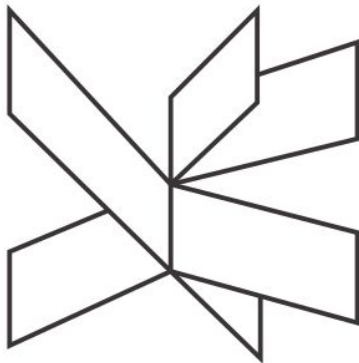


# Kashikoi Restaurant System

Project Report



# VIA University College

3. Semester, August 2018 - January 2019

Software Engineering

31400 characters

## Supervisors

- Jakob Knop Rasmussen
- Line Lindhardt Egsgaard
- Christian Flinker Sandbeck
- Jan Munch Pedersen
- Erland Ketil Larsen
- Ole Ildsgaard Hougaard

## Group 5

- Angel Petrov, 266489
- Kenneth Petersen, 269379
- Josipa Babic, 266757
- Remedios Pastor Molines, 266100

<b>Abstract</b>	<b>1</b>
<b>Introduction</b>	<b>2</b>
<b>Requirements</b>	<b>3</b>
Functional Requirements	3
Non-functional Requirements	3
<b>Analysis</b>	<b>4</b>
<b>Design</b>	<b>10</b>
<b>Implementation</b>	<b>19</b>
<b>Test</b>	<b>23</b>
Test Specifications	23
<b>Result and Discussion</b>	<b>27</b>
<b>Conclusions</b>	<b>28</b>
<b>Project Future</b>	<b>29</b>
<b>Source of information</b>	<b>31</b>
<b>Appendices</b>	<b>37</b>

## Abstract

The purpose of the project was to create a distributed application. The application helps restaurants to easily handle the orders and provides an online web application. The requirements for this semester project were to make use of a socket connection and the use of a 3-tier architecture. Two programming languages - Java and C# were mandatory to implement in the system. Workflow was managed with the use of Scrum. As a result of the methodologies that were used, the project matched the requirements. User Interface and web page allowed the user to interact with the system. This, utilizes all three tiers of the system.

## 1. Introduction

In today's integration of taking orders within a restaurant, customers will make orders to a waiter or at a bar counter and would wait for their order to be served. However, from here on out, the customers would face inconsistency in communication between them and the waiter about their order therefore, this leading to probable mistakes in the order, longer wait times and potential business profit losses. Having less control and information over their order, customers are more prone to feeling dissatisfaction, leading to less satisfaction on the restaurant's side and the experience of the customers. **(Restaurant Technology Guys, 2015).**

Field study has shown, that the hospitality industry can benefit financially from the use of self-service systems. In one study, the introduction of table-top self-service devices resulted in customer processing time being reduced by over 20 minutes when comparing only with customers using the system, and by 5 minutes when averaging the processing time of all customers, of which some did not use the system. While this shows significant cost reduction potential, it also indicates, that reluctance among customers to use the systems is a large inhibitor for this cost reduction. **(VirginiaTech, 2016)**

Online ordering, as mentioned in the first paragraph, is part of one of the system's client integrations. It is estimated that this implementation would be successful in terms of business profits, ease of accessibility and use. **(GloriaFood, 2017)**

## 1. Requirements

All functional and non-functional requirements represent the core architecture of how the system is supposed to be structured.

The system is built following the non-functional requirements, provided by the supervisors and functional the requirements decided upon by the project participants.

### 1.1. Functional Requirements

- 1) Staff should be able to view the menu
- 2) Staff should be able to modify the menu
- 3) Staff should be able to view the orders
- 4) Staff should be able to modify the status of the order
- 5) Staff should be able to change table status
- 6) Staff is able to change the availability of an item
- 7) Customer should be able to order items from a menu
- 8) Customer should be able to leave comments when placing an order
- 9) Customer should have the possibility to pay
- 10) Customer should be able to search for dishes
- 11) Customer should be able to filter the search results
- 12) Customer should be able to receive a receipt after payment has been done
- 13) Customer should be able to see the ingredients and description of the dish

### 2.2. Non-functional Requirements

- 1) System should be implemented in Java and C#.
- 2) System should be designed following the 3-tier architecture.
- 3) System should support multiple clients.
- 4) System should support client validation.
- 5) The main server and the web server should communicate over a socket protocol.
- 6) Communication between database and server should use RMI protocol.
- 7) Communication between server and clients should use the HTTPS protocol.
- 8) The main server should be a web-service.
- 9) Web service should be able to communicate with an end-user website representation.
- 10) System should have an uptime of at least 95%.

### 3. Analysis

#### 3.1. Use Case Diagram

In the analysis phase, a graphic study of the functionalities of the system was made in order to understand the performances depending on different scenarios. That is why a use case diagram was made. This diagram includes actions that the users can make in the system.

The system has two different types of users. Both types have their own actions but also they share the functionality “View menu”.

The staff members have the possibility of editing the menu, the orders or the status table whilst the customers can only interact with the menu and make an order.

Each use case is described in the use case descriptions.

Figure 1: Use Case Diagram

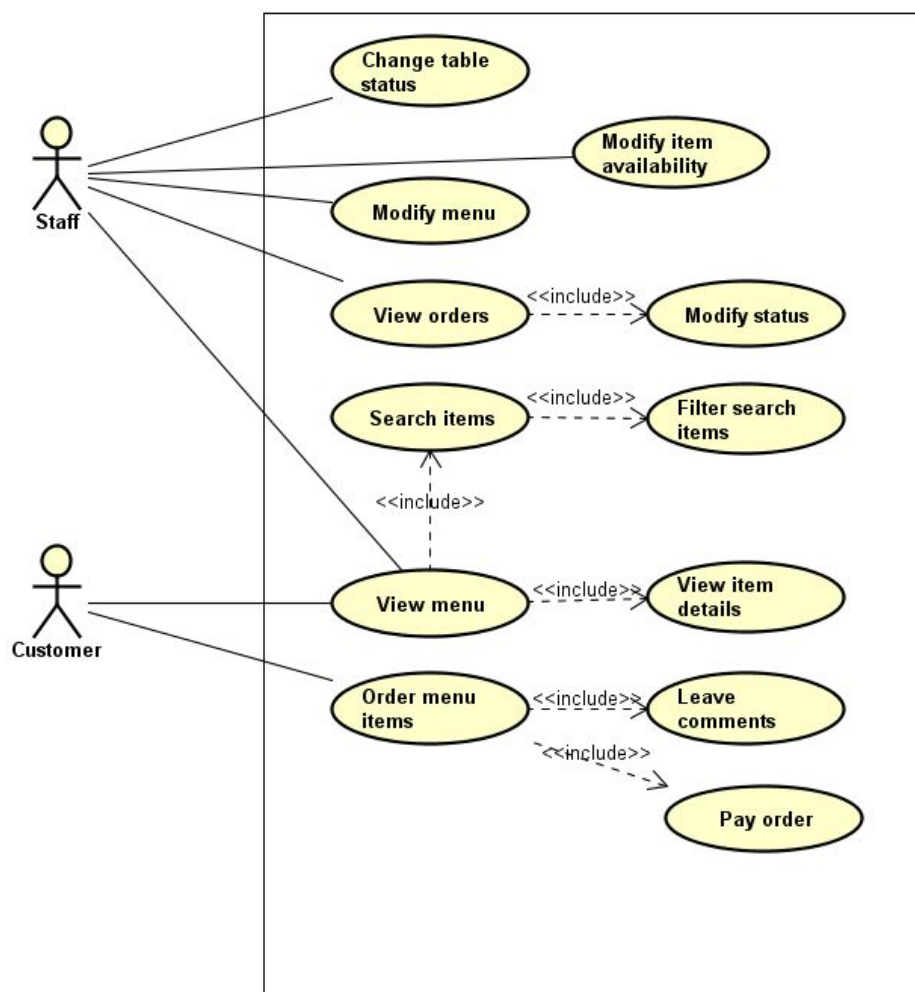
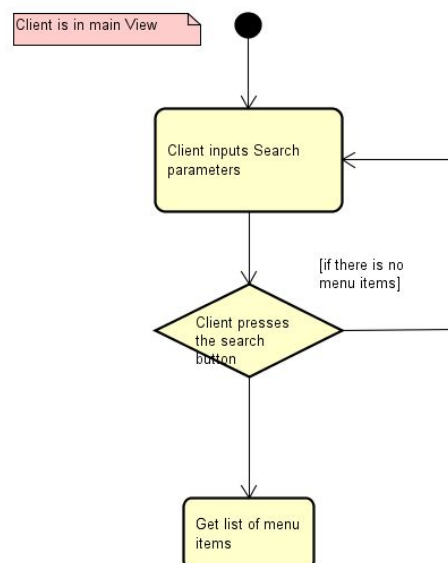


