Software Architecture

Coursework

40340711

2019

Table of Contents

[2 Introduction 3](#_Toc26028753)

[3 Description of two architectures 3](#_Toc26028754)

[3.1 Pipes & Filters Architecture 3](#_Toc26028755)

[3.1.1 What is it? 3](#_Toc26028756)

[3.1.2 Why use it? 4](#_Toc26028757)

[3.1.3 Advantages 4](#_Toc26028758)

[3.1.4 Disadvantages 4](#_Toc26028759)

[3.2 THREE-TIERED Architecture 5](#_Toc26028760)

[3.2.1 What is it? 5](#_Toc26028761)

[3.2.2 Why use it? 5](#_Toc26028762)

[3.2.3 Advantages 6](#_Toc26028763)

[3.2.4 Disadvantages 6](#_Toc26028764)

[4 Architecture Comparison (SAAM) 6](#_Toc26028765)

[4.1 3 Tier Architecture 6](#_Toc26028766)

[4.2 Pipes and Filters 6](#_Toc26028767)

[5 Chosen Architecture 7](#_Toc26028768)

[5.1 Justification 7](#_Toc26028769)

[5.2 Three-Tier 7](#_Toc26028770)

[5.3 MVC – Object-Oriented Programming 7](#_Toc26028771)

[5.4 Heterogeneous Styles 8](#_Toc26028772)

[6 Design 8](#_Toc26028773)

[6.1 Analysis 8](#_Toc26028774)

[6.1.1 User Requirements 8](#_Toc26028775)

[6.1.2 MoSCoW Table 9](#_Toc26028776)

[6.2 Design 9](#_Toc26028777)

[6.2.1 Use Case Diagram 9](#_Toc26028778)

[6.2.2 UML Class Diagram 10](#_Toc26028779)

[6.2.3 Database 10](#_Toc26028780)

[7 Evaluation 11](#_Toc26028781)

[7.1 ATAM 11](#_Toc26028782)

[7.2 Design 12](#_Toc26028783)

[7.3 Implementation 12](#_Toc26028784)

[8 Testing 12](#_Toc26028785)

[8.1 Introduction 12](#_Toc26028786)

[8.2 Objectives 12](#_Toc26028787)

[8.3 Scopes 12](#_Toc26028788)

[8.4 Testing Methods 13](#_Toc26028789)

[8.5 Testing Strategy 13](#_Toc26028790)

[8.6 Data Validation 13](#_Toc26028791)

[8.7 Event Validation 18](#_Toc26028792)

[8.8 Functional Acceptance 20](#_Toc26028793)

[8.9 Navigational Test 21](#_Toc26028794)

# Introduction

The written report is a documentation for the requested DE-Store distributed system. The company has asked for a prototype to be developed with some functional requirements specified. The system is a distributed system first that integrates an architecture which will allow for future expansions as well as to keep it maintainable. For this, I have offered my expert advice below in the following chapters of my proposed architectures followed by the final decision of only one. I will be discussing the strengths and weakness of each and will go in-depth about the chosen architecture and how I will ensure the store requirements are met to the highest standard.

# Description of two architectures

The system being developed for DE-Store needs to be expandable and adaptable to future business needs. It will be important to change parts of the system as the store grows; therefore, it will be useful to choose an architecture which allows for specific area upgrades without affecting the rest of the system. The two architecture I have chosen to discuss are the **Pipes & Filters** and **Three-Tiered Architecture**.

## Pipes & Filters Architecture

The pipe and filter architecture is a simple concept only made up of 2 basic components. The input data on one end, apply a filter or transformation to the data then output the data on the other end. They are easy to understand in small implementation but may get overcomplicated in the larger one. I will discuss this more in further sections.

### What is it?

The 2 components it’s made of are the filters which are applied to the data and the pipes which are used to connect the filters together. They are used in a system which constantly has data flowing in on one end and gets passed through all the filters coming out at the other end. The filters may be either active or passive, the former will pull data from other filters and push their data out while the latter will have the data pushed into them or pulled out of them.

They do not hold any contextual or state data, they read in data from one end, transform it using an implemented filter and output it.

