DESIGN DOCUMENTATION FOR NTU STORAGE TRACKING SYSTEM

Group Name - Eksi

Alex Humphries (N0643498) + Daniel Holland (N0697611) + Krystian Michalik (N0702465) + Noor Al Aji (N0789108) + Luke Barrett (N0699220) + James Earle (N0719214)

Contents

[1 - Introduction](#_30j0zll) **2**

[2 - Architecture](#_1fob9te) **2**

[2.1 - Architectural Pattern (Front-End & Back-End)](#_2et92p0) 2

[2.1.1 – Advantages of MVC](#_tyjcwt) 3

[2.1.2 – Disadvantages of MVC](#_3dy6vkm) 3

[2.2 - Design Patterns](#_1t3h5sf) 3

[2.2.1 – Advantages of (Design Pattern Name)](#_4d34og8) 4

[2.2.2 – Disadvantages of (Design Pattern Name)](#_2s8eyo1) 4

[2.2.3 - Alternative Design Patterns](#_cc1ypbpfaqpu) 4

[2.3 - Architecture Diagram](#_lflk972k0dd8) 5

[3 - Physical Deployment](#_17dp8vu) **6**

[3.1 - Deployment Diagram](#_3rdcrjn) 6

[4 - Behaviour Diagrams](#_26in1rg) **6**

[4.1 - Sequence Diagram](#_o6ziszxpsr8p) 7

[4.2 - Process Diagram](#_euay4lwovq41) 9

[5 - Structure Diagrams](#_3j2qqm3) **10**

[5.1 - Class Diagram](#_4i7ojhp) 10

[5.2 - Component Diagram](#_aey9ny40iawf) 11

[6 - System Security](#_1ci93xb) **11**

[6.1 - Database Usage](#_45baq6qfblgj) 11

[6.2 - API Usage](#_kx63d06rdcgq) 12

[6.3 - Security Evaluation](#_37135hbzg9fl) 12

[7 - Major Project Changes](#_qg2oe079medv) **13**

[8 – User Help Documentation](#_pgmpld9njvv3) **13**

[9 - Testing and Results](#_3whwml4) **13**

[9.1 – Functional Tests](#_qsh70q) 14

[9.2 - Performance Requirements](#_rzd6bgsvhr7q) 18

[9.3 - API Tests (Back-end Testing)](#_h9cqgg8onx6j) 20

[9.4 - Front-End Testing](#_c0m8k1khb4mv) 21

[9.5 - Results](#_3as4poj) 21

[10 – Conclusion](#_1pxezwc) **21**

[11 – References](#_49x2ik5) **21**

[12 - Bibliography](#_2p2csry) **23**

[13 - Appendix](#_147n2zr) **23**

# 1 - Introduction

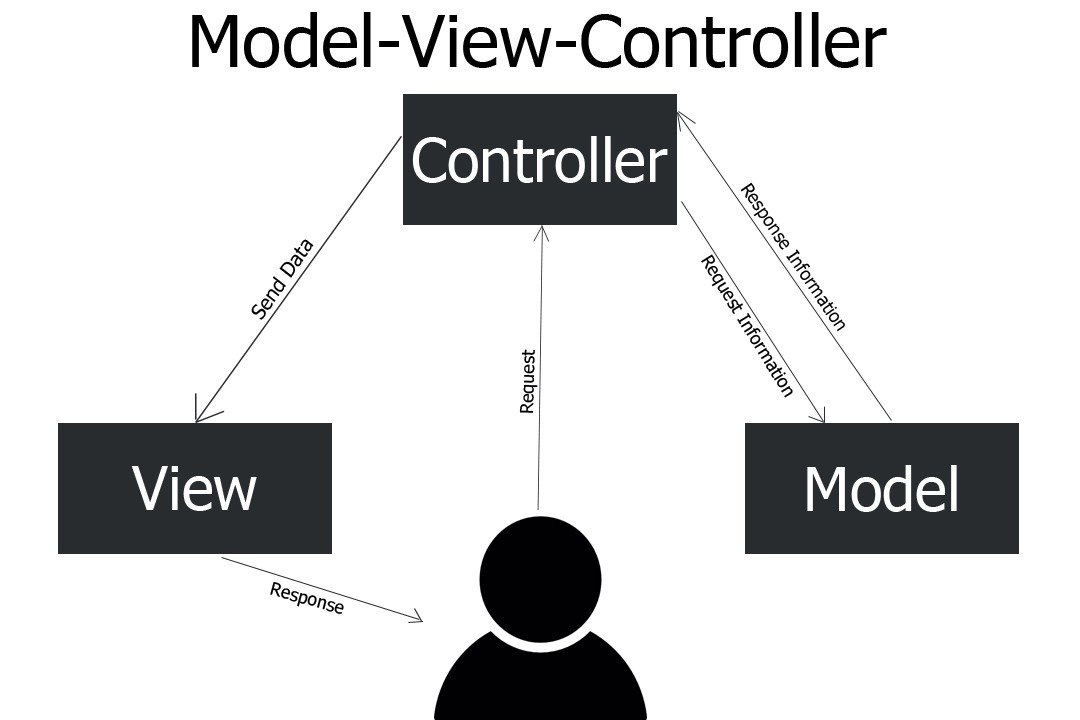
This report has the purpose of functioning as an analysis of the completed ‘storage tracking system’ designed for the usage of the storage department of Nottingham Trent University. Initially, this system was proposed to this team in early October, with the completion date is set for the 14th of February of the following year, with the main focus of the proposal being to develop a piece of software that can help streamline the work of the departments staff members and help them migrate to a much more organised electronic tracking system as opposed to their current paper-based system. To help determine if the final version of the designed system meets the requirements that were gathered from the representatives of the storage department via the various held interviews/Q&As, a full systems analysis will be conducted within this report. Alongside this system analysis, the inner workings of this system will be dissected in varying degrees to demonstrate its functionality, specifically how it achieves its presented features. Finally, this report will function as an examination of the work completed on this project by this team, analysing what went well, what didn’t and ultimately what will need to be done in future projects to ensure the same or similar mistakes are avoided.

# 2 - Architecture

In software terminology, the word ‘architecture’ is used to describe the shared understanding and mutual agreement that exists between a team of developers that relates to the way that a particular system is organised (Medium, 2019). Within this section of the report, the architecture of the storage tracking system that this team has built will be analysed to determine the patterns that were followed alongside relevant reasoning.

## 2.1 - Architectural Pattern (Front-End & Back-End)

In terms of what architectural patterns were followed in the creation of this system, this team worked following the “Model–view–controller (MVC)” approach (Atwood, J, 2019). Specifically, this is one of the most well-known types of pattern to follow for the UI of a system and places a focus on dividing the logic of a program into three interconnected elements, following as:

* **Models** - These represent information or data, either in the form of a single object or a structure of objects. The model is used to directly manage the data, logic and rules of the application.
* **Views** - This is a visual representation of the systems model, functioning as what is known as a ‘presentation filter’. This section retrieves all of the data necessary for its presentation aspect to function from the model through the use of requests. 
* **Controller** - This is used as a link between the systems current user and the system itself. Controllers work by providing the user with interactable inputs, presenting them in appropriate places on the screen. Additionally, controllers provide the means for user output by presenting the user with menus or other means of giving commands and data, passing any and all data it receives to the systems view.

