



## Software Architecture Coursework

SET10101

Enrique Belenguer

10106742@live.napier.ac.uk

### Abstract

A nationwide retailing corporation (such as Homebase <http://www.homebase.co.uk/> or B&Q <http://www.diy.com/>) is planning to develop a new distributed store management system for their retail branches to provide better coordination of their business. It has named the proposed system DE-Store.

Your company want to pitch for the software development contract and plan to do this by developing a software prototype of an architecture that you believe would show that you could meet the needs of the project.

DE-Store is NOT an online shopping system; instead it is a DISTRIBUTED business management system. Please note that DE-Store is a DISTRIBUTED system. You are supposed to use the appropriate architecture styles and technologies you've learnt to develop an effective solution, such as client/server, peer to peer, service-oriented, RMI, three-tiered, etc. DE-Store is expected to be an expandable and adaptive system to accommodate changing business requirements in the future.

DE-Store aims to have a suite of store management functionalities such as price control, inventory control, delivery charge, approval of financial support, and performance analysis.

- Price Control: DE-Store allows the store manager to set the price of the products and to select products on a variety of sale offers, which include 3 for 2, buy one get one free, free delivery charges.
- Inventory Control: stock is monitored all the time by uploading data from the warehouse database. Items out of stock will be ordered from the central inventory system at the headquarters. DE-Store generates warning messages for items in low stock automatically and also sends them to the mobile message box of the store manager.
- Loyalty Card: the store can make further special offers to customers who regularly use their branches.
- Finance Approval: DE-Store offers its customer the opportunity to buy now and pay later using an online finance system, Enabling, which is linked to DE-Store via a portal.
- Reports and Analysis: DE-Store tracks the purchase activities of customers from the accounting database and generates reports on how the store is performing.

DE-store is expected to be an expandable and adaptive system to accommodate changing business requirements in the future.

## Discussion of Two Architectures

An important aspect to bear in mind when choosing an architecture is the specifications of the software you are going to develop. In this case, the client specified clearly two very important things that we should consider. The first one is that the DE-Store is a DISTRIBUTED system, and the second and very important one is that the DE-Store is expected to be an expandable and adaptive system to accommodate changing business requirements in the future. Because of those two reasons, the two Architecture Systems that I have chosen are Client-Server and Peer to Peer.

### Client-Server

Client-Server is one of the most used architectures when in terms of distributed system. It allows users to obtain the access to the information in a clear way, in a multi-platform environment.

It's a model of distributed application where the tasks are shared between the following: service providers, a client who request services, and a server who provides services. A server is a machine that acts as a data-warehouse and operates as a database management system. This is responsible for responding to customer demands. This architecture is applied in different computer models worldwide.

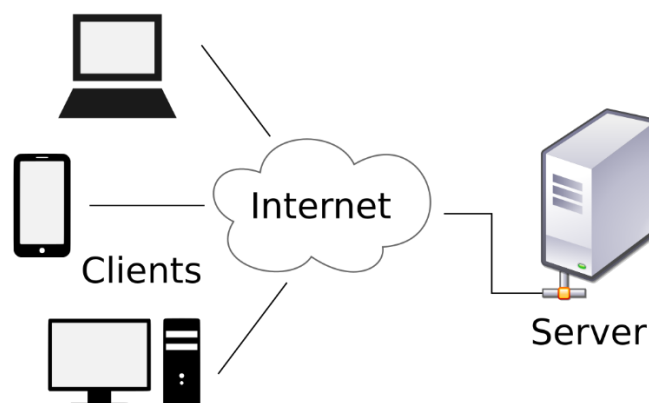


Figure 1. Client-Server

Its' purpose is to maintain information communication between the entities of a network using established protocols and proper storage of this. The clearest example of client-server architecture is the Internet network - there are different computers of various people connected around the world. The computers are then connected through the servers of their Internet provider which redirect them to servers of the pages they request. This way, the information of the required services travels through the Internet in response to the demand request.

