Francis Pollock

40618059  SET10101

Software architecture

Contents

[1. Recommendation of Architectures 2](#_Toc152101285)

[1.1. Aspect Oriented Software Architecture 2](#_Toc152101286)

[Context: What is Aspect Oriented Software Architecture? 2](#_Toc152101287)

[1.1.1. Components and Connectors 2](#_Toc152101288)

[1.2. Three-Layered Architecture 3](#_Toc152101289)

[Context: What is Three-Layered Architecture? 3](#_Toc152101290)

[1.2.1. Components and Connectors 3](#_Toc152101291)

[2. Architecture Decision and Justification 4](#_Toc152101292)

[3. Design 4](#_Toc152101293)

[UML Diagram 5](#_Toc152101294)

[Activity Diagram 8](#_Toc152101295)

[Class Diagram 10](#_Toc152101296)

[MoSCoW 11](#_Toc152101297)

[Tools Used 11](#_Toc152101298)

[4. Evaluation 12](#_Toc152101299)

[4.1. MoSCoW Evaluation 12](#_Toc152101300)

[4.2. Future Iteration 13](#_Toc152101301)

[Testing: 14](#_Toc152101302)

# Recommendation of Architectures

## Aspect Oriented Software Architecture

### Context: What is Aspect Oriented Software Architecture?

Aspect-Oriented Software Architecture (AOSA) is a method used in software design to organize tricky, wide-reaching tasks—like logging, security, and error handling—into separate sections called "aspects." These tasks, known as cross-cutting concerns, tend to affect multiple parts of a system and are tough to fit neatly into just one section.

The whole idea behind AOSA is to pinpoint these cross-cutting concerns and keep them apart, making it possible to add them into existing sections without tinkering with their code. This way, different parts of the system stay tidier and easier to manage, making it simpler to keep things working smoothly and making it easier to reuse parts of the system in other places.

The components in AOSA are the individual modules of the system, such as the store management, loyalty card, finance approval, and reporting and analysis module. The connectors in AOSA include aspect weaving, which connects the aspects to the target modules, and remote procedure call (RPC) for communication with external systems.

## Components and Connectors

*Components*:

* Store Management Module: Responsible for managing store functionalities such as price control, inventory control, delivery charge, and performance analysis.
* Loyalty Card Module: Handles the special offers and discounts for customers who regularly use the store branches.
* Finance Approval Module: Integrates with the Enabling finance system via a portal to handle the buy now, pay later functionality.
* Reporting and Analysis Module: Tracks and analyses the purchase activities of customers from the accounting database.

*Connectors*:

* Aspect Weaving: This connector is used to weave cross-cutting concerns into the different modules.
* Remote Procedure Call (RPC): This connector is used for communication between the modules and the Enabling finance system via the portal.

## Three-Layered Architecture

### Context: What is Three-Layered Architecture?

The Three-Layered Architecture is a blueprint for organizing software into three main parts: the presentation layer, business logic layer, and data access layer.

The presentation layer is all about how users interact with the application. It's in charge of showing information and taking in what users’ input. Think of it as the face of the application, whether it's a graphical interface or a web setup.

The business logic layer, or application layer, holds the brains of the operation. It's where the core functions and rules live. When a user asks for something in the presentation layer, this layer figures out how to make it happen. It handles stuff like making sure data makes sense, doing calculations, and running the logical bits. This layer acts as the go-between for the presentation and data access layers.

Then there's the data access layer—it's all about dealing with where data lives and how to get it. This layer talks to databases or other systems to handle tasks like creating, reading, updating, and deleting data. It often makes things easier for the application layer by hiding the nitty-gritty details of where the data's stored, so the application layer can just focus on getting what it needs without getting bogged down in the specifics.

## 1.2.1. Components and Connectors

*Components*:

* Presentation Layer: Handles the user interface and user interaction. It includes components like web pages, GUI forms, or mobile app interfaces.
* Application Layer: Contains the business logic and processes user requests. It coordinates and manages the application's functionality.
* Data Layer: Manages the storage and retrieval of data. It includes databases or external systems that store and provide access to the application's data.

*Connectors*:

* API or Web Services: The presentation layer communicates with the application layer through APIs or web services. This allows the user interface to send requests and receive responses from the application layer. In the case of this project, it will be done through MySQL connections through a Java Driver.
* Database Connections: The application layer interacts with the data layer by establishing connections to the database. It can perform CRUD operations on the data, retrieve information, or update records.