### 2.1.1 – Advantages of MVC

Behind every architecture choice, there is reasoning that helps to explain why this particular architecture was chosen for this project out of the wealth of others that exist. For this project, MVC architecture was chosen due to its multiple benefits that closely relate to this type of project (Interserver Tips, 2019). Below, a table of these identified benefits complete with a description follows as:

|  |  |  |
| --- | --- | --- |
| **Advantage ID** | **Title** | **Description** |
| 1. | Quicker application development. | With the logic of an application being split into 3 parts, it is much easier for a group to quickly split the work between themselves and develop the different parts of an application. |
| 2. | Easier group collaboration. | With MVC it is much easier for a group to work on a project as the three distinct segments stop team members from getting in each other's way. |
| 3. | Straightforward debugging. | Due to the structure of the project being split up into different components, it is much quicker and easier for a group to find and fix a bug they encounter. |

### 2.1.2 – Disadvantages of MVC

However like with any architecture there are often disadvantages that make development using it difficult, with a list of flaws that that team have identified with MVC following as:

|  |  |  |
| --- | --- | --- |
| **Disadvantage ID** | **Title** | **Description** |
| 1. | Complex structure. | Splitting the logic into different segments may lead to the code of the project being complex and difficult to understand at first glance, making it vital that a team properly structures and comments their code, as well as working to regularly communicate with each other. |
| 2. | Strict method rules. | In order for MVC to be utilised correctly, strict rules regarding the architecture implementation must be implemented through the design of a project, a task that may become increasingly challenging in large groups. |

## 2.2 - Design Patterns

Often, with software projects of this scale problems are run into that may lead to project delays, as well as necessitate the implementation of project design changes. A common way to avoid this problem is with the usage of ‘design patterns’, which are a kind of development paradigm used to provide a general repeatable solution to a commonly occurring problem in the development of software (Sourcemaking, 2020). Although similar to the above-mentioned architecture pattern, the main difference between the two terms is that the architecture of an application refers to its larger structure and organization, whereas a design pattern refers to a method of solving a specific type of problem. There are multiple major benefits that occur through the usage of design patterns, including:

* **Saving Time for Developers** - Design patterns are already specified and well documented, helping provide an industry-standard approach to solve a problem found within a piece of software that developers find to be recurring. Through its usage, a design pattern, therefore, may help developers save time with their project, allowing them more time to work on other critical aspects of a system.
* **Better Quality Code** - Using design patterns promotes the reusability of written code, helping to ensure that it is not only more robust but also highly maintainable. It helps in reducing the total cost of ownership (TCO) of the software product.

#### Dependency Inversion

The server-side software uses dependency inversion to separate the declaration and implementation of methods. Dependency Inversion separates these, and in our software uses a separate object, ApplicationContext, to inject the implementation into the application. This allows for the implementation of components to be changed without causing knock on effects, as the interface which defines the behaviour is kept separate. It also makes unit testing easier, as the application can be injected with a mock implementation which gives deterministic results. A downside of this is that DI can be initially more difficult to grasp for new developers than DI-free code, and can increase the bloat of the program.

#### Model-View-Controller

The client uses the MVC design pattern. This is a pattern used for developing User Interfaces and Web Applications. Model holds the data, the View describes how it is displayed, and the Controller responds to input and mutates the data. This keeps the User Interface in a clean and maintainable structure.

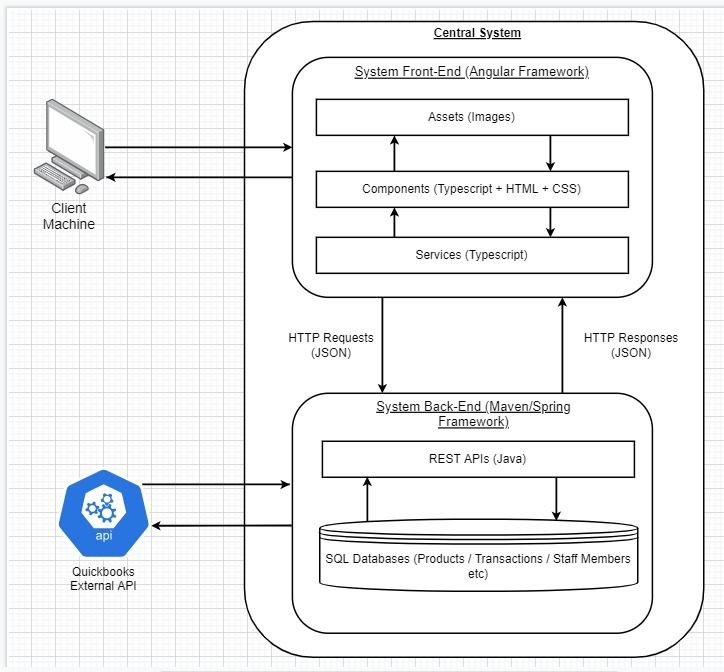
### 2.2.3 - Alternative Design Patterns

#### Singleton Pattern

A Singleton is a class that can have at most one instance. This could have been used in implementing our Implementation classes for the dependency inversion. Instead of this, the ApplicationContext lazy-loads one instance of the non-singleton class, and returns this to each request for the class. This is different from a singleton because the code ensuring that there is only one instance of the class is in the ApplicationContext rather than the class itself.

## 2.3 - Architecture Diagram

Despite the details of how the system was created included above, it is often inadequate to help a reader visualise what the actual architecture of the final version of a project looks like and how it has ultimately been structured. To help negate this, an architecture diagram has been drawn and included below. Specifically, this type of diagram is used as a kind of graphical representation of a set of concepts, that are part of the architecture of a particular system, including aspects such as the systems elements and components and ultimately what methods were used to link these aspects together (Edrawsoft, 2020).

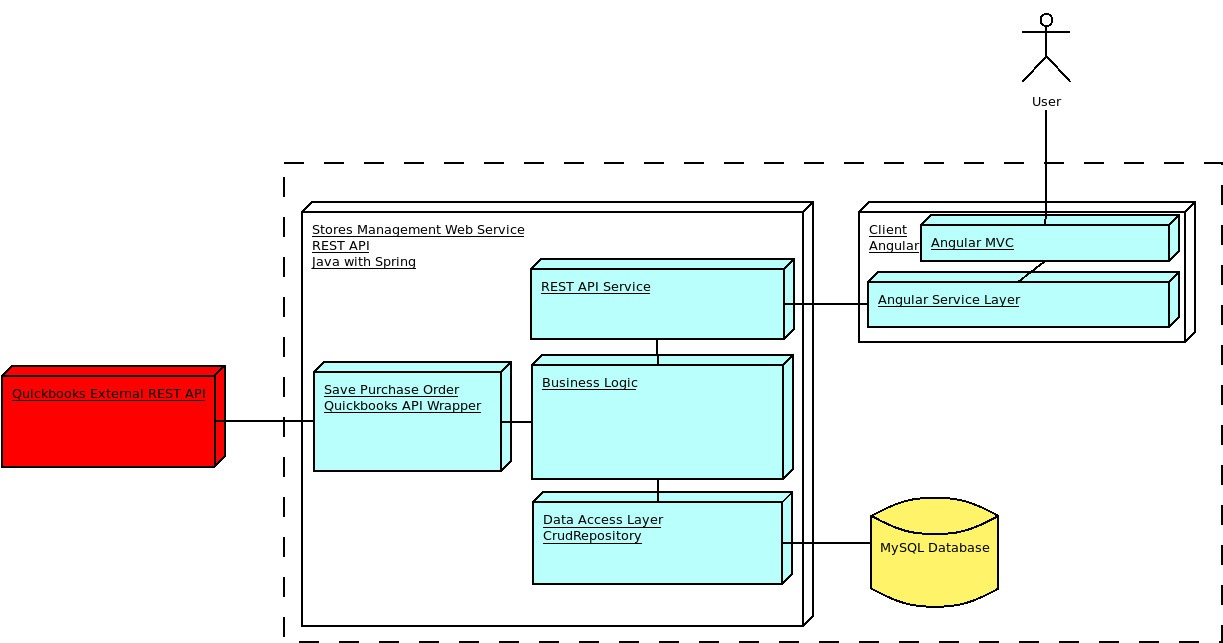


# 3 - Physical Deployment

Over the course of the operation of a system, various background processes are often running at any given moment either keeping track of certain variables or staying idle until a user interacts with the system, with the storage tracking system designed by this team being no different (Bobology, 2019). Within the following section of this report, various diagrams will be included and explored that relate to the systems physical deployment, specifically taking a look at what the systems inner features are doing during runtime, why they perform these actions and how these actions relate to the completion of the systems initial set functional and non-functional requirements, many of which are only carried out once a user interacts with the system.