Figure 2: Use Case Description

ITEM	VALUE
UseCase	Order menu items
Summary	Choose items from the menu and make an order.
Actor	Customer
Precondition	Visualization of the menu.
Postcondition	
Base Sequence	1. Click on a specific item. 2. Choose the option "yes" when ask: for being added in the cart. When all the items of the order have been added to the cart: 3. Click on "Pay..." button. 4. Select the way of payment. 5. Click on "Pay..."
Branch Sequence	
Exception Sequence	
Sub UseCase	Leave comments Pay order
Note	

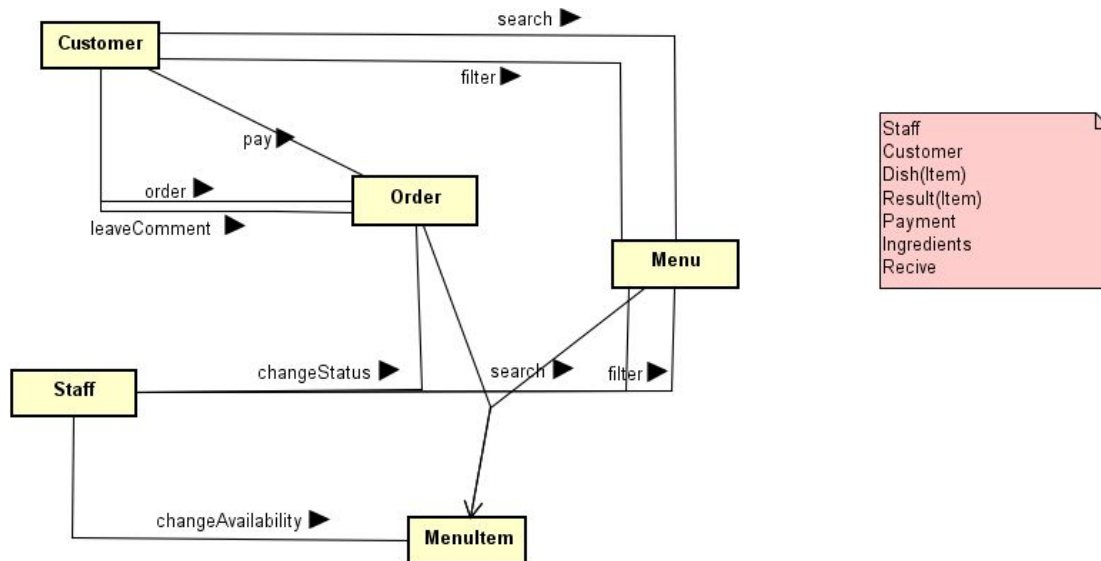
### 3.2. Activity Diagram

Figure 3: Activity Diagram



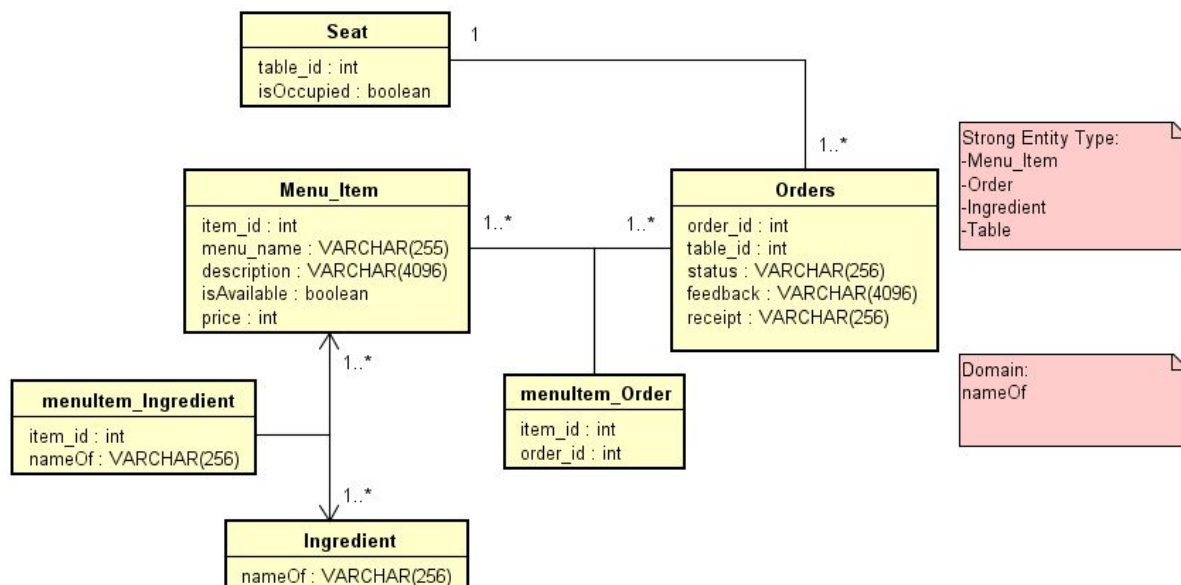
### 3.3. Model Class Diagram

Figure 4: Model Class Diagram



### 3.4. EER Diagram

Figure 5: EER Diagram



The main elements in the system are order, menu items and seat. To make an order, the customer will have to choose items from the menu. If the order is made at the restaurant,



then it will belong to a specific seat. The staff members will modify the menu by adding / removing items or by editing the availability of an item. Staff members can also update the availability of the seats and the status of the orders. Each of these three elements have an id. Order is dependent on the seat and on the menu items. Table and menu items are independent. Ingredients are dependent on menu items. If the orders are made remotely, then the system will not have the seat relation.

### 3.5 Security analysis

The table below represents security aspects that should be considered for the current system implementation. It shows the goal of the attacker, the way that the attack will be performed, what place of the system is the vulnerability going to take place, likelihood of it happening and how that affects the system and the party that is using it.

*Table 1: Threat & Risk Assessment Model*

№	Threat from third-party	Description - Attacker tries what?	Risk possibility likelihood	Impact
1	Get ahold of information	Get information that is only available to staff	Medium	Potential business losses
2	Gains access to staff section	Change order information	Medium	Order data is compromised
3	Gains access to staff section	Delete menu item information	Medium	Menu data is lost
4	Gains access to staff section	Add menu items	Medium	Potential business losses
5	Gets ahold of authentication token	Acquire token file	High	Attacker can perform various operations that are only to be done by the staff
6	Gets access to connection between business and data layer	Brute-force on unsecured socket connection	Low	Connection is breached
7	Gets access to connection between business and data layer	Man-in-the-middle on brute-forced socket connection	Low	Connection is altered
8	Gets access to the web service	Input correct URL of hosted web service and port and perform operations	Low	Access to hosted web server

9	DDoS performed on menu	Spam data requests to server	High	Server gets overloaded from too many requests
10	Gets access to database	Tries to establish an RMI connection to database	High	Database is compromised

- Threat № 1:  
An attacker gets ahold of information that they are not supposed to. Therefore, they will be able to read sensitive data in the system.
- Threats № 2, 3 and 4:  
The attacker has access to a portion of the system implementation (or is replicating it). Therefore, they are able to modify the information for that section. Repudiation is not part of these threats because the system does not handle unique identities of each user.
- Threat № 5: Unlike threats 2, 3 and 4, threat № 5 is not associated with tampering and information disclosure. The authentication token can be considered as a “master key” when communicating with the system.
- Threat № 6: This threat can provide access to sensitive information that is communicated between the data and business layer and in-between the servers that are only on the business layer.
- Threat № 7: The attacker is parsing on information from one end to the other whilst modifying it if they so desire.
- Threat № 8: An attacker is able to get and set information based on what action they perform on the web service.
- Threat № 9: The third-party performs multiple requests of various type to one or both of the servers.
- Threat № 10: The database is vulnerable when an attacker recreates the RMI connection and uses it as a tunnel to the database.

Table 2: Thread type classification based on STRIDE

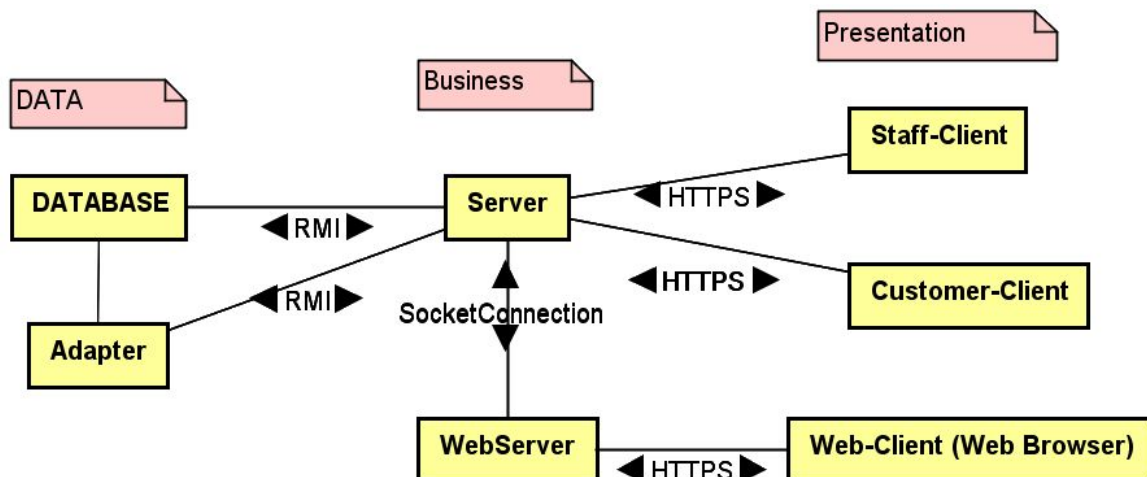
Threat №	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
1	✓			✓		
2	✓	✓		✓		✓
3	✓	✓		✓		✓
4	✓	✓		✓		✓
5	✓					✓
6				✓		
7		✓	✓	✓		
8				✓		✓
9					✓	
10	✓			✓		

## 4. Design

### 4.1 3-Tier Architecture

The system is designed following the 3-tier architecture. Its purpose is to enable replacement or relocation of each part of the system whilst not affecting other parts of the system. The data access layer handles interaction with the actual database and builds the objects used on the business layer. Communication between the data access layer and the business layer is handled by Java RMI. The business layer and presentation layer communicate using HTTPS. The system is heterogeneous, two programming languages were used for implementation - Java and C#. Data layer uses SQL and is implemented and called from Java.

*Figure 6: 3-Tier Architecture diagram*

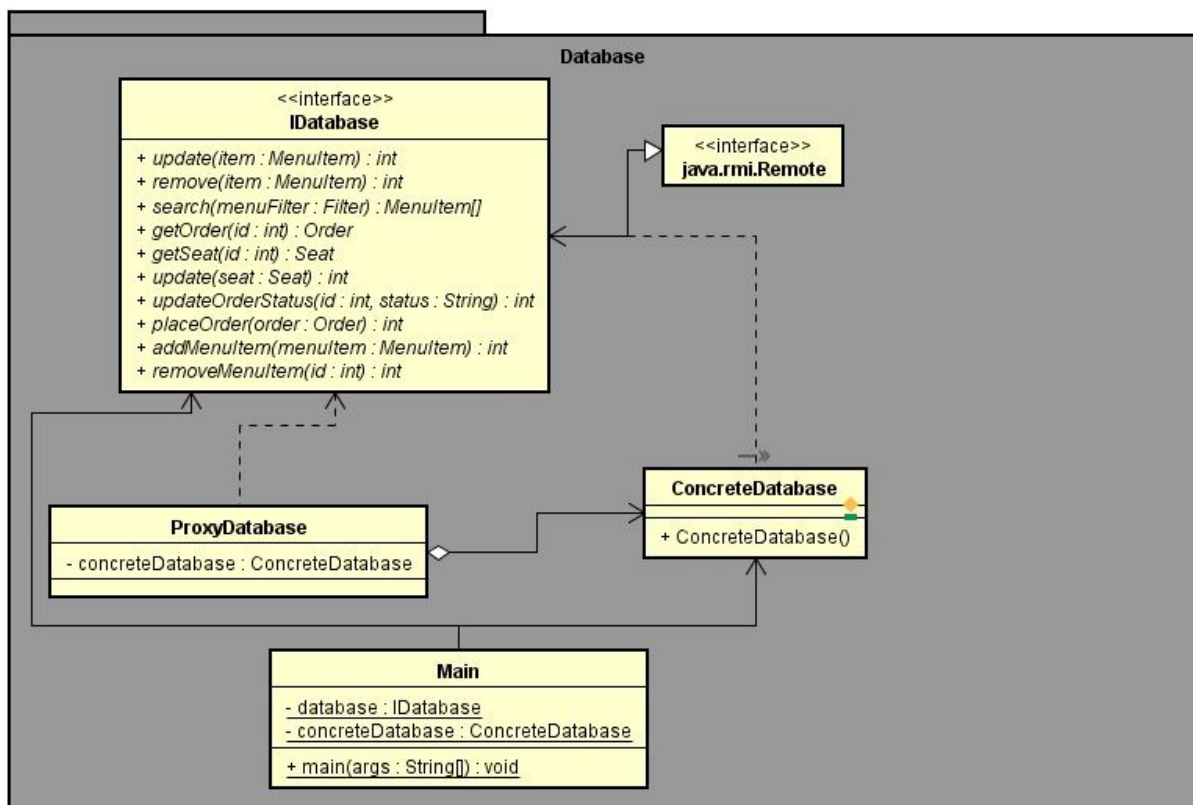


A socket connection is used on business layer for communication between the server and web server. This protocol allows for communication between different programming languages. Main server is programmed in Java and web server is programmed in C#. HTTPS is used between the business and presentation layer components which provides a secure communication. This protocol allows communication between business and presentation layer. Clients initiates request message and server sends a response to that request. There are two types of requests that the client is able to send. When making a GET request, the parameters are contained in the URI. When a PUT request is performed the identifying parameter is specified in the URI by the client and the values are in the body of the request. Client sends a request to the server and receives a response with an appropriate response code following his request.

## 4.2 Design Patterns

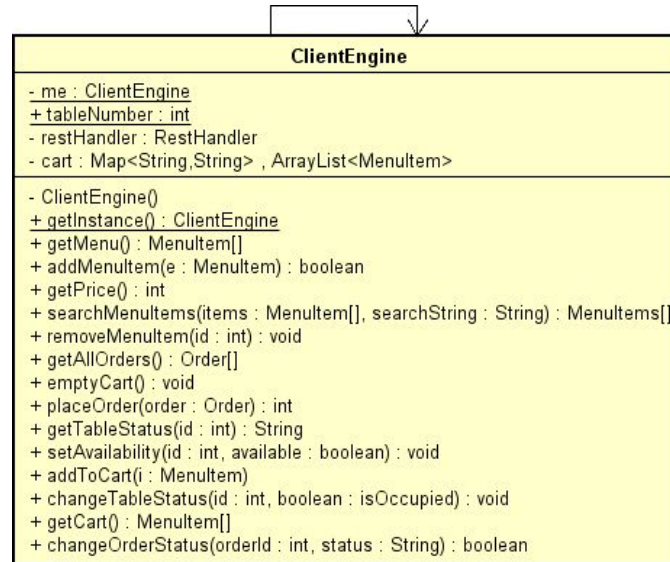
The design patterns that are used in this system are: Adapter and Singleton. Adapter design pattern is used on database side. Pattern provides interface of a class to be used as another interface. Using adapter design pattern avoids modifying source code. The AdapterDatabase class translate and redirect work to ConcreteDatabase class.

Figure 7: Database package



Singleton design pattern is used on Client side on a Class Client Engine. This class is responsible for handling functionality that are not in UI and getting and putting requests from and to server. This pattern provides one instance that can be used by many UI controllers.

Figure 8: ClientEngine class



### 4.3 UI Design choices

Both the customer and staff client use a graphical user interface for providing access to functionality. They were made using SceneBuilder and therefore, all UI elements are exported in an XML structured format. As a result of this an FXML loader is used for loading the XML file into the scene in line 13. Afterwards, on line 14, the controller that is responsible for all logic, performed by the UI elements, is executed.

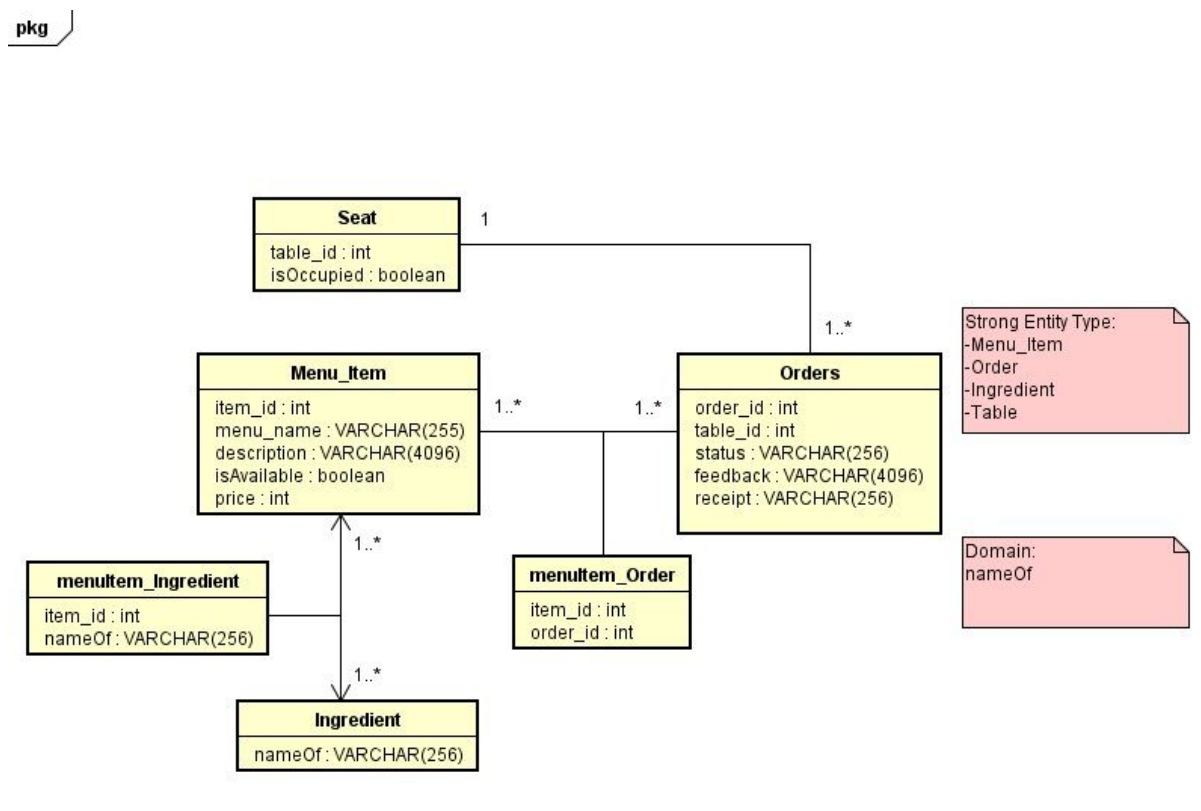
Figure 9: Snapshot of Client.UI.Customer.UIMain.java

```

10 public class UIMain extends Application {
11     @Override
12     public void start(Stage primaryStage) throws Exception {
13         FXMLLoader loader = new FXMLLoader(getClass().getResource("Table_prompt.fxml"));
14         loader.setController(new EnterTableNumberController(primaryStage));
15         Parent p = loader.load();
16         Scene s = new Scene(p);
17         primaryStage.setScene(s);
18         primaryStage.setResizable(false);
19         primaryStage.show();
20     }
21
22     public static void main(String[] args) {
23         Launch();
24     }
  
```

## 4.4 Database architecture

Figure 10: Snapshot of EER diagram



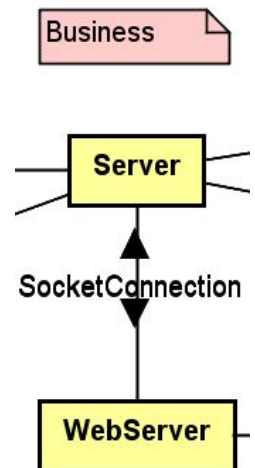
EER diagram represents the structure of database. Seat table has two attributes, table\_id is primary key. Relation between Seat and Orders is one to many, which means one Seat can have one to many Orders. Orders table contains five attributes where order\_id is primary key and table\_id is foreign key from table Seat. There is many to many relation between Orders and Menuitem, following the mapping steps relation table is created. Relation table has two attributes item\_id and order\_id. Menu\_Item table has five attributes and item\_id is primary key. Relation between Menu item and ingredient is many to many. As already mentioned following the mapping steps relation table is created. Table Ingredient has one attribute which is also primary key. Database is populated, tested and documented with EER diagram and proper documentation.

## 4.5 Web operations between server and web server

HTTPS TCP connection establishment between server and web server in business tier:

Figure 11: Performed operations and given parameters. Sent from C# to Java server.

ID	Commands	Param1	Response
1	GETMENUITEMS	None	Menu items - 1 per line; terminate with 0x00 (byte end)
2	SUBMITORDER	JSON OrderData	OrderID assigned by database



### 4.5.1 Description of web related operations

#### GETMENUITEMS:

The web server sends “GETMENUITEMS” and in turn, the server sends an array of menu items.

#### SUBMITORDER:

The web server sends a command “SUBMITORDER” and in turn, the server returns an OK response.

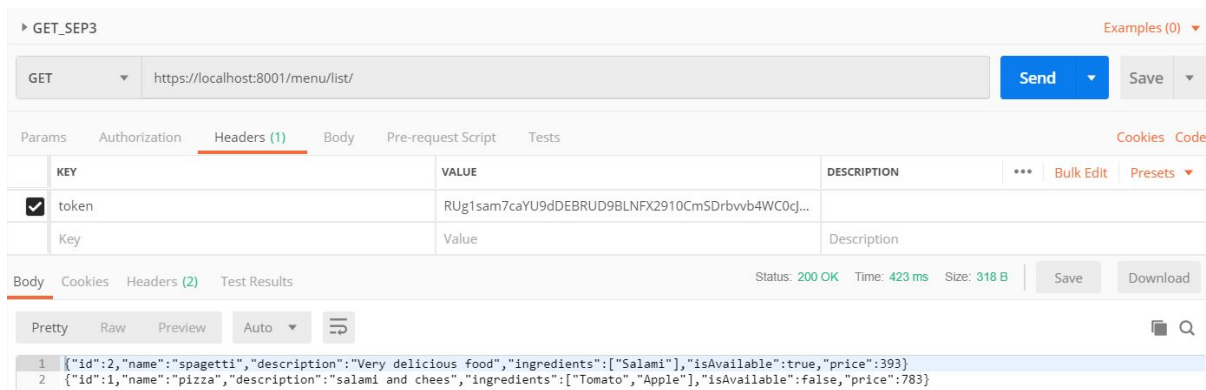


## 4.7 Security

### 4.7.1 Current security built into the system

The current system implementation does not handle any identity authentication of users on the client side in both customer and staff. The system cannot identify the users that are performing operations onto it. However, the staff side in the client package uses an authentication token which is used when performing GET and SET requests to the server. If a request is performed without this authentication token, the request will be denied with a status code 401. The token file must be placed in the root folder of the current user in order for it to be recognized by the client. If a user would like to perform requests without doing this, they would need to have the string token that can be generated by the server and put it as a header in the request that they will perform.

Figure 12: Snapshot of Postman performing a successful GET request using a generated string token



### 4.7.2 Security measures needed for preventing unauthorized access

A necessity for the current implementation of the system is the addition of a password authentication. This, in combination with the currently used way of making requests onto the system means that there can be two-factor authentication. Therefore, an attacker cannot brute-force into the system without having the token if they have managed to crack the password of a user. The web service is also protected this way as it uses the login session and the provided token for it to be accessed securely. This protection scheme can be used on threats № 1, 2, 3, 4 and 8 in the security analysis section.

The initial handshake, including the MAC (Message Authentication Code) and key exchange for the HTTPS encryption is already handled by the used HTTPSServer that is used in the current system implementation. This protection scheme can be used on threats № 1, 2, 3, 4 and 8 in the security analysis section.

In the database, the passwords can be stored using the SHA-256 hashing algorithm for a higher level of security in case the database is compromised. This protection scheme can be used on threat № 10 in the analysis section.

The current system is using a dummy certificate that is not signed by a real authority. To avoid this and add a layer of trust, the system can benefit from the use of a digitally signed signature. This protection scheme can be applied on threat № 8 in the security analysis section.

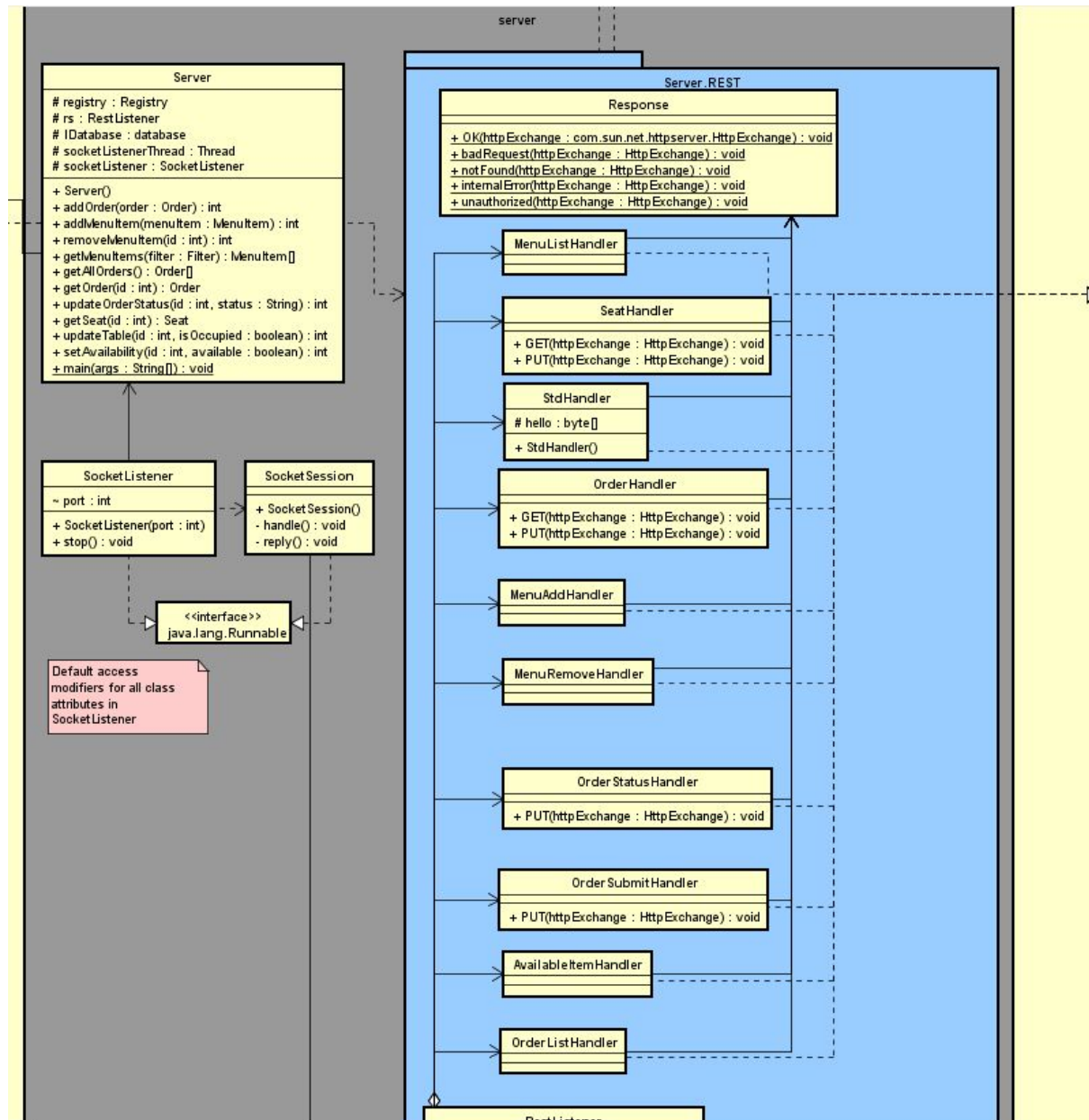
If the company that owns the system would like to separate the server and web server, then they would need to run the socket connection between the two through a VPN. This can also be applied for the RMI connection that is between the data layer and business layer. This protection scheme can be used on threat № 6, 7 in the security analysis section.

DDoS can be prevented by using a reverse proxy on the server in order for the reverse proxy to filter malicious requests. It can act as a server (firewall) by itself that filters and sends only filtered data to the main server. This protection scheme can be used on threat № 9 in the security analysis section.

Finally, for maximum security, the generated key store token file can also be encrypted, requiring a PIN code in order for its contents to be read. This protection scheme can be applied on threat № 5 in the security analysis section.

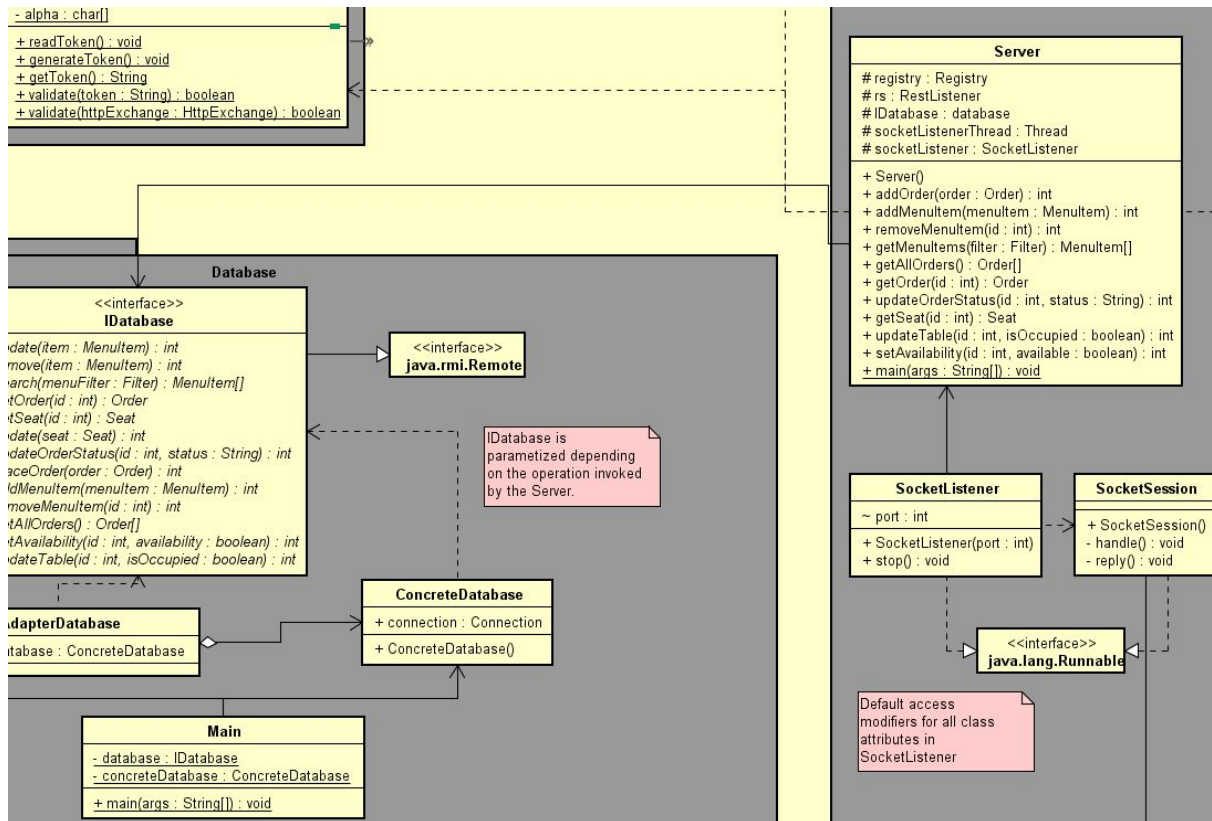
## 4.8 Class Diagram

Figure 12: Snapshot of system class diagram of server package



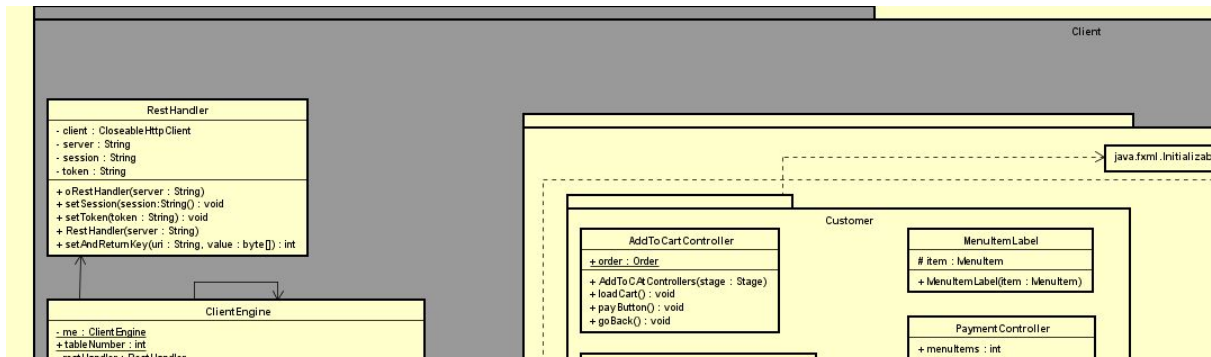
The illustration above represents the Java business layer which is responsible for establishing a socket connection to the web server in Figure 12.

Figure 13: Socket communication zoom-in on class diagram



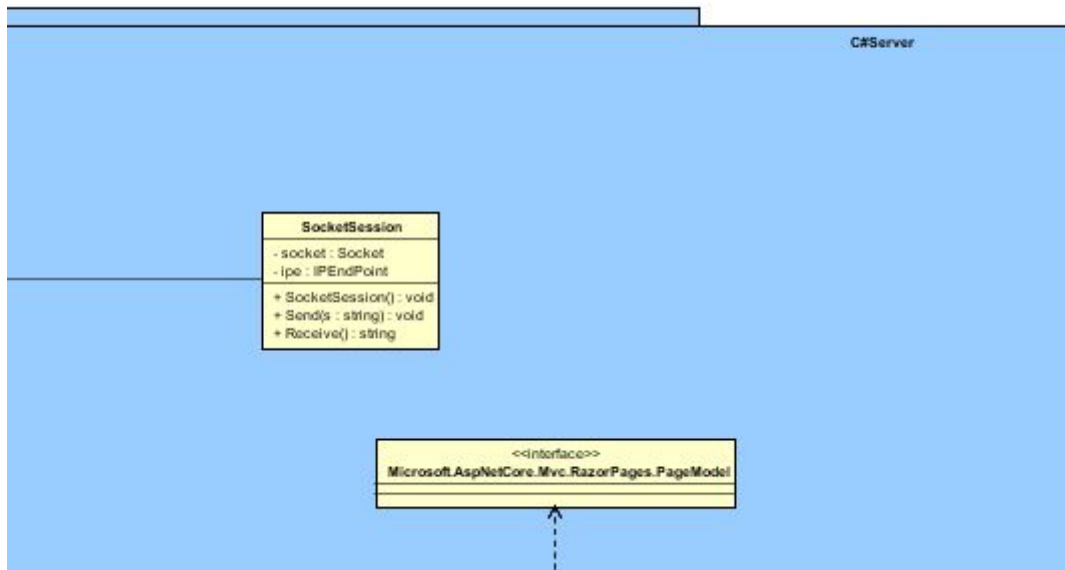
The **ConcreteDatabase** communicates with the server using an RMI communication. (Figure 13)

Figure 14: Handler communications in class diagram



RestListener has an instance of an HttpServer which communicates with an instance of CloseableHttpClient in the RestHandler class. (Figure 14)

Figure 15: C# Web server representation

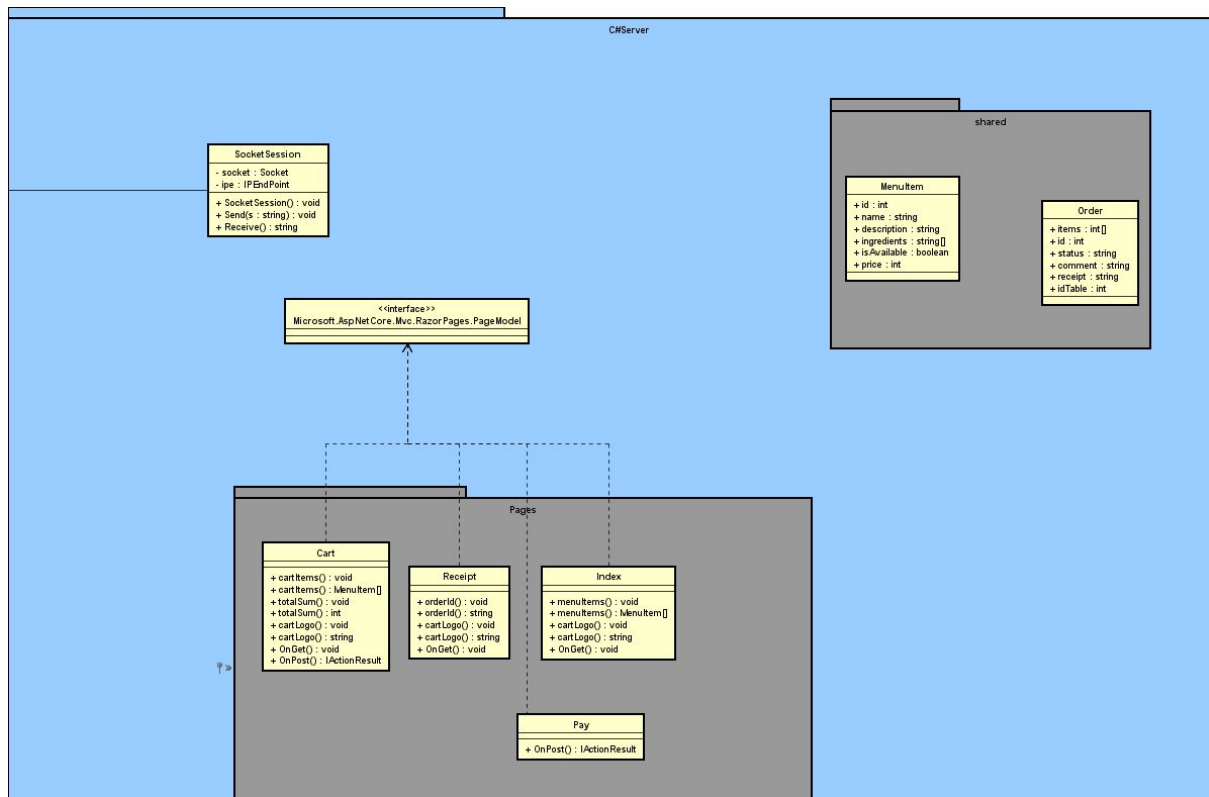


The class SocketSession on the WebServer and on the Server packages, communicate by a socket communication as shown by the association line (Figure 15).

## 4.9 Website and communication to C# server

When initializing the C# server, the URI on which the server is hosted is <https://localhost:5001> for an HTTPS connection and <http://localhost:5000> for an HTTP connection. The web representation makes use of the commands found in section 4.6 - GETMENUITEMS, SUBMITORDER.

Figure 16: Snapshot of C#Server.



The web server makes use of razor pages managed code framework.

## 5. Implementation

Figure 17: Snapshot of Database.ConcreteDatabase.java

```
96 public Order [] getAllOrders() {
97     PreparedStatement statement;
98     try {
99         statement = connection.prepareStatement("SELECT * FROM \"Kartofil\".orders");
100         ArrayList<Order> orders = new ArrayList<>();
101         ResultSet rs = statement.executeQuery();
102         statement = connection.prepareStatement("select * from \"Kartofil\".menuitem_order");
103         ResultSet rel = statement.executeQuery();
104         ArrayList<int[]> relation = new ArrayList<>();
105         while(rel.next()) {
106             int m = rel.getInt(1);
107             int o = rel.getInt(2);
108             relation.add(new int[] {m, o});
109         }
110         while (rs.next()) {
111             Order order = new Order();
112             order.id = rs.getInt(1);
113             order.idTable = rs.getInt(2);
114             order.status = rs.getString(3);
115             order.comment = rs.getString(4);
116             order.receipt= rs.getString(5);
117             ArrayList<Integer> items = new ArrayList<>();
118             for(int[] i : relation) {
119                 if(i[0] == order.id) {
120                     items.add(i[1]);
121                 }
122             }
123             order.items = new int[items.size()];
124             int j = 0;
125             for(int it : items) {
126                 order.items[j++] = it;
127             }
128             orders.add(order);
129         }
130         Order[] result = new Order[orders.size()];
131         orders.toArray(result);
132         return result;
133     }catch (SQLException e) {
134         e.printStackTrace();
135         return null;
136     }
137 }
138 }
```

On the Data Access tier, the method `getAllOrders()` returns an array of `Order` objects. This class is used on the business layer to represent the data associated with an order. The information needed to construct this object is not contained within a single table in the database, but also in a relational table mapping the order to items on the menu. This partition of the data is hidden from the business layer.