**Filter**: Can process a stream of input data to some output data.

**Pipe**: a channel that allows the flow of data between or among different filters.

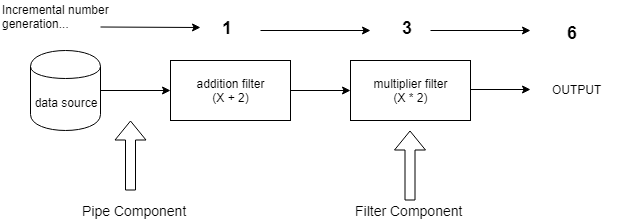


Figure 1. Example of a simple addition and multiplication filter process. Data is inputted from the data source which is an incremental number generator (1...2…3…). The filters are applied to the numbers. The first iteration can be seen above the components.

### Why use it?

The pipes and filters are useful when a large system can be broken down into smaller steps.

If a system requires flexibility and will often have to change some of its components, then it would be made much easier. As the filter components are highly interchangeable it would make it very easy to switch certain filter for another with perhaps an update algorithm or other changes. It would also allow for removing steps very easily.

### Advantages

* The pipes and filters can be grown easily into more complex ones
* You can reuse some filters in other applications or in the same application.
* They enable concurrency which means they can run, process and output with overlapping times. If they do not depend on each other.
* They have low latency and parallel computation
* They enable for rapid prototyping due to their low complexity when starting a project (may become complex later in development)

### Disadvantages

* As the complexity grows it may result in poorer performance
* Even though concurrency and parallelism are possible it does not guarantee an increase in performance if not implemented correctly
* Errors and mistakes could be passed on through the system making it harder to determine which filter is the cause
* Making interactive UI may become a challenge in complex systems

## THREE-TIERED Architecture

The three-tiered architected or the three-layered architecture is a client/server architecture in which the user interface (presentation), logic layer (business logic) and data access layer have been separated into separate components which are usually developed on different platforms. This splits the problems up and will make it much easier to manage and maintain the system.

The Three-tier can also implement a 3 layered MVC architecture where the presentation layer is further decoupled into (**M)**odels, (**V)**iews and (**C**)ontrollers. This allows for another separation of problems that make the UI tier of the system to maintain and expanded much easier which is very useful in e-commerce systems. See *Figure 2 for how the whole system can look.*

### What is it?

In a three-tier system, the layers are separated into their components.

**Presentation Layer**: In the presentation layer the user can interact with the UI controls like buttons, text fields, text boxes and drop-down menus. Depending on the control the event may be passed onto the logic layer which will take it on from there.

**Business Logic Layer**: This layer is usually made from a language like C# or Java which implement the business logic and functionality. For example, when a customer places an order then for it to be completed the logic layer will have to communicate with the data access layer to retrieve some data. It will then perform some checks or validation like ensuring there enough stock and the inputted client details are correct. From there the transaction can continue.

**Data Access Layer**: This layer will have direct communication with a database or data source like SQL Database, excel or text files. It will retrieve any requested data and give it to the logic layer for further functionality.

**Protocols:** 3 tier architecture protocols are HTTP which are web-based protocols. HTTP requests are sent to the business logic tier where it resolves the query and sends back a HTTP response back to the presentation layer.

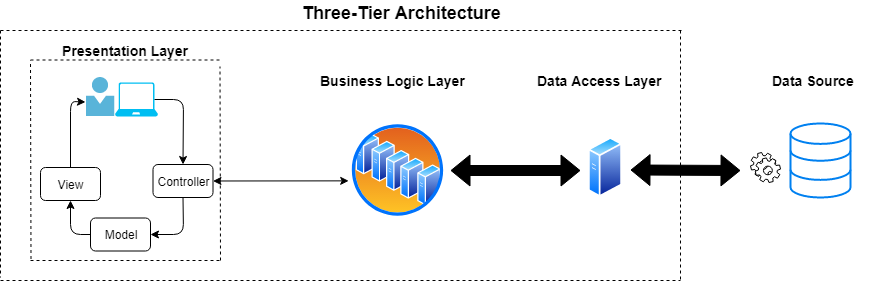


Figure 2. In the diagram, we can see the 3 main layers separated, Presentation Business Logic and Data Access. There is a database at the far right which would be accessible using the access layer. In the presentation layer, there can also be seen an MVC architecture which allows for better control over the view for the client.

