation.

In the EER diagram below, the entities "*OfficeWorker*", "*TruckWorker*", and "*Picker*" share the newly formed superclass called "*Worker*". Similarly, the "*PickerOrder*" and "*TruckOrder*" share the superclass "*Order*".

It is also presented that the "*OfficeWorker*", "*TruckWorker*", and "*Picker*" have an aggregation connected to the "*Worker*" because they are a part of it and also the "*PickerOrder*" and "*TruckOrder*" have an aggregation to the "*Order*". There is also a composition between the "*Warehouse*" and the "*Worker*" because the "Warehouse" has a "*Worker*".

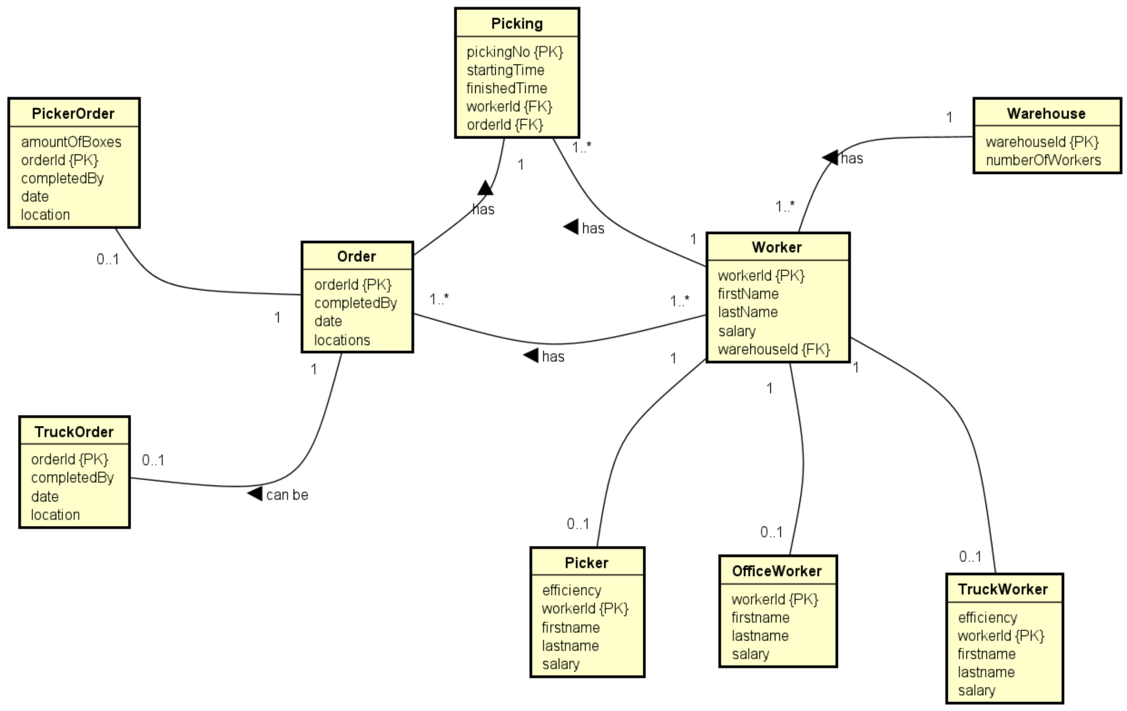


3.4 GR diagram and Relational Schema

The GR diagram displays an overview of all the relationships, attributes and keys between the entities. Both the relational schema and the GR diagram are based on the EER diagram. The main difference between the EER diagram and the GR diagram is the disappearance of the aggregation, composition and the *:* relationship. Furthermore, in addition to the primary keys which were already present in the EER diagram, the GR diagram also shows the foreign keys (FK) of each table.

Warehouse (warehouseId, numberOfWorkers) **Primary key**: warehouseId	**TruckWorker** (workerId, firstName, lastName, salary, efficiency) **Primary key**: workerId
Worker (workerId, firstName, lastName, salary, warehouseId) **Primary key**: workerId **Foreign key**: warehouseId **references** Warehouse(warehouseId)	**Order** (orderId, completedBy, date, locations) **Primary key**: orderId
Picker (workerId, firstName, lastName, salary, efficiency) **Primary key:** workerid	**PickerOrder** (orderId, completedBy, date, locations, amountOfBoxes) **Primary key**: orderId
OfficeWorker (workerId, firstName, lastName, salary) **Primary key**: workerId	**TruckOrder** (orderId, completedBy, date, locations) **Primary key**: orderId
Picking (workerId, orderId, startingTime, finishedTime,	

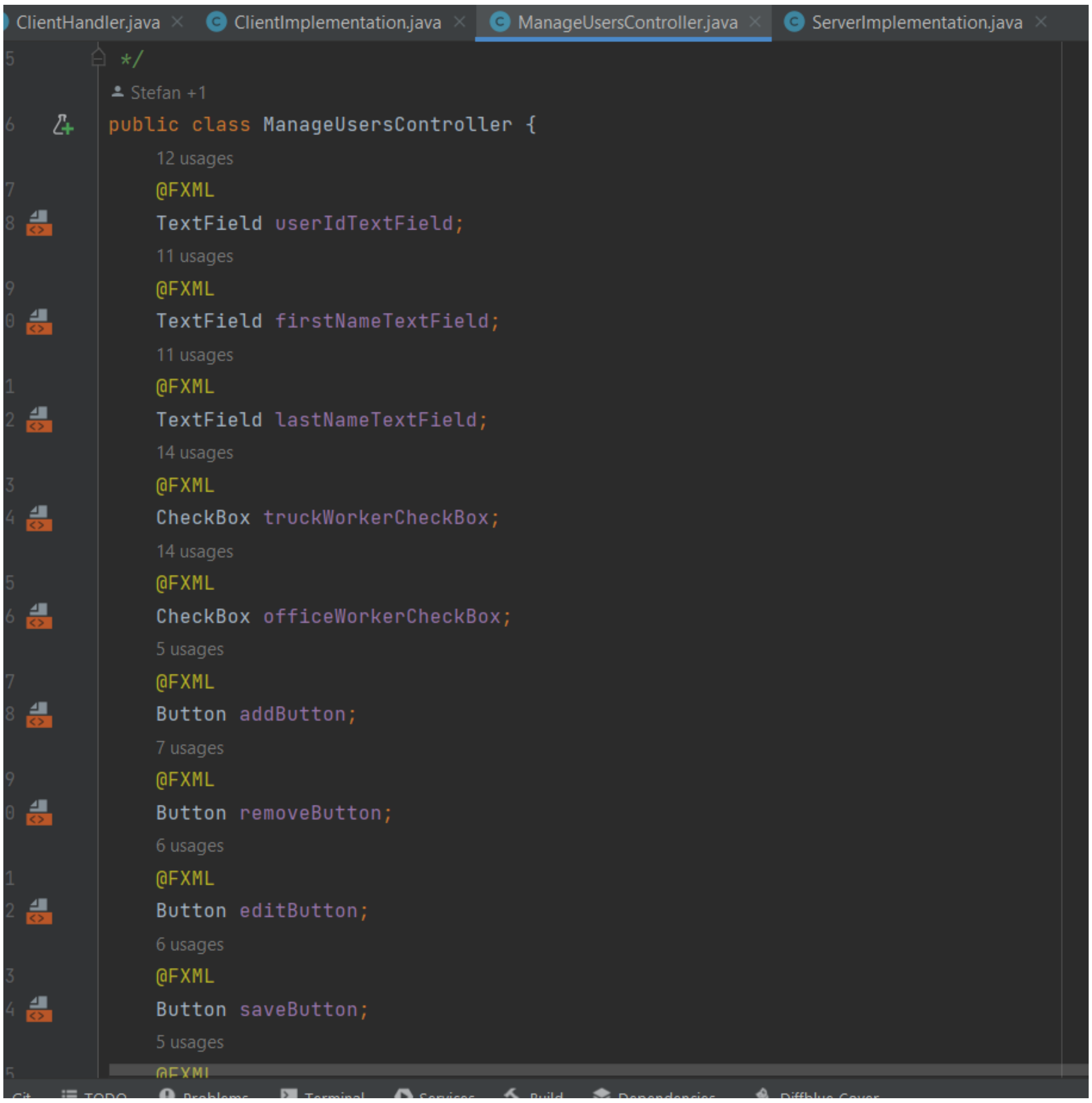
pickingNo) **Primary key**: pickingNo **Foreign key**: workerId **references** Worker(workerId) orderId **references** Order(orderId)	



4 Implementation

In order to have a better understanding of the code that is running behind the scenes, some code snippets were attached bellow, as well as the necessary explanation for each one of them.

Bellow, it is illustrated the *ManageUsersController* class that is part of the *view* package. The purpose of this class is to connect the *view* with the *viewmodel*. The first snippet contains the initialization of the *FXML* elements present on the page.

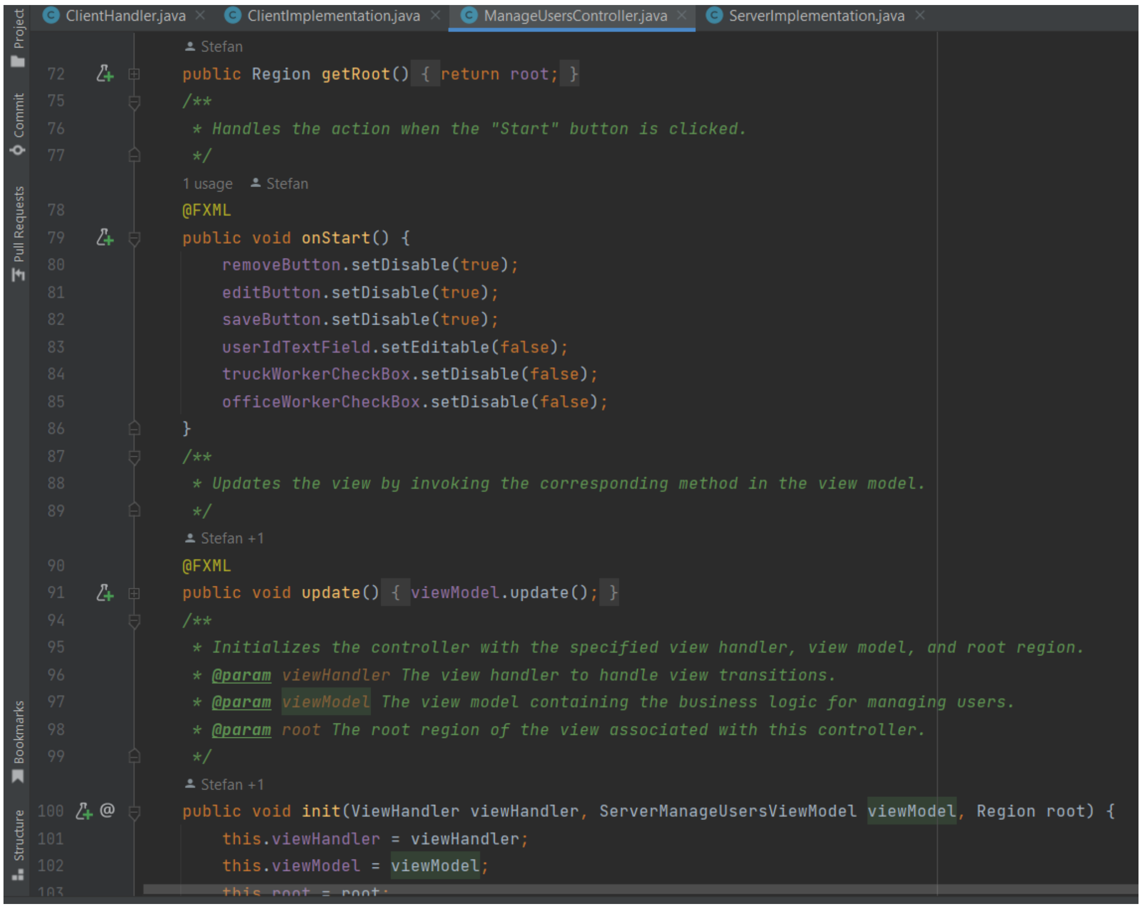


ClientHandler.java ClientImplementation.java ManageUsersController.java ServerImplementation.java

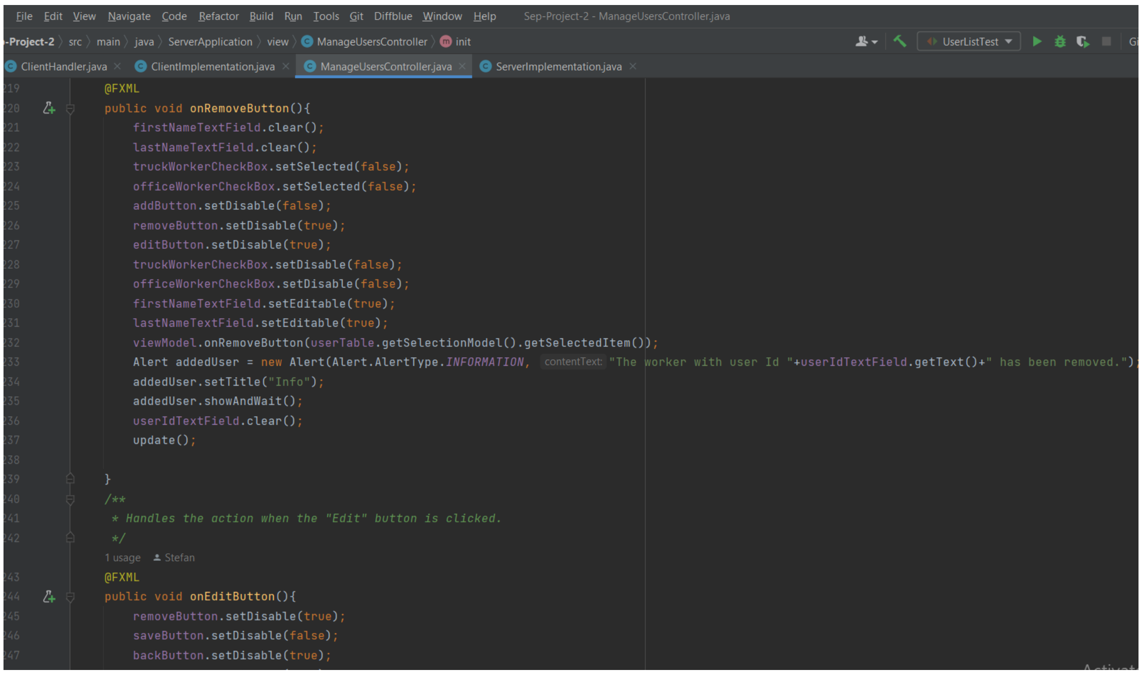
```
*/
Stefan +1
public class ManageUsersController {
    12 usages
    @FXML
    TextField userIdTextField;
    11 usages
    @FXML
    TextField firstNameTextField;
    11 usages
    @FXML
    TextField lastNameTextField;
    14 usages
    @FXML
    CheckBox truckWorkerCheckBox;
    14 usages
    @FXML
    CheckBox officeWorkerCheckBox;
    5 usages
    @FXML
    Button addButton;
    7 usages
    @FXML
    Button removeButton;
    6 usages
    @FXML
    Button editButton;
    6 usages
    @FXML
    Button saveButton;
    5 usages
    @FXML
```

Furthermore, there are all the methods that connect the according buttons on the page with the wanted piece of code. A relevant example would be the *onRemoveButton* method, that will disable the necessary field and buttons, when pressing the remove button, moreover an alert will be displayed that contains an informative message.

According to the MVVM principles, all the changes to the database are being made throgh the *viewModel*. The *update* method contains an example where this principle was used.


ClientHandler.java
ClientImplementation.java
ManageUsersController.java
ServerImplementation.java
Project
Commit
Pull Requests
Bookmarks
Structure

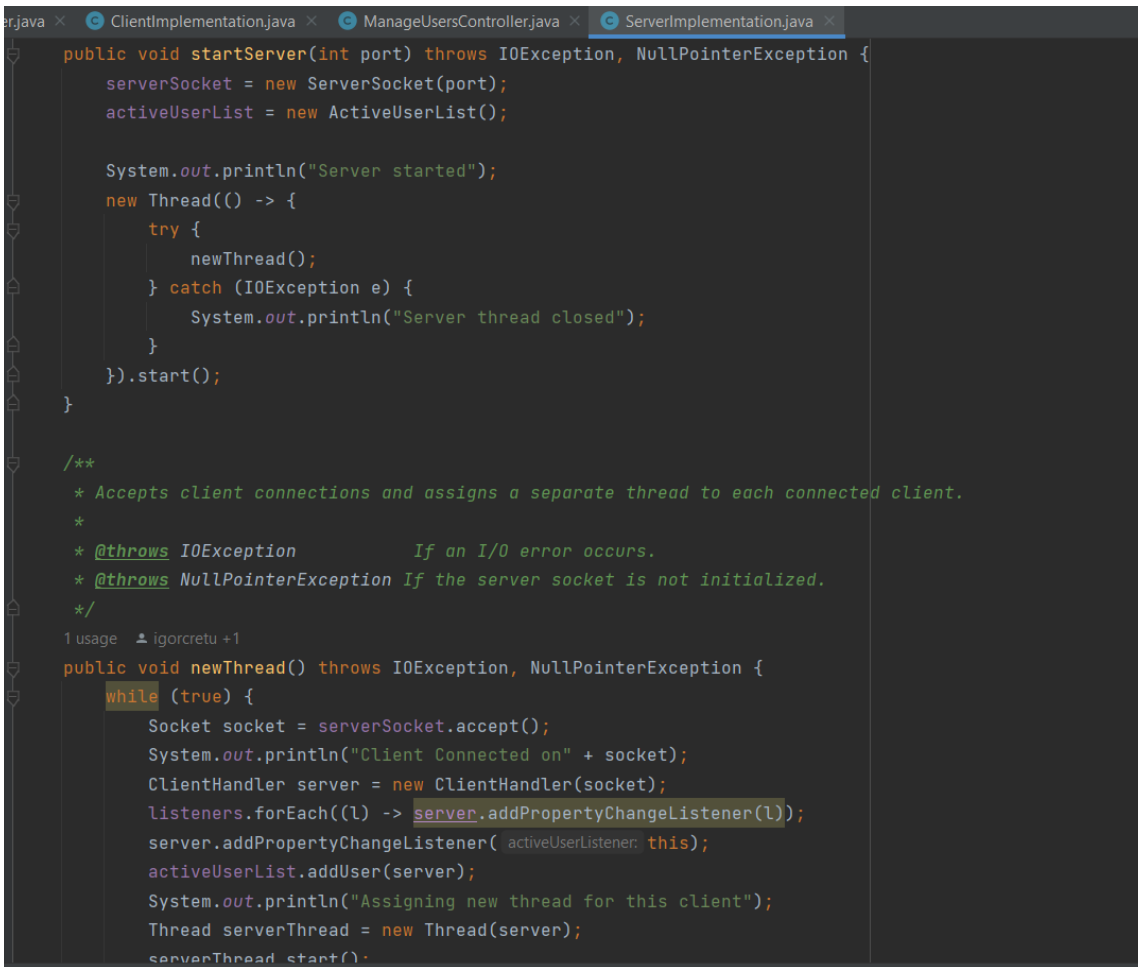
```java
    Stefan
    public Region getRoot() { return root; }
    /**
     * Handles the action when the "Start" button is clicked.
     */
    1 usage   Stefan
    @FXML
    public void onStart() {
        removeButton.setDisable(true);
        editButton.setDisable(true);
        saveButton.setDisable(true);
        userIdTextField.setEditable(false);
        truckWorkerCheckBox.setDisable(false);
        officeWorkerCheckBox.setDisable(false);
    }
    /**
     * Updates the view by invoking the corresponding method in the view model.
     */
    Stefan +1
    @FXML
    public void update() { viewModel.update(); }
    /**
     * Initializes the controller with the specified view handler, view model, and root region.
     * @param viewHandler The view handler to handle view transitions.
     * @param viewModel The view model containing the business logic for managing users.
     * @param root The root region of the view associated with this controller.
     */
    Stefan +1
    public void init(ViewHandler viewHandler, ServerManageUsersViewModel viewModel, Region root) {
        this.viewHandler = viewHandler;
        this.viewModel = viewModel;
        this.root = root;
```


File Edit View Navigate Code Refactor Build Run Tools Git Diffblue Window Help
Sep-Project-2 - ManageUsersController.java
-Project-2 〉 src 〉 main 〉 java 〉 ServerApplication 〉 view 〉 ManageUsersController 〉 init
UserListTest
ClientHandler.java
ClientImplementation.java
ManageUsersController.java
ServerImplementation.java

```java
    @FXML
    public void onRemoveButton(){
        firstNameTextField.clear();
        lastNameTextField.clear();
        truckWorkerCheckBox.setSelected(false);
        officeWorkerCheckBox.setSelected(false);
        addButton.setDisable(false);
        removeButton.setDisable(true);
        editButton.setDisable(true);
        truckWorkerCheckBox.setDisable(false);
        officeWorkerCheckBox.setDisable(false);
        firstNameTextField.setEditable(true);
        lastNameTextField.setEditable(true);
        viewModel.onRemoveButton(userTable.getSelectionModel().getSelectedItem());
        Alert addedUser = new Alert(Alert.AlertType.INFORMATION, contentText: "The worker with user Id "+userIdTextField.getText()+" has been removed.");
        addedUser.setTitle("Info");
        addedUser.showAndWait();
        userIdTextField.clear();
        update();

    }
    /**
     * Handles the action when the "Edit" button is clicked.
     */
    1 usage   Stefan
    @FXML
    public void onEditButton(){
        removeButton.setDisable(true);
        saveButton.setDisable(false);
        backButton.setDisable(true);
```

