

College of Science and Technology

School of Science and Technology

# SOFT30121: Advanced Analysis and Design

# Systems Analysis Design and Implementation

*By*

*N0683858: Hamid Mujtaba*

*N0688119: Elliot Harding*

*N0756079: Steffan Walker*

*N0704050: Michael Abraham*

*N0772082: Hamed Alshebli*

*T0084963: Haowei Goh*



NTU Stores Management System

￼

**SOFT30121: Advanced Analysis and Design i**

**Systems Analysis Design and Implementation [i](#_30j0zll)**

**1.** **Introduction To Our System 3**

**1.1.** **Introduction 3**

**1.2.** **Architectural Pattern 3**

**1.3.** **Design Patterns 3/4**

**2.** **Design Documentation 4**

**1.1.** **Architecture Diagram 3**

**1.2.** **Deployment Diagram 3/4**

**1.3.** **Process Diagram 3**

**1.4.** **Structure Diagram [3](#_26in1rg)**

**1.5.** **Component Diagram 3**

**3.** **User Help documentation [3](#_lnxbz9)**

**4.** **Variable Naming Conventions [3](#_lnxbz9)**

**5.** **Acceptance test plan 4**

**3.1.** **Functional tests 4**

**3.2.** **Performance requirements 4**

**3.3.** **Unit tests 4**

**6.** **Conclusion**

**3.2.** **References**

# Introduction to Our System

## Introduction

//TO DO

## Architectural Pattern

Our system uses MVC architecture. This architecture comes with its share of advantages and disadvantages…

**ADVANTAGES:**

### Supports Collaboration

One benefit of using an MVC architecture is the division of the system into the three separate components. The Model, View and Controller are loosely coupled which means there is a reduced chance that a change in one will affect the other. One instance being if the View of the system changes there will be no need to change the Model. This loose coupling allows multiple developers to work on the system without the fear of interfering with another’s work.

### Faster Development

An additional benefit of the previously mentioned loose coupling is the hastened development. By having components where change in one is less likely to affect the other, developers can spend less time waiting for others to complete a feature and can focus more on developing other components.

### Easy Modification

MVC has a good capacity for future expansion/modification due to the reduced dependencies between components. This allows for additions to be made to components without having to change a great amount in others. For instance, if a client requests additional features after development has already begun, their implementation would be easier due to the loose coupling. This makes MVC useful for developing systems where requirements may change frequently.

### Multiple Views

Since the View component of the system is separated from the business logic, it is possible to support multiple views. This is useful for systems with multiple types of users that could require their own individual view of the system. For instance, in a business system the administrators would have greater privileges than regular staff and as such would need a different system view to reflect this.

### Code Reusability

Since the business logic and the view of the system are separate entities, the code has an improved reusability since it is not closely tied to any system. This provides a large advantage to organisations that develop a multitude of programs since they can reuse parts of the program in other software they are developing.

DISADVANTAGES:

### Complexity

Separating the system into multiple components can introduce complexity in relation to how control flows between the view, controller and model. Due to this, a good level of understanding for the architecture is needed in order to design a system that makes proper use of MVC’s loose coupling. A poor understanding of the architecture can lead to difficulties debugging due to the complicated flow of control.

### Unsuitable for Small Applications

The divided workload and loose coupling of components is only advantageous to large applications with numerous developers. For small applications with fewer developers and a reduced need for complexity, the MVC architecture can be more work to utilise than is worth. Due to the greater complexity of the architecture, a small team may end up over engineering their program and creating a greater workload for themselves than was needed.

## Design Patterns

Below are the various design patterns used in the system.

### Singleton

A singleton is a class with only one instance during runtime and a global point of access. This pattern is used in the design of the ‘DatabaseComms’ class. While the ‘DatabaseComms’ class is designed as a static class, the closest design pattern to it is that of the singleton.

The singleton design pattern was used for the ‘DatabaseComms’ class due to its widespread use throughout the system. It made sense to not waste the resources required to instantiate and destruct an object of a class which would be used in every page of the system.

