SOFTWARE ARCHITECTURE

Ozioma Eunice Onwubiko Student No: 40491921

Table of Content

1 | ARCHITECTURE RECOMMENDATION 1

1.1 LAYERED SYSTEM ARCHITECTURAL STYLE 1

1.1.1 Components 3

1.1.2 Connectors 3

1.1.3 Protocol for Information Exchange 3

1.1.4 Advantages and Disadvantages 4

1.2 HETEROGENEOUS ARCHITECTURAL STYLE 5

1.2.1 Components 6

1.2.2 Connectors 6

1.2.3 Protocol for Information Exchange 7

1.2.4 Advantages and Disadvantages 7

1.3 SUMMARY 8

2 | SELECTION & JUSTIFICATION OF ARCHITECTURE 9

2.1 REVIEW OF QUALITY ATTRIBUTES 9

2.1.1 Develop Scenarios 9

2.1.2 Describe Candidate Architectures 10

2.1.3 Classify Scenarios and Perform Scenario Evaluations 11

2.1.4 Reveal Scenario Interaction 11

2.1.5 Overall Evaluation 12

2.2 SUMMARY 12

3 | DESIGN & IMPLEMENTATION 13

3.1 UNIFIED MODELLING LANGUAGE (UML) AND DESIGN PROTOTYPE 13

3.2 TECHNOLOGY, FRAMEWORK AND SET-UP 13

3.3 PURPOSE AND RESPONSIBILITIES 15

3.4 COMPONENTS, CONNECTORS AND PROTOCOL 19

3.5 MEASURES 19

4 | EVALUATION 21

4.1 CRITICAL EVALUATION 21

4.2 FUTURE WORK 22

REFERENCES 23

List of Figures

Figure 1. Layered System and components ... 2
Figure 2. Client with its sub-components - layered system architecture ... 2
Figure 3. Protocol for Information Exchange ... 3
Figure 4. 3-Tier style ... 6
Figure 5. TCP/IP - Information Exchange ... 7
Figure 6. UML ... 13
Figure 7. Design Prototype ... 13
Figure 8. Database Connector and Reference Library ... 14
Figure 9. VSCode - code structure ... 15
Figure 10. DE-Store - Main Frame ... 15
Figure 11. DE-Store - Functionalities ... 16
Figure 12. Results ... 16
Figure 13. Sales.java ... 17
Figure 14. SalesDAO.java ... 17
Figure 15. SalesDAOImpl.java ... 17
Figure 16. Setting up database ... 18
Figure 17. Connecting MariaDB ... 18
Figure 18. Tables in database ... 18
Figure 19. 'Customer' table ... 19
Figure 20. PreparedStatement object in SQL ... 20

List of Tables

Table 1. Summary Table of Scenarios 11
Table 2. Reveal Scenario Interactions 12
Table 3. Overall Evaluation of candidate architectures 12

1 | Architecture Recommendation

In response to the evolving need of a nationwide retailing corporation, this document presents an architecture recommendation by providing a comprehensive overview of the selected software architectures best suited for the development of the DE-Store system. The primary goal in this chapter is to provide insights into two potential architectural styles, and discuss their benefits and drawbacks. Targeted at the technical decision-makers of the company, the aim of this recommendation is to facilitate an informed choice, ensuring that it aligns with the specific needs and requirements that the organisation requires for the development of the system.

DE-Store is a distributed store management system for a nationwide retailing corporation, aiming to provide better coordination of their business for their retail branches. To develop such system, the company would need a software architecture that is able to manage diverse components and functionality of the business in a structured approach, and flexible enough to accommodate changes in the future. In a distributed system, there are multiple machines that utilise their computational resources to work on the tasks they have been assigned with, and efficiently use their capacity to construct the system (*Architecture Styles in Distributed Systems - GeeksforGeeks*, n.d.). The goal is to avoid central points of failure coming from a system by decentralising and establishing a shared communication network. This network enables machines to interact with each other to complete a shared and common goal (*What Is a Distributed System? | Atlassian*, n.d.).

The DE-Store system requires store management functionalities such as price control, inventory control, delivery charge, approval of financial support, and performance analysis. To design a software system and guarantee its quality attributes, it is important to consider the benefits and drawbacks of the selected software architectural styles (Moaven et al., 2009), and how easily the systems can be updated when the need arises.

There are two software architectures that can be effectively used for this development: the **layered system architectural style**, and the **heterogeneous architectural style** (or 3-tier architecture).

1.1 Layered System Architectural Style

The layered system architectural style is a design that allows a program or a system to be organised into distinct, separate layers, with each having a specific role or function

(Xiaodong, 2023c). This style ensures that each layer works together to build an efficient and structured application.

The structure of this style is a call-and-return system, as it allows each single layer to provide a service to the layer above it, while acting as a client service to the layer below it. This means that they interact with their immediate adjacent layer. It uses the concept of abstraction to denote how close or distant a layer is to the low-level system. In addition, each layer can have several components within it, to allow a task that requires additional help to be divided into smaller sub-tasks for its components to work on (see Figure 1).

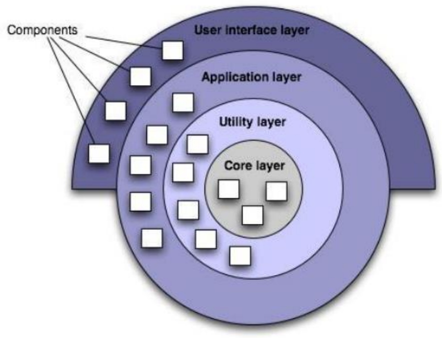


Figure 1. Layered System and components (Xiaodong, 2023c)

The layered system follows a model of enabling a client service to issue a request to its subsequent layer. That subsequent layer can either invoke a service to one of its components, or to the layer below it, to carry out the response of the client service. The communication keeps going until it reaches the lowest layer. The lowest layer will then construct a response and send it back to the layer above, making the communication go in reverse order (see Figure 2 for example).

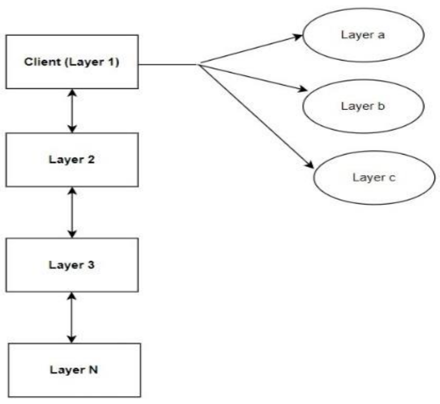


Figure 2. Client with its sub-components - layered system architecture

The following sections will discuss the components and connectors of this architectural style, as well as the protocol for information exchange, and finally, its advantages and disadvantages.

1.1.1 Components

The components of this architecture are the layers, implemented to ensure inter-connected communication in an adjacent manner. Each component is limited to a scope and only deals with the logic relevant to its layer, for example, the presentation layer is responsible for the user-interface of the system; the application/business layer is responsible for handling functional requirements; the infrastructure layer handles the data. This adheres to the separation of concerns concept, as it easily builds effective roles and responsibilities within the architecture.

1.1.2 Connectors

The connectors of this architecture are the procedure calls, which are protocols that determine the interaction between layers in the system. These procedure calls can be data objects, function calls, a query request, or basically any connector that conveys an information or a request.

1.1.3 Protocol for Information Exchange

The protocol for information exchange is briefly described in section 1.1, in the area of the layered system model. The diagram below in Figure 3, essentially shows the client issuing a request to layer N. The links between each component in each layer show the communication and constructive responses that go from the top to the lowest layer, and then goes back up through the layers. Types of protocols include the TCP/IP, ISO OSI Model, etc.