Within the client server architecture there are two types:

- **2-tier architecture.** This is used to describe the client-server systems where the client requests resources and the server responds directly to the request with its' own resources. That means that the server does not require another application to provide the service.
- **3-tier architecture.** In the three-layer architecture there is an intermediate level. That means that the architecture is normally integrated by:  
Client. The equipment requesting resources equipped with a user interface, usually a web browser.  
Web Server. Which provides the requested resources after requesting it from another service. Inside the web server there is number of components (Figure 2) like Servlets, Web Components, JavaBeans Components that deal with the procedures previously explained.  
Data Server. It provides the web server with the data requested.

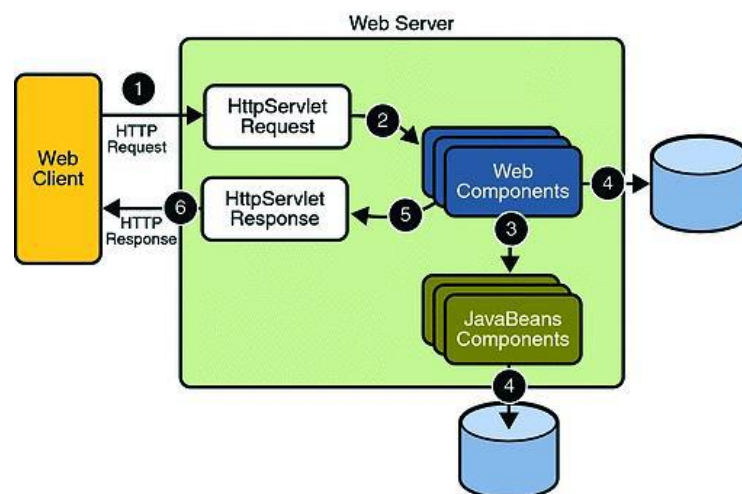


Figure 2. 3-Tier Architecture on Client-Server

### Advantages

- Centralization: the server controls the resources and data integrity. None of the nodes can damage the system.
- Scalability: it is easy to upgrade- either by upgrading parts or adding new nodes to the system.
- Easy maintenance: because its' modular and functions are distributed, it is easy to replace or repair parts independently (encapsulation).
- There are enough technologies developed for this model.

## Disadvantages

- Traffic congestion: when many clients send request to the same server it might cause problems for the server.
- The hardware and software from the server are very expensive. They must be very powerful, and this obviously increases the costs.
- The client does not know what may exist on the server.

## Peer to Peer

Unlike client-server networks, in the Peer to Peer network all machines have the same capabilities and responsibilities. Basically, Peer to Peer networks are computer networks that work without having clients or fixed servers. This gives them flexibility that would otherwise be impossible to achieve. This is achieved because the network works in the form of a series of nodes that behave as equal.

Another advantage associated with P2P networks is that they can take advantage of the use of available bandwidth between users for file sharing, which allows this way to obtain a better performance which means better transfer speed.

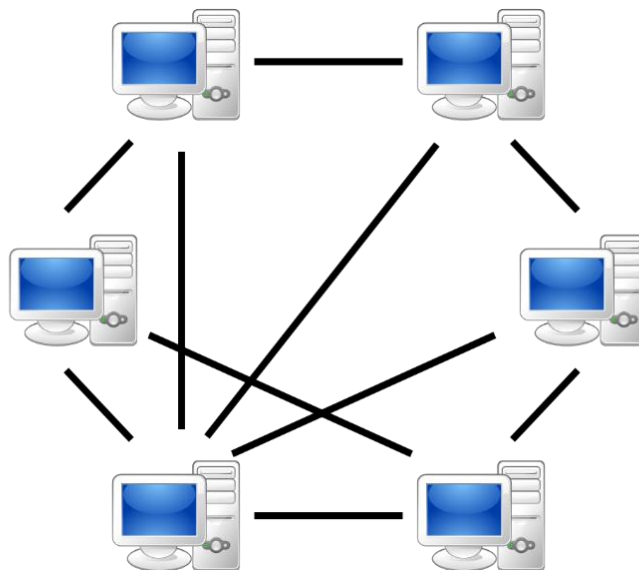


Figure 3. Peer to Peer Architecture

The way in which Peer to Peer networks manage the use of available bandwidth makes a noticeable difference in performance with other types of more centralized networks, where the bandwidth is provided by a set of servers that could never outnumber those found in Peer to Peer networks.