## 3.1 - Deployment Diagram

Specifically, a ‘deployment diagram’ is a well known visual way of demonstrating the configuration of any and all run time processing nodes and the inner components that live on them within a particular system (Visual Paradigm, 2019). These kinds of diagrams are vital for showing the inner components of a system and how they relate to its connected hardware, with a deployment diagram for the designed storage tracking system following as:

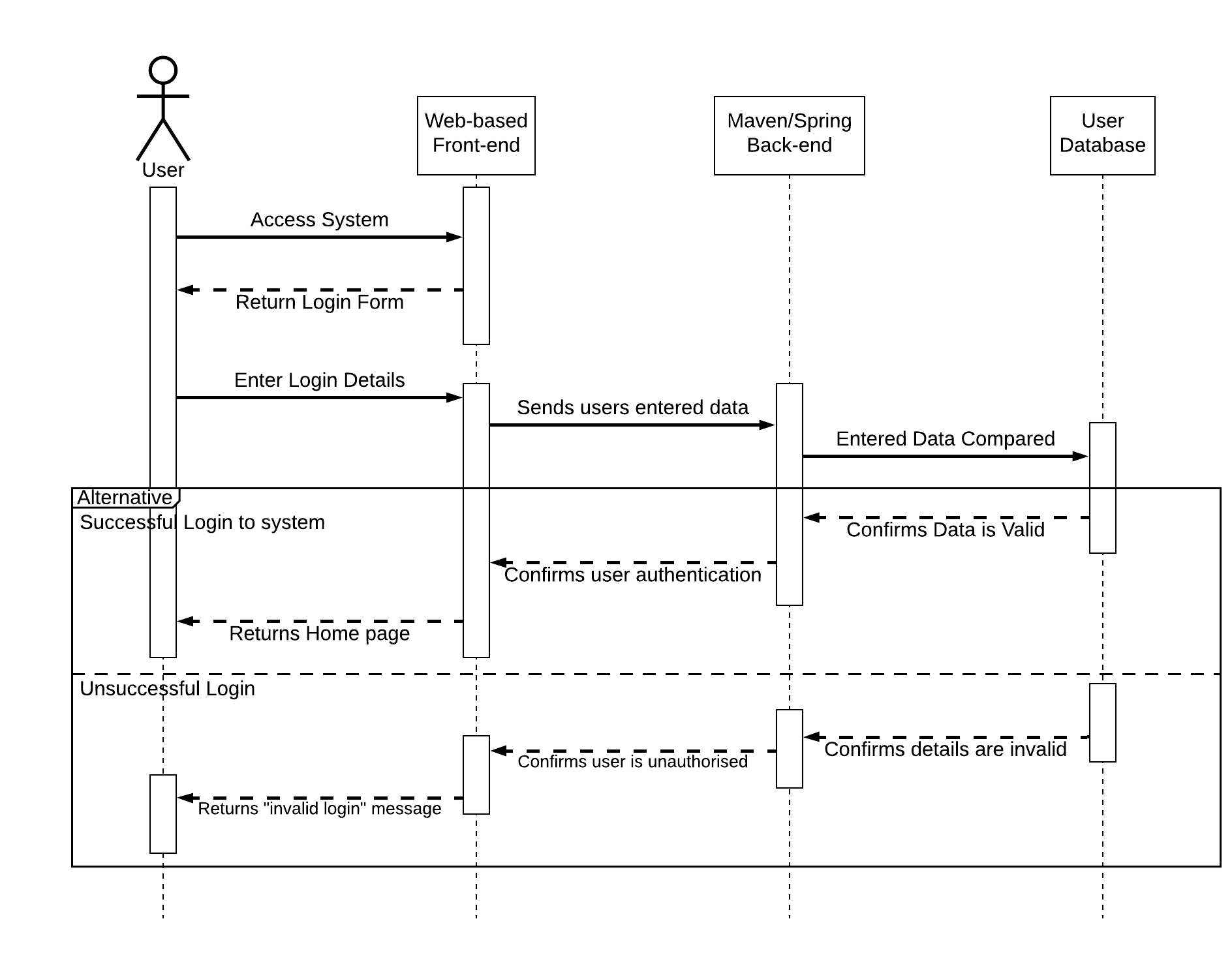


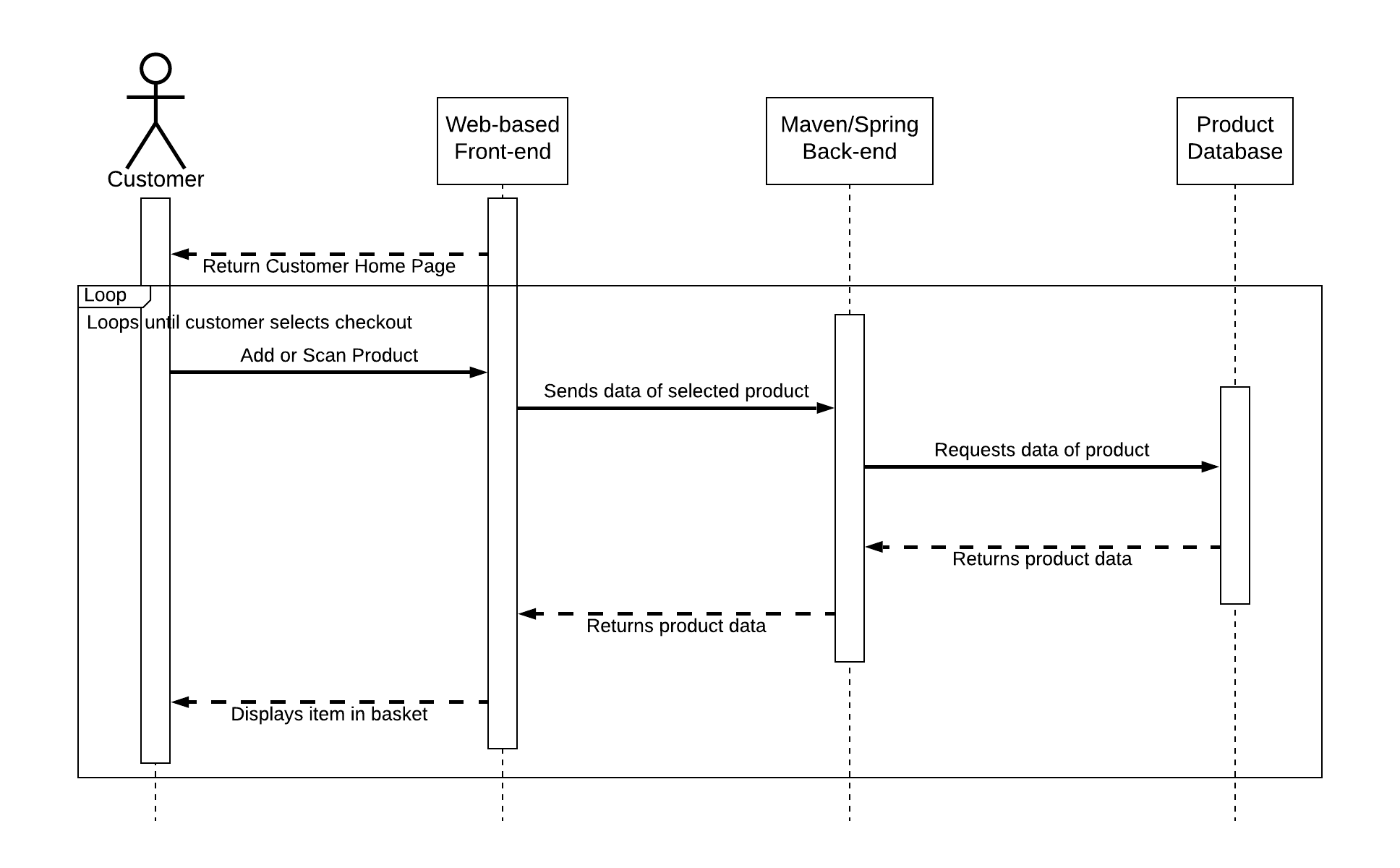
# 4 - Behaviour Diagrams

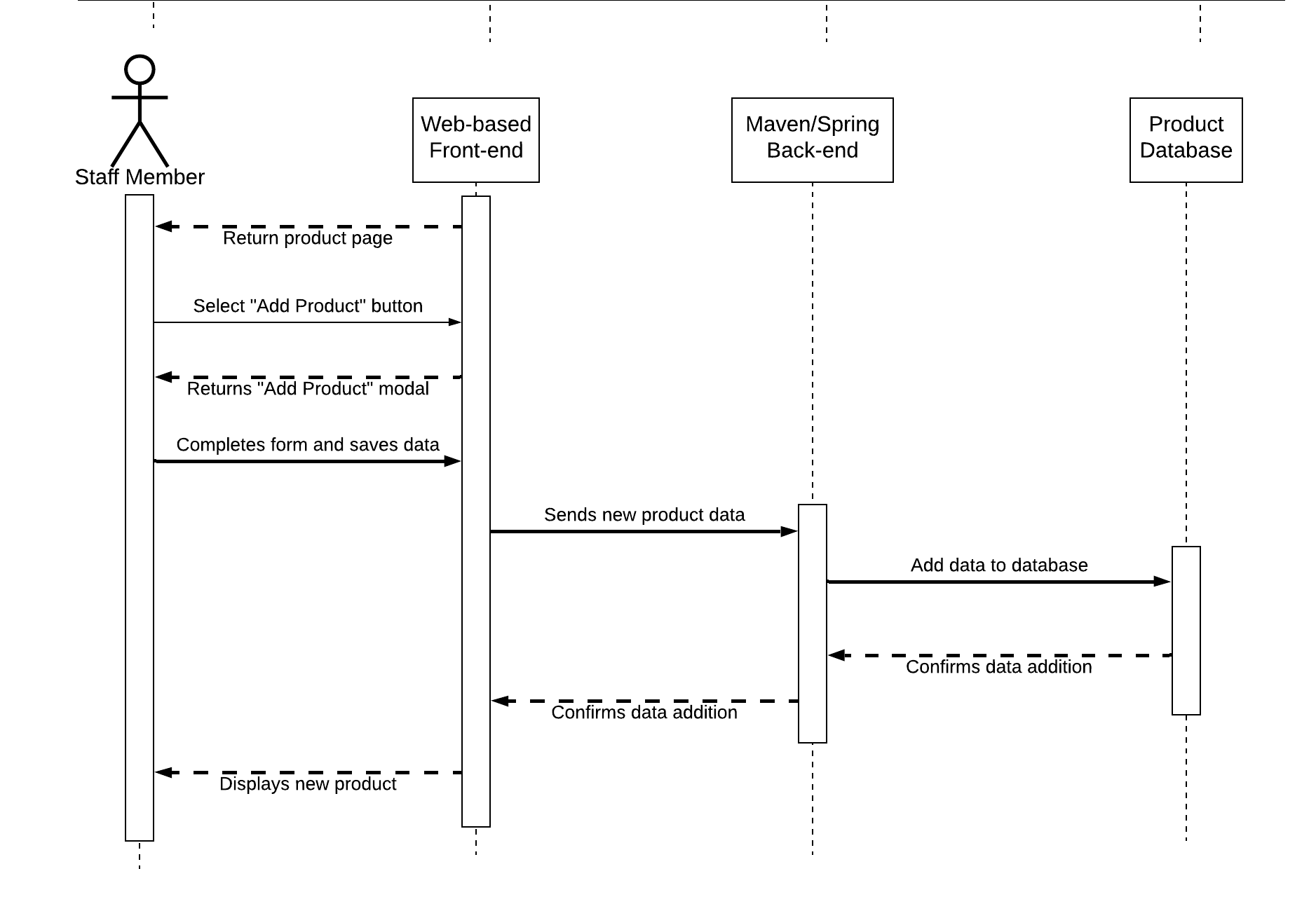
As with any system, different actions performed by the user can produce varying results due to the system’s built-in dynamic aspects affecting the collaborations between system objects, also known as its ‘behavioural aspects’ (Tutorialspoint, 2019). Due to the size of this system characteristic, the next section of this report will go into depth analysing the behaviour of the system through the usage of a variety of different diagrams.

## 4.1 - Sequence Diagram

In order to help demonstrate the flow of operations within a set amount of time inside a particular system, ‘Sequence Diagrams’ are often made use of to visually show this aspect of a system, specifically showing the collaboration of certain operations and how it helps to achieve a certain goal (Visual Paradigm, 2019). Below, a sequence diagram has been included showing (***Describe the diagram***), following as:

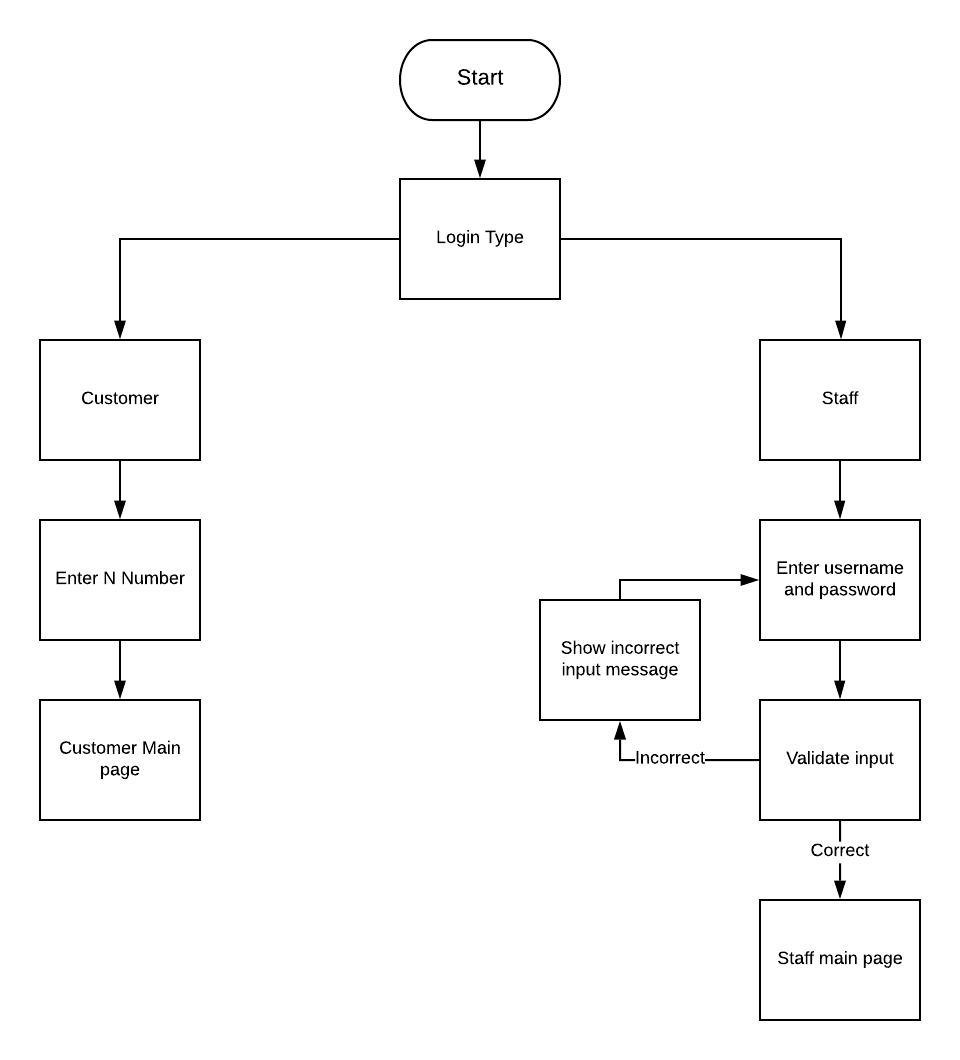


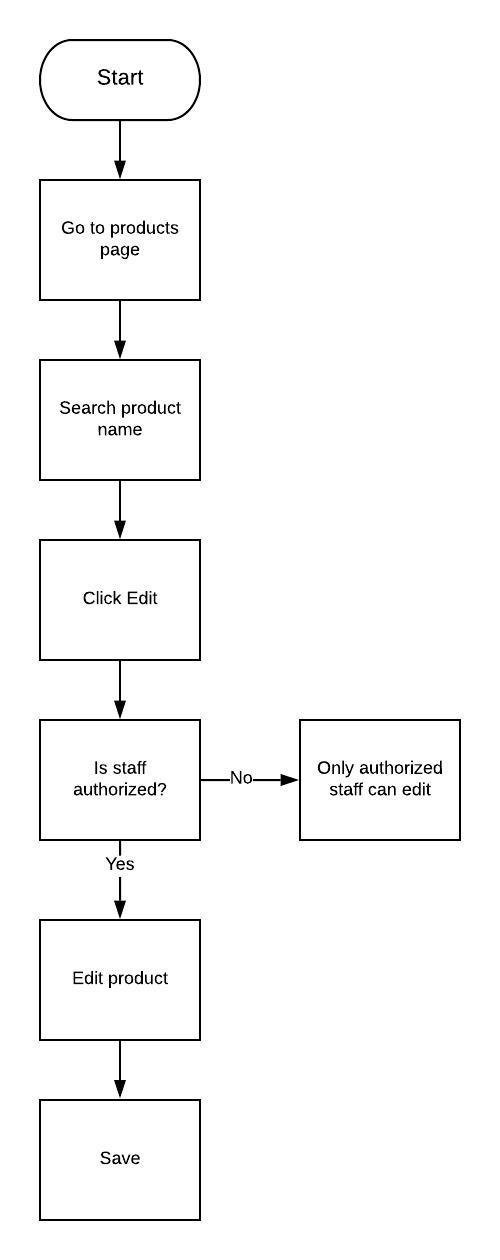




## 4.2 - Process Diagram

Often with a software system, it is unclear how to ‘travel’ through the system to perform a particular action or reach a specific goal, with some diagrams trying to rectify this but a lot of them being too high level and overly complicated for the average user to fully grasp or understand. To help negate this issue, a process diagram has been included below, known more commonly as a flowchart. This type of diagram is widely used as a way of visualizing how a user interacts with the system and how the system responds, showing the various actions a user may perform and what their ultimate results are (Tallyfy, 2020).



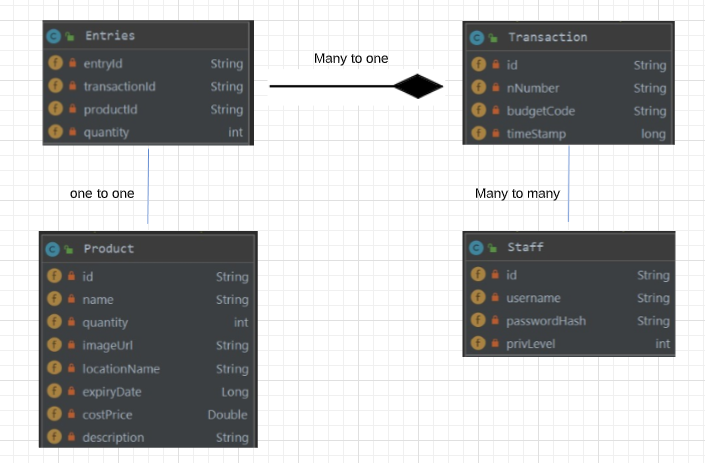


# 5 - Structure Diagrams

As with any system, each of the components built within are positioned and structured in a particular way, usually to help with data flow and to ensure they each meet their set goals fully (Techopedia, 2019). Due to the complexity of the inner structure of a system this size, the next section of this document includes multiple different structure diagrams that each possess the purpose of showing various different aspects of the designed systems structure along with an analysis as to why each diagram takes this particular form.

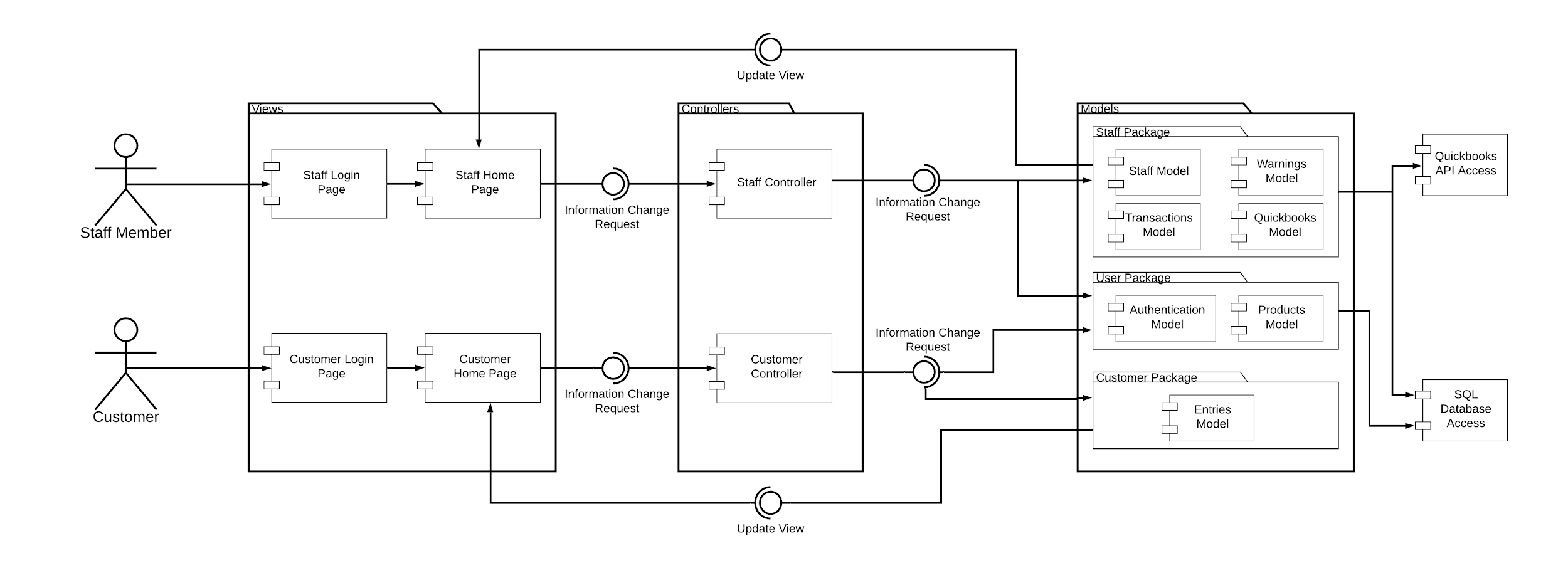
## 5.1 - Class Diagram

Another way of viewing the structure of a particular system is through the use of classes, each of which would have built-in attributes and operations, along with their interconnected relationships and what function of the system these relationships ultimately achieve (Lucidchart, 2019). To help show the structure of the system in this way, a ‘class diagram’ has been included below, following as:



## 5.2 - Component Diagram

Often with higher-level systems, the complex relationships of its physical components can confuse a reader due to them not being properly mapped in the above diagrams in a high enough level of detail. To help circumvent this, a component diagram has been drawn and included below, which has the specific purpose of functioning much like the above class diagram, but with much more of an emphasis on a system's components in a way that makes it easier to double-check that every aspect of the system's required functions is covered and implemented (Visual-paradigm, 2020).



# 6 - System Security

As with any system that contains information, it is vital that adequate security is implemented to help ensure the data is secured, with these requirements being necessary by law depending on the contained data and its overall sensitivity (Sciencedirect, 2020). Within this section of the report, the utilised security measures of the system will be analysed and ultimately evaluated to fully determine the reasoning for their usage compared to alternatives as well as what else could have been implemented to help enhance the project should this team possessed more time.