Lazy initialization; the delaying of the creation of an object until it is required could have been an alternative pattern for the ‘DatabaseComms’ class. This would have meant that resources of the class would have only been added to memory if used, something which would have proved highly effective as not all features of the class are used within all use cases of the system.

### Module

The module design pattern groups related elements, such as methods or classes into a single theoretical entity. This design pattern was also used for the ‘DatabaseComms’ class as a way of grouping together all database networking methods.

### Publish/Subscribe

This describes behavioural design pattern, in which a one-to-many dependency among objects change for one object resulting in all its dependents being notified. This design pattern is used throughout the system, an example being the deleting of a ‘TransactionRow’ control causing all other invoice rows to update their position to maintain a consistent series of ‘TransactionRow’ controls.

An alternative publish/subscribe to consider for the ‘TransactionRow’ controls is the servant behavioural pattern. This defines common functionality for a group of classes, i.e. a helper class. The helper class would manage the ‘TransactionRow’ controls.

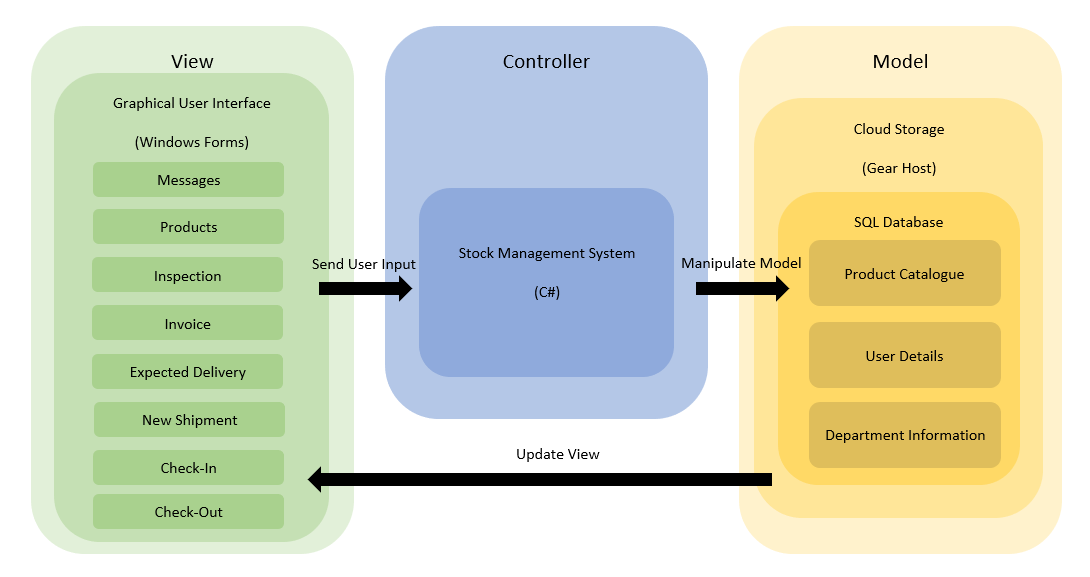
### Façade

A façade design pattern provides an interface to a more complex subsystem. Each of the page classes of the system utilize this design pattern, to make the system easier to use.

# Design Documentation

## Architecture Diagram

### Diagram



### Purpose of Diagram

MVC, or Model-View-Controller, is a software design pattern used for designing the flow of control within a system. The architecture tends to vary from project to project but in general makes use of three separate entities that form the overall system.

The View relates to the user’s interface and is how they see/interact with the systems underlying data model. One advantage to using MVC is that a multitude of views can be implemented to reflect the needs of the different users. The view allows the user to input data and interact with the system through traditional UI elements such as buttons, combo-boxes etc. Interactions with the view are then sent to the controller.

The controller is responsible for validating any input data from users before sending the relevant request to the model. The controller effectively acts as a medium between the view and the model, sending requests that the model can understand with information it has been given in the view. When the model sends back the required information, the controller then updates the view.

