1. Introduction
2. Architectural pattern
3. Design patterns
4. Alternative design patterns
5. Diagrams
6. Architecture diagram
7. Structure diagram
8. Deployment diagram
9. Process diagram
10. Variable naming conventions
11. Testing & Results
12. Conclusion

**Introduction**

**Architectural pattern**

Using mvc

Describe mvc

Advantages

Disadvantages

**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 in which a change for one object results 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 to consider to publish/subscribe for the TransactionRow controls is the servant behavioural pattern. This defines common functionality for a group of classes, ie a helper class. The helper class would manage the TransactionRow controls.

Façade

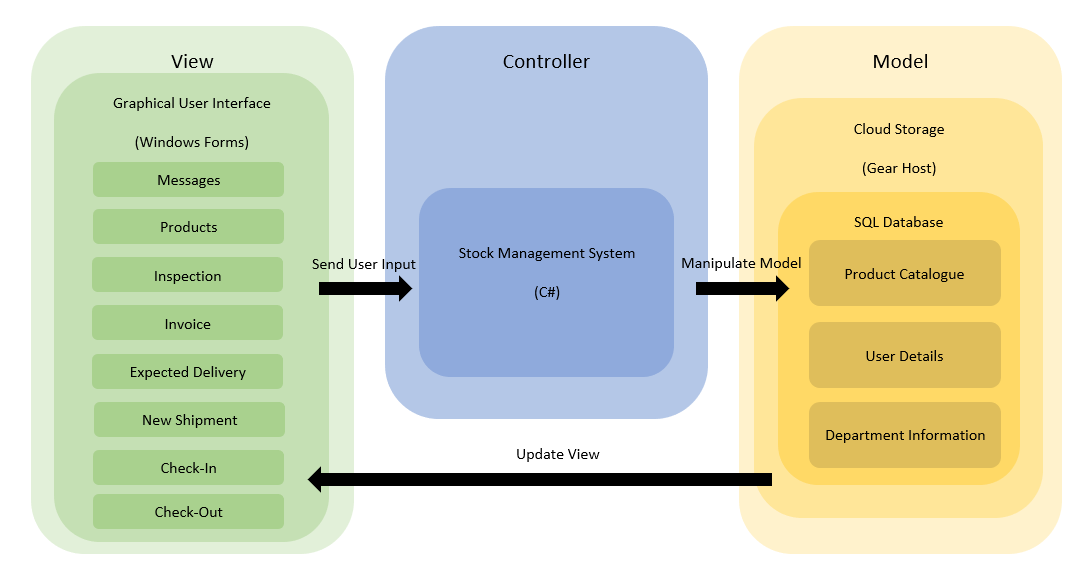
A façade design pattern provides an interface to a more complex subsystem. Each of the page classes of the system utilize this design pattern, to make the system easier to use.

**Diagrams**

**Architecture diagram**

**Description of architecture**

**Diagrams purpose**



The architecture the 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 particular 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 an 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.

**Structure diagram**

**Description of structure**

**Diagrams purpose**

A structure diagram is a development tool used in demonstrating the relationship between every parts of a system. It models how the parts cooperate to create the whole process, to outlining the details from the smallest part’s objects and classes being used to program the system.

It is important to identify and know how constructs are being used in reality and how close these usages are from or to the creators’ aim. It visualizes how a system works from the initial input, to processing and, finally, to the planned (Oliver & Luukkala, 2006).