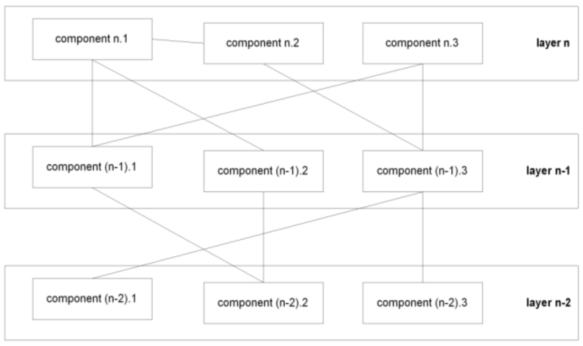


Figure 3. Protocol for Information Exchange (Xiaodong, 2023c)

1.1.4 Advantages and Disadvantages

It is important to consider the benefits of the layered system style, as well as considering the drawbacks it has. Understanding these aspects will help the organisation make an informed decision about the choice of software architecture.

1.1.4.1 Advantages

There are several benefits of using the layered system style to develop the DE-Store system:

- *Modifiable*: the system supports enhancement independence when functions in a layer require changes. This is beneficial for the DE-Store system because it allows updates to be made without affecting the system much.
- *Portable*: each abstract layer is deployed independently to support portability. This is also beneficial for the DE-Store system because it can be moved from one machine to another.
- *Simplified design with abstraction*: the design is based on the incremental levels of abstraction. This is beneficial for the DE-Store system because of the simplicity of implementing the layered structures and their logic.
- *Reusability and economy*: the same layer can be used interchangeably when it has different implementations. This is beneficial for the DE-Store system because components can be reused, and the cost is fairly low.

1.1.4.2 Disadvantages

The layered system style, however, presents some drawbacks:

- *Less efficient*: the client's request or response would have to go through potentially several layers in order to have its request received and acted on. This causes lower runtime performance. This will mean that the DE-Store system could struggle to efficiently provide a response when the company needs an information at a particular moment.
- *Layer partitioning issues*: it causes issues for exception and error handling since faults in one layer tends to spread to other layers. In the case of the DE-Store system, if an error occurs in one of its layers, it can affect the whole system to the point of causing friction for the organisation.
- *Duplication of functionality*: it duplicates functionality, causing data overhead and processing issues.

- *Difficulties in scalability*: it is difficult to scale because the structure of the frameworks (presentation layer, application/business layer, infrastructure layer) do not support growth. DE-Store system requires scalability for future changes and expansion.

1.2 Heterogeneous Architectural Style

The heterogeneous architectural style is a design, or method, that uses more than one architectural style to build a system. In essence, this style can be used for complex systems, as it can combine different architectural styles, leveraging the best approaches in each of them to optimise scalability, functionality, and adaptability.

This style can be classified in two ways: hierarchically heterogeneous and simultaneously heterogeneous. A hierarchically heterogeneous style consists of an architecture that has a component of one style composed of components of another style used in a system (Xiaodong, 2023d). For example, a component of a system may have a part that uses the client/server architectural style, and within this component, another style may be of an object-oriented (OO) architectural style. All this will be in placed in a hierarchical form. This style also often includes a multi-tiered architecture, for instance, the 3-tier architecture consists of a system with multiple layers of architectural design: client, application server, and database (Xiaodong, 2023d).

On the other hand, a simultaneously heterogeneous style consists of a mix of architectural styles applied simultaneously within a system. This means that, for example, a component might access a repository through its interface, while simultaneously interacting through pipes with another part of its interface (Andrade & Crnkovic, 2019).

As shown in Figure 4, the structural style uses the 3-tier architecture which includes the client, application server, and database server. The first layer is the client, also known as the top-tier, and it comes from the client/service (c/s) architecture. It is used as the user interface of an application (e.g., web browser). The c/s focuses on presenting an interface to the user, instead of showing the backend of an application (e.g., database, code, etc). The second layer is the application server, also known as the middle layer, and this focuses on the business logic and data processing of a system. It stands between the client and the database because it ensures that data is passed across. Finally, the database layer, or bottom-tier, maintains the data of the application. It comes from the data-centred architecture, and runs separately in another server to allow the data processing logic in the application server run smoothly to implement load balancing.

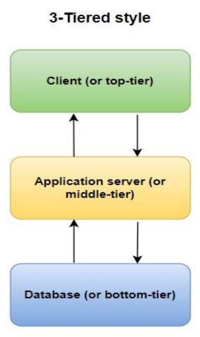

3-Tiered style
Client (or top-tier)
Application server (or middle-tier)
Database (or bottom-tier)

***Figure 4. 3-Tier style** diagram inspired by* Xiaodong (2023b)

It can be used to address the challenges that the DE-Store system requires, as it can manage different architectural components of the system in a structured manner. For instance, a category that focuses on obtaining information of past purchases from clients will be placed in a database storage, while displaying information of prices of items will be handled by the business logic of the software.

The heterogeneous style has components and connectors in its architectural structure. The following sections will discuss these components and connectors. Furthermore, they will explore the protocols used for information exchange, and advantages and disadvantages of the architecture.

1.2.1 Components

The components of this architecture are the different architectural styles used within a system. In the case of the 3-tier architecture, different architectural styles (client-server, data-centred) are used to develop an application.

1.2.2 Connectors

The connectors of this architecture use communicating processes or communication networks to enable different components to communicate with each other. The communicating processes consist of components exchanging information through a messaging service (Xiaodong, 2023b). This messaging service passes data between components and sometimes, if not taken with caution, the data may be lost or delayed. Hence, it is advisable that message transfer error detection and correction is built into the system, right from the start (Xiaodong, 2023b).

1.2.3 Protocol for Information Exchange

Components run concurrently, meaning that they either run on the same machine or not. To be able to exchange information, they use protocols. These protocols are communicating processes like TCP/IP, HTTP, UDP, etc. For example, TCP/IP is the transport control protocol and internet protocol that provides a means on how data can be transferred (see Figure 5). It is used to connect the Internet with other applications, through interconnected networks (Xiaodong, 2023b).

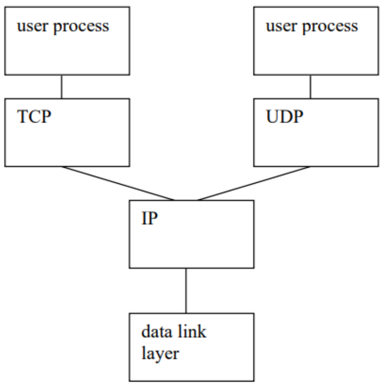


Figure 5. TCP/IP - Information Exchange (Xiaodong, 2023b)

1.2.4 Advantages and Disadvantages

This section will consider both the advantages and disadvantages of the heterogeneous architectural style, particularly the 3-tier style.

1.2.4.1 Advantages

There are a number of advantages as to why the 3-tier architectural style is also an optimal choice to develop the DE-Store system, here a few to mention:

- *Maintainability*: it is easy to maintain and modify, without affecting the system much. As this is one of the requirements laid out by the organisation, DE-Store can benefit from using this architecture because of the ease in maintaining the system.
- *Performance*: it is easier and faster since the business logic of the system is implemented on a centralised server. In addition, the top-tier in the 3-tier architecture minimises the load balance because it caches requests.
- *Parallel task division and processing speed*: different processors work in parallel to accelerate the time it takes to execute tasks. This is particularly useful for the DE-Store system.
- *Economy*: it is less expensive because of the ‘thin’ client.