The model is the data centric portion of the architecture. It is responsible for storing, managing and providing any data that has been requested of it from the controller. Depending on the storage method used, the model may have to interact with a database to retrieve the data it requires. The business logic of the system is contained within the model and is required to process the data. After processing, the model sends the processed data to the controller which then updates the view.

### How System Relates to Our Architecture Diagram?

The architecture our group has chosen for this project is MVC or Model-View-Controller. This architecture was deemed most appropriate based on the project’s requirements. The group consisted of 6 members meaning the Model, View and Controller workload could be split up to allow quicker development. Furthermore, since MVC is flexible, it would allow changes to the interface (view) to be made by one member without affecting any progress made to the model by another member and thus reducing wasted development hours. The provided architecture diagram should provide a clear view of the subcomponents that make up the system in addition to the interactions between them.

The View consists of a graphical user interface the user can utilise to interact with the system. The interface was designed using Windows Forms which is included with Visual Studio. This allowed the View and Controller parts to be developed using the same IDE which allowed quicker and simpler development of the application. Windows Forms is widely used among developers to create well designed interfaces with ease. Furthermore, it is developed by the industry leading Microsoft who are known to develop efficient, robust and powerful software.

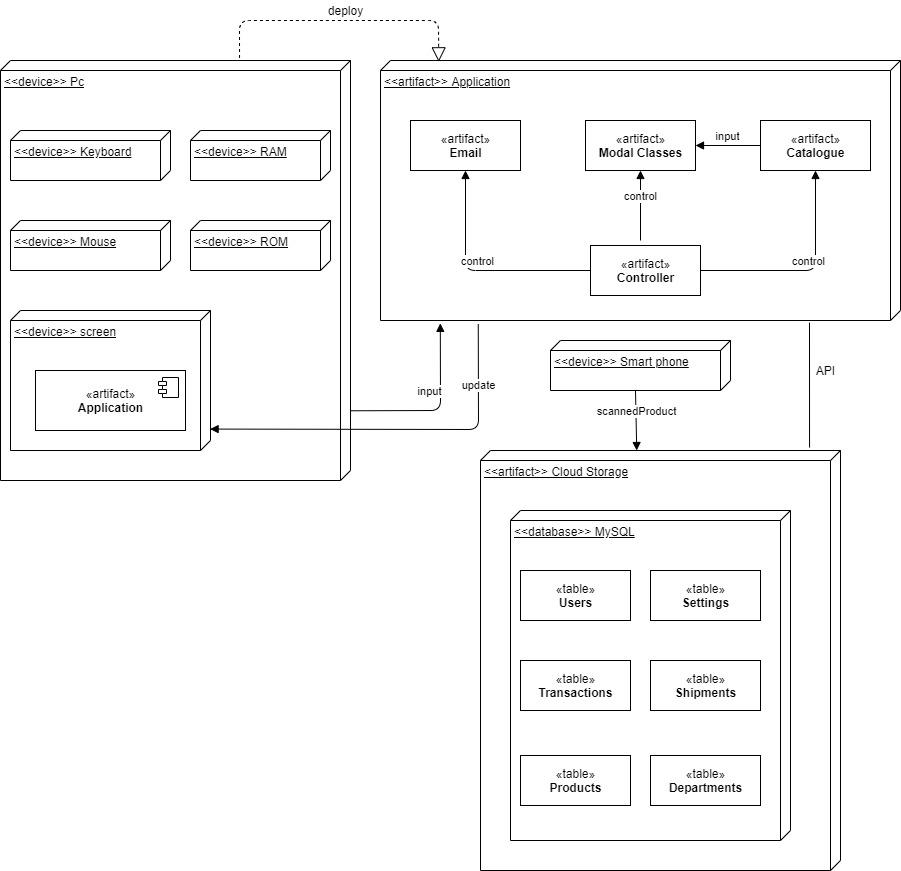
The Controller is responsible for receiving and validating the user input from the view before sending the relevant instructions to the model. Validation ensures the input data is not erroneous and if so, prevents it from being stored within the model. The controller uses listeners to wait and respond to specific events occurring in the view such as a button being clicked. The data from the input fields is then sent to the model for processing after it has been validated. The controller is programmed using C# which is supported by Visual Studio and allows for easy data retrieval from the view.