Types of Peer to Peer Systems:

- **Pure P2P Model.** This model works without relying on any central servers. Communication occurs without any server assistance, just amongst connected peers.
- **P2P with Simple Discovery Server.** The only function of the server in this model is to notify the new nodes about the list of connected peers.
- **P2P with Discovery and Lookup Server.** In this model there is a server, but the function of it is just to notify the new nodes where the files can be found.

### Advantages

- Very reliable: since central dependency is eliminated, failure of one peer does not affect the functioning of other peers.
- Simple to configure: each P2P network is very easy to install and relatively cheap in comparison with a client-server network.
- There is no need to have any specialist staff. Each user is their own administrator and can also choose what to share.

### Disadvantages

- May have duplication in resources. You can find many files multiple times in the network.
- Difficult to uphold security policy. This is probably the main disadvantage of the P2P networks. Virus can easily be transmitted.
- Backup must be performed on each computer separately.

## Reason for Selecting One of Them.

In order to select the most suitable architecture, I have used the Software Architecture Analysis Method (SAAM).

To start with the Analysis, I have chosen a couple scenarios where I have compared both architectures. Scenarios were typical to the kind of evaluations that the system must support. They have represented tasks relevant to all stakeholders. I have selected the following as my scenarios:

1. Expanding the system.
2. Maintaining the system.
3. Maintaining it day by day.
4. Making backups.
5. Keeping it free from virus.
6. Improving the user interface.
7. Changing data structure.
8. Making changes in the database.

The next step in the SAAM is the description of both candidates' architectures. It has already been done in the previous chapter, so instead I have classified the scenarios. To classify the scenarios, I have created a table with the scenarios and each architecture. In addition, I have evaluated if the architecture should be modified or not to achieve the scenarios' specifications.

Furthermore, I have evaluated the costs - which is part of the perform scenario evaluation. This was the next step of SAAM. Alas, I did not know the exact cost of each scenario, so I have marked them with: cheap, normal and expensive.

Architecture	Scenario	Direct/Indirect	Cost
Client-Server	1. Expand the system	Direct	Medium
	2. Maintain the system.	Direct	Cheap
	3. Maintain it day by day.	Indirect	Medium
	4. Make backups	Direct	Cheap
	5. Keep it free from virus.	Direct	Cheap
	6. Improve the user interface.	Direct	Medium
	7. Change data structure.	Direct	Medium
	8. Make change in the database.	Direct	Medium
P2P	1. Expand the system	Direct	Medium
	2. Maintain the system.	Direct	Cheap
	3. Maintain it day by day.	Direct	Cheap
	4. Make backups	Indirect	Expensive
	5. Keep it free from virus.	Indirect	Expensive
	6. Improve the user interface.	Direct	Medium
	7. Change data structure.	Direct	Medium
	8. Make change in the database.	Direct	Medium

For the overall evaluation I have ranked the Architectures and I assigned a weight to each scenario as well as each interaction.

Architecture	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6	Scenario 7	Scenario 8
Client-Server	0	0	-	+	+	0	0	0
P2P	0	0	0	-	-	0	0	0

After doing the SAAM, I have decided to use the Client-Server Architecture System style. In my case, both architectures meet the most important requirements requested by the Client. Both are Distributed Systems and are an expandable and adaptive systems to accommodate changing business requirements in the future. The main reason why I have decline Peer to Peer was because it is not very secure and viruses can be easily transmitted. In addition, Client-Server ensures the companies that any employer from any of the nodes can damage the system. It also offers clients the possibility to easily keep a backup of the server. For companies this makes it very good and important.