- *Scalability*: application servers can be deployed on many machines, making it easy for the database server to no longer make longer connections with client servers. This improves scalability in the DE-Store system.
- *Security and Reliability*: It improves security by preventing the client from directly accessing data from the database. It also proves to be reliable because of data integrity. Therefore, DE-Store system can ensure that its retail branches and users' data are kept safe.
- *Reusability*: this architecture promotes reusability.

1.2.4.2 Disadvantages

However, the 3-tier architectural style also has some drawbacks. Here are the main disadvantages of this style:

- *Handling of unreliable communication*: the messaging service can sometimes lose data or be delayed.
- *Less flexible*: if application servers that contain the business logic, and database servers that contain data are disconnected from each other, it affects the performance of the entire system.

1.3 SUMMARY

In summary, both architectural styles present a possibility of being used to develop the DE-Store system. We have seen and evaluated the benefits and drawbacks, as well as understanding how they can be applied to the system. Nevertheless, one of these architectural styles presents itself as the optimal choice due to its alignment with the requirements laid out by the business. The subsequent chapter will delve into the chosen architectural design, justifying the reason of choice by providing an overview of the quality attributes.

2 | SELECTION & JUSTIFICATION OF ARCHITECTURE

Having considered the architectures discussed in the previous chapter, including their benefits and drawbacks, this chapter will focus on the selection and justification of the architecture best suited for the company, by using the software architecture analysis method (SAAM).

2.1 REVIEW OF QUALITY ATTRIBUTES

This section will be looking at the quality attributes for the system by using SAAM. SAAM is a method used to analyse software architecture against a set of requirements, by using a scenario based analysis undertaken as part of a review process. The following subsections will consider six SAAM steps for five quality attributes: scalability, usability, maintainability, optimisation, and security.

2.1.1 Develop Scenarios

Scenarios are a beneficial way to consider and understand the perspectives of different stakeholders when using an application. The stakeholders may include users, a system administrator, maintenance engineer, and so on. Scenarios serve as a proposal for what stakeholders might do with a developed system, but also what changes they might want to request (Xiaodong, 2023a). In this context, five scenarios regarding the five quality attributes will be discussed:

System Administrator:

- ***Scalability***: enabling system expansion by adding additional classes, objects, information in the system, for the growth of the business.
- ***Security***: updating the security protocols of the system to avoid data leaks.

Maintenance Engineer:

- ***Maintainability***: monitoring and maintaining the security of the system.
- ***Optimisation***: improving the performance of the system, and ensuring that users have a smooth experience when navigating the system.

User:

- ***Usability***: ensuring the system is user-friendly, allowing easy navigation of the items presented in the system. Additionally, implement accessibility features, especially for those with disabilities.

2.1.2 Describe Candidate Architectures

Two architectures will be compared based on the given scenarios to evaluate which of them is more or less useful to use.

Scalability

Layered system architectural style: this architecture does not support scalability because the structure of its frameworks do not allow room for growth. Basically, if any changes are to be made in any of its layers, it can affect the rest of its layers.

3-Tier architectural style: this architecture supports scalability because each tier can be scaled independently. However, it is important to consider what changes are done in the system, as it could sometimes lead to issues with the protocol of information exchange among components.

Security

Layered system architectural style: it is secured because layers are given specific functions on what data they process. However, caution should be taken, as any wrong exchange of information between layers can pose security risks to the system.

3-Tier architectural style: it is secured because it is designed to prevent the client from accessing unauthorised data. In addition, this helps in preventing malicious exploits in the database (IBM, n.d.).

Maintainability

Layered system architectural style: it is maintainable because it supports separation of concerns, which means it is able to manage complexities by partitioning each layer within a system.

3-Tier architectural style: it is also maintainable as it makes it easy for software architects to modify individual layers, without affecting the system much.

Optimisation

Layered system architectural style: not ideal as partitioning layers causes it to be prone to error and exception handling issues.

3-Tier architectural style: faster in performance making it an optimisable architecture.

Usability

Layered system architectural style: it is a good choice for maintaining modularity for a system.

3-Tier architectural style: since it separates the user-interface from the server platforms, it makes it easy for the user to focus on what is shown to them on an interface.

2.1.3 Classify Scenarios and Perform Scenario Evaluations

Scenarios can be classified as direct or indirect, based on the necessary changes in components and connectors required to support the scenarios (Xiaodong, 2023a). Direct scenario refers to the scenario that requires no architectural modifications, whilst indirect scenario requires modifications, specifically to the components and connectors. The most favourable scenario is one that requires little to no modifications in the architecture, while the worst scenario would mean that it demands a lot of changes (see Table 1 for reference). Evaluation of indirect scenarios will be employed using the performance scenario evaluation. It will consider the changes required, as well as the cost of implementing them (also see Table 1 and Table 2 for reference).

Architecture	Scenario	Direct (D) / Indirect (I)
Layered System	1	I
	2	D
	3	D
	4	D
	5	I
3-Tier	1	I
	2	D
	3	D
	4	D
	5	I

Table 1. Summary Table of Scenarios

2.1.4 Reveal Scenario Interaction

The goal in revealing scenario interactions is to analyse the impact of different indirect scenarios. It is a process that identifies scenarios that affect a common set of components, and measures the extent to which the architecture supports an appropriate separation of concerns (Kazman et al., 1996; Xiaodong, 2023a). Below, Table 2 shows the cost of each indirect scenarios.

Architecture	Scenario	Indirect (I)	Cost
Layered System	1	I	Between 7-10 days of implementation and testing the functionality of the system - programmer
	5	I	10 days - programmer 3 days – users test UI
3-Tier	1	I	Between 5-7 days of implementation and testing - programmer
	5	I	5-10 days - programmer 3 days – users test UI

Table 2. Reveal Scenario Interactions

2.1.5 Overall Evaluation

Candidate architectures will be ranked based on each scenario and interaction. The overall evaluation will use zero, plus, and minus, as a way of evaluating each architecture in respect to the scenario. Zero (0) indicates that the architecture is neither superior nor inferior with respect to the scenario; plus (+) indicates that it is superior; minus (-) indicates that it is inferior (see Table 3 for reference).

Architecture	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Summary
Layered System	-	0	0	-	0	-
3-Tier	+	0	0	+	0	+

Table 3. Overall Evaluation of candidate architectures

2.2 SUMMARY

We have reviewed and analysed both architectural styles using the SAAM method. Based on the overall evaluation of the candidate architectures, the 3-tier architectural style stands out to be the optimal architecture for the DE-Store system. It promotes scalability, maintenance, optimisation, usability, and security for a system. The next chapter will focus on the design and implementation aspect proposed for the DE-Store system, considering the functionalities laid out by the nationwide retailing corporation.

3 | DESIGN & IMPLEMENTATION

This chapter focuses on the design and implementation of the selected software architecture, that is, the 3-tier architectural style. It will discuss the architectural design and the prototype developed for the DE-Store system, outlining the components and connectors.

