

# **Software Architecture Coursework Design Document**

## **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 one which best fits the scope and confines of the project is used. 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 input data from the user, as well as displaying various types of output, in an adequate format that the user can 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 or calculations 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 Layer is most commonly stored within databases, sometimes stored within their own separate secure servers for the sake of improved data security.

#### 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 comprehensively 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 on 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 server 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's 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 any required input from the client to then be processed by the application. The processing aspect involves managing input from the user, validating that input, then transforming it into an appropriate form to then be sent to and 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 or possession of money or goods. 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 server 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 by 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 the architectures on is the complexity of the implementation using both of them. 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 has two Layers to develop, compared to the Three-Tier architectures' three. 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, resulting in the Client/Server architecture being the most effective within this field of consideration.

The next point of comparison is the cost of development and upkeep. It is safe to assume that the development cost of the system would increase proportionally to the time spent on developing the system, regardless of the architecture used, as more required man hours equates to increased development cost. As well as development cost, upkeep costs, which include server prices 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 heavier nature, meaning it consumes more server processing power, leading to further server upkeep costs when compared to the Client/Server architecture.

However, due to the small user base of DE-System, this is a much less vital point to consider than the other considerations comparing the two architectures. 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 using the Client/Server architecture. The lightweight nature of the system makes it especially useful in the context of 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, since less-powerful local hardware is likely required by 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. This means the Three-Tiered architecture holds a key advantage in this regard.

The last point of consideration between the two architectures is their expandability and adaptability, both points having been 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 (i.e. without the ambition to upgrade in the future), the Client/Server architecture would likely be a more cost effective and time efficient method of implementation. However, due to the wide scope of the project, with the desire for it to be maintained and possibly upgraded over the long-term, it is best to recommend the Three-Tiered architecture. This justifies the use of the Three-Tiered architecture for the design and development of the prototype model, as it contains all of the most desired functionality, 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 on the Presentation Layer. In a fully implemented version of the system the "Price Control" and "Loyalty Card" sections would likely be part of the same form. The positive side to keeping each of the key features separate however, was the fact each feature could have their own flowchart during the design of the system before starting development. Since flowcharts are a simple and convenient method of breaking down a program into smaller parts, this was the most ideal

method of designing of the prototype in a swift and efficient fashion without wasting too much potential development time.

The first feature to be flowcharted was the Price Control feature. The flowchart can be seen in **figure 1**, found in the *Appendix*. The flowchart shows a list of each of the products on sale being displayed, allowing the user to click on any one of the products to open a sub-window with each of the product details that can be modified as part of this feature, including a price field, planned to take the form of a simple input box, and an offers field, 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**, found in the *Appendix*. For the sake of the prototype this feature was planned to be somewhat more simple than the specification describes, only being active when clicked on within the prototype, instead of running full-time when the application is running. Two is the highest value planned to be counted as “low stock”, 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**, found 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**, found in the *Appendix*. The flowchart for this feature ended up being much more barebones than those of the other four features because the feature intends to utilise a Portal. Within the prototype the portal will not be fully accessible, however the idea of a portal will still be implemented in a basic form as a dummy portal. When the dummy portal, which will likely be a simple function, is connected a demo message will be sent back as a response.

The final feature to be flowcharted was the Reports and Analysis feature. The flowchart can be seen in **figure 5**, found 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 within the prototype after all of the key features, time permitting.

## Chapter 3 – Implementing the Prototype:

### Section 1 – Choosing a programming language:

Having completed the planning with 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 using Visual studio, also allows C# access to various useful design tools, such as the simple drag and drop form designer, saving time during the prototype's development, since user interface (UI) features are of a lower concern than the actual functionality of the prototype's features. C# also contains simple and convenient database handling using SQL based databases, with Visual studio having inbuilt local SQL Server based database functionality. This allows for a convenient means of demonstrating competence using databases without the need of an individual server to host the database for the sake of the prototype, meaning the whole prototype can be contained within a single Visual studio project.

### Section 2 – Data Layer:

During the design phase of the project, little thought was put towards the Data Layer as most of the decisions to be made were relatively small, with some reliant on choice of development tools and environment. Since C# was used for the prototype development, using Visual Studio, a local database using SQL Server was the optimal option, due to the small scale of the prototype. In the full system an SQL based service would still likely be used since SQL is the industry standard, with the database(s) instead being stored on a secure server(s), away from the rest of the application, unlike it is in the prototype.

Two database tables were required by the prototype system, one to store the inventory records, and the other to store the accounting records. In the full system these would be stored on entirely separate databases for security purposes. However, in the prototype the tables were stored on the same database, named "DESystemDatabase", for the sake of simplicity and convenience.

The "inventory" table consists of six fields; ItemId, ItemName, Price, StandardOffer, LoyaltyOffer and Stock. ItemId is an integer field that serves as the primary key. It is a simple count-up Id field, starting at 1 and increasing by 1 with each new item added. ItemName is a string field and contains the plain text name of each item in the table (e.g. "Step Ladder"). Price is a decimal field, set to have up to 4 digits before the point and 2 after, to take the shape of a monetary value with a maximum of 9999.99, as I deemed no single item would likely cost more than £10,000, however this could easily be changed in the future. StandardOffer is an integer field that contains a number to represent an offer type, which is understood by the program (e.g. 1 is translated to "Buy 1 Get 1 Free"). Akin to StandardOffer, LoyaltyOffer is also an integer field which uses a number to represent an offer type. In the case of LoyaltyOffer, each of the deals are percentage-based discounts (e.g. 1 is translated to "10% Off"). Lastly, Stock is an integer value used to represent how many of each item were left in the store inventory.

The "accounting" table consists of four fields; PurchaseId, PurchaseDate, TotalSpend and TotalItemsPurchased. Each record using the accounting table represents a single purchase

made in the store. PurchaseId, akin to ItemId within the inventory table, is a simple integer field used as the primary key. The field starts at 1 and is incremented by 1 each time a new record is made. PurchaseDate is the date which a purchase was made, in the format “dd/mm/yyyy”, used for calculating any time-based statistics that the store needs. TotalSpend is a decimal field, set to have 10 digits before the point and 2 after (making a limit of 9999999999.99) to represent a maximum spend in a single purchase. The set limit was entirely arbitrary and can be easily modified to better represent the sorts of values the store expects. Lastly, TotalItemsPurchased is an integer value used to count the number of individual items (including duplicates) within a purchase (e.g. if a purchase contains 1 “Step Ladder” and 2 “Dremel”, the field would be 3).

SQL databases easily allow new fields to be added, and existing fields to be removed or modified, meaning the database can be easily modified to closer fit the data desired for DE-Store to save.

Filled out versions of each of the two database tables were made for testing and example purposes as part of the prototype. The instance of the inventory table can be seen in **figure 6**, and that of the accounting table can be seen in **figure 7**, both found within the *Appendix*.

Alongside the database, there is also a C# class named “dbQuery” within the Data Layer. This class contains the connection string for the database, a main method used to send a query to the database when provided one, and lastly, a series of small methods containing a single query each to be sent to main query method. Some of the methods within the small series are used to extract database records into variables of the DataTable format, meanwhile others contain formatted data to be written to the database over existing data when given an Id (since no new data records are being added to the database within the prototype).

### Section 3 – Business Layer:

The Business Layer of the prototype contains six C# classes. The first of these classes is called “InventoryCommunication”. This class acts as a bridge between the Data and Presentation Layers for the Price Control, Inventory Control and Loyalty Card features, translating and processing information from the inventory database table for use, as well as verifying input data to be sent back to the database. “InventoryCommunication” also makes use of two more of the six classes within the Layer. Firstly “inventoryItem”, a get-set class used as a template when returning a single item to the Presentation Layer, where an item consists of the same six fields as the inventory database table (as shown in *section 2*). Secondly, “StandardAndLoyaltyOffers”, a class used by both the Business and Presentation Layers, containing methods to translate each of the plaintext offer names to and from the numerical form they’re stored in the database as. The class is also used to list each of the offer types to be used within the according dropdown menus within the appropriate windows of the Presentation Layer.

The fourth class within the Business Layer is named “StockMessageFormulation” and is used in regards to the dummy models of the Manager’s Mobile Message Box, and the Central Inventory System within the Inventory Control feature. The method is used to formulate the messages to send to each, being warning messages to the manager for every item in low stock and restock requests to the central inventory for every item out of stock. A small

change to this section from the planning stage was the increase of the “low stock threshold” (the point where the low stock warning for an item is triggered), increasing the threshold from 2 to 5 to perhaps better represent the point where a manager would want to know stock is getting low (although there still is a variable which can be used to modify this threshold, as planned).

The fifth and sixth classes within the Business Layer both revolve around the Reports and Analysis feature. First is “AccountingCommunication”, a class which serves a similar purpose to “InventoryCommunication”, serving as a bridge between the Data and Presentation Layers, handling the translation and processing of data from the accounting database table. “AccountingCommunication” relies heavily on the sixth and final of the six Business Layer classes, “performanceReport” as a means of storing and transferring its formatted data to the Presentation Layer. “performanceReport” is another get-set class, containing 9 variables, one for each field within the generated report within the prototype. “AccountingCommunication” acquires the accounting records from the Data Layer, formats them and stores them within an instance of “performanceReport” to return to the Presentation Layer when called.

The Business Layer has no classes used for the Finance Approval feature within the prototype, as the feature is lightweight enough to be handled entirely within the Presentation Layer and has no persisted data associated with it.

#### Section 4 – Presentation Layer:

The Presentation Layer is the most sizable layer of the application, containing 11 WPF windows, each with their own associated C# file to handle all of the display and interaction logic.

When the application is launched, the first of the WPF windows to open is “MainWindow”, which acts as a hub to view each of the five separated features within the prototype. This window can be seen running within **figure 8** of the *Appendix*. The clean and simplistic yet functional approach to window design used throughout the prototype can be previewed within the hub window. Within the window, each of the five buttons with the according names lead to their appropriate features, and the exit button runs a “Shutdown” command to fully and properly exit out of the application.

When selecting “Price Control” from the hub window, a new window opens, closing the hub in its place. The new window can be seen within **figure 9** of the *Appendix*. This new window allows the selection of any item from within the inventory database. The display ListBox was not formatted very cleanly (an issue which plagues all ListBoxes in the prototype) due to time constraints, however the issue would be rectified in the full system through the use of either a different display format, or through character-based logic. If the “<- Back” button is pressed, the application will return to the previous window (a feature included in almost every one of the WPF windows within the prototype). If an item in the ListBox is clicked, a sub-window will open to allow the corresponding item’s details to be modified. An example of this, using the first record within **figure 9**, “Power Drill”, can be observed in **figure 10** of the *Appendix*. Clicking on the dropdown will allow the user to select one of the pre-set offer types from a dropdown window. Pressing confirm will send the data to be validated. If the



data is valid it will be sent to the database to be updated. If not, an error message will show, which can be seen of **figure 11** in the *Appendix*.

When selecting “Inventory Control” from the hub window, three separate windows open in the place of the hub window. All three windows can be seen within **figure 12** of the *Appendix*. The first of these windows is “Stock Monitor”, the window representing the local side of DE-System, which shows the current stock of every item in the inventory database. If the back button on “Stock Monitor” is clicked, all three windows will close, returning the user to the previous window. The second window of the three that open in this feature is “Mobile Message Box”, used to represent the manager’s inbox, where “low stock” and “no stock” warnings are sent for each item that is deemed worthy of a warning within the Business Layer. In the prototype, this box alongside that of “Stock Requests” are both filled when the hub button to view “Inventory Control” is selected, however in the full version of the system, the both forms will not exist and will instead be updated either on a timer or when a change is made in the stock, to ensure the system works all the time and not just when a specific button is pressed. The last of the three windows opened is “Stock Requests”, which fills a very similar role to that of “Mobile Message Box”.

When selecting “Loyalty Card” from the hub window, a very similar window to that of “Price Control” takes the place of the hub. “Loyalty Card” is a feature that very closely mirrors that of “Price Control” in both appearance and function, as the two would be part of the same feature within the full system but were kept separate in the prototype for the purpose of individual feature demonstration. The window displayed can be seen within **figure 13** of the *Appendix*. Much like “Price Control”, when an item is clicked on, a sub-window opens to allow for the fields to be modified and saved, which can be seen in **figure 14**.

When selecting the “Finance Approval” option on the hub window, a quite large window takes the place of the hub. The window can be seen in **figure 15** within the *Appendix*. Breaking the window down: The top ListBox contains the current connection status to the finance portal, “Awaiting Connection” by default, but can also be “connected” or “disconnected”, as well as a separate disconnected message if it was done distinctly so by the server. The buttons on the bottom left connect and disconnect the portal, and also change make the portal’s end of the Live Support box appear and disappear. Lastly in the bottom left is a Live Support chat window, used solely to demonstrate the live 2-way connection between DE-System and the Portal. Clicking the “Connect to Portal” button opens the portal’s end of the live support chat box and changes the display message of the status ListBox of the main window, as can be seen in **figure 16** of the *Appendix*. The figure also shows a live conversation happening. The Live Chat feature itself may well be absent from the final system but is a useful example in demonstrating the live connection. When the disconnect button is pressed, the Live Chat window closes and the message in the connection status box changes, as can be seen in **figure 17**. If DE-System is disconnected by the portal, a different message will display. This can be emulated by closing the chat window manually without the use of DE-System. The result of being disconnected can be seen within **figure 18**. Lastly, a series of error messages are associated with clicking things that are already in effect or not in effect, such as trying to connect when you’re already connected, trying to disconnect when you aren’t connected, or trying to use the Live Chat without being connected. An example of the error messages can be seen in **figure 19**.

Finally, when selecting the “Reports and Analysis” option on the hub window, a single form displaying a series of calculated statistics, based upon the accounting database, takes the place of the hub window. The statistics window can be seen within **figure 20** of the *Appendix*.

## **Chapter 4 – Evaluating the Prototype:**

### Evaluation:

Despite the functionally incomplete and visually unimpressive nature of the prototype, I feel as though it demonstrates most of the fundamental skills and ideas required to develop the full and complete version of the system, given a fair development timeframe. The visual design element would likely be left to a different developer who specialises in User Experience (UX) design, while I work on the code to manage the functionality of the application.

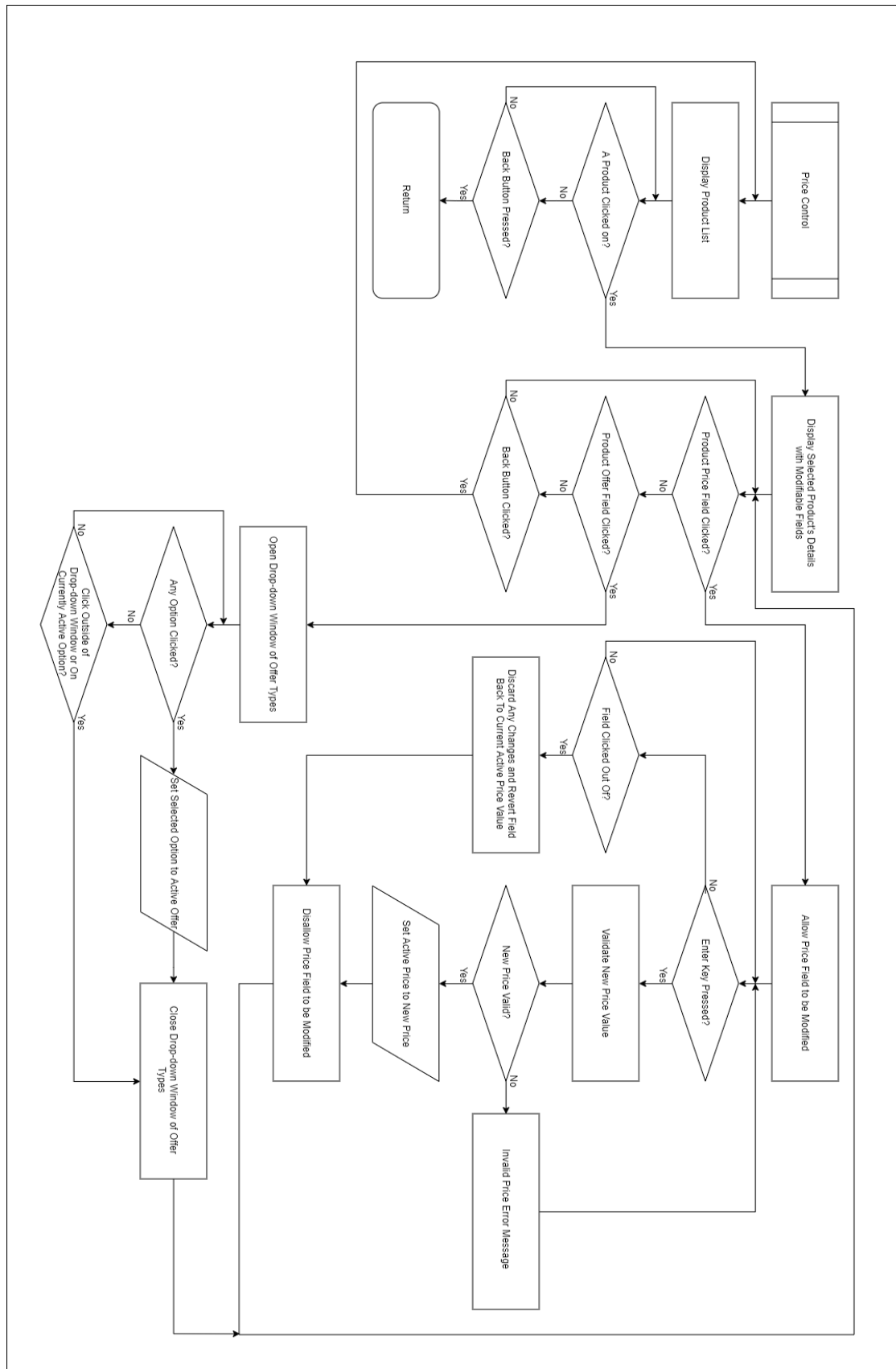
The prototype shows a solid understanding of Three-Tier architecture on a small scale, which should easily translate to the larger scale of the full-size system.

