-

Team members

Steven Kennedy 40025827

Jonathan Leathem 40059090

Conor McAleavey 40047330

Chris McClune 40138365

Rebecca McKevitt 40026952

Kevin O’Hare 15481042

Victoria Stacey 40030110

**CSC 7051**

**Report**

Contents

**1.** **Peer marking** 4

**2.** **Describe and evaluate the development model used** 6

**3.** **Design** 6

**3.1. Database Schema** 6

**3.2. UML Diagram** 6

**3.3. User Interface Design** 6

**4.** **Implementation** 7

**4.1.User Interface (UI)** 7

**4.2. Database** 7

**4.3. Queue System** 7

**5.** **Testing** 8

**6.** **Evaluation of Project and Product Produced** 8

**7.** **Appendix** 9

**7.1 Database Schema** 9

**7.2 UML NHSystem** 9

**7.3 UML NHSystem** 10

# **Peer marking**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student Number** | **Student Email** | **Student Name** | **Section(s) Tested** | **Peer mark (whole numbers only)** |
| 40025827 | skennedy2001@qub.ac.uk | Steven Kennedy |  |  |
| 40059090 | jleathem03@qub.ac.uk | Jonathan Leathem |  |  |
| 40047330 | cmcaleavey02@qub.ac.uk | Conor McAleavey |  |  |
| 40138365 | cmcclune02@qub.ac.uk | Chris McClune |  |  |
| 40026952 | rmckevitt03@qub.ac.uk | Rebecca McKevitt |  |  |
| 15481042 | kohare08@qub.ac.uk | Kevin O’Hare |  |  |
| 40030110 | vstacey01@qub.ac.uk | