VIA University College

IT-SEP2+SW-SEP2 - Semester Project 2 - re exam - August 2022

Oral exam in room A04.12B

Predefined Information

Start date:	2022-08-09 12:00 PM	**Grading scale:**	Danish 7-point scale
End date:	2022-08-12 01:00 PM	**ECTS:**	10
Flow code:	PH06650511		
Internal assessor:	Steffen Vissing Andersen		
Internal assessor:	Troels Mortensen		
Internal assessor:	Jørn Martin Hajek		
Internal assessor:	Henrik Kronborg Pedersen		

Participant

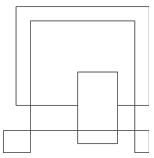
Name:	Tomás Gres
Candidate No.:	024d4ba8-c417-ed11-a316-005056b27d6c
VIA-id:	315185@viauc.dk

Information from participant

Declaration of Honesty Yes
***:**

Group

Group name:	Group XZ
Group number:	6
Other members:	Paul-Mihai Cosarca, Bartosz Zielinski, Eduard-Gabriel Vlad



Winter Rental Store

Development Team:

Tomás Gres - 315185

Gabriel Vlad - 315193

Bartosz Zielinski - 308820

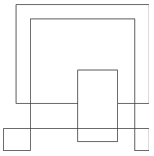
Paul Mihai - 308834

37569 characters

Software Technology Engineering

Semester II

12.08.2022



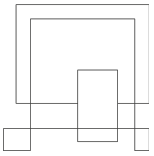
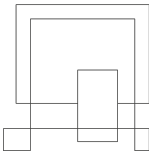
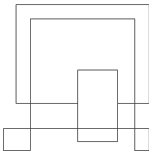


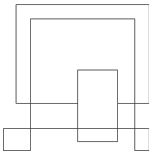
Table of content





- Abstract

The aim of this project is to make renting of equipment in the customer's store as simple and almost effortless. We chose Unified Process and Scrum to guide us through the development of the project. For the project itself we chose Java and JavaFX for the application, Remote Method Invocation (RMI) for the connection between the server and the client and PostgreSQL (scripting language) and ElephantSQL (database provider) for the database. The testing of the requirements will be done using Test Scenarios and Test Cases.

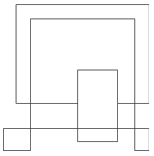


- 1. Introduction

The winter equipment rental company is set to open in 2022, in the Villars-sur-Ollon. Winter Olympia rental will be offering a variety of equipment for the winter season. Based in the Villars-sur-Ollon area, this new store will serve both international and local customers. The company's founder Herman Maier, believes that each of his customers deserves an experience in the mountains, and they would be able to achieve it by using the amazing products that they are offering. (Background Description)

The purpose of this project is to give Mr. Herman Maier a software solution that is going to help him manage his company. He expects that the software is going to be easy to access and will meet his requirements. The system will have multiple types of users: a manager and employees, which will have the ability to access the system based on their credentials. The employee being at the front desk is able to create reservations and handle the customers information. And the manager being able to have control of the software he is able to add and remove new employee into the system, create different equipment into the system, edit existing ones and remove the items that are not in stock anymore.

Based on the discussion with the customer to clarify different aspects such as payment, it was concluded that the payment process is it going to be handled by a different system.



2. Analysis

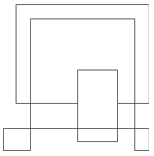
The purpose of the Winter Rental Store application is to simplify management of renting the equipment in the customer's store, as well as management of the employees, store customers and equipment itself.

The system also have to contain two different roles: Employee and Manager. Employee will be responsible for managing renting and customers, and Manager will be responsible for managing employees and equipment. A manager will also be able to perform actions employee can if that becomes necessary.

2.1. Functional Requirements

2.1.1. Critical Priority

1 As a manager I want to have a default account which I can access and log into using a username and a password, in order to be able to use the system.
2 As a manager I want to be able to add a new employee account specifying the first and last name, email, username and the password so that a new employee can use the system.
3 As an employee I want to be able to log in using a username and a password, in order to be able to use the system.
4 As a manager I want to be able to add new equipment to the store specifying the id of item, name (generally brand and model), type (skis, ski poles, ski boots, snowboard, snowboard boots, goggles, helmet or a custom type), size (skis, poles, snowboard - length in cm, snowboard/ski boots - shoe size in EU standard, goggles, helmet - age), price for a day of renting and how many pieces of this equipment (with each of the items having a generated unique inventoryId) I purchased for the shop and added, so that a customer can rent it.
5 As an employee I want to be able to add a new customer specifying his/her name, surname, phone number, email and passport/id number so that he/she will be able to rent equipment.



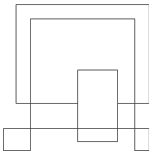
6 As an employee I want to be able to see and filter the list of items in the inventory using start and end date and time of the rent, type of the equipment, size and price per day in order to find a specific item and rent it to the customer.
7 As an employee, I want to be able to add and remove equipment from a “shopping basket”, in order to select everything the customer wants to rent
8 As an employee I want to be able to create new renting from the equipment in a shopping basket to a customer for a period of time, in order to rent items to the customer.

- 2.1.2. High Priority

9 As an employee I want to be able to search for a given customer using their email, phone number or passport/id number from the list of customers so that I can finish his rent.
10 As an employee I want to be able to remove one or multiple items from the items a customer rented, so that a customer doesn’t have to return everything at once.
11 As a manager I want to be able to perform any action the employee can, in order to be able to oversee/replace the employee if necessary
12 As a manager I want to be able to search for equipment using name and type and remove equipment from the store so that I can get rid of unavailable items.
13 As a manager I want to be able to remove old employee accounts so that I can remove unused accounts.
14 As an manager I want to be able to see all of the current and previous rentals of an item, in order to see information about the previous use of the items

- 2.1.3. Low priority

15 As a manager, I want to be able to see all the log in and log out activity of each of the employees and how many hours they worked in order to know how much time they spent using the system.
16 As an employee, I want to be able to set the description and state (damaged, undamaged) of an item after it was returned, in order to document if there was any damage done on it.



17 As a manager, I want to be able to search for damaged items, in order to easily remove damaged items from the inventory
18 As a manager, I want to be able to see the list of all the employees of the store showing their first name, last name, email and username, in order to know about all of the employees that work for the store
19 As a manager, I want to be able to change employee account password, in order to create new access phrase for the employee if he/she forgets it

- 2.2. Non-Functional Requirements

1. Program must support English as the main language.
2. Program must be able to run on Windows 10

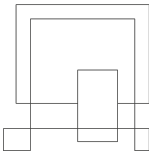
- 2.3. Actor Description

- 2.3.1. Primary Actors

- **Employee -** An employee creates accounts for customers and manages renting items from the inventory to them.
- **Manager -** A manager creates/removes accounts of employees, and is responsible for adding new pieces of equipment to the inventory and removing old/damaged equipment from the inventory

- 2.3.2. Offstage Actors

- **Customer -** A customer rents equipment from the store and provides his personal information to the employee in order to have his account created and to be able to rent equipment (note: a customer does not operate the system directly)



- 2.4. Use Case Diagram

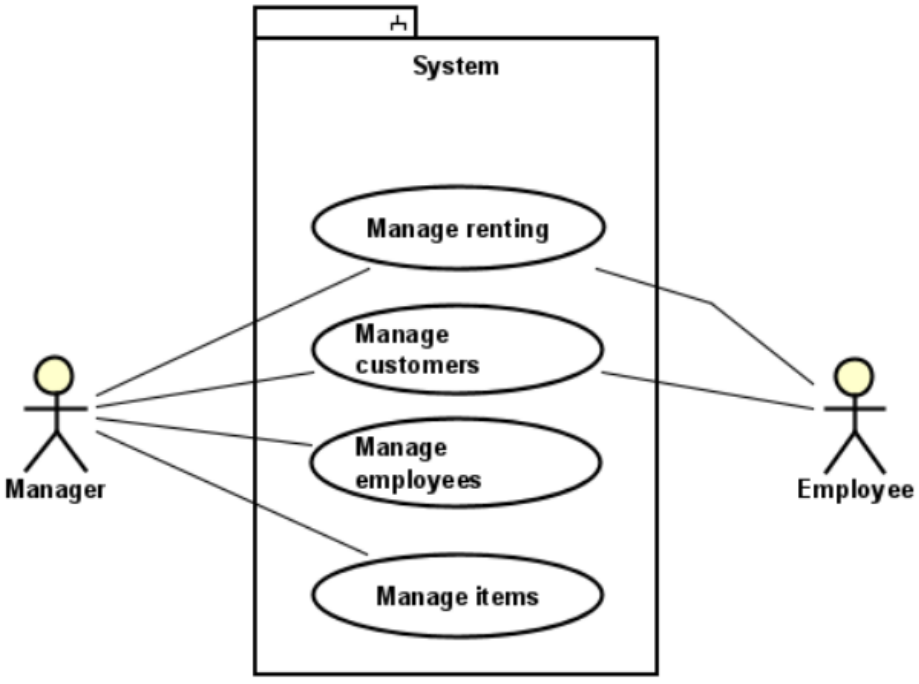
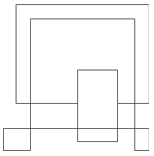


Figure 1 - Use Case Diagram

- 2.5 Use Case - Brief Description (Manage Renting)

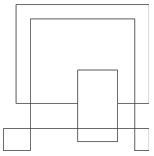
A customer comes in to the store to rent a specific piece (or multiple pieces) of equipment (e.g. skis) for a certain period of time. An employee of the shop will then use the system to search for equipment that fits the customers' needs and is not already rented at the given time period. If there is such a piece of equipment, the employee selects it and adds it to the shopping basket. When he selected all the desired items, he sets the start and finish time and date of renting. If the customer already has an account in the store, the employee can search it by his passport/id number, phone number or email. Otherwise a new customer account has to be created (Manage Customers use case). When the customer returns the equipment, the employee will check its state and optionally write a brief description about it (usually in case it is



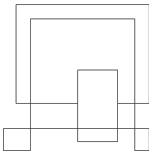
damaged). After that depending on the state of the equipment, the employee sets it as either damaged or undamaged.

- 2.6 Use Case - Fully Dressed Description (Manage Renting)

Use Case	Manage Renting
Summary	A manager/employee selects items from the inventory to a “shopping cart” and creates a new rent from these items for a customer or confirms receival of the items from the customer and closes/concludes the rent
Actor	Manager/Employee
Precondition	A manager/employee is logged in
Postcondition	A new rent is added to the system or all (or a part of) items are returned and the rent is closed/concluded
Base Sequence	1. CREATE RENT a. A manager/employee chooses “Manage Rent” option in the menu b. A manager/employee chooses “Create Rent” option in the menu c. The system shows the inventory of items d. A manager/employee enters information about a specific item: i. start date and time of the rent [datetime] ii. end date and time of the rent [datetime] iii. type of the equipment [selection] iv. size [whole number] v. price per day [decimal number] e. A manager/employee selects “Search” option f. The system shows items that correspond with the given information g. A manager/employee then selects an item h. A manager/employee selects “add to shopping cart” option i. A manager/employee selects “shopping cart” option j. The system shows the details of the rent including start and end date time and all the selected items k. A manager/employee selects “search customer” option



	l. The system shows the list of customers with the ability to filter through them m. A manager/employee the executes steps 2b-2f in the Manage Customer Use Case n. The system shows the information about the rent and adds information about the customer o. A manager/employee selects “Confirm” option p. The system shows a success message 2. SEARCH / CLOSE RENT a. A manager/employee selects “Manage Rent” option in the menu b. A manager/employee enters customer’s email, passport number or phone number c. The system shows all of the rents from the customer d. A manager/employee selects the rent he/she wants to close e. A manager/employee selects the “Close” option f. The system shows a message that the rent was successfully closed
Branch Sequence	Branch 1a - Instead of Step 1a and 1b A. The employee/manager can go through steps 2a-2f in the use case - Manage Customer Branch 1b - Step 1j A. The employee/manager went through Branch 1a instead of steps 1a and 1b B. Steps 1k to 1m will be skipped Branch 2 - Step 1d, 1e and 1f A. The manager/employee can enter no information apart from start and end date (those are necessary) B. The system then shows all of the items available at that time frame Branch 3 - Step 1k A. The employee/manager can select “add customer” option instead of “search customer”. B. The employee/manager the executes Steps 1c-1f in Manage Customer C. The system then shows the information about the rent including the newly added customer Branch 4 - Step 1d - 1h A. The employee/manager can repeat these steps as many times



	as necessary to add all the items the customer wants to rent Branch 5 - Step 1b - 1i, 2b-2d A. The employee/manager can choose “Cancel” option to go terminate adding new rent
Exception Sequence	Step 1d - The manager/employee does not enter a start and end date and time and selects “Search” option OR - The employee/manager does not enter a whole number for size and whole or decimal number for price per day THEN - The system will show an error message Step 1d - 1h - The manager/employee skippers one or multiple steps from 1d - 1g and selects “add to shopping cart” option without selecting an item - The system shows an error message Step 1o - The manager skips one or multiple steps from 1j - 1n and selects “Confirm” option - The system shows an error message
Note	This use case covers requirements: 1, 6, 7, 8, 9, 10, 11, 14, 16

Note: For the rest of the Use Case Descriptions see Appendix B

- 2.7 Activity Diagram (Manage Renting)

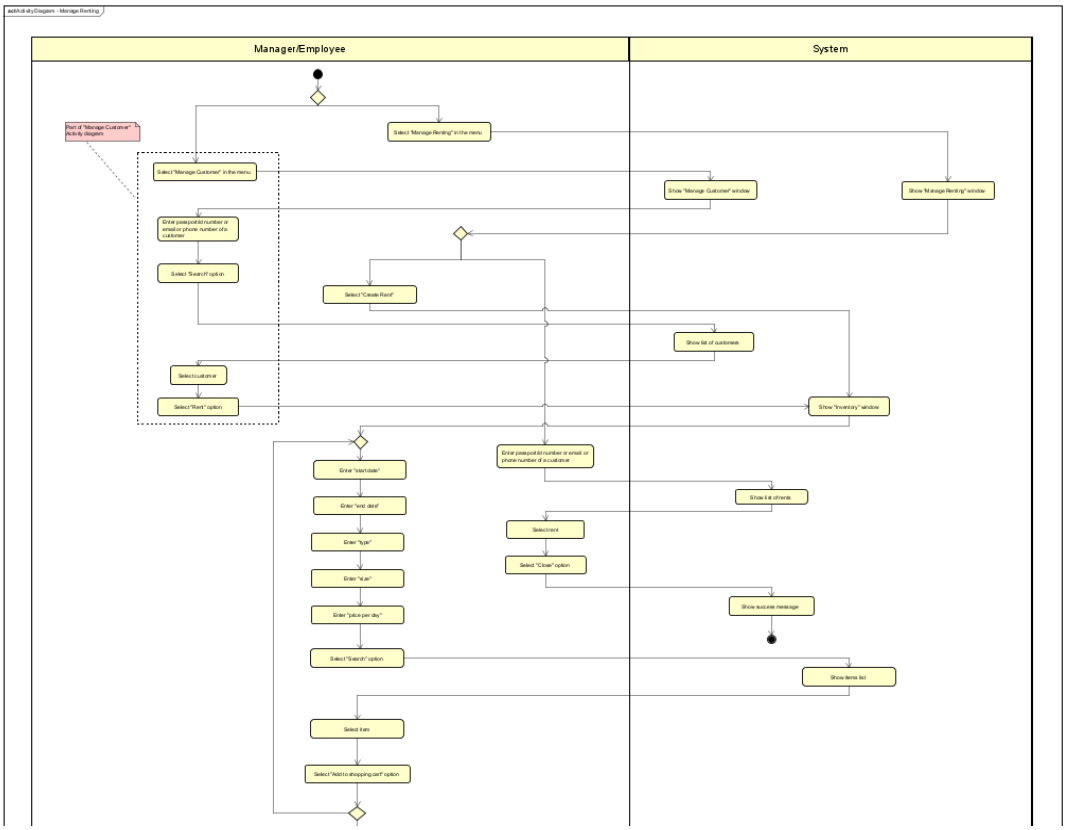


Figure 2 - Activity Diagram (Manage Rent)

Note: For the whole and better quality diagram as well as the rest of the Activity Diagrams see “Appendix G - Activity Diagrams”

- 2.8. System Sequence Diagram (Create Rent)

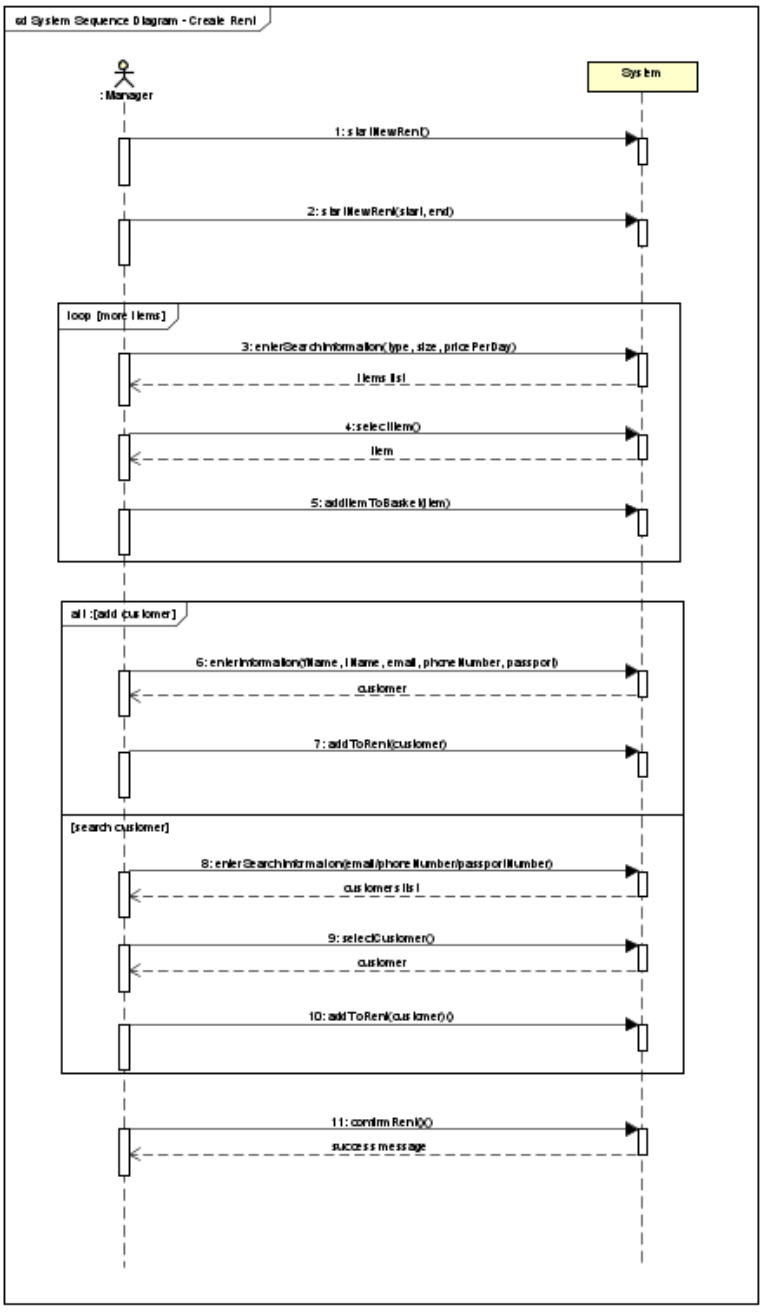


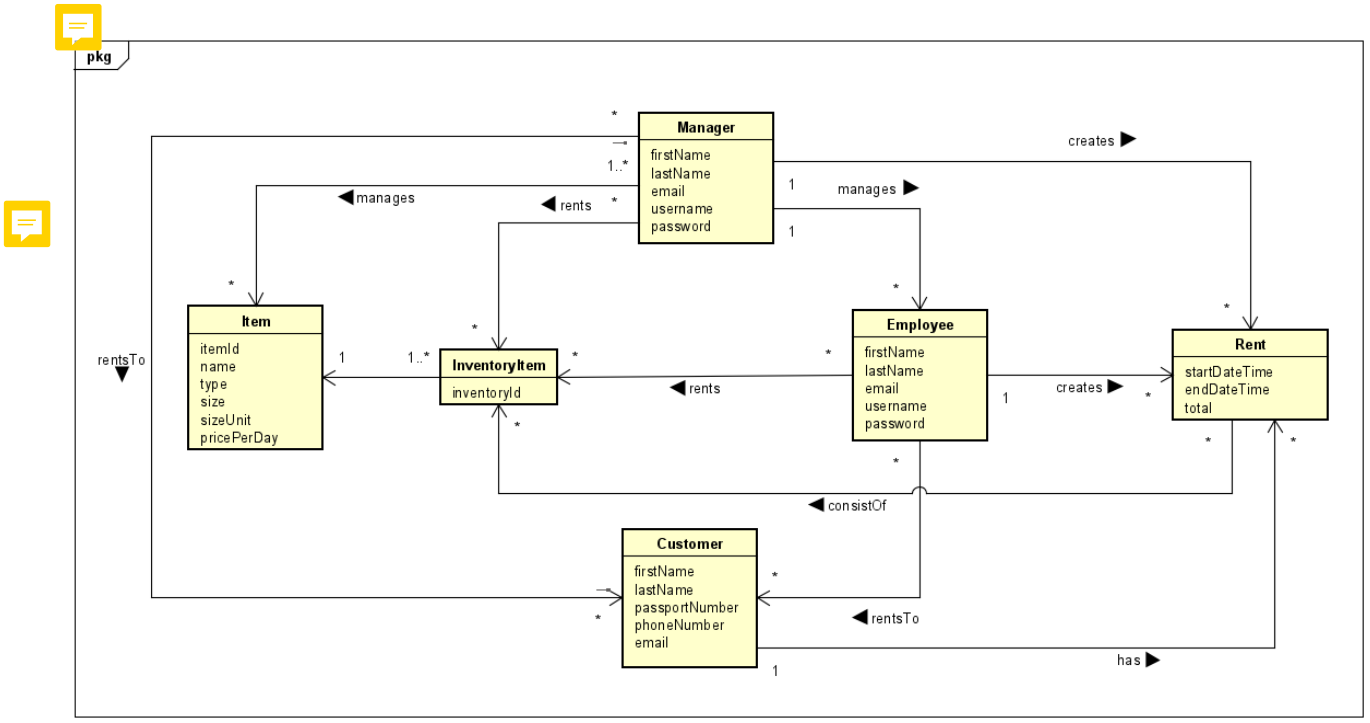
Figure 3 - System Sequence Diagram (Create Rent)

Note: For the rest of the System Sequence Diagrams, see “Appendix I - System Sequence Diagrams”

- 2.9. Domain Model Description

The manager and the employee are responsible for managing everything in the system. The employee manages renting and customers, while the manager manages items in the inventory (manager can add multiple pieces of the same item) and adds and removes employees from the system. A customer can rent items for a period of time via the employee, which has to add a customer to the system, so he can create a new rent for him/her.

- 2.10. Domain Model



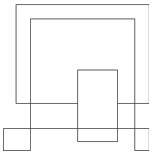


Figure 4 - Domain Model

- 3. Design

After completing the analysis, the next step was design. The following section will present, among others, the technologies used during the work on the project, the selected visual design, the structure of the entire project, design patterns used, and diagrams.

- 3.1 Graphical user interface - GUI

The graphical user interface was created to allow employees easy-to-use interactions with the system.
An initial GUI sketch was made in paint to provide an overall design plan for the application.

- 3.1.1 Add customer

The Figure 5.1 below shows an overview sketch of the - create customer window.

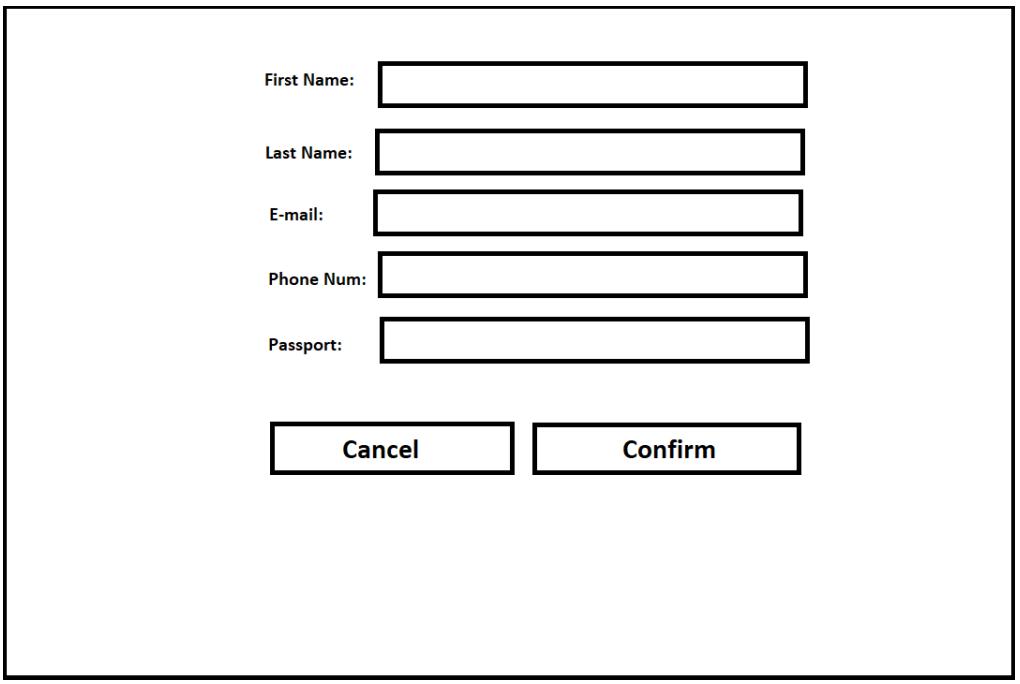


Figure 5.1 - Add customer view - sketch

The picture below shows the final view of the new customer creation window. There are 5 text fields needed to fill in the information about the new customer and the 'confirm' and 'cancel' buttons to confirm the data and then create a new account or cancel the entire action.

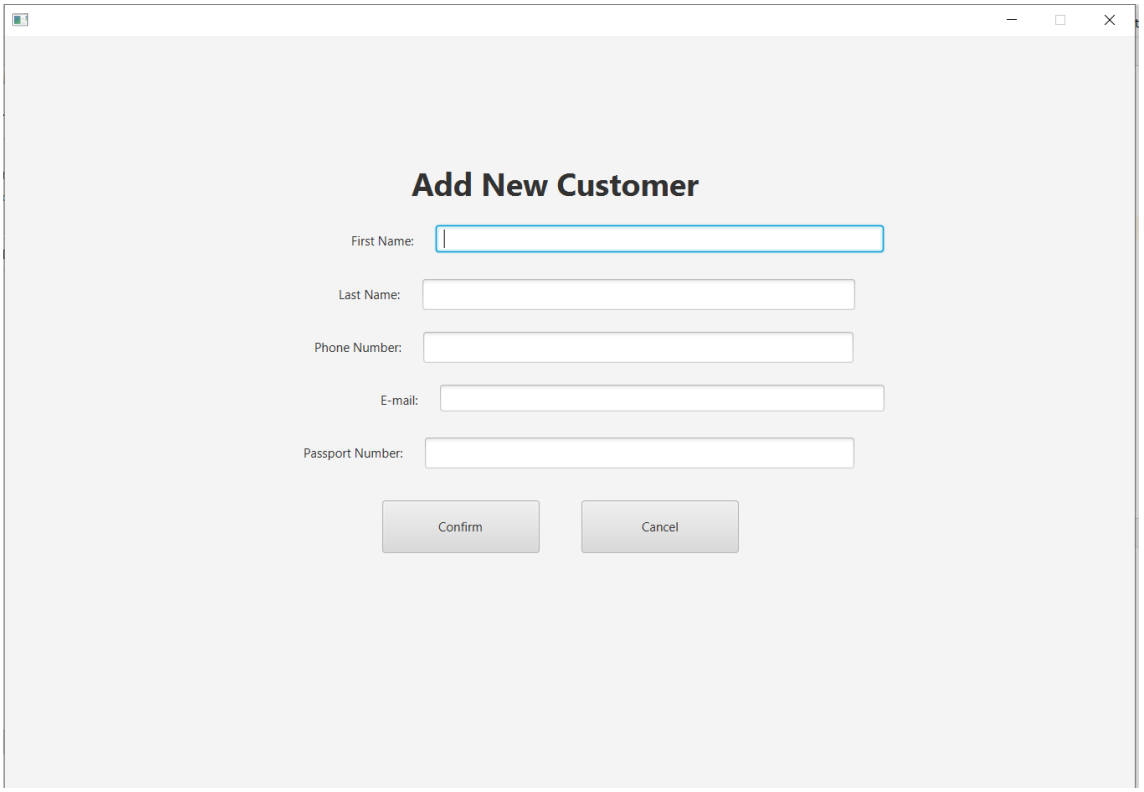


Figure 5.2 - Add new customer view

- 3.1.2 Add new employee

Adding a new employee is very similar to adding a new customer shown above. There are also 5 text fields to fill and 2 buttons to confirm or cancel an action.

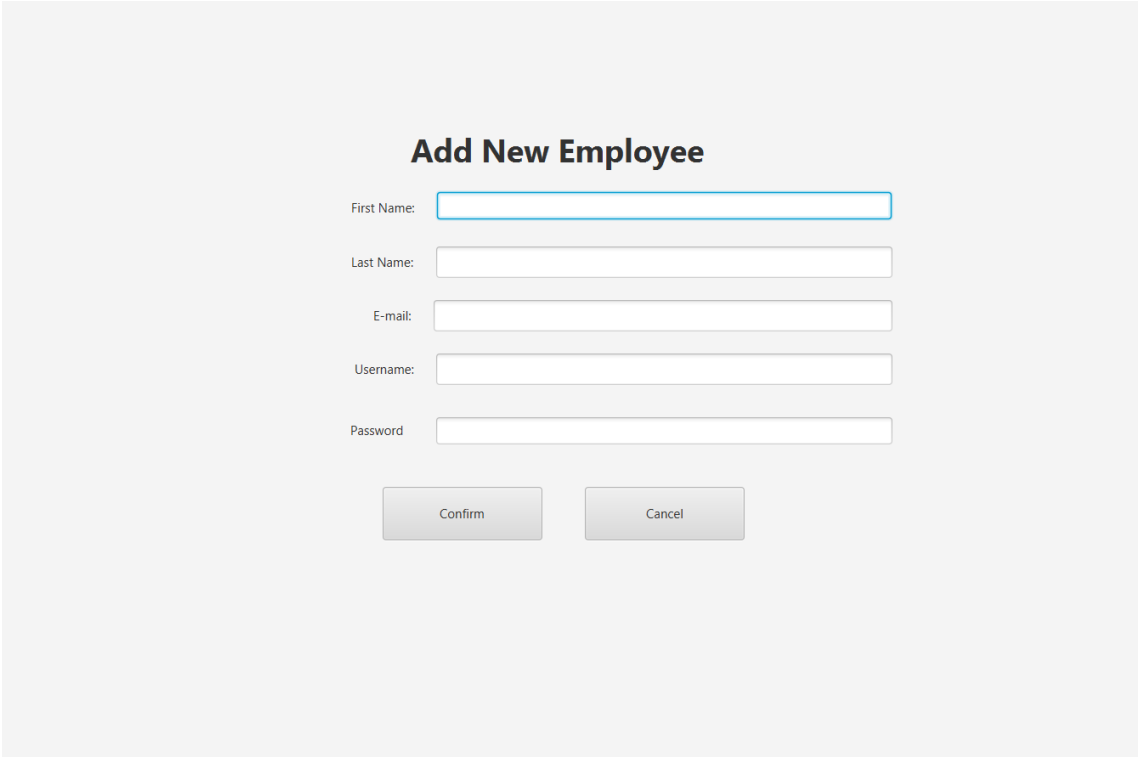


Figure 5.3 Add new employee view

- 3.1.3 Add new item

To add a new item to the rental shop, you need to fill in individual fields, such as: name of the item, daily price, size and quantity. You must also select an item type by tapping in the drop box to see the possible types of items that are in the rental shop.

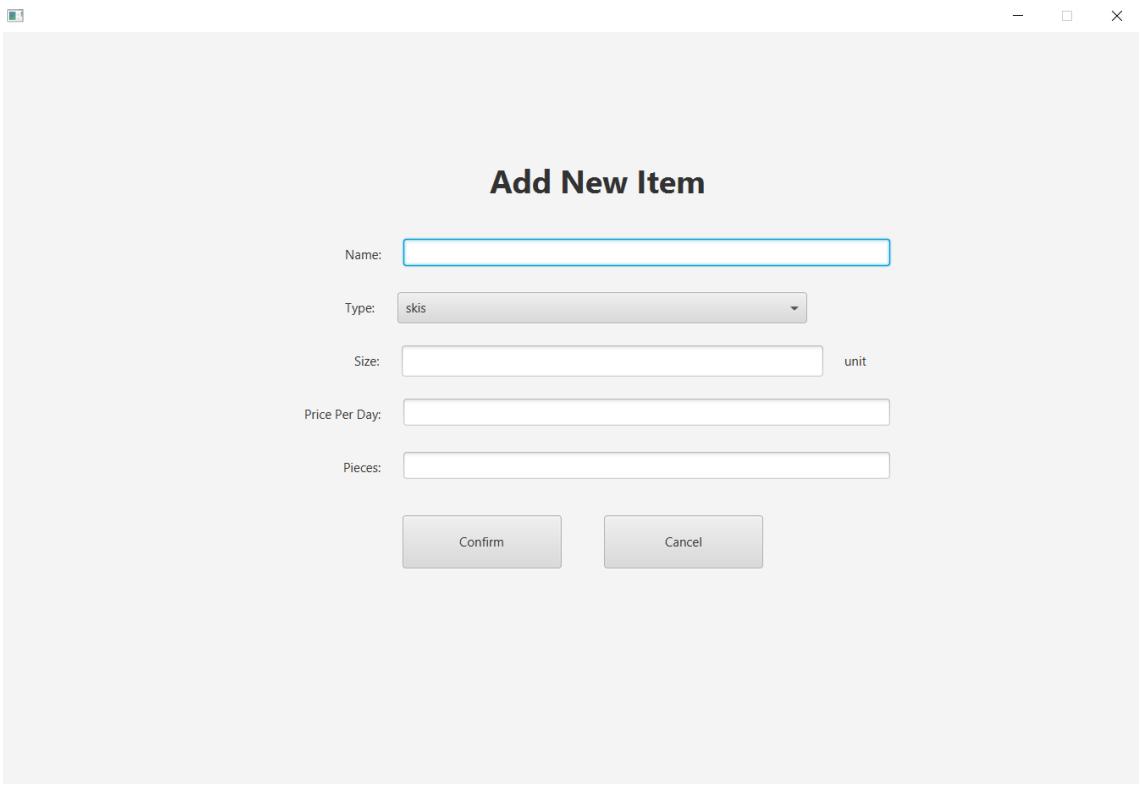


Figure 5.4 Add new item view

- 3.1.4 Filter customers

Searching for a given client from the list of all clients can be very complicated if a lot of them have already been created. Therefore, you can search for a customer using his email, document number, or telephone number. **Figure 5.5** shows what the client list view looks like. Above the table divided into columns showing customer data such as: First name, Last name, passport number, phone and email, there is a text field that allows us to search for a given customer by entering his data and the 'search' button to perform the search.

In the lower corner there are buttons that allow you to start the rental of the selected customer, remove him from the list or add a new one.

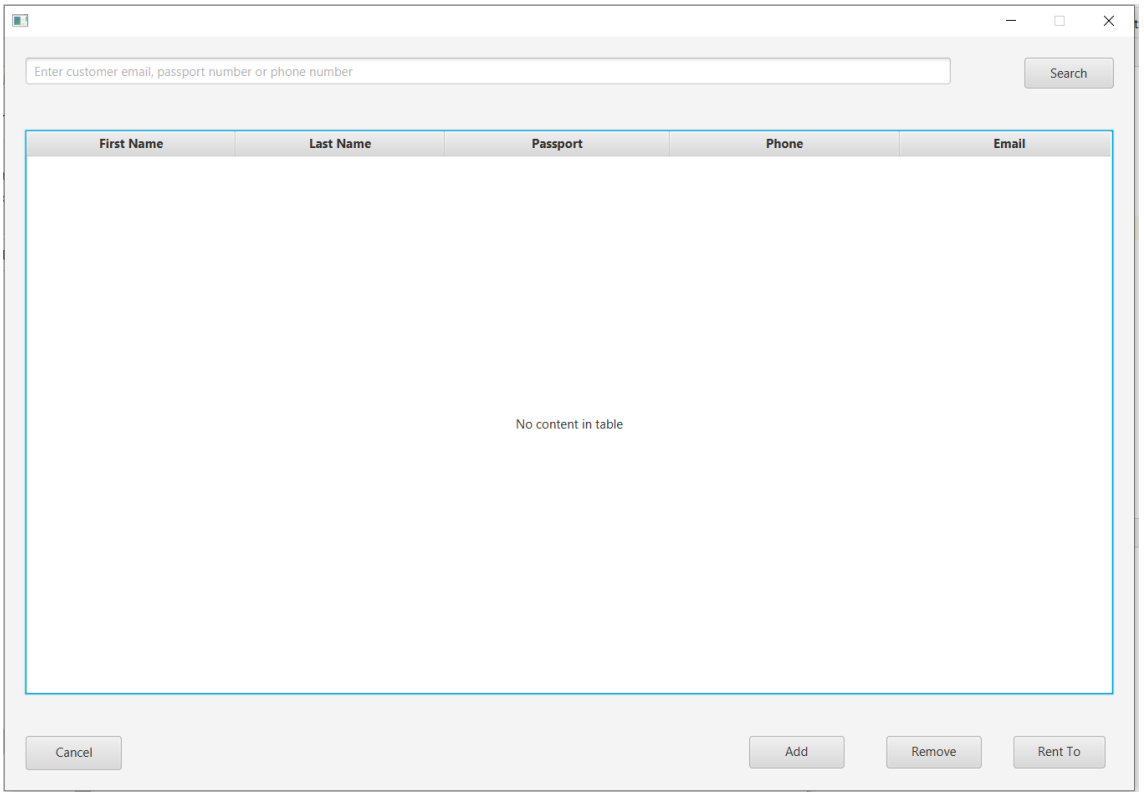


Figure 5.5 Filter customers view

- 3.1.5 Filter items

To assign items to a selected customer, select the item type using the drop box. You can search for the exact item by entering its data such as size or price. Using buttons you can add items to the card.

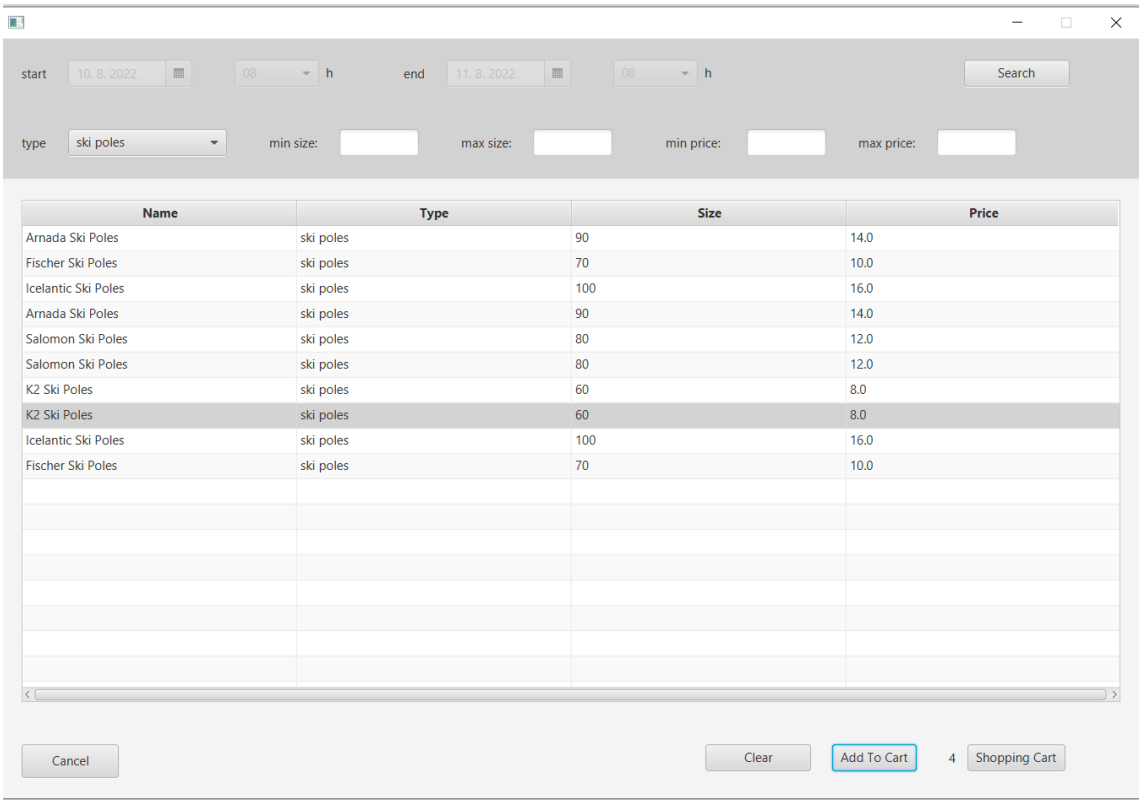


Figure 5.6 Filter items view

- 3.1.6 Rent view

Figure 5.7 shows the appearance of the equipment rental by a given customer. All information about the customer, as well as each item borrowed by him, as well as the rental itself, for example the beginning and the end of the rental, are visible.

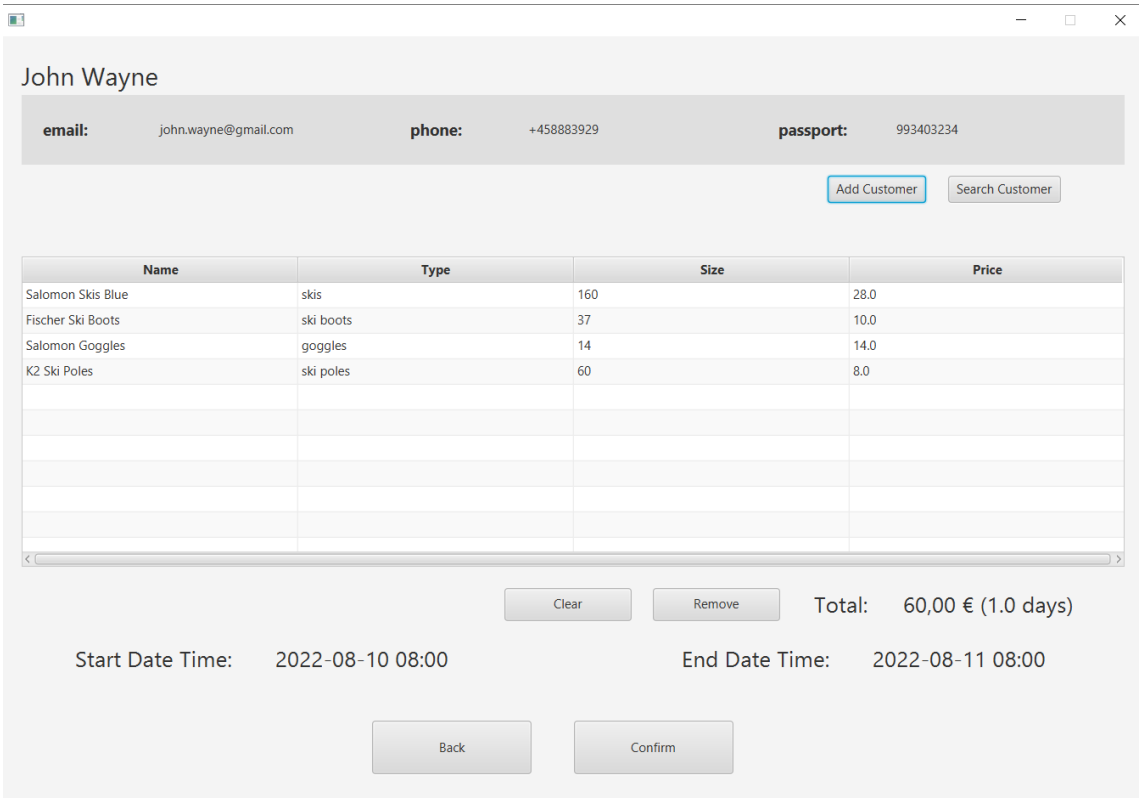
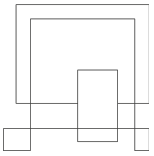


Figure 5.7 Rent view

- 3.2. Client-Server Connection

To create the connection between the server and the client part of the system, a Java API called Remote Method Invocation (RMI) was used. Java RMI performs remote method invocation with support for serialized Java classes and distributed garbage-collection.

The decision to choose RMI over Socket (another method of communication between java applications) was made for several reasons. The most important reason was the simplicity of implementation of RMI over Sockets. Even though Sockets offer more power to the developers, for this project it was not found necessary and the conclusion was made that RMI would be better.



- **3.3. Class diagram**

The class diagram shows the structure of the entire project, the main two packages of which are the server and the client. The entire class diagram can be seen in the Appendix P.

- **3.4 Design patterns**
- **3.4.1 MVVM**

To keep the proper and clean structure of the project MVVM pattern was used. (Model-View-ViewModel). Each view has its own controller, view model and fxml file. To separate functionalities. The model classes are located in the model folder.

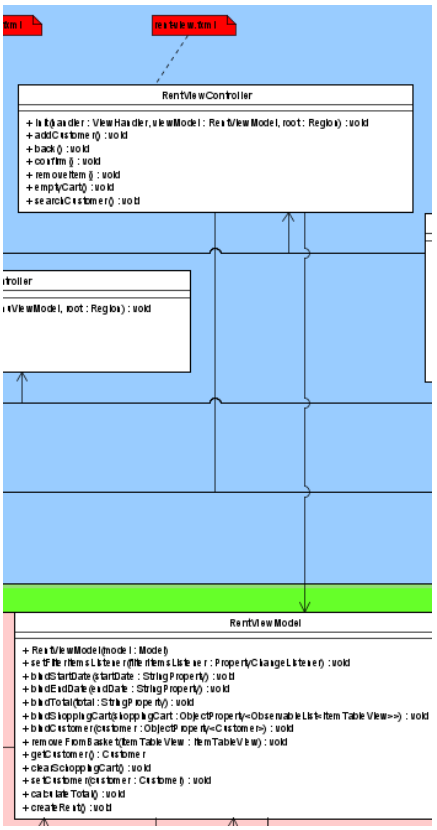
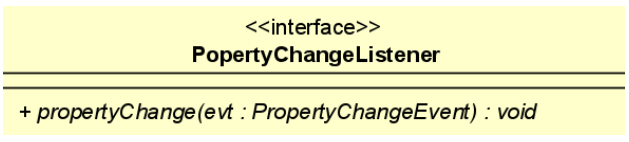


Figure 6 MVVM - UML

- **3.4.2 Observer pattern**

Due to the various changes constantly introduced by the employee, the observer pattern had to be used to inform the individual classes about the changes that affect them.



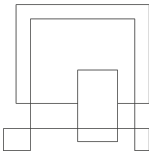


Figure 7 Observer pattern - UML

- 3.4.3 Singleton pattern

Singleton pattern was used to ensure that only one object of a given class can be created.

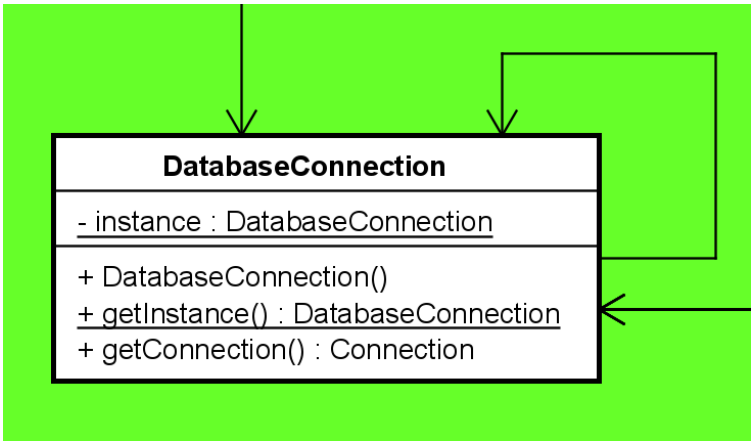
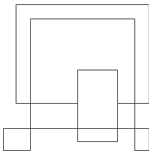


Figure 8 Singleton pattern - UML



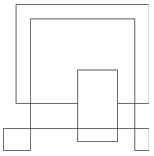
- 4. Implementation

- 4.1. FXML Files

For the user interface, we used FXML files, which is JavaFX's own interpretation of the XML notation, and these files are used to create a graphical user interface. It contains a tree-like structure of components. In the figure below you can see a snippet of the Rent View file, that contains a TableView for items in the shopping cart and TableColumns that specify each column of the table, each of these components have fx:id, through which they are located by the Controller classes.

```
<TableView fx:id="items" prefHeight="296.0" prefWidth="1040.0">
   <columns>
      <TableColumn fx:id="nameColumn" prefWidth="260.0" text="Name"/>
      <TableColumn fx:id="typeColumn" prefWidth="260.0" text="Type"/>
      <TableColumn fx:id="sizeColumn" prefWidth="260.0" text="Size"/>
      <TableColumn fx:id="priceColumn" prefWidth="260.0" text="Price"/>
   </columns>
   <VBox.margin>
      <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
   </VBox.margin>
</TableView>
```

Figure 9 - rent-view (FXML)



-

-

- 4.2. Controller

Controllers or Controller classes are used to connect Java code with the FXML files. It contains Java Classes that control specific FXML components and methods using their fx:id. In the figure below, you can see the Java Class interpretation of the FXML components and methods shown in figure 9. In the login method.

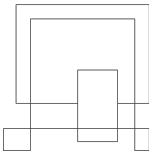
```
@FXML
private TableView<ItemTableView> items;

@FXML
private TableColumn<ItemTableView, String> nameColumn, typeColumn;

@FXML
private TableColumn<ItemTableView, Integer> sizeColumn;

@FXML
private TableColumn<ItemTableView, Double> priceColumn;

@FXML
private Label customerName, total, startDate, endDate, email, phone, passport;
```

Figure 10 - RentViewController (Java)

-

- 4.3. Checking User Input

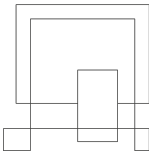
In the figure below, you can see how user input checking was handled in the code. The user input was handled in the Controller class. First it checks whether a customer has



been selected (the user has an option to search for a customer or add a completely new one). Then it checks whether any items were selected (the user can search for desired items and add them to cart) and in the end it checks whether start and end date were selected. If all of these tests pass, a new rent is added to the database.

```
@FXML
void confirm() {

    if(customer.getValue() == null){
        new ErrorAlert("No customer selected");
        return;
    }

    if(startDate.getText() == null || startDate.getText().equals("")
            || startDate.getText() == null || endDate.getText().equals("")){
        new ErrorAlert("Missing end or start date");
        return;
    }

    try {
        viewModel.createRent();
        new SuccessAlert("New rent successfully created");
        handler.openView(ViewHandler.FILTER_ITEMS_VIEW);
    }catch (NoItemsSelectedException e){
        new ErrorAlert("No items were selected");
    } catch (SQLException | RemoteException | NotBoundException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 11 - RentViewController (Java)



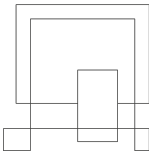
- ### 4.4. ViewHandler and ViewFactory

ViewHandler (Figure 13) is used to switch between views. Each controller has access to this object. ViewFactory (Figure 12) loads the controllers and FXML files when they need to be accessed and stores the controllers. ViewFactory is accessible to the ViewHandler, which allows it to load and switch between views using openView method.

```
public Region loadRentView(){
    if(rentViewController == null){
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(getClass().getResource(RENT_VIEW));
        try {
            Region root = loader.load();
            rentViewController = loader.getController();
            rentViewController.init(viewHandler, viewModelFactory.getRentViewModel(), root);
        } catch (IOException e) {
            throw new IOError(e);
        }
    }
    filterCustomersViewController.getViewModel().setPrevView(null);
    addCustomerViewController.getViewModel().setRentViewListener(viewModelFactory.getRentViewModel());
    return rentViewController.getRoot();
}
```

-

Figure 12 - Load Method - ViewFactory (Java)

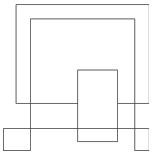


```
public ViewHandler(ViewModelFactory viewModelFactory) {
    this.viewModelFactory = viewModelFactory;
    this.viewFactory = new ViewFactory( viewHandler: this, viewModelFactory);
    this.scene = new Scene(new Region());
}


public void start(Stage stage) {
    this.stage = stage;
    openView(LOGIN_VIEW);
}


public void openView(String viewId) {

    Region root;

    switch (viewId) {
        case LOGIN_VIEW:
            root = viewFactory.loadLoginView();
            break;
        case ADD_EMPLOYEE_VIEW:
            root = viewFactory.loadAddEmployeeView();
            break;
        case MANAGE_ITEM_VIEW:
            root = viewFactory.loadManageItemView();
            break;
```

Figure 13 - ViewHandler (Java)

- 4.5. ViewModel

ViewModel classes are used to connect the user interface (Controllers and FXML files) with the business model (Model). They are used as a middle man between those two parts and for comunication between specific views. In the Figure below you can see two methods, first of them “bindCustomer” is used to bind the list of customers in the Controller to the list of customers in the ViewModel. This is used for comunication between the ViewModel and its Controller. Whenever the shopping cart changes in the viewModel, it is also changed in the controller. The second method calculates the total the customer has to pay, it is calculated by taking a difference in days and hours



between the start and end date and mutliplied by price per day of all the items rented together.

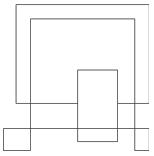
```java
public void bindShoppingCart(ObjectProperty<ObservableList<ItemTableView>> shoppingCart){
    shoppingCart.bind(this.shoppingCart);
}

public void calculateTotal(){
    if(startDate.get() == null || endDate.get() == null)
        return;
    LocalDateTime start = LocalDateTime.parse(startDate.get(), FORMATTER);
    LocalDateTime end = LocalDateTime.parse(endDate.get(), FORMATTER);
    double days = DAYS.between(start.toLocalDate(), end.toLocalDate()) + Math.abs(start.getHour()-end.getHour())/12.0;
    double sumForDay = 0;
    for (ItemTableView i : shoppingCart.get()) {
        sumForDay += i.getItem().getPricePerDay();
    }
    totalAmount = sumForDay * days * 1.0;
    total.set(String.format("%.2f",totalAmount) + " € ("+ days+ " days)");
    System.out.println(sumForDay*days);

}
```

Figure 14 - RentViewModel (Java)

-

- 4.6. ObserverPattern Implementation

In the system, the observer pattern is used to update viewModels with information from different views, for example in the RentViewModel, updates of the shopping cart, customer selection, start date and end date are received and handled. For example when we receive update on the Shopping Cart, we need to calculate the total price again.



```java
@Override
public void propertyChange(PropertyChangeEvent evt) {
    switch (evt.getPropertyName()){
        case UPDATE_SHOPPING_CART:
            ObservableList<ItemTableView> cart = (ObservableList<ItemTableView>) evt.getNewValue();
            shoppingCart.set(cart);
            calculateTotal();
            break;
        case UPDATE_CUSTOMER:
            Customer customer = (Customer) evt.getNewValue();
            this.customer.setValue(customer);
            break;
        case UPDATE_START_DATE:
            String startDate = (String) evt.getNewValue();
            calculateTotal();
            this.startDate.setValue(startDate);
            break;
        case UPDATE_END_DATE:
            String endDate = (String) evt.getNewValue();
            this.endDate.setValue(endDate);
            calculateTotal();
            break;
    }
}
```

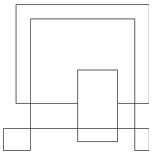
Figure 15 - RentViewModel - Observer Pattern (Java)

- 4.6. Model - Client Side

We have model both on the client and the server side, in the client side, the ModelManager keeps track of the current Staff member logged in and delegates method calls to the RemoteModel.

- 4.7. Connection Client/Server

On the server side, the Server object is exported to the registry using Remote Method Invocation (Figure 16) and then it is looked up and received by the client (Figure 17). RemoteModel is used as an API to access the Server functions.



```java
@Override
public void start(Stage stage) throws IOException {
    stage.setResizable(false);
    RemoteModel server;
    try {
        server = (RemoteModel) java.rmi.Naming.lookup( name: "rmi://localhost:1099/RMIServer");
    } catch (NotBoundException e) {
        throw new RuntimeException(e);
    }
    Model manager = new ModelManager(server);
    System.out.println("Server found");
    ViewModelFactory viewModelFactory = new ViewModelFactory(manager);
    ViewHandler viewHandler = new ViewHandler(viewModelFactory);
    viewHandler.start(stage);
}
```

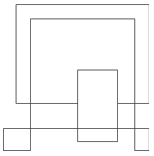
Figure 16 - RemoteModel lookup (Java)

```java
private void startRegistry() throws RemoteException, MalformedURLException {
    Registry registry = LocateRegistry.createRegistry(PORT);
    try {
        UnicastRemoteObject.exportObject( obj: this,  port: 1099);
    } catch (RemoteException e) {
        System.out.println("Object already exported");
    }
    Naming.rebind( name: "RMIServer",  obj: this);
    System.out.println("Starting server...");
}
```

Figure 17 - RemoteModel export (Java)

- 4.8. Server

The Server object implements the RemoteModel interface which is exported to the registry and received by the client. It delegates the functions in its API to the ModelManager. In the figure below you can see the login method of the Server, that checks whether a user with the provided username and password is in the database.



```java
@Override
public Staff logIn(String username, String password) throws RemoteException, NotBoundException, SQLException, IllegalStateException {
    Staff staff = model.logIn(username, password);
    if (staff != null) {
        System.out.println("User " + username + " successfully logged in");
        return staff;
    } else {
        System.out.println("User " + username + " was refused access");
        throw new IllegalStateException("Username or pasword is wrong");
    }
}
```

Figure 18 - Server Login Method (Java)

- 4.9. Model - Server Side

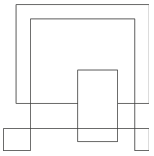
ModelManager, which implements the Model interface, contains all the database access objects (DAOs) that communicate with the Database.

```java
private RentDAO rentDAO;

public ModelManager() {
    staffDAO = new StaffImplementation();
    itemDAO = new ItemImplementation();
    typeDAO = new TypeImplementation();
    customerDao = new CustomerImplementation();
    rentDAO = new RentImplementation();
}

@Override
public Staff logIn(String username, String password) throws SQLException, IllegalStateException {
    return staffDAO.logIn(username, password);
}
```

Figure 19 - ModelManager - Server side (Java)

- 4.10. Database Access Objects

Database access objects (DAOs) are used for communication between the application and the data stored in the database. In the figure below, you can see the



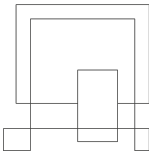
implementation of the insert method for a new rent. Connection object (which is used to create statements for the database and execute them) is received from the DatabaseConnection object (Figure X+11). A PreparedStatement (Prepared Statements are used to prevent SQL Injection) is created and updated with values from the Rent object. The statements is executed and the database is updated.

```java
@Override
public void insert(Rent rent) throws SQLException {

    Connection connection = databaseConnection.getConnection();
    System.out.println(rent.getStart().format(FORMATTER));
    System.out.println(rent.getEnd().format(FORMATTER));

    String sql = "INSERT INTO "  + SCHEMA + "." + RENT_TABLE + "(" + START_DATE_TIME + "," + END_DATE_TIME + "," +
            USERNAME + "," + PASSPORT_NUMBER + ", " + TOTAL + ") VALUES (?, ?, ?, ?, ?)";
    PreparedStatement statement = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
    statement.setTimestamp( parameterIndex: 1,  Timestamp.valueOf(rent.getStart()));
    statement.setTimestamp( parameterIndex: 2, Timestamp.valueOf(rent.getEnd()));
    statement.setString( parameterIndex: 3, rent.getStaff().getUsername());
    statement.setString( parameterIndex: 4, rent.getCustomer().getPassportNumber());
    statement.setDouble( parameterIndex: 5, rent.getTotal());
    statement.executeUpdate();

    ResultSet set = statement.getGeneratedKeys();
    if(set.next()){
        int id = set.getInt( columnLabel: "rentId");
        System.out.println(id);
        for (Item i : rent.getItems()) {
            System.out.println("item");
            statement = connection.prepareStatement( sql: "INSERT INTO "+SCHEMA+"."+RENTED_ITEM_TABLE+"(rentId, inventoryItemId) VALUES (?,?)");
            statement.setInt( parameterIndex: 1, id);
            statement.setInt( parameterIndex: 2, i.getItemId());
            statement.executeUpdate();
        }
    }
}
```

Figure 20 - RentImplementation - DAO (Java)

- 4.11. DatabaseConnection - Singleton

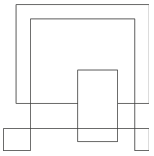
DatabaseConnection object implements Singleton Pattern, which ensures there is only one instance of the class. In the constructor, it initializes the postgresql Driver, and the



in the getConnection method, it uses it to get the Database connection. getInstance method is used to initialize the object for the first time and store the static instance.

```java
private static final DatabaseConnection instance = new DatabaseConnection();


private DatabaseConnection(){
    try {
        Class.forName( className: "org.postgresql.Driver");
    }
    catch (java.lang.ClassNotFoundException e) {
        System.out.println(e.getMessage());
    }
}


public static synchronized DatabaseConnection getInstance() { return instance; }


public Connection getConnection() throws SQLException {
    return DriverManager.getConnection(URL, USERNAME, PASSWORD);
}
```

Figure 21 - DatabaseConnection (Java)



- **4.12. Database - SQL**

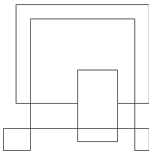
The Database is programmed using PostgreSQL language. In the figure below, you can see the table for the Rent. It includes foreign keys to reference the customer (passport) and the staff member that created the rent (username)

```
CREATE TABLE Rent
(
    rentId         serial NOT null
        CONSTRAINT rent_pk
            PRIMARY KEY,
    startDateTime  timestamp,
    endDateTime    timestamp,
    username         VARCHAR
        CONSTRAINT rent_staff_fk
            REFERENCES Staff,
    passportNumber VARCHAR
        CONSTRAINT rent_customer_fk
            REFERENCES Customer,
    total double precision
);
```

Figure 22 - Rent Table (SQL)

- **4.12. Database - Filter Items SQL**

In the figure below you can see the SQL for filtering available items. It checks for all the items that are not currently in any rent, that have a certain size and price range and type, and all the items that are currently in a rent, but will be available in the period the customer needs them.



```sql
SELECT inventoryId, name, I.type, size, pricePerDay, T.sizeUnit
FROM InventoryItem II
INNER JOIN Item I  1..n<->1: on II.itemId = I.itemId
INNER JOIN Type T  1..n<->1: on I.type = T.type
WHERE inventoryId NOT IN (SELECT inventoryId FROM RentedItem) and I.size >= 0::int
  and I.size <= 1_000_000::int and I.pricePerDay >= 0::double precision and
      I.pricePerDay <= 1_000_000::double precision
UNION
SELECT inventoryId, name, I.type, size, pricePerDay, T.sizeUnit
FROM InventoryItem II
INNER JOIN Item I  1..n<->1: on II.itemId = I.itemId
INNER JOIN Type T  1..n<->1: on T.type = I.type
INNER JOIN RentedItem RI  1<->1..n: on II.inventoryId = RI.inventoryItemId
INNER JOIN Rent R  1..n<->1: on RI.rentId = R.rentId
WHERE (R.startDateTime > '02-02-2022 8:30'::timestamp or R.endDateTime < '01-02-2022 8:30'::timestamp)
  and I.size > 120::int and I.size < 150::int and I.pricePerDay > 20::double precision and
      I.pricePerDay < 40::double precision;
```

Figure 23 - Filter Items (SQL)

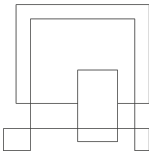
- 4.13. Rest of the Implementation

If you want to see the whole code you can see it in “ Appendix A - Java Code” and SQL in “Appendix F - PostgreSQL” or follow the GitHub link:
https://github.com/YoUnGi102/WinterRentalStore
If you want to see the user guide for the application check ‘’Appendix E’’.

Note:
To try out the system, use:

Manager:
username: admin
password: password

Employee:
username: morgan.jones



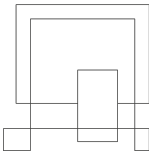
password: password

- 5. Test

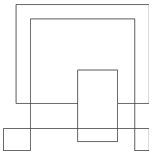
To test the system, Black Box method called Test Cases was used, the test cases contain the test id, the requirement they test, preconditions, steps in order to execute the test, test data, the expected result of the test and whether the test Passed or Failed

- 5.1. Create Rent - Test Cases

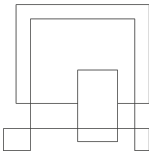
9	Create Rent	Creating rent without selecting customer or items	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Select "Shopping cart" 4. Select "Confirm"	-	An error message is shown saying that information is missing and the rent creation will not be finalized	Pass
10	Create Rent	The start and end datetime cannot be changed after one item is added to the shopping cart	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information as shown in the Test Data 4. Select "Search" 5. Select the first item in the list 6. Select "Add to basket"	start date: 2.8.2022 8:30 end date: 3.8.2022 8:30	The field for entering start and end date and time is disabled	Pass
11	Create Rent	Add an item to the shopping basket	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information as shown in the Test Data 4. Select "Search"	start date: 2.8.2022 8:30 end date: 3.8.2022 8:30 type: "skis"	Number showing how many items are in the shopping basket is increased by one and the item	Pass



				5. Select the first item in the list 6. Select "Add to basket"		information can be seen when selecting "Shopping Basket"	
12	Create Rent	Selecting a customer before starting rent	The user is logged in as a manager/employee	1. Select "Manage Customers" 2. Enter "98989898" into search bar 3. Select "Search" 4. Select first customer in the list 5. Select "Rent" 6. Select "Shopping Cart"	search: "98989898"	Customer information is shown in the view, including first name, last name, email, phone number and passport/id number	Pass
13	Create Rent	Select one item for the rent	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information into the search bar as shown in the Test Data 4. Select the first item in the list 5. Select "Search" 6. Select "Add to Shopping Basket" 7. Select "Shopping Basket"	start date: 2.8.2022 8:30 end date: 3.8.2022 8:30 type: "skis"	Item information is show in the view, including name, type, size and price per day	Pass
14	Create Rent	Select multiple items for the rent	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information into the search bar as shown in the Test Data 4. Select the first item in the list	start date: 2.8.2022 8:30 end date: 3.8.2022 8:30 Item 1: type: "skis"	Information about all three items should be shown in the view, including their types, sizes and prices per day	Pass



				5. Select "Add to Shopping Basket" 6. Repeat steps 3-5 two more times 7. Select "Shopping Basket"	Item 2: type: "ski boots" Item 3: type: "ski poles"		
15	Create Rent	Remove an item from the shopping cart	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information into the search bar as shown in the Test Data 4. Select the first item in the list 5. Select "Add to Shopping Basket" 7. Select "Shopping Basket" 8. Select "Remove" next to the item	start date: 2.8.2022 8:30 end date: 3.8.2022 8:30 type: "skis"	The item will be removed from the shopping basket	Pass
16	Create Rent	Add new customer after selecting items	The user is logged in as a manager/employee	1. Select "Manage Rent" 2. Select "Create Rent" 3. Enter information into the search bar as shown in the Test Data 4. Select the first item in the list 5. Select "Add to Shopping Basket" 6. Select "Shopping Basket" 7. Select "Add New Customer" 8. Enter customer information as	Item: start date: 2.8.2022 8:30 end date: 3.8.2022 8:30 type: "skis" Customer: first name: "John" last name: "Wills" email: "j.wills@g	The customer should be added to the system, and information about the customer will be shown in the view including first name, last name, phone number, email and passport number	Pass



				shown in the Test Data 9. Select "Confirm"	mail.com" phone: "+941223455" passport/id num: "88993029"		
17	Create Rent	Create rent, after adding an item and a customer	The user is logged in as a manager/employee	1. Execute steps in Test 18 2. Select "Confirm" option to confirm creating new rent	Test data from test 16	Rent is added to the system and a success message is shown	Pass

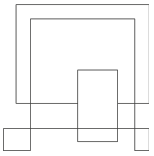
Note: For the rest of the Test Cases, see Appendix D - Test Cases

-

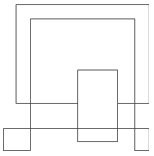
- 6. Results and Discussion

The main aspect of a successful project consists in the customer receiving the product with all the requirements fulfilled. Due to all the factors and impediments that stood in the making of the project not all the features have been implemented. Therefore, this will be presented by having a list with all of them and their status.

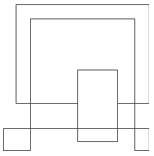
1. As a manager I want to have a default account which I can access and log into using a username and a password, in order to be able to use the system.	Implemented



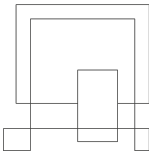
2. As a manager I want to be able to add a new employee account specifying the first and last name, email, username, and the password so that a new employee can use the system.	Implemented
3. As an employee I want to be able to log in using a username and a password, in order to be able to use the system.	Implemented
4. As a manager I want to be able to add new equipment to the store specifying the id of item, name (generally brand and model), type (skis, ski poles, ski boots, snowboard, snowboard boots, goggles, helmet or a custom type), size (skis, poles, snowboard - length in cm, snowboard/ski boots - shoe size in EU standard, goggles, helmet - age), price for a day of renting and how many pieces of this equipment (with each of the items having a generated unique inventory) I purchased for the shop and added, so that a customer can rent it.	Implemented
5. As an employee I want to be able to add a new customer specifying his/her name, surname, phone number, email and passport/id number so that he/she will be able to rent equipment.	Implemented
6. As an employee I want to be able to see and filter the list of items in the inventory using start and end date and time of the rent, type of the equipment, size and price per day in order to find a specific item and rent it to the customer.	Implemented



7. As an employee, I want to be able to add and remove equipment from a “shopping basket”, in order to select everything, the customer wants to rent	Implemented
8. As an employee I want to be able to create new renting from the equipment in a shopping basket to a customer for a period of time, in order to rent items to the customer.	Implemented
9. As an employee I want to be able to search for a given customer using their email, phone number or passport/id number from the list of customers so that I can finish his rent.	Implemented
10. As an employee I want to be able to remove one or multiple items from the items a customer rented, so that a customer doesn’t have to return everything at once.	Not Implemented
11. As a manager I want to be able to perform any action the employee can, in order to be able to oversee/replace the employee if necessary	Not Implemented
12. As a manager I want to be able to search for equipment using name and type and remove equipment from the store so that I can get rid of unavailable items.	Not Implemented
13. As a manager I want to be able to remove old employee accounts so that I can remove unused accounts.	Not Implemented



14. As a manager I want to be able to see all of the current and previous rentals of an item, in order to see information about the previous use of the items	Not Implemented
15. As a manager, I want to be able to see all the log in and log out activity of each of the employees and how many hours they worked in order to know how much time they spent using the system.	Not Implemented
16. As an employee, I want to be able to set the description and state (damaged, undamaged) of an item after it was returned, in order to document if there was any damage done on it.	Not Implemented
17. As a manager, I want to be able to search for damaged items, in order to easily remove damaged items from the inventory	Not Implemented
18. As a manager, I want to be able to see the list of all the employees of the store showing their first name, last name, email, and username, in order to know about all of the employees that work for the store	Not Implemented
19. As a manager, I want to be able to change employee account password, in order to create new access phrase for the employee if he/she forgets it	Not Implemented



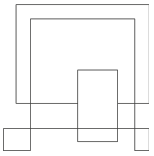
- 7. Conclusions

Winter Rental Store is the result of the analysis, design, implementation, and testing. In the development of the software, it was used Scrum and Unified Process as frameworks to follow.

The analysis was an essential foundation of the project. It established requirements, drew diagrams, and analysed them with the customer. This avoided a lot of possible questions and misunderstandings. The team identified the affected parties and went through many different scenarios the system might encounter. Analysis served as a foundation for designing the inner workings of the system. Every diagram, figure, or table was done according to UML standards.

Designing was a part that needed a lot of attention, it involved building on the diagrams that were developed in the analysis. Here it was thought how the system would react in all kinds of scenarios, considering the positive and the negative outcomes. All related artifacts have been made.

Winter Rental Store can be improved by adding additional features in the implementation. The delivery of the product it was done with a list of Appendices with the necessary documentation.



- 8. Project future

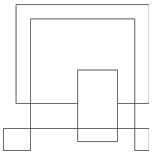
The most important part for the future of the project is to implement the rest of the requirements, after this have been achieved, we can discuss upon the following:

- A notification features that alerts the employee when a new item is added into the system

- An options menu that may include resizing of letter for the visually impaired, a choice of themes that for browsing at different times of day.

- A client for customers that provides them with certain information about their booking

- A log keeping track of employee and customer modifications and interactions with the database.



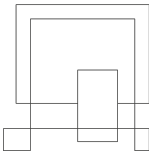
9. Sources of information

Anon., 2020. *Scrum.* [Online]

Available at: https://www.scrum.org/resources/what-is-scrum [Accessed 11 08 2022].

Jira Documentation [Online]

Available at: https://support.atlassian.com/jira-software-cloud/resources/ **[**Accessed 11 08 2022**].**



10. Appendices

Appendix A - Java Code

Appendix B - Use Cases

Appendix C - Astah Files

Appendix D - Test Cases

Appendix E - User Manual

Appendix F - PostgreSQL

Appendix G - Activity Diagrams

Appendix H - Sequence Diagrams

Appendix I - System Sequence Diagrams

Appendix J - Domain Model

Appendix K - EER Diagram

Appendix L - Global Relations Diagram

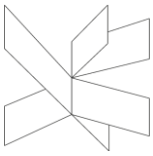
Appendix M - Project Description

Appendix N - Group Contract

Appendix O - UI Design

Appendix P - Class Diagram

Appendix R - Scrum Documentation



Process Report

Winter Rental Store

Tomás Gres, 315185

Gabriel Vlad, 315193

Paul-Mihai Cosarca 308834

Bartosz Zieliński, 308820

Number of characters: 15689

Software Technology Engineering

Semester II

12.05.2022

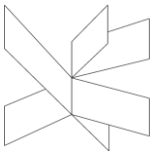
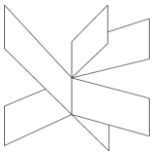


Table of content



1 Introduction

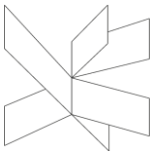
In the Ski rental project, we combined Unified Process(UP), SCRUM and agile development as the methodology and frameworks to develop the system.

Before we started the project we already had experience with SCRUM due to the fact that our group is formed with members from 3rd semester and members from 2nd semester that already worked on a semester project that follows agile methodology, plus the fact we learned about it from the lessons.

After we decided that our Project will be a Ski rental system, we started planning and introducing the Unified process which established the 4 phases of Unified process: Inception, Elaboration, Construction, Transition.

Afterwards we established the project backlog that shows the user stories that construct the system, each sprint had a sprint backlog, for the graphical part representation we had a sprint burndown chart for each sprint and was updated in every SCRUM meeting.

During SEP we were analyzing, designing, documenting, implementing and testing the project that we were working on. Since the re-exam period was shorter, therefore we were working full time on the project focusing on fulfilling all the requirements.
To give an insight into our group during the project period, we are going to describe and detail the parts of the process, showing the team interaction, tools and the methods that we used.



2 Group Description

Our SEP 2 group consists of: Tomás Gres, Gabriel-Eduard Vlad, Paul-Mihai Cosarca and Bartosz Zieliński.

Gabriel-Eduard Vlad:

He is from Romania, where he studied his bachelor of science in Economics afterwards he got his master degree in Information Management with he utilized in 2 of his jobs during education and afterwards, in Denmark he has worked as a UI/UX Designer/Front-end web developer where he has worked on web applications done for the transport system of Denmark across the big belt.

Tomás Gres:

He is from Slovakia, where he previously studied Informational and Networking technologies on Secondary Vocational school of Electrical Engineering. He worked in several teams during his education on various projects during his education, usually single-user systems and games..

Paul-Mihai Cosarca:

He is from Romania, where he worked in sales advancing to a managerial position, after completing an AP degree in Service Hospitality in Denmark due to the pandemic situation he decided to come back to his roots since Paul absolved high school in Mathematics and Informatics.

Bartosz Zieliński:

He is from Poland, where he finished high school in math and informatics. He gained experience in the group by working with a client in several holiday jobs and playing in a sports and music team.

1. Communication		
Low context	1 2 3 4 5 6 7 8 9 10 Team	High context
2. Evaluating		
Direct feedback	1 2 3 4 5 6 7 8 9 10 Team	Indirect feedback
3. Persuading		
Principle first	1 2 3 4 5 6 7 8 9 10 Team	Application first
4. Leading		
Egalitarian	1 2 3 4 5 6 7 8 9 10 Team	Hierarchical
5. Deciding		
Consensual	1 2 3 4 5 6 7 8 9 10 Team	Top-down
6.Trusting		

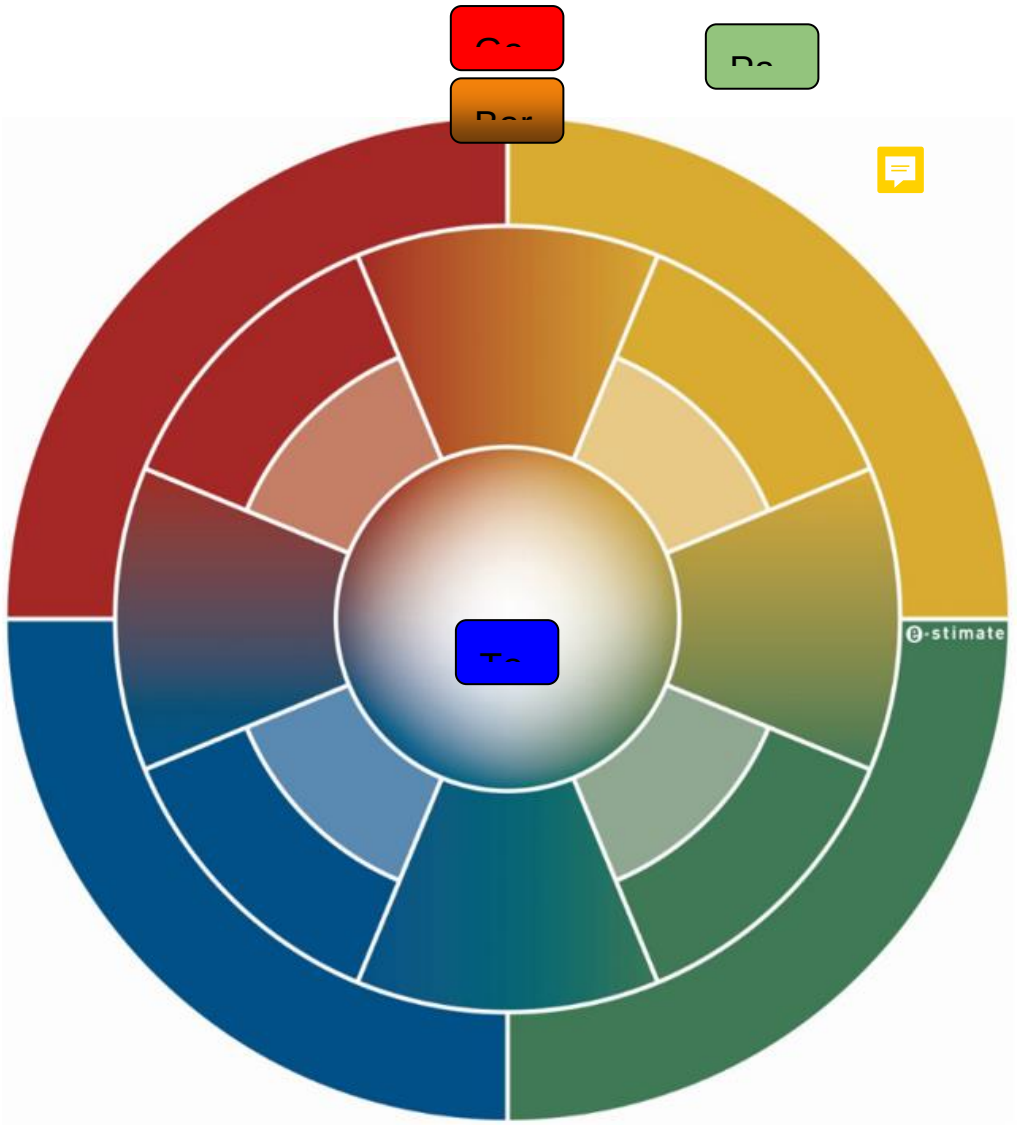
Task-based	1 2 3 4 5 6 7 8 9 10 Team	Relationship-based
7. Disagreeing		
Confrontational	1 2 3 4 5 6 7 8 9 10 Team	Avoidant
8. Scheduling		
Linear time	1 2 3 4 5 6 7 8 9 10 Team	Flexible time

Although we were from different countries, we found a lot of similarities in our cultutres and even in ourselves personally. We all agreed that we are more avoidant in disagreeing and we try not to escalate a conflict into an arguement. We also agreed that even though are countries have more of a hierarchical leadership, we personally think that Egalitarian leadership will be better for us.

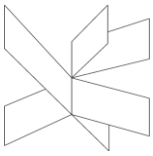
For Comunication, we agreed that our cultures and ourselves fall more into the low-context comunication. Which has proven quite good, as low context communication is better for mostly written comunication, which thanks to working online was our main way of communication.

Decision making was done consentually, even though we all discussed that our countries have a tendency of top-down decision making, which might be good in a big company, but for our team it would not be applicable, as we were all on the same level.

Feedback in our team vs indirect and so it is in our cultures. We tried to sugarcoat any negative feedback we had on each other, in order to avoid conflicts, which had quite a negative impact on the project.



In the presented picture we are displaying our placement on the color wheel based on our results from the personality profile test in SEP. Gabriel has red with yellow making him a strong candidate for the leadership role within SCRUM framework.Because of his focus on being result oriented, direct and ambitious he was chosen as the Product owner. Tomas was chosen as the SCRUM master because of his detail oriented, modest and structured way of being.



3 Project Initiation

After each group member was in the position to be re-examined for the semester project, we had a zoom meeting where we were explained what we need to do further,

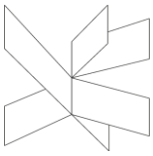
Group 4 was formed during the zoom meeting, we met for an initial meeting afterwards where we drafted our first ideas of a new semester project, this time making sure we are going to fulfill the only criteria Client-Server-Database application.

We considered a game rental store, then we considered a hospital system. In the end we decided to go for a Ski rental project. At this point we started to include the concepts of agile from SWE, such as SCRUM and UP. We didn't have problems understanding the key concepts due to the fact that all members were already in the position that they have worked with SCRUM and up before. We agreed on having 5 workdays per sprint.

We agreed on set deadlines of finishing the planning and project description, then we started having daily sprints forward, with the agreement that if everything is kept on track we should finish the sprints by our estimated timeframe. We discussed setting some deadlines where we would switch phases of UP. We discussed where inception would end, and elaboration would begin. We chose Tomas as the SCRUM master because he showcasted the most fit leadership and organizational skills.

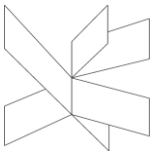
Gabriel was selected as the product owner because of his decisions which best reflect the interests of the team.

For our group the choices made sense also due to the fact that both Gabriel and Tomas were having red traits, which was important for these positions, the other group members filling in the remaining positions of being developers.



4 Project Description

The project description (Appendix M - Project Description) represented the first stage in the making of the project. Having this first part gave us an overview of the project idea. Writing it gave us insight into identifying the main problem and the subproblems. Establishing the purpose and finding out what is out limitations are represented a crucial part before any work could be done. Knowing them helped us to select the working strategies, therefore this resulted in using Scrum and it was decided to divide the tasks in equal periods of time of 165 hours called sprints. Having a good schedule was crucial since the period was shorter in developing the project. Due to vacation and individual plans as a method of communication, we have used Discord and as was mentioned we have pre-planned meetings to ensure high participation of the members.



5 Project Execution

5.1.1 Sprint 1

5.1.2 Sprint planning

11.07 – 9:00

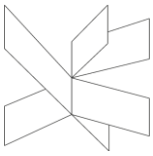
1 hour

Everyone present

For this sprint our goal is to have the basic architecture of the system done including basic GUI, Client-Server Connection and Database connection as well as requirements 1 through 5

Requirements/Tasks for this sprint - 5*(7.5*5-5) = 165h:

Project Report - Basic Architecture (4h)
Process Report - Basic Architecture (3h)
Domain Model (3h)
ER Diagram (5h)
GR Diagram (4h)
Database DDL (5h)
Login View Design (1h)
Main Menu Design (1h)
Class Diagram for basic MVVM (3h)
Implement Basic MVVM Structure (4h)
-- Requirement 1
Class Diagram with Login (1h)
Server Model (4h)
Class Diagram - Server Connection (3h)
Client-Server Connection (10h)
Update Documentation - Server Connection (3h)
Sequence Diagram - Login (1h)
LoginViewController + FXML (4h)
LoginViewModel (1h)
Update Model/ModelManager (1h)
Unit Tests - Login (2h)
Update Documentation with Diagrams/Implementation (3h)
-- Requirement 2
Use Case Description - Manage Employees (7h)
Activity Diagram - Manage Employees (4h)
System Sequence Diagram - Add Employees (3h)
System Sequence Diagram - Login (1h)
Manage Employees - Test Cases (3h)
Class Diagram - Add Employees (2h)
ManageEmployeeViewController (4h)
ManageEmployeeViewModel (1h)
Update Model/ModelManager (1h)
Unit Tests - Add Employees (2h)
Update Documentation with Diagrams/Implementation (3h)
-- Requirement 4
Use Case Description - Manage Items (8h)
Activity Diagram - Manage Items (4h)
System Sequence Diagram - Add Items (3h)
Manage Items - Test Cases (3h)
Sequence Diagram - Add Item (2h)
Class Diagram - Add Item (2h)
ManageItemsViewController (3h)
ManageItemsViewModel (2h)
Unit Tests - Add Items (2h)
Update Documentation with Diagrams/Implementation (3h)
-- Requirement 5
Use Case Description - Manage Customers (8h)
Activity Diagram - Manage Customers (5h)
System Sequence Diagram - Add Customers (3h)
Manage Customers - Test Cases (2h)
Sequence Diagram - Add Customer (2h)
Class Diagram - Add Customer (2h)
ManageCustomerViewController (6h)
ManageCustomerViewModel (2h)
Update Model/ModelManager (1h)
Unit Tests - Add Customer (2h)
Update Documentation with Diagrams/Implementation (3h)



5.2.1 Daily scrum 1

9:00 – 15 min

Everyone present

GR Diagram , ER Diagram, Domain Model , Process Report - Basic Architecture , Client-Server, Connection, Class Diagram - Server Connection , Login View Design – DONE

5.2.2 Daily scrum 2

9:00 – 15 min

Everyone present

Database DDL, Connect Database to JAVA , Class Diagram with Login, Implement Basic MVVM Structure, Class Diagram for basic MVVM , System Sequence Diagram - Login , Update Model/ModelManager + FXML – DONE

5.2.3 Daily scrum 3

9:00 – 15 min

Everyone present

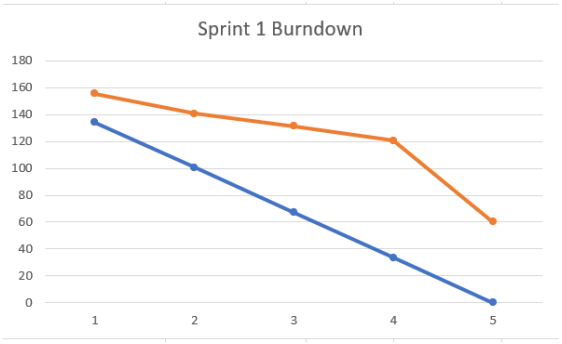
LoginViewModel , LoginViewController, Sequence Diagram - Login , Class Diagram - Add Customer , ManageItemsViewModel , Class Diagram - Add Item - DONE

5.2.4 Daily scrum 4

9:00 – 15 min

Everyone present

Activity Diagram - Manage Items , Use Case Description - Manage Items , Item Class in Model , Customer Class in Model , List classes in Model - DONE



5.2.5 Retrospective

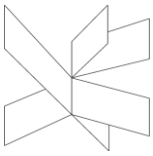
What should we stop doing:
- *leaving tasks that are in progress, uncompleted for multiple days*

What should we continue doing:
- *meet every day for few minutes and say/write down what we did the previous day*
-

What should we start doing:
- *communicate more*
- *pair up, work in teams*
- *write tasks in more details so that everyone can understand them properly*

5.2.6 Review

We managed to connect the client to the server, but we still need connection from the server to the database, as well as to fix some bugs in the Server connection. We did not manage to finish any of the first 5 requirements, but we did some of the use case descriptions and activity diagrams.



6 Personal Reflections

- **Bartosz**

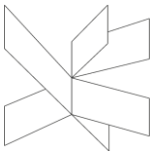
Work on the project began with the formation of a group of people who were partially never in contact with each other. At first, it was hard for me to imagine how we would work together without being able to meet face to face in real terms. As it was a holiday period and each of us was in a different country, it was not possible. The work started quite smoothly, I think that because it was a re-exam, everyone wanted to go through this period of the project efficiently but also with good quality.

It was not that simple, however. Due to vacations and pre-planned activities by everyone, it was not always easy to stick to a rigid work schedule. I had to combine summer work with project work, and an attempt to spend this time with my family after a long break. Our collaboration had its ups and downs, but I got a lot out of it. We were able to rely on each other throughout this period and cooperate despite not being able to discuss things here and now.

Despite being online only, I am surprised how often we managed to meet and how we stuck to scrum work. Working in scrum allowed us to fall into a certain rhythm, which, despite the holidays, kept us working on the project.

Although we come from other countries, the cultural differences were not noticeable in my opinion. I think that as this is our fourth project, or for some the third, we have already learned to cooperate with newly met people so that we are in some way open and understanding, so that just any cultural differences are not noticeable.

Also when it comes to our characters, I can say the same. Everyone in the team found their role, and after a while, everyone understood the role of others as well and accepted it.



Tomas

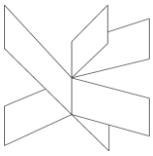
When we started working, everyone was very eager to work, collaborate and give as much as possible to the project. We were meeting online during the entire period of development. In the begining everyone attended all the meetings and it seemed like it would not be hard to finish everything in time.

But as we went further into the summer it began harder for people to work on the tasks, which showcased in our burndown charts and sprint progress. It began to fall down and many times some of the group members did not attend the meetings. It was pretty tough to finish the project even to the point we have right now, not to mention the entire project. Although we did not finish all of the requirements, we managed to do quite a bit, and we were making sure we follow Scrum and Unified process as much as possible.

The team consisted of members from three different countries. We discussed the culture map by Erin Meyer and we found out that we have a lot of similarities for example in hierarchical leading and top-down deciding. Even though we have our differences in culture, it did not show that much. I believe it was mainly because we were working online and our communication was purely proffesional. We did not know each other before.

Paul

At the first glance starting the project was ferly easy, we have attended to all the meetings, and we have been to move into the first parts of the project with ease. We agreed on the topic, wrote the project description, each of us contributed in an equal part. Since the first part was running smoothly, I have not thought that we are going to face any kind of issue, but eventually even though we didn't have major conflicts we have not achieved the same impact on the project. Each of us shared the same goal in finishing the project and correct the issues that we faced in previous projects, due to the project being online we have experienced some miscommunication, I feel like if we



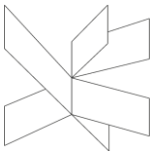
could discuss the way everyone is preferring to work in a group. We would be able to achieve much more, not only in terms of implementation, but we could maximise the learning experience. In the future I'm looking forward to working with people from different cultures, I feel like having an international experience is closer to reality since most of the time in this field we are not going to work with people that have the same cultural basis. Is important to have a nice environment but in some cases I think is even more important to create a workflow in the group, where each member is contributing in the same manner, even tough this can be achieved by being blunt and harsh.

Gabriel

Our group formed on the remnants of the two disbanded groups after the events of SEP-1. Me and Tomas joined the 3 men team Antoniu, Paul and Bartosh

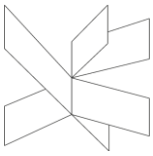
Using our experience from SEP1 we set up the ground rules for a new group contract and we tried to stick to it, due to events we couldn't stick to the contract and it was difficult but we tried our best to have a project don't till summer end, and I honestly think we did our best.

We went for a ski rental project and we did as much of it as possible to the best of our abilities under the circumstances.



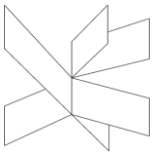
7 Supervision

Because it was the Re-examination we had no direct supervision from the teachers.



8 Conclusions

Overall, development of this project was very difficult due to various reasons. For example working online, not in person was very difficult, as it was easier not to go to a meetings, and the attention during the online meetings was lower than it would have been in person. Another reason was that it was summer, and some of us had work, some of us visited family and that was hard to combine with the work on the project.
If it werent for SCRUM, and we had to use other (not Agile) method, we would have not done even what we have. Working together was also hard because we did not know each other at all, or we just knew one or two people from the group. But in the end we managed to hand in a working piece of software that fullfilled almost half of the requirements.



Appendices

Appendix M - Project Description

Appendix N - Group Contract

Appendix R - Scrum Documentation