The ways in which features were separated over various small and easy to understand methods allowing for easy maintenance and modification by other developers is also a strong fundamental skill demonstrated by the prototype as a whole.

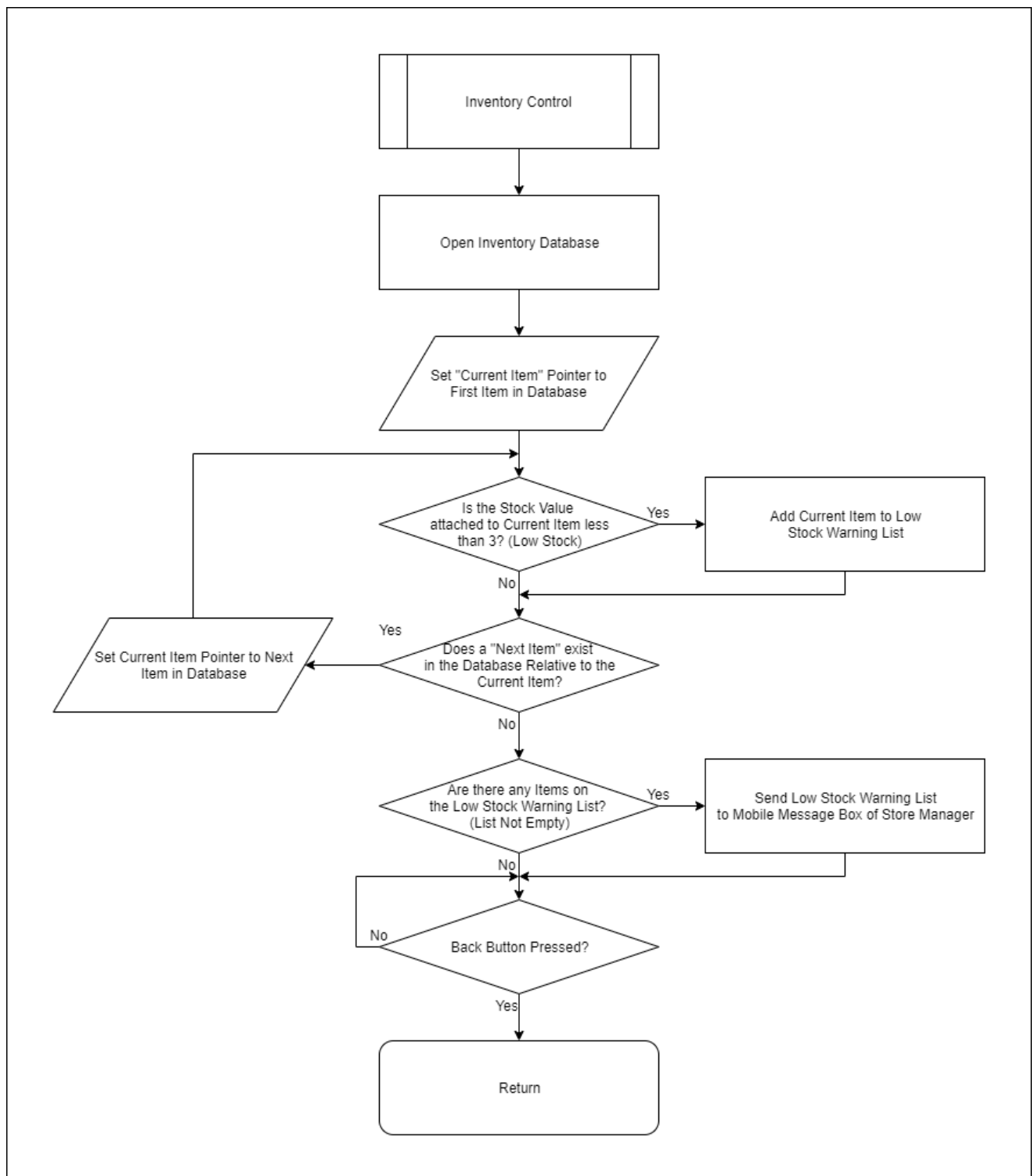
The shortcuts taken in the database design for the prototype would be easy rectified in the full-size system, as they were only taken to save on the effort in regard to the query class on the Data Layer as it was already nearing completion when the accounting database table was added, and with further hindsight wouldn't likely have been done that way had it been developed first.

Most of the primary issues with the prototype system rest within the visual design department due to a mix of lacking time and low skill within the role, but other than those issues, there is very little left to worry about when translating skills demonstrated in the prototype to the full-size system. Any other potential issues with developing the full-size system would simply boil down to a lack of development time or overambitious expectations.

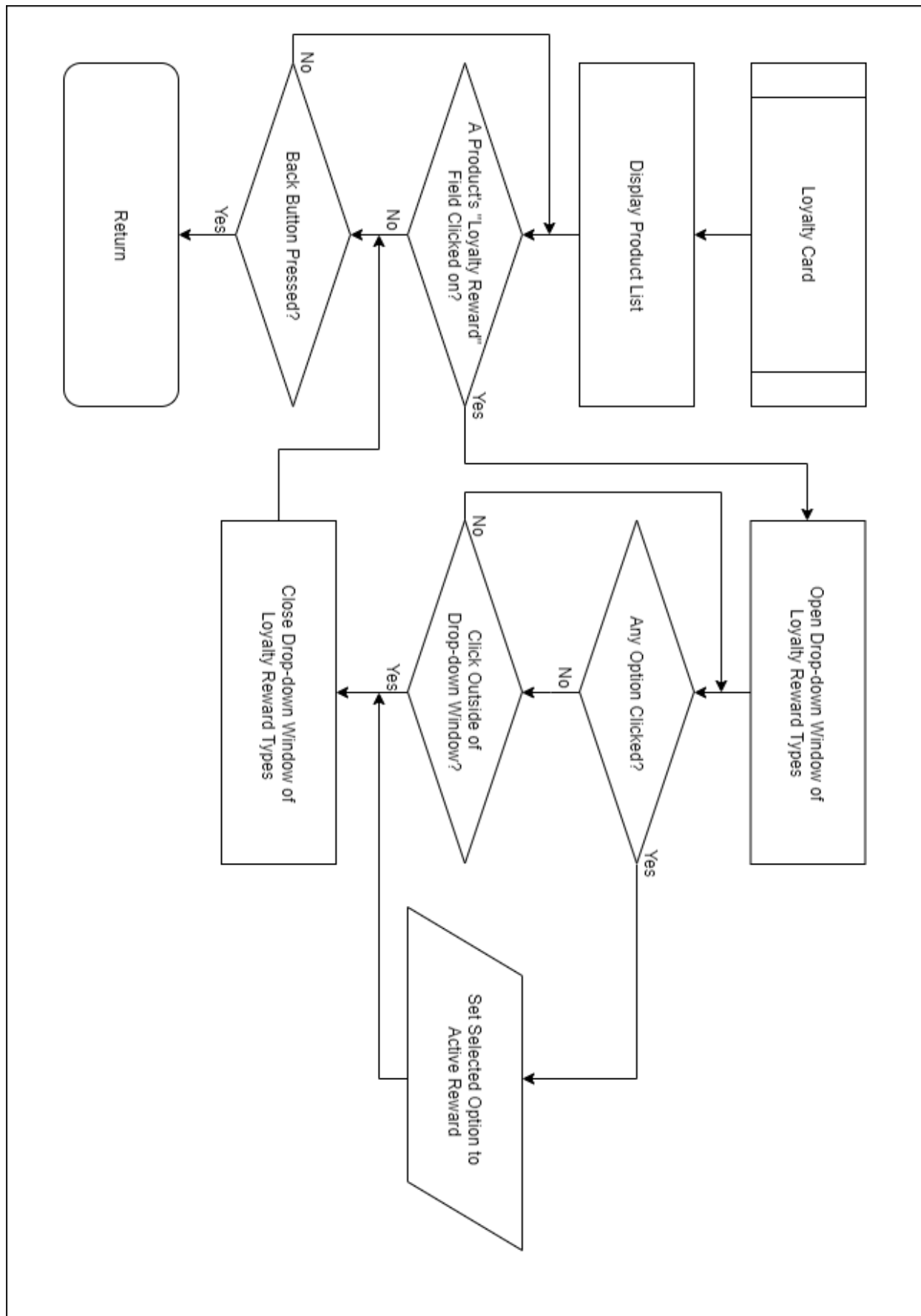
## Appendix:

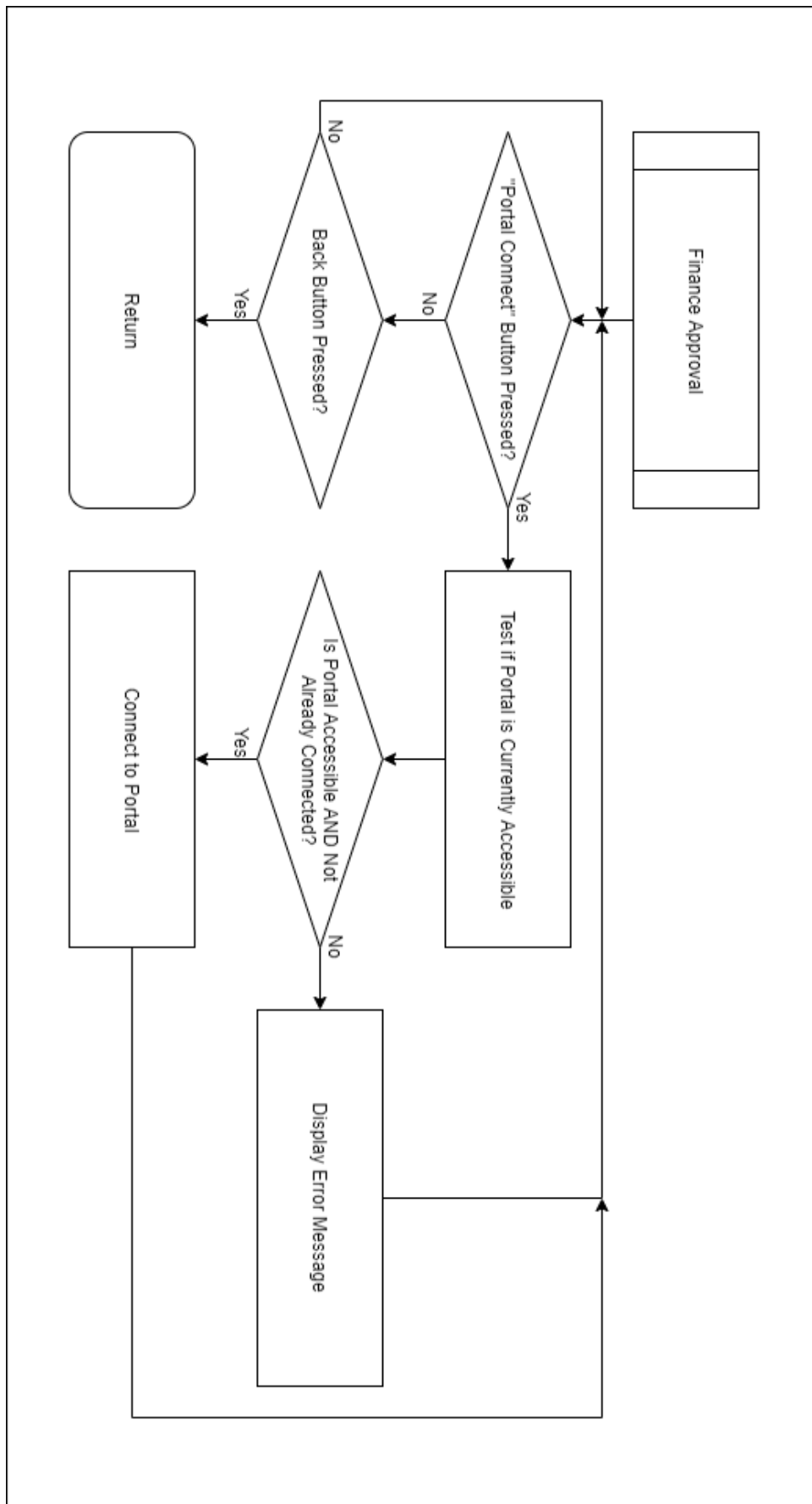


**Figure 1. Price Control Flowchart.**

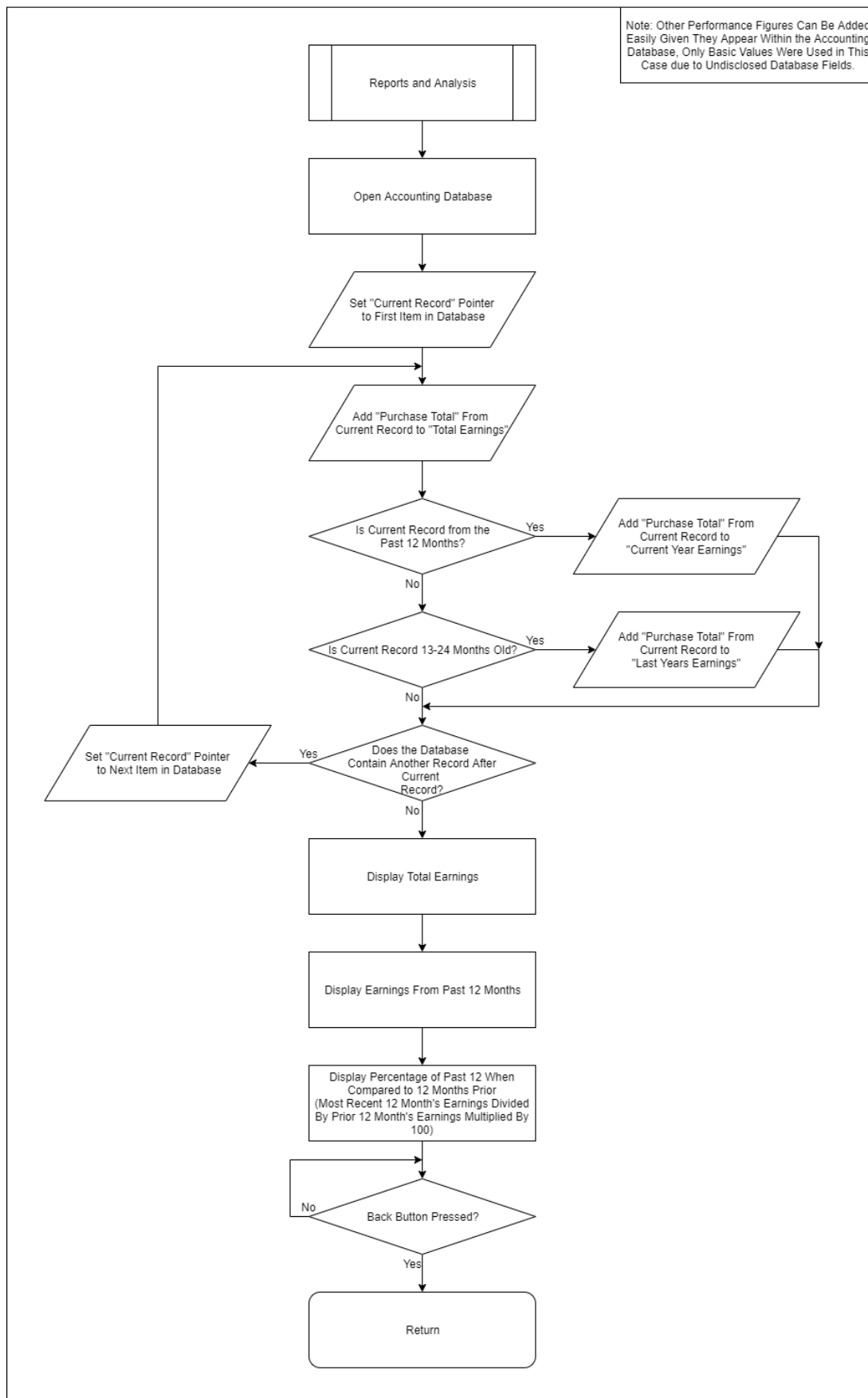


**Figure 2.** Inventory Control Flowchart.

**Figure 3.** Loyalty Card Flowchart.



**Figure 4.** Finance Approval Flowchart.



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

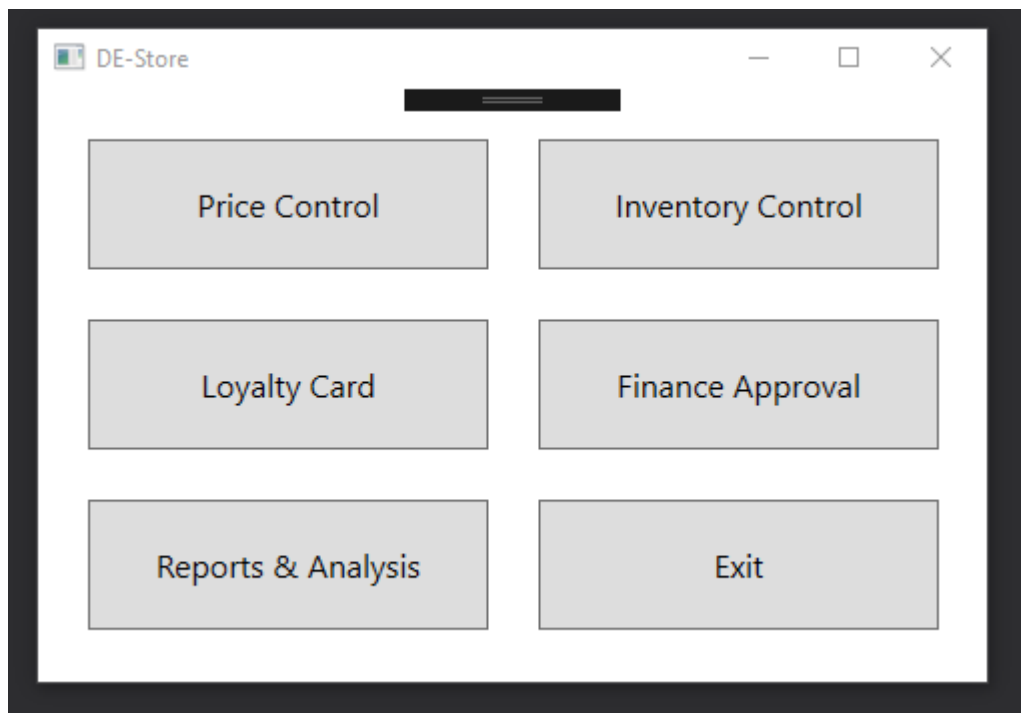
	ItemId	ItemName	Price	StandardOffer	LoyaltyOffer	Stock
	1	Power Drill	19.99	3	1	15
	2	Step Ladder	27.99	1	0	10
	3	Screwdriver Kit	7.50	0	0	30
	4	Claw Hammer	3.95	1	1	34
	5	Disposable Dus...	2.40	2	0	2
	6	Electric Screwdr...	26.00	0	0	1
	7	Nail Gun	40.00	0	0	5
	8	Dremel	37.47	0	1	0
	9	Worklight	19.89	0	0	6
	10	Tape Measure	4.97	0	0	25
	11	Spanner Kit	15.75	0	0	11
	12	Hand Saw	8.00	1	0	19
	13	Soldering Iron	20.00	3	0	2
	14	Blow Torch	18.53	0	1	3
	15	G-Clamp	4.72	2	0	3

**Figure 6.** Example inventory database records from prototype.

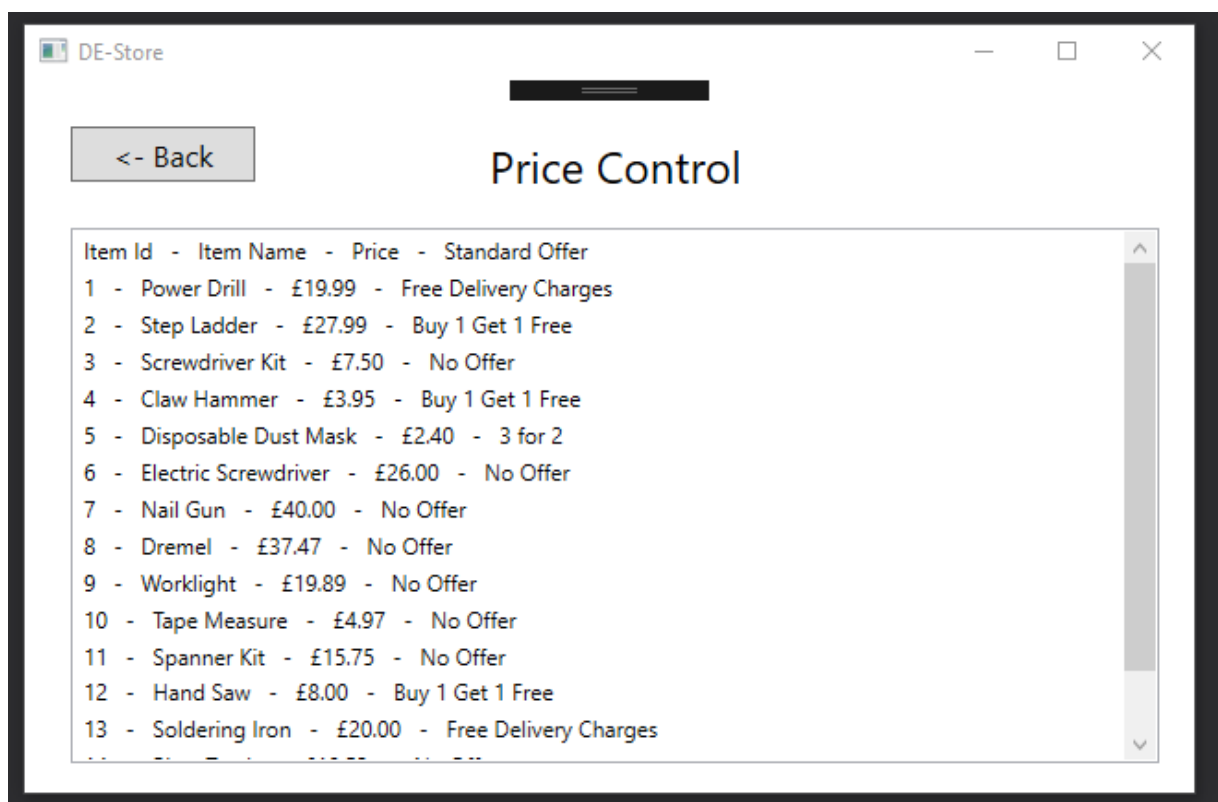
	PurchaseId	PurchaseDate	TotalSpend	TotalItemsPurc...
	1	01/01/2018	35.00	2
	2	17/01/2018	47.20	5
	3	04/02/2018	133.96	18
	4	26/02/2018	2.50	1
	5	19/03/2018	14.98	2
	6	12/04/2018	66.00	2
	7	08/06/2018	6.47	1
	8	02/07/2018	9.44	3
	9	29/08/2018	24.97	2
	10	03/09/2018	48.00	20
	11	24/10/2018	24.00	4
	12	05/11/2018	112.41	3
	13	01/12/2018	257.16	15
	14	01/12/2018	39.87	2
	15	27/12/2018	12.47	2
	16	18/01/2019	13.90	1
	17	02/02/2019	1.20	1
	18	09/03/2019	99.99	6
	19	22/04/2019	70.80	4
	20	19/05/2019	6.00	1
	21	14/06/2019	39.78	3
	22	15/06/2019	20.00	1
	23	29/07/2019	15.00	3
	24	03/09/2019	15.75	1
	25	31/10/2019	80.00	7