3.1 UNIFIED MODELLING LANGUAGE (UML) AND DESIGN PROTOTYPE

To help visualise the software system, the UML diagram as well as the design prototype for the DE-Store systems are shown below in Figures 6 and 7:

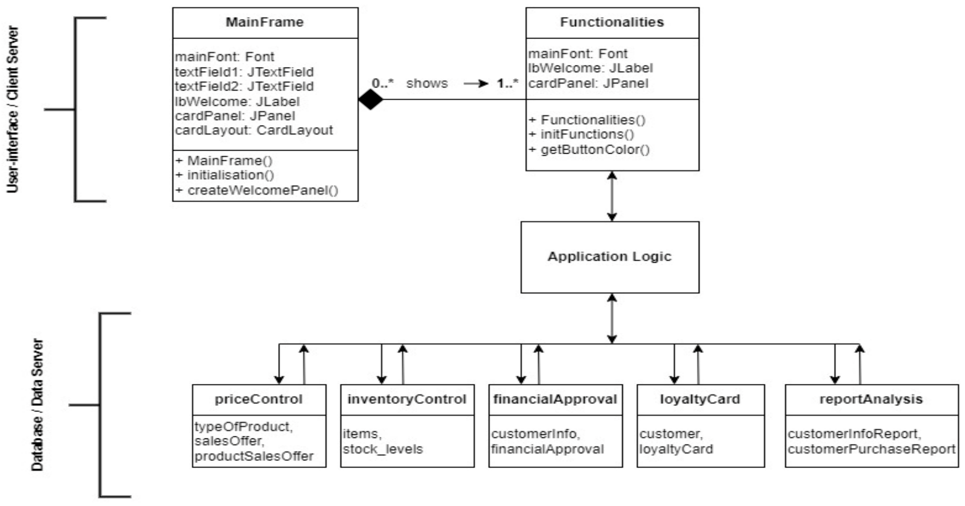


Figure 6. UML

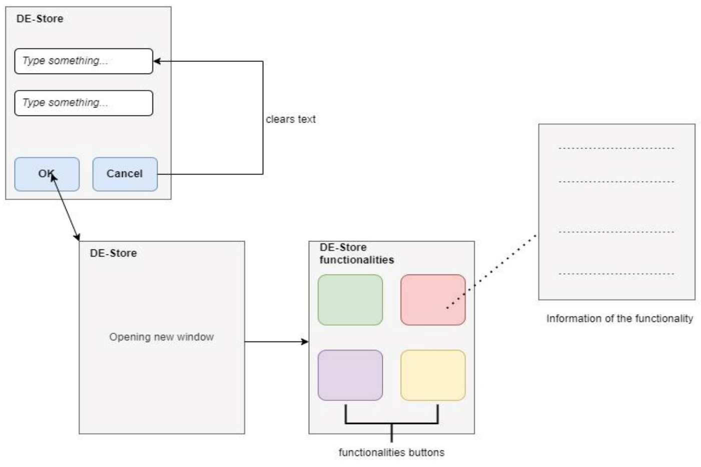


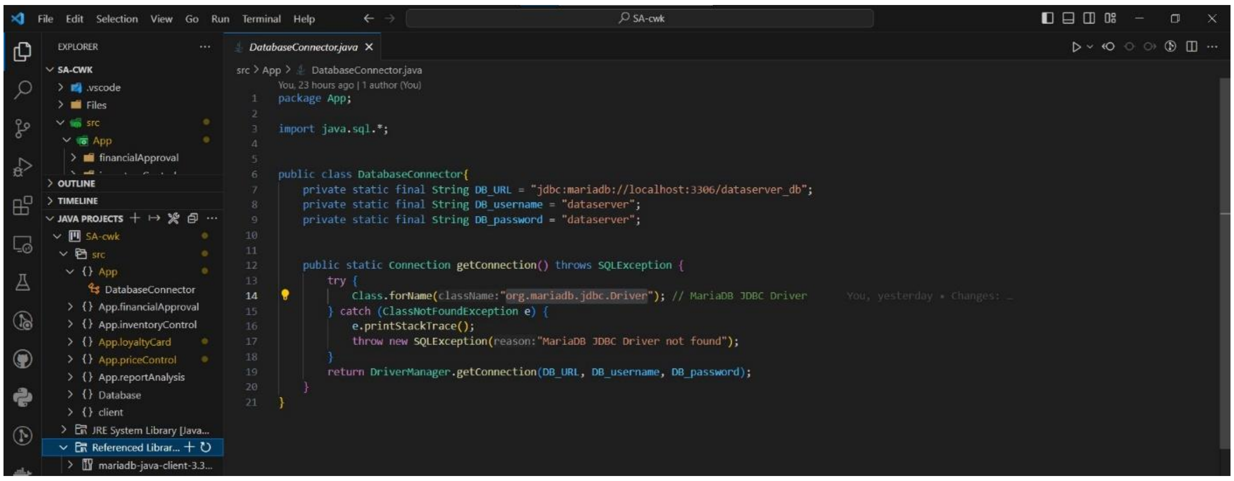
Figure 7. Design Prototype

3.2 TECHNOLOGY, FRAMEWORK AND SET-UP

The DE-Store system was developed using VSCode editor, MariaDB for database, Java as the main programming language, Java Frame (JavaFX) for GUI of the software, and JDBC driver connector for connecting MariaDB with Java applications.

To set-up these things, Java extension and debugging packs in the ‘Extensions’ section in VSCode need to be installed. For the database, SQLTools will also need to be installed for the necessary databases any individual would want to use (like MariaDB, MySQL, etc). For the purpose of the prototype development, MariaDB was chosen for its simplicity for data, however, other SQL services can offer similar approach depending on the experience of an individual.

To connect components and connectors in the software, JDBC (Java Database Connectivity) driver connector is used for message passing between a database server and a Java application. Finally, to ensure that the driver connector works properly, the MariaDB connector/J .jar file must be installed and referenced to a library (see Figure 8 for code).

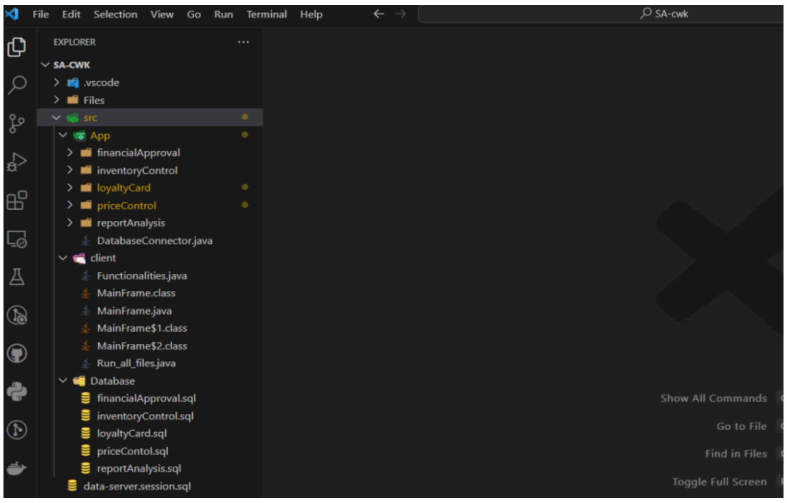


```java
package App;

import java.sql.*;


public class DatabaseConnector{
    private static final String DB_URL = "jdbc:mariadb://localhost:3306/dataserver_db";
    private static final String DB_username = "dataserver";
    private static final String DB_password = "dataserver";


    public static Connection getConnection() throws SQLException {
        try {
            Class.forName(className:"org.mariadb.jdbc.Driver"); // MariaDB JDBC Driver      You, yesterday • Changes: …
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            throw new SQLException(reason:"MariaDB JDBC Driver not found");
        }
        return DriverManager.getConnection(DB_URL, DB_username, DB_password);
    }
}
```

Figure 8. Database Connector and Reference Library

In Java, when connecting the database and the .jar file in the code, use **“jdbc:mariadb://localhost:3306/name_of_your_database”** and **“org.mariadb.jdbc.Driver”**.