To avoid making an excessive number of database queries, the entire relational table (102) is loaded into a simple data structure in the form of an `ArrayList` (104) containing `int` arrays of length 2.

Afterwards, when constructing the individual `Order` object (110), this data structure is iterated to fetch the `MenuItem` objects (118).



Figure 18: Snapshot of Server.Server.java

```
public Order [] getAllOrders() {
    try {
        return database.getAllOrders();
    } catch (RemoteException e) {
        e.printStackTrace();
        return null;
    }
}
```

The server uses the Remote object “database” to call getAllOrders() in ConcreteDatabase.java. In case of the RMI connection failing, and throwing a RemoteException, this is handled in the method and null is returned.

Figure 18: Snapshot of Server.REST.RestListener.java

```
public RestListener(Server server) throws IOException {
    this.server = server;
    hServer = HttpsServerCreator.create(new InetSocketAddress(8001));
    hServer.setExecutor(null);
    hServer.createContext("/menu/list/", new MenuListHandler());
    hServer.createContext("/", new StdHandler());
    hServer.createContext("/seat/", new SeatHandler());
    hServer.createContext("/order/", new OrderHandler());
    hServer.createContext("/order/list/", new OrderListHandler());
    hServer.createContext("/order/status/", new OrderStatusHandler());
    hServer.createContext("/order/submit/", new OrderSubmitHandler());
    hServer.createContext("/menu/add/", new MenuAddHandler());
    hServer.createContext("/menu/remove/", new MenuRemoveHandler());
    hServer.createContext("/menu/availability/", new AvailableItemHandler());
    hServer.createContext("/table/status/", new SeatHandler());
}
```

The RestListener is instantiated by the Server instance with a reference to the Server instance. It then creates a com.sun.net.httpServer.HttpsServer using the HttpsServerCreator from the utility package. The individual HttpHandler implementations are mapped to URIs that they should handle requests on.



Figure 19: Snapshot of *Server.REST.OrderListHandler.java*

```
public void handle(HttpExchange httpExchange) throws IOException {  
    if(!httpExchange.getRequestMethod().equals("GET")) {  
        badRequest(httpExchange);  
        return;  
    }  
    if(!Utils.Token.validate(httpExchange)) {  
        unauthorized(httpExchange);  
        return;  
    }  
    Order[] orders = RestListener.server.getAllOrders();  
  
    StringBuilder sb = new StringBuilder();  
    ObjectMapper mapper = new ObjectMapper();  
    for (Order m : orders) {  
        sb.append(mapper.writeValueAsString(m));  
        sb.append('\n');  
    }  
    OK(httpExchange, sb.toString().getBytes());  
}
```

The `HttpExchange` object handles the HTTP communication with the client. It is created by the `HttpsServer` when the bound URI is requested and the `handle()` method in the `HttpHandler` interface is called. When relevant, the `handle()` method can check the request method from `HttpExchange.getRequestMethod()`.

In the code above, the `handle()` method of `OrderListHandler.java` is called as a result of a http request to `/order/list/`. The request is validated by checking that the request header contains a valid “token” key. If not, the request is rejected with a 401 http response by parsing the `HttpExchange` to the static method `unauthorized()` in `Response`, a class containing various static methods for giving common http responses. If the validation yields true, the array of `Order` objects is fetched using `Server.getAllOrders()` and a for loop adds the JSON representation of each of the objects to a `StringBuilder`, separating each by a new line character. The JSON representation of the objects is generated using the `ObjectMapper` class from the `com.fasterxml.jackson` libraries.

Figure 20: Snapshot of Client.ClientEngine.java

```
public Order[] getAllOrders() {
    String[] order = restHandler.get("/order/list/").split("\n");

    ArrayList<Order> p = new ArrayList<>();
    ObjectMapper mapper = new ObjectMapper();

    try {
        for (String o : order) {
            p.add(mapper.readValue(o, Order.class));
        }
    }

    catch (IOException e) {
        e.printStackTrace();
        return null;
    }

    Order [] res = new Order[p.size()];
    p.toArray(res);
    return res;
}
```

When the Java clients need to fetch Order objects, the get() method on the class RestHandler is called, performing an http request to the specified URI and returning the string contained in the body of the http response. This string is split by new line character and each individual string is converted into Order objects using an ObjectMapper.

When the frontpage of the web server is accessed, the OnGet() method of the Razor PageModel fetches the MenuItem objects that are available in the database. A SocketSession is created. The web server then calls the method Send() with the command GETMENUITEMS. As soon as the Send() method returns, the java server will have stopped listening on the socket and moved on to replying. Upon calling Receive() on the web server's SocketSession, the method will read from the socket until the stream is terminated by a 0x00 byte. The underlying socket is closed as the Receive() method returns. The web server then continues to split the received string by a '|' delimiter and deserializes each of the individual JSON strings into MenuItem objects.

Figure 21: Snapshot of Index.cshtml.cs

```
public class IndexModel : PageModel
{
    2 references
    public MenuItem[] menuItems {get; set;}
    2 references
    public string cartLogo{get; set;}

    0 references
    public void OnGet()
    {
        SocketSession ss = new SocketSession();
        ss.Send("GETMENUITEMS");
        string r = ss.Receive();
        string[] rs = r.Split(new char[]{'|'});
        menuItem[] = new MenuItem[rs.Length];
        int j = 0;
        foreach(string ms in rs)
        {
            MenuItem m = JsonConvert.DeserializeObject<MenuItem>(ms);
            menuItem[j++] = m;
        }

        if(Request.Cookies.ContainsKey("cart"))
        {
            cartLogo = "/lib/img/cart.png";
        }
        else {
            cartLogo = "/lib/img/cart_gray.png";
        }
    }
}
```

Figure 22: Snapshot of Cart.cshtml.cs

```
public IActionResult OnPost()
{
    StringBuilder sb = new StringBuilder();
    foreach(KeyValuePair<string, StringValues> kvp in Request.Form)
    {
        if(!kvp.Key.Equals("__RequestVerificationToken") && kvp.Value.Equals("on"))
        {
            sb.Append(kvp.Key);
            sb.Append(',');
        }
    }
    Response.Cookies.Delete("cart");
    if(sb.Length < 1)
    {
        return RedirectToPage("/Cart");
    }
    sb.Remove(sb.Length - 1, 1);
    Response.Cookies.Append("cart", sb.ToString());
    return RedirectToPage("/Cart");
}
```

When the users has checked the checkboxes of the items he wants to add to the cart and submits the form, the data is submitted to the OnPost() method of the PageModel for the

cart. The checkbox name, containing the MenuItem id, are extracted from the http request put in a comma separated string. This string is then stored in a cookie named "cart". Finally the method redirects to the same page, resulting in the OnGet() method being invoked and returning the page. If the user navigates to other parts of the site and comes back to the cart, the items will be remembered and will be loaded into the cart page by the OnGet() method.




## 6. Testing

### 6.1 Test Specifications

Testing on the system was performed in two ways: Black Box and White Box testing. The presentation layer which includes both the website and client UI for both customer and staff were tested following the Black Box methodology. White Box testing was performed on all methods that are essential for the functionality of the system.

### 6.2 Black Box and White Box testing

Legend:

Test Conclusion	Sign
Pass / Success	
Not applicable / None	
Fail	

*Table 3: Core functionality testing*











Test ID	Used API	Target component	Test procedure	Comments / Remarks	Event type	Outcome	System reacts on error?
1	Java	System clients - Customer AND Staff	Execute client main runnable	Server must be running first	Application launch		
2	Java	System clients - Customer AND Staff	Run client from multiple computers	Server must be running first	Application launch		
3	Java	Server	Execute server main runnable	Automatically launches web service	Application launch		
4	Java	Database	Execute database main runnable	-	Application launch		
5	C#	Web Server	Execute web server main runnable	Java server should be running first	Application launch		

Table 4: System content testing based on Black Box testing procedures performed on the Web Service

Test ID	Target test section	Test procedure	Comments / Remarks	Event type	HTTP status code	Outcome	System reacts to wrong input?
1	HTTPS - /menu/list/	Send a GET request from browser	-	Browser receives a JSON response	200 - OK	✓	✓
2	HTTPS - /	Send a GET request from browser	Forwards to navigation help page	Browser receives a response	200 - OK	✓	✓
3	HTTPS - /table/status/*/	Send a GET request from browser	"*" should be a valid ID	Browser receives a response	200 - OK	✓	✓
4	HTTPS - /table/status/*/	Send a PUT request with text data	"*" should be a valid ID	Browser receives a response	200 - OK	✓	✓
5	HTTPS - /order/*/	Send a GET request from browser	"*" should be a valid ID	Browser receives a JSON response	200 - OK	✓	✓
6	HTTPS - /order/submit/	Send a PUT request with JSON data	Order ID is handled on server-side	Browser receives a response	200 - OK	✓	✓
7	HTTPS - /order/status/*/	Send a PUT request with status in body	"*" should be a valid ID	Browser receives a JSON response	200 - OK	✓	✓
8	HTTPS - /menu/add/	Send a PUT request with JSON data	Body should NOT be empty	Browser receives a JSON response	200 - OK	✓	✓
9	HTTPS - /menu/remove/*/	Send a request to the server-side	"*" should be a valid menu item ID	Browser receives a response	200 - OK	✓	✓
10	/menu/availability/*/	Send a PUT request with text data	"*" should be a valid menu item ID	Browser receives a response	200 - OK	✓	✓

Table 5: System content testing based on Black Box testing procedures performed on Customer client - all UI scene transitions are marked in orange colour. Back button functionality excluded from testing procedure.

Test ID	Target test section	Test procedure	Comments / Remarks	Event type	Outcome	System reacts to wrong input?
1	EnterTableNumberController	Input data in text area	-	Data type Integer input	✓	⊖
2	EnterTableNumberController	Press "Start" button	Validates & saves table no. and redirects to MainViewController	Button press	✓	✓
3	MainViewController	Enter search criteria in text area	-	Data input	✓	⊖
4	MainViewController	Press "Search" button	-	Button press	✓	⊖
5	MainViewController	Click on menu item from list	Click performed on bold text	Mouse click	✓	⊖
6	MainViewController	Select search filter checkboxes	Does not function	Checkbox select	✗	⊖
7	MainViewController	Press "View Cart button"	Redirects to AddToCartController	Button press	✓	⊖
8	AddToCartController	Input data in text area	-	Data input	✓	⊖
9	AddToCartController	Press "Pay" button	Redirects to PaymentController	Button press	✓	⊖
10	PaymentController	Press radio button	Dummy payment method	Radio button press	✓	⊖
11	PaymentController	Press "Pay" button	Redirects to ReceiptController	Button press	✓	⊖
12	ReceiptController	Press "mainView" button	Redirects to MainViewController	Button press	✓	⊖

Table 6: System content testing using Black Box testing procedures performed on the Staff client - all UI scene transitions are marked in orange colour. Back button functionality excluded from testing procedure.

Test ID	Target test section	Test procedure	Comments / Remarks	Event type	Outcome	System reacts to wrong input?
1	StaffMainController	Press "Menu" button	Redirects to RestaurantMenuController	Button press	✓	✗
2	StaffMainController	Press "Orders" button	Redirects to OrderController	Button press	✓	✗
3	StaffMainController	Press "Table" button	Redirects to TableController	Button press	✓	✗
4	RestaurantMenuController	Press "Show Items" button	Loads all menu items in table	Button press	✓	✗
5	RestaurantMenuController	Click on item name	Shows prompt for removing item	Table click	✓	✗
6	RestaurantMenuController	Click on item availability	Shows prompt for setting the availability of the item	Table click	✓	✗
7	RestaurantMenuController	Click on "Add" button	Redirects to AddMenuItemController	Button press	✓	✗
8	AddMenuItemController	Input data in name text area	-	Data input	✓	✗
9	AddMenuItemController	Input data in ingredients text area	-	Data input	✓	✗
10	AddMenuItemController	Input data in price text area	Price must be an Integer	Data type Integer input	✓	✓
11	AddMenuItemController	Input data in description text area	-	Data input	✓	✗
12	AddMenuItemController	Press "Add Item to the Menu" button	Clears all fields upon success	Button press	✓	✓
13	OrderController	Click on item availability	Shows prompt for setting order status	Table click	✓	✗
14	TableController	Press on a select table button - 1 to 10	Changes the table status	Button click	✓	✓

Table 7: System content testing using White Box testing procedures performed on ClientEngine class

Test class name	Target test class	Test class method	Target method	Assert outcome
ClientEngineTest	ClientEngine	searchMenuItems()	searchMenuItems()	PASS
ClientEngineTest	ClientEngine	getMenu()	getMenu()	PASS
ClientEngineTest	ClientEngine	getAllOrders()	getAllOrders()	PASS
ClientEngineTest	ClientEngine	changeOrderStatus()	changeOrderStatus()	PASS

### 6.3 Test conclusions:

A total of 41 test cases were performed using the Black Box testing methodology on the Customer and Staff clients. Four additional tests were done on the ClientEngine class which contains critical system functionality methods leading to a total of 45 test cases overall.



## 7. Results and Discussion

The application is implemented in Java and C# programming languages. SQL query language is used for database implementation. Application is functional, but there is always room for improvements of functionalities and the UI design.

Firstly, on the customer side the system contains search functionality in a main window. The users are able to search for menu items and see all the ingredients that an item contains. Pressing the name of the menu item gives the user the possibility to add the item to the cart. Payment view contains different payment methods ('Cash','Debit-Card','Credit-Card') and total sum of all items that are ordered. Following the submission of the payment, user is presented with a confirmation view for successfully paying. After the payment confirmation has been made, they are able to go back to the main view and continue ordering if they so desire.

All things considered, the system is implemented and fully documented with all diagrams, and reports.

When publishing the system onto a real market and selling it to a client, the user interface should be improved. More functionalities should be added overall to fulfill the purpose that the system should perform. The backend code is modular and easy to maintain.

## 8. Conclusions

The purpose of working on this system was to help owners of restaurants in serving customers faster and more efficient. All in all, system is implemented and contains most of the requirements. There are two requirements that have not been implemented. System is designed in a simplistic manner and documented with reports and diagrams. Improvements on the design part are necessary for real-life usage.

## 9. Project Future

Future implementations for the system:

- 1) System should have a feature for requesting a new token and for invalidating a token if the token has been stolen.
- 2) System should have a backup of the database in case of security breach.
- 3) Proper payment features should be implemented.
- 4) Search functionality should be improved.
- 5) Users should be able to filter items on the menu categorically.
- 6) A proper receipt system should be implemented for when an order has been completed.
- 7) Customers should be able to leave feedback after an order has been delivered.
- 8) Users should be able to log in on the web page to avoid re-entering personal information when ordering.
- 9) When launching the server, both it and the database should be launched simultaneously. The current way of launching the server and database is to start them one by one.

## 10. Sources of information

- 1) Restaurant Technology Guys, 2015. 5 Reasons Your Restaurant Needs a Table Ordering System. [online] Available at:  
<<https://restauranttechnologyguys.com/5-reasons-restaurant-needs-table-ordering-system/>> [Accessed 23 September 2018].
- 2) VirginiaTech, 2016. The Influence of Table Top Technology in Full- service Restaurants. [online] Available at:  
<<https://vtechworks.lib.vt.edu/handle/10919/81890>> [Accessed 19 September 2018].

## 11. Appendices

Appendix A: Use Case Diagram (file: Group\_5\_Use\_Case\_Diagram.asta)

Appendix B: Class Diagram (file: Group\_5\_Class\_Diagram.svg)

Appendix C: Project Description (file: Group\_5\_Project\_Description.pdf)

Appendix D: User Guide (file: Group\_5\_User\_Guide.pdf)

Appendix E: Database EER diagram (file: Group\_5\_EER\_Diagram.pdf)

Appendix F: Source code (directory: Code)

Appendix G: System diagram (file: Group\_5\_System\_Diagram.pdf)

Appendix H: Requirements (file: Group\_5\_Requirements.pdf)

Appendix I: System sequence diagram (file: Group\_5\_System\_Sequence\_Diagram.pdf)

Appendix J: Analysis Diagram (file: Group\_5\_Analysis\_Diagram.pdf)

Appendix K: Mapping Steps (file: Group\_5\_Mapping\_Steps.pdf)

Appendix L: 3-Tier Architecture (file: Group\_5\_3Tier\_Architecture.pdf)

Appendix M: Model Diagram (file: Group\_5\_Model\_Diagram.pdf)

Appendix N: Activity Diagram (file: Group\_5\_Activity\_Diagram.pdf)