### **Client-Server Architecture**

Amongst large companies, the Client-Server Architecture - using the 3-tier Architecture – is very common. Even though the companies will have to spend some money on the server and its' equipment, the Client- server has several stronger advantages. In this case, I have prioritized the security over anything else. Even if both architectures were good enough for the business model, Client-Server model offers a much greater security in terms of viruses compared to the Peer to Peer model. In the Client-Server model, no node can infect the server. Moreover, the model is very easy to maintain which – despite the initial investment - results in an affordable maintenance.

### **3-tier Architecture**

The previously mentioned advantages of the client server architecture were as follows: centralization, scalability, easy maintenance and the fact that there is enough technologies developer for this model. In addition, JAVA EE is a multitiered application model, which makes the system very easy to modify and upgrade in the future. The 3-tier architecture allows developers to modify one layer without affecting any other layers. There are no tier-to-tier dependencies, which means that the backend developer can modify things in the database with no effect at all on front-end developer.

## Design

### Analysis

The program is a Database Administration Application. This allows the administrator of the application to: manage products (add, edit, delete), manage clients (add, edit, delete and apply certain offers), and analyze the performance of the shop. Registers are added using forms. Information is displayed using tables that allow the administrator to see the information clearly.

### Modelling

In my approach, for modeling my problem I have chosen to use 3tier by using JAVA EE. There are three different tier well differentiates.

- Client Tier: the client tier consists of application clients that access a Java EE server and that are usually located on a different machine from the server. In this Tier the files HTML and JSP are located.
- Web Tier: the web tier consists of components that handle the interaction between the client and the Enterprise Information Systems using APIs. In this tier, this is where my Servlets are located. They dynamically process requests and construct responses usually using the HTML pages from the Client Tier.
- Enterprise Information System (EIS) Tier: the EIS tier consist of database servers, enterprise resource planning systems and other legacy data sources. In this tier my database is located. It gets request from the servlets, process it and responds back.