The code is structured around three integral tiers: client, application server, and database. Each tier is distinct with a purpose and a responsibility, yet interacts with other tiers. The source file has these three tiers, and within each tier there are files, some for user-interface, some for business logic, while others for SQL statements like creation of tables, etc (see Figure 9).



```
EXPLORER
SA-CWK
  .vscode
  Files
  src
    App
      financialApproval
      inventoryControl
      loyaltyCard
      priceControl
      reportAnalysis
      DatabaseConnector.java
    client
      Functionalities.java
      MainFrame.class
      MainFrame.java
      MainFrame$1.class
      MainFrame$2.class
      Run_all_files.java
    Database
      financialApproval.sql
      inventoryControl.sql
      loyaltyCard.sql
      priceContol.sql
      reportAnalysis.sql
    data-server.session.sql
```

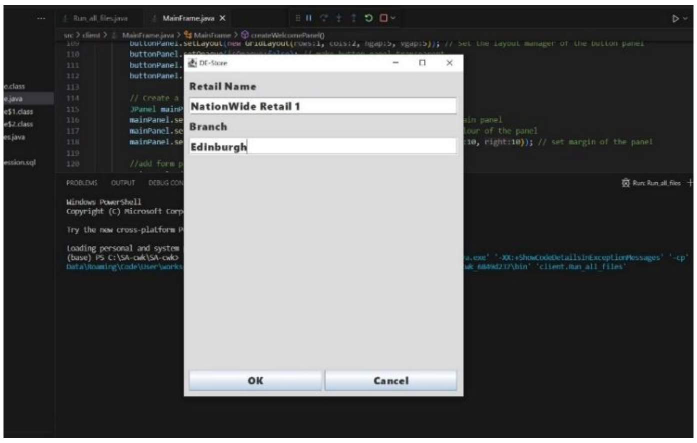
Figure 9. VSCode - code structure

3.3 PURPOSE AND RESPONSIBILITIES

<u>Client Tier</u>: the client tier present within the DE-Store system serves as the GUI (user-interface) of the system. It is responsible for showing a user each functionality present within the system, and getting information from them. The framework used to develop this tier was by using the Java Frame GUI (JavaFX) to craft the interface.

There are 2 classes included within this client tier: MainFrame.java and Functionalities.java. The MainFrame focuses on having the user type in a branch and location of one of the retail branches of the corporation; once confirmed, a window opens up to show 5 functionality tabs (see Figure 10 and 11).

This interface is designed to be simple and intuitive for anyone to use, as well as providing the exact information they need. It also displays information in the console when running the Run_all_files.java file (see Figure 12).



```
DE-Store
Retail Name
NationWide Retail 1
Branch
Edinburgh
```

Figure 10. DE-Store - Main Frame

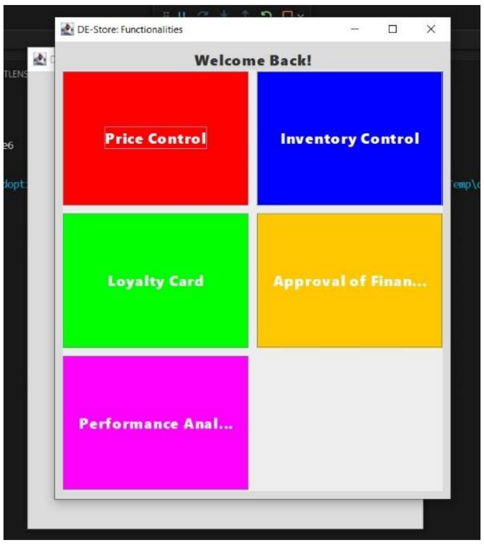
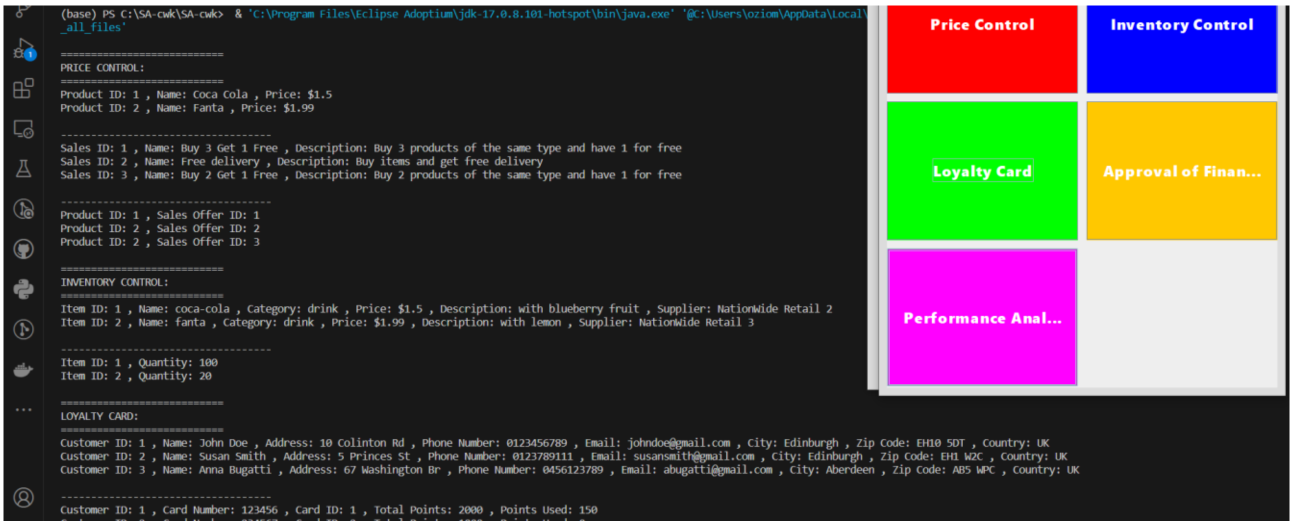


Figure 11. DE-Store - Functionalities

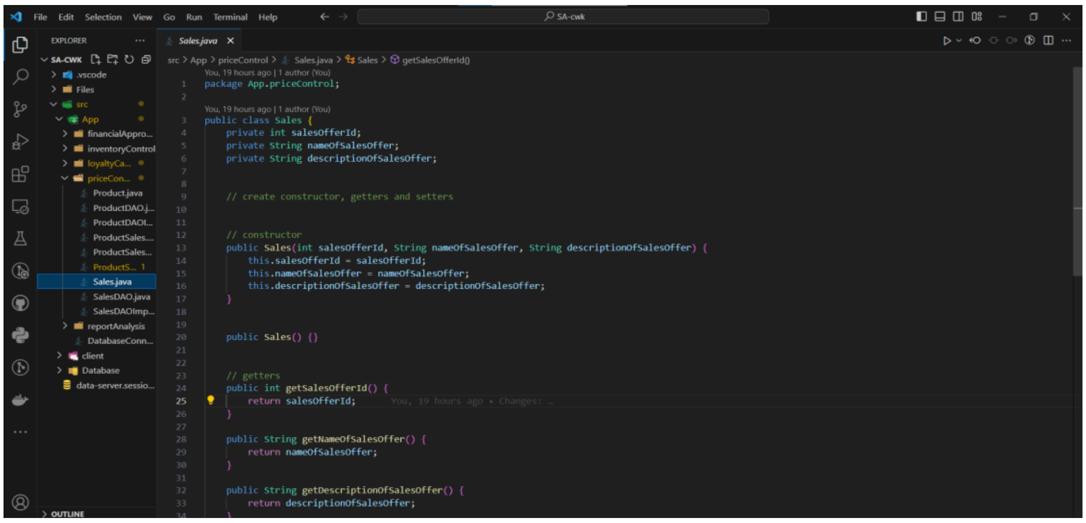


```
(base) PS C:\SA-cwk\SA-cwk>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.8.101-hotspot\bin\java.exe' '@C:\Users\oziom\AppData\Local\
_all_files'

===========================
PRICE CONTROL:
===========================
Product ID: 1 , Name: Coca Cola , Price: $1.5
Product ID: 2 , Name: Fanta , Price: $1.99

-----------------------------------
Sales ID: 1 , Name: Buy 3 Get 1 Free , Description: Buy 3 products of the same type and have 1 for free
Sales ID: 2 , Name: Free delivery , Description: Buy items and get free delivery
Sales ID: 3 , Name: Buy 2 Get 1 Free , Description: Buy 2 products of the same type and have 1 for free

-----------------------------------
Product ID: 1 , Sales Offer ID: 1
Product ID: 2 , Sales Offer ID: 2
Product ID: 2 , Sales Offer ID: 3

===========================
INVENTORY CONTROL:
===========================
Item ID: 1 , Name: coca-cola , Category: drink , Price: $1.5 , Description: with blueberry fruit , Supplier: NationWide Retail 2
Item ID: 2 , Name: fanta , Category: drink , Price: $1.99 , Description: with lemon , Supplier: NationWide Retail 3

-----------------------------------
Item ID: 1 , Quantity: 100
Item ID: 2 , Quantity: 20

===========================
LOYALTY CARD:
===========================
Customer ID: 1 , Name: John Doe , Address: 10 Colinton Rd , Phone Number: 0123456789 , Email: johndoe@gmail.com , City: Edinburgh , Zip Code: EH10 5DT , Country: UK
Customer ID: 2 , Name: Susan Smith , Address: 5 Princes St , Phone Number: 0123789111 , Email: susansmith@gmail.com , City: Edinburgh , Zip Code: EH1 W2C , Country: UK
Customer ID: 3 , Name: Anna Bugatti , Address: 67 Washington Br , Phone Number: 0456123789 , Email: abugatti@gmail.com , City: Aberdeen , Zip Code: AB5 WPC , Country: UK

-----------------------------------
Customer ID: 1 , Card Number: 123456 , Card ID: 1 , Total Points: 2000 , Points Used: 150
```

Figure 12. Results

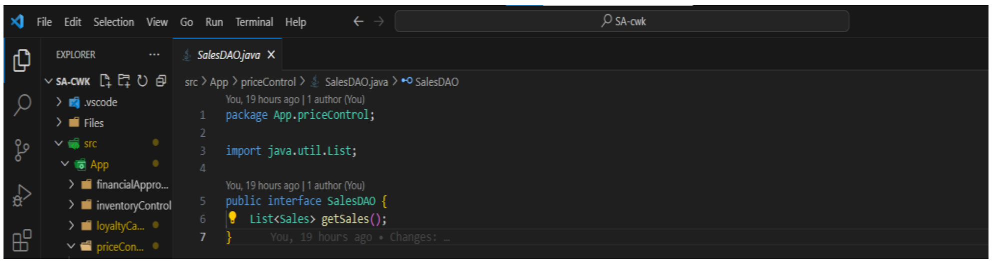
Application Tier (or App Tier): is the second layer of the DE-Store system, and focuses on processing user request. For example, when a user selects the 'price control' tab, this tier processes the request by retrieving data from the database, and sending a message back to the client server by displaying the information (see Figures 13, 14, and 15 for sample code). This application uses Data Access Object as a design pattern to separate logic of data into separate layers. For example, sales is divided into three layers: Sales.java encapsulates attributes related to sales offers; SalesDAO.java represents the interface; and SalesDAOImpl.java implements the SalesDAO interface and interacts with a database to retrieve data and/or insert new data.

The framework used to develop the application is using JDBC. JDBC is an API implementation that connects components of the DE-Store system, executes queries and update database. More on this in section 3.4.



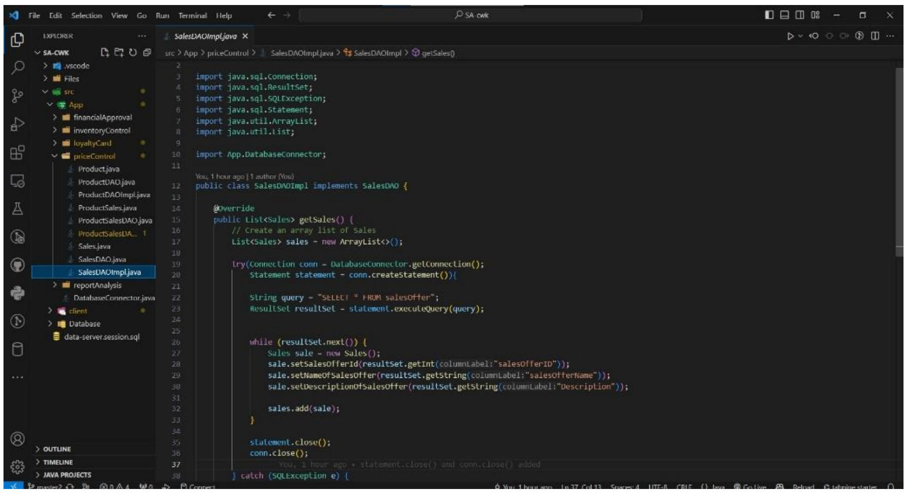
```java
package App.priceControl;

public class Sales {
    private int salesOfferId;
    private String nameOfSalesOffer;
    private String descriptionOfSalesOffer;


    // create constructor, getters and setters


    // constructor
    public Sales(int salesOfferId, String nameOfSalesOffer, String descriptionOfSalesOffer) {
        this.salesOfferId = salesOfferId;
        this.nameOfSalesOffer = nameOfSalesOffer;
        this.descriptionOfSalesOffer = descriptionOfSalesOffer;
    }


    public Sales() {}


    // getters
    public int getSalesOfferId() {
        return salesOfferId;
    }

    public String getNameOfSalesOffer() {
        return nameOfSalesOffer;
    }

    public String getDescriptionOfSalesOffer() {
        return descriptionOfSalesOffer;
```

Figure 13. Sales.java



```java
package App.priceControl;

import java.util.List;

public interface SalesDAO {
    List<Sales> getSales();
}
```

Figure 14. SalesDAO.java

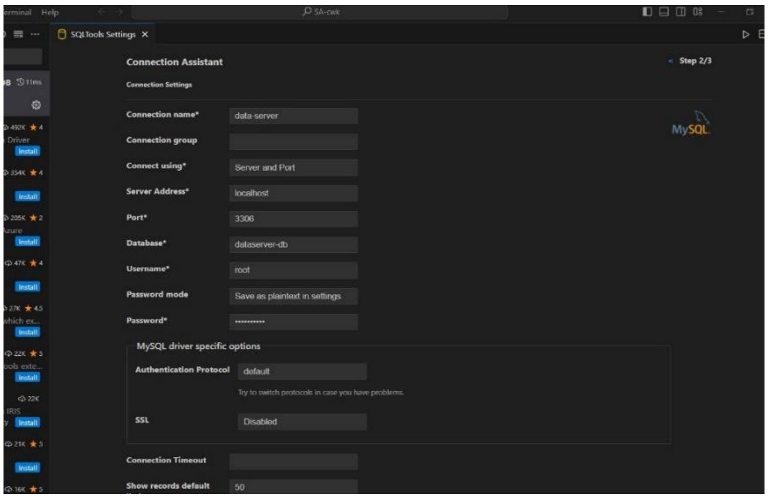


```java

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import App.DatabaseConnector;

public class SalesDAOImpl implements SalesDAO {

    @Override
    public List<Sales> getSales() {
        // Create an array list of Sales
        List<Sales> sales = new ArrayList<>();

        try(Connection conn = DatabaseConnector.getConnection();
            Statement statement = conn.createStatement()){

            String query = "SELECT * FROM salesOffer";
            ResultSet resultSet = statement.executeQuery(query);


            while (resultSet.next()) {
                Sales sale = new Sales();
                sale.setSalesOfferId(resultSet.getInt(columnLabel:"salesOfferID"));
                sale.setNameOfSalesOffer(resultSet.getString(columnLabel:"salesOfferName"));
                sale.setDescriptionOfSalesOffer(resultSet.getString(columnLabel:"Description"));

                sales.add(sale);
            }

            statement.close();
            conn.close();

        } catch (SQLException e) {
```