### Why use it?

The three-tier architecture is developed so that the corresponding components cannot interfere all with each other and are limited to certain components. This allows the system to be expandable and upgradable as usually if the system was all in one then upgrading the logic, UI or database would require other components to be also looked at as they may be dependent on each other. In three-tier if for example the database is upgraded then all is needed is the databases layer to be upgraded as it’s the only part that communicates with it.

### Advantages

* **Maintainability:** The 3 tiers are physically separated and can be worked on independently which makes the system highly maintainable.
* **Scalability**: It’s good with increasing business need and is expandable
* **Reusability**: Due to the separation of tiers the components in each one can be reused to upgrade components on their respective tiers.
* Upgrading of the database is easy
* Since the bulk of the logic is spread across the rest of the layers then the client can remain small and be operated on any device.

### Disadvantages

* It has increased complexity in communicating with each layer. Since it’s split into 3 components extra care need to be taken to ensure performance.
* If changes are made top-down, so presentation layer needs new controls then the rest of the tiers will have to be updated so the databases layer can communication the new controls.
* Costs are increased as tiers must be installed separately in different areas

# Architecture Comparison (SAAM)

## 3 Tier Architecture

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Direct/ Indirect** | **Cost** |
| 1. Make improvements to create UI views for product and customers | D |  |
| 2. Components can be reused to add new functionality | D |  |
| 3. Add features to enable people with disabilities to use the system | I | 4-person day - programmer  2-person day – tester |
| 4.Implement 3rd party APIs like the financing portal to allow for additional functionality | I | 5-person day – programmer  3-person day - tester |
| 5. Make additional reports and statistics on the report view | D |  |
| 6.Improve the login security system and allow for users to change passwords | D |  |
| 7.develop application tier for different platforms like phones | D |  |

## Pipes and Filters

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Direct/ Indirect** | **Cost** |
| 1. Make improvements to create UI views for product and customers | I | 3-person day – programmer  2-person day – tester |
| 2. Components can be reused to add new functionality | D |  |
| 3. Add features to enable people with disabilities to use the system | I | 5-person day – programmer  3-person day – tester |
| 4.Implement 3rd party APIs like the financing portal to allow for additional functionality | I | 2-person day – programmer  1-person day – tester |
| 5. Make additional reports and statistics on the report view | I | 4-person day – programmer  2-person day – tester |
| 6.Improve the login security system and allow for users to change passwords | I | 4-person day – programmer  2-person day – tester |
| 7.develop application tier for different platforms like phones | I | 9-person day – programmer  3-person day – tester |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Architecture | S1 | S2 | S3 | S4 | S5 | S6 | S7 | summary |
| 3 tier-architecture | + | 0 | + | - | + | + | + | + |
| Pipe and Filters | - | 0 | - | + | - | - | - | - |

# Chosen Architecture

## Justification

The architecture I have chosen for the business is the Three-Tier architecture. This is because it gives the best **expandability** options and allows for development on separate components which will increase efficiency.3 Tier also provide great **scalability** and **maintainability** due to its **separation** of problems. While the overall architecture will be a three-tier one, to develop the client side of it I will also be implementing Object Orientated Programming (OOP) techniques. This will allow for a further decoupling of the Presentation layer into 3 components which help achieve separation of concerns of each component.

## Three-Tier

The first reason I have chosen this architecture is that it allows multiple developers to work on individual components. If any of the layers need to be revised or improved, then they can be independently worked on by different developers which will greatly increase the efficiency development time and reduce the overall costs. Further, since the components are decoupled it allows for them to also be individually upgraded. So, if the database needs to be upgraded the logic for user registration do not have to be touched or revised. This fits the store requirement very well as they expect lots of upgrades in the future as the business expands. It also makes the system much more maintainable as each tier helps with the separation of problems.

## MVC – Object-Oriented Programming

The **Model-View-Control** (MVC) pattern allows a clean separation of problems. The 3 components are:

**Model** (Data): The model contains the logic of the application and the operations that it can perform.

**View** (User Interface): The view receives data from the controller and responds by displaying the correct UI to the user.

**Controller** (Application Logic): This layer is responsible for handling the interactions and updates the model to reflect any changes then passes it to the view.

Furthermore, the client application will also be implanting the “**4 Pillars of OOP**” which are:

**Abstraction**: This process means that we only show what is necessary and hide the rest from the user.

**Encapsulation:** We encapsulate the data and methods by making them private and only making the properties public. This would make the data hidden and private which helps in keeping the code secure and maintainable.