The model is responsible for storing the systems data and processing it depending on the instructions it receives from the controller. The interaction is done through the ‘DatabaseComms’ class which is written in C# and the data is stored in an SQL database on a cloud server. After the model has processed instructions from the controller and updated the stored data, it will then update the view.

An example of the system from a Model-View-Controller perspective would be the user clicking on an item from the catalogue. The view would recognise which item has been clicked and sends that data to the controller. The controller would then make a request to the model for the products page and the information regarding the product. The model would then update the view with the desired product page along with information associated with said product.

## Deployment Diagram

### Diagram



### Purpose of Diagram

The deployment diagram is used to show the underlying hardware used in the system and the software that runs on this hardware. The diagram communicates how the software system and the hardware work together to execute different tasks.

### Why Use It?

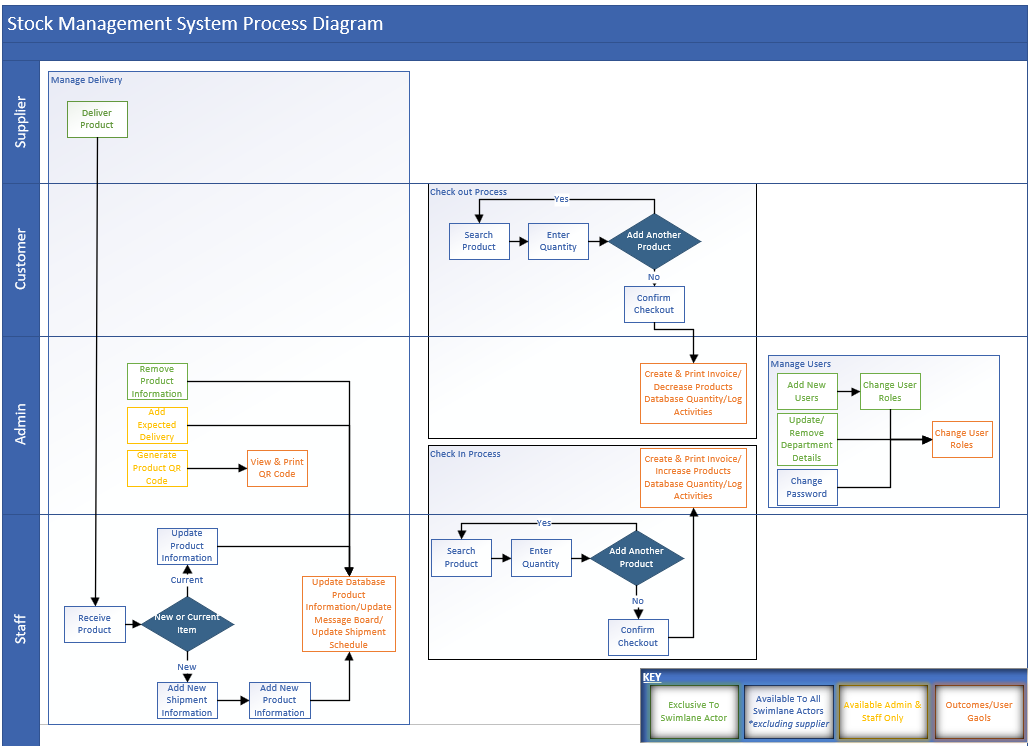
Where deployment shows hardware and software components of a system, it also acts as a guideline to setup the proposed system. It is for this reason, a deployment diagram was used, so any organization/person who wishes to implement the system, knows the resources required and has the knowledge of what to expect from the system.