### Case Diagram

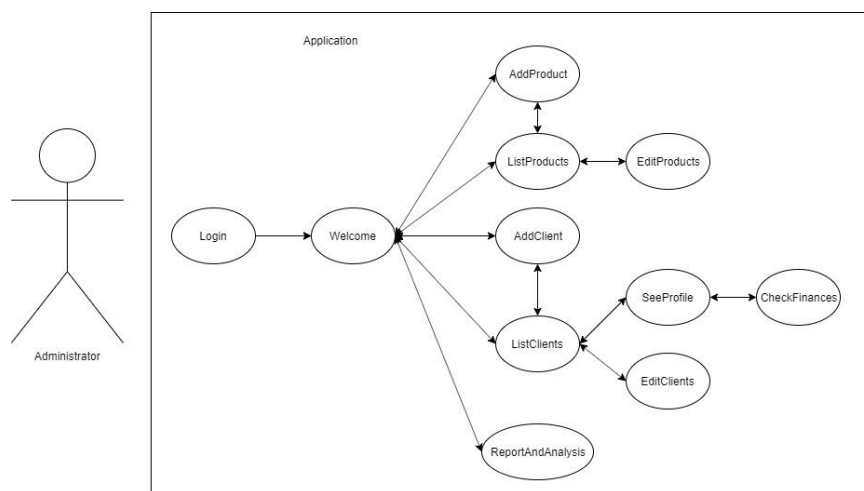


Figure 4. Navigation Map



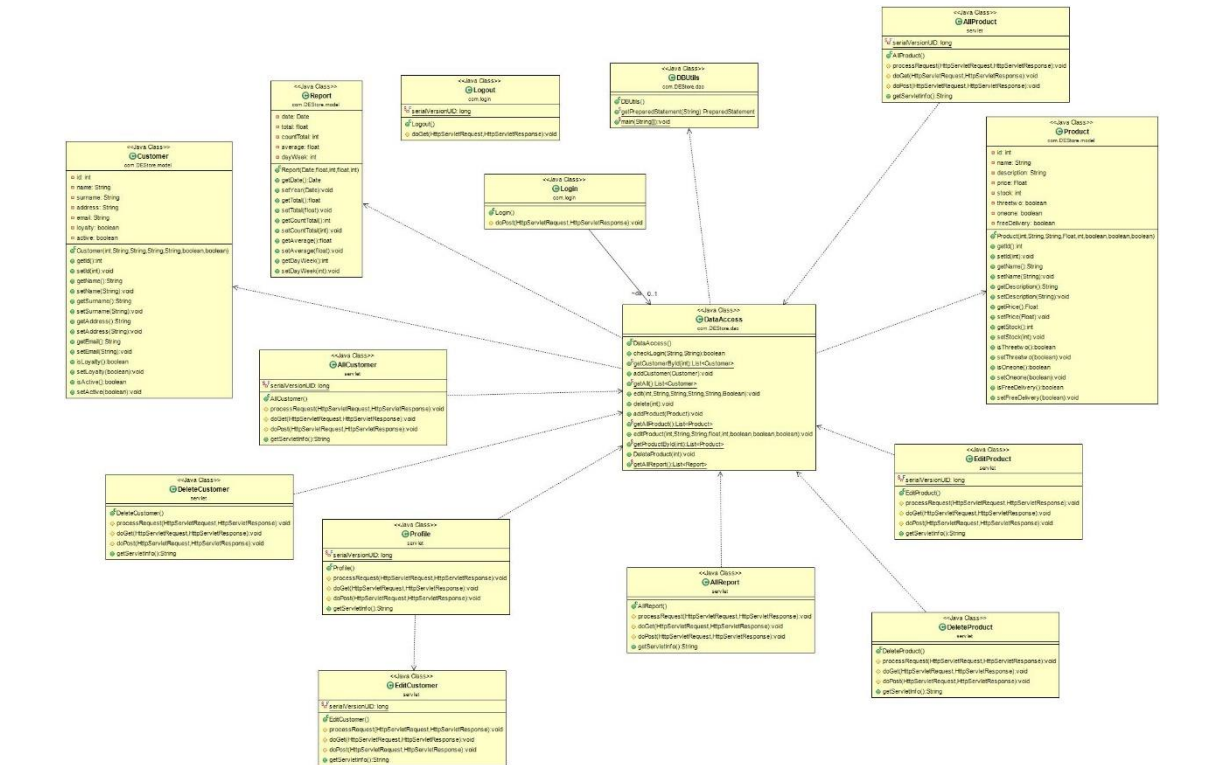


Figure 5. UML Class Diagram

## Class Design.

My Class Diagram consists of 13 different Java classes.

- **DBUtils**: it is a part of the Data Access Object (DAO). It is a public class with a main method called **PreparedStatement()**. Its' functions are to create the connection with the database, then to execute the query that is received by parameter and return the result.
- **DataAccess**: it is also part of the DAO. It is a public class with many methods in it like: **checkLogin()**, **getCustomerById()**, **addCustomer()**, etc. **DataAccess** is used to prepare the queries (Insert,Delete,Edit, CheckLogin,etc) requested from the servlets and send it to the **DBUtils** where they will be executed.
- **Customer, Products and Report** classes: those classes match with the three entities with the same names from the database. They are used to obtain the information from tables. This class does not offer any other functionality other than to follow the Model-View-Controller architecture. It has a method that returns and sets each of the variable in the class and a Constructor. The classes are used to either add, edit, delete or list the items from any of the tables and they

are called from different method from the class DataAccess depending of task required.

- **Servlets** Classes: they are classes which responds to a particular type of network request, in this case all of them are requests from the views and are send to the IES tier. When it obtains the answer from them, the servlets send the answer back to the views to generate a dynamic HTML. The name of the classes are: AllCustomer, AllProduct, AllReport, DeleteCustomer, DeleteProduct, EditCustomer, EditProduct, Profile, Login and Logout.