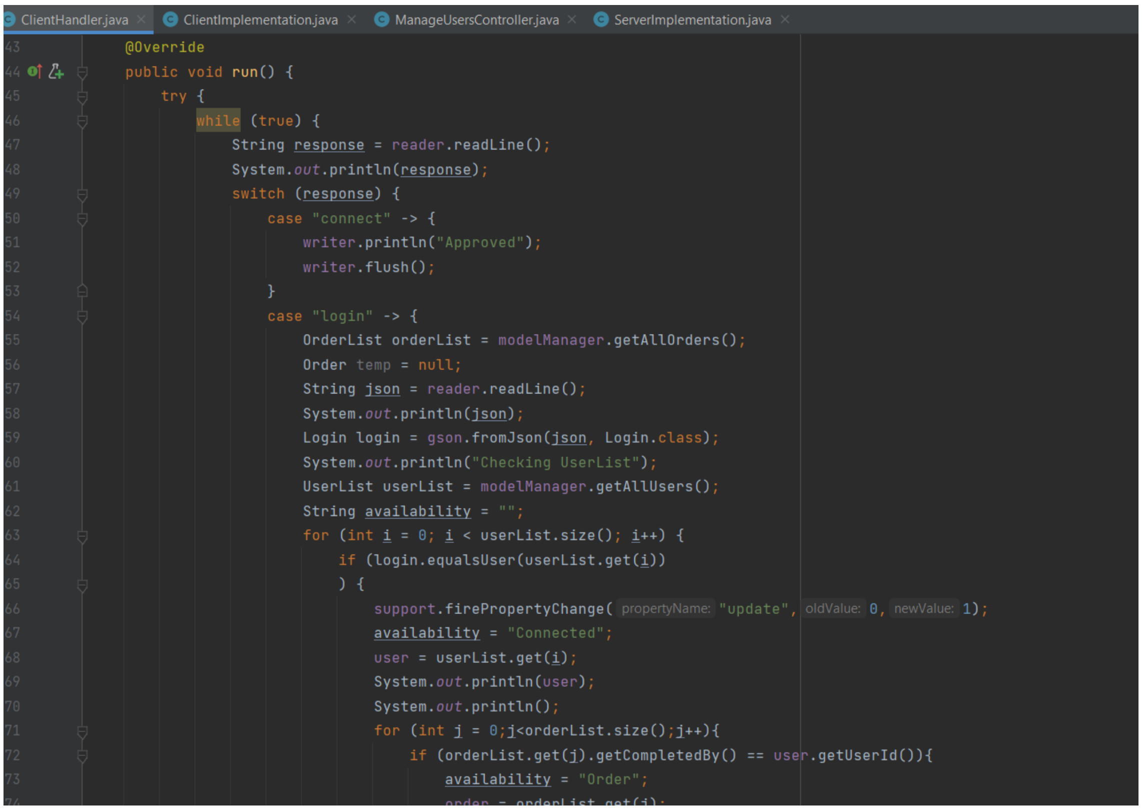
The following screenshots will show the *ServerImplementation* class, that represents the server implementation for handling client connections. Moreover it also manages active users, as well as the communication with the Data Access Object for data operations.

When a new client connection is being requested, the *ServerImplementation* class will assign a separate thread everytime, in order to ensure the efficiency and the speed of the system.

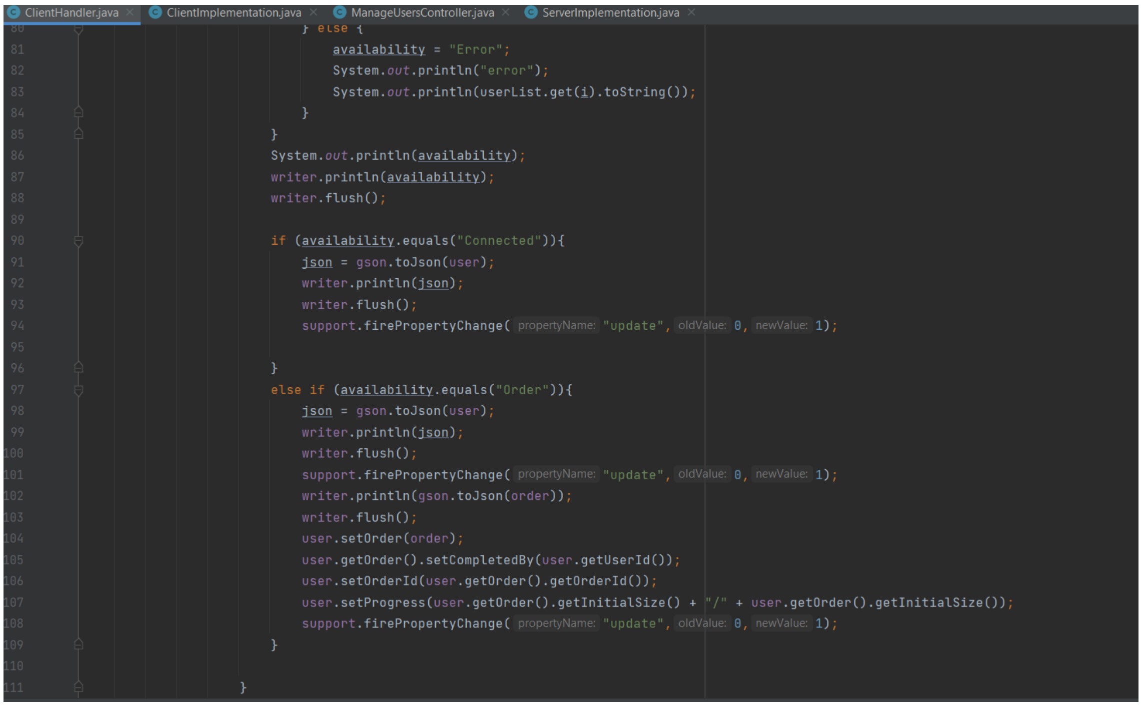

er.java
ClientImplementation.java
ManageUsersController.java
ServerImplementation.java

```java
public void startServer(int port) throws IOException, NullPointerException {
    serverSocket = new ServerSocket(port);
    activeUserList = new ActiveUserList();

    System.out.println("Server started");
    new Thread(() -> {
        try {
            newThread();
        } catch (IOException e) {
            System.out.println("Server thread closed");
        }
    }).start();
}

/**
 * Accepts client connections and assigns a separate thread to each connected client.
 *
 * @throws IOException           If an I/O error occurs.
 * @throws NullPointerException If the server socket is not initialized.
 */
1 usage   igorcretu +1
public void newThread() throws IOException, NullPointerException {
    while (true) {
        Socket socket = serverSocket.accept();
        System.out.println("Client Connected on" + socket);
        ClientHandler server = new ClientHandler(socket);
        listeners.forEach((l) -> server.addPropertyChangeListener(l));
        server.addPropertyChangeListener( activeUserListener: this);
        activeUserList.addUser(server);
        System.out.println("Assigning new thread for this client");
        Thread serverThread = new Thread(server);
        serverThread.start();
```

Another class major class within the system is the *ClientHandler* that processes each client request and furthermore communicates with it. Moreover, each functionality presented in the use case diagram before will be handled in this class. As a result, all the clien't possible operations are present here. An appropriate example would be the login request from the client. Here, the *ClientHandler* will expect a json object from the client, furthermore it will convert the given object into a *Login* object and check if it matches the information from the database. Finally, the system will send a message back to the client, that will state if the login was accepted or not. The same principle applies to all the other requests.


ClientHandler.java
ClientImplementation.java
ManageUsersController.java
ServerImplementation.java

```java
    @Override
    public void run() {
        try {
            while (true) {
                String response = reader.readLine();
                System.out.println(response);
                switch (response) {
                    case "connect" -> {
                        writer.println("Approved");
                        writer.flush();
                    }
                    case "login" -> {
                        OrderList orderList = modelManager.getAllOrders();
                        Order temp = null;
                        String json = reader.readLine();
                        System.out.println(json);
                        Login login = gson.fromJson(json, Login.class);
                        System.out.println("Checking UserList");
                        UserList userList = modelManager.getAllUsers();
                        String availability = "";
                        for (int i = 0; i < userList.size(); i++) {
                            if (login.equalsUser(userList.get(i))
                            ) {
                                support.firePropertyChange( propertyName: "update", oldValue: 0, newValue: 1);
                                availability = "Connected";
                                user = userList.get(i);
                                System.out.println(user);
                                System.out.println();
                                for (int j = 0;j<orderList.size();j++){
                                    if (orderList.get(j).getCompletedBy() == user.getUserId()){
                                        availability = "Order";
                                        order = orderList.get(j);
```

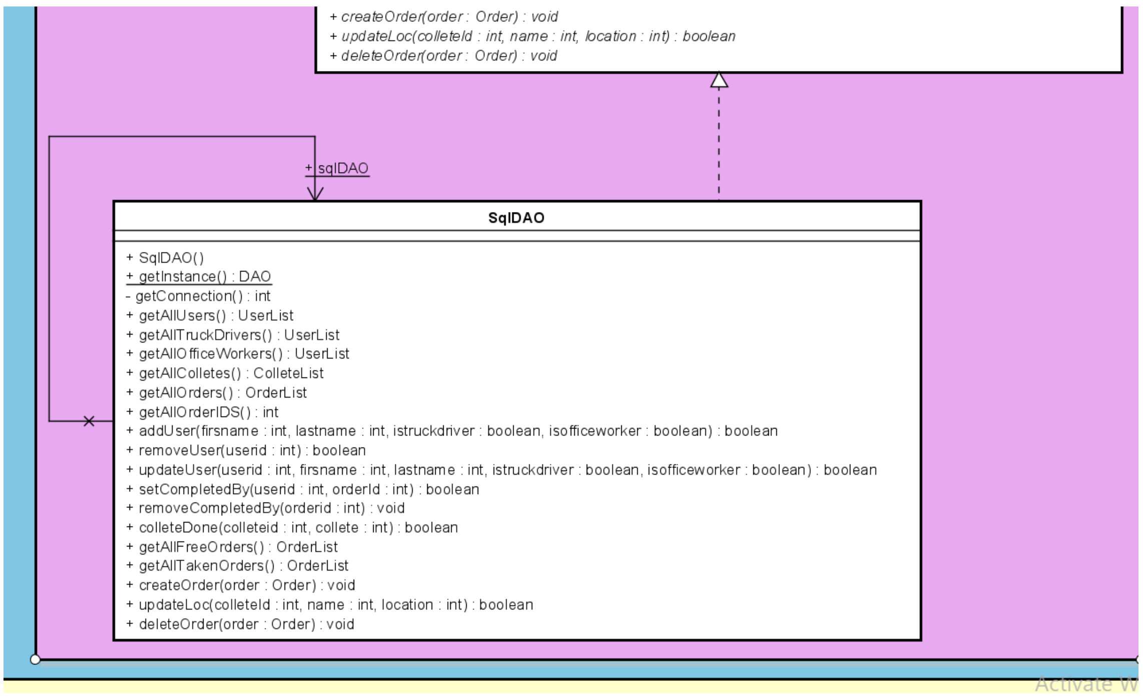

ClientHandler.java
ClientImplementation.java
ManageUsersController.java
ServerImplementation.java

```java
                            } else {
                                availability = "Error";
                                System.out.println("error");
                                System.out.println(userList.get(i).toString());
                            }
                        }
                        System.out.println(availability);
                        writer.println(availability);
                        writer.flush();

                        if (availability.equals("Connected")){
                            json = gson.toJson(user);
                            writer.println(json);
                            writer.flush();
                            support.firePropertyChange( propertyName: "update", oldValue: 0, newValue: 1);

                        }
                        else if (availability.equals("Order")){
                            json = gson.toJson(user);
                            writer.println(json);
                            writer.flush();
                            support.firePropertyChange( propertyName: "update", oldValue: 0, newValue: 1);
                            writer.println(gson.toJson(order));
                            writer.flush();
                            user.setOrder(order);
                            user.getOrder().setCompletedBy(user.getUserId());
                            user.setOrderId(user.getOrder().getOrderId());
                            user.setProgress(user.getOrder().getInitialSize() + "/" + user.getOrder().getInitialSize());
                            support.firePropertyChange( propertyName: "update", oldValue: 0, newValue: 1);
                        }

                    }
```