Component Diagram

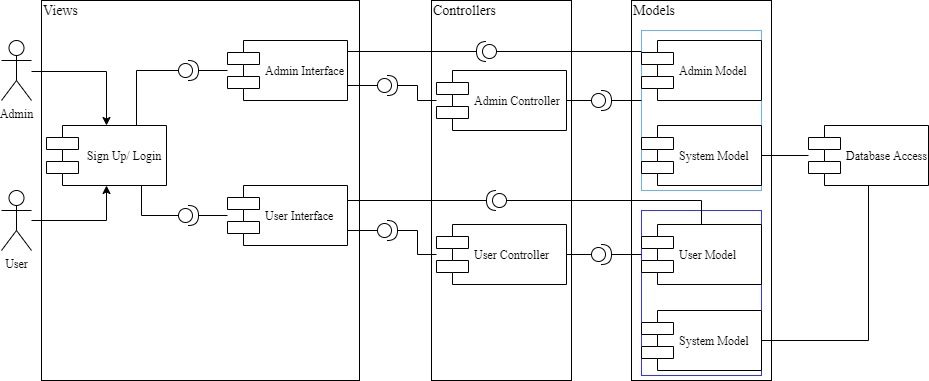


Figure X: Component Diagram

The component diagram’s main purpose is to show the structural relationship between the components of a system.

Figure X show the component diagram for the system. It has been developed to the MVC architecture pattern that separates the system into three main logical component which is views, controllers and models. Strong relationship between them used to handle specific development aspects of our system.

In Figure X, view is the interface. After login, the interface will distinguish users and admins by login username. After differentiating, they would be logged into different interfaces. Controllers used to be an interface between model and view components to process all the request, using model components to process data, and interact with the view to render the final output. For example, when the user clicks on different buttons to request data from database, it will request from the view to the model through the controller. Model represent either the data that is being transferred between the View and Controller component or any other business logic-related data. It used to read data from database and gain request from controller or view.

The component diagram is a very important architecture diagram create early on project but it worth spans the life of the system (Bell, 2004).

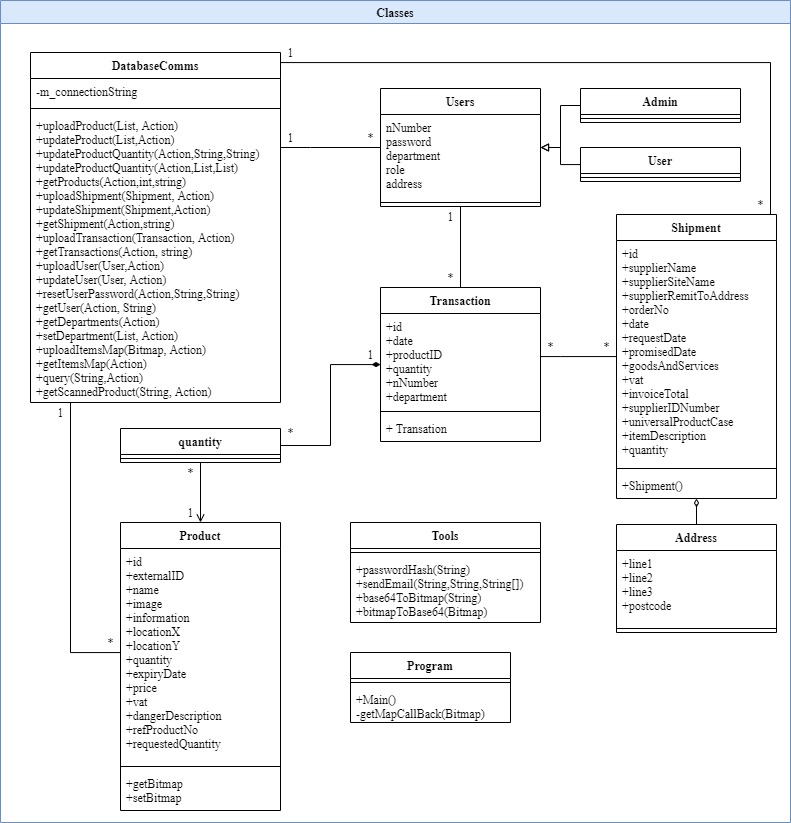
Class Diagram

Figure X+1: Class Diagram (System)