**Inheritance:** This process will enable for code reuse and enhance the code. For common classes, we can write one class which will be used to inherit from, and we only need to edit that class to make any changes.

**Polymorphism**: This gives the ability to redefine methods for derived classes.

The combination of these architectures should result in a very maintainable software which can easily be read by new developers. New features and functionality can be implemented as the whole software is easily understood.

## Heterogeneous Styles

The heterogeneous term recognises that architectures do not exist in isolation but rather co-exits between each other and most of the time one architectures will contain parts of a different one. In the case of this project, I have decided to use a 3-Tier Architecture which is the physical separation of hardware with a 3-layered user application which happens on the presentation layer. These 2 decoupling methods break up the system in smaller components which make the system very maintainable and allow for maximum expandability. Major upgrades can be made to the overall systems independently and the MVC architecture allows for the presentation layer to be expanded upon with ease and new views and controller can be created to match the business requirements.

# Design

## Analysis

The client needs an administration tool which allows for control of their store online. This includes management of stock and their prices along with any offers, updates on lows stock and a page for analysing their client purchases with a useful statistic on how the store is performing.

### User Requirements

**Price Control**: The clients want to control their inventory like stock, price, offers and free delivery.

**Inventory Control**: The client wants to have an automated warning when stock is running low so that they can order more.

**Loyalty Card**: The client wants to have loyalty cards which will offer additional discounts to their employees or business clients who make lots of orders.

**Financial Approval**: The client wants to have a portal which allows users to buy products now and pay for them later.

**Reports and Analysis**: The client wants to have reports generated with statistics and analysis showing if items are selling well.

### MoSCoW Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Req. ID** | **Description (Feature to…)** | **Must** | **Should** | **Could** | **Won’t** |
| 1 | create new products and insert into database | ✔ |  |  |  |
| 1.1 | edit products in the database | ✔ |  |  |  |
| 1.3 | delete products from the database | ✔ |  |  |  |
| 1.4 | Allow different offers to be selected for products | ✔ |  |  |  |
| 1.5 | Offer free delivery for products | ✔ |  |  |  |
| 2.1 | add new customers into the database | ✔ |  |  |  |
| 2.2 | edit customers in the database | ✔ |  |  |  |
| 2.3 | delete customers from the database | ✔ |  |  |  |
| 3.1 | Create new orders in the database |  | ✔ |  |  |
| 3.2 | Delete order from the database |  | ✔ |  |  |
| 4 | Allow for different loyalty cards to be selected for customers |  | ✔ |  |  |
| 5 | Alert admin when stock is low | ✔ |  |  |  |
| 6 | Open a portal where financing checks can be performed |  | ✔ |  |  |
| 7 | Page with reports and statistics on the store like bar graphs |  |  | ✔ |  |
| 8 | Login system for security |  |  | ✔ |  |
| 9 | Separate employee and admin views |  |  |  | ✔ |
| 10 | Allow for download of reports and statistics |  |  | ✔ |  |

## Design

### Use Case Diagram

A close up of text on a white background

Description automatically generated

Figure 3. Use case diagram of the functionality in the application

The use case diagram shows the basic interaction the client will be able to perform once the system is developed. Price and client control will allow for a further interaction like adding new entries, editing or deleting entries from the database.

### UML Class Diagram

A close up of text on a white background

Description automatically generated

From the above UML class diagram, we can see a clear 3 way split between the presentation layer, business logic layer and the database access layer. It is easy to see why this architecture enables for future expansions as each tier encapsulates their own responsibilities.

### Entity relationship Diagram

A screenshot of a cell phone

Description automatically generated

### Database

The database is a local one which has 2 tables, one for the product and one for the customer. The datatypes are described below.