### How System Relates to Deployment Diagram

The deployment diagram shows the fundamental resources required to carry out the tasks of the system. It shows a computer system that has input devices to feed data to the system and internal devices that run the application; which is also shown and is further elaborated through mentioned modules of modal classes, controller class and email (which are the underlying software components to run the system). A database is connected to the software system and tables shown that keep records required by the system and a smartphone is present to scan the products.

## Process Diagram

### Diagram



### Purpose of Diagram

Process Diagrams give a visual representation, in linear order, of the interactions between an actor and the system. Separated by swimlanes to show accountability between each connected event, the diagram denotes what needs to be accomplished to progress through each process’s timeline.

### Why Use It?

We have used a process diagram to clarify every activity occurring in each process and illustrate a logical path for the collection of flows that form our stock control management programme.

### How System Relates to Our Process Diagram?

Our diagram divides the system into several major processes that transpires across multiple pages of the programme:

* **Manage Delivery:**

This process represents the procedure that unfolds when a new shipment of products is supplied and documented into the system. After the transfer of goods between the supplier and the staff member, this diagram highlights each event (carried out in the systems ‘Shipment’ flow) required to update or add new stock information to the database.

* **Check out/Check in Process:**

Both these processes occur within the same concept of a ‘Basket’ flow, sharing the same functionality to append products to a list, however the outcome of each process delivers altered results; either increasing or decreasing the quantity associated to the product within the database whilst creating a log of these activities. Additionally, the ‘Check out’ process composes an invoice ready to be sent to the appropriate body.

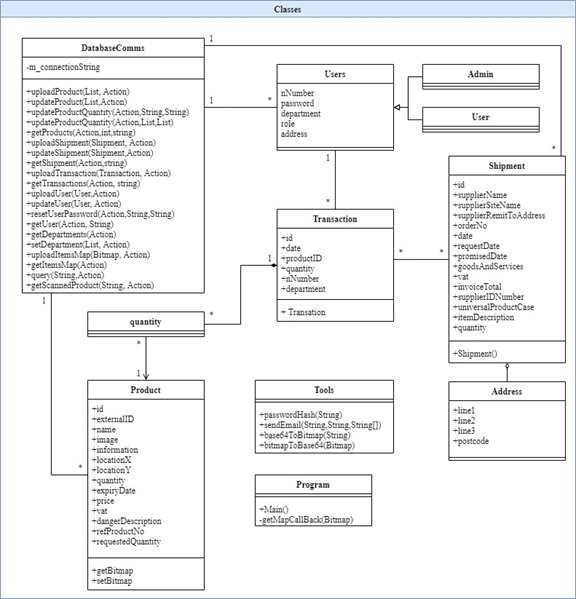
* **Manage User:**

This highlights the administrative capabilities the system offers. Access through different actors invoke separate functionalities; admin possess the highest clearance of authorisation, permitting access to additional features.

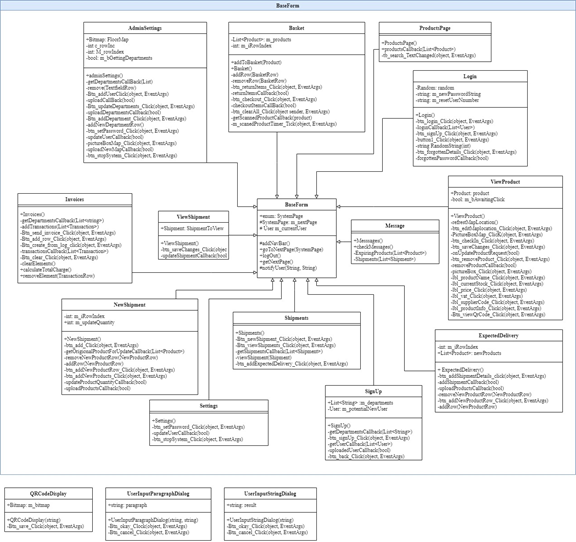
As our system involves user roles with overlapping functionalities so provided below the diagram is a key to illustrate which events and activities are also available to other actors.