## 6.1 - Database Usage

One of the most common ways to store data for a software project is within a database, which is an electronically organised collection of various pieces of data (Oracle, 2020). For this project, it became clear from the initial stages of its planning that a database or databases would be utilised for the variety of data that this system would need to contain and make use of. Initially, the suggested database that this team would make use of for the project is MongoDB, which is an online document database that emphasises flexibility with a variety of different projects as well as a focus on scalability (MongoDB, 2020). Although the usage of this type of database for the project showed initial promise, it became clear that due to the teams overall inexperience with the software as well as the limited development time that we possessed for the project the decision was made to switch databases from MongoDB to an SQL database that would be integrated within the project itself. The decision to switch specifically to an SQL database was made due to each member of the team, especially those working on the backend, already possessed experience with its usage, having made use of these kinds of databases within previous projects. Additionally, this type of database is known for its ease of use and streamlined design, with the act of creating a database being easy and straightforward.

## 6.2 - API Usage

During the planning stage of this project, the team formulated and theorised on the usage of several different APIs that would work in unison to achieve multiple functional and nonfunctional requirements specified for the project. The integration of these APIs had varying success due to unfortunate unforeseen factors that led to multiple project design revisions that helped to ensure that the project was still delivered on its deadline.

Firstly, one of this team's early ideas was to contact the NTU security team to help gain permission to integrate the university's single sign-on system within the project, helping to secure the software against unauthorised access by only allowing properly authenticated students and staff access to the software. Whilst this idea showed initial promise, the slow response times of the security team to our emails combined with the approaching deadline led to the team abandoning the idea in favour of the usage of an SQL database, one that would contain the account details of each staff member of the storage department. Although this solution functions, it is limited, with no proper method of authenticating whether a customer’s entered university ID is valid, meaning that the system relies instead on customers entering a valid ID and not entering either one that is incorrect or someone else's.

Another API that this team planned to integrate within this designed system is the usage of the Quickbooks API, which is accounting software that was originally part of the old system used by the staff members of the storage department (intuit, 2020). Through the hard work of the back-end subteam, this API was full integrated within the system, functioning as a way for the system to automatically generate a new purchase order for a new order of products made by a staff member.

## 6.3 - Security Evaluation

Overall, it is clear from the above information that security for this project has been greatly considered from its initial planning stages, with steps being taken early on to try to gain access to the Universities single sign-on system as well as research being completed on the implementation of MongoDB within the backend of this project. However, it is also clear that unforeseen setbacks such as the lack of granted permission from the security team as well as the unforeseen difficulty of implementing MongoDB set the work of this team back significantly, and led to other aspects of the system being delayed. Despite this, the team worked hard to ensure that other alternatives to our initial ideas were implemented instead to help ensure that the system would still be secure. For example, in order for a customer to access the features of the system the user must enter a user ID, which is used as an identifier that attaches itself to any transactions that the user makes. This feature helps to ensure that the individuals that make a transaction are easily identifiable for the storages departments staff members but unfortunately has a major flaw. This flaw takes the form of a lack of ID validation, specifically meaning that, due to the lack of access to the universities single sign-on system, the ID entered by a user cannot be validated to ensure it is an official university ID, placing the majority of the reliance of the system on the user being honest.

On a similar note, in order for a staff member to log into the system they must provide both a valid staff ID and the password associated with it, otherwise they will not be able to interact any further with the system. This extension of the login page helps to ensure that no customers can access the system and manipulate any of the systems stored data regarding products or transactions. Alongside this feature, the staff profiles stored in the system’s database have been built implementing an ‘authorisation system’. Specifically, this means that certain staff profiles possess higher-level access to the more advanced features of the system, with an example being that an administrator will have the ability to order any kinds of stock for the stores main stockroom but other access levels such as ‘manager’ will only have this ability to a lesser extent, being able to only order certain stock and needed the approval of an administrator for others.

# 7 - Major Project Changes

As with any project, the move to the design stage may help highlight changes to the original proposed design to help ensure the system is designed both meeting its specified design requirements and fully finished before its set deadline (Brighthubpm, 2020). During the development stage of this project, it became clear that multiple ideas that we originally had or suggested would have to either be scrapped or modified in some way to ensure the creation of the project ran smoothly.

The first of these changes was to the systems proposed user interface, with the original idea being that the system would be developed using a framework called “Angular”, which would communicate with the back-end of the project via HTTP requests and would follow the design specified in the requirements specification completed for this system during its planning stage. However, once the development had started it became clear to the subteam working on that particular aspect that the original designs had room for improvement, with several pages being deemed not necessary and could instead be streamlined to take the form of popup boxes that would appear once the user has pressed a button and would either request information or present it. This design change ultimately helps the system give off a much sleeker and refined feel, with the time saved by implementing it allowing the team to focus on more critical aspects of the project.

Another major change to the structure of this system came in the form of needed to implement an alternative method of allowing users to log into the system, which was due to this team unfortunately not being granted access to the universities single sign-on system. The alternative that was eventually implemented was the usage of an SQL database, one that would store the details of the storage departments staff members. This compromise, as has been mentioned earlier in this report, came at the cost of system functionality, with the database providing no way to authenticate whether or not a customer’s entered university ID was valid, limiting the systems overall provided security.

# 

# 8 – User Help Documentation

## **Client Deployment Instructions**

### Pre-Requisites

* NPM

### Launch the Client

1. Open directory AADClient/eksi cd AADClient/eksi
2. Install Angular npm install -g @angular/cli
3. Serve Angular ng serve
4. Open http://localhost:4200

### Alternatively: To build as a static site

ng build --prod

### Set the Server IP:

1. Click the Account Info Button in the top right corner of the page
2. Change the IP, click save, and then click logout
3. Then login to the new server

## **Server Deployment Instructions**

### Pre-Requisites

* MySQL server
* Java jdk 1.8
* Maven

### Database

1. Setup MySQL Server
2. Run dbSetup.sql on the server
3. git clone https://github.com/DanielRHolland/AADGroupProject.git or download zip file.
4. Open folder AADGroupProject
5. Open file /src/main/resources, set MySQL database username and password
6. Run Maven Install mvn install
7. Open target folder cd target
8. Run jar file java - jar

# 

# 9 - Testing and Results

Although a system may be built by a team of experts it is always vital that adequate and comprehensive testings have been performed at regular intervals during the development of a system. This testing can come in multiple forms, such as extensive user testing with random members of the systems target audience, running a series of test suites based on the systems original requirements and stress testing any and all of the features of the system involving user input (EDUCBA, 2019). All of this testing is completed to help ensure that the system meets all of the requirements that were gathered within the projects analysis stage, as well as helps to ensure that the system meets basic aspects of software checklist such as:

* Functional.
* Performs well.
* Scalable.
* Documented.
* Secure.

Within the next section of the report, an analysis will be completed on this project's testing stage, specifically focusing on the various kinds of tests that were held on the designed system, why they were held, what results were gathered and ultimately how these results can be used to improve both the system itself and any future projects this team may find themselves working on.