#### Customer

The customer table (tblCustomer)

* Id data type int
* CustomerId data type int
* FirstName data type varchar(50)
* LastName data type varchar(50)
* Email data type varchar(50)
* PhoneNo data type varchar(50)
* Address data type varchar(50)
* LoyaltyCard data type tinyint

#### Product

The product table (tblCustomer)

* Id data type int
* ProductId data type int
* Name data type varchar(50)
* Price data type int
* Stock data type int
* Category data type int
* Offer data type int
* Delivery data type tinyint

#### Order

The product table (tblOrder)

* Id data type int
* OrderId data type int
* ProductId data type int
* CustomerId data type int
* Quantity data type int
* Total data type float

# Evaluation

PRIORITY/DIFFUCLTY

## ATAM

|  |  |  |  |
| --- | --- | --- | --- |
| Utility | Performance | Data latency | **(M, L)** Minimise storage latency on DB to 200ms |
| **(L, M)** Critical alert messages will be delivered in less than <1 second |
| Response time | **(M,M)** System under load will be able to execute commands in <2 seconds |
| Modifiability | New product categories | **(M, L)** New categories can be added at any time to encompass more products |
| Change Loyalty Card Discounts | **(L, L)** Add additional loyalty card options |
| Availability | DDOS Attacks | **(L, H)** Upgrade system security |
| **(M,M)** Redirect packets if too many come from the same location |
| Data loss | **(M, M)** A database will always return all the data |
| Adaptability | Mobile phone UI | **(M, L)** web application is displayed correctly on phone |
| New business functionality | **(M, M)** New functionality can be implemented at any time and done fairly easily |

## Design

The designed prototype suits the requested requirements very well and has the quality attributes that make it a great architecture to be implemented in this domain. The design diagrams shown show an overview of all the classes involved and has a clear separation of tiers top-to-bottom and a final database at the end. The layer communicates by calling methods in the Processor classes and mapping the data received to Models which can then be used to display data in views. The presentation layer has been split into 3 layers where the client only sees the view and changes ones the Action for the Controller is called. This make is very easy to add additional features as you know exactly where to create classes and methods. If a new feature is getting added like being able to sort the products by price then you only need to code the ProductProcessor.cs to have a method which loads Product data from database and sorts the list then passes the model back to the presentation layer to be displayed.

If the database is to be upgraded or changed then it can make happen and the only change that would have to be made is the connection from the SqlDataAccess class to the database. This further proves that the system is adaptive and would be able to be expanded and changed as business needs required it to.

The database has only 4 tables being the Order, Product, Customer and Login. The Order, Product and Customer have a relationship where if an order is made then the customer id and product id are stored in the order table along with the total amount to pay. The order processor will take care of calculating the total and will take into consideration the loyalty card that the customer has by taking off a parentage depending on which card is owned.

## Implementation

### Presentation Tier Classes

A screenshot of a cell phone

Description automatically generated

### Business Logic Tier Classes

A screenshot of a cell phone

Description automatically generated

### Data Access Tier Class

A screenshot of a cell phone

Description automatically generated

### Controllers

The controllers call methods inside the Processor classes to either retrieve data or save data to the database. They use Models to achieve this by other saving data to a model and passing it to the Processor for saving or request for a model to be created in the processor class like a list of customers and passes it back to the controller to display the data.

### Models

The Model classes (CustomerModel, ProductModel, OrderModel, BarDataPointModel) are used to pass data around the application. If a new customer is made then the data inputted into the fields are saved into the model class, it then is passed to the customer processor which maps it to its own version of the model class which includes the unique ID for the row. It saves it to the database by creating a string SQL query and passes it to the SqlDataAccess.cs class which does the query and returns an int indicating rows affected, should be 1 if data has been inserted correctly. The same logic applies for editing and deleting rows in all the tables like product or orders.

### Processors

The processors are split (ProductProcessor, CustomerProcessor, OrderProcessor) and each has logic and methods to deal with their responsibilities. Each class creates a string which can be passed to the SqlDataAccess class for the query to execute.

### SqlDataAccess

The SqlDataAccess has 3 methods in it to load, insert or delete data from the database. A query is passed along with the data which is then communicated to the database by executing the query. The read data is then passed through the system to confirm it has been done correctly.

### Functions

### Processors

Processors have 4 main methods (Create, Load, Edit and Delete) with an extra one in some to implement additional functionality, GetLowStock which is in the ProductProcessot and creates a query which returns any stock under a specified number. The 4 main methods are exactly the same I each but are different in the sense they use a different model to save data.

#### Code



### SQLDataAccess

The SqlDataAccess class 3 methods being Create, Load and Delete. The create is executed inside the database and will return an integer if any rows are affected. If inserting a customer, it would return 1 to indicate 1 row has been affected. If the load method is called then it will turn a list of rows that match the query, this can then be used to display customers. Finally, the delete is executed with an ID which tells the database which row to delete.