## Data Structure

The main data structure used in this project is the LinkedList.

*LinkedList* is a linear data structure in which elements are not stored in contiguous locations. Every element is a separate object with a data part and address part. I have decided to use LinkedList instead of Array because they perform better on insertions/deletions and they are easy to grow or shrink because there is not limit on ListSize.

## Relationship.

There are a few relationships between the classes:

- DataAccess(dependency) calls method from the DBUtils class to connect to the database.
- Customer (association) AllCustomer/ EditCustomer/ DeleteCustomer/Profile: The servlets receive the request from the view and use to Model Constructor to provide the answer to the views.
- Product (association) AllProduct/EditProduct/DeleteProduct: The servlets receive the request from the view and use to Model Constructor to provide the answer to the views.
- Report (association) AllReport: The servlets receive the request from the view and use to Model Constructor to provide the answer to the views.
- Login(association) DataAccess: The servlets receive the request from the view and use the DataAccess to provide the answer to the view.

## Evaluation of design and implementation.

### Evaluation of design

I am very happy with the result of my project. Before I have started to code, I did a lot of research about various architectures I could implement in it. I did not want to start the development and then realize that I was not right. It would have made me lose a lot of time. Each of the architectures I have studied during the research had pros and cons. During the development time of my project, I have realized that I have chosen the right one. I have learnt how simple it is to use the servlets and how easy it is to build dynamic content for web-oriented applications with them.

### Implementation

#### Appendix 1.

#### Lessons learned

This coursework made me understand better the different software architecture styles. I have learnt their pros and cons and identified which one to use in a determinate situation. Along with it, I have learnt the Java EE architecture. This can be very useful for me in the future because it is used a lot for many companies in the globally competitive world. I have improved my knowledge of JAVA and learnt JSP. Basically, this coursework has taught me a lot of things that I will use in the future in the professional field.

## Appendix

### Appendix 1. Implementation

- **DBUtils class.** It is a public class with one Method in it.

- *Methods.*

- *getPreparedStatement(String sql).* It is a static method that receives a parameter String, makes the connection with the database and performs the query received by the parameter and returns the results using reflection techniques.

```
public class DBUtils {
    public static PreparedStatement getPreparedStatement(String sql)
    throws ClassNotFoundException, SQLException{

        PreparedStatement ps = null;
        Class.forName("com.mysql.cj.jdbc.Driver");
        String url = "jdbc:mysql://localhost:3306/destore";
        String username = "root";
        String password = "";

        Connection con = DriverManager.getConnection(url, username,
password);
        ps = con.prepareStatement(sql);

        return ps;
    }
}
```

- **DataAccess class**. It is a public class with twelve Methods in it. All method in this class make instances of `getPreparedStatement()` to execute queries.

- *Methods*.

-*checkLogin(String uname, String pass)*. It is a public Boolean method that receives two parameters and concatenates them to a query and then returns true in case that it obtains any result from it.

-*getCustomerById(int id)*. It has an int a parameter and return a `List<Customer>`.

-*addCustomer(Customer c)*. It has a costumer as a parameter and execute a query to add a customer to the database.

-*getAll()*. It returns a `LinkedList` of `Customer`.

-*edit(int id, String name, String surname, String address, String email, Boolean loyalty)*. It receives six parameters and execute and update query with all of them.

```
public void edit(int id, String name, String surname, String address,
String email, Boolean loyalty) {
    try {
        String sql = "update Customer SET name = ?, surname = ?,
address = ?, email = ?, loyalty = ?"
            + " where id = ?";
        PreparedStatement ps = DBUtils.getPreparedStatement(sql);
        ps.setString(1, name);
        ps.setString(2, surname);
        ps.setString(3, address);
        ps.setString(4, email);
        ps.setBoolean(5, loyalty);
        ps.setInt(6, id);
        ps.executeUpdate();
    } catch (ClassNotFoundException | SQLException ex) {
        Logger.getLogger(DataAccess.class.getName(), null).log(
            Level.SEVERE, null, ex);
    }
}
```

-*delete(int id)*. It has an int parameter and execute a query into the database.