Figure 15. SalesDAOImpl.java

Database: this tier focuses on storing data of management functionalities, as well as keeping customers' information and purchases. MariaDB was the database used in VS Code. The process done to connect it to the service provider is by installing the SQLTools extension in the editor, and set up a database connection (see Figure 16). To ensure that the connection is successful, have the MariaDB server set up prior to connecting it (see Figure 17). Once connected, it confirms that it is added to the editor's settings and now is available for use.

Figure 18 shows the tables available in the database server, including key data (e.g. IDs, customerInfo, etc); while Figure 19 shows a customer table.



Connection Assistant

Connection Settings

Connection name*	data-server
Connection group	
Connect using*	Server and Port
Server Address*	localhost
Port*	3306
Database*	dataserver-db
Username*	root
Password mode	Save as plaintext in settings
Password*	••••••••••

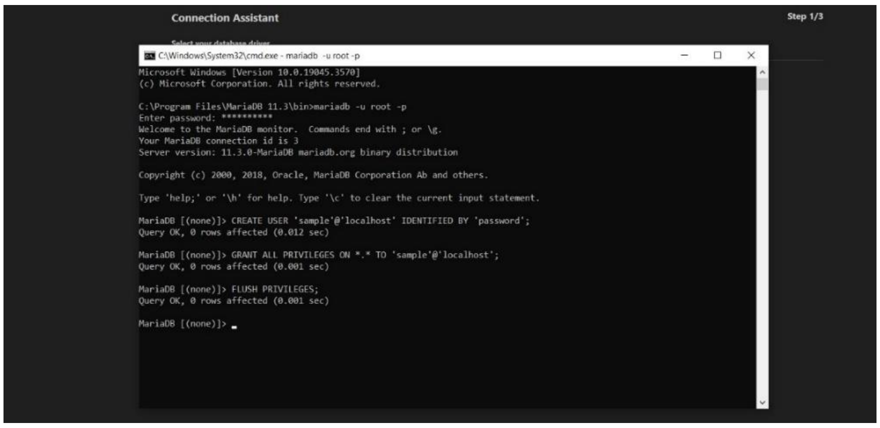
MySQL driver specific options

Authentication Protocol	default
SSL	Disabled

Try to switch protocols in case you have problems.

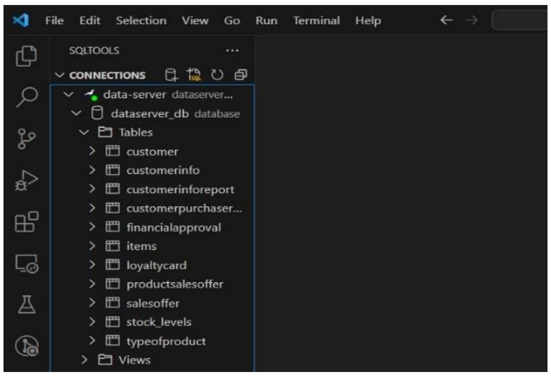
Connection Timeout	
Show records default	50

Figure 16. Setting up database



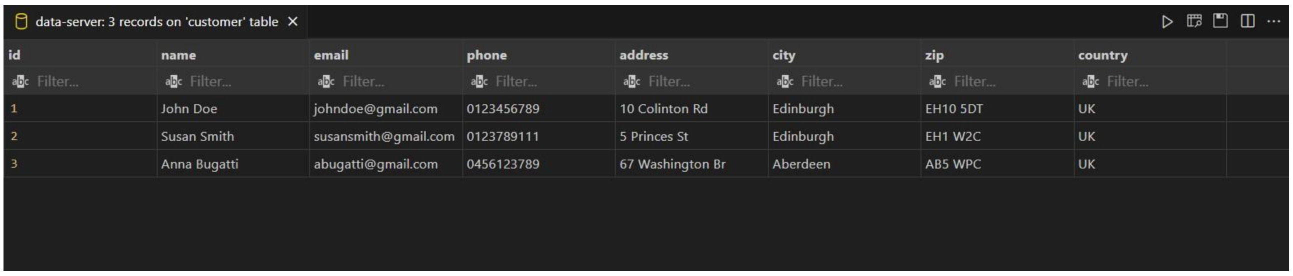
```
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\MariaDB 11.3\bin>mariadb -u root -p
Enter password: **********
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 11.3.0-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE USER 'sample'@'localhost' IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.012 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'sample'@'localhost';
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]> _
```

Figure 17. Connecting MariaDB



```
data-server dataserver...
  dataserver_db database
    Tables
      customer
      customerinfo
      customerinforeport
      customerpurchaser...
      financialapproval
      items
      loyaltycard
      productsalesoffer
      salesoffer
      stock_levels
      typeofproduct
    Views
```

Figure 18. Tables in database



data-server: 3 records on 'customer' table

id	name	email	phone	address	city	zip	country
1	John Doe	johndoe@gmail.com	0123456789	10 Colinton Rd	Edinburgh	EH10 5DT	UK
2	Susan Smith	susansmith@gmail.com	0123789111	5 Princes St	Edinburgh	EH1 W2C	UK
3	Anna Bugatti	abugatti@gmail.com	0456123789	67 Washington Br	Aberdeen	AB5 WPC	UK

Figure 19. 'Customer' table

3.4 Components, Connectors and Protocol

Previous section discussed on the tiers of this system following the 3-tier architectural style. These tiers present components, connectors and protocols. For the main components of the client service, the MainFrame.java and Functionalities.java files are the GUI component of this architecture style (or system). For the application logic, components follow the separation of concerns concept and are broken down into layers, each processing logic for the client service and data. As for the database, the components are tables created in the system and its management of functionalities.

The communication between these components is facilitated by JDBC API. This API enhances the communication and collaboration amongst each other to allow smooth integration, interaction and data flow. The client interacts with the application tier, that is, users send their requests for a particular information on a product. The application tier interacts with the database tier to retrieve data and send back information to the user. In other words, the application tier is responsible for connecting these two tiers since it handles the data requests and implements functions to enable both sides to have the desired outcomes.

3.5 Measures

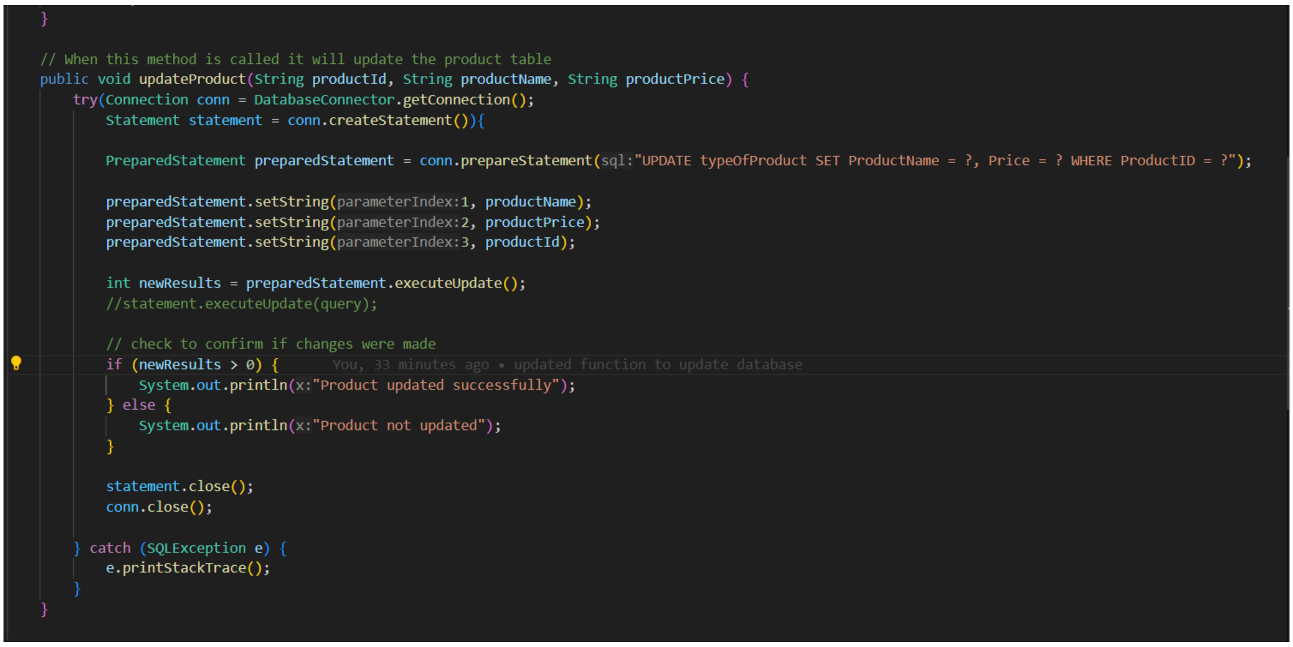
Scalability: the DE-Store system design allows room for scalability because of the division of layers within it. Thanks to the JDBC API, these layers work independently, hence, they allow architects to be able to make updates when needed without affecting the entire system.

Security: the system is secured since users cannot access nor insert data without permission. When new data is to be added in the system, the JDBC API uses a secure PreparedStatement object to provide security in the SQL data. For example, in a method found in ProductDAOImpl.java, contains the object that securely updates the new data inserted into the product table (see Figure 20).

Usability: the system promotes usability for users as the design is built to be intuitive, simple to use and understand, and perform actions.

Maintenance: the system clearly shows that it is easy to maintain as each component is structured in layers, making it easy to work with, change or update it, and maintain it.

Optimization: optimisation is applied in the system, i.e., the time taken to process and retrieve information is fast. In addition, the code structure is well-organised boosting the system's performance and efficiency.



```java
    }

    // When this method is called it will update the product table
    public void updateProduct(String productId, String productName, String productPrice) {
        try(Connection conn = DatabaseConnector.getConnection();
            Statement statement = conn.createStatement()){

            PreparedStatement preparedStatement = conn.prepareStatement(sql:"UPDATE typeOfProduct SET ProductName = ?, Price = ? WHERE ProductID = ?");

            preparedStatement.setString(parameterIndex:1, productName);
            preparedStatement.setString(parameterIndex:2, productPrice);
            preparedStatement.setString(parameterIndex:3, productId);

            int newResults = preparedStatement.executeUpdate();
            //statement.executeUpdate(query);

            // check to confirm if changes were made
            if (newResults > 0) {       You, 33 minutes ago • updated function to update database
                System.out.println(x:"Product updated successfully");
            } else {
                System.out.println(x:"Product not updated");
            }

            statement.close();
            conn.close();

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Figure 20. PreparedStatement object in SQL

4 | EVALUATION

This final chapter will critically evaluate the work undertaken in the development of the DE-Store system, and provide future work.

4.1 CRITICAL EVALUATION

The five quality attributes used to develop the DE-Store system were scalability, security, usability, optimisation and maintenance. These attributes have helped in achieving the goals by:

- Ensuring that the system accommodates future scalability without affecting its overall performance.
- Implementing security measures throughout the system using reliable API to pass data to the right components.
- Maintaining and keeping the system in compliance with the corporation's requirements.
- Designing a user interface that is intuitive and simple enough to enable a user to use.
- Enhancing the optimisation of the system by having it process and retrieve information quickly.

Also, the DE-Store system followed the UML and design prototype outlined in the previous chapter, making this a successful application.

Moreover, it demonstrates that the 3-tier architectural style is indeed the optimal architecture to use for developing the DE-Store system. It provides several benefits for the system and will help the nationwide retailing corporation provide better coordination of their business for their retail branches.

Despite these achievements, however, one of the challenges faced during the development was in dealing with the configurations required to test the system, in this case, using MariaDB's query logging configuration to log every SQL query received from a client server. It requires gaining permission from the operating system to effectively have the query log run successfully, and if not done properly it could cause and impose security risk to the system.

4.2 Future Work

Overall, the project benefits from further refinements and enhancements as it demonstrates capabilities in managing data, possibilities in expansion of the system to incorporate new technologies.

Regarding the design of the system, DE-Store can benefit from not only querying data but also updating and/or creating new ones. It can be further developed to have interfaces that nicely outline information requested from the client instead of displaying them in the console.

Finally, nationwide retailing corporation can create a list of its retail branches and provide each with a login mechanism or system tailored to their specific requirements. For example, a retail branch focused on selling food products would need to know how many items they sold, while another branch focused on selling beverages would need to know the quantity of items in stock and out-of-stock. The company should make the system in such a way that each retail can have their own customised system within the DE-Store system, where they can access necessary information they would need.

REFERENCES

Andrade, H., & Crnkovic, I. (2019). *A Review on Software Architectures for Heterogeneous Platforms*. http://arxiv.org/abs/1905.01695

Architecture Styles in Distributed Systems - GeeksforGeeks. (n.d.). Retrieved November 13, 2023, from https://www.geeksforgeeks.org/architecture-styles-in-distributed-systems/

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). *Scenario-Based Analysis of Software Architecture*.

Moaven, S., Kamandi, A., Habibi, J., & Ahmadi, H. (2009). Toward a framework for evaluating heterogeneous architecture styles. *Proceedings - 2009 1st Asian Conference on Intelligent Information and Database Systems, ACIIDS 2009*, 155–160. https://doi.org/10.1109/ACIIDS.2009.68

What is a distributed system? | Atlassian. (n.d.). Retrieved November 13, 2023, from https://www.atlassian.com/microservices/microservices-architecture/distributed-architecture

What is Three-Tier Architecture | IBM. (n.d.). Retrieved November 17, 2023, from https://www.ibm.com/topics/three-tier-architecture

Xiaodong, L. (2023a). *L10a ADL ATAM*.

Xiaodong, L. (2023b). *SET10101 SA Unit 3: Communicating Processes and Implicit Invocation Unit 3: Communicating Processes and Implicit Invocation Part I Communicating Processes*.

Xiaodong, L. (2023c). *SET10101 SA Unit 4: Layered Systems and Data Centred Style Unit 4: Layered Systems and Data Centred Style*.

Xiaodong, L. (2023d). *SET10101 SA Unit 5: 3-Tiered Style and Aspect-Oriented Architecture Unit 5: Heterogeneous Style (3-Tiered) and Aspect-Oriented Architecture*.