## Structure Diagram

*Diagram*

**

*Figure X+1: Class Diagram (System)*

**

*Figure X+2: Structure Diagram (Interface)*

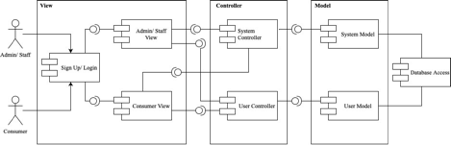
*Why Use It?*

The main purpose of structure diagram is to model the static view of an application. It is used to bring understanding and modifiability to the information being input and output. It consists of classes, interfaces, associations and collaborations.

*How System Relates To Our Structure Diagram?*

## Component Diagram

*Diagram*



*Purpose of Diagram*

The component diagram shows the three different entities that comprise the developed system which are the Model, View and Controller. These entities are loosely coupled meaning each component has few dependencies on the other. Within each of these entities are the components that direct the flow of control through the system. The Model, View and Controller are responsible for handling different aspects of the system as can be seen in the Architecture diagram.

*Why Use It?*

The developed system utilises multiple components with the Model, View and Controller modules. The component diagram should provide some insight into the flow of control between the multiple components in addition to the interaction differences between different users.

*How System Relates to Our Component Diagram?*

As seen in the component diagram, the staff, admins and users all access the same signup page but are directed to their own corresponding views upon logging in or signing up. The views are customized for each type of user and reflect the privileges available. For instance, all users may access the catalogue but only admins may view and make changes to it. Any interaction made in the view is then sent to the general user controller which validates and converts it to then be processed by the user model. The model accesses the database for any data it requires before updating the system model. After any updates have occurred the system controller makes any required changes to the users view.

# User Help documentation

### Consumer (Regular User):

When the application is executed, the user will be presented with the ‘Login’ interface. The user will have the option to login in if they already have a registered account or they can use their NTU details to register. The ‘Registration’ will require the user to input their username, their department and choose a password (re-enter password) [Figure UG1].

To recover a password, click on the ‘Forgot Password’ button which will prompt for a username [Figure UG2]. Once a valid username has been confirmed, an email with a new password will be sent. A change of password can be done by navigating to the ‘Settings’ page and choosing a new password [Figure UG3].

Once successfully logged in a products page with the navigation menu at the bottom will appear [*Figure UG4*]. The products page will include all the products available for the consumers in store, which gets displayed with their name and a picture. The user can browse the catalogue for the products they wish to add to their basket. You can also search for the product using the search bar which will filter the products displayed [Figure GU5].

Once a product is clicked, a screen displaying all the product information will appear [Figure GU6]; such as the name, product description, an image and a map of the products location in the store. Press on the ‘check in/out’ button after confirming the quantity which adds the product(s) to the basket. This will allow the user the option of browsing for more products or to go to the checkout page where the basket can be edited. Alternatively, a scanning app is available on their phone to scan the products which are then added to the basket.

After product searching is completed, the user should navigate to the checkout page where the products they added in the basket are displayed [Figure GU7]. The user will still have the option to edit the basket if they want to remove/add products or increase/decrease the quantity. Once finished the user can checkout which will update the system and log the transaction.

If a product is accidently checked out or is no longer needed the user is able to ‘Return Items’; this is located in the the checkout page from the navigation menu. They will first need to find the product again from the catalogue (or scan it) and add it to the basket first before they can click on the ‘Return Item’ which amends the stock amount.

### Staff:

The (employee) staff will login through the same login page as consumer staff; however, they will have more accessibility and more features available in the application. Staff accounts can be created through the normal signup page.

Two additional options become available to staff on the navigation menu. This gives more control over the system as they can modify products, create invoices, add shipments and much more.

To change an existing product, the staff will need to find the product from the catalogue or search it using the search bar; click the product to view it [Figure GU8]. Click on the name of the product, stock amount, price, VAT, Supplier ID and product description to display a popup textbox allowing staff to change details. Location of the product marked on the map can be changed by clicking the ‘Edit Map Location’ and clicking on the new location which will mark it with a red cross. A products QR code can be seen by clicking on the ‘View QR Code’. QR code can be saved on the local storage, in different formats, by clicking ‘Save’. ‘Save Changes’ must be clicked to store changes made or click the ‘Remove Product’ to remove it from the system completely.

On the invoice page [Figure GU9], the staff/admin users can view the transaction log by each department or all departments and choose the dates to search for specific transactions. To view the transaction log, click on the dropdown menu to select the department needed or select ‘all departments’ option to view the transaction logs for all departments. Then select the ‘Date from’ and the ‘Date to’ to view the transactions between a certain periods; click the ‘Fill Form log’ to display the data on the GUI. Screen can be cleared by clicking the ‘Clear’ button to view more logs for different departments. Transactions can be added manually by clicking the ‘Add Row’ button which creates a row on the form. To create the invoice of the data displayed on the GUI in an ‘.exe’ document press the ‘Create and Send Invoice’ button.

Users can view incoming shipments on the Shipment page.   
Incoming shipments (expected delivery) results can be filtered by changing the ‘Date from’ and ‘Date to’. Each result has a view option which when clicked will navigate the user to another page showing the details of the selected shipment. To add a new expected delivery, click ‘Add Expected Delivery’; a new page will appear allowing users to add the required data to register a new delivery. If ‘Add Row’ is clicked, details for registering product being shipped may be added (i.e. product name, description, price, etc.).

To stop the system, navigate to the ‘Settings’ page and press the ‘Stop System’ button. When clicked, the system will shut down.

*Admin:*

Admin capabilities are very similar to the employee (staff) with a few added features available. To manage users' information, go to the ‘Settings’ page [Figure GU10]. New employee or admin users can be added by selecting the privilege level from the dropdown. Map of the stores can be changed by clicking on it and choosing a new one from the local storage. Add departments by clicking the ‘+’. They can remove departments by clicking the ‘X’ button and update the database to save the changes by clicking the ‘Update Departments’.

# Code Naming Conventions

For the systems code variable and function naming conventions were outlined. Reasons for using conventions included:

* Reducing effort needed to read and understand source code
* To allow for easier debugging of source code, such as knowing the scope of member variables.
* Code reviewers can focus on issues other than syntax

The naming conventions are as follows:

|  |  |
| --- | --- |
| **Code element** | **Rule** |
| Functions | CamelCase |
| Variables | All variables are in CamelCase   * Class member variables start with m\_ * Static public member variables start with an upper-case letter * Constant variables start with a c\_ |
| Controls | Control variable names start with the type of control they are followed by an underscore. For example; ‘txt\_name’ is a text control |
| Dialogs | Dialog classes contain the word Dialog within their name. |

## Acceptance test plan

## Functional tests

### User: Consumer

### User: Admin & Employee (Staff)

## Performance requirements

//TO DO

## Unit tests

//DOING

## Conclusion

//TO DO

## References

<https://en.wikipedia.org/wiki/Software_design_pattern>

Oliver, L. & Luukkala, V. (2016). On UML’s Composite Structure Diagram.

<https://www.researchgate.net/profile/Ian_Oliver9/publication/268374023_On_UML's_Composite_Structure_Diagram/links/54f60d230cf27d8ed71d4af3.pdf>

Bell, D. (2004). UML basics: The component diagram. <https://www.softwareresearch.net/fileadmin/src/docs/teaching/WS13/SE/UML_basics-_The_component_diagram.pdf>

Genero, M. & Manso, T. & Visaggio, A. & Canfora, G. & Piattini, M. (2007). Building measure-based prediction models for UML class diagram maintainability.

<https://link.springer.com/article/10.1007/s10664-007-9038-4>