-*addProduct(Product p)*. It has a product as a parameter and execute a query to add a product to the database.

-*getAllProduct ()*. It returns a `LinkedList` of `Product`.

-*editProduct(int id, String name, String description, float price, int stock, boolean threetwo, boolean oneone, boolean freeDelivery)*. It receives seven parameters and execute and update query with all of them.

-*getProductById(int id)*. It has an int a parameter and return a `List< Product >`.

-*deleteProduct(int id)*. It has an int parameter and execute a query into the database.

-*getAllReport ()*. It returns a `LinkedList` of `Report`.

- **Customer class.** It is a public class with a Constructor Method and two more Methods for each variable in the class. One to return the variable and the other to set the variable.

- *Methods.*

- *Customer(int id, String name, String surname, String address, String email, boolean loyalty, boolean active).* It's the constructor.

- *getId().* It return the ID of the customer.

- *setId(int id).* It set the ID of the customer.

It has exactly the same two methods for each of the parameter in the Constructor.

```
public Customer(int id, String name, String surname, String address,
                String email, boolean loyalty, boolean active) {
    super();
    this.id = id;
    this.name = name;
    this.surname = surname;
    this.address = address;
    this.email = email;
    this.loyalty = loyalty;
    this.active = active;
}
```

- **Product class.** It is a public class with a Product Method and two more Methods for each variable in the class. One of them to return the variable and the other to set the variable.

- *Methods.*

- *Product(int id, String name, String description, Float price, int stock, boolean threetwo, boolean oneone, boolean freeDelivery)*

- *getId().* It return the ID of the product.

- *setId(int id).* It set the ID of the product.

It has exactly the same two methods for each of the parameter in the Constructor.

- **Report class.** It is a public class with a Report Method and two more Methods for each variable in the class. One of them to return the variable and the other to set the variable.

- *Methods.*

- *Report(Date date, float total, int countTotal, float average, int dayWeek)*

- *getDate ().* It return the date of the report.

- *setDate (Date date).* It set the date of the report.

It has exactly the same two methods for each of the parameter in the Constructor.