5 The SOLID principles

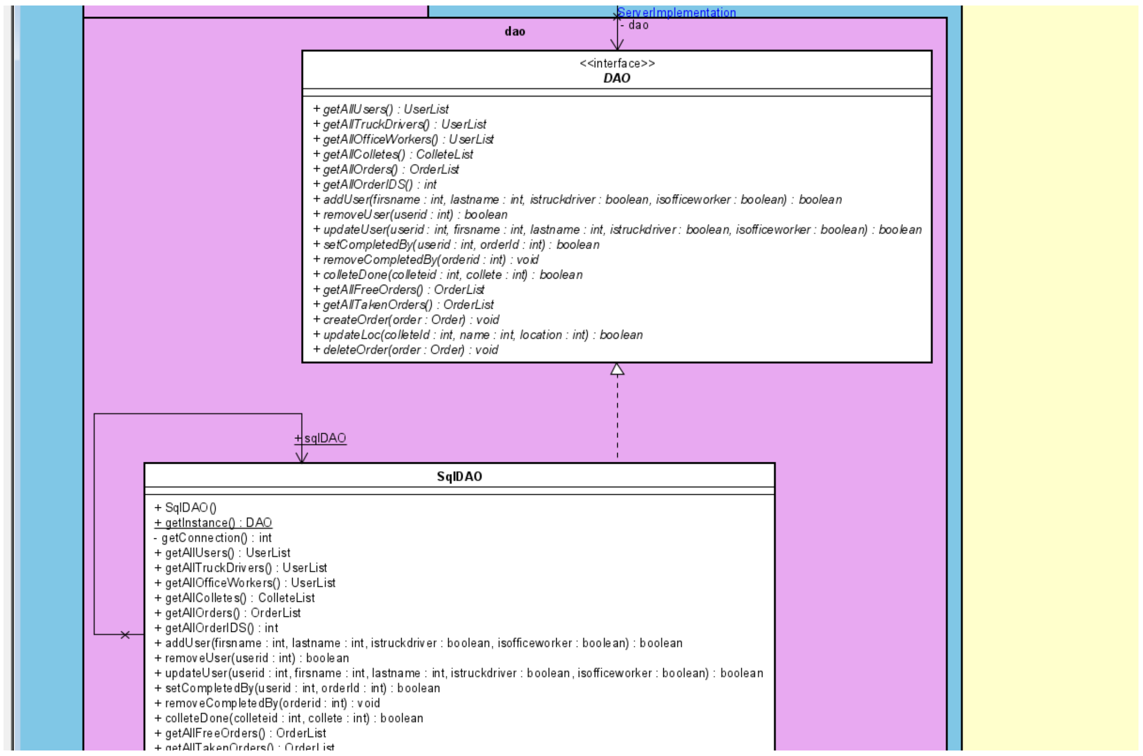
5.1 Single Responsibility Principle (SRP)

The SRP principle means that each class has only one responsibility, which can be seen in the classes below within the Server package.


+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void
+ sqlDAO
SqlDAO
+ SqlDAO()
+ getInstance() : DAO
- getConnection() : int
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void

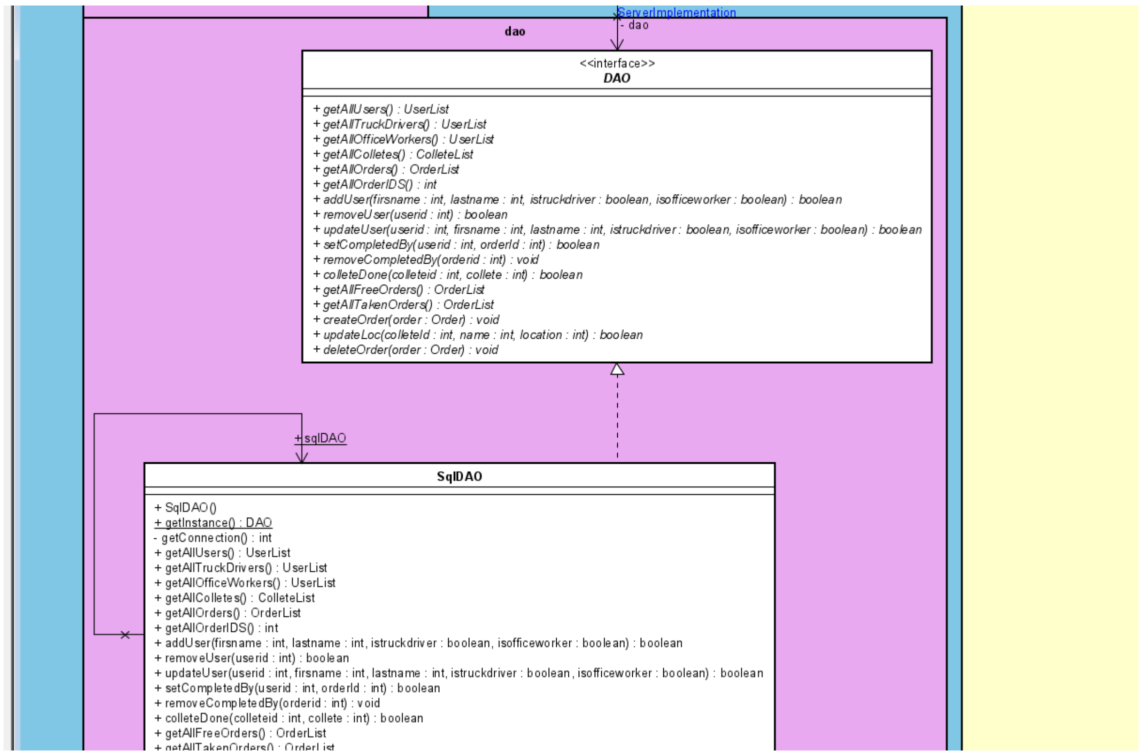
5.2 Open Close Principle (OCP)

The OSP means that software modules should be opened for extension and closed for modification. This can be done by using abstraction.


ServerImplementation
- dao
dao
<<interface>>
DAO
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void
+ sqlDAO
SqlDAO
+ SqlDAO()
+ getInstance() : DAO
- getConnection() : int
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList

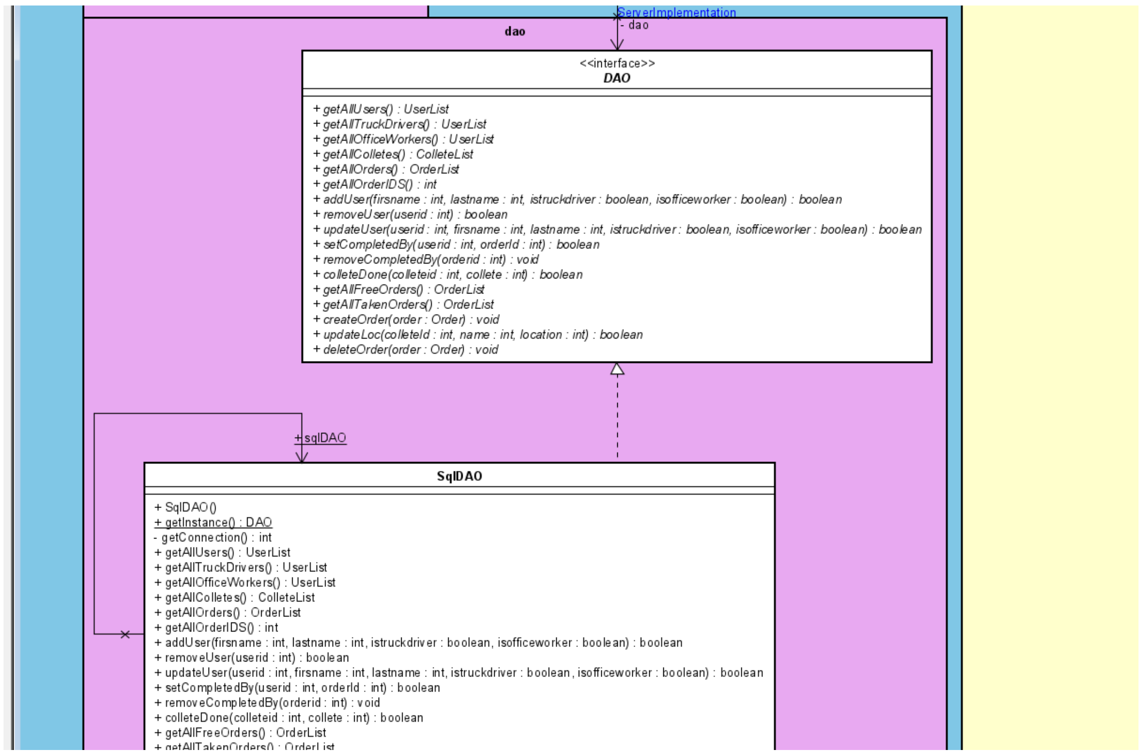
5.3 Liskov Substitution Principle (LSP)

The LSP means that the “is-a” rule is not violated and that deep inheritance should be avoided.


ServerImplementation
- dao
dao
<<interface>>
DAO
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void
+ sqlDAO
SqlDAO
+ SqlDAO()
+ getInstance() : DAO
- getConnection() : int
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList

5.4 Interface-Segregation Principle (ISP)

The LSP means that the clients shouldn't depend on classes that they do not use.

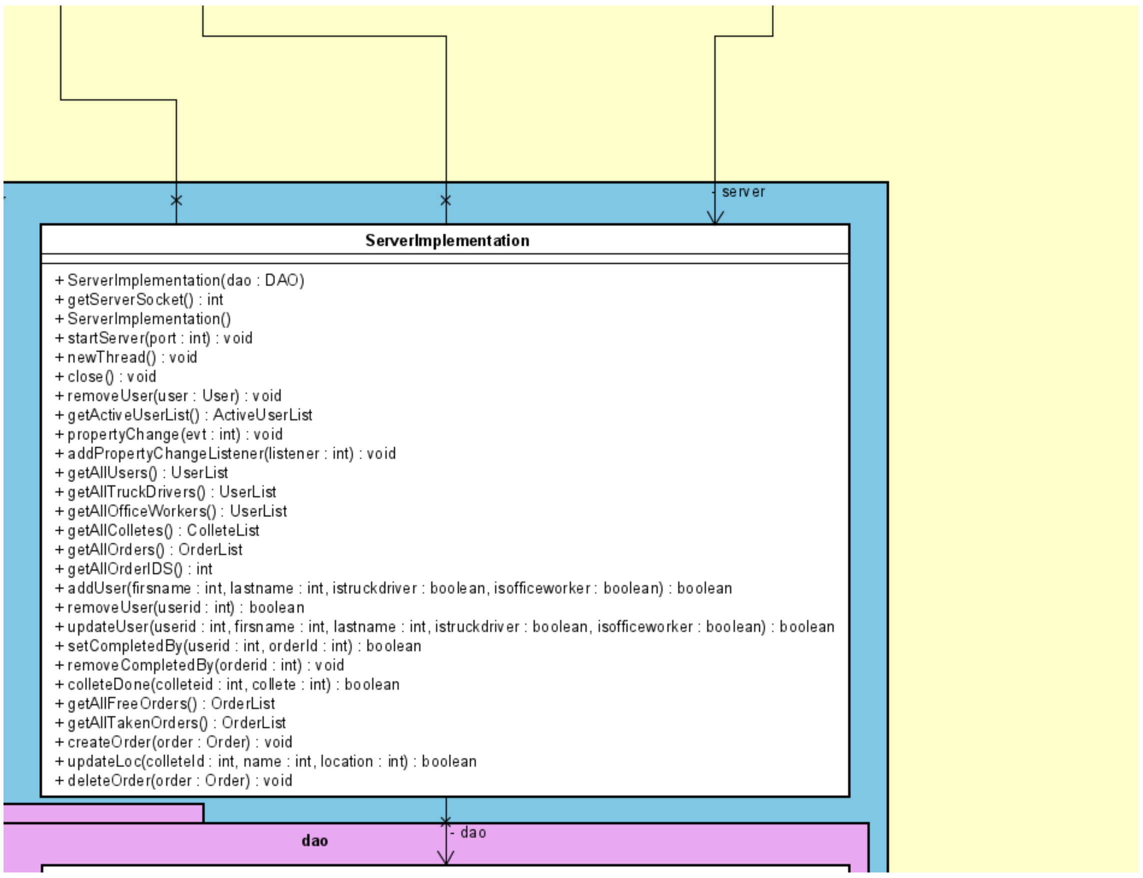


```
<<interface>>
DAO

+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void
```

```
SqlDAO

+ SqlDAO()
+ getInstance() : DAO
- getConnection() : int
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
```

5.5 Dependency Inversion Principle (DIP)

The DIP means that no high-level module should depend on a low-level one. In fact, both should depend on abstractions.