## 9.1 – Functional Tests

Perhaps the most well-known form of testing that can be completed on a software system is known as ‘functional testing’, within which each of the systems built-in functions are verified one by one to ensure that they conform to the functional and non-functional requirements determined within this projects analysis stage (Guru99, 2019). To aid a team with their analysis of the final product of their designed system a common occurrence is for a comprehensive table of system functionality to be created. Specifically, this table will take the original functional and nonfunctional requirements created by a team in their initial requirements gathering stage, as well as the full details of requests made by the client during the meetings that were held, and ultimately compare them with the capabilities of the final version of their created product.

|  |  |  |
| --- | --- | --- |
| *Test* | *Procedure* | *Outcome* |
| User Login/Logout | Start up the application.    -------------------------------------    Input Login Details.    -------------------------------------    Confirm details on database if they match.    -------------------------------------    Display appropriate display when criteria is matched with their authority on the website.    -------------------------------------  Click Logout to come back to Login/Front Page of the application.    -------------------------------------  User is no longer logged in and needs to input details again to gain access to further views of the application. | Choose between customer or staff login.  Customer inputs N number.      Staff inputs username and password.        Staff details are authenticated.      Customer goes to the customer main page.  Staff goes to staff main page.        Staff logs out and goes to staff login page.  Customer logs out and goes to customer login.      User must input details again to access application |
| Access to the correct View | Each user type has access to different areas of the application, and this should be noted when logging in. This information should be stored on the database. | Correct display for both types of users. |
| Connecting to Database and Network on start up | When starting up the application, network and database connection should be established.    -------------------------------------  Active/Live database of items when browsing/editing items | Connection successful.        Database is active and responsive. |
| Accessible account information to the user | Retrieve data of the user on the user page from database where they can edit their information (some may require admin permission) | Authorized staff can view details of user and what server they are on. |
| Accessing Items | Being able to click on a product to see its information.  (All Users)  -------------------------------------  Being able to access the editor of items to amend information and save it to the database. (Depending on the authority level on the application). | Users are able to check product information.      Authorized staff are able to edit items and save changes made. |
| Creating New Item | A pop up box should appear to input new data of the new item.  -------------------------------------  Saved inputted information gets saved on the database and refreshes the list on the visible application | Authorized staff are able to input details for new item.    Saved information is saved and updated on application. |
| Customer Log | Accessing view of recorded user use.  -------------------------------------  Specific users have enabled access to this. | You can access user usage through transaction logs.    Only authorized staff can make changes. |
| Navigation on the Application | Smoothly accessing different areas of the application through NavBar and Buttons on the application.  -------------------------------------  Making sure user is not kicked off the network when navigating to another page on the application or when adjusting objects on the database. | Navigating is smooth, fast and easy.      Navigation through pages is seamless and smooth. |
| Allowing Admin of the application to manage accounts | Admin can add or delete accounts on request.    -------------------------------------  Requires update to database so that the user is validated to log on either as Staff or Customer | Authorized staff can make changes to staff accounts.      Changes are saved to database and updated. |
| Product quantity management | Staff/User can view number of available stock that can be purchased.  -------------------------------------  Staff can update number of stock manually.  -------------------------------------  Receiving notification when products are low on quantity. | Users are able to view item current stock number.    Authorized staff can manually change stock number.  This functionality was misclassified as a core functionality. |
| Creating a purchase order with budget codes of a customer which allows them to make a transaction | Manager Level user or higher can create a purchase order accessing budget code from the system to match with the customer account if eligible to buy a chosen product. | Authorized staff can make any changes on transaction, or make new transactions if necessary. |
| Processing Transactions and Returns | Customer matching all necessary criteria to buy a wanted product from stores.  -------------------------------------  Database updated automatically/manually by stores when an item is sold.  -------------------------------------  Database allows access for staff to update product number is stock when return is processed. | Customer enters budget code.      Database is updated and saved when items are sold.    Only authorized staff can update product number. |
| Transaction Log | Being able to access the Transaction Page as a staff member.  -------------------------------------  User is able to add a transaction log with all required information like budget code, number of products sold, price of order, who accessed/purchased the item from stores and if item is returned or sold.  -------------------------------------  User can edit any transaction log in case of human error during handling of an order or when item is returned.  -------------------------------------  Being able to get a spreadsheet created for the transaction log to be able to later process it to the finance office. | Only authorized staff can make changes on transaction page.      Authorized staff can make a new transaction with all necessary data.        Authorized staff can make any changes on transaction logs.      A spreadsheet is made and can be downloaded for each transaction log. |

## 9.2 - Performance Requirements

In order to determine the way in which a system behaves outside of its normal functionality, extensive performance testing has been performed. Specifically, this type of testing involves defining how well the system performs certain functions under specific conditions, with examples of the type of system functions being tested including the systems overall speed of response, execution time and storage capacity for any kind of entered data (1202Performance, 2020).

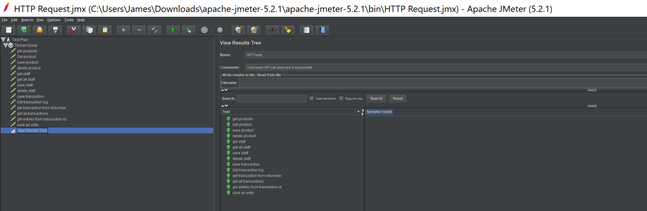
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Type** | **No.** | **Requirement** | **Description** | **Result** |
| Maintainability Requirements | Should | NFR1 | High-Quality Code | Designed to agreed upon code and design standards, so that it is easy to maintain, update, and modify. | Test passed, code has been created and separated in different classes and layers to ensure easy to maintain. |
| Performance Requirements | Should | NFR2 | Sufficient Capacity | Be built so that it is able to contain hundreds of records for a variety of different items. | Test Passed, able to store hundreds of products. |
| NFR3 | Responsive Handling | Responds quickly to inputs. | Test passed, is very responsive and Jmeter connect times very low. |
| Could | NFR4 | Responsive Data Retrieval | Be able to access stored data quickly. | Test passed, Jmeter connect times very low. |
| Reliability Requirements | Should | NFR5 | Robust | The system should be robust and resistant to crashing | Test passed, through all frontened testing system did not crash. |
| NFR6 | Low Downtime | Be built so that, if any updates are being installed, disruption to the work of the users is minimised. | Test failed, when maintaining users can not use system. |
| NFR7 | Restorable | Be built so that, in the event of a major power outage/crash, it will be minimal effort to restore the system to full functionality. | Test Passed, if out application is easy to start back up. |
| Scalability Requirements | Could | NFR8 | Extensible | Designed to agreed modular framework model and design standards, so that in the future it will be easy to add new features and elements to the application. | Test passed, design pattern followed. |
| Security Requirements | Must | NFR9 | Data Protection | All data stored on both customers and members of staff are stored correctly and safely abiding by all rules stated in the GDPR Act 2018. | Test not carried out, data stored currently not checked against GDPR. |
| Usability Requirements | Must | NFR10 | Works on existing hardware | Be designed in a way that the system can be deployed to the existing machine being used in the storeroom; they should not need any additional hardware to use this system. | Test passed, runs on jvm. |
| NFR11 | Ease of Use | Be built in a way so that it is intuitive for intended users to operate the system (Store worker, manager, administrator and customer). | Test passed, application built is simple and easy to use, with main navigation through large icons. |
| NFR12 | Cross-Platform | Be accessible from either mobile, desktop or laptop devices. | Test passed, accessible through all devices. |
| Could | NFR13 | Redeploy able | Be designed so that the entire system can be redeployed and migrated to another virtual/physical machine in the future. | Test passed, runs on jvm. |

## 9.3 - API Tests (Back-end Testing)

As has been previously stated within this report, various API’s were either incorporated or planned to be incorporated within this project to enhance its overall functionality. To ensure its overall workability, it is important for developers to carry out ‘API tests’, an act that generally consists of making requests to either a single or multiple endpoints of each API with the goal being to validate its response, specifically validating factors such as the software overall performance or security (Smartbear.com, 2020).

We decided to use Apache Jmeter to carry out our API testing. This tool was decided on as it gave us the ability to create a test plan where every request made to the server could be checked each time the program was executed.

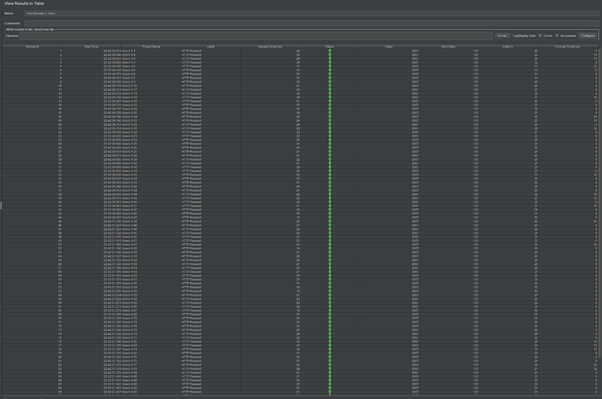
By doing this, whenever any member of the team made a change to the program, they could run this test plan and check all requests still worked as expected.



