**Software Architecture Coursework Specification**

**Chapter 1 – Deciding an Architecture:**

Section 1 – Understanding the Project:

When starting work on a new project requiring the use of design architectures, multiple potential architectures should be compared and evaluated to ensure the use of one which best fits the scope and confines of the project. The context of this project is a development bid for a distributed system to be used as a retail management network, used to provide better coordination for business being done. The system is named “DE-Store” and must be able to handle; modification of item prices and deals, inventory/stock management, a customer loyalty card system, interconnectivity to an online finance payment portal for approving customer finance choices, and lastly, the tracking of customer activity for use in reports for store performance analysis.

Section 2 - Potential Architectures:

The first and most immediate architecture taken into consideration is three-tiered architecture. Three-tier architecture is a variant of Multi-tier architecture (also known as n-tier architecture), an architecture type which focuses on a modular approach to development. The key feature of the three-tier architecture is, as suggested by the name, it’s three tiers, also known as layers. The tiers of the architecture each encompass a different level of abstraction from the highest layer, that which is accessible by a user, so that the user is not directly interacting with lower layer information such as stored data.

In the most common format of the three-tiered architecture, the highest layer is the Presentation Layer. This is the layer which provides the direct interfacing capabilities of the application to a given user. These interfacing capabilities can take various forms such as a Web Page or a Graphical User Interface (GUI). The whole purpose of this layer is to communicate directly with the user of the application, both in terms of receiving commands and inputs from the user, as well as displaying outputs in an adequate format for the user to understand. This layer only communicates with the one layer below it, and essentially only serves the purpose of being a messenger between the user and the application, without actually doing any serious logic of its own.

The next layer down from the Presentation Layer is the Business Layer, also known as the Application Layer, Logic Layer or Middle Layer. This is the layer where the actual “hard work” of the application takes place, such as processing information and commands, making logical decisions, and performing calculations. This layer behaves as a “middleman” between the user of the application and the data contained within.

The lowest layer of the architecture is the Data Layer. This is the layer where all persisted data within the application gets stored and retrieved from. It contains a means of communicating with the Business Layer to allow its methods of managing, storing and retrieving data to be accessed by the Business Layer when desired. The data within the later is most commonly stored within a database, often saved within its own separate secure server for the sake of data security in the context of commercial applications.

Key Advantages of the three-tiered architecture;

* Easily managed and worked on by multiple different developers/teams.
* Centralised maintenance for many end-users.
* Allows the application to easily be rescaled to fit different levels of engagement and usage.
* Supports Multi-threading to improve performance and reduce network traffic build-up.
* Modular approach allows a stronger focus on security in the most vital points within the application.
* Lightweight for client use, meaning it’s accessible for a wider range of hardware, especially since it can often take an ultra-light client-side format such as a website in many cases.

Key Disadvantages of the three-tiered architecture;

* More complex to develop than many of the more basic architectures using singular or two-tiered approaches.
* Will be more commercially expensive to implement due to its more complex nature, though arguably for good reason.
* Can prove difficult to test due to lack of available tools and resources.

The second architecture of consideration is the Client/Server architecture. This is also a variant of the Multi-Tiered architecture format, this time taking a two-tiered approach. The key focus of this architecture is having a dedicated application for the client, aka the user/consumer of the application. A second application is then being run off of a server, which contains the appropriate persisted data and logical capabilities to be utilised by a connected client. Within the context of this project, the Client/Server architecture will refer to the Thick/Fat-client model, as it’s the most common and appropriate within the context of modern commercial Client/Server systems where the intended capabilities of the Client are known in advance. This model also reduces the load and requirements of the sever by offloading part of the “hard work” on to the client, which is effective when multiple individual concurrent users are each trying to utilise the application.

Within this version of a Client/Server system, the client side of the application manages the presentation and application processing. The presentation aspect includes the communication of information to a client, such as text or images, as well as taking in input from the client to then be processed by the application. The processing aspect involves managing input from the user, possibly validating said input, then transforming it to a form to be sent to and further processed by the server. This aspect also involves receiving information from the server then converting it to an adequate form to be presented back to the user of the application.