**Figure 7.** Example accounting database records from prototype.

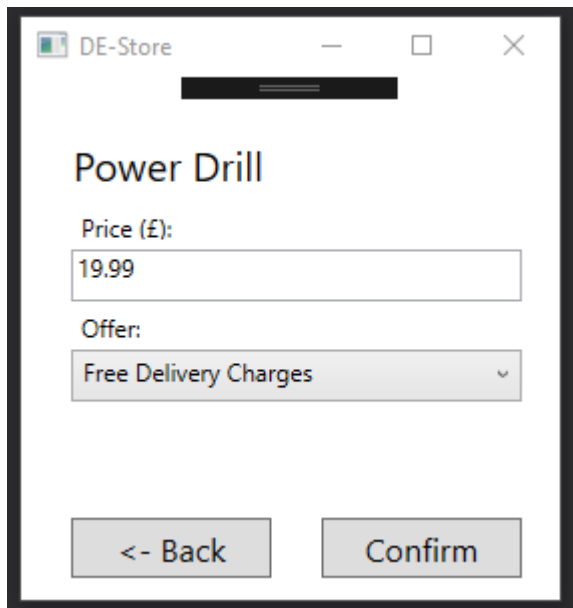




**Figure 8.** Main “Hub” menu in the application.



**Figure 9.** Price Control Window.



DE-Store

Power Drill

Price (£):

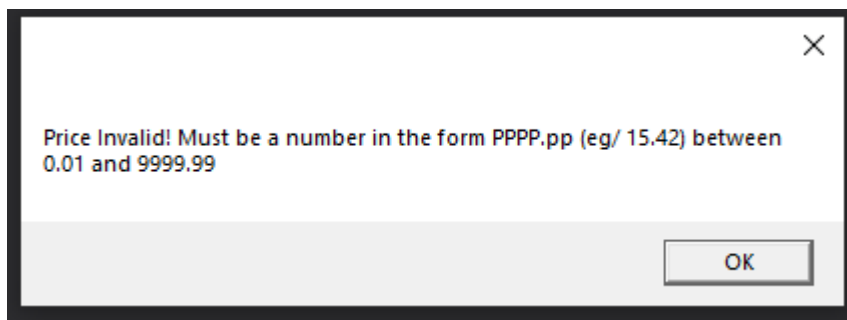
19.99

Offer:

Free Delivery Charges

<- Back   Confirm

**Figure 10.** Single item sub-window of Price Control.



Price Invalid! Must be a number in the form PPPP.pp (eg/ 15.42) between 0.01 and 9999.99

OK

**Figure 11.** Invalid Price error message.

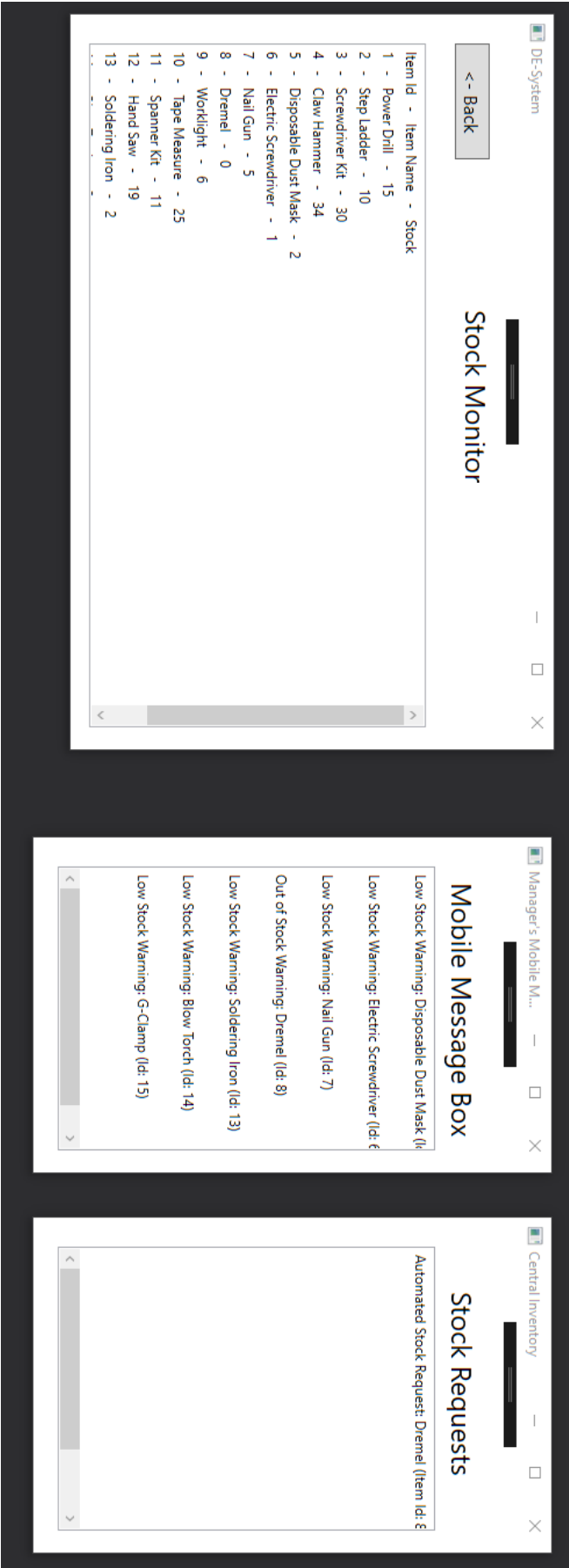
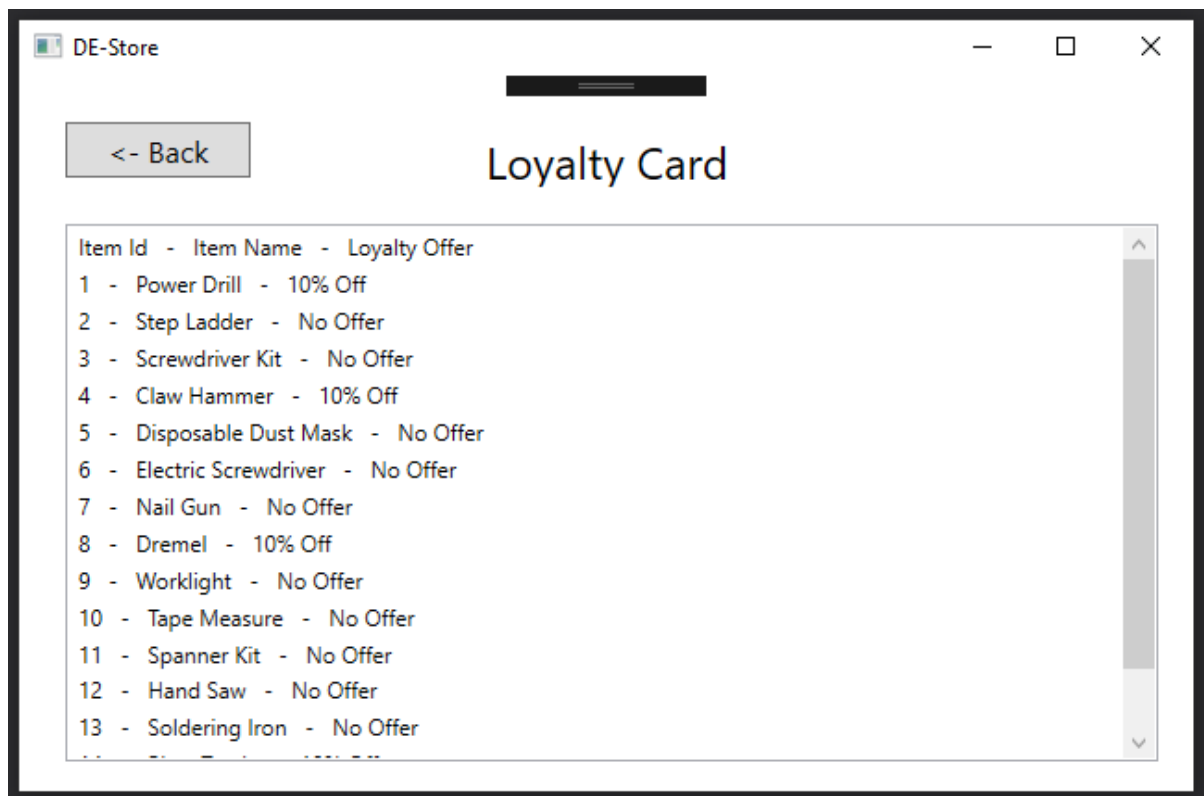
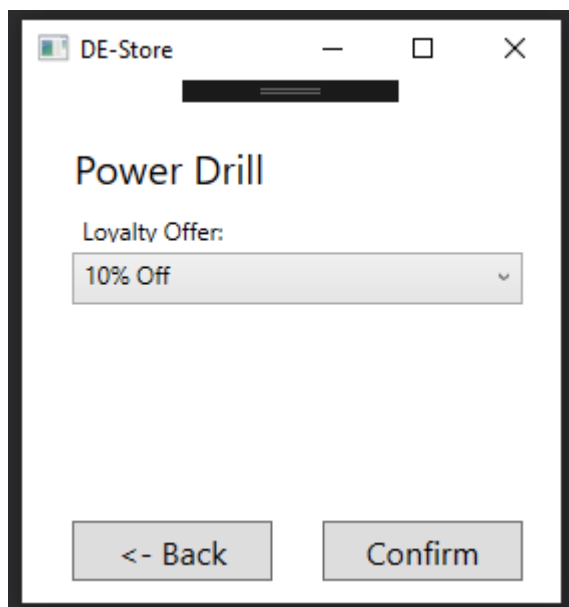