- server
ServerImplementation
+ ServerImplementation(dao : DAO)
+ getServerSocket() : int
+ ServerImplementation()
+ startServer(port : int) : void
+ newThread() : void
+ close() : void
+ removeUser(user : User) : void
+ getActiveUserList() : ActiveUserList
+ propertyChange(evt : int) : void
+ addPropertyChangeListener(listener : int) : void
+ getAllUsers() : UserList
+ getAllTruckDrivers() : UserList
+ getAllOfficeWorkers() : UserList
+ getAllColletes() : ColleteList
+ getAllOrders() : OrderList
+ getAllOrderIDS() : int
+ addUser(firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ removeUser(userid : int) : boolean
+ updateUser(userid : int, firsname : int, lastname : int, istruckdriver : boolean, isofficeworker : boolean) : boolean
+ setCompletedBy(userid : int, orderId : int) : boolean
+ removeCompletedBy(orderid : int) : void
+ colleteDone(colleteid : int, collete : int) : boolean
+ getAllFreeOrders() : OrderList
+ getAllTakenOrders() : OrderList
+ createOrder(order : Order) : void
+ updateLoc(colleteId : int, name : int, location : int) : boolean
+ deleteOrder(order : Order) : void
dao
- dao

6 Test

The purpose of the test section is to document the result of your testing; to verify if the content of the requirements section has been fulfilled. How is the system tested, which strategy has been used; e.g. White Box (Unit Test), Black Box, etc.

In order to check if the system meets the expectations, testing was done during its development. For the testing, the team has used the JUnit framework. During the testing, the Gray Box strategy was used to verify the content fulfils the requirements. The Gray Box strategy implies that the testers have limited knowledge of the system's internal working mechanisms. This way of working was selected, because it was believed to provide a better understanding of the system, while also focusing only partially on it.

6.1 Test Specifications

When it comes to test specification the team has agreed on some test requirements that will need to be covered.

- Each test will need to have goals specified.
- Each test will contain an expected result on how the system would react.
- Each test will have a criteria that will state if the test was successful or not.

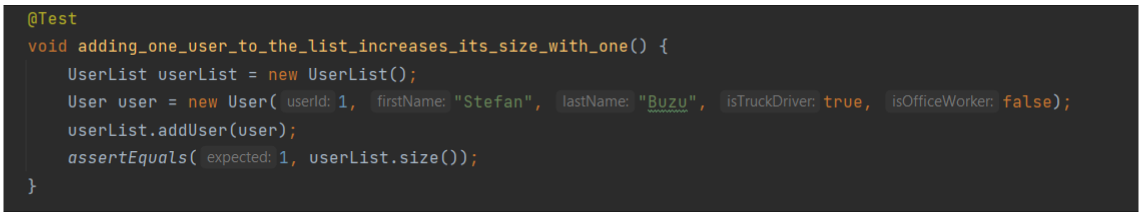
Bellow it is represented a table that covers the log-in use case that will be done by the employee. Moreover, it can be observed the client-system interaction and what happens behind the scenes in the process of logging in. The same principle applies to all the other use-cases.

Step	Action	Reaction	Result
1	Enter the firstname, lastname and the user Id of an existing employee. (John Doe 1)	The system starts to validate the data.	The system assigns the employee to the active-users list.
2	Wait for the system.	The system displays the Request Order menu.	The system moves the employee to the Request Order menu.

6.2 JUnit

As mentioned before, the framework used for testing was JUnit, which allowed the team to properly divide the code into smaller parts, and furthermore test them individually. Moreover, JUnit allowed the testers to get used to test-driven development and its practices.

Bellow, it is illustrated a test case for testing the *UserList* within the system. According to the test requirements stated above, the goal of the test can be understood from the name of the test. Moreover, the expected result of the system is also present. Finally, the JUnit framework allows the testers to check if the test was successful or not, according to their code.



```
@Test
void adding_one_user_to_the_list_increases_its_size_with_one() {
    UserList userList = new UserList();
    User user = new User( userId: 1,  firstName: "Stefan",  lastName: "Buzu",  isTruckDriver: true,  isOfficeWorker: false);
    userList.addUser(user);
    assertEquals( expected: 1, userList.size());
}
```

7 Results and Discussion

In order to better present the results of the project, each use-case needs to be analyzed, along with their outcome. According to the table presented below, the majority of the use cases were completed, nevertheless, one of them was not finished. Overall the core requirements were achieved, moreover, because the SOLID principles were applied, the code has the potential to be maintained and developed more in the future.

When it comes to the Design used for the application, it is easy to pick up, because of the simple concept behind it. Nevertheless, if any issues regarding the use of the system would occur, the user guide will handle them.

Finally, the testing of the use cases was executed on the most important classes, in order to ensure their stability.

Use Case	Actor	Test Result	Comments

Register an employee.	Owner	Works	The Owner registers an employee in the system.
Receive an order.	Owner	Works	The Owner receives an order in the system.
Manage an employee.	Owner	Works	The Owner updates the information of an employee if needed.
Manage an order.	Owner	Works	The Owner updates the information of an order if needed.
Manage active users.	Owner	Works	The Owner can manage the active users.
Request an order.	Employee	Works	The Employee can request an order.
Complete an order.	Employee	Works	The Employee can finish their order, once there are no more collets in the order.
Change language.	Owner/ Employee	Works	The Owner and the Employee can choose to change the language while operating with the system.
Manage Locations.	Owner	Works	The Owner can manage the locations for each collete.
Refil Locations.	Employee	Does not work	The Employee can refil a location if

			there are no more colletes there.

8 Conclusions

The goal of the project was to develop a system that would solve the language barrier present in the warehouse industry in Denmark, as well as provide an alternative for Managing a warehouse. In order to analyze the achievements of the project, it’s important to examine each step that was executed.

The first step of the project was to find functional and non-functional requirements that will ultimately be formulated in form of user-stories. Furthermore, the requirements were analyzed and the next steps occurred.

Design was another key step in the making of the project, where the SOLID principles were used, in order to provide the possibility to maintain the code in the future. As a result of using these principles the software developed became robust, easy to understand and open to changes. Furthermore, another step in the execution of the Design was structuring the MVVM packages, while applying the according principles.

Finally, after implementing the software, it needed to be tested. For this final step, as stated before the team has used the JUnit framework, that was used along with the Gray Box methodology.

To conclude, the main goals of the project were achieved, thus the project can be considered as a success. Nevertheless, there is always room for improvement, thus in the next section there will be presented the aspects that need improvement for future projects.

9 Project future

Our team believes in constant improvement, thus in the process of executing the project, there were identified some things that need changing, in order to become better developers, testers and designers.

The most important aspect that needs improvement is the motivation that is so necessary when working as a team. Because the tasks are connected to one another, the lack of motivation that is present while doing a certain task may impact the other team member’s success later on. Determination is the way to go, moreover what everyone lacks from time to time.

When it comes to software, there is no such concept as perfect, as a result every part of the project can be adjusted. The Design for each tab has been made as simple as possible, nevertheless, an advanced design could provide more professionalism to the application. Moreover, some extra features could be implemented in the system. For example, the Owner may want to have a calendar with the vacations chosen by each employee, thus they can understand in what period of time more manpower would be needed and if they lack employees.

Another feature that would be useful is for the employees to have access to the system, by using their mobile phones. This feature would allow the employees to save more time while working, as a result being more efficient.

Finally, the feature that was not implemented during the sprints would also be useful for the managing of the warehouse. Sending a message that states if a location ran out of collets would again improve the efficiency of the employees, and ease the job of the owner of managing the warehouse.

To conclude, changes are always welcome and can always be added, thus it is important to be open-minded and seek improvement.

10 Sources of information

- **Connolly & Begg]** : Database Systems: A Practical Approach to Design, Implementation, and Management (5th edition or 6th edition), Thomas Connolly and Carolyn Begg , ISBN: 978-1-292-06118-4
- https://balticworkforce.dk/en/warehouses/

11 Appendices

Appendix A - Project description

Appendix B - Analysis

Appendix C - Class diagram

Appendix D - Sequence diagram

Appendix E - Code + JavaDocs + User guide

Appendix F - Database

Appendix G - Other diagrams

Process report

Stefan Buzu, 331907

Alin Abdallh, 331979

Igor Cretu, 331461

Ahmed Omar, 333467

Supervisors: Henrik Kronborg Pedersen, Ole Ildsgaard Hougaard

[Number of characters]

[Software Engineering]

[Semester 2]

[Date: 01.06.2023]

Table of content

1 Introduction

The process report will showcase the team members' feelings and conclusions regarding the way the team worked together to deliver the project. Additionally, we will go into detail about the process of working together.

The team was assembled during the first week of the second semester. Igor and Stefan had already agreed to be in the same group for the project, then Alin and Ahmed joined them, making forming the team easy. Shortly after, a group contract was created and agreed upon together, as a result, the group got started.

As a primary communication tool, we used Messenger for simple questions and scheduling meetings. For meetings, we either met on campus or online to discuss more important topics regarding the project. We agreed that presence during these meetings is a must. Moreover, if any issue arises, we would contact a supervisor as soon as possible to solve the problems quickly before it influences the group's performance.

The learning process was done individually, although, for some of the assignments, we would work together. Nevertheless, when we started working on the project, we would combine our knowledge and help each other out. Soon we discovered how essential and enjoyable teamwork can be. Moreover, each group meeting taught us more and more aspects of group work.

The project was completed using the Scrum framework and Unified Process, which taught the team new ways and disciplines of working together. Moreover, this way of working has proved to be highly efficient when it comes to handling any issues, as well as assigning the workload equally. Nevertheless, we had to get used to the new way of working, as well as using the appropriate tools to achieve our goals.

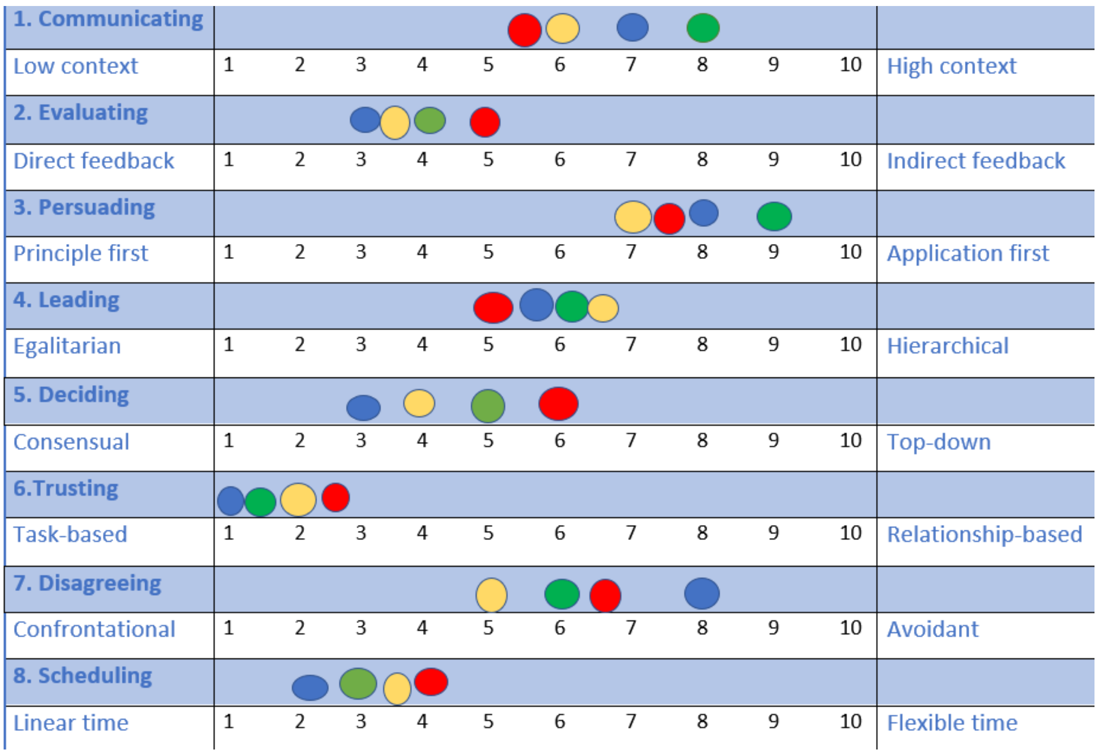
In conclusion, the group work was efficient because we knew from previous experiences that we should handle every issue as soon as possible. Moreover, we quickly agreed on the rules that would suit everyone and create an appropriate professional growth environment.

