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**

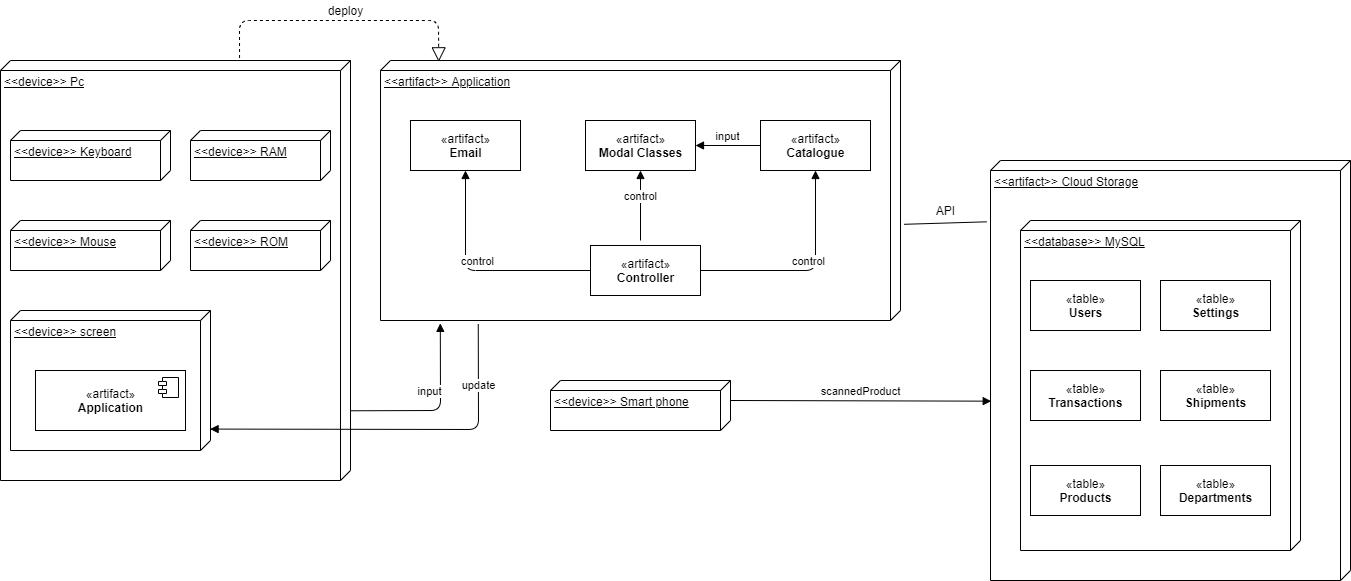
**Diagrams purpose**

**Structure diagram**

**Description of structure**

**Diagrams purpose**

**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 proposed system is an inventory management system. It is run on a computer machine, which is installed in multiple locations. The system uses a database to keep records of the products, shipments and users. New products are added through input devices and an order can be made through scanning a product through a smart phone. When an order is completed, an invoice is emailed to the customer.

The deployment diagram shows the fundamental resources required to carry out the above tasks. 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>