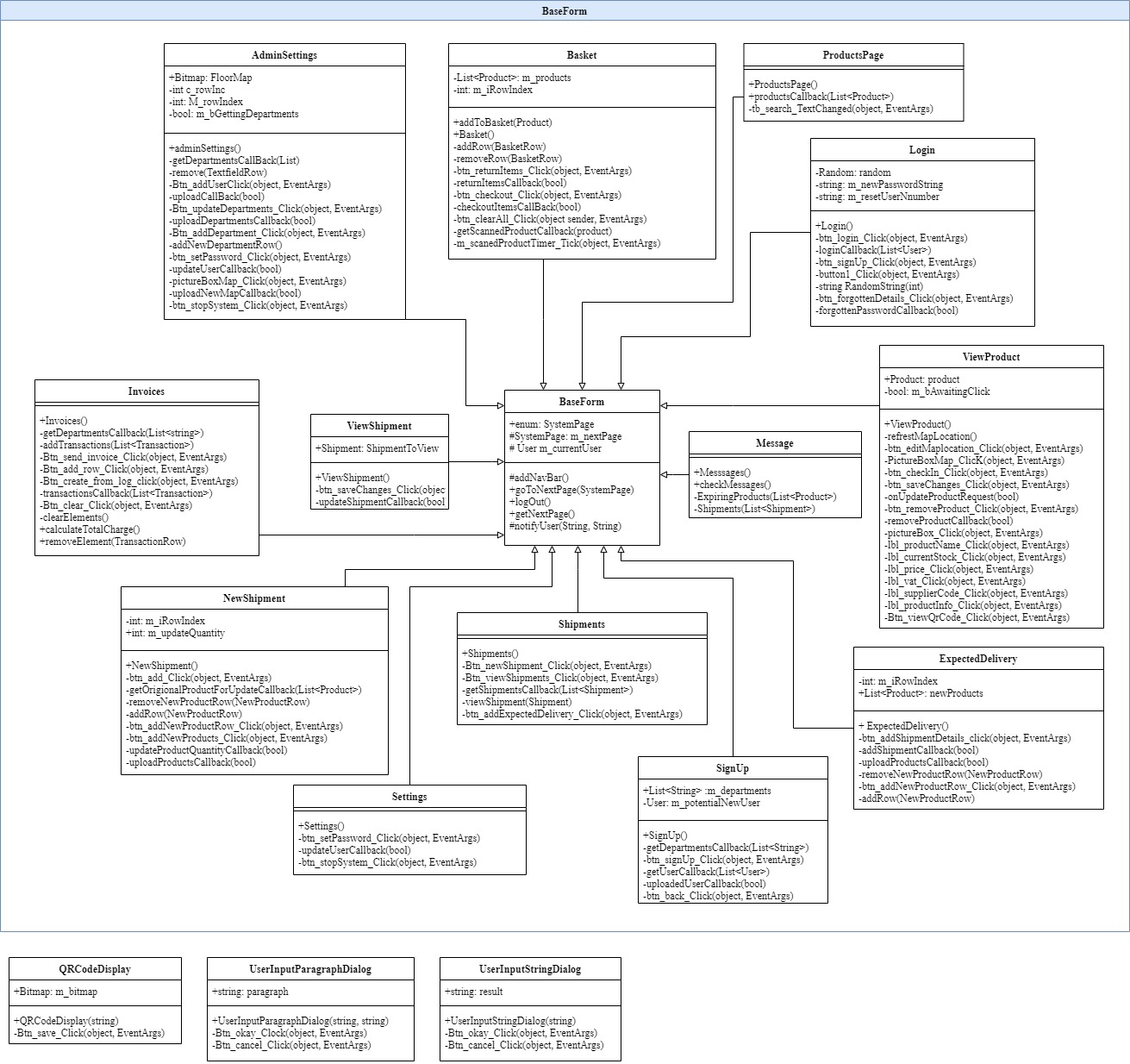


Figure X+2: Class Diagram (Interface)

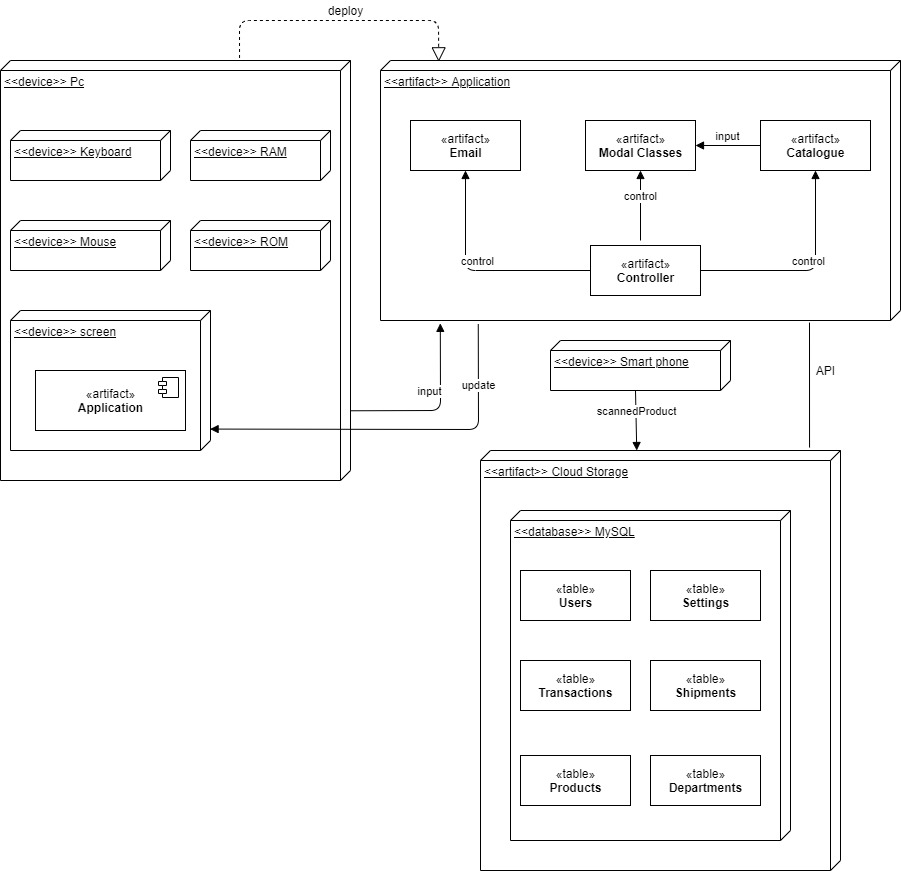
The main purpose of class diagram is to model the static view of an application. It is understandability and modifiability. It consists of classes, interfaces, associations, and collaboration. It could be built on the basis of early measures, in particular, measures that detention structural complexity through relations and generalizations. (Genero & Manso et al., 2007).

In Figure X+1, Class Diagram (System), DatabaseComms, Users, Transaction, Product, Shipment, Address, Tools and Program are main classes. It is easier to understand the relationship between each class. Admin and user classes are derived from users. Address is a part of shipment, there are an aggregation and transaction with quantity are composition. Tools and program classes are out of system.

In Figure X+2, it has a BaseForm saving all the system page. It has AdminSettings, Basket, ProductPage, Login, ViewProduct, Message, ExpectedDelivery, SignUp, Shipments, Settings, NewShipment, ViewShipment, Invoices classes. There are derived from BaseForm. On the other hand, QRCodeDisplay, UserInputParagraphDialog and UserInpitStringDialog is out of BaseForm.

Class diagrams proposal some benefits for builder to show data models no matter simple or complex, easier understand the general overview of the modelling of an application and it is a details charts that best part of any specific code needed to be programmed and executed.

**Deployment 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 get the desired task done.

**Why Use It?**

Where deployment shows hardware and software components of a system, it also acts 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 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 smart phone is present to scan the products.

**Process diagram**

**Diagrams purpose**

**Description of processes**

**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. |

**Testing & Results**

**Conclusion**

**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>