2 Group Description

Introduction of the group members

- Alin (19 years old, Romania) has experience in working as a group, likes to make new friends, as well as playing different sports.
- Stefan (19 years old, Romania) has experience programming and likes to lift weights as well as practice sports.
- Ahmed (29 years old, Syria) has previous experience in working as a team. He also likes watching movies.
- Igor (20 years old, Romania) has previous experience with Front-end development, as well as fond computer science. In his spare time, he also likes reading books.

Cultural background



Alin – green

Stefan – blue

Igor – yellow

Ahmed – red

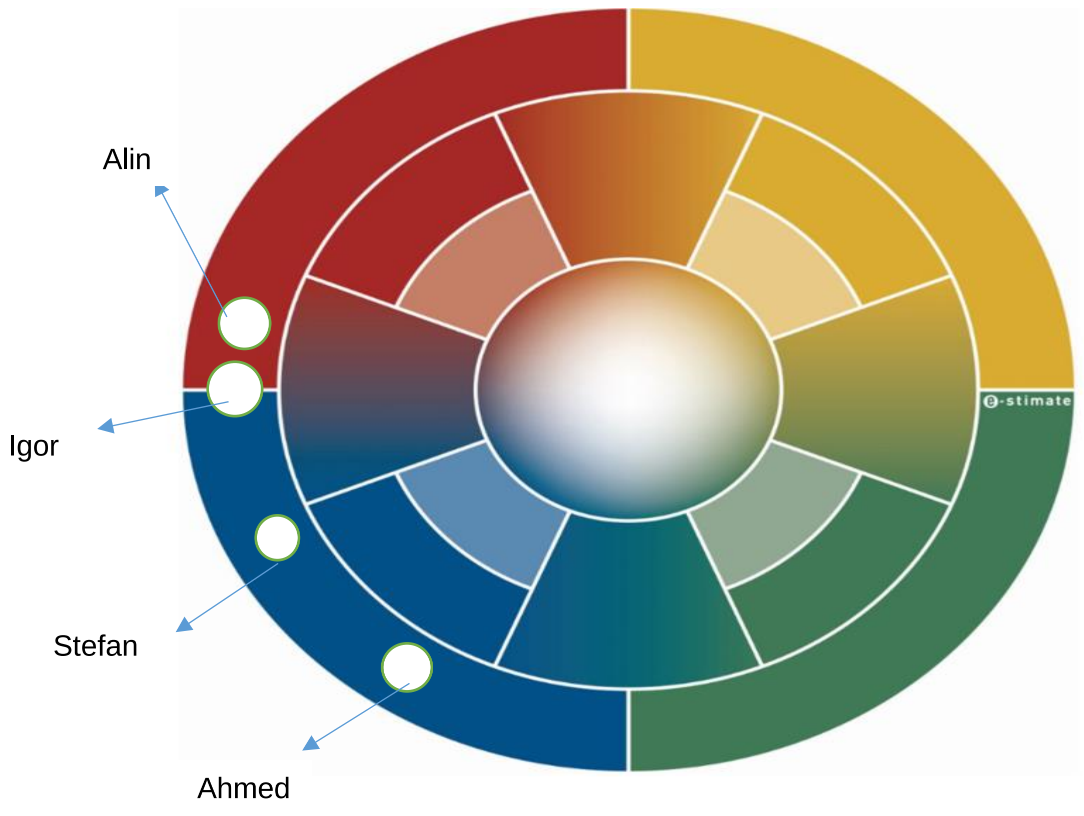
According to the culture map above, our group doesn't have many cultural differences in most of the eight cultural dimensions. This proved to be a big advantage for us in the process of working as a team, because we could anticipate how each one of would react in different scenarios, moreover what role preferences we have in a team.

For example, all of us likes to evaluate by giving direct feedback to each other, moreover the feedback received can be as direct as possible, without worrying about hurting someone's feelings. Moreover, everyone was on the same page when it came up to the "leading and deciding scale", thus any issue that would appear, be discussed consensually, involving everyone in the process, nevertheless sometimes we would let the Product Owner or the Scrum Master decide on the appropriate answer to the problem at hand.

Additionally, our team is very application first and task-based oriented, thus the Scrum framework was picked up easily. Furthermore, we also preferred a linear schedule, where we would focus on tasks one at a time in order to avoid confusion and adhere to the deadlines that we set for each task.

To conclude, our team wasn't very culturally diverse, thus the process of understanding each other was easier. Nevertheless, we understood that still we have some cultural differences, thus the key to pursue our goals together is to understand and discuss problems out, which is the key behind delivering the product successfully.

Personal Profiles



According to the team wheel above, all of us are either red, blue or in between, thus our profiles end up being similar to one another, that ultimately helps us solve problems faster and understand each other's point of view in a better way.

Alin and Igor have a majority red profile, which means they are more direct in their approach towards any issues within the group, whereas Ahmed, who is represented by the colour blue, is trying not to engage in any arguments and deescalate any conflicts. Stefan, being more in the middle, was always focused on the bigger picture, nevertheless he also preferred not to engage in conflicts.

Having a mixture of blue and red worked perfectly in our favour because we had a good mix of being direct in the way we expressed our feelings and opinions, while also realizing when to stop in order for the situation not to escalate, nevertheless find a middle ground when solving an issue. Moreover, we agreed not to take any disagreement personally and to focus on the project before anything else.

In conclusion, analysing the personal reports helped us better understand each other's ways of working in a team, as well as be more productive and avoid unwanted behaviour or conflicts.

Project experience

Besides the experience that every member gathered during the 1st semester project, bellow is presented each member's project experience.

- Alin had previously worked in a team-based projects, thus he had an idea of what it would take and how to work in a team.
- Stefan had a lot of experience in working in a team, both from previous classwork from before university, as well as from working in teams for different projects outside of school.
- Ahmed worked previous jobs that required working as a group, so he was prepared for any project work that would need to be done.
- Igor has previously participated in many volunteer projects, nevertheless he also has experience with freelancing.

3 Project Initiation

During the project initiation, we were informed that the teams can be created without the implication of a supervisor, which played a big role for our team.

The team was initially formed by Igor and Stefan, later Alin and Ahmed were invited to join the team.

After the team was created, the next step that followed was the group contract, that needed to be approved by everyone, thus a meeting was arranged. During the meeting everyone expressed their desires on how the group should function. After each opinion was taken into consideration the group contract was created.

This semester, we had the responsibility of submitting the project proposal, which was decided to be in the form of a warehouse system that needs its management eased by creating a more efficient program that can provide a better work environment for both the manager and its employees.

4 Project Description

The project related task was the project description based on the project proposal that we submitted before.

Firstly, we decided on a day to meet and booked a room, in order to have the meeting take place in there. Then, we used Google Docs to simultaneously work on the same document, in order to work more efficient. Before starting the project description, we analyzed what we want to work on, as well as each member's expectations for the project.

Afterwards, the project description was divided into smaller parts, and each member was assigned with one. After finishing each part, everyone would read their final product, and the rest would give feedback, thus when combining everything everyone would be pleased. This method proved to be very well organized, because the work was done nimble and effective.

Shortly after receiving feedback from the supervisors, another meeting was arranged, in order to adjust the project description, once again the work was done based on discussion and team-work.

In conclusion, the project description is a very important step in the making of the project, because it is the first task we have to do as a team, moreover it is the first chance we have to get used to each other's way of working. Furthermore, the project description got us started on how we would later work on the whole project and taught us what we need to do in order to succeed.

5 Project Execution

Unified Process and Scrum

UP – Inception

During the inception phase, the team-members have defined the most important user-stories within project, moreover a risk-list was elaborated that has the purpose of predicting the possible issues while working on the project. Nevertheless, it was decided that three days sprints would be optimal and would suit every team member. During inception, the use case model was started, and the main requirements were defined. Moreover, after discussing the design for the application, the team has agreed on a specific design that would later be implemented. Another part that was implemented during Inception was the project description, where the team members applied a technique from the Scrum framework, by dividing the work into smaller tasks and assigning them to each member.

UP – Elaboration

During Elaboration, the team focused on identifying the most major user stories, as well as defining them in detail. Moreover, another goal during Elaboration was to set up the significant parts of the architecture for the code, in order to get a clearer view on how the final product will look like. Finally, all the necessary diagrams were created, in order to have a good foundation.

Product Backlog

ID Product	User Story	Priority	Estimated Time
1	As an Owner, I want to register new employees in my system.	High	3 days
2	As an Owner, I want to receive new orders in my system, so they can be assigned to my employees.	High	3 days
3	As an Owner, I want to manage the information about my employees, so it is always updated.	High	3 days
4	As an Owner, I want to manage the information about the orders, so it is always updated.	High	3 days
5	As an Owner, I want to store the data about my employees/orders in a database, so it is easier to use.	High	3 days
6	As an Owner, I want to manage my active workers, so I can keep track of them and their orders in real time.	High	3 days

7	As an Owner, I want to remove employees from my system, in case they want to leave or I have to fire them.	Low	1.5 days
8	As an Owner, I want to start my system both on a Local Host, as well as a Local Area Network, so I can operate with the system both in my office and remote.	Low	1.5 days
9	As an Owner, I want to keep track of the amount of boxes that I have in the warehouse, so I can refill the locations if needed.	Low	3 days
10	As an Owner, I want to have my system both in English and in Danish, so all my employees have an easy time operating with it	Low	3 days
11	As an Owner, I want to have all the steps implemented within the system documented, so I can	High	3 days

	better understand how it works.		