### Controller

The controller is made up of ActionResults which return a specific view. Once one is called it will then create a model which will be mapped to some data which being access from the database by the data access layer. It will then return the view with a model passed into it to display the data.



# Testing

## Introduction

This test plan will be written about the current task I have taken of creating a distributed systems prototype for an e-commerce company. The system is taking in a lot of user inputs therefore a lot of constraints had to be put in place as to have uniform formatting that can be followed. All the testing for these will be planned out in this document and the test logs will be attached at the end of the document.

## Objectives

I will be using different test logs to try and capture the full range of options the user has when interacting with the software. All major bugs and crashes should hopefully be addressed and fixed before the product delivery date.

## Scopes

I will test the functionally of the program to ensure that it matches the requirement specification provided by the client. Each function will have its own test case and will be documented in a log.

A data validation test will be run to try and catch any unexpected input that may cause crashes to bugs. This will also be used to check that the program only accepts a specific text formatting. The events of the software will be tested to check all the user controls like buttons/dropdown menus perform the correct functions. The ease of navigation will be put under test to make sure each page can be reached or closed where applicable.

## Testing Methods

In this case, I will be using the black box testing approach. The code is not looked at but rather we will be feeding it test data and document carefully the outputs.

This approach helps the customer see that all the functional requirements are met and perform to their standards, the code and structure is not necessarily shown as they may not fully understand all the inner workings but rather they are only interested in the software working as intended.

## Testing Strategy

The test strategy we will be using is bottom-up testing. With this strategy, we will be looking at the smaller less significant parts of the software and then work out way through the software testing more of the individual functions until we can start combining them all together into one single program.

This strategy goes hand in hand with object-oriented programming as our program is already made up of different multiple different objects that can be tested by themselves. One they all behave and perform as intended they can be combined.

## Data Validation

Page: LoginView

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Items | Expected Results | Results |
| User and Password | Correct username and password | Takes user to home view | **Pass** |
|  | Incorrect username and password | Shows error screen | **Pass** |

Page: CreateCustomerView

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Items | Expected Results | Results |
| CustomerId | 798 | Accepts input | **Pass** |
|  | 43234 | Accepts input | **Pass** |
|  | dsfds | Displays error message | **Pass** |
|  | apple | Displays error message | **Pass** |
|  | oranges | Displays error message | **Pass** |
|  | @~:~{} | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| First Name | Chris | Accepts input | **Pass** |
|  | Jim | Accepts input | **Pass** |
|  | Steveooo | Accepts input | **Pass** |
|  | Yessir | Accepts input | **Pass** |
|  | Chris98 | Displays error message | **Pass** |
|  | Jimmy’@ | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Last Name | Smith | Accepts input | **Pass** |
|  | Harry | Accepts input | **Pass** |
|  | JJJJJJ | Accepts input | **Pass** |
|  | ACDFD | Accepts input | **Pass** |
|  | DWOR(\* | Displays error message | **Pass** |
|  | \*( | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Email | [Cdworczy@gmail.com](mailto:Cdworczy@gmail.com) | Accepts input | **Pass** |
|  | hsmth@yahoo.com | Accepts input | **Pass** |
|  | YESASHARRY£££@gmail.com | Accepts input | **Pass** |
|  | R@gmail.com | Accepts input | **Pass** |
|  | @.com | Displays error message | **Pass** |
|  | bob@com | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| PhoneNo | 06767867896 | Accepts input | **Pass** |
|  | 07734432987 | Accepts input | **Pass** |
|  | 07743932847 | Accepts input | **Pass** |
|  | 11111111111 | Accepts input | **Pass** |
|  | 12312312312 | Accepts input | **Pass** |
|  | 123 | Displays error message | **Pass** |
|  | 07731073248973 | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Address | 43 Larry Road | Accepts input | **Pass** |
|  | 123A Marry Land | Accepts input | **Pass** |
|  | LEGITADDRESS | Accepts input | **Pass** |
|  | YES | Accepts input | **Pass** |
|  | iquoeioUWqeoiqweoiqwouewqqwewopqiepoq | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| LoyaltyCard | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |

Page: CreateProductView

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Items | Expected Results | Results |
| ProductId | 798 | Accepts input | **Pass** |
|  | 43234 | Accepts input | **Pass** |
|  | dsfds | Displays error message | **Pass** |
|  | apple | Displays error message | **Pass** |
|  | oranges | Displays error message | **Pass** |
|  | @~:~{} | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Name | Cattot | Accepts input | **Pass** |
|  | Apple | Accepts input | **Pass** |
|  | iPad | Accepts input | **Pass** |
|  | Yessir | Accepts input | **Pass** |
|  | IPPADDD | Displays error message | **Pass** |
|  | Jimmy’@ | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Price | 67 | Accepts input | **Pass** |
|  | 7686 | Accepts input | **Pass** |
|  | 1 | Accepts input | **Pass** |
|  | 1000000 | Accepts input | **Pass** |
|  | DWOR(\* | Displays error message | **Pass** |
|  | \*( | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Stock | 12 | Accepts input | **Pass** |
|  | 233 | Accepts input | **Pass** |
|  | 1 | Accepts input | **Pass** |
|  | 123123231123 | Accepts input | **Pass** |
|  | @ | Displays error message | **Pass** |
|  | bob@com | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Category | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |
| Offer | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |
| Deliver | Checked Box | Accepts input | **Pass** |
|  | Unchecked Box | Accepts input | **Pass** |

Page: EditCustomerView

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Items | Expected Results | Results |
| CustomerId | 798 | Accepts input | **Pass** |
|  | 43234 | Accepts input | **Pass** |
|  | dsfds | Displays error message | **Pass** |
|  | apple | Displays error message | **Pass** |
|  | oranges | Displays error message | **Pass** |
|  | @~:~{} | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| First Name | Chris | Accepts input | **Pass** |
|  | Jim | Accepts input | **Pass** |
|  | Steveooo | Accepts input | **Pass** |
|  | Yessir | Accepts input | **Pass** |
|  | Chris98 | Displays error message | **Pass** |
|  | Jimmy’@ | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Last Name | Smith | Accepts input | **Pass** |
|  | Harry | Accepts input | **Pass** |
|  | JJJJJJ | Accepts input | **Pass** |
|  | ACDFD | Accepts input | **Pass** |
|  | DWOR(\* | Displays error message | **Pass** |
|  | \*( | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Email | [Cdworczy@gmail.com](mailto:Cdworczy@gmail.com) | Accepts input | **Pass** |
|  | hsmth@yahoo.com | Accepts input | **Pass** |
|  | YESASHARRY£££@gmail.com | Accepts input | **Pass** |
|  | R@gmail.com | Accepts input | **Pass** |
|  | @.com | Displays error message | **Pass** |
|  | bob@com | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| PhoneNo | 06767867896 | Accepts input | **Pass** |
|  | 07734432987 | Accepts input | **Pass** |
|  | 07743932847 | Accepts input | **Pass** |
|  | 11111111111 | Accepts input | **Pass** |
|  | 12312312312 | Accepts input | **Pass** |
|  | 123 | Displays error message | **Pass** |
|  | 07731073248973 | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Address | 43 Larry Road | Accepts input | **Pass** |
|  | 123A Marry Land | Accepts input | **Pass** |
|  | LEGITADDRESS | Accepts input | **Pass** |
|  | YES | Accepts input | **Pass** |
|  | iquoeioUWqeoiqweoiqwouewqqwewopqiepoq | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| LoyaltyCard | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |

Page: EditProductView

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Items | Expected Results | Results |
| ProductId | 798 | Accepts input | **Pass** |
|  | 43234 | Accepts input | **Pass** |
|  | dsfds | Displays error message | **Pass** |
|  | apple | Displays error message | **Pass** |
|  | oranges | Displays error message | **Pass** |
|  | @~:~{} | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Name | Cattot | Accepts input | **Pass** |
|  | Apple | Accepts input | **Pass** |
|  | iPad | Accepts input | **Pass** |
|  | Yessir | Accepts input | **Pass** |
|  | IPPADDD | Displays error message | **Pass** |
|  | Jimmy’@ | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Price | 67 | Accepts input | **Pass** |
|  | 7686 | Accepts input | **Pass** |
|  | 1 | Accepts input | **Pass** |
|  | 1000000 | Accepts input | **Pass** |
|  | DWOR(\* | Displays error message | **Pass** |
|  | \*( | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Stock | 12 | Accepts input | **Pass** |
|  | 233 | Accepts input | **Pass** |
|  | 1 | Accepts input | **Pass** |
|  | 123123231123 | Accepts input | **Pass** |
|  | @ | Displays error message | **Pass** |
|  | bob@com | Displays error message | **Pass** |
|  | \*EMPTY\* | Displays error message | **Pass** |
| Category | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |
| Offer | Dropdown menu option 1 | Accepts input | **Pass** |
|  | Dropdown menu option 2 | Accepts input | **Pass** |
|  | Dropdown menu option 3 | Accepts input | **Pass** |
| Deliver | Checked Box | Accepts input | **Pass** |
|  | Unchecked Box | Accepts input | **Pass** |

## Event Validation

Page: LoginView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Login” | Button Click | Calls login controller | **Pass** |

Page: HomeView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “**Product Control**” | Button Click | Calls product controller | **Pass** |
| (Button) “**Customer Control**” | Button Click | Calls customer controller | **Pass** |
| (Button) “**Financing**” | Button Click | Calls financing controller | **Pass** |
| (Button) “**Reports and Analysis**” | Button Click | Calls report controller | **Pass** |
| (Button) “**Email Box**” | Button Click | Calls email controller | **Pass** |

Page: CreatCustomerView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Create” | Button Click | Calls create method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: CreatProductView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Create” | Button Click | Calls create method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: EditProductView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Edit” | Button Click | Calls edit method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: EditCustomerView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Edit” | Button Click | Calls edit method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: DeleteCustomerView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Delete” | Button Click | Calls delete method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: DeleteProductView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Delete” | Button Click | Calls delete method | **Pass** |
| (Button) “Back To Homepage” | Button Click | Calls home action | **Pass** |

Page: ReportsView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Back” | Button Click | Calls home method | **Pass** |

Page: EmailBoxView

|  |  |  |  |
| --- | --- | --- | --- |
| Control | User Interaction | Expected Results | Results |
| (Button) “Back” | Button Click | Calls home method | **Pass** |

## Functional Acceptance

Page: LoginView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User inputs username and password | Username and password are checked and validated | **Pass** |

Page: CreateCustomerView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed create button | Data is saved to the database | **Pass** |

Page: EditCustomerView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed edit button | Correct data is edited inside the database | **Pass** |

Page: DeleteCustomerView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed create button | Customer entry is removed from the database | **Pass** |

Page: CreateProductView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed create button | Data is saved to the database | **Pass** |

Page: EditProductView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed edit button | Correct data is edited inside the database | **Pass** |

Page: DeleteProductView

|  |  |  |
| --- | --- | --- |
| Input Test Case | Expected Results | Results |
| User pressed create button | Customer entry is removed from the database | **Pass** |

## Navigational Test

|  |  |  |  |
| --- | --- | --- | --- |
| "Caller" Page | Control | Expected Result | Results |
| LoginView | “Login” button clicked | HomeView is opened | **Pass** |
| HomeView | “Manage products” button clicked | ProductsListView is opened | **Pass** |
|  | “Manage Customers” button clicked | CustomerListView is opened | **Pass** |
|  | “Financing” button clicked | FinancingView is opened | **Pass** |
|  | “Reports and Analysis” button clicked | ReportsView is opened | **Pass** |
|  | “Email Box” button clicked | EmailView is opened | **Pass** |
| CustomerListView | “Edit” button clicked | CustomerEditView is opened | **Pass** |
|  | “Create” button clicked | CustomerCreateView is opened | **Pass** |
|  | “Delete” button clicked | CustomerDeleteView is opened | **Pass** |
|  | “Back” button clicked | HomeView is opened | **Pass** |
| ProductListView | “Edit” button clicked | ProductEditView is opened | **Pass** |
|  | “Create” button clicked | ProductCreateView is opened | **Pass** |
|  | “Delete” button clicked | ProductDeleteView is opened | **Pass** |
|  | “Back” button clicked | HomeView is opened | **Pass** |
| EmailView | “Back” button clicked | HomeView is opened | **Pass** |
| ReportsAndAnalysisView | “Back” button clicked | HomeView is opened | **Pass** |