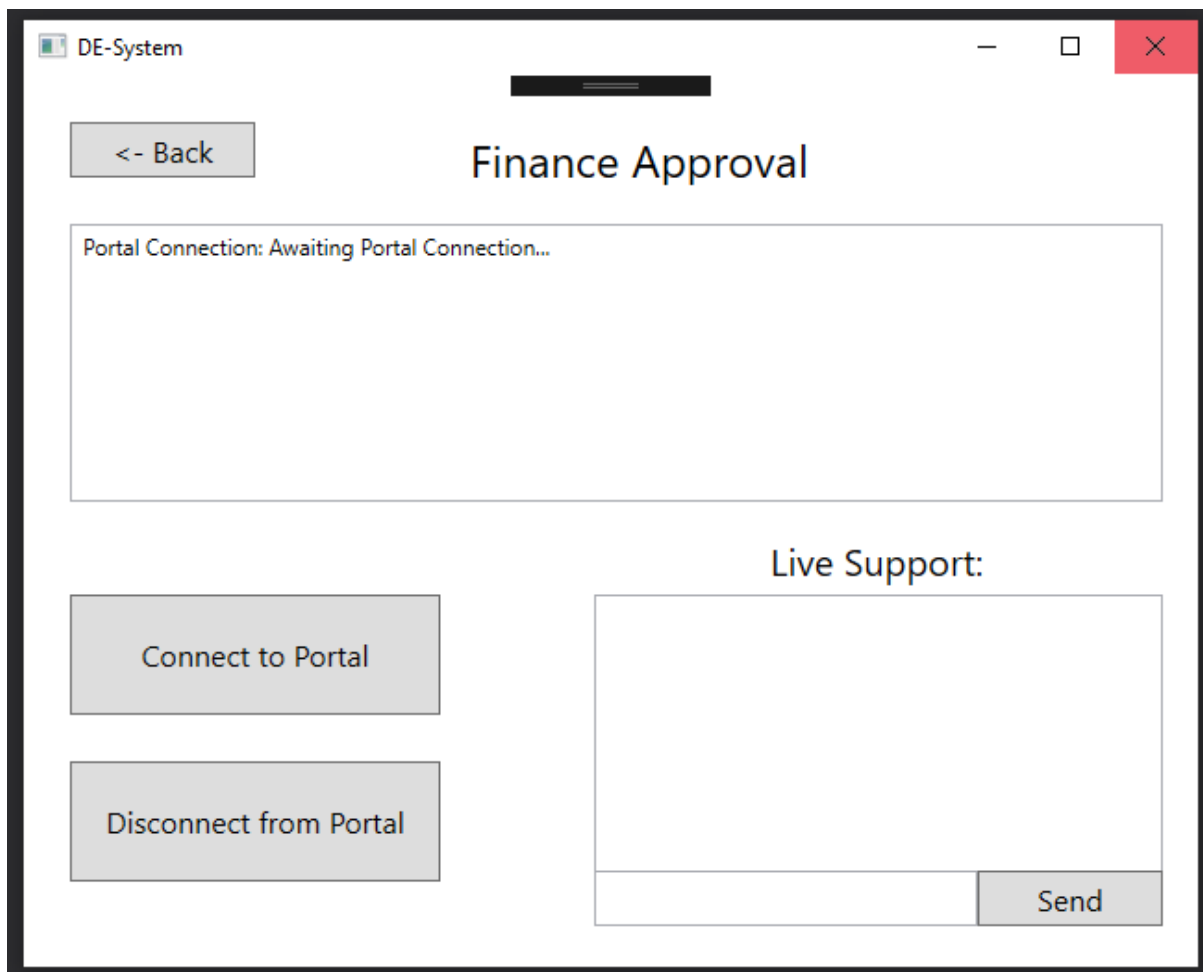
Figure 12. The three windows of the Inventory Control feature.



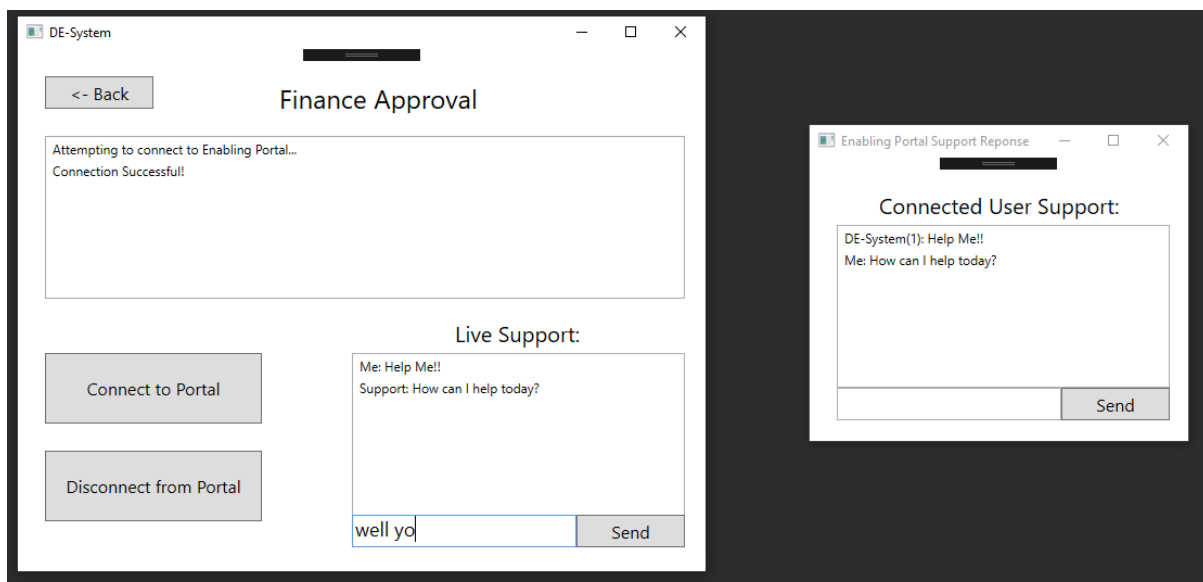
**Figure 13.** Loyalty Card Window.



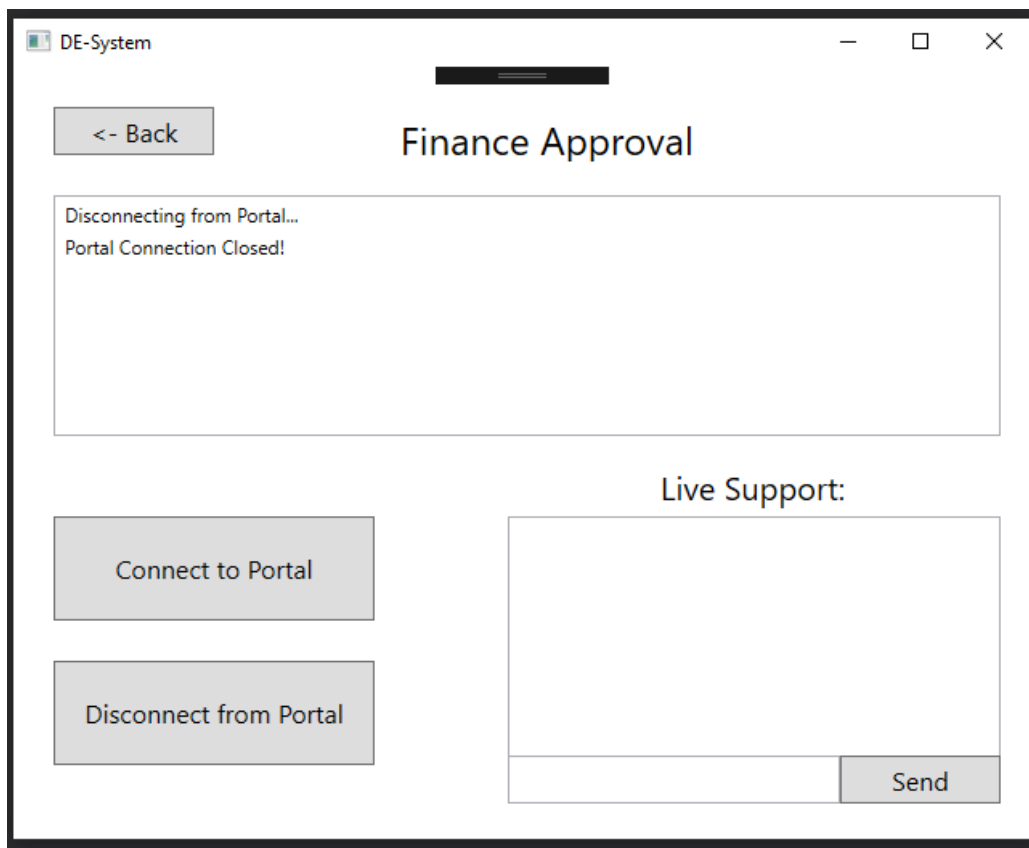
**Figure 14.** Loyalty Card single item sub-window.



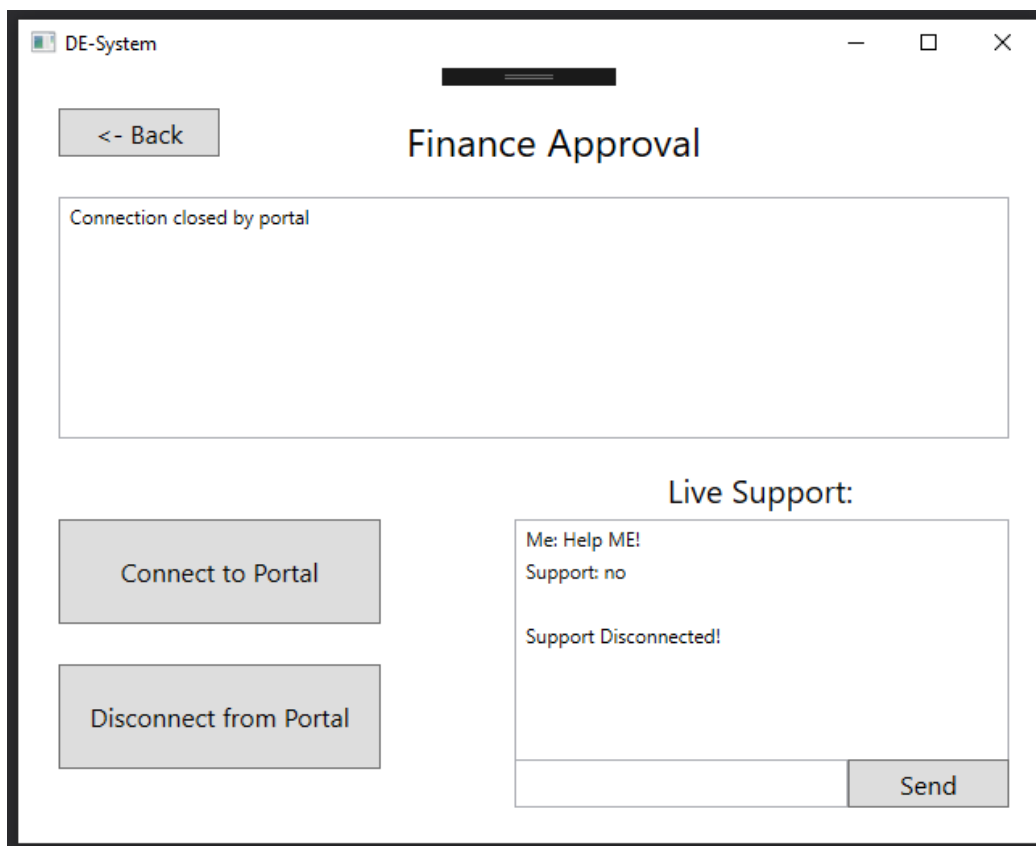
**Figure 15.** Finance Approval Window.



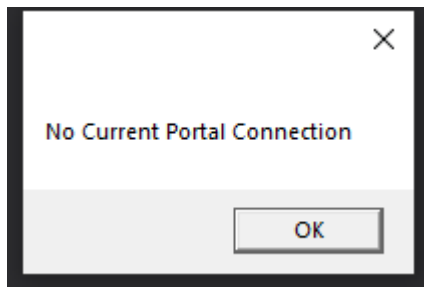
**Figure 16.** Finance Approval Window – Connected to Portal and chatting.



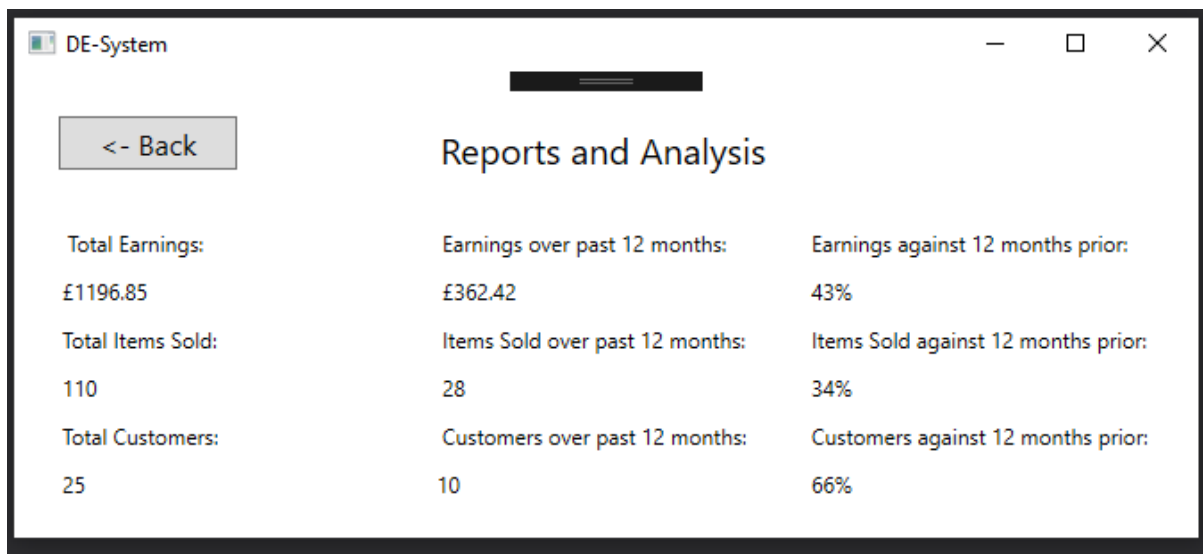
**Figure 17.** Finance Approval Window – Local Disconnect.



**Figure 18.** Finance Approval Window – Disconnected by Portal.



**Figure 19.** Finance Approval Window – One of the error message-prompts.



**Figure 20.** Reports and Analysis Window.