1st Sprint

May 2nd – May 4th

Sprint Backlog

ID Product	ID Sprint	Task	Estimated Time	Person
1	1	Design View	1 day	Ahmed
1	1	Implement model + view-model	1 day	Igor
5	1	Implement database	1 day	Stefan
1	1	Testing	1 day	Alin
2	1	Design View	1 day	Alin
2	1	Implement model + view-model	1 day	Igor
5	1	Implement server + database	1 day	Stefan
2	1	Testing	1 day	Ahmed
3	1	Design View	1 day	Ahmed
3	1	Implement model + view-model	1 day	Stefan
5	1	Implement database	1 day	Igor
3	1	Testing	1 day	Alin

Sprint Planning

ID Product	Task	Person	Estimated Time
1	Design View	Ahmed	1 day
1	Implement model + view-model	Igor	1 day
5	Implement database	Stefan	1 day
1	Testing	Alin	1 day
2	Design View	Alin	1 day
2	Implement model + view-model	Igor	1 day
5	Implement server + database	Stefan	1 day
2	Testing	Ahmed	1 day
3	Design View	Ahmed	1 day
3	Implement model + view-model	Stefan	1 day
5	Implement database	Igor	1 day
3	Testing	Alin	1 day

Daily Scrum

Day	Success	Igor	Stefan	Ahmed	Alin
1	Done/In process	Done	Done	Done	Done
2	Done/In process	Done	Done	Done	Done
3	Done/In process	In process	In process	Done	In process

Sprint Review

ID Product	Task	Real Time	Progress	Notes
1	Design View	1 day	Done	-

1	Implement model + view-model	1 day	Done	-
5	Implement database	1 day	Done	-
1	Testing	1 day	Done	-
2	Design View	1 day	Done	-
2	Implement model + view-model	1 day	Done	-
5	Implement server + database	1 day	Done	-
2	Testing	1 day	Done	-
3	Design View	1 day	Done	-
3	Implement model + view-model	-	In progress	Because of the insufficient time the task was not finished.
5	Implement database	-	In progress	Because the model was not ready, the implementation of the database was not completed.
3	Testing	-	In progress	Because the implementation was not finished the testing could not be done.

Sprint Retrospective

Comments	Solutions
The Sprint planning should be more accurate in the future. Because of the poor planification and the unexpected issues with the implementation, caused the team members to not be able to finish their tasks on time.	In the future the team will analyze the tasks that need to be done more in depth in order to prevent any unexpected problems.

2nd Sprint

May 9th – May 11th

Sprint Backlog

ID Product	ID Sprint	Task	Estimated Time	Person
3	2	Implement model + view-model	1 day	Stefan
5	2	Implement database	1 day	Igor
3	2	Testing	1 day	Alin
4	2	Design View	1 day	Igor
4	2	Implement model + view-model	1 day	Alin
5	2	Implement database	1 day	Stefan
4	2	Testing	1 day	Ahmed
6	2	Design View	1 day	Alin

6	2	Implement model + view-model	1 day	Stefan
5	2	Implement database	1 day	Igor
6	2	Testing	1 day	Ahmed
7	2	Design View	1 day	Ahmed

Sprint Planning

ID Product	Task	Person	Estimated Time
3	Implement model + view-model	Stefan	1 day
5	Implement database	Igor	1 day
3	Testing	Alin	1 day
4	Design View	Igor	1 day
4	Implement model + view-model	Alin	1 day
5	Implement database	Stefan	1 day
4	Testing	Ahmed	1 day
6	Design View	Alin	1 day
6	Implement model + view-model	Stefan	1 day
5	Implement database	Igor	1 day
6	Testing	Ahmed	1 day
7	Design View	Ahmed	1 day

Daily Scrum

Day	Success	Igor	Stefan	Ahmed	Alin
1	Done/In process	Done	Done	Done	Done

2	Done/In process	Done	Done	Done	Done
3	Done/In process	Done	Done	Done	Done

Sprint Review

ID Product	Task	Real Time	Progress	Notes
3	Implement model + view-model	1 day	Done	-
5	Implement database	½ day	Done	-
3	Testing	1 day	Done	-
4	Design View	1 day	Done	-
4	Implement model + view-model	1 day	Done	-
5	Implement database	1 day	Done	-
4	Testing	1 day	Done	-
6	Design View	1 day	Done	-
6	Implement model + view-model	1 day	Done	-
5	Implement database	1 day	Done	-
6	Testing	1 day	Done	-
7	Design View	½ day	Done	-

Sprint Retrospective

Comments	Solutions
The overall result of the sprint was a success, however because some tasks were	In the future the team will take a closer look at the estimated time for each assigned task.

similar to the ones from the previous sprint, they took less time to do, as a result the estimated time was not 100% accurate.	

UP – Construction

During the Construction phase of Unified Process, our team has focused on rounding up the important user-stories that were defined previously, as well as starting to create the necessary templated, in order to continue with the documentation part of the project. Moreover, during construction, the team focused on finishing and preparing the system for deployment, which means that any additional features would be added only if time will not be a problem. As usual, during this phase, the sprints will be documented, in order to provide a clear view on what was achieved during this period.

3d Sprint

May 16th – May 18th

Sprint Backlog

ID Product	ID Sprint	Task	Estimated Time	Person
11	3	Create the templates for the documentation.	1 day	Ahmed
7	3	Implement model + view-model	½ day	Alin
5	3	Implement database	½ day	Stefan
7	3	Testing	½ day	Igor
8	3	Design View	1 day	Igor

8	3	Implement model + view-model	1 day	Alin
5	3	Implement database	1 day	Stefan
8	3	Testing	1 day	Ahmed
9	3	Design View	1 day	Stefan
9	3	Implement model + view-model	1 day	Ahmed
5	3	Implement database	1 day	Igor
9	3	Testing	1 day	Alin

Sprint Planning

ID Product	Task	Person	Estimated Time
11	Create the templates for the documentation.	Ahmed	1 day
7	Implement model + view-model	Alin	½ day
5	Implement database	Stefan	½ day
7	Testing	Igor	½ day
8	Design View	Igor	1 day
8	Implement model + view-model	Alin	1 day
5	Implement database	Stefan	1 day
8	Testing	Ahmed	1 day
9	Design View	Stefan	1 day
9	Implement model + view-model	Ahmed	1 day
5	Implement database	Igor	1 day
9	Testing	Alin	1 day

Daily Scrum

Day	Success	Igor	Stefan	Ahmed	Alin
1	Done/In process	Done	Done	Done	Done
2	Done/In process	Done	Done	Done	Done
3	Done/In process	In process	Done	In process	In process

Sprint Review

ID Product	Task	Real Time	Progress	Notes
11	Create the templates for the documentation.	1 day	Done	-
7	Implement model + view-model	½ day	Done	-
5	Implement database	½ day	Done	-
7	Testing	½ day	Done	-
8	Design View	1 day	Done	-
8	Implement model + view-model	1 day	Done	-
5	Implement database	1 day	Done	-
8	Testing	1 day	Done	-
9	Design View	1 day	Done	-
9	Implement model + view-model	-	In process	Because of the complexity of the

				task, it was not finished.
5	Implement database	-	In process	Because the model was not ready, the implementation of the database was not completed.
9	Testing	-	In process	Because the implementation was not finished the testing could not be done.

Sprint Retrospective

Comments	Solutions
Some of the team-members were not able to finish their task on time, due its complexity.	In the future the team will engage more in helping each other out, as well as provide better answers, when asked for help.

UP – Transition

During Transition, the system was ready for deployment, thus only some minor changes were made. Moreover, our team was focused on finishing the documentation part, in order to have a well-documented system with every aspect of it covered in the reports. Even though the tasks were assigned separately, each member was constantly providing and at the same time receiving feedback, in order to have everyone present during the making of each step of the reports.

4th Sprint

May 23nd – May 25th

Sprint Backlog

ID Product	ID Sprint	Task	Estimated Time	Person
11	4	Work on the first part of Process Report	1 day	Stefan
11	4	Work on the first part of Process Report.	1 day	Igor
11	4	Work on the second part of Process Report.	1 day	Ahmed
11	4	Work on the second part of Process Report.	1 day	Alin
11	4	Work on the first part of Project Report.	1 day	Igor
11	4	Work on the first part of Project Report.	1 day	Stefan
11	4	Work on the second part of Project Report.	1 day	Alin

11	4	Work on the second part of Project Report.	1 day	Ahmed
11	4	Analyse your colleagues' work and provide feedback.	½ day	Stefan
11	4	Analyse your colleagues' work and provide feedback.	½ day	Ahmed
11	4	Analyse your colleagues' work and provide feedback.	½ day	Igor
11	4	Analyse your colleagues' work and provide feedback.	½ day	Alin
11	4	Make changes according to the feedback received.	½ day	Stefan
11	4	Make changes according to the feedback received.	½ day	Ahmed
11	4	Make changes according to the feedback received.	½ day	Igor
11	4	Make changes according to the	½ day	Alin

		feedback received.		

Sprint Planning

ID Product	Task	Person	Estimated Time
11	Work on the first part of Process Report.	Stefan	1 day
11	Work on the first part of Process Report.	Igor	1 day
11	Work on the second part of Process Report.	Ahmed	1 day
11	Work on the second part of Process Report.	Alin	1 day
11	Work on the first part of Project Report.	Igor	1 day
11	Work on the first part of Project Report.	Stefan	1 day
11	Work on the second part of Project Report.	Alin	1 day
11	Work on the second part of Project Report.	Ahmed	1 day
11	Analyse your colleagues' work and provide feedback.	Stefan	½ day
11	Analyse your colleagues' work and provide feedback.	Ahmed	½ day
11	Analyse your colleagues' work and provide feedback.	Igor	½ day

11	Analyse your colleagues' work and provide feedback.	Alin	½ day
11	Make changes according to the feedback received.	Stefan	½ day
11	Make changes according to the feedback received.	Ahmed	½ day
11	Make changes according to the feedback received.	Igor	½ day
11	Make changes according to the feedback received.	Alin	½ day

Daily Scrum

Day	Success	Igor	Stefan	Ahmed	Alin
1	Done/In process	Done	Done	In process	Done
2	Done/In process	Done	Done	In process	Done
3	Done/In process	In process	In process	In process	Done

Sprint Review

ID Product	Task	Real Time	Progress	Notes
11	Work on the first part of Process Report.	1 day	Done	-
11	Work on the second part of Process Report.	1 day	Done	Not all the team-members have done their part.

11	Work on the first part of Project Report.	1 day	Done	-
11	Work on the second part of Project Report.	1 day	Done	Not all team-members have done their part.
11	Analyse your colleagues' work and provide feedback.	½ day	Done	Due to not everyone completing their tasks', providing feedback was also not possible.
11	Make changes according to the feedback received.	½ day	Done	Due to not everyone completing their tasks', making changes according to the feedback was not possible.

Sprint Retrospective

Comments	Solutions
Due to the previous experience with sprints, there were no unexpected problems during this one. As a result, every task was completed and the documentation was done successfully. Nevertheless, some team-members have had some problems and could not complete their tasks, as a result	In the future we will keep working by providing support, as well as using the feedback received from the other team-members to improve.

some other team-members had to complete their task.	

6 Scrum Overview

While implementing the project, the SCRUM methodology was used. In order to respect its principles our team has chosen a Scrum Master, as well as a Product Owner, that would have different tasks according to their status. After analyzing how the SCRUM team works, there will be presented the Burndown Chart.

6.1 Sprint Planning

The purpose of the sprint planning phase was to assign tasks for each member, thus create an overview of how the daily scrum would look like. After discussing the user stories, it was decided to start with the most important requirements. Nevertheless, each member has presented their preferred tasks, and as a result the tasks were assigned accurately to each individual. This aspect has proven to be effective in providing a shared responsibility within the team, leading to collaboration and high-performance.

6.2 Daily Scrum

The Daily Scrum in the team was organized online, because the team-members figured this would be more efficient and save time for everyone. During these meetings, each member was attending with their cameras on, in order to provide more transparency and for creating a friendly and a safe environment for sharing opinions. Moreover, the team was encouraged to attend the meeting by standing up, in order to keep the topic to the point. The Scrum Master was also ensuring that everyone respects the format of the meeting, as well as the time-limit. As a result, any questions that were issued during the meeting were later discussed outside the meeting.

6.3 Sprint Review

After finishing the sprint, it was time for each member to share their thoughts on how the work was done, and their opinions overall. During the Sprint Review, the team along with the Product Owner and the Scrum Master have analyzed the achievements during each sprint. The meeting was always kept simple and to the point by the Scrum Master.

6.4 Sprint Retrospective

The Sprint retrospective occurred almost immediately after the Sprint Review, in order to apply the feedback acquired. Moreover, a problem that was solved during these meetings was communication, because when everyone is busy with their tasks it can be hard or uncomfortable to ask for help. As a result, everyone agreed to help each other as much as they could, thus to promote a safe environment for growing together. The final goal of these meetings was to improve for the future sprints.

6.5 Scrum Master

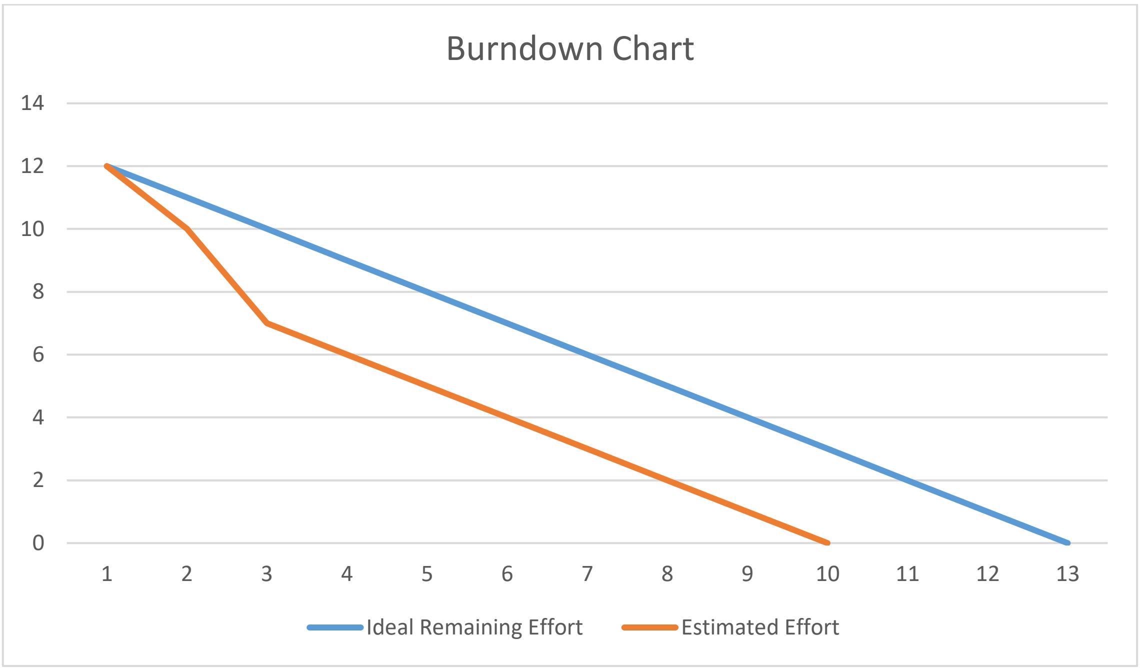
For this project, the team has voted for Stefan to be the Scrum Master, his duties were to solve conflicts, as well as making sure that everyone is respecting the Scrum principles while working. The Scrum Master was also responsible for the Scrum events and their effective organization and execution. Moreover, the Scrum Master would provide guidance for the team members when it comes to applying the necessary values and principles. Stefan was the one who encouraged the team to keep improving, by promoting the Scrum retrospective and other major tools within the Scrum framework.

6.6 Product Owner

The Product Owner for the project was again voted to be Igor. He was the one who analyzed the project’s vision and its constant progress. The Product Owner was the one responsible for creating the Product Backlog, which is a prioritized list of all the features that were implemented within the project. Because of his long-term vision, Igor was able to set goals for the team and provide directions when needed. He was also constantly helping the Scrum Master with the planning of the Sprint Planning and the Sprint Review. Even though the team was not that big, the Product Owner was responsible for accepting the tasks for each team-member and providing feedback.

6.7 Burndown Chart

Finally, the Burndown Chart presented bellow will showcase the team’s success overtime and what did the team manage to achieve in the end. The team had four sprints, each one consisting of three total days. The team has agreed on using days in the process of making the chart, as it made more sense for everyone. It can also be noticed that the estimated effort did not match the Ideal Remaining Effort Line, which is common in the Scrum framework. In the process of making the project, the Scrum Master was the one to analyze the Burndown Chart, in order to keep track of how the things are evolving overtime. Overall, this chart servs as a way of communication, that allows the team to understand how well are they completing the tasks overtime. Moreover this key tool, allows better time management within the team.



7 Personal Reflections

Bellow there will be presented the personal reflections of each team member. The personal reflection aims to illustrate how the team members felt during the semester, as well as during the project period. Each individual will present their views on how things went during the semester.

Igor Cretu

Every new semester at VIA means a new challenge, and the second one was no exception. Therefore, a new challenge is always a push to learn new things and methods that will help us achieve our goals.

During this semester, we learned about how complex an application can really be and how architecture models like MVVM help us keep the code clean, making each piece easily removable or updatable. We also learned how to work with and connect a SQL database to a Java program, which, at first sight for a newbie, seems impossible due to the differences between these two programming languages. Thus, we had to plunge into self-learning and search for solutions on different platforms, which led us to great new ideas.

In class, we were introduced to many new subjects that were more complex and high-level compared to the ones we had last year. There were many subjects I hadn't thought of before, despite having some experience in programming. Two of them were the MVVM application architecture and the GoF patterns, which were not easy to understand initially due to the complexity of dependencies they rely on. I thought everything would be as easy as during the first year, and this was my biggest mistake. Firstly, I started procrastinating and doing other stuff instead of studying, thinking that I could easily recover later. Then I started leaving the lessons earlier without finishing my tasks. As a result, I allowed myself to procrastinate while the small snowball of missed subjects started rolling down the hill until it became an avalanche. When I finally understood what I was doing wrong, it was already too late, so I had to study each of the subjects one by one by myself in a fast-paced way in order to keep up with my team and classmates. This led to several sleepless nights where a caffeinated student tries to do three times the work he could have done. During this period, I understood that I should never overestimate myself but think and act as rationally as possible to avoid such situations in the future.

The group work for the SEP project went smoothly. We never had any conflicts, even though we didn't interact much during the semester and didn't know much about each other. But working together helped me realize that I'm surrounded by really nice people who, instead of arguing, discuss things calmly and bring their bright ideas to life. It is logical that each of us is completely different, and someone can be better at something while the other one can be better at something else. Thus, we tried combining our forces and splitting the work properly depending on our talents.

In conclusion, this year taught me that overestimating myself isn't the best idea; however, it pushed me to learn on my own, which offered me a new spectrum of ideas and methods beyond what we were taught. Additionally, I gained experience working in a group, which was something unusual for me before coming to VIA.

Stefan Buzu

This semester was a bit more difficult than the previous one, because of the more complex material. I was introduced to the concept of self-learning that seemed strange to me before, as well as new ways of working within a team. I had to learn to constantly motivate myself, in order to learn every new material, however this new experience prepared me for working during the project period.

When it comes to the classes, I am glad for having each one of my teachers that were always happy to help and explain each question in depth. The DBS classes seemed easy, because the material was always fully explained. The knowledge gained during these classes, were directly used for the SEP project.

The SDJ classes, were more challenging because they seemed to always bring something new, moreover I had to keep up with the material. Nevertheless, consistency was the skill I had to improve in order to succeed. Even though the class itself was demanding, the teacher has helped me stay on the right path.

Finally, the SEP and the SWE classes, have taught me the methodologies and ways of working that I would ultimately use during the SEP project. I think all the knowledge gained during these classes will help me work on future projects, even outside the university.

To conclude, this year was exciting and full of challenges, however in the end I felt that by completing more complex tasks I was acquiring more knowledge. Ultimately, I liked working by the new methodologies and organizing thing different with the team-members. I think I learned more things during this semester, moreover I am excited to the future projects that I will be working on.

Alin Abdallh

The second semester was, for me, very different from the first one. First of all, before any project work even started, I struggled a bit because I missed the first two weeks of tuition, and it took me a while to get back and recover. Also, by this time, we also got the first SDJ assignment and I struggled a bit in doing it.

After I got through the very tough beginning to the semester, the project work started, and we had to do the project description. That proved to be an easy task in the end, we gathered together and did the task in a short time.

Fast-forwarding to the project period, we started dividing the work and we started working on the project. Because we had similar cultural background, we didn't find any problems in organizing and working together. I personally think that each member did their work and contributed to the project in their best way. Also, anytime each one of us finished a part of the project, he would present it to us to look over and change anything that we deemed wrong.

My profile colour was red which meant I was very keen on giving direct feedback to anyone who came to me for an honest opinion, and I felt that the others would be sincere to me as well. The experience from the first semester project proved to be very useful in a lot of scenarios. Firstly, two of my teammates from this semester were my teammates last semester also. Secondly, I think that even though my profile colour being red means that I am very direct and non-patient, the first semester taught me to be a bit more patient and to understand that everyone has their own way of working and in the end, the project is the only thing that matters.

In conclusion, even though the semester didn't start very well for me, the experiences gathered from the previous project and also the SEP lessons on different profiles and cultural background made the project period go a lot smoothly than before and in the end it was overall a success.

Ahmed Omar

8 Supervision

The supervisors were very important to us during the semester. We didn’t request any supervisor meetings, but we did ask them for help when it came to issues related to the code or reports to which they were glad to help and guide us.

The SDJ and DBS teachers helped us a lot not only during the project period, but also beforehand, every time a problem would occur, they were glad to help and put us on the right path. Moreover, the material taught during the lessons, was fully explained and applied in practice, so there was no room for failure.

The SWE teacher helped us better understand the SCRUM framework and Unified Process, moreover how to use their disciplines to our advantage. Nevertheless Mr. Henrick, was always trying to let us find the right answers, by providing small hints to our questions. This way of supervising, kept us in a continuous hunt for improve.

On the other hand, Mr. Olé, was helping us solve all the code problems, thus preparing us to deliver the code part for the final product. During the SDJ lessons, we were introduced to the necessary techniques, as well as their implementation, that would later be applied for certain tasks. Once again, every issue was promptly solved, and all questions answered due to the teacher’s interest in helping.

In conclusion, our supervisors helped us a lot, both during the tuition and the project period. They were always ready to help, thus we were confident that any problem that would appear would be solved.

9 Conclusions

To sum up, our project this semester was a success, we understood each other's differences and we took them into account in order to avoid conflicts and we proved to be a well-functioning group, where everyone did their part and respected the group rules that were agreed upon in the group contract.

From the beginning, we agreed on who was going to be the Scrum Master and the Product Owner, as well as how the work process will look like. We used Messenger in order to set deadlines and arrange meetings, as well as Zoom to have our Scrum meetings.

Because some of us had previous experience with working with each other, we knew how to set our expectations, moreover, how to work as a team. Mutual respect was always a must, even while disagreeing on certain topics, thus everyone's opinions were taken into consideration in the process of taking decisions. We can see these characteristics also in the leading and deciding scales in the Culture Map.

The Scrum framework helped us a lot with the organization, because every task was done and assessed during the sprints, which provided a clear view of how everyone is doing and how to constantly improve. Each sprint helped us understand and work better as a team, thus by the end of the project period we were working in perfect harmony which helped the team deliver the desired product on time.

To conclude, this project taught us that everyone has their own way of working as a team based on their profile and cultural background, which is an important aspect to treat each other equally and with respect in order to come to a middle ground and overcome all the problems.

Appendices