- **AllCustomer class.** It is a public class, servlet in this case, with four different methods.

```
public class AllCustomer extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException{
        request.setAttribute("AllCustomer", DataAccess.getAll());
        RequestDispatcher rd =
        request.getRequestDispatcher("AllCustomer.jsp");
        rd.forward(request, response);
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
        response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
        response)
        throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

- *Methods.*



-*doGet(HttpServletRequest request, HttpServletResponse response).* It handles with request coming from forms with post methods

-*doPost(HttpServletRequest request, HttpServletResponse response).* It handled request coming from get method.

-*processRequest(HttpServletRequest request, HttpServletResponse response).* It handles request if and only if it's called from any of the methods above.

AllProduct, AllReport, DeleteCustomer, DeleteProduct, DeleteCustomer, EditCustomer, EditProduct and Profile works exactly the same as the Servlet previously explained. They just get request, handle them and dispatche them to a dynamic html.

## Appendix 2. User Interface

Enter Username:	<input type="text" value="admin"/>	
Enter Password:	<input type="password" value="....."/>	
<input type="button" value="Login"/>		

Login Page

## Welcome admin

<a href="#">Add Customer</a>	<a href="#">Customer Listed</a>	<a href="#">Add Product</a>	<a href="#">Inventory Control</a>	<a href="#">Reports and Analysis</a>
<input type="button" value="Logout"/>				

Welcome Page

## Add Customer

Name:	<input type="text"/>
Surname:	<input type="text"/>
Address:	<input type="text"/>
Email:	<input type="text"/>
Loyalty Card	<input type="checkbox"/>
<input type="button" value="Add Customer"/>	

<a href="#">Go Back</a>	<a href="#">All Customer</a>
-------------------------	------------------------------

Add Customer Page

## All Customer

ID	Name	Surname	Address	L.Card	
1	John	Stockton	Utah	Yes	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">See Profile</a>
2	Kobe	Bryant	LA	Yes	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">See Profile</a>
3	Stephen	Curry	SF	No	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">See Profile</a>
4	Enrique	Belenguier	Edinburgh	No	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">See Profile</a>

<a href="#">Go Back</a>	<a href="#">Add Customer</a>
-------------------------	------------------------------

All Customer Page

## Edit Customer

Name:	<input type="text" value="John"/>
Surname:	<input type="text" value="Stockton"/>
Address:	<input type="text" value="Utah"/>
Email:	<input type="text" value="John@gmail.com"/>
Loyalty Card:	<input checked="" type="checkbox"/>
<input type="button" value="Update Customer"/>	

[Go Back](#)

Edit Customer Page

## Add Product

Name:	<input type="text"/>
Description:	<input type="text"/>
Price:	£ <input type="text"/>
Stock:	<input type="text"/>
3x2 Offer:	<input type="checkbox"/>
Get One, One Free:	<input type="checkbox"/>
Free Delivery:	<input type="checkbox"/>
<input type="button" value="Add Product"/>	

[Go Back](#) [All Products](#)

Add Product Page

## Inventory Control

ID	Name	Description	Price	Stock	3x2	Buy1Get1	Free Delivery	
1	Chair	Chair Description	14.6	60	Yes	Yes	No	<a href="#">Edit</a> <a href="#">Delete</a>
2	Table	Table Description	40.5	50	Yes	No	Yes	<a href="#">Edit</a> <a href="#">Delete</a>
3	Desk	Desk Description	60.5	50	Yes	No	Yes	<a href="#">Edit</a> <a href="#">Delete</a>

[Go Back](#) [Add Product](#)

Inventory Control Page

## Edit Product

Name:	<input type="text" value="Chair"/>
Description:	<input type="text" value="Chair Description"/>
Price:	<input type="text" value="14.6"/>
Stock:	<input type="text" value="60"/>
3x2 Offer:	<input checked="" type="checkbox"/>
Get One, One Free:	<input checked="" type="checkbox"/>
Free Delivery:	<input type="checkbox"/>
<input type="button" value="Update Product"/>	

[Go Back](#)

Edit Product Page



## Report And Analysis

Day Of Week	Date	Number Of Sales	Average Sale	Total Sales
Thursday	23-10-2019	2	£ 467.85	£ 935.7
Sunday	12-10-2019	5	£ 557.59	£ 2787.96
Tuesday	07-10-2019	2	£ 2345.26	£ 4690.52
Monday	06-10-2019	2	£ 515.36	£ 1030.72
Monday	15-09-2019	2	£ 789.11	£ 1578.22
Friday	05-09-2019	2	£ 3456.26	£ 6912.52
Wednesday	03-09-2019	2	£ 95.84	£ 191.68
Saturday	23-08-2019	2	£ 87.74	£ 175.48
Saturday	16-08-2019	3	£ 886.9	£ 2660.69
Sunday	03-08-2019	2	£ 745.34	£ 1490.68

[Go Back](#)

Report and Analysis Page

## Customer Profile

Name: John  
Surname: Stockton  
Address: Utah  
Email: John@gmail.com  
Loyalty Card: ☒  
**Finannce Approval**

[Go Back](#) [All Customer](#)

Customer Profile Page

## References.

1. Client-Server. [https://en.wikipedia.org/wiki/Client%E2%80%93server\\_model](https://en.wikipedia.org/wiki/Client%E2%80%93server_model)
2. 3-Tier Architecture. <https://moodle.napier.ac.uk/course/view.php?id=32065>
3. P2P architecture . <https://en.wikipedia.org/wiki/Peer-to-peer>