# Architecture Decision and Justification

The architecture that I have decided to go with will be Three-Layered Architecture. I believe that is suits the needs of the developed system best and has quality attributes that directed the choice towards this architecture. Those qualities are as follows:

* **Modifiability**: Three Layered architecture allows for easier modification and maintenance as each layer is independent of the others. Changes made to one layer do not affect the other layers, making it easier to modify specific functionalities without impacting the entire system.
* **Scalability**: With Three Layered architecture, it is easier to scale the system horizontally by adding more instances of each layer. This allows for better handling of increased user load and improved performance.
* **Reusability**: Three Layered architecture promotes reusability of components. Each layer can be developed independently, and the components within each layer can be reused across different modules or applications, reducing development time and effort.
* **Testability**: Three Layered architecture allows for easier testing of individual layers. The separation of concerns makes it easier to write unit tests for each layer independently, ensuring proper functionality and reducing the chances of bugs.
* **Security**: Three Layered architecture allows for better security as each layer can have its own security mechanisms and access controls. This helps in securing sensitive data and preventing unauthorized access to different layers of the system.
* **Maintainability**: The separation of concerns in Three Layered architecture makes it easier to maintain and update the system. Each layer can be maintained and updated independently, reducing the risk of introducing issues or breaking the entire system during maintenance.

This will lead to a robust and adaptable system, and adaption is something that is required for future iterations for the DE system.

# Design

The design of the system will be done in parts, first with a UML diagram overview of the system, it’s classes and attributes.

Then, an activity diagram will be constructed to show the use case of a normal user or manager respectively.

Third, a class diagram to show the pathways of the application.

Lastly, a MoSCoW table for what is mandatory elements, should haves, could haves and won’t haves.

## UML Diagram

Login GUI: Handles the user logging in to either shopper or manager accounts.

A close up of a box

Description automatically generated

ManagerUI: Handles user interaction with manager UI:

A screenshot of a computer program

Description automatically generated

ManagerApp: Manager application layer, handling methods and interactions with data layer:

A white text with black text

Description automatically generated

ShopperUI: Handles user interaction with ShopperUI

A screenshot of a computer program

Description automatically generated

ShopperApp: Shopper Application Layer, handling methods and interactions with data layer:

A screenshot of a computer

Description automatically generated

dbQueries, dbUpdate and dbInsert: Handling database requests for user:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Red box**: Presentation layer (ShopperUI, ManagerUI)

**Blue Box**: Application layer (ManagerApp, ShopperApp)

**Black Box**: Data layer (dbQueires, dbInsert, dbUpdate)

## Activity Diagram

Manager Update Inventory: Manager View Branch Peformance

View Sales Request

Get Sales Request

Select View Performance Request

Get branch ID

Add updated info to database.

Get updated Inventory.

View Inventory Request

Select Inventory Request

Get branch ID

Shopper Buying Stock:

Add updated info to database.

Get updated Inventory.

View Inventory Request

Select Inventory Request

Get branch ID

## Class Diagram

A diagram of a diagram

Description automatically generated

## MoSCoW

The MoSCoW system is a way to order the importance of the specification for prioritising the development:

|  |  |
| --- | --- |
| Classification | Description |
| Must Have | Take input of User ID |
| Must Have | Verify ID and assign appropriate role |
| Must Have | Verify Branch of Manager |
| Must Have | Retrieve Branch specific Inventory |
| Must Have | Update specific inventory |
| Must Have | Request More Stock |
| Must Have | View Branch Performance |
| Must Have | Allow shopper to purchase with cash |
| Must Have | Allow shopper to purchase with finance form |
|  |  |
| Should Have | Input Validation |
| Should Have | Database security for connections |
| Should Have | Tooltips |
| Should Have | GUI for user interaction |
| Should Have | User account creation with default branch |
|  |  |
| Could Have | Manager password |
| Could Have | Shopper email login |
|  |  |
| Won’t Have | Admin system for manager creation and moderation |
| Won’t Have | Security for all classes/database interactions |
| Won’t Have | Allow managers to add inventory or delete inventory |

## Tools Used

Programming Language Used: Java

Database used: MySQL MariaDB

IDE used: Eclipse 2022/23

# Evaluation

## MoSCoW Evaluation

|  |  |  |
| --- | --- | --- |
| Classification | Description | Achieved (Y/N) |
| Must Have | Take input of User ID | Y |
| Must Have | Verify ID and assign appropriate role | Y |
| Must Have | Verify Branch of Manager | Y |
| Must Have | Retrieve Branch specific Inventory | Y |
| Must Have | Update specific inventory | Y |
| Must Have | Request More Stock | Y |
| Must Have | View Branch Performance | Y |
| Must Have | Allow shopper to purchase with cash | Y |
| Must Have | Allow shopper to purchase with finance form | Y |
|  |  |  |
| Should Have | Input Validation | Y |
| Should Have | Database security for connections | N |
| Should Have | Tooltips | N |
| Should Have | GUI for user interaction | Y |
| Should Have | User account creation with default branch | N |
|  |  |  |
| Could Have | Manager password | N |
| Could Have | Shopper email login | N |
|  |  |  |
| Won’t Have | Admin system for manager creation and moderation | N |
| Won’t Have | Security for all classes/database interactions | N |
| Won’t Have | Allow managers to add inventory or delete inventory | N |

I have managed to successfully achieve all of the must have features of the application, and even had time to include some should haves.

## Future Iteration

**Database security for connections**

For better security and data confidentiality and integrity being transmitted from application to the database. By implementing this secure connection, it will provide a layer of protection against unauthorized access, data interception or tampering. Providing a much-needed level of protection and security for anything to do with data.

**Tooltips**

Tooltips added in the future of the program can enhance the user experience and provide additional information or guidance. They can be helpful in providing contextual information or explanations, making it easier for users to understand and interact with the system. Tooltips can also improve accessibility by offering descriptive text for users with visual impairments who rely on assistive technologies.

**User Account Creation with Default Branch**

Allowing users to create their own account with a default branch in the DE-Store program would provide them with a personalized experience and make it easier for them to access information and make purchases. By selecting their preferred branch, users can receive targeted promotions and offers specific to that location. Additionally, it allows for better customer tracking and analysis, allowing the store to provide more personalized recommendations and improve overall customer satisfaction.

**Manager Password**

The addition of a manager password in the future the program would provide an extra layer of security and access control. It would allow the store manager to have exclusive access to certain sensitive functionalities and data, ensuring that only authorized personnel can make changes to the system. This would help to prevent unauthorized access and potential misuse of the system by unauthorized accounts.

**Shopper email login**

An email login can enhance the security of shopper accounts, ensuring that only authorized individuals have access to their personal information and purchase details.

**Admin system for manager creation and moderation**

Adding an Admin system for manager creation and moderation in the program would provide enhanced security and control over user access and permissions. It would enable the system to have different levels of authorization, allowing managers to have specific privileges and capabilities based on their roles. This would facilitate efficient management of the system and ensure that only authorized individuals can perform administrative tasks.

**Allow managers to add inventory or delete inventory**

Allowing managers to add or delete inventory will be added in the future to enhance the functionality and flexibility of the program. This feature will enable managers to easily update and manage the stock levels in real-time, ensuring accurate records and efficient stock management. Additionally, it will allow for more effective inventory control, preventing overstocking or understocking of products.

## Testing:

Testing will be light and only for essential components. This is due to some time constraints and not being able to do all possible test cases, so for efficiency only the must haves from the MoSCoW will be tested for.

|  |  |  |
| --- | --- | --- |
| Test | Data | Pass/Fail |
| Take input of User ID | 1 | Pass |
| Verify ID and assign Appropriate Role | “1”, “Manager” | Pass |
| Verify Branch of Manager | “1”, “North” | Pass |
| Retrieve Branch specific inventory | “North” | Pass |
| Update Specific Inventory | invID=”1”, invName=”pillows” | Pass |
| Request More Stock | 2 | Pass |
| View Branch Performance | saleID=241, bID=1, cID=1, total=100.0, discount=0.0, profit=100.0 | Pass |
| Allow shopper to purchase with cash | User Clicks purchase | Pass |
| Allow shopper to purchase with finance form | User clicks form, form opens and once closed stock is ordered | Pass |

From the tests, all of the main requirements are met.