The server side of the application manages its own application processing as well as data management. The application process aspect covers much of the same ground as the same aspect within the client side of the application, with the likely additions of security aspects, in attempt to avoid potential cyber-attacks aimed towards the server, especially if it manages the transferal of money or products. The data management aspect of the server involves the storage, persistence and retrieval of information required by the server, often to be used by both the sever and client.

Key advantages of Client/Server architecture;

* Simplifies design to expediate development.
* Very effective in situations where the number of clients vastly outnumbers the number of servers.
* Server components are easily replicated for multiple servers.
* Separates some aspects for modular design approach, allowing certain elements of the application to be easily developed/modified my multiple different developers/teams.
* Easy and simple method of creating a distributed system/environment.

Key disadvantages of Client/Server architecture;

* Security can be difficult to properly integrate in the most necessary areas.
* Clients side is somewhat difficult to manage, maintain and upgrade due to Presentation and Business/Processing aspects being stuck together.
* Somewhat limited client hardware choices as processing is done before sending information to the server.
* Potential issues when running on devices which it isn’t directly developed for and tested on.

Section 3 – Selecting an Architecture:

Given the considerations of each of the two architectures explored in the previous section, the pair should be critically compared against one another on various aspects within the context of the project to best discern which of them is the most suitable.

The first point worth comparing is the complexity of the implementation using each architecture, as this will directly correlate to development time, given the same developer/development team would be used regardless of the architecture. In this case Client/Server architecture only requires two layers of development, compared to the three-tier architectures three layers. Although the layers of the Client/Server system are likely more complex than the layers of a three-tier system, the overall complexity is more often than not going to be higher within the three-tier system. This results in the Client/Server architecture being the most effective within this field of consideration.

The next point of comparison is development and upkeep costs. It is safe to assume that the development cost of the system using either architecture would increase proportionally to the time spent on developing the system, as more required man hours is equal to increased development cost. As well as development cost, upkeep including server and maintenance costs should also be considered. Where maintenance costs are concerned, an architecture which is more modular and easily broken down into parts is more effective, leading the three-tier architecture to having an edge over the Client/Server architecture. However, the key drawback to the three-tiered system within this context is its server-heavy nature, leading to likely further server upkeep and scope costs when compared to the Client/Server architecture, however due to the small user base intended by this system, this point is not worth a huge amount of consideration. Both architectures have solid arguments to be the most effective in this field. Within the short term, the immediate development cost of a Client/Server system is cheaper than that of a three-tiered system, however where future upkeep is concerned, a three-tiered system likely has the advantage in the context of this project.

Another worthwhile point of consideration is user hardware requirements. Within this point of consideration, the lightweight user-end of a three-tiered system has a clear advantage over the Thick/Fat-model implemented within the Client/Server architecture. The lightweight nature of the system makes it especially useful in the context if this project as it assumes there’s a central server which connects multiple branches of the business together, saving device costs as more branches end up with the implemented system, as less-powerful hardware is required within the three-tiered system.

One of the most vital points of consideration is the implementation of security elements. When comparing the development of a three-tiered system to that of a Client/Server system, the three-tiered system will often have a much easier time implementing appropriate and effective security elements due to its more modular nature as well as its general structure. This means the three-tiered architecture holds a key advantage in this regard.

The last point of comparison between the two architectures is their expandability and adaptability, both being points strongly emphasised within the specification of this project. In this regard the much more modular nature of the three-tiered system is the clear frontrunner when compared to the Client/Server architecture, which does not separate certain elements (such as the processing/logic and data management) therefore complicating the maintenance and upgrading process to some extent. This leads to the three-tiered architecture to also holding the advantage in this regard.

To conclude the comparison, if the system regarding this project were to be on a smaller scope or only used for the short term, the Client/Server architecture would likely be a much more cost and time effective method of implementation. However, due to the wide scope of the project, and the desire for it to be maintained over the long-term, as well as be expanded when desired, it is best to recommend development utilising the three-tiered architecture. This justifies the use of the three-tiered architecture in the designing and development of the prototype model, as it contains all of the most desired functionality and important features, while still being a relatively simple and developer-friendly.

**Chapter 2 – Designing the System:**

Due to the nature of the expected prototype, the decision was made to keep each of the features entirely separate from each other. In a fully implemented version of the system the “Price Control” and “Loyalty Card” sections would likely be on the same form. The positive side to keeping each key feature separate however, was the fact each feature could have their own flowchart. Since flowcharts are a very simple and convenient method of breaking down a program into simpler parts, this was the most ideal method of designing of the prototype.

The first feature to be flowcharted was the Price Control feature. The flowchart can be seen in **figure 1** in the *Appendix*. The flowchart shows the idea behind the feature, having it display a list of each of the products on sale, then allowing a user to click on one of the products to open a sub-window with each of the product details to be modified, including a price field, which is planned to take the form of a simple input box, and a offers field, which is planned to be a drop-down window, due to only having a finite amount of possible offers to choose from.

The second feature to be flowcharted was the Inventory Control feature. The flowchart can be seen in **figure 2** in the *Appendix*. For the sake of the prototype this feature was planned to be somewhat more simple than the specification describes, only being activated when clicked on within the prototype, instead of running all the time. Two is the highest value that would be counted as “low stock”, this was chosen arbitrarily and if an item has three or more stock, it will not be counted as “low stock” within the prototype. This is planned to be easily modifiable with a single variable within the prototype, so that it can be tuned accordingly with little-to-no effort.

The third feature to be flowcharted was the Loyalty Card feature. The flowchart can be seen in **figure 3** in the *Appendix*. This is a simplified version of the Price Control feature, having each item in the database having a dropdown field where a selection of set deals can be applied to each item, to only be provided to Loyalty Card customers.

The next feature to be flowcharted was the Finance Approval feature. The flowchart can be seen in **figure 4** in the *Appendix*.

The final feature to be flowcharted was the Reports and Analysis feature. The flowchart can be seen in **figure 5** in the *Appendix*. This feature is where the most assumptions had to be made as the contents of the “Accounting Database” were quite vague. Because of this, only the most barebones features were planned, those being; total earnings, earnings over the past 12 months and percentage earnings of the past 12 months compared to the 12 months prior. Other features could easily be added to this section if the associated “Accounting Database” contains enough information to permit them, these could include; net earnings, total customers, average customers per day, average spend per customer and total items sold. These may be implemented in the prototype after all of the key features, if time permits.

**Chapter 3 – Implementing the Prototype:**

Section 1 – Choosing a programming language:

Provided the planning from the flowcharts, the decision was made to implement the prototype using C#. This is because C# is a language that I’m comfortable with, as well as being an ideal language for the three-tiered architecture. Being a part of the .net platform allows C# access to various useful design tools, such as the simple drag and drop form designer, saving time during the prototypes development as user interface (UI) features are of a lower concern than the actual features as part of the prototype. C# also contains convenient database handling using SQL based databases, which is helpful in demonstrating the data storage element within the prototype.

Section 2 – TitleHere:

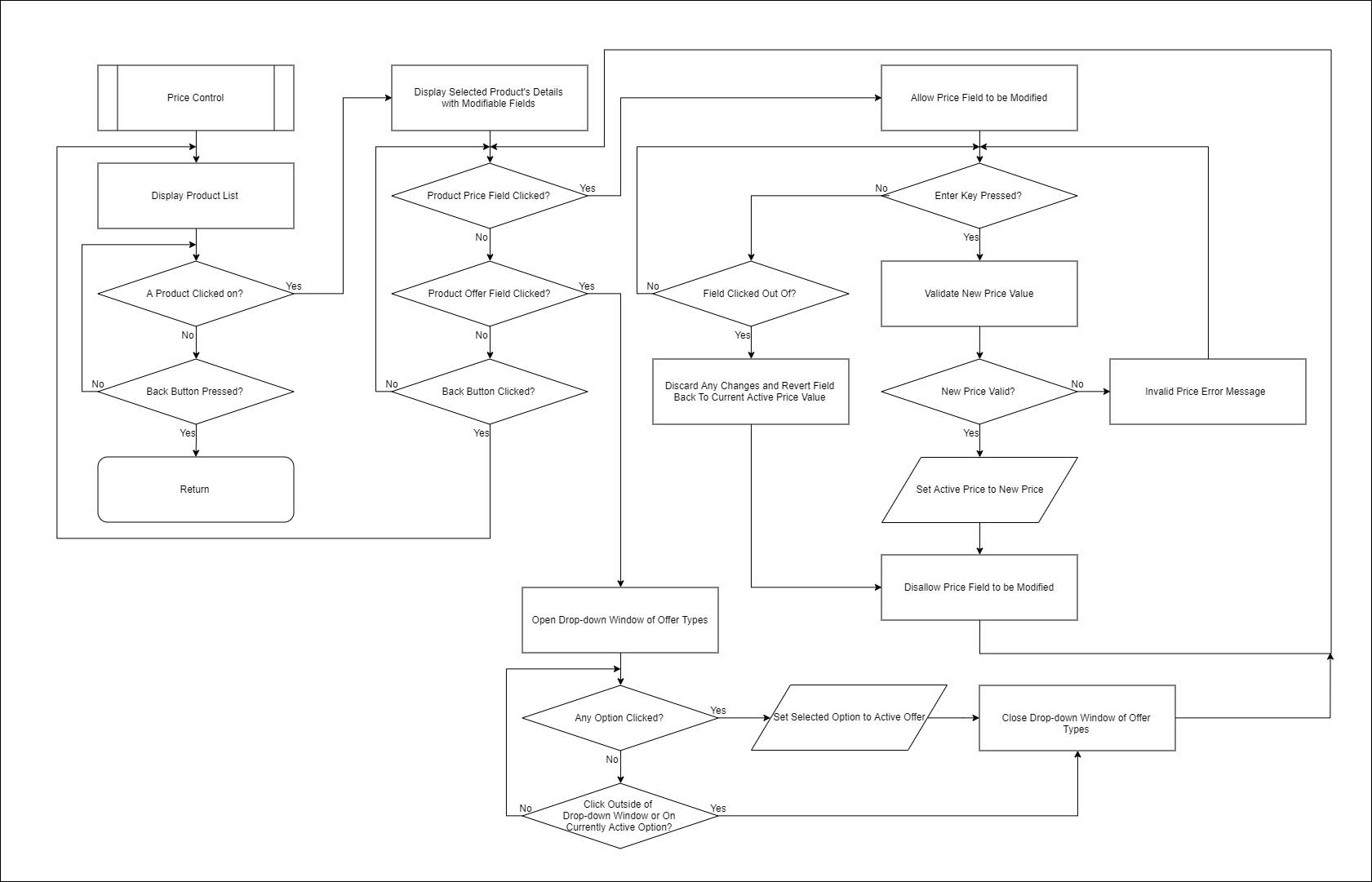
ContentHere

**Chapter 4 – Evaluating the Prototype:**

Section 1 - TitleHere:

ContentHere

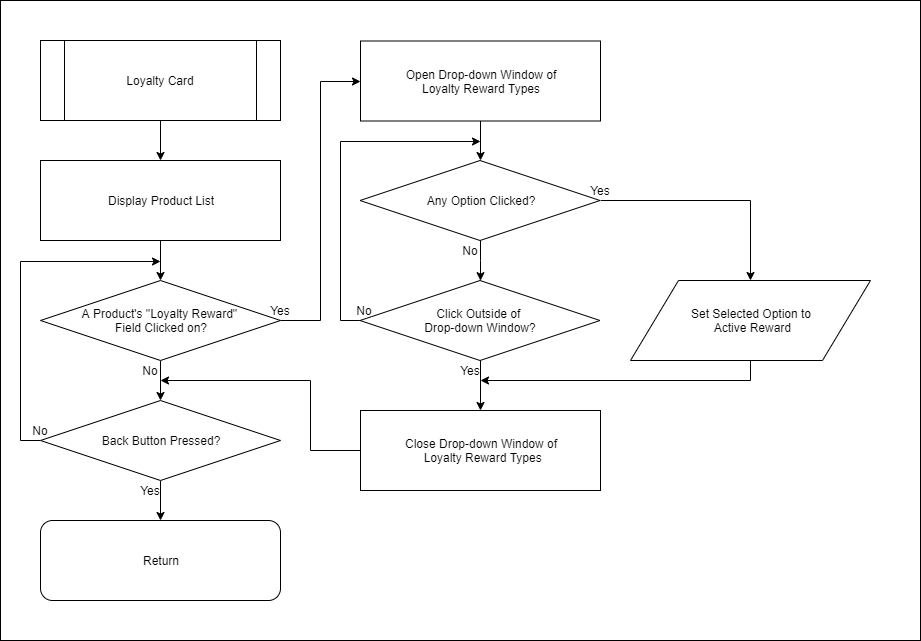
**Appendix:**



**Figure 1**. Price Control Flowchart.

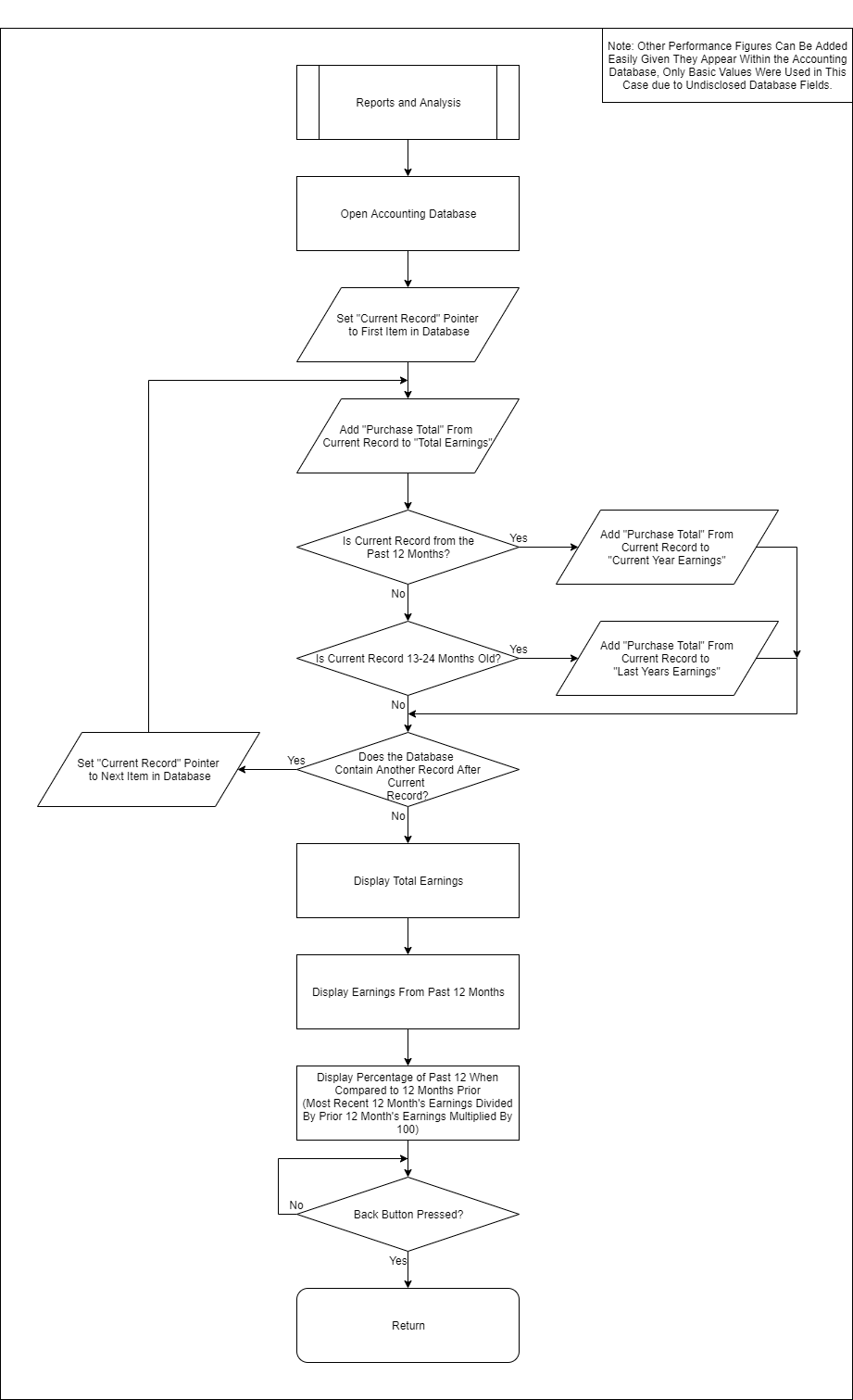


**Figure 2**. Inventory Control Flowchart.



**Figure 3**. Loyalty Card Flowchart.

**Figure 4**.



**Figure 5**. Reports and Analysis Flowchart.