As you are able to see above, API requests that we have created are now successful and this has been a useful tool throughout the development process.

In addition to this, we also carried out load testing. This was carried out to ensure that our final product was more then capable of handling the demands it would be put through at any one time.

To do this, Apache Jmeter was used again to simulate 100 concurrent users all making a request at the same time. Doing this enabled us to test whether the product will be able to handle the max amount of people that could possibly be using the system at one time. Below are the results of the load testing.

.

From the above, it is clear that the application is more then capable to handle as many users as needed. This is because for all 100 users requesting data, the average connect time was just 9.2ms, clearly illustrating how performance is not an issue.

## 9.4 - Front-End Testing

During the analysis stage of this project, it became clear from the beginning that the system would need to be designed with the technical capabilities of the user completely in mind, specifically the fact that it was most likely that whoever was intended to use the system would possess a minimal amount of technical experience. Therefore, the system would need to be built in a way that allowed these types of users to still be capable of utilising it to fulfil the tasks of their jobs without becoming too confused or overwhelmed. To help ensure that the designed system meets these expectations, significant front-end testing has been performed, which is a specific type of testing that focuses on checking the GUI of the software, which refers to anything that is visible on-screen and capable of being interacted with by a user (Guru99, 2019). The results of this testing have been included below:

**(Krystian has completed this but Google docs isn’t letting me paste the tests in so I will need to do it after i’ve downloaded this report after everything else has been completed and added.)**

## 9.5 - Results

With the testing of each system aspect completed above it is now essential for this team to analyse their results in order to see what has gone well, what has held us back and gone awry and how this will ultimately influence our decisions in future projects.

# 10 – Conclusion

To conclude, is is clear from the report above that a significant level of thought and effort has done into the development stage of this project, with various level of analysis being performed on the designed system including a look at how it was designed, any changes that were made that differed from the original plan and ultimately what these changes resulted in test wise. By completing this project, the individual members of this team have learned a great deal not just in the areas of computing but also in how to work together with strangers to work towards a common goal, all of which will be valuable knowledge in the future.

***This should be no more than 20 A4 pages including diagrams.***

# 11 – References

Medium. (2019). *Software Architecture - The Difference Between Architecture and Design*. [online] Available at: https://codeburst.io/software-architecture-the-difference-between-architecture-and-design-7936abdd5830 [Accessed 30 Nov. 2019].

Medium. (2020). *Model-View-Controller (MVC)*. [online] Available at: https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66 [Accessed 26 Jan. 2020].

Bobology.com. (2019). *What is the Background Process?*. [online] Available at: https://www.bobology.com/public/What-is-a-Background-Process.cfm [Accessed 30 Nov. 2019].

Visual-paradigm.com. (2019). *What is Deployment Diagram?*. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ [Accessed 30 Nov. 2019].

Tutorialspoint.com. (2019). *OOAD - UML Behavioural Diagrams - Tutorialspoint*. [online] Available at: https://www.tutorialspoint.com/object\_oriented\_analysis\_design/ooad\_uml\_behavioural\_diagrams.htm [Accessed 30 Nov. 2019].

Visual-paradigm.com. (2019). *What is a State Machine Diagram?*. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/ [Accessed 30 Nov. 2019].

Visual-paradigm.com. (2019). *What is a Sequence Diagram?*. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ [Accessed 30 Nov. 2019].

SearchSoftwareQuality. (2019). *What is a Collaboration Diagram?*. [online] Available at: https://searchsoftwarequality.techtarget.com/definition/collaboration-diagram [Accessed 30 Nov. 2019].

Techopedia.com. (2019). *What is an Activity Diagram? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/27489/activity-diagram [Accessed 30 Nov. 2019].

Visual-paradigm.com. (2019). *What is Data Flow Diagram?*. [online] Available at: https://www.visual-paradigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/ [Accessed 30 Nov. 2019].

Techopedia.com. (2019). *What is the Structure Diagram? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/22358/structure-diagram [Accessed 30 Nov. 2019].

Fakhroutdinov, K. (2019). *UML component diagram shows components, provided and required interfaces, ports, and relationships between them.*. [online] Uml-diagrams.org. Available at: https://www.uml-diagrams.org/component-diagrams.html [Accessed 1 Dec. 2019].

Lucidchart.com. (2019). [online] Available at: https://www.lucidchart.com/pages/uml-class-diagram [Accessed 1 Dec. 2019].

Moresteam.com. (2019). *Process Flow Chart Tutorial*. [online] Available at: https://www.moresteam.com/toolbox/process-flow-chart.cfm [Accessed 1 Dec. 2019].

Smartdraw.com. (2019). *Entity Relationship Diagram (ERD) - What is an ER Diagram?*. [online] Available at: https://www.smartdraw.com/entity-relationship-diagram/ [Accessed 1 Dec. 2019].

EDUCBA. (2019). *System Testing | Different Types And Key Focus Area of System Testing*. [online] Available at: https://www.educba.com/system-testing/ [Accessed 1 Dec. 2019].

Guru99.com. (2019). *What is Functional Testing? Types & Examples (Complete Tutorial)*. [online] Available at: https://www.guru99.com/functional-testing.html [Accessed 1 Dec. 2019].

Guru99.com. (2019). *What is Front End Testing? Tools & Frameworks*. [online] Available at: https://www.guru99.com/frontend-testing.html [Accessed 2 Dec. 2019].

Atwood, J. (2019). Understanding Model-View-Controller. [online] Blog.codinghorror.com. Available at: https://blog.codinghorror.com/understanding-model-view-controller/ [Accessed 17 Dec. 2019].

Interserver Tips. (2019). What is MVC? Advantages and Disadvantages of MVC - Interserver Tips. [online] Available at: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/ [Accessed 17 Dec. 2019].

Developer.intuit.com. (2020). *Intuit Developer*. [online] Available at: https://developer.intuit.com/app/developer/qbo/docs/develop [Accessed 17 Jan. 2020].

Sciencedirect.com. (2020). *Software Security - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/computer-science/software-security [Accessed 17 Jan. 2020].

Oracle.com. (2020). *What is a database?*. [online] Available at: https://www.oracle.com/uk/database/what-is-database.html [Accessed 17 Jan. 2020].

MongoDB. (2020). *What Is MongoDB?*. [online] Available at: https://www.mongodb.com/what-is-mongodb [Accessed 17 Jan. 2020].

Docs.spring.io. (2020). *Spring Framework Overview*. [online] Available at: https://docs.spring.io/spring/docs/current/spring-framework-reference/overview.html [Accessed 17 Jan. 2020].

Software Testing Fundamentals. (2020). *Integration Testing - Software Testing Fundamentals*. [online] Available at: http://sReferences

Smartbear.com. (2020). What Is API Testing? | API Testing Basics | SmartBear Solutions. [online] Available at: https://smartbear.com/solutions/api-testing/ [Accessed 24 Jan. 2020].oftwaretestingfundamentals.com/integration-testing/ [Accessed 17 Jan. 2020].

Brighthubpm.com. (2020). [online] Available at: https://www.brighthubpm.com/change-management/5737-change-management-in-software-development-projects/ [Accessed 2 Feb. 2020].

Edrawsoft.com. (2020). Architecture Diagram Overview. [online] Available at: https://www.edrawsoft.com/architecture-diagram.html [Accessed 2 Feb. 2020].

Visual-paradigm.com. (2020). What is a Component Diagram?. [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/ [Accessed 2 Feb. 2020].

Sourcemaking.com. (2020). Design Patterns and Refactoring. [online] Available at: https://sourcemaking.com/design\_patterns [Accessed 3 Feb. 2020].

Tallyfy. (2020). *What is a Process Flowchart and How to use it [5+ Examples] - Tallyfy*. [online] Available at: https://tallyfy.com/process-flowchart/ [Accessed 8 Feb. 2020].

1202Performance. (2020). How to write Performance Requirements with Example. [online] Available at: http://www.1202performance.com/atricles/how-to-write-performance-requirements-with-example/ [Accessed 14 Feb. 2020].

# 12 - Bibliography