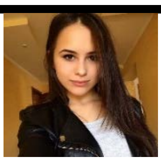
Prest Energy

3 Tier System

Nicoleta Sova, 267069, 

Dementie Bors, 279948,

Justinas Jancys, 280151,

Sabin-Daniel Sirbu, 280143,

Supervisor: Jakob Knop Rasmussen,

Jan Otto Munch Pedersen

[Number of characters]

Software engineering

3rd Semester

20.12.2019

Table of content

Abstract

Nowadays, pollution is one of the biggest concerns of the century, that is why sanitation businesses represent a necessity.
One company that operates in this area is Prest Energy. They required a system to help to manage their work.
The goal of the project was to build a useful tool that could help improve their way of working. The system supports drivers, accountants, cashiers, and administrators in handling their tasks faster than before.

The preferred methods for this project were the use of a three-tier architecture and implementation of a heterogeneous system.
The system uses Java and C# technologies, and for the database part, the SQL query language.

The company provided a list of requirements for the system, but unfortunately, not all of them could be fulfilled, due to the lack of time.

1 Introduction

“Prest Energy” is a company that provides sanitation services, parks arrangement, and 30 other services in the Republic of Moldova. The company has its’ central office in Chisinau. Prest Energy has more than 4000 contracts with homeowners and 500 with companies.

The company has four departments:

Administration – makes contracts with small, medium, and big companies to provide sanitation services;

Cashier – makes contracts with homeowners, and if the administration allows making a contract with a specific small company. Also, the cashier can supply the account for customers paying by cash and record new orders for extra services. For example, someone needs a SKIP container or draining the sewer;

Accounting – gets data from the Excel file that the cashier created and inserts the data in another program called “1C”, then from that program, the accountant can make and print the invoices;

Drivers and auxiliary workers – at the beginning of the day they get a paper with customers who did not pay for services and another one with new contracts, and during the day they are collecting the trash bins, after that, they are starting to deliver the trash bins for new customers.

The problem that the company is facing is that the departments lack communication with each other.

The current system that the company has in place consists of a Microsoft Excel file. The system stores the customers' data. For example, the customers' names, address, and if the customer paid for the services or not. The current set allows the drivers to download the excel document and work without the internet connection.

2 Analysis

The requirements were elaborated based on customer problems and their wishes. The analysis was made based on information that was given by the company and their needs.

2.1 Use case diagram

The system was made that each actor will have different access to the system. The administrator will be able to register, edit or delete an employee or a customer, also the administrator could have access to do the work for cashier. Cashier's access is to manage customers money. Accountant have access to view all customers and manage invoices. Driver has possibility to change the amount of trash that the customer had and count as an extra service. Customer's account has access to view their balance and past services.

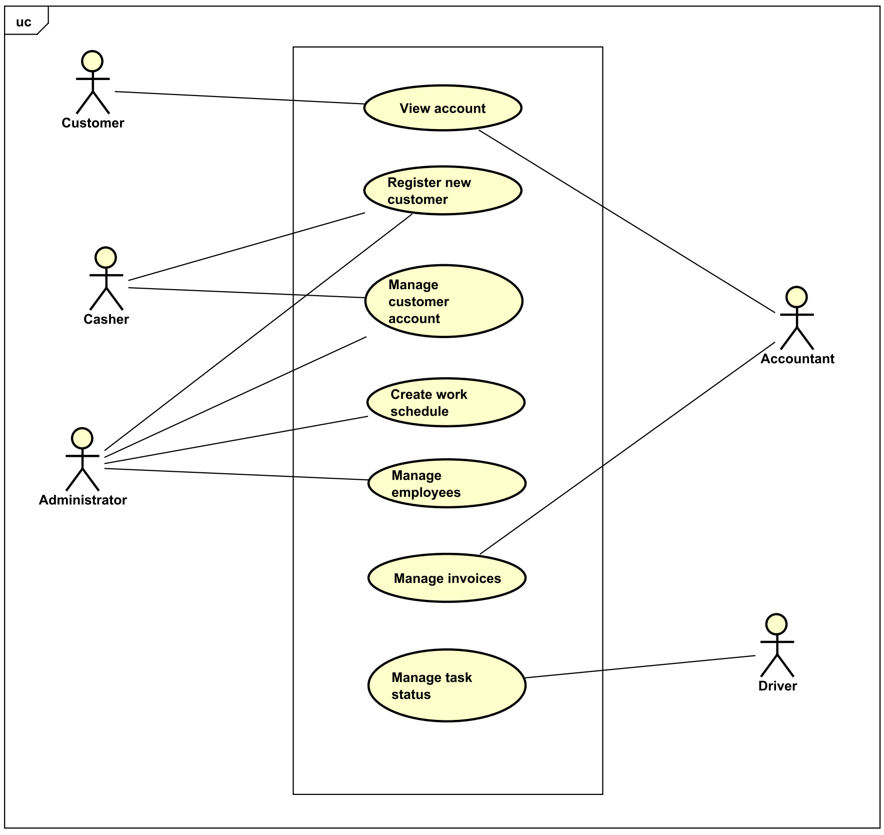


Image 1 Use Case

2.2 Conceptual Domain Model

A system of abstractions that describes selected aspects of a sphere of knowledge, influence or activity (a **domain**).

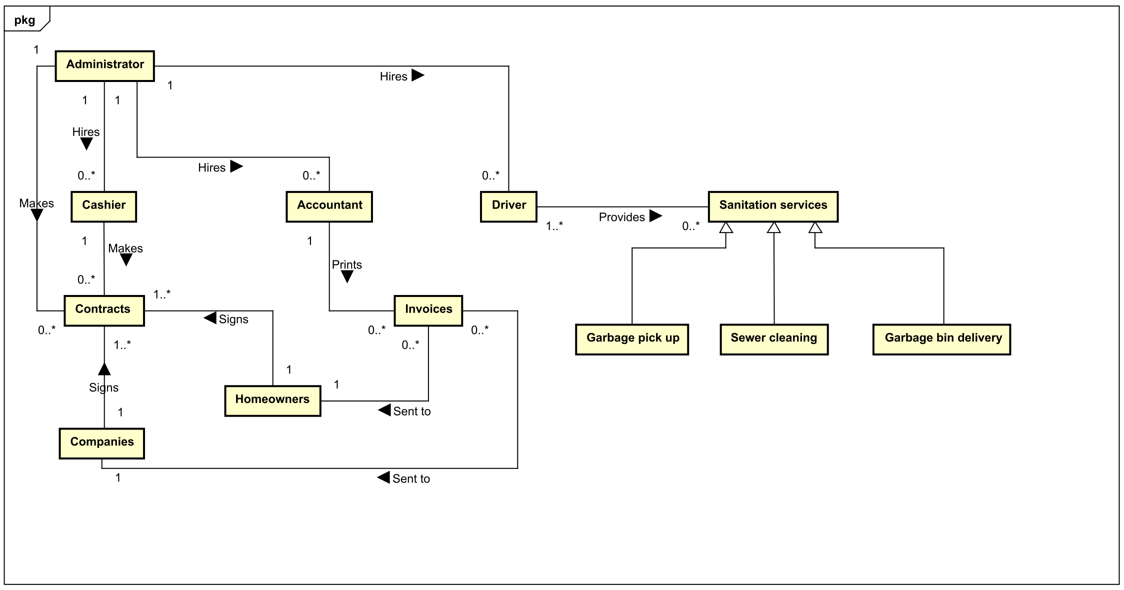


Image 2Domain Model

2.3 Use case description

A written description of how users will perform tasks

Table 1 Create Work Schedule

Item	Value
UseCase	Create work schedule
Summary	The administrator updates the existing work schedule for the drivers.
Actor	Administrator
Precondition	The administrator needs to be logged in.
Postcondition	A work schedule is updated
Base Sequence	1. Press "Manage work schedule". 2. Select the driver. 3. Press "Change"/"Add". 4. Update the data. 5. Save
Branch Sequence	none
Exception Sequence	none
Sub UseCase	none
Note	Each action can be cancelled at any moment and none of the data will be changed.

Table 2 Manage Customer account

Item	Value
UseCase	Manage customer account
Summary	Employees can update customer's profile and add additional money to the account.
Actor	Administrator, Casher, Accountant
Precondition	Need to be logged in
Postcondition	Customers account updated
Base Sequence	1. Open customers account. 2. Click edit button (or double click on the field to edit it). 3. Change the data. 4. Press "Save" Add money to the account: 1. Open customers account. 2. Press "Add money to the account". 3. Enter the amount. 4. Click "Save"
Branch Sequence	none
Exception Sequence	none
Sub UseCase	
Note	Each action can be cancelled at any moment and none of the data will be changed.

Table 3 Add Employees

Item	Value
UseCase	Addemployees
Summary	The administrator is able to add new employees to the system.
Actor	Administrator
Precondition	The administrator needs to be logged in.
Postcondition	An employee is added.
Base Sequence	1. Press "Create new Employee". 2. Fill in the required information about the employee 3. Press "Add new Employee".
Branch Sequence	none
Exception Sequence	none
Sub UseCase	none
Note	Each action can be cancelled at any moment and none of the data will be changed.

Table 4 Create New Customer

Item	Value
UseCase	CreatenewCustomer
Summary	Actors register new customer.
Actor	Casher, Administrator.
Precondition	Actors must be logged in.
Postcondition	New customer is added
Base Sequence	1.The actor must press the ”Create new customer" button. 2.The Actor fills in the registration form. 3.The Actor presses the "Add new Customer" button.
Branch Sequence	none
Exception Sequence	If all fields are not filed in, the user cannot press the button.
Sub UseCase	none
Note	none

Table 5 View Account

Item	Value
UseCase	View account
Summary	A customer can view his account information. The accountant can view customers account
Actor	Customer, Accountant
Precondition	Actors must be logged in.
Postcondition	None
Base Sequence	The customer: 1. Chooses the year The accountant: 1. Chooses the area 2. Searches the customer
Branch Sequence	none
Exception Sequence	The actor can't change anything on its side.
Sub UseCase	none
Note	None

2.4 Security

The security part for a system is the most important item to have. It needs to check each threat and pitfall, because it could affect personal information that is in the database.

In SEP-3 project it is a requirement to use 3 tier architecture, which means that the system needs at least 2 connections using networking, this may cause a few risks that the system could be cracked and need preventative measures. For example:

- If an unauthorized intruder will get access to the database, the intruder will be able to see everything from the database. The goal of the intruder may be information disclosure, tampering or repudiation.
- Another problem with security is information disclosure - the intruder could be a passive attacker using Wireshark and listening everything that is sent between tiers or access the data that is not for the intruder. This breaks confidentiality. The risk is high, because sensitive data could be leaked.
- Denial of Service attacks could cause availability problems for the system. If someone tried to make too many requests in a short time the system would crash.
- The sockets from the system does not require any authentication and if the attacker knows the IP address and the port number of the server socket, the intruder could have access to anyone. That could be damaging for the system.
- HTTPS type of connection will be making the communication more secure and it would be harder for passive attackers to listen what kind of data is being sent. SSL protocol will keep confidentiality

Furthermore, if the system is not safe, then the trespasser could get data easily and with that data he could wreak havoc on the system.

In the following table is implemented Threat and Risk Assessment model for 3rd semester project.

Table 6 Threat & Risk Assessment model

#	Threat	Risk	Risk possibility likelihood	Risk assessment model	
1	Accessing the database	Manipulate data	High	Losses the important data	
2	Impersonate employee by getting employee's password	Spoofing identity	Medium		
3	Being able to listen/access to the data that the intruder should not see between socket connection	Information Disclosure	High	Could happen some changes in the system that nobody will know	
4	Brute-Force attack	Get the password	High	Could have the password for a user	
	Denial of Services	Losing the availability	High	Nobody will be able to access the system	

2.5 Functional Requirements.

The sanitation company requirements were elaborated based on customer problems and their wishes. A list of functional requirements is listed below.

Table 7 Requirements

ID	Priority	Estimated. h.	Item
1	Critical	25	As an administrator I want to be able to give access from the drivers to edit client's data for that specific day, so that the drivers could pick up trash and log it.
2	Low	5	As an administrator I want to upload contracts into the system in a pdf format, so that I can have access to all the customers who requested the company's services. (saved with the name of the customer and the number of contract).
3	High	10	As an administrator I want to be able to register new clients to the system
4	Medium	15	As an administrator I want to be able to register new employees and give different access for them.
5	High	20	As an administrator I want to be able to create a schedule for the drivers, so that the drivers know when to go to which area.
6	Medium	10	As an administrator I want to be able to view all clients that are registered to the system
7	High	13	As a casher I want to be able to register new clients to the system.
8	Medium	5	As a casher I want to be able to add money to their accounts. (no bank transactions) (log every transaction that the casher does)
9	Medium	19	As a casher I want to be able to undo the changes to customers account for 24 hours.
10	High	20	As a driver I want to be able to see which customer need they bins emptied.
11	High	20	Drivers should be able to log how many trash bags did they collect on each customer's account.
12	High	29	As a driver I want to be able to mark which customers bins have been emptied.
13	Low	10	As an accountant I want to print invoices for each client.

14	Low	15	As an accountant I want to be able to view all the clients that are registered to the system
15	Medium	17	As a customer I want to be able to view my balance in the account.
16	Low	20	As a customer I want to be able to download my invoice.
17	Medium	20	As a customer I want to be able to see when the collection day is.

Functional requirement nr 3 is described below.

UseCase diagram: Actor for this action is "Administrator" that register new customers.

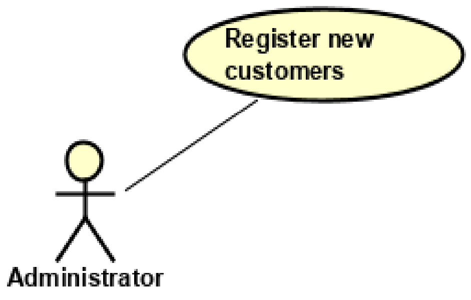


Image 3 Use case diagram example of registering a new customer

UseCase description

It shows how user (in our case Administrator) will perform the task.

Table 8 Use Case description for administrator

ITEM	VALUE
UseCase	Register new customer
Summary	The Actor goes through every step, in order to add a new customer.
Actor	Administrator
Preconditions	Actor must be logged in the system
Postconditions	A new customer is added
Base sequence	1.Press the button “Add new client”. 2.Fill in the form. 3. Press “Add new client” button.
Branch sequence	None
Exception sequence	None
Sub UseCase	None
Note	The user will not be able to add new client if he didn’t complete the form.

2.6 Non-Functional Requirements

Non-Functional Requirements are requirements that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions.

Non-Functional Requirements:

- The system needs to be heterogenous;
- The system needs to have 3 tier Architecture;
- The system needs to have different type of coding CSS, HTML, C#, Java, SQL;
- The system needs to have socket connection;

- The system needs to have web services;
- Each client will have different GUI.

2.7 Delimitations

Delimitations that are set for the project are:

The system will be available only in English- The only language that all of the members of the group speak.

The customers' data will be fake- The data will be fake because there are no real company that will give real data.

3 Design

Class Diagrams could be found in Appendix B.

3.1 Security

The point of security in this project is to make the system safe from trespassers, to avoid attacks, to prevent attacks and what mechanisms are needed to prevent attacks. In the system was implemented the following things:

1. The system has encryption protocol SSL. The data is encrypted and works in this way - Database (C#) →C# →Blazor and back. At each step the information is encrypted and decrypted.
2. The passwords are hashed from the client side and send it through the sockets.

Unfortunately, the team did not have enough time and knowledge to improve the systems security and needs to implement few mechanisms to avoid attacks:

1. Having everything hashed and working just with hashed data. Besides that, to avoid copies of the database, the database needs to be encrypted and using a very good password to avoid brute-force attacks.
2. Authentication and using encryption protocols between sockets communication is needed to prevent information disclosure.
3. To prevent active attackers, for example DoS attacks, a custom rule could be implemented. As an example, the system could have maximum 100 clients and if someone is trying to connect, the client needs to wait in a queue for several seconds/minutes.
4. Brute force attacks - the system should have some limitation of trying to log into the system. For example, after several attempts of someone inserting a wrong password or username, the system should block this port and IP for few minutes.

3.2 Architecture

The architecture represents the blueprint of the system. Its main role is to set a structure of how the application is going to be built. It is crucial for a heterogenous system to have it, in order to establish the relationship between the parties involved. In our case the layers of the application.

3.2.1 Three-Tier Architecture

The system has been built using the three-tier application architecture model, because it offers a clear separation of concerns.

3.2.1.1 Initial architecture

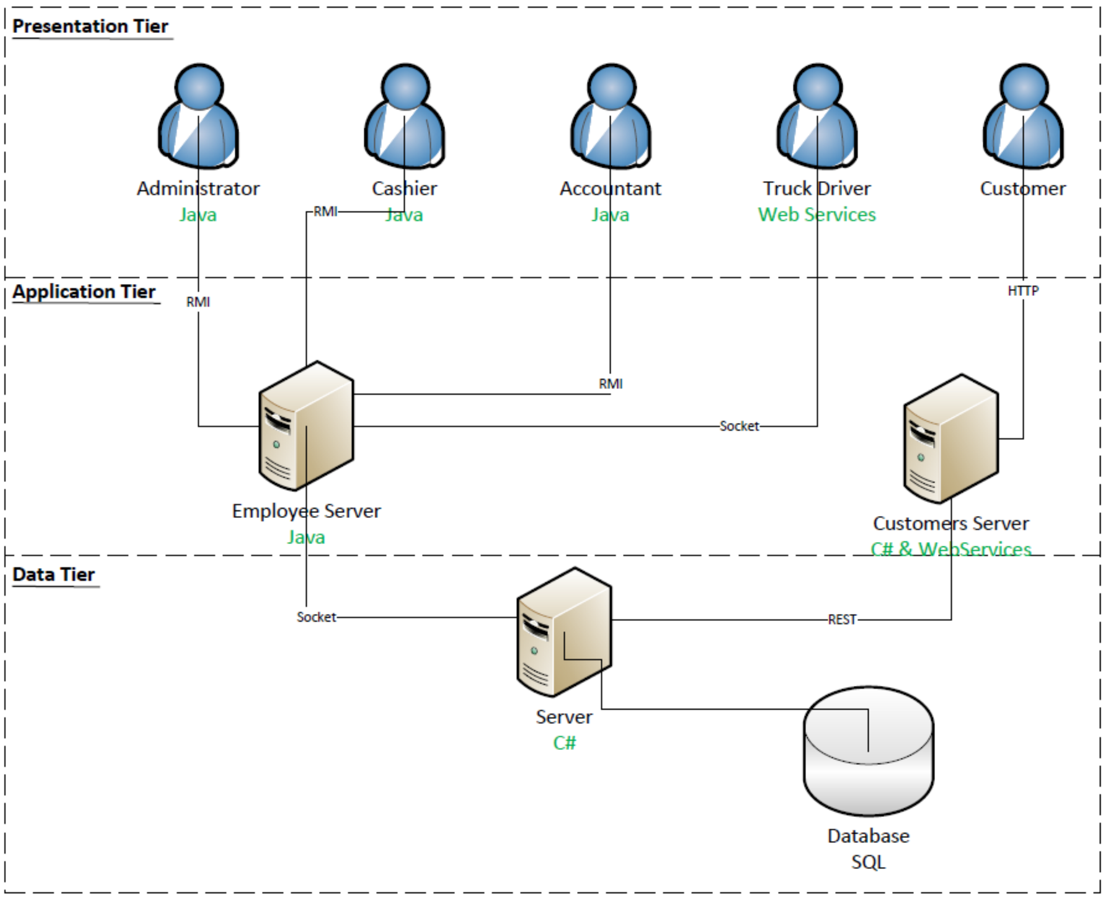


Image 4 Initial 3 Tier Architecture

The first version of our system's design had the following components (*Image 4*):

- Tier 3: a server responsible of the connection with the database
- Tier 2: two servers that allowed the clients to get and send data to the database.
- Tier 1: five clients - each of them having a different access level

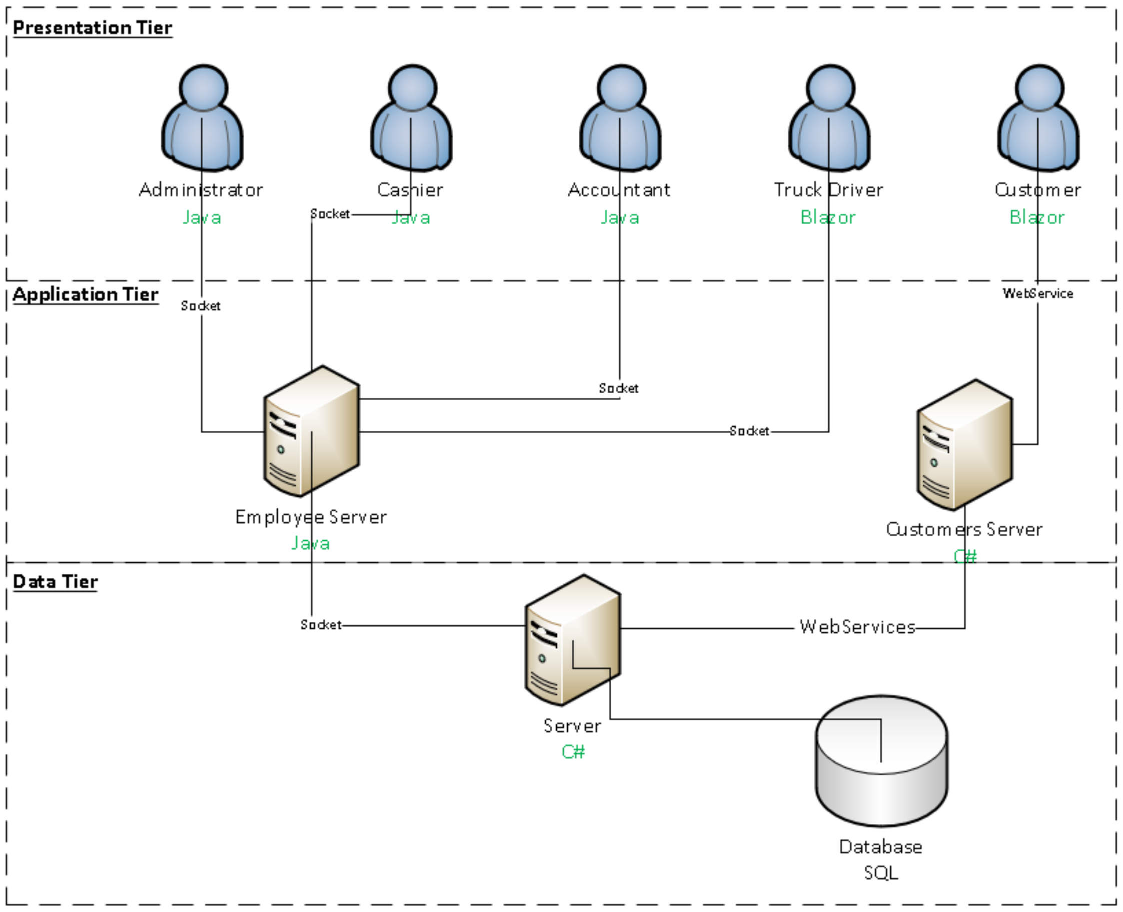


Image 5 Final version of 3 Tier Architecture

In the final version of the system architecture (*Image 5*), the technologies used for each client and server were modified as it follows:

- The connection between the clients on the administration side (Administrator, Cashier and Accountant) and the server responsible for the employee side, Employee Server, is made through TCP Sockets instead of Java RMI, because of the already implemented connection between the server in the third tier and the first server in the second tier, Employee Server.
- The Driver and Customer clients were implemented using the C# framework Blazor.

- The Customer client communicates with the second server on the second tier using web services, Http requests.
- The server on the customer side communicates with the server on the third tier through web services, Http requests.

3.2.1.2 Tier three

The third tier consists of a server. This server is used to establish a connection with the database, then write and retrieve Customer, Employee and Schedule objects from there and then provide the data to the servers in the second tier. The group has taken the decision of having two servers in the second tier, so that the administration side and the customers traffic would not influence each other.

- The connection with the first server, the one in charge of the administration side, Employee Server, is made through TCP sockets because one of the non-functional requirements demanded the use of at least one such connection.
- The connection with the second server, Customer Server, is made through web services, Http requests.

3.2.1.2.1 ER diagram

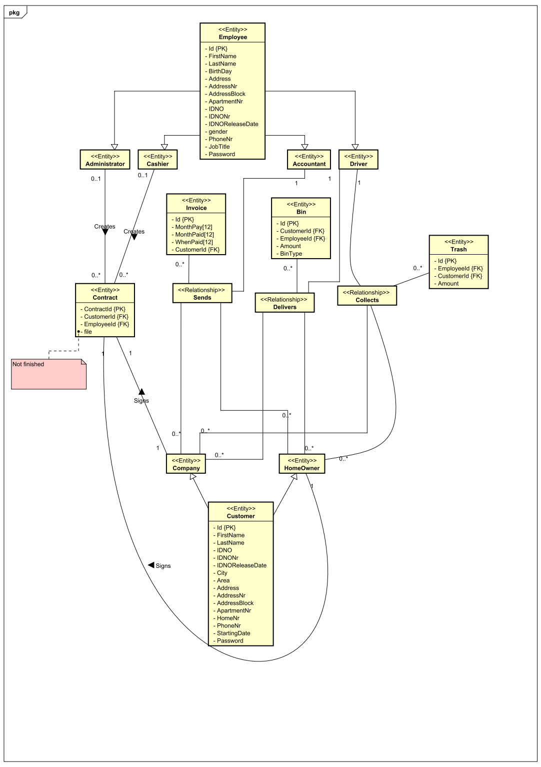


Image 6 ER Diagram

An ER diagram was created to have a well-structured database with all the data that is used across the system. The final version is presented in the above image.

3.2.1.3 Tier two

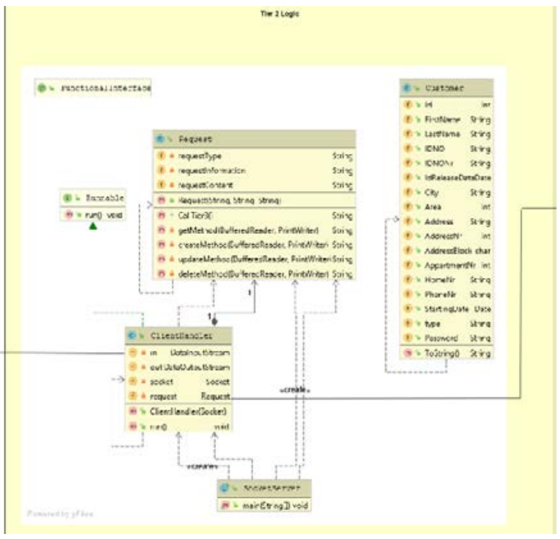


Image 7 Tier 2 Web Services

In *Image 7* is represented class diagrams for tier 2, ClientHandler class has socket method to receive and sent JSONs

The Application tier is responsible for the logic behind the Presentation tier. In this tier there are two servers:

- The first one, Employee Server, is responsible for the administration side, where only the clients who possess employee access - Administrators, Cashiers, Accountants, and Drivers - have access to it. And the connection between the server and clients is made through TCP Sockets.
- The second one, Customer Server, is responsible for the customer side.

3.2.1.4 Tier one

The Presentation tier establishing a connection with the server or servers in the second tier, and then retrieving and sending data to it.

There are five different types of clients:

- Administrator, Cashier and Accountant which were implemented in Java.
- Driver implemented using C#

- Customer also implemented using C#

3.2.1.4.1 MVVM

The MVVM, Model-View-View Model, design pattern was used for the administration side of the system, for Administrator, Cashier and Accountant clients, so that the connection between the view, the view logic, and the business logic could have a smooth connection to each other and have a clear separation of concerns (*Image 8*).

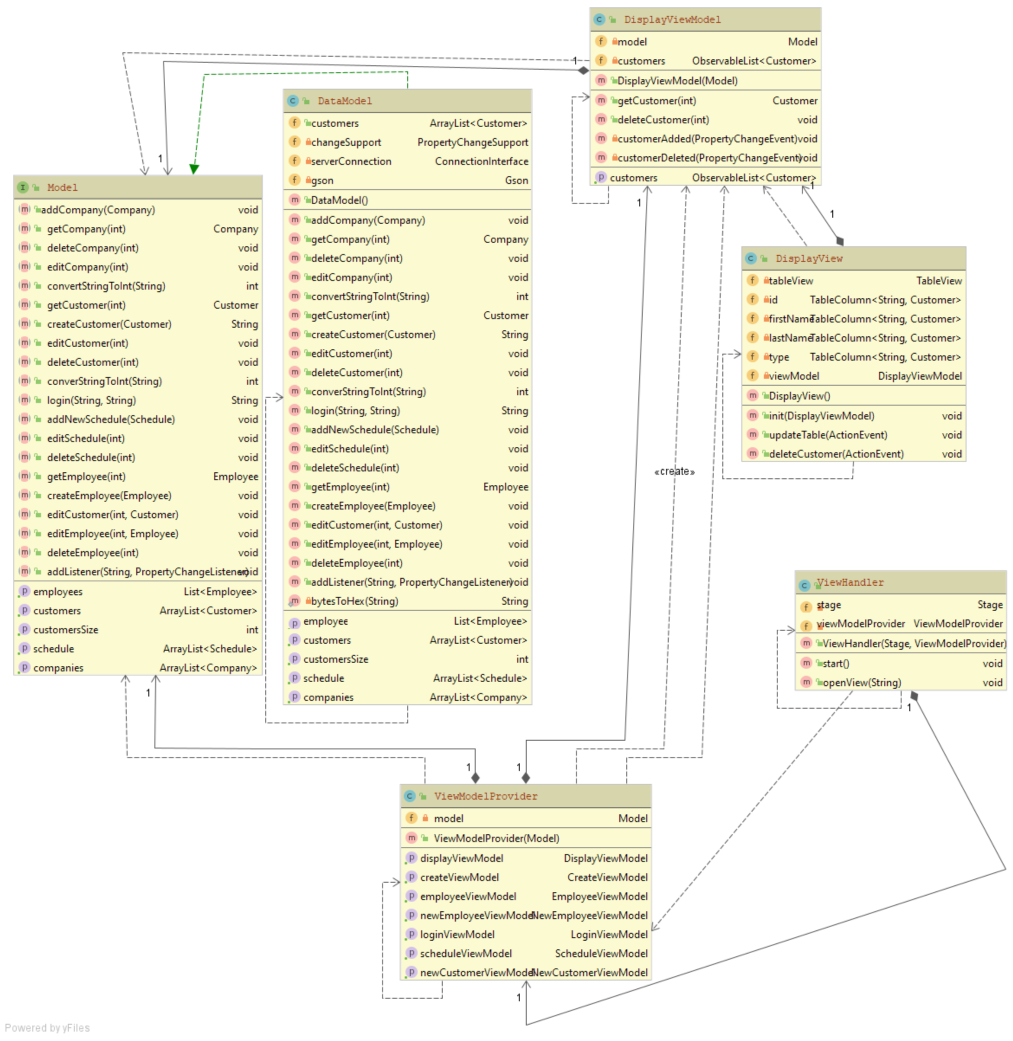

DisplayViewModel
model Model
customers ObservableList<Customer>
DisplayViewModel(Model)
getCustomer(int) Customer
deleteCustomer(int) void
customerAdded(PropertyChangeEvent) void
customerDeleted(PropertyChangeEvent) void
customers ObservableList<Customer>
DataModel
customers ArrayList<Customer>
changeSupport PropertyChangeSupport
serverConnection ConnectionInterface
gson Gson
DataModel()
addCompany(Company) void
getCompany(int) Company
deleteCompany(int) void
editCompany(int) void
convertStringToInt(String) int
getCustomer(int) Customer
createCustomer(Customer) String
editCustomer(int) void
deleteCustomer(int) void
converStringToInt(String) int
login(String, String) String
addNewSchedule(Schedule) void
editSchedule(int) void
deleteSchedule(int) void
getEmployee(int) Employee
createEmployee(Employee) void
editCustomer(int, Customer) void
editEmployee(int, Employee) void
deleteEmployee(int) void
addListener(String, PropertyChangeListener) void
bytesToHex(String) String
employee List<Employee>
customers ArrayList<Customer>
customersSize int
schedule ArrayList<Schedule>
companies ArrayList<Company>
Model
addCompany(Company) void
getCompany(int) Company
deleteCompany(int) void
editCompany(int) void
convertStringToInt(String) int
getCustomer(int) Customer
createCustomer(Customer) String
editCustomer(int) void
deleteCustomer(int) void
converStringToInt(String) int
login(String, String) String
addNewSchedule(Schedule) void
editSchedule(int) void
deleteSchedule(int) void
getEmployee(int) Employee
createEmployee(Employee) void
editCustomer(int, Customer) void
editEmployee(int, Employee) void
deleteEmployee(int) void
addListener(String, PropertyChangeListener) void
employees List<Employee>
customers ArrayList<Customer>
customersSize int
schedule ArrayList<Schedule>
companies ArrayList<Company>
DisplayView
tableView TableView
id TableColumn<String, Customer>
firstName TableColumn<String, Customer>
lastName TableColumn<String, Customer>
type TableColumn<String, Customer>
viewModel DisplayViewModel
DisplayView()
init(DisplayViewModel) void
updateTable(ActionEvent) void
deleteCustomer(ActionEvent) void
«create»
ViewHandler
stage Stage
viewModelProvider ViewModelProvider
ViewHandler(Stage, ViewModelProvider)
start() void
openView(String) void
ViewModelProvider
model Model
ViewModelProvider(Model)
displayViewModel DisplayViewModel
createViewModel CreateViewModel
employeeViewModel EmployeeViewModel
newEmployeeViewModel NewEmployeeViewModel
loginViewModel LoginViewModel
scheduleViewModel ScheduleViewModel
newCustomerViewModel NewCustomerViewModel
1
Powered by yFiles

Image 8 MVVM Pattern

4 Implementation

Administrator Login

The log in is the first step for the Administrator to acces the program.

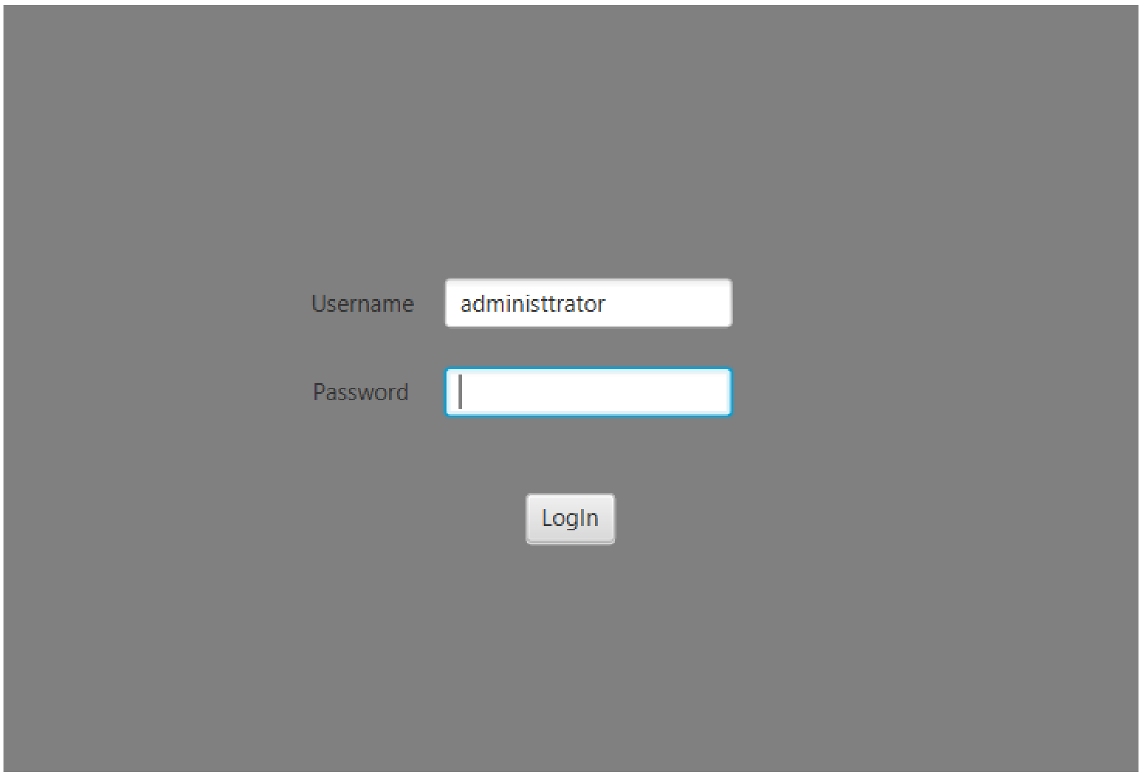
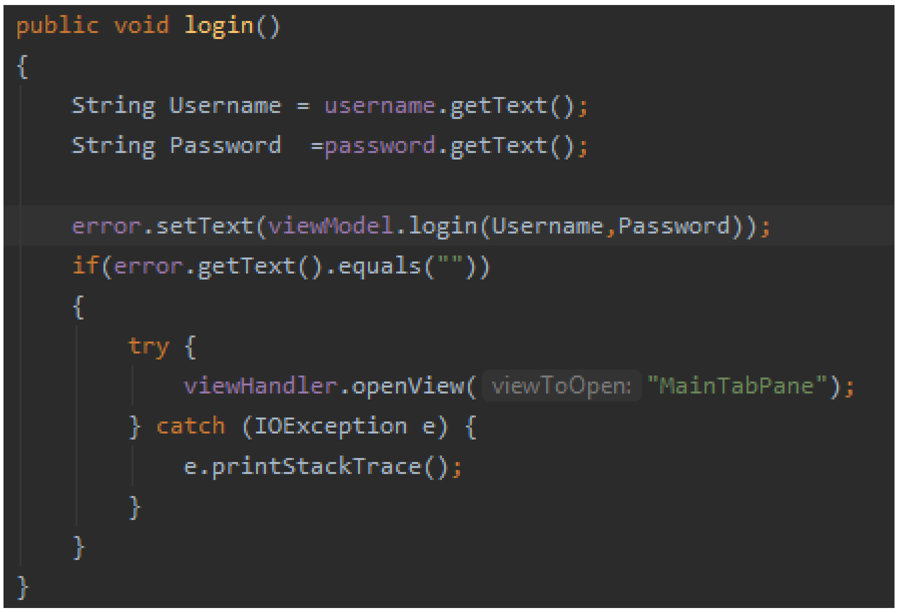


Image 9 Administrator login view

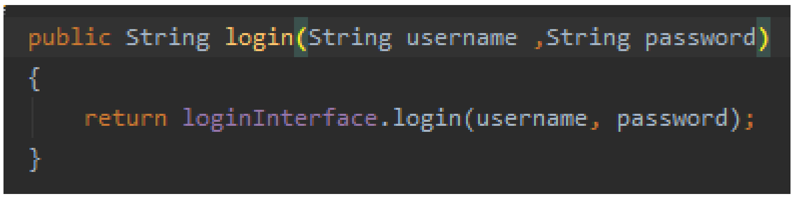
The administrators graphical user interface(GUI) is made with ModelViewViewModel design pattern.



```
public void login()
{
    String Username = username.getText();
    String Password  =password.getText();

    error.setText(viewModel.login(Username,Password));
    if(error.getText().equals(""))
    {
        try {
            viewHandler.openView( viewToOpen: "MainTabPane");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Image 10 Login View - login method

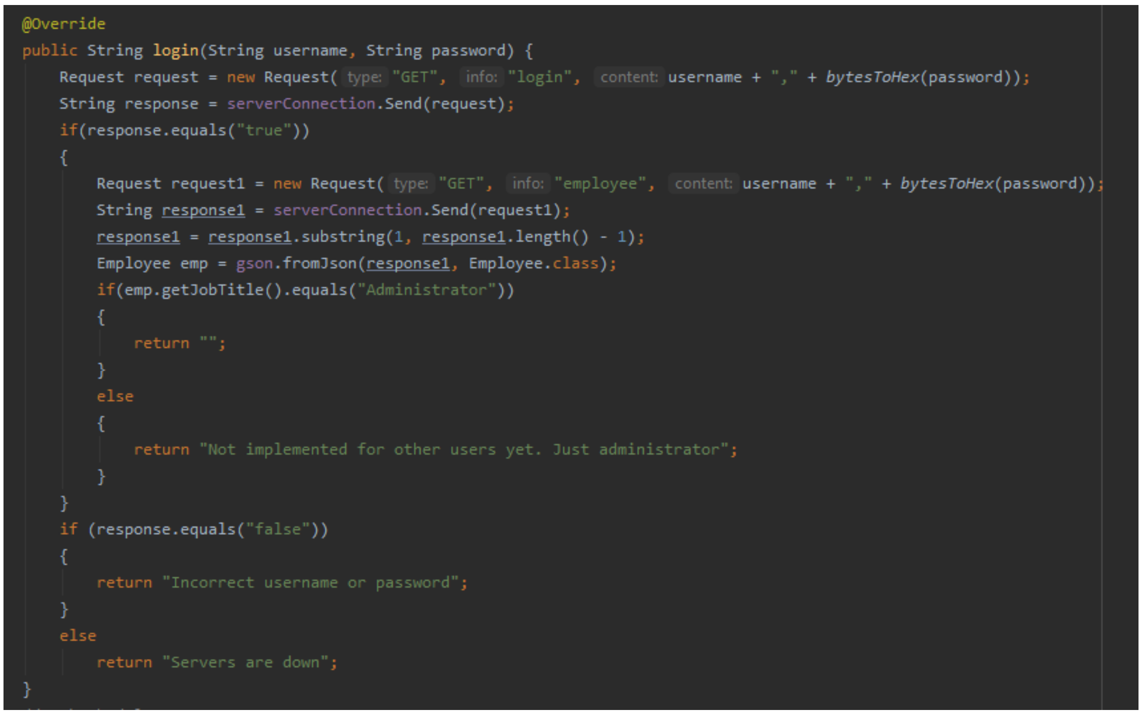
The view takes the values from the text fields in the view and sends those values to the viewmodel that is the intermediate between the view and the model.



```
public String login(String username ,String password)
{
    return loginInterface.login(username, password);
}
```

Image 11 Login view model

The view model just sends the received information to the model.

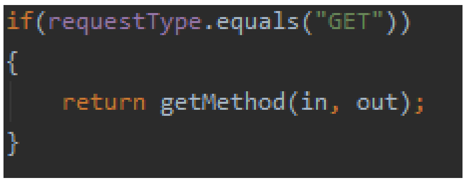


```
@Override
public String login(String username, String password) {
    Request request = new Request( type: "GET",  info: "login",  content: username + "," + bytesToHex(password));
    String response = serverConnection.Send(request);
    if(response.equals("true"))
    {
        Request request1 = new Request( type: "GET",  info: "employee",  content: username + "," + bytesToHex(password));
        String response1 = serverConnection.Send(request1);
        response1 = response1.substring(1, response1.length() - 1);
        Employee emp = gson.fromJson(response1, Employee.class);
        if(emp.getJobTitle().equals("Administrator"))
        {
            return "";
        }
        else
        {
            return "Not implemented for other users yet. Just administrator";
        }
    }
    if (response.equals("false"))
    {
        return "Incorrect username or password";
    }
    else
        return "Servers are down";
}
```

Image 12 Login model

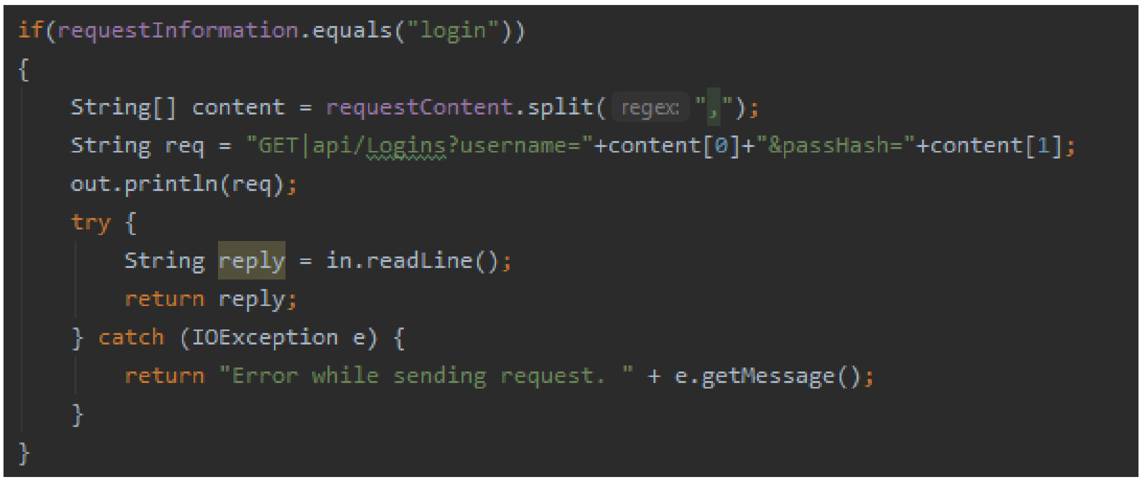
In the model the password is hashed using SHA-256 algorithm and sent to the tier 2 server via socket connection. Acording to the response from the tier 2 server, the model will send back a response to the view model and to the view.

Socket protocol consists of the type of the request(GET, CREATE, UPDATE or DELETE), information field that lets the server know which table the request should be applied to and a context field that is the necessary information for the changes.



```
if(requestType.equals("GET"))
{
    return getMethod(in, out);
}
```

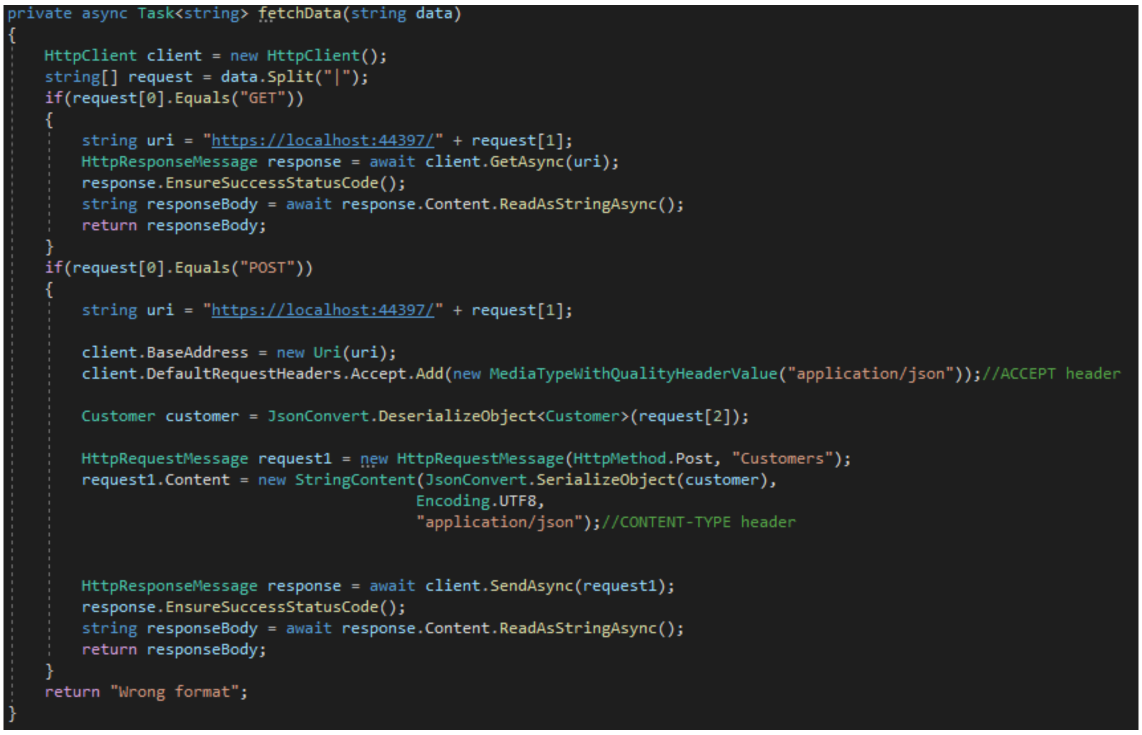
Image 13 Tier 2 Login request



```
if(requestInformation.equals("login"))
{
    String[] content = requestContent.split( regex: ",");
    String req = "GET|api/Logins?username="+content[0]+"&passHash="+content[1];
    out.println(req);
    try {
        String reply = in.readLine();
        return reply;
    } catch (IOException e) {
        return "Error while sending request. " + e.getMessage();
    }
}
```

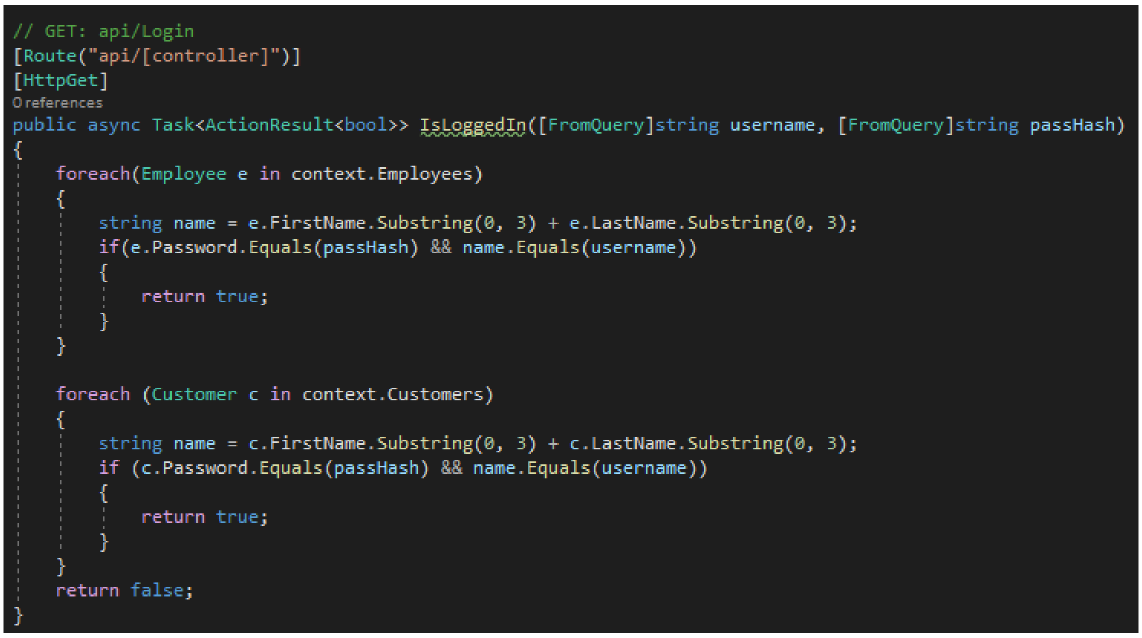
Image 14 Tier 2 Login

On the tier 2 server the request is split up. First part of the request is type and it determines which method to run. Next up the information variable determines which exact request to send to the tier 3 server. And the context is added accordingly to the request. The request is sent through the socket connection between tier 2 and tier 3.



```
private async Task<string> fetchData(string data)
{
    HttpClient client = new HttpClient();
    string[] request = data.Split("|");
    if(request[0].Equals("GET"))
    {
        string uri = "https://localhost:44397/" + request[1];
        HttpResponseMessage response = await client.GetAsync(uri);
        response.EnsureSuccessStatusCode();
        string responseBody = await response.Content.ReadAsStringAsync();
        return responseBody;
    }
    if(request[0].Equals("POST"))
    {
        string uri = "https://localhost:44397/" + request[1];

        client.BaseAddress = new Uri(uri);
        client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));//ACCEPT header

        Customer customer = JsonConvert.DeserializeObject<Customer>(request[2]);

        HttpRequestMessage request1 = new HttpRequestMessage(HttpMethod.Post, "Customers");
        request1.Content = new StringContent(JsonConvert.SerializeObject(customer),
                                    Encoding.UTF8,
                                    "application/json");//CONTENT-TYPE header


        HttpResponseMessage response = await client.SendAsync(request1);
        response.EnsureSuccessStatusCode();
        string responseBody = await response.Content.ReadAsStringAsync();
        return responseBody;
    }
    return "Wrong format";
}
```

Image 15 Tier 3 fetch data



```
// GET: api/Login
[Route("api/[controller]")]
[HttpGet]
0 references
public async Task<ActionResult<bool>> IsLoggedIn([FromQuery]string username, [FromQuery]string passHash)
{
    foreach(Employee e in context.Employees)
    {
        string name = e.FirstName.Substring(0, 3) + e.LastName.Substring(0, 3);
        if(e.Password.Equals(passHash) && name.Equals(username))
        {
            return true;
        }
    }

    foreach (Customer c in context.Customers)
    {
        string name = c.FirstName.Substring(0, 3) + c.LastName.Substring(0, 3);
        if (c.Password.Equals(passHash) && name.Equals(username))
        {
            return true;
        }
    }
    return false;
}
```

Image 16 Tier 3 web service HttpGet

When someone connects to the tier 3 sockets, the server fetches data through the web service running on the tier 3 server. The web service returns a json object from the database and return it all the way back to tier 1.

At the moment he logged in, administrator sees the view that displays a list of the customers.

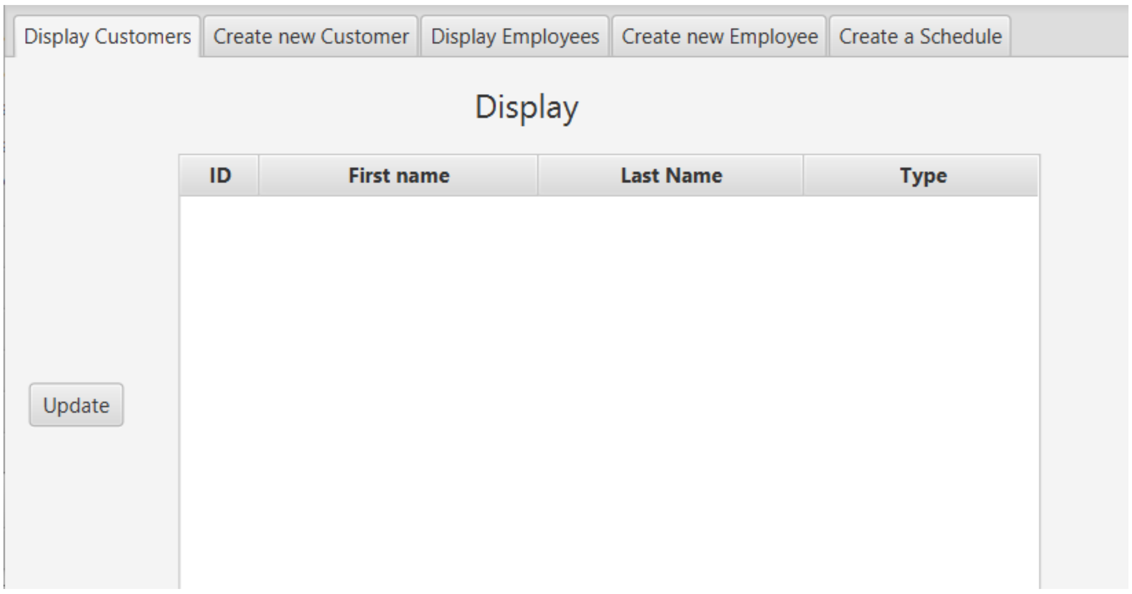
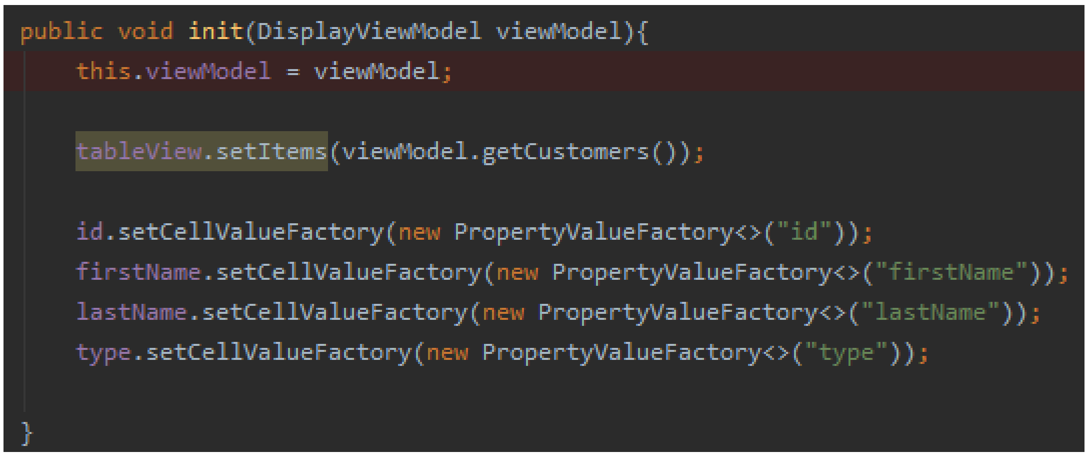


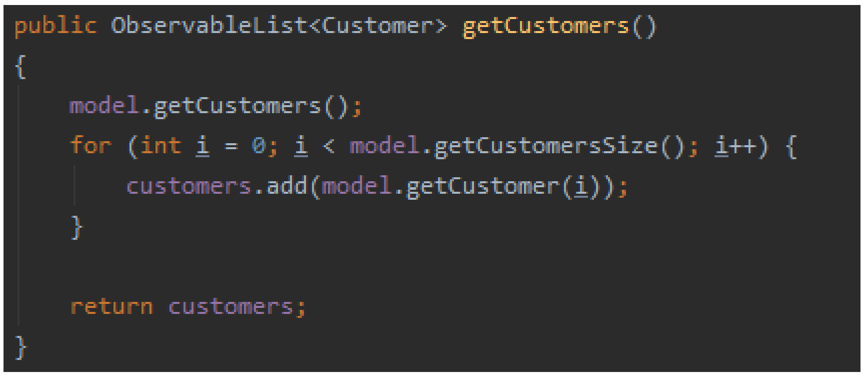
Image 17 Administrator view all customers



```
public void init(DisplayViewModel viewModel){
    this.viewModel = viewModel;

    tableView.setItems(viewModel.getCustomers());

    id.setCellValueFactory(new PropertyValueFactory<>("id"));
    firstName.setCellValueFactory(new PropertyValueFactory<>("firstName"));
    lastName.setCellValueFactory(new PropertyValueFactory<>("lastName"));
    type.setCellValueFactory(new PropertyValueFactory<>("type"));

}
```

Image 18 Display customers initialization method

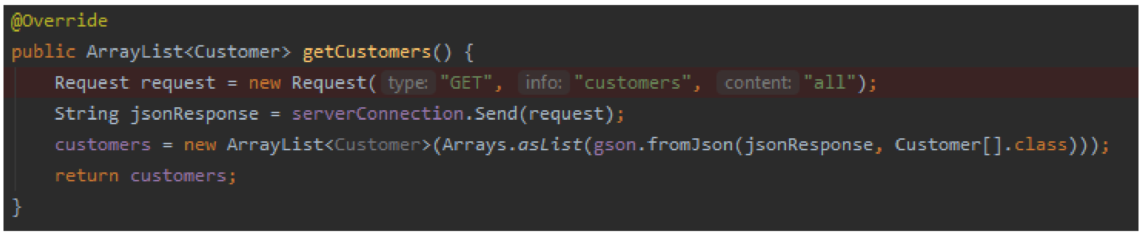
As the display tab is initialized a get method is called on the view model.



```
public ObservableList<Customer> getCustomers()
{
    model.getCustomers();
    for (int i = 0; i < model.getCustomersSize(); i++) {
        customers.add(model.getCustomer(i));
    }

    return customers;
}
```

Image 19 Display view model get all customers

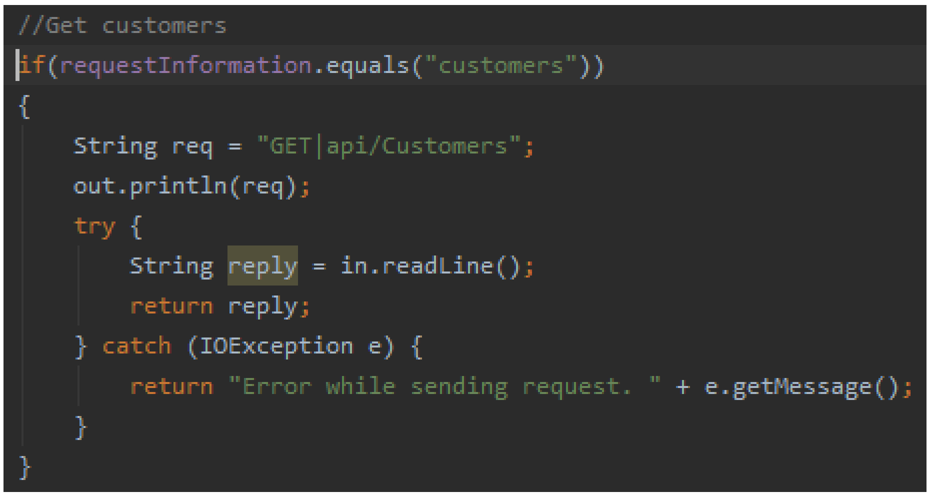
All of the customers are retrieved and added to a observable list.



```
@Override
public ArrayList<Customer> getCustomers() {
    Request request = new Request( type: "GET",  info: "customers",  content: "all");
    String jsonResponse = serverConnection.Send(request);
    customers = new ArrayList<Customer>(Arrays.asList(gson.fromJson(jsonResponse, Customer[].class)));
    return customers;
}
```

Image 20 Model customer request

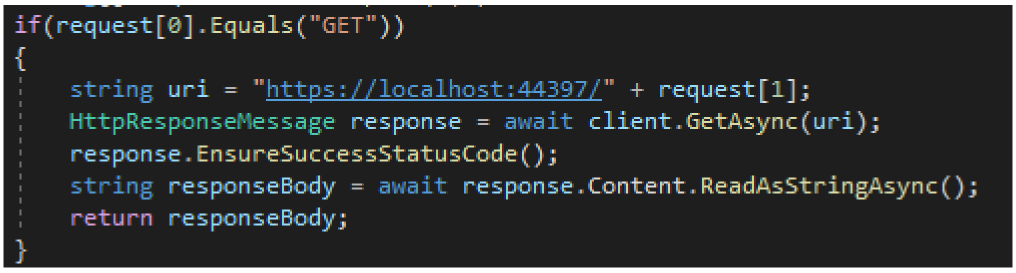
A request is created and sent to the tier 2 server.



```
//Get customers
if(requestInformation.equals("customers"))
{
    String req = "GET|api/Customers";
    out.println(req);
    try {
        String reply = in.readLine();
        return reply;
    } catch (IOException e) {
        return "Error while sending request. " + e.getMessage();
    }
}
```

Image 21 Tier 2 customer request

The tier 2 server reads the request and sends a new request to the tier 3 server.

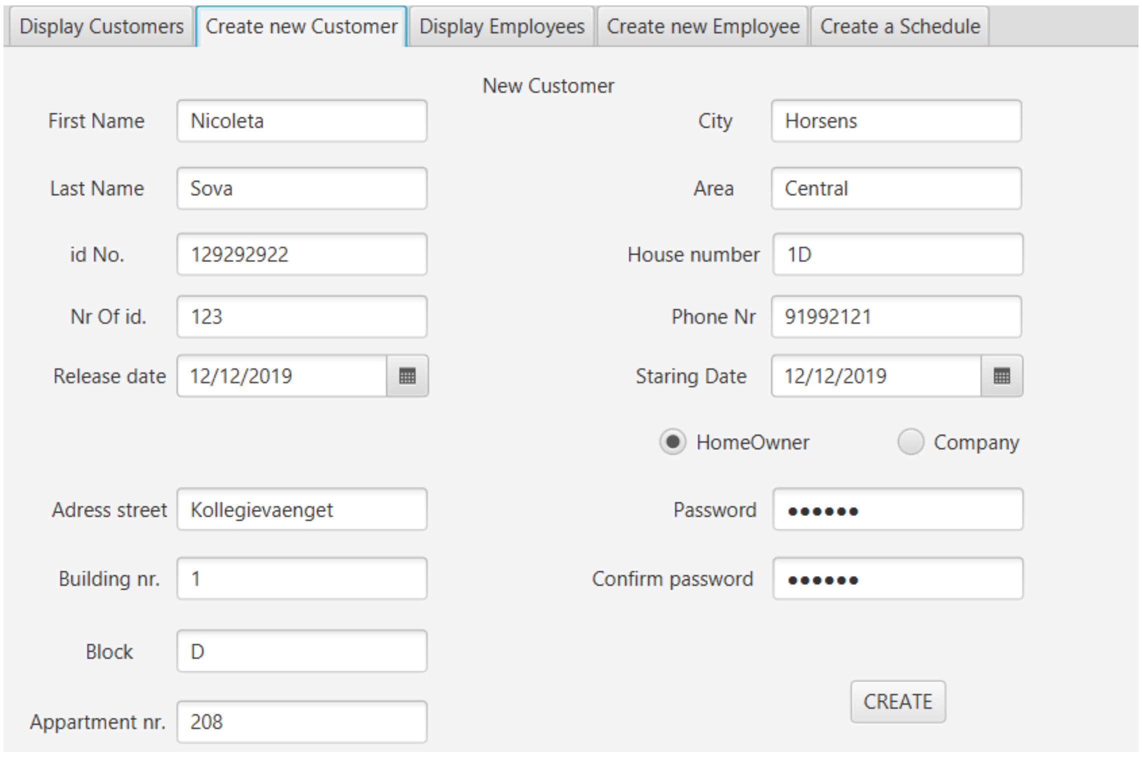


```
if(request[0].Equals("GET"))
{
    string uri = "https://localhost:44397/" + request[1];
    HttpResponseMessage response = await client.GetAsync(uri);
    response.EnsureSuccessStatusCode();
    string responseBody = await response.Content.ReadAsStringAsync();
    return responseBody;
}
```

Image 22 Tier 3 get all customers

Where the request is read and the required informations is retrieved from the database through the web service.

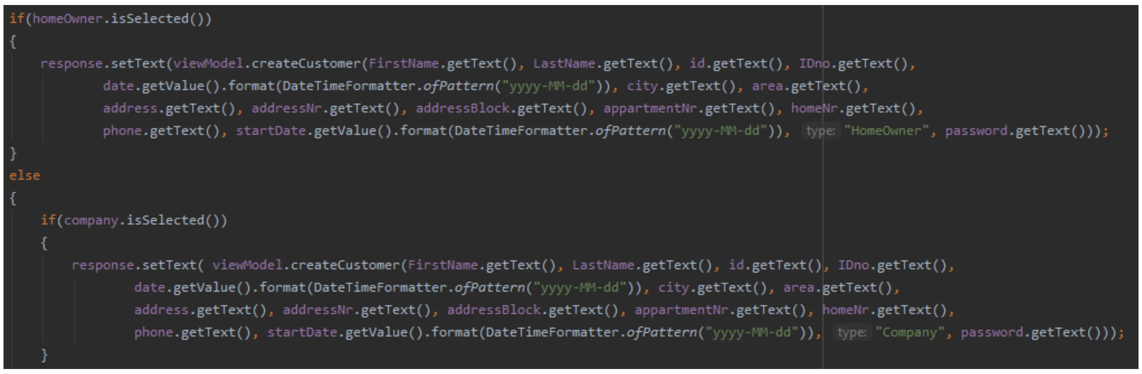
In order to Create new customer the administrator has to press “Create new Customer” button and fill in the form.



New Customer

First Name	Nicoleta	City	Horsens
Last Name	Sova	Area	Central
id No.	129292922	House number	1D
Nr Of id.	123	Phone Nr	91992121
Release date	12/12/2019	Staring Date	12/12/2019
		[✓ HomeOwner]	Company
Adress street	Kollegievaenget	Password	••••••
Building nr.	1	Confirm password	••••••
Block	D		
Appartment nr.	208		

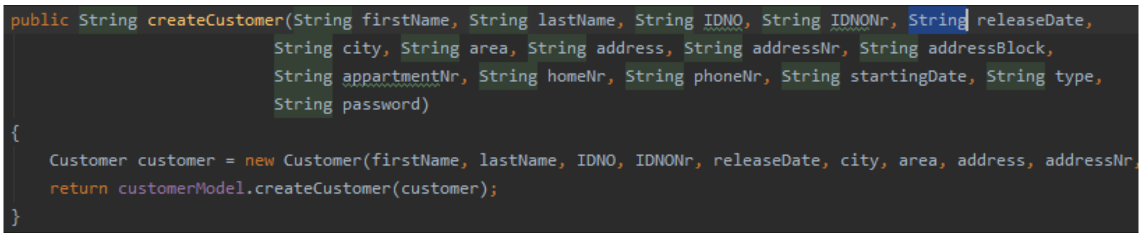
Image 23 Administrator create a new customer



```
if(homeOwner.isSelected())
{
    response.setText(viewModel.createCustomer(FirstName.getText(), LastName.getText(), id.getText(), IDno.getText(),
            date.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")), city.getText(), area.getText(),
            address.getText(), addressNr.getText(), addressBlock.getText(), appartmentNr.getText(), homeNr.getText(),
            phone.getText(), startDate.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")),  type: "HomeOwner", password.getText()));
}
else
{
    if(company.isSelected())
    {
        response.setText( viewModel.createCustomer(FirstName.getText(), LastName.getText(), id.getText(), IDno.getText(),
                date.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")), city.getText(), area.getText(),
                address.getText(), addressNr.getText(), addressBlock.getText(), appartmentNr.getText(), homeNr.getText(),
                phone.getText(), startDate.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd")),  type: "Company", password.getText()));
    }
```

Image 24 Customer info send to view model

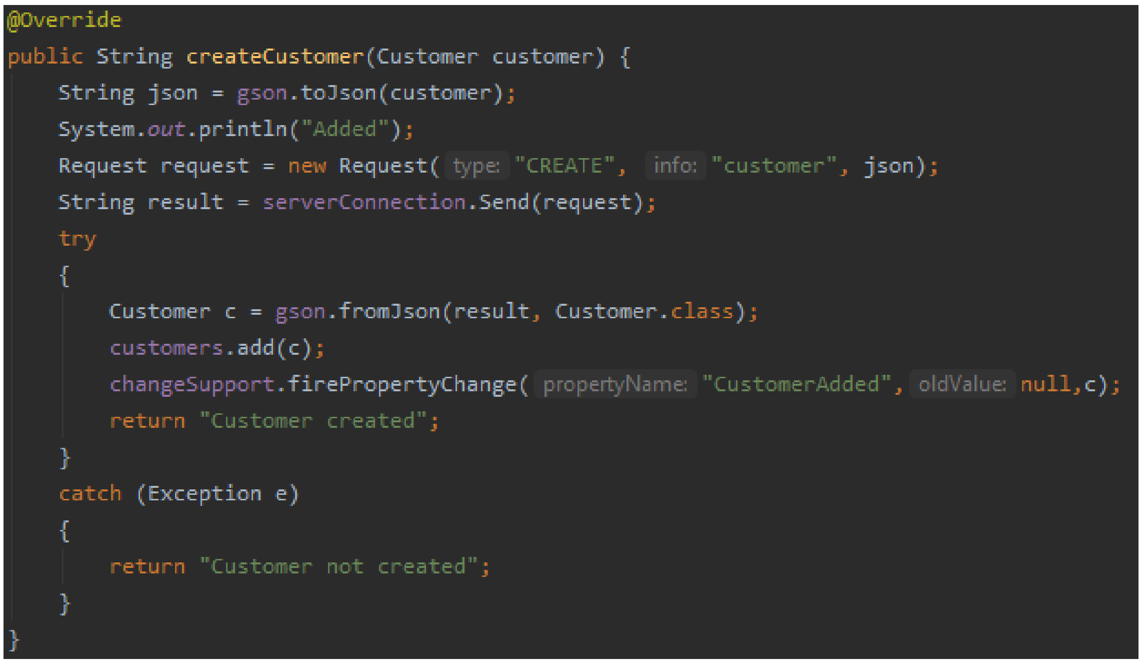
As the “Create” button is pressed, all of the text field information is sent to the view model.



```
public String createCustomer(String firstName, String lastName, String IDNO, String IDNONr, String releaseDate,
                             String city, String area, String address, String addressNr, String addressBlock,
                             String appartmentNr, String homeNr, String phoneNr, String startingDate, String type,
                             String password)
{
    Customer customer = new Customer(firstName, lastName, IDNO, IDNONr, releaseDate, city, area, address, addressNr,
    return customerModel.createCustomer(customer);
}
```

Image 25 Create customer view model

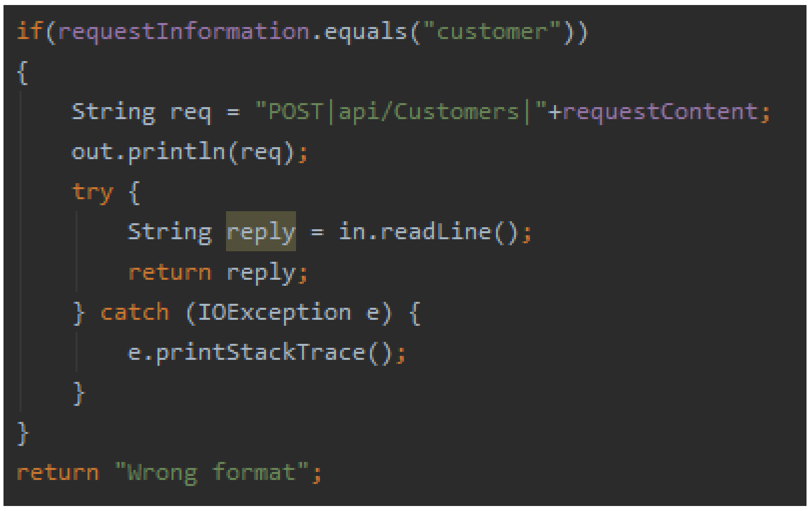
In the view model a customer object is created and sent to the model.



```
@Override
public String createCustomer(Customer customer) {
    String json = gson.toJson(customer);
    System.out.println("Added");
    Request request = new Request( type: "CREATE",  info: "customer", json);
    String result = serverConnection.Send(request);
    try
    {
        Customer c = gson.fromJson(result, Customer.class);
        customers.add(c);
        changeSupport.firePropertyChange( propertyName: "CustomerAdded",  oldValue: null,c);
        return "Customer created";
    }
    catch (Exception e)
    {
        return "Customer not created";
    }
}
```

Image 26 model create customer

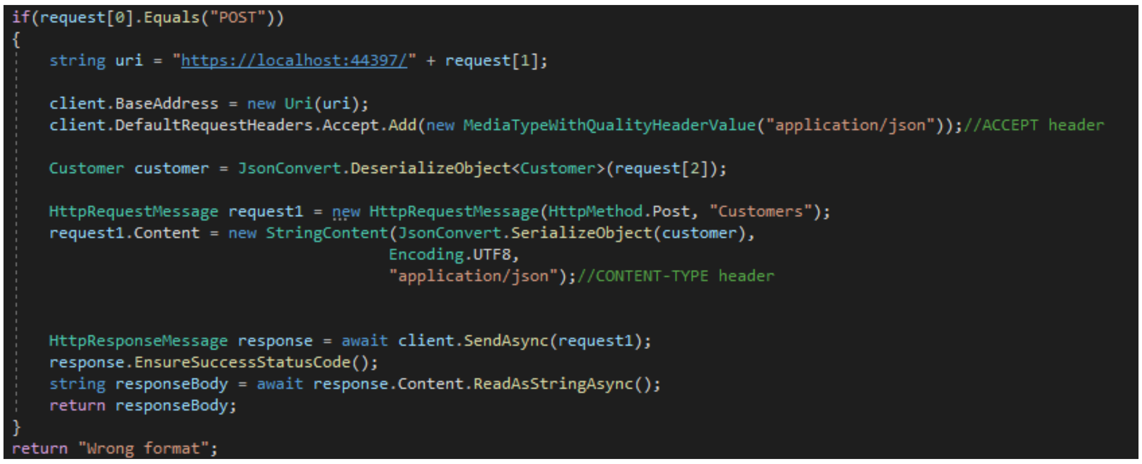
A create request is created and sent to the tier 2 server. According to the response from the tier 2 server the label in the view will change.



```
if(requestInformation.equals("customer"))
{
    String req = "POST|api/Customers|"+requestContent;
    out.println(req);
    try {
        String reply = in.readLine();
        return reply;
    } catch (IOException e) {
        e.printStackTrace();
    }
}
return "Wrong format";
```

Image 27 Tier 2 create customer

Tier 2 servers receives the request, creates a new request and sends it to tier 3 server and waits for a response. When the response is received, sends it back to tier 1.

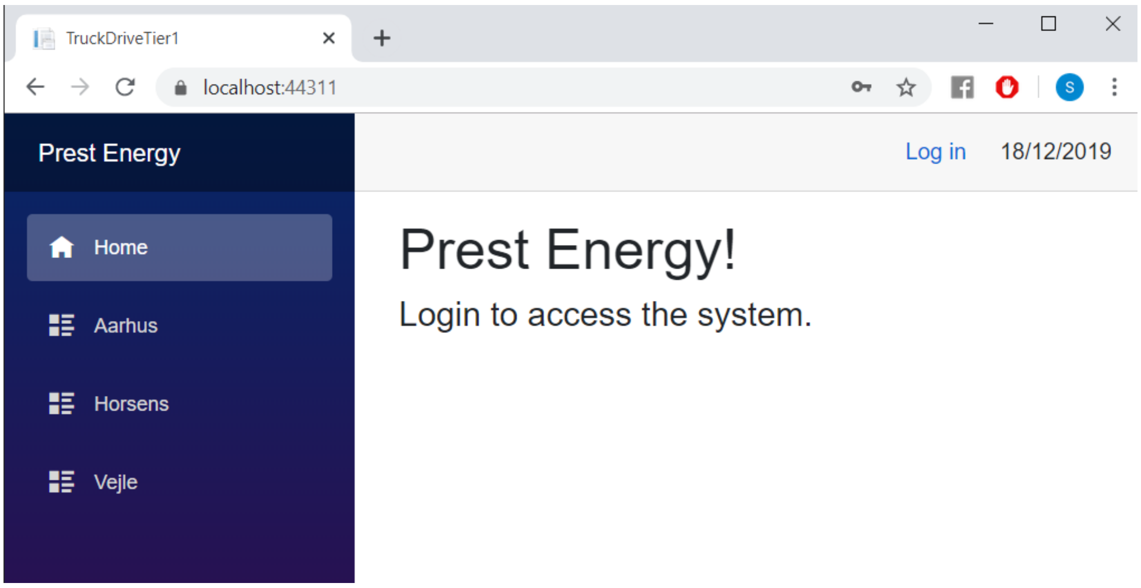


```
if(request[0].Equals("POST"))
{
    string uri = "https://localhost:44397/" + request[1];

    client.BaseAddress = new Uri(uri);
    client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));//ACCEPT header

    Customer customer = JsonConvert.DeserializeObject<Customer>(request[2]);

    HttpRequestMessage request1 = new HttpRequestMessage(HttpMethod.Post, "Customers");
    request1.Content = new StringContent(JsonConvert.SerializeObject(customer),
                                        Encoding.UTF8,
                                        "application/json");//CONTENT-TYPE header


    HttpResponseMessage response = await client.SendAsync(request1);
    response.EnsureSuccessStatusCode();
    string responseBody = await response.Content.ReadAsStringAsync();
    return responseBody;
}
return "Wrong format";
```

Image 28 Tier 3 create customer

In the tier 3 socket handler reads the request, creates a HTTPS request and retrieves the desired data from the database through the web service.

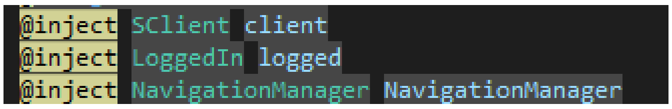
For the driver user, the first widow he sees is Home page.



Prest Energy!

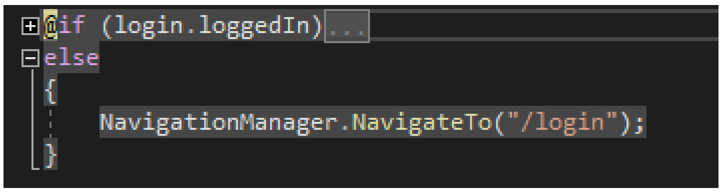
Login to access the system.

Image 29 Driver main view



```
@inject SClient client
@inject LoggedIn logged
@inject NavigationManager NavigationManager
```

Image 30 Page infections



```
@if (login.loggedIn)...
else
{
    NavigationManager.NavigateTo("/login");
}
```

Image 31 Driver check if logged in

The user is not able to go to any page except login. If the user tries to go to any page, they will be redirected to the login.

He cannot access anything without being logged in. He has to insert the username and password and then press log in button.

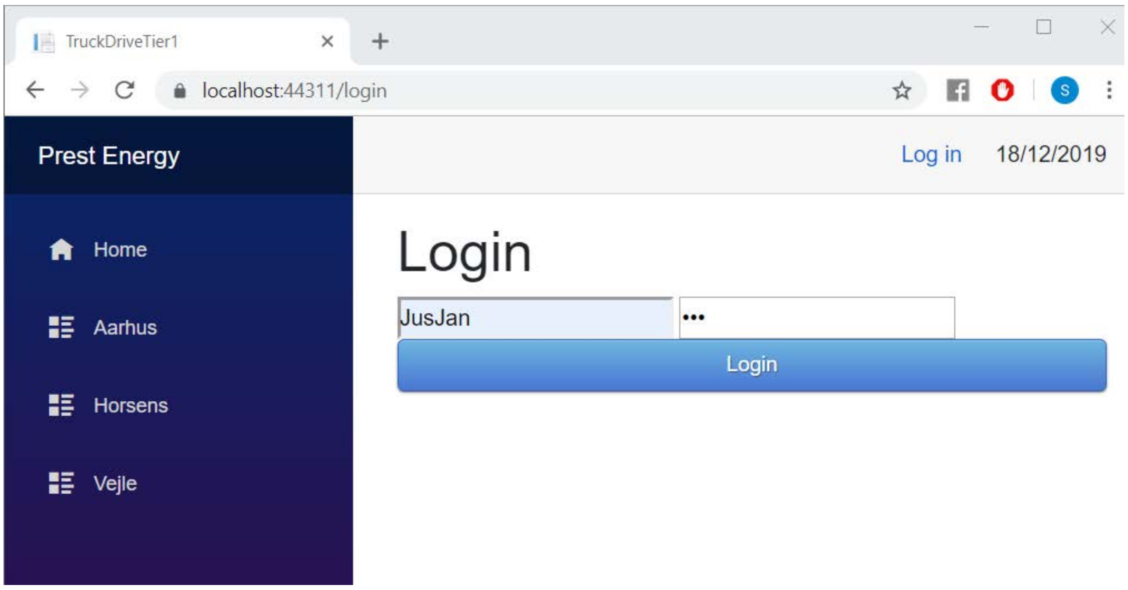
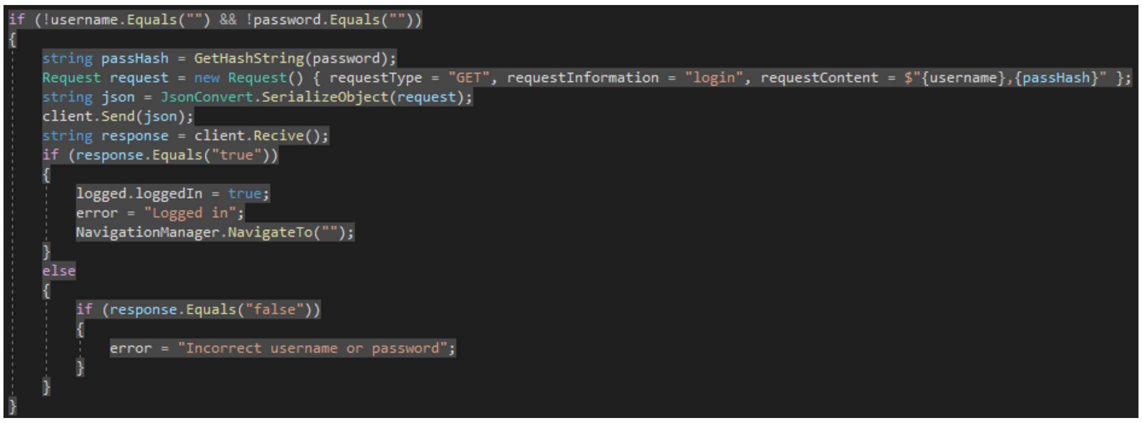


Image 32 Driver login



```
if (!username.Equals("") && !password.Equals(""))
{
    string passHash = GetHashString(password);
    Request request = new Request() { requestType = "GET", requestInformation = "login", requestContent = $"{username},{passHash}" };
    string json = JsonConvert.SerializeObject(request);
    client.Send(json);
    string response = client.Recive();
    if (response.Equals("true"))
    {
        logged.loggedIn = true;
        error = "Logged in";
        NavigationManager.NavigateTo("");
    }
    else
    {
        if (response.Equals("false"))
        {
            error = "Incorrect username or password";
        }
    }
}
```

Image 33 Driver login method

Once the user inputs a username, a password and presses the login button, a request object is created, parsed to a json string and sent to the socket client class. If the response from the servers is true, then all of the websites become accessible.

From the moment he logged in, he has access to the cities and can see the clients that are in that cites

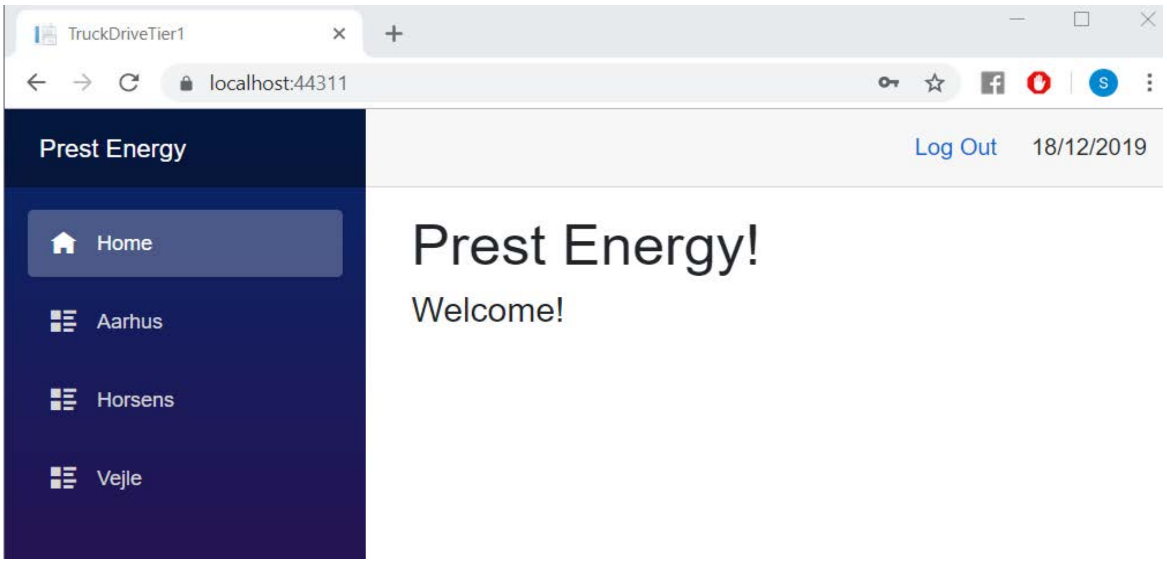
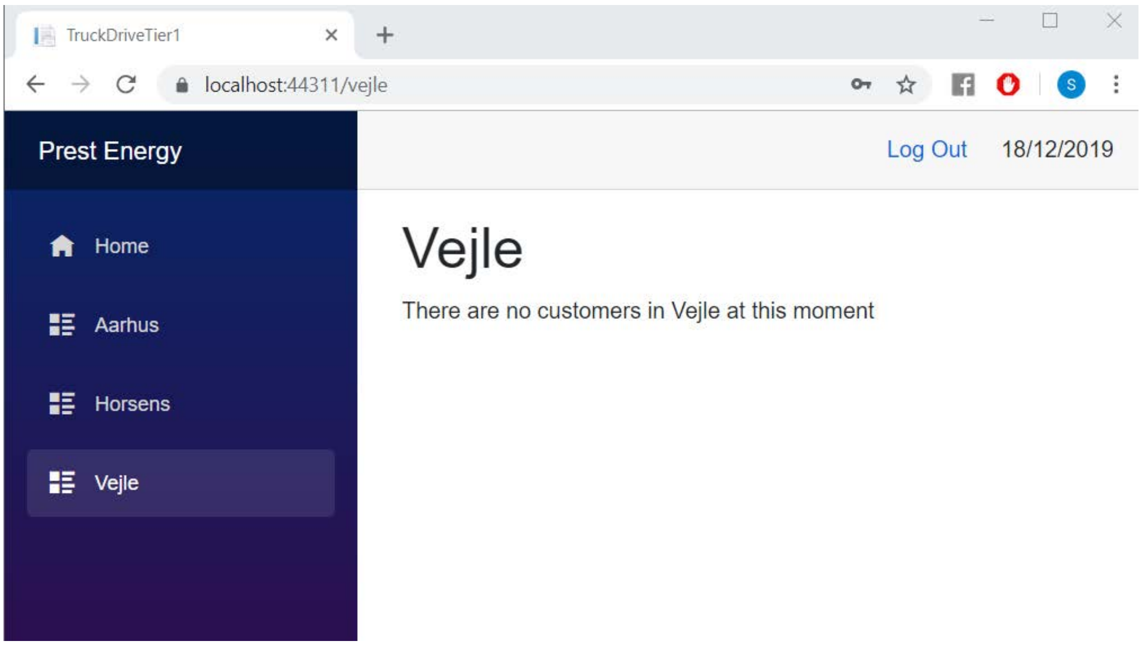


Image 34 Driver main view

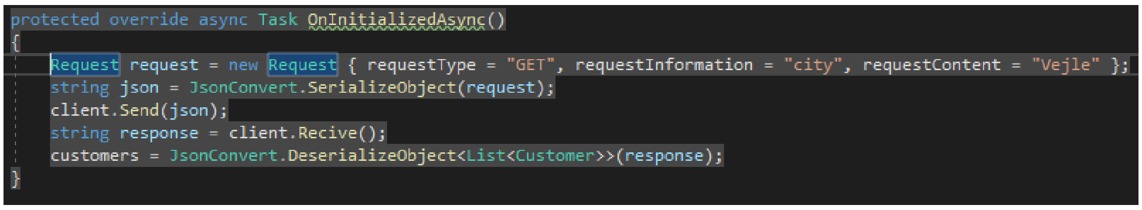
If there are no client, it will be written “There are no customers at this moment”



Vejle

There are no customers in Vejle at this moment

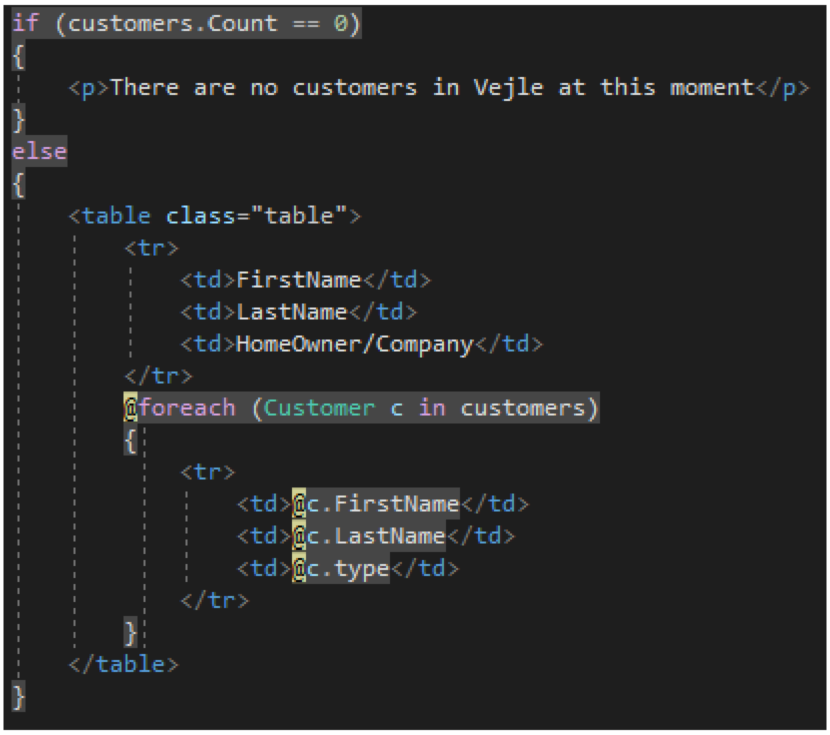
Image 35 Vejle city customers



```
protected override async Task OnInitializedAsync()
{
    Request request = new Request { requestType = "GET", requestInformation = "city", requestContent = "Vejle" };
    string json = JsonConvert.SerializeObject(request);
    client.Send(json);
    string response = client.Recive();
    customers = JsonConvert.DeserializeObject<List<Customer>>(response);
}
```

Image 36 Vejle city initialize method

When the page is open, then the client sends a request of all of the customers in that specific city.

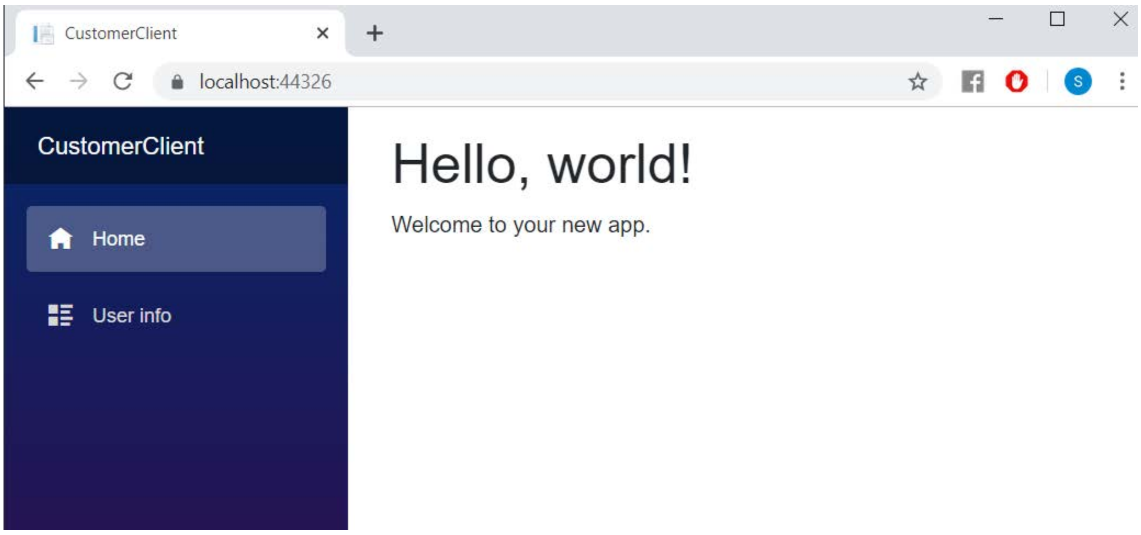


```
if (customers.Count == 0)
{
    <p>There are no customers in Vejle at this moment</p>
}
else
{
    <table class="table">
        <tr>
            <td>FirstName</td>
            <td>LastName</td>
            <td>HomeOwner/Company</td>
        </tr>
        @foreach (Customer c in customers)
        {
            <tr>
                <td>@c.FirstName</td>
                <td>@c.LastName</td>
                <td>@c.type</td>
            </tr>
        }
    </table>
}
```

Image 37 City page write all customers

And displays the response.

For the client, the first page is Home page.

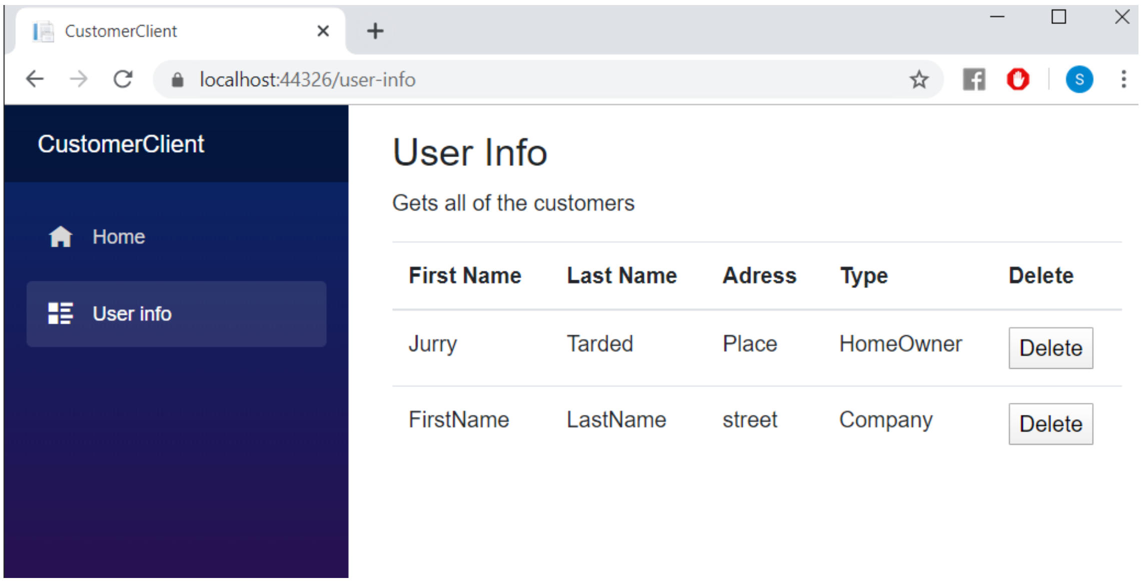


Hello, world!

Welcome to your new app.

Image 38 Customer main view

The user can access User info

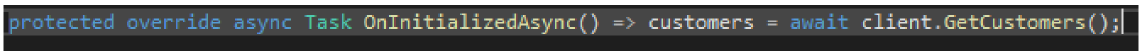


User Info

Gets all of the customers

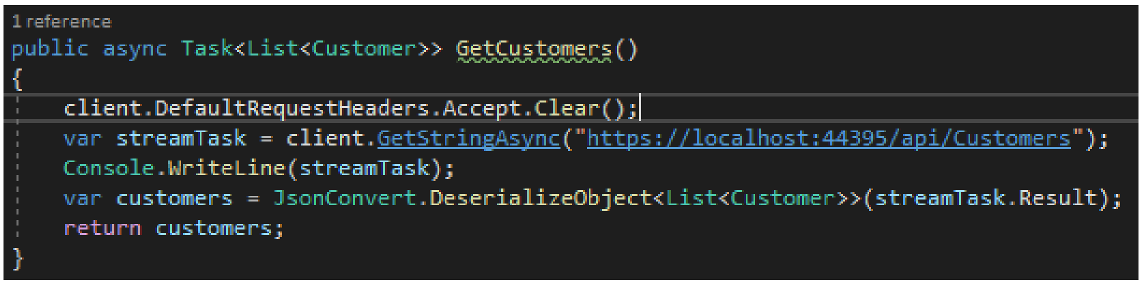
First Name	Last Name	Adress	Type	Delete
Jurry	Tarded	Place	HomeOwner	Delete
FirstName	LastName	street	Company	Delete

Image 39 Customer show all customers



```
protected override async Task OnInitializedAsync() => customers = await client.GetCustomers();
```

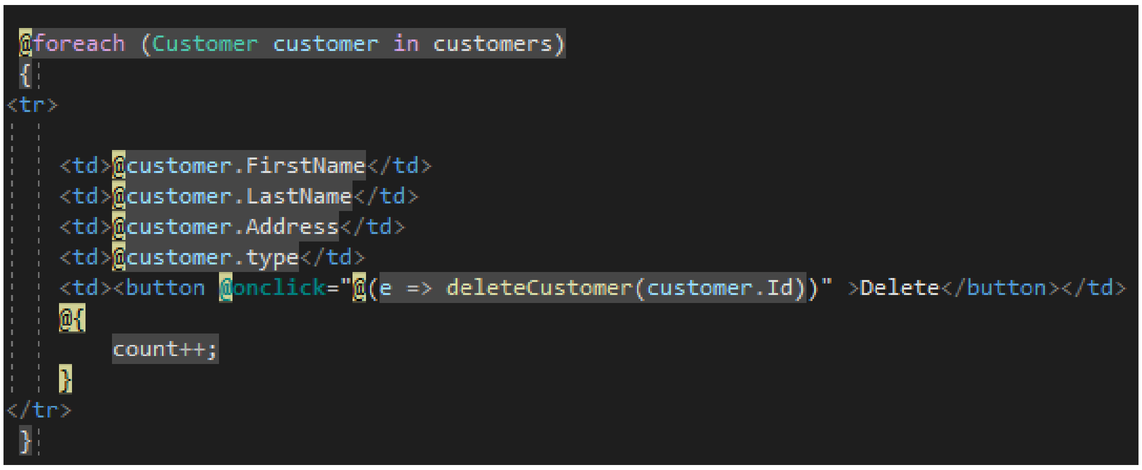
Image 40 Show all customers initialization



```
1 reference
public async Task<List<Customer>> GetCustomers()
{
    client.DefaultRequestHeaders.Accept.Clear();
    var streamTask = client.GetStringAsync("https://localhost:44395/api/Customers");
    Console.WriteLine(streamTask);
    var customers = JsonConvert.DeserializeObject<List<Customer>>(streamTask.Result);
    return customers;
}
```

Image 41 Tier 1 get all customers

When the user info page is open, the program sends a request to the web service that is located on the C# tier 2 server.

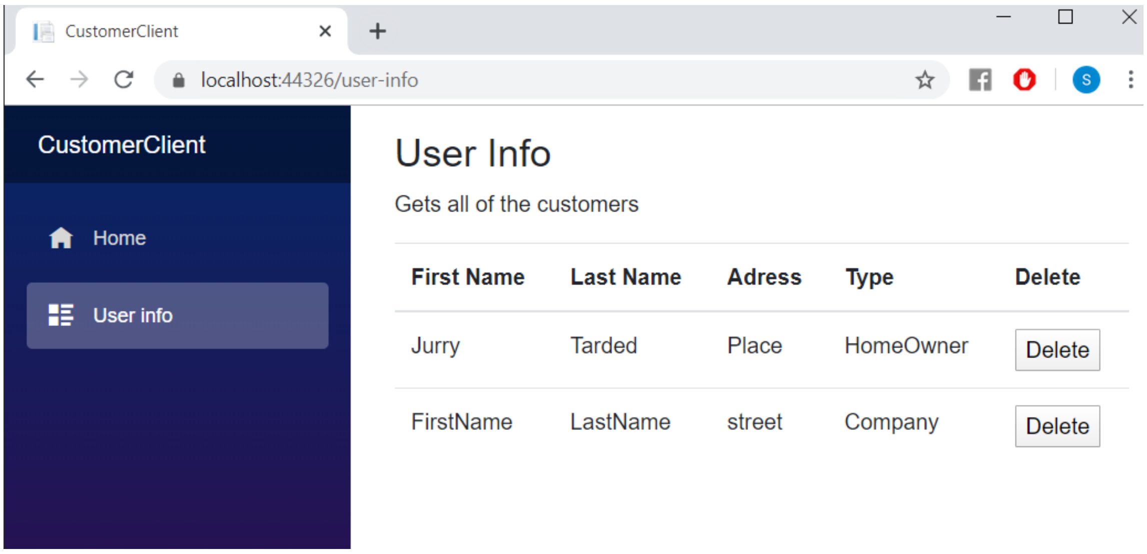


```
@foreach (Customer customer in customers)
{
<tr>

    <td>@customer.FirstName</td>
    <td>@customer.LastName</td>
    <td>@customer.Address</td>
    <td>@customer.type</td>
    <td><button @onclick="@(e => deleteCustomer(customer.Id))" >Delete</button></td>
    @{
        count++;
    }
</tr>
}
```

Image 42 Customer view write all customers

When the list with all of the customers is received it is printed to the page.

Also, in order to show that the tiers are connected we made a delete button, so the user can delete itself from the list.

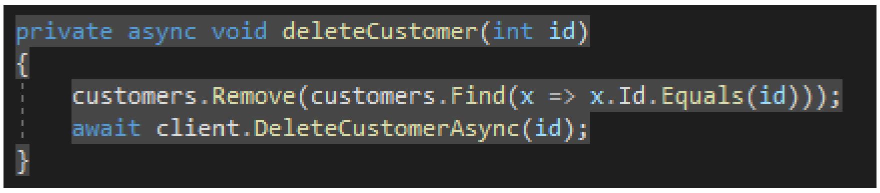


User Info

Gets all of the customers

First Name	Last Name	Adress	Type	Delete
Jurry	Tarded	Place	HomeOwner	Delete
FirstName	LastName	street	Company	Delete

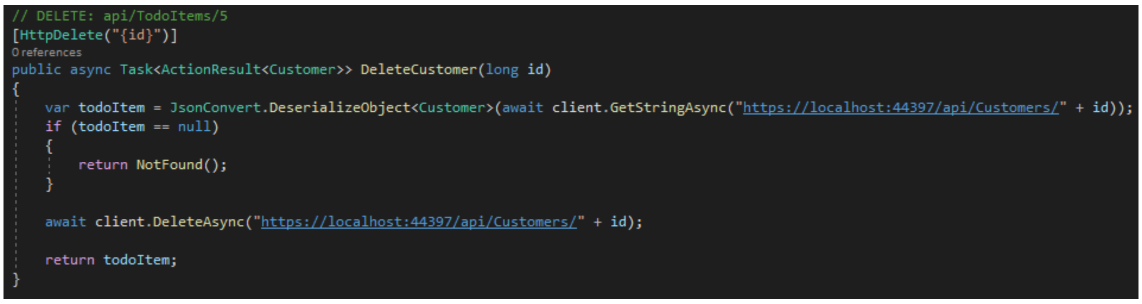
Image 43 Customer view - all customers



```
private async void deleteCustomer(int id)
{
    customers.Remove(customers.Find(x => x.Id.Equals(id)));
    await client.DeleteCustomerAsync(id);
}
```

Image 44 Delete customer

When the delete button is pressed the customer is removed from the list of customers and a delete request is sent to the web service.



```
// DELETE: api/TodoItems/5
[HttpDelete("{id}")]
0 references
public async Task<ActionResult<Customer>> DeleteCustomer(long id)
{
    var todoItem = JsonConvert.DeserializeObject<Customer>(await client.GetStringAsync("https://localhost:44397/api/Customers/" + id));
    if (todoItem == null)
    {
        return NotFound();
    }

    await client.DeleteAsync("https://localhost:44397/api/Customers/" + id);

    return todoItem;
}
```

Image 45 Tier 2 delete customer

Tier 2 web service sends another request to the tier 3 web service using HTTPClient.

5 Test

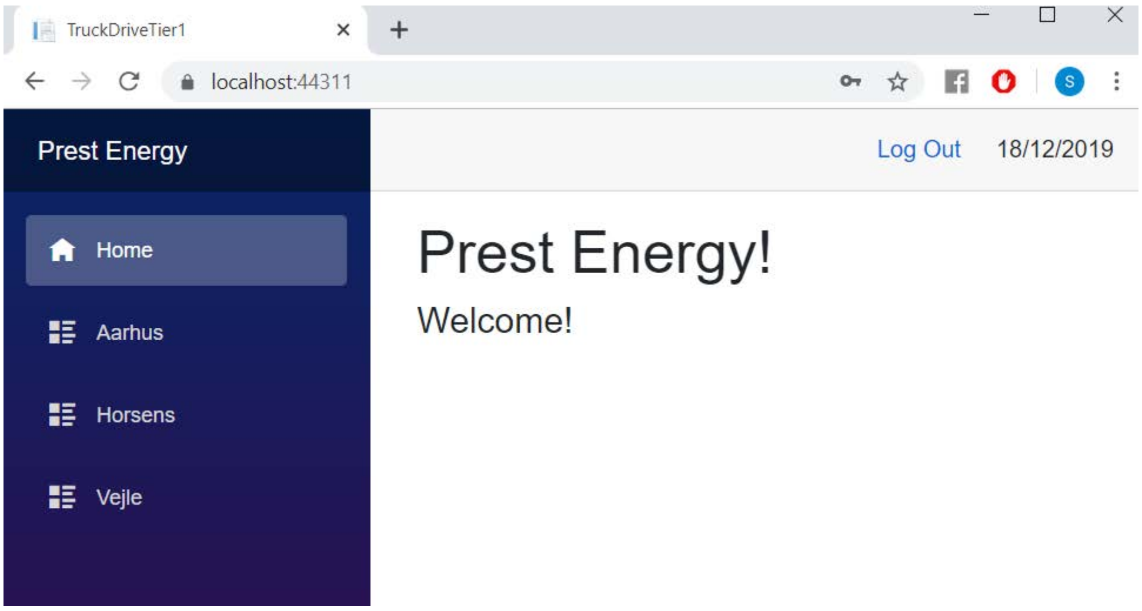
The tier 1 is tested with black box strategy (tested by checking the GUI), tier 2 is tested with Junit and tier 3 with black box (checking if the data matches the database).

In order to have access the user must be logged in. This is the example with the right credentials (that are in our database).

5.1 3 tier architecture

5.1.1 Tier 1

5.1.1.1 Driver blazor page



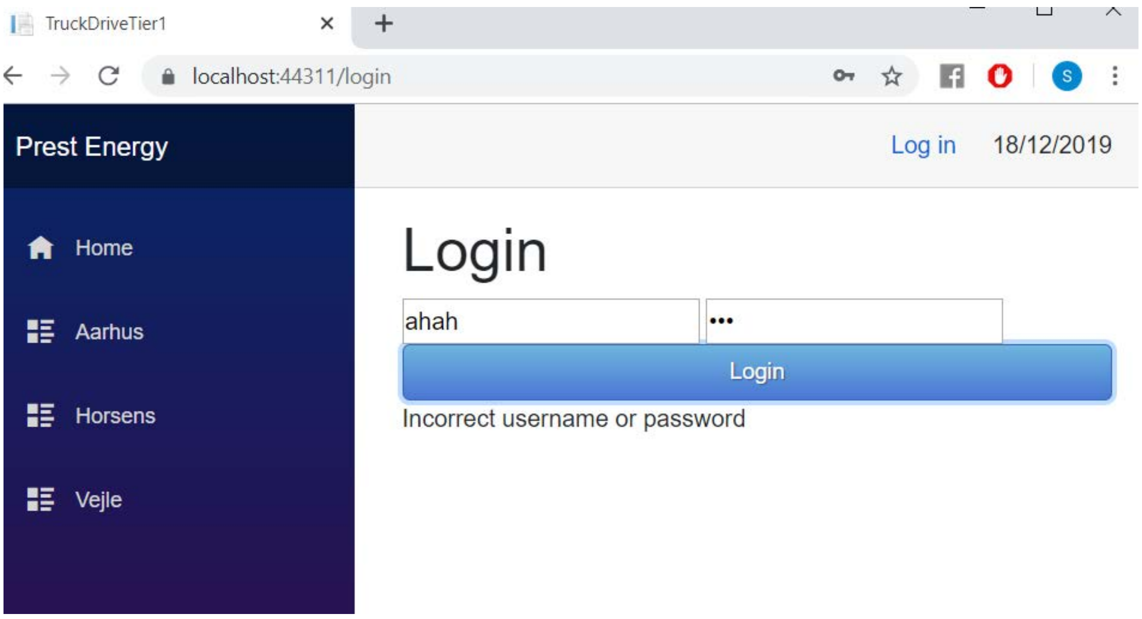
Prest Energy!

Welcome!

Image 46 Truck driver main view

After filing in with the correct credentials you get the access to the system.

This is an example with wrong credentials.

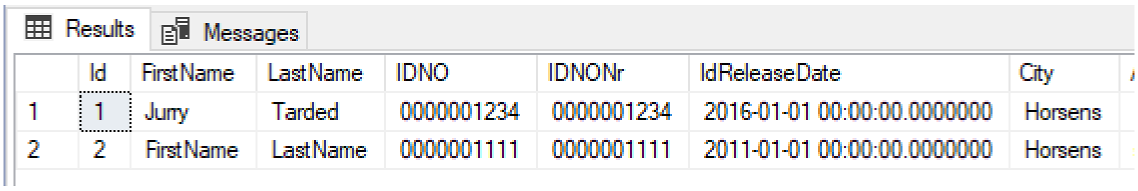


Incorrect username or password

Image 47 Truck driver login view

You get an advertisement “Incorrect username or password”.

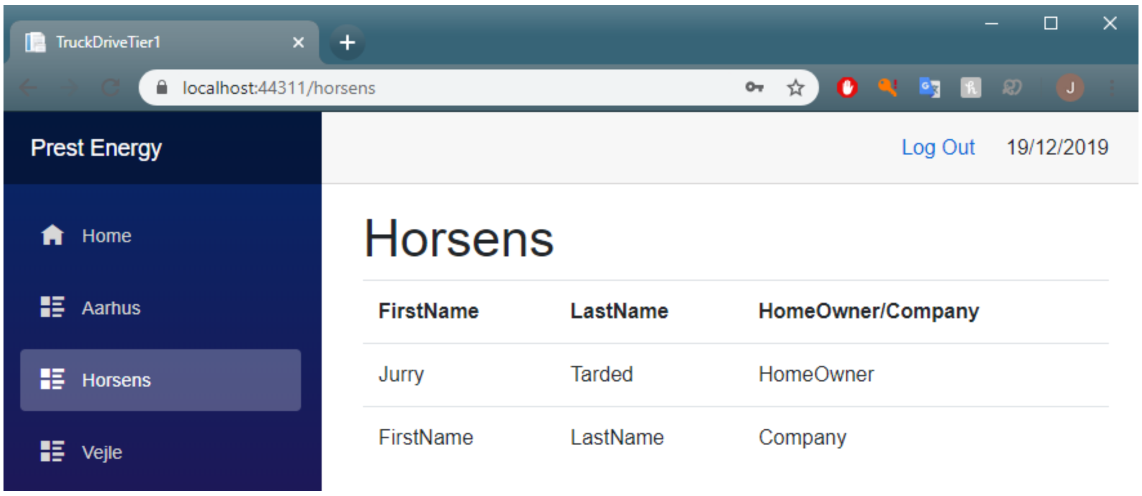
The initial data in the database:


Results
Messages

	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens

Image 48 Customer data in the database

The truck driver can also see all of the customers that are in different cities, for example:

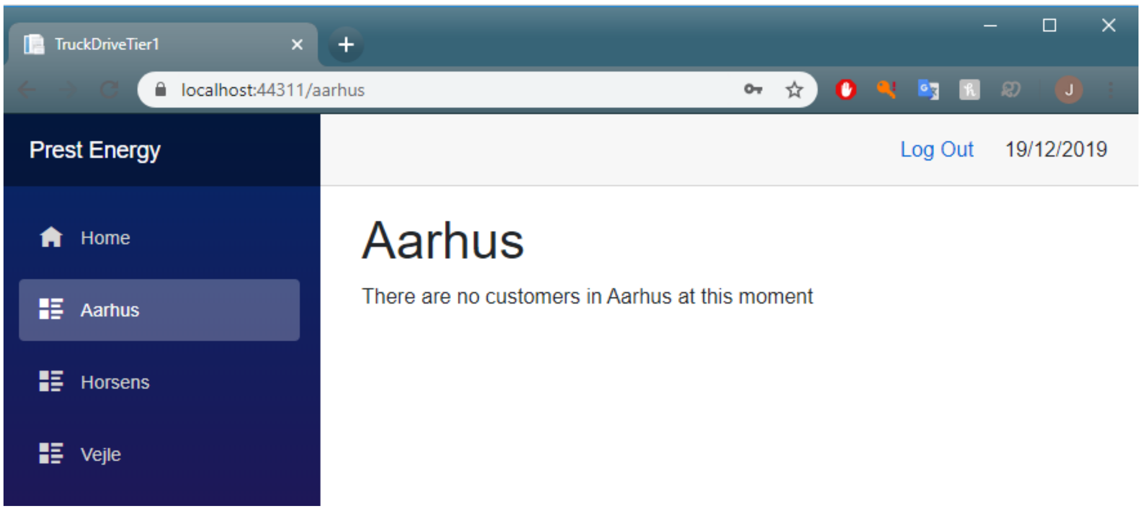


Horsens

FirstName	LastName	HomeOwner/Company
Jurry	Tarded	HomeOwner
FirstName	LastName	Company

Image 49 Truck driver Horsens view

This list shows all of the customers in Horsens.



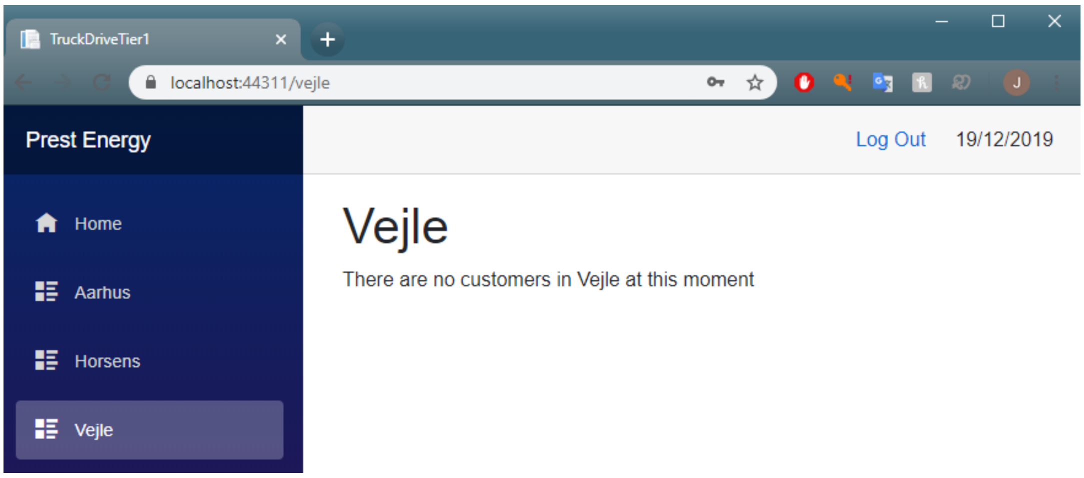
Aarhus

There are no customers in Aarhus at this moment

Image 50 Truck driver Aarhus view

If the user presses “Aarhus” in the Nav bar, then all of the customers in Aarhus will be shown.

Since there are no customers in Aarhus, the browser will show that there are no customers in Aarhus.



Vejle

There are no customers in Vejle at this moment

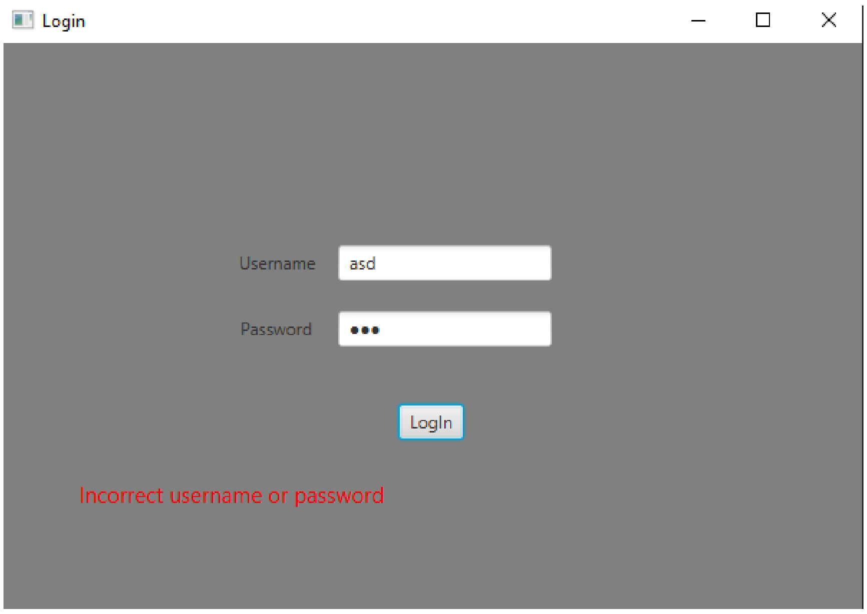
Image 51 Truck driver Vejle view

Vejle page is the same as the Horsens and Aarhus. There are no customers in Vejle, thus there is no list.

5.1.1.2 Administrator

If an administrator wants to use the system, then they need to login into the system first.

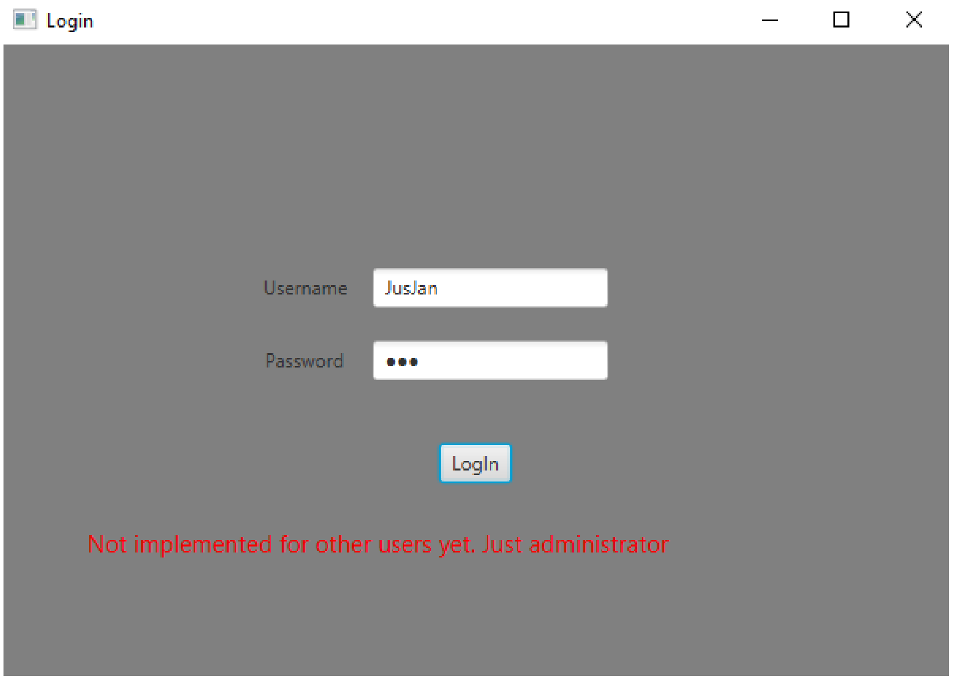
If the credentials are incorrect, the GUI shows an error



Incorrect username or password

Image 52 Administrator login

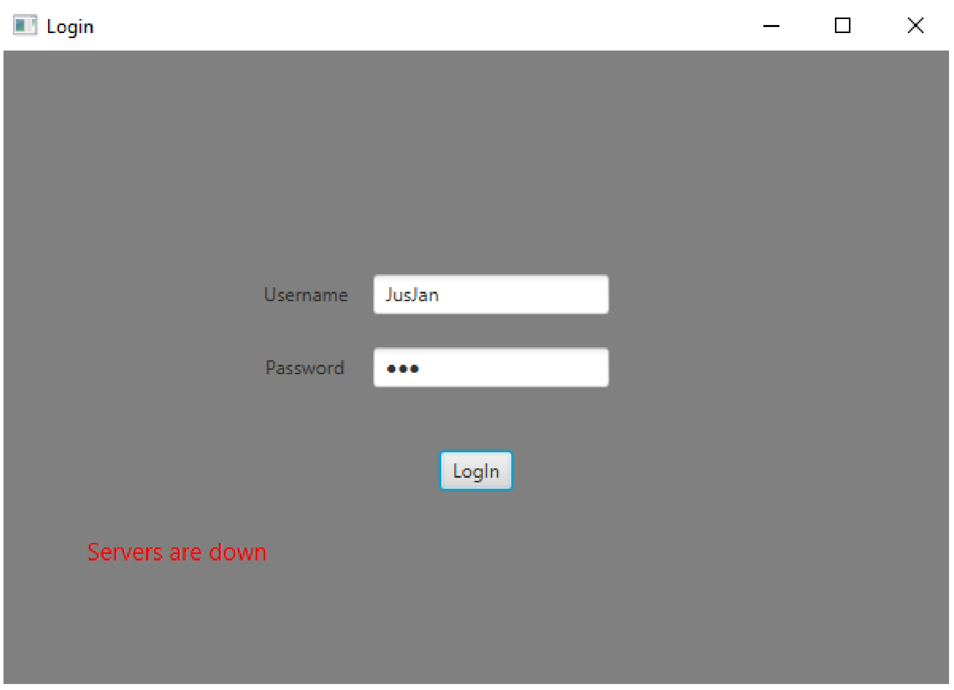
If the user who is trying to login is not an administrator, but some other employee, then the GUI shows that the other views are not finished and doesn't allow the user to login.



Not implemented for other users yet. Just administrator

Image 53 Administrator login

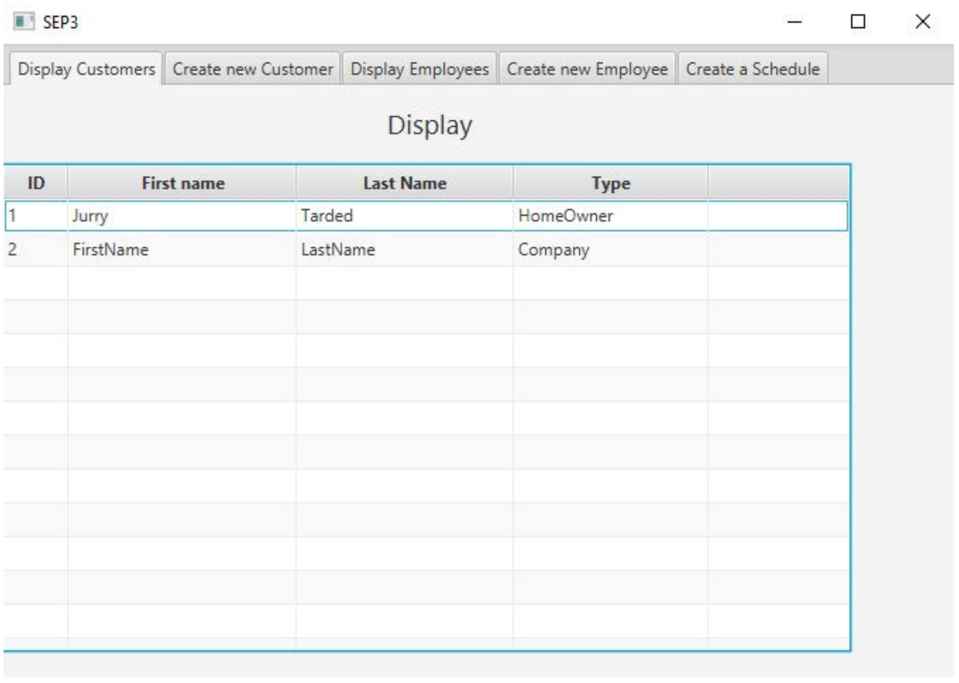
If the user is trying to login into the system while the Tier 2 server is not running, then the GUI will show that the T2 server is down



Servers are down

Image 54 Administrator view

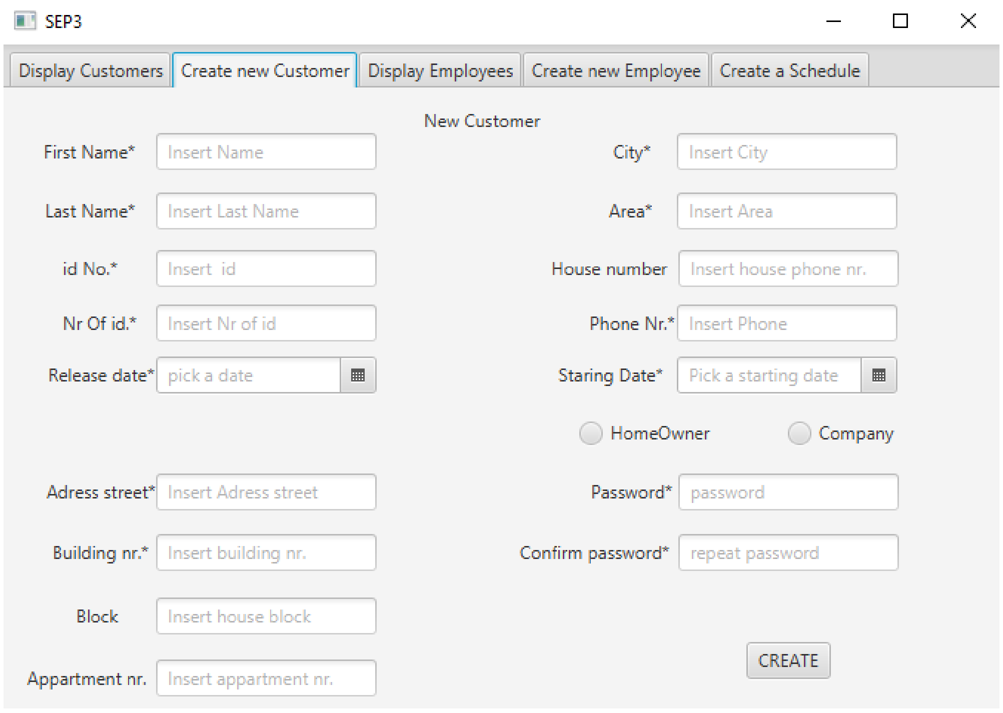
When the administrator inputs his correct credentials, then the GUI will proceed to the next view. The new view shows all of the customers that are in the system.



ID	First name	Last Name	Type
1	Jurry	Tarded	HomeOwner
2	FirstName	LastName	Company

Image 55 Administrator all customers

In the main view, there are additional tabs. One of the is to create a new customer.



SEP3

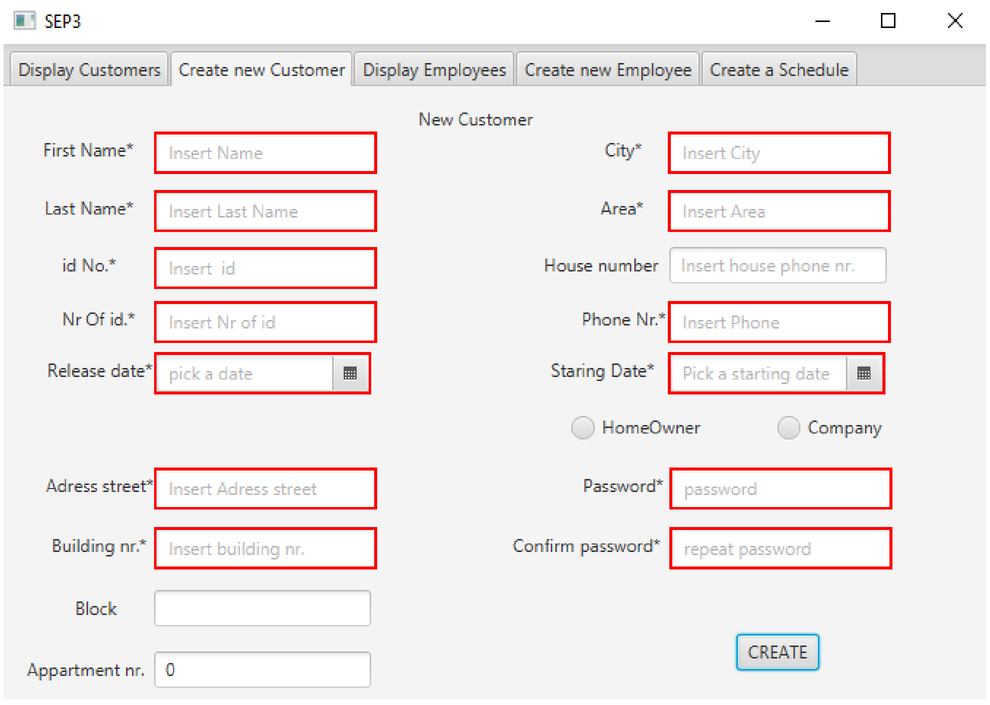
Display Customers | Create new Customer | Display Employees | Create new Employee | Create a Schedule

New Customer

First Name*	Insert Name	City*	Insert City
Last Name*	Insert Last Name	Area*	Insert Area
id No.*	Insert id	House number	Insert house phone nr.
Nr Of id.*	Insert Nr of id	Phone Nr.*	Insert Phone
Release date*	pick a date	Staring Date*	Pick a starting date
		HomeOwner	Company
Adress street*	Insert Adress street	Password*	password
Building nr.*	Insert building nr.	Confirm password*	repeat password
Block	Insert house block		
Appartment nr.	Insert appartment nr.		CREATE

Image 56 Administrator create new customer

If the administrator tries to press the “Create” button without filling in the necessary text fields, then the view will highlight the required fields.


SEP3

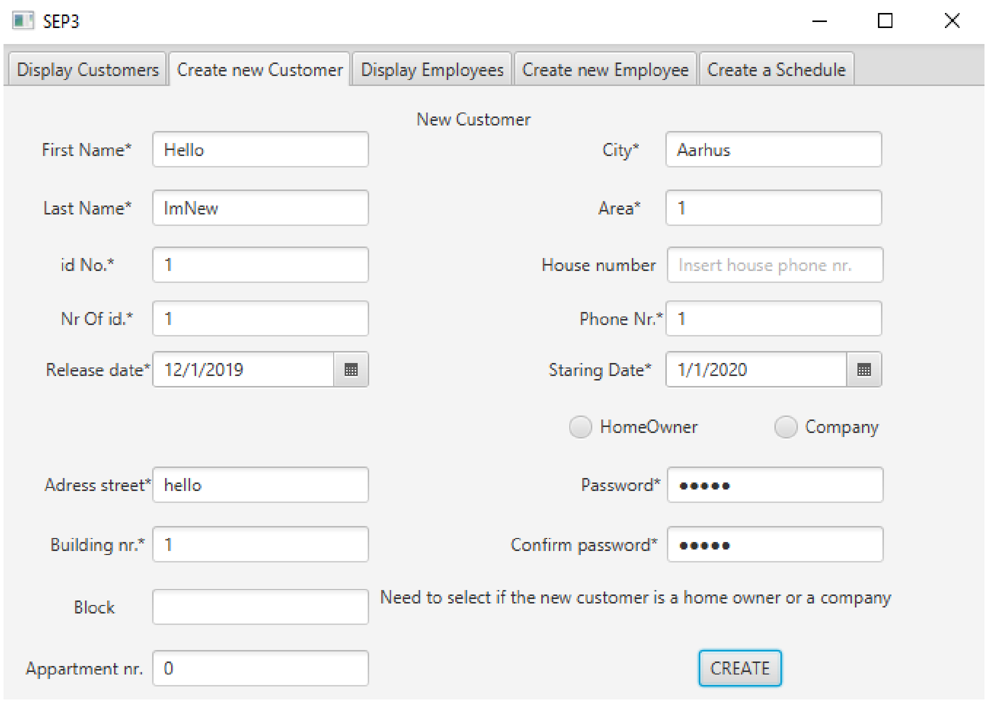
Display Customers | Create new Customer | Display Employees | Create new Employee | Create a Schedule

New Customer

First Name*	Insert Name	City*	Insert City
Last Name*	Insert Last Name	Area*	Insert Area
id No.*	Insert id	House number	Insert house phone nr.
Nr Of id.*	Insert Nr of id	Phone Nr.*	Insert Phone
Release date*	pick a date	Staring Date*	Pick a starting date
		HomeOwner	Company
Adress street*	Insert Adress street	Password*	password
Building nr.*	Insert building nr.	Confirm password*	repeat password
Block			
Appartment nr.	0		CREATE

Image 57 Administrator create new customer required fields

When the administrator fills out the necessary fields but doesn't select one of the radio buttons then the view will write to select a radio button and does not proceed.


SEP3

Display Customers | Create new Customer | Display Employees | Create new Employee | Create a Schedule

New Customer

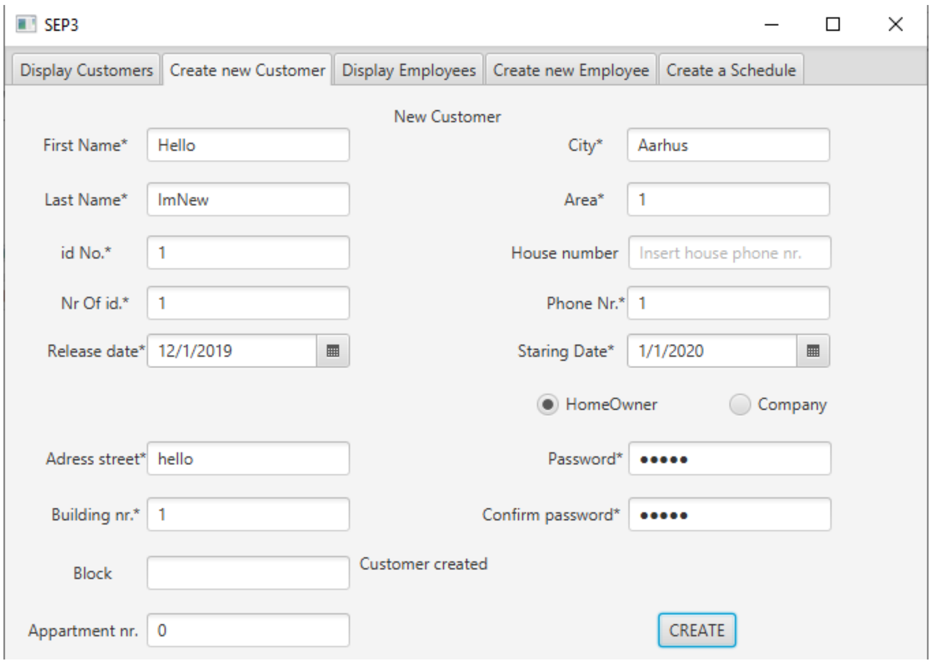
First Name*	Hello	City*	Aarhus
Last Name*	ImNew	Area*	1
id No.*	1	House number	Insert house phone nr.
Nr Of id.*	1	Phone Nr.*	1
Release date*	12/1/2019	Staring Date*	1/1/2020
		HomeOwner	Company
Adress street*	hello	Password*	●●●●●
Building nr.*	1	Confirm password*	●●●●●
Block			
Appartment nr.	0		

Need to select if the new customer is a home owner or a company

CREATE

Image 58 Administrator create new customer filled out

Only when the required text fields are filled out and a radio button is selected, only then the system adds a new customer to the system.


SEP3

Display Customers | Create new Customer | Display Employees | Create new Employee | Create a Schedule

New Customer

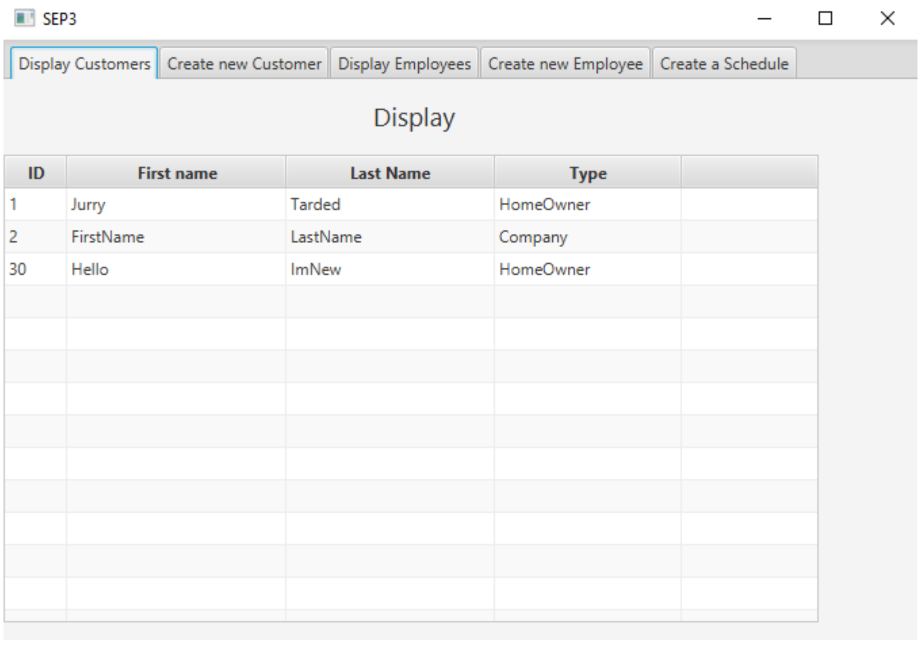
First Name*	Hello	City*	Aarhus
Last Name*	ImNew	Area*	1
id No.*	1	House number	Insert house phone nr.
Nr Of id.*	1	Phone Nr.*	1
Release date*	12/1/2019	Staring Date*	1/1/2020
		● HomeOwner	Company
Adress street*	hello	Password*	●●●●●
Building nr.*	1	Confirm password*	●●●●●
Block			
Appartment nr.	0		

Customer created

CREATE

Image 59 Administrator create new customer filled out and type selected

The results can be seen in the display customers tab.


SEP3

Display Customers | Create new Customer | Display Employees | Create new Employee | Create a Schedule

Display

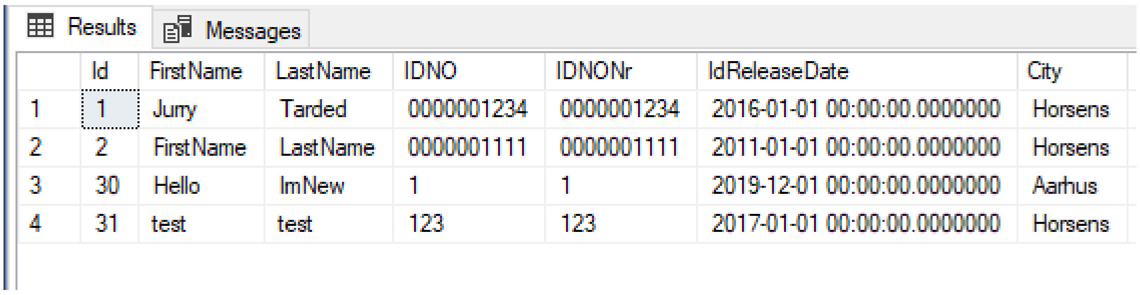
ID	First name	Last Name	Type
1	Jurry	Tarded	HomeOwner
2	FirstName	LastName	Company
30	Hello	ImNew	HomeOwner

Image 60 Updated administrator display all customers

5.1.1.3 Customer view

Even though the customer view is not fully implemented, but we are able to show that the tiers communicate.

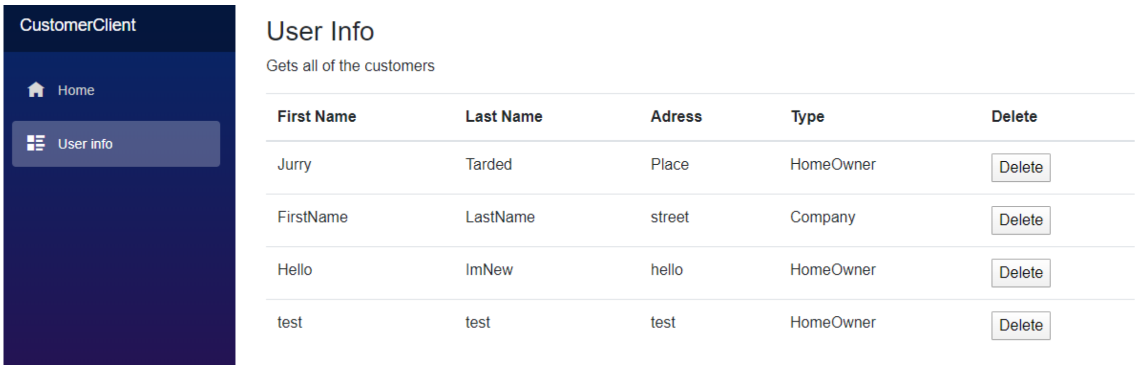
The initial data in the database while testing the view is:


Results | Messages

	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus
4	31	test	test	123	123	2017-01-01 00:00:00.0000000	Horsens

Image 61 Customer database

When the user opens the customer website and in the navigation bar opens the “User info”, all of the customers are shown.



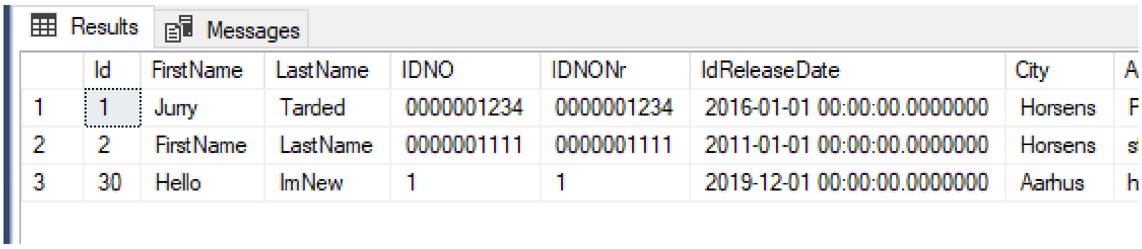
User Info

Gets all of the customers

First Name	Last Name	Adress	Type	Delete
Jurry	Tarded	Place	HomeOwner	Delete
FirstName	LastName	street	Company	Delete
Hello	ImNew	hello	HomeOwner	Delete
test	test	test	HomeOwner	Delete

Image 62 Customer view

The customer should be able to terminate the contract, so it is possible to delete a customer from the database. But because the login is not implemented anyone that enters this website is able to delete any customer.



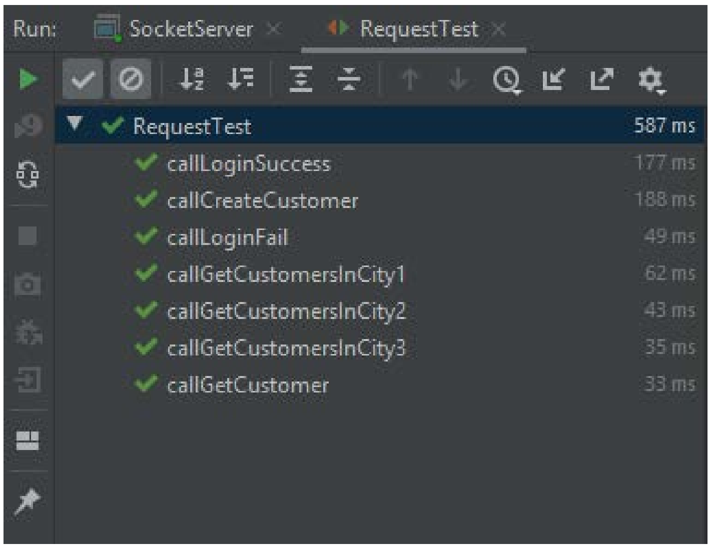
	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City	A
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens	F
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	s
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus	h

Image 63 Updated customer database

5.1.2 Tier 2

Tier 2 is tested with Junit4. All of the possible data request are tested.

5.1.2.1 Java tier 2 server

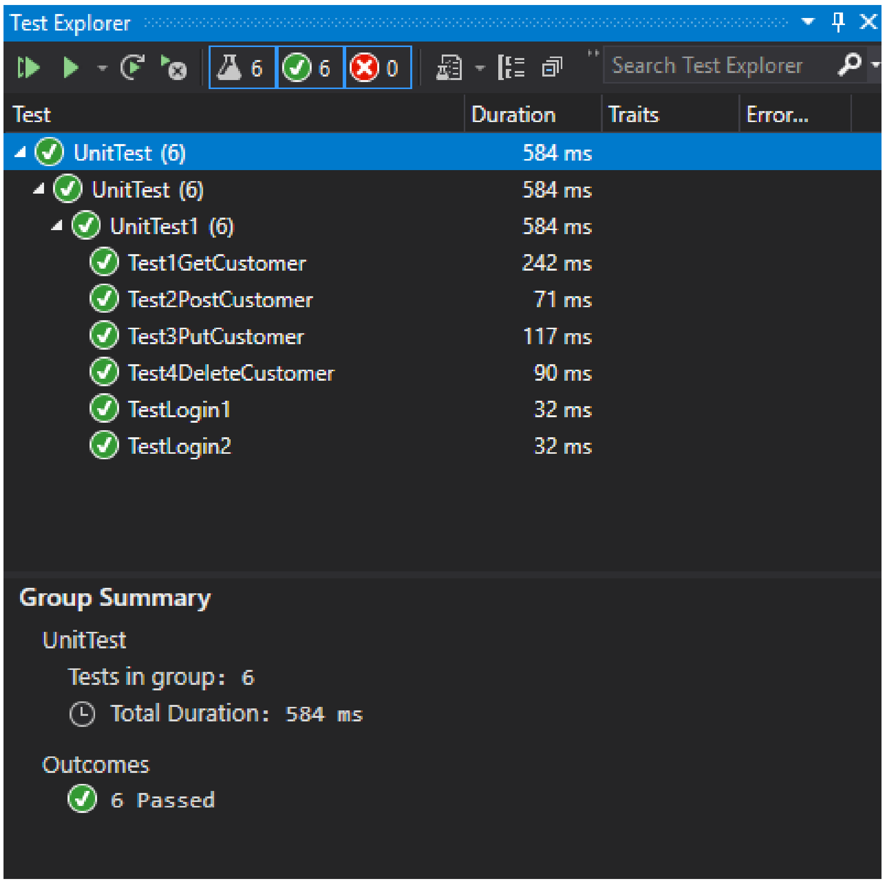


```
RequestTest                    587 ms
  callLoginSuccess             177 ms
  callCreateCustomer           188 ms
  callLoginFail                 49 ms
  callGetCustomersInCity1       62 ms
  callGetCustomersInCity2       43 ms
  callGetCustomersInCity3       35 ms
  callGetCustomer               33 ms
```

Image 64 Java tier 2 JUnit testing

All of the possible request that the tier 2 server could receive are tested.

5.1.2.2 C# tier 2 server


Test Explorer

Test	Duration	Traits	Error...
✓ UnitTest (6)	584 ms		
✓ UnitTest (6)	584 ms		
✓ UnitTest1 (6)	584 ms		
✓ Test1GetCustomer	242 ms		
✓ Test2PostCustomer	71 ms		
✓ Test3PutCustomer	117 ms		
✓ Test4DeleteCustomer	90 ms		
✓ TestLogin1	32 ms		
✓ TestLogin2	32 ms		

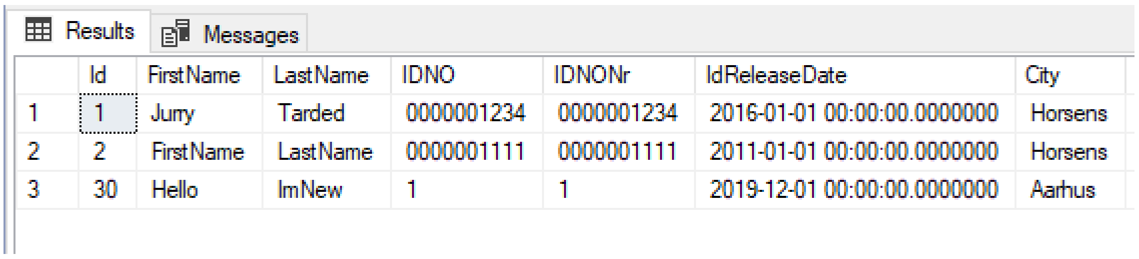
Group Summary
UnitTest
Tests in group: 6
Total Duration: 584 ms
Outcomes
✓ 6 Passed

Image 65 C# Tier 2 JUnit testing

All of the possible request that the server could receive are tested.

5.1.3 Tier 3

Most of the controllers in the tier 3 web service server are identical(They do the same functionality but return different data), for that reason only one of the controllers will be tested.
The starting data in the database is:



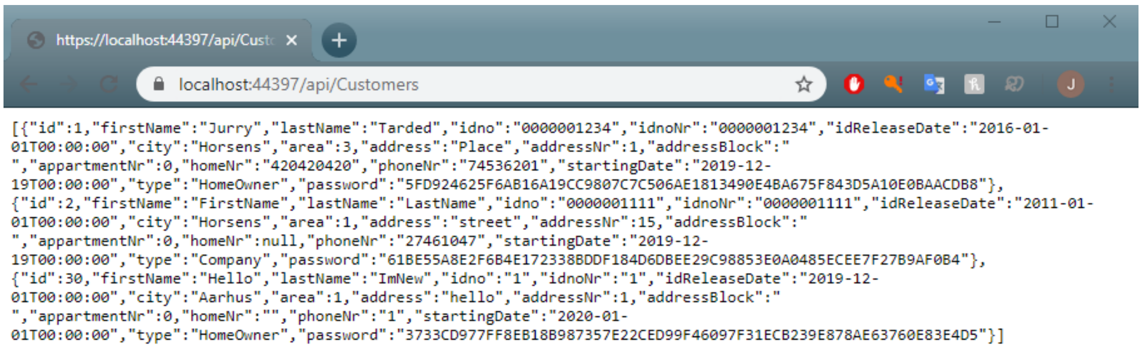
Results | Messages

	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus

Image 66 Customer database

5.1.3.1 GET

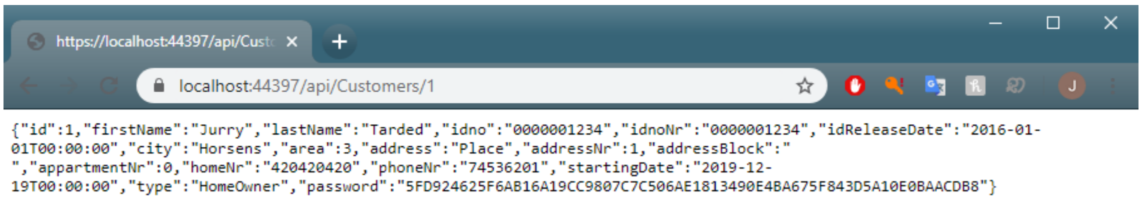
There are 2 get methods: get all of the customers and get 1 customer with a specific id.



https://localhost:44397/api/Custo
localhost:44397/api/Customers

```
[{"id":1,"firstName":"Jurry","lastName":"Tarded","idno":"0000001234","idnoNr":"0000001234","idReleaseDate":"2016-01-
01T00:00:00","city":"Horsens","area":3,"address":"Place","addressNr":1,"addressBlock":"
","appartmentNr":0,"homeNr":"420420420","phoneNr":"74536201","startingDate":"2019-12-
19T00:00:00","type":"HomeOwner","password":"5FD924625F6AB16A19CC9807C7C506AE1813490E4BA675F843D5A10E0BAACDB8"},
{"id":2,"firstName":"FirstName","lastName":"LastName","idno":"0000001111","idnoNr":"0000001111","idReleaseDate":"2011-01-
01T00:00:00","city":"Horsens","area":1,"address":"street","addressNr":15,"addressBlock":"
","appartmentNr":0,"homeNr":null,"phoneNr":"27461047","startingDate":"2019-12-
19T00:00:00","type":"Company","password":"61BE55A8E2F6B4E172338BDDF184D6DBEE29C98853E0A0485ECEE7F27B9AF0B4"},
{"id":30,"firstName":"Hello","lastName":"ImNew","idno":"1","idnoNr":"1","idReleaseDate":"2019-12-
01T00:00:00","city":"Aarhus","area":1,"address":"hello","addressNr":1,"addressBlock":"
","appartmentNr":0,"homeNr":"","phoneNr":"1","startingDate":"2020-01-
01T00:00:00","type":"HomeOwner","password":"3733CD977FF8EB18B987357E22CED99F46097F31ECB239E878AE63760E83E4D5"}]
```

Image 67 Tier 3 web service get all customers



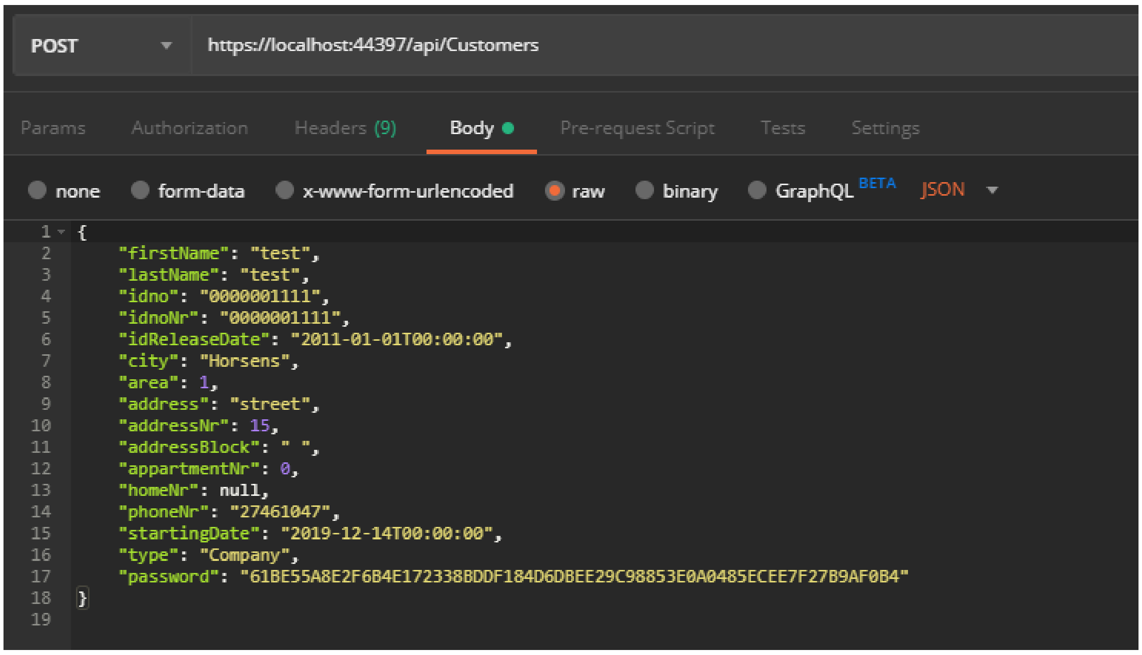
https://localhost:44397/api/Custo
localhost:44397/api/Customers/1

```
{"id":1,"firstName":"Jurry","lastName":"Tarded","idno":"0000001234","idnoNr":"0000001234","idReleaseDate":"2016-01-
01T00:00:00","city":"Horsens","area":3,"address":"Place","addressNr":1,"addressBlock":"
","appartmentNr":0,"homeNr":"420420420","phoneNr":"74536201","startingDate":"2019-12-
19T00:00:00","type":"HomeOwner","password":"5FD924625F6AB16A19CC9807C7C506AE1813490E4BA675F843D5A10E0BAACDB8"}
```

Image 68 Tier 3 web service get customer with id 1

5.1.3.2 POST

Postman program will be used to demonstrate that objects could be uploaded to the database.

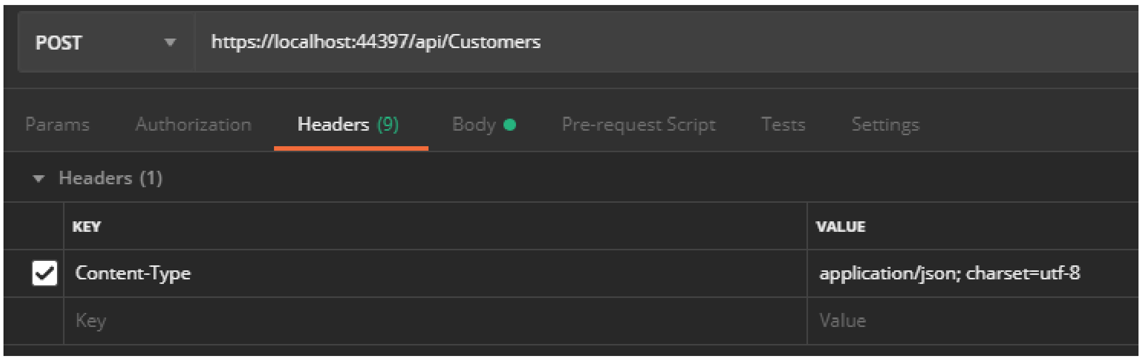


https://localhost:44397/api/Customers

```
{
    "firstName": "test",
    "lastName": "test",
    "idno": "0000001111",
    "idnoNr": "0000001111",
    "idReleaseDate": "2011-01-01T00:00:00",
    "city": "Horsens",
    "area": 1,
    "address": "street",
    "addressNr": 15,
    "addressBlock": " ",
    "appartmentNr": 0,
    "homeNr": null,
    "phoneNr": "27461047",
    "startingDate": "2019-12-14T00:00:00",
    "type": "Company",
    "password": "61BE55A8E2F6B4E172338BDDF184D6DBEE29C98853E0A0485ECEE7F27B9AF0B4"
}
```

Image 69 Postman POST to tier 3. Body

Input the link up top, select to POST and in the body fill in the json file to create.

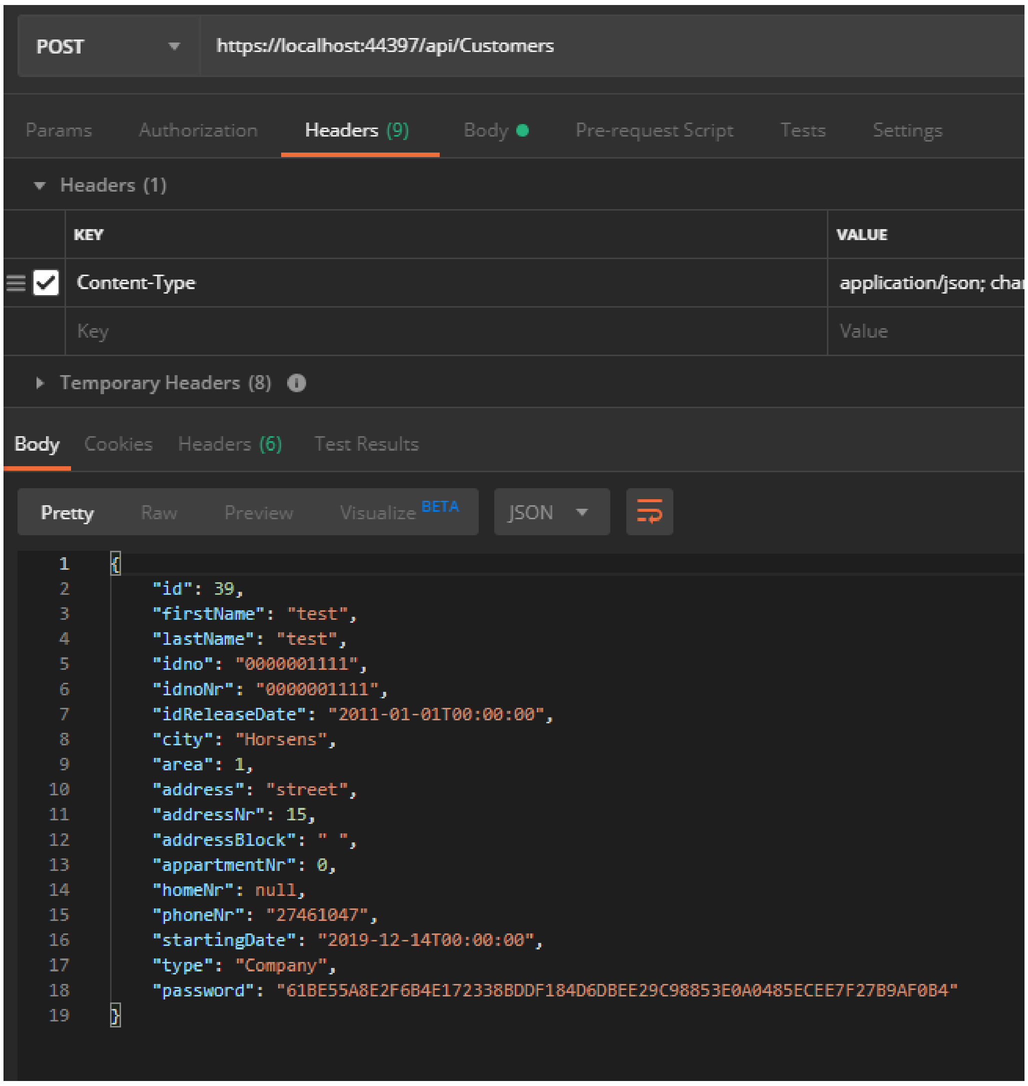


https://localhost:44397/api/Customers

	KEY	VALUE
✓	Content-Type	application/json; charset=utf-8
	Key	Value

Image 70 Postman POST to tier 3. Headers

Then add to the header that the content type is json and that the character set is UTF-8



POST https://localhost:44397/api/Customers

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers (1)

KEY	VALUE
Content-Type	application/json; cha
Key	Value

Temporary Headers (8)

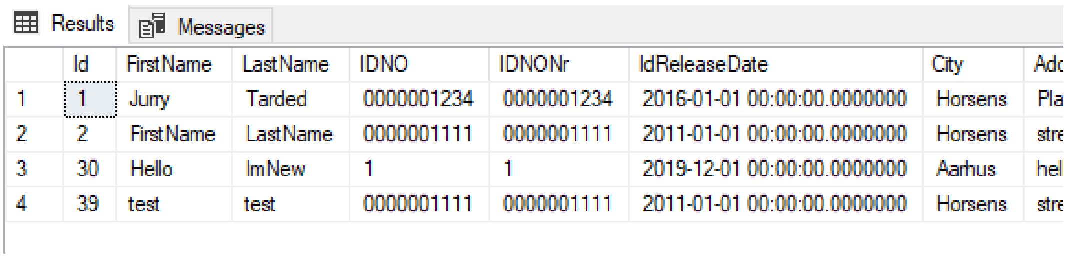
Body Cookies Headers (6) Test Results

Pretty Raw Preview Visualize BETA JSON

```
{
    "id": 39,
    "firstName": "test",
    "lastName": "test",
    "idno": "0000001111",
    "idnoNr": "0000001111",
    "idReleaseDate": "2011-01-01T00:00:00",
    "city": "Horsens",
    "area": 1,
    "address": "street",
    "addressNr": 15,
    "addressBlock": " ",
    "appartmentNr": 0,
    "homeNr": null,
    "phoneNr": "27461047",
    "startingDate": "2019-12-14T00:00:00",
    "type": "Company",
    "password": "61BE55A8E2F6B4E172338BDDF184D6DBEE29C98853E0A0485ECEE7F27B9AF0B4"
}
```

Image 71 Postman POST result

When we send the json file to the web service it returns the created object.



Results Messages

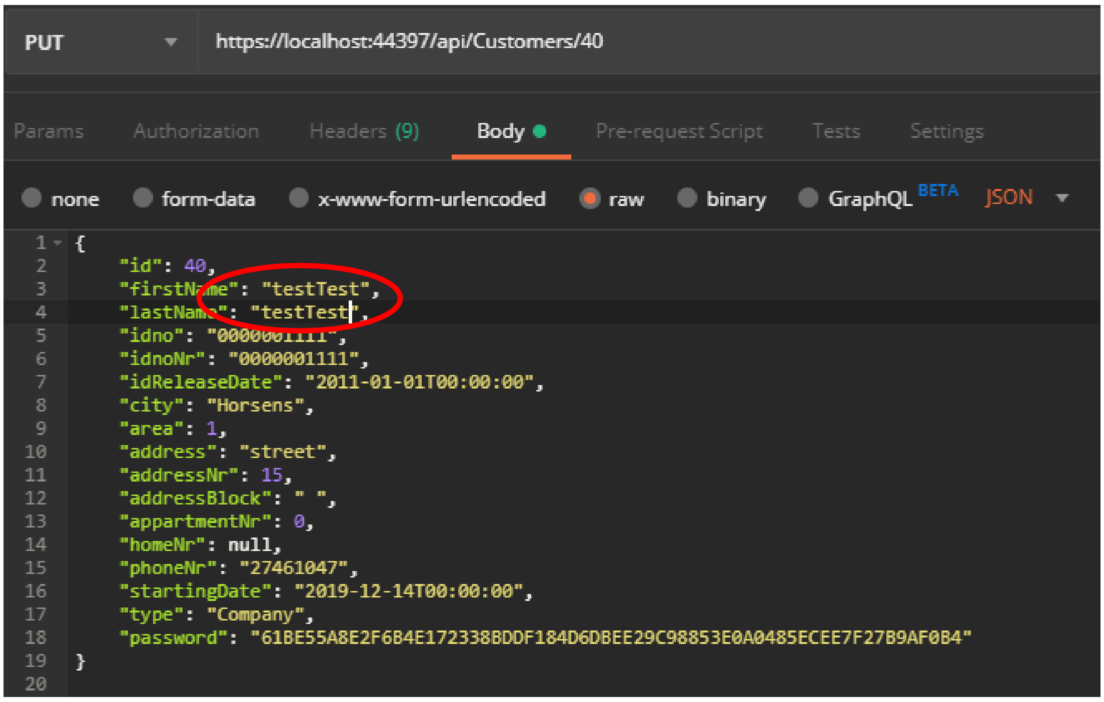
	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City	Adc
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens	Pla
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	stre
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus	hel
4	39	test	test	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	stre

Image 72 Updated customer database

And the database is updated.

5.1.3.3 PUT

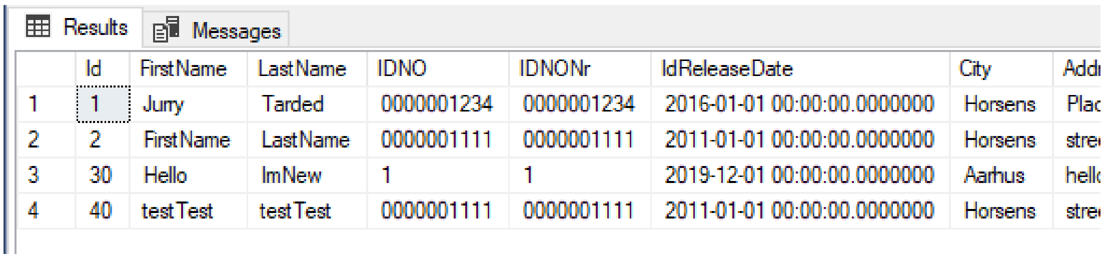
To demonstrate the PUT functionality Postman is also used.



```
{
    "id": 40,
    "firstName": "testTest",
    "lastName": "testTest",
    "idno": "0000001111",
    "idnoNr": "0000001111",
    "idReleaseDate": "2011-01-01T00:00:00",
    "city": "Horsens",
    "area": 1,
    "address": "street",
    "addressNr": 15,
    "addressBlock": " ",
    "appartmentNr": 0,
    "homeNr": null,
    "phoneNr": "27461047",
    "startingDate": "2019-12-14T00:00:00",
    "type": "Company",
    "password": "61BE55A8E2F6B4E172338BDDF184D6DBEE29C98853E0A0485ECEE7F27B9AF0B4"
}
```

Image 73 Postman PUT to tier 3. Change entry with id 40

Instead of the POST function, PUT is chosen. The link is updated to point to the Id of the element in the database and the json is also filled with the id of the same element. The header is not changed.



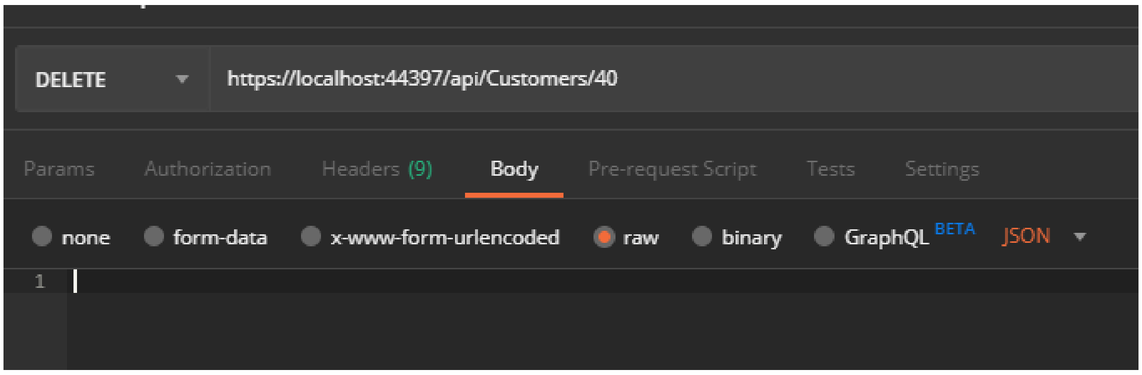
	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City	Addr
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens	Plac
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	stre
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus	hell
4	40	testTest	testTest	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	stre

Image 74 Updated customer database

After sending the request it changes the database content and edits the old content.

5.1.3.4 DELETE

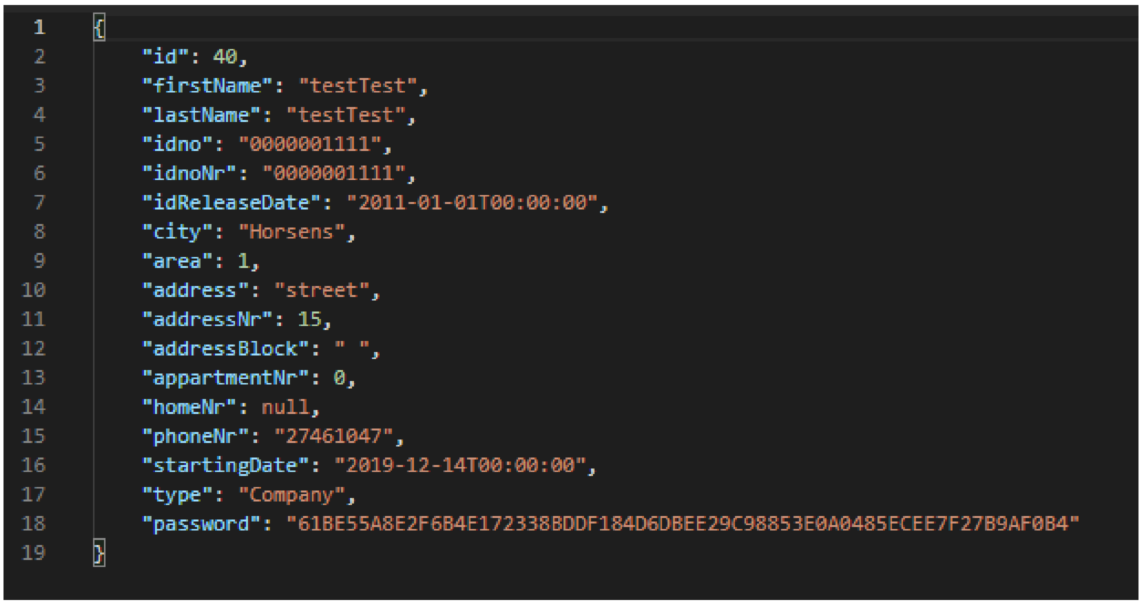
To delete a entry from the database using the web service the only thing needed is a link with the entry id in the route



```
DELETE    https://localhost:44397/api/Customers/40
```

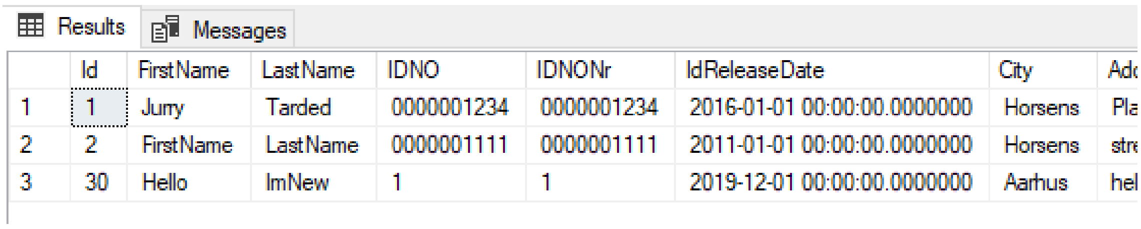
Image 75 Postman DELETE customer with id 40

After executing this code, the entry is removed from the database and sent to the user.



```
{
    "id": 40,
    "firstName": "testTest",
    "lastName": "testTest",
    "idno": "0000001111",
    "idnoNr": "0000001111",
    "idReleaseDate": "2011-01-01T00:00:00",
    "city": "Horsens",
    "area": 1,
    "address": "street",
    "addressNr": 15,
    "addressBlock": " ",
    "appartmentNr": 0,
    "homeNr": null,
    "phoneNr": "27461047",
    "startingDate": "2019-12-14T00:00:00",
    "type": "Company",
    "password": "61BE55A8E2F6B4E172338BDDF184D6DBEE29C98853E0A0485ECEE7F27B9AF0B4"
}
```

Image 76 Postman result



Results
Messages

	Id	FirstName	LastName	IDNO	IDNONr	IdReleaseDate	City	Ad
1	1	Jurry	Tarded	0000001234	0000001234	2016-01-01 00:00:00.0000000	Horsens	Pla
2	2	FirstName	LastName	0000001111	0000001111	2011-01-01 00:00:00.0000000	Horsens	stre
3	30	Hello	ImNew	1	1	2019-12-01 00:00:00.0000000	Aarhus	hel

Image 77 Updated customer database

5.2 Test Specifications

Works/Partially implemented/Doesn't work

Table 9 Test

Table 10

ID	Priority	Estimated. h.	Item	
1	Critical	25	As an administrator I want to be able to give access from the drivers to edit client's data for that specific day, so that the drivers could pick up trash and log it.	🟥
2	Low	5	As an administrator I want to upload contracts into the system in a pdf format, so that I can have access to all the customers who requested the company's services. (saved with the name of the customer and the number of contract).	🟥
3	High	10	As an administrator I want to be able to register new clients to the system	🟩
4	Medium	15	As an administrator I want to be able to register new employees and give different access for them.	🟨
5	High	20	As an administrator I want to be able to create a schedule for the drivers, so that the drivers know when to go to which area.	🟥
6	Medium	10	As an administrator I want to be able to view all clients that are registered to the system	🟩
7	High	13	As a casher I want to be able to register new clients to the system.	🟨
8	Medium	5	As a casher I want to be able to add money to their accounts. (no bank transactions) (log every transaction that the casher does)	🟥
9	Medium	19	As a casher I want to be able to undo the changes to customers account for 24 hours.	🟥
10	High	20	As a driver I want to be able to see which customer need they bins emptied.	🟥

11	High	20	Drivers should be able to log how many trash bags did they collect on each customer's account.	🟥
12	High	29	As a driver I want to be able to mark which customers bins have been emptied.	🟥
13	Low	10	As an accountant I want to print invoices for each client.	🟥
14	Low	15	As an accountant I want to be able to view all the clients that are registered to the system	🟥
15	Medium	17	As a customer I want to be able to view my balance in the account.	🟥
16	Low	20	As a customer I want to be able to download my invoice.	🟥
17	Medium	20	As a customer I want to be able to see when the collection day is.	🟥

6 Results and Discussion

The outcome of this project was to create a three-tier heterogeneous system with at least one web service and a custom socket protocol. The system was built with Java and C# programming languages. Security was also an essential part of the project. Even though the system's security was not implemented, multiple security threats were considered and explained above, and a couple of solutions were presented to counter the threats.

7 Conclusions

In conclusion, the project is not done, and a big part is left to finish. Even thought the security aspect of the project is not done, but a couple of threats are mentioned in the analysis part with some solutions to prevents those attacks in design. The project was created using the 3-tier architecture. Testing was done using white box testing (JUnit 4) and black box testing (GUI and Web Services).

8 Project future

For the project future, first, we want to have all the requirements fulfilled.

We will implement the security part which is not completely implemented in our system.

Since we have some errors, we need to fix them.

We need to redesign the GUI and make everything more interactive because now the system looks different for different users.

Sell it to the company so they can work.

9 Sources of information

https://www.allerin.com/blog/software-product-development [Accessed December 5, 2019]

Coulouris & Dollimore & Kindberg & Blair, Distributed Systems -- Concepts and Design, Fifth Edition, Addison-Wesley, 2012.

10 Appendices

All appendices are in “Appendices” file.

Appendices:

- Appendix A – User Guide
- Appendix B – UML Diagrams
- Appendix C – ER Diagram
- Appendix D – Conceptual Domain Model
- Appendix E – System Sequence Diagram
- Appendix F – Group Contract
- Appendix H – Use Case Diagram
- Appendix I – Project Description
- Appendix J – 3-Tier Architecture
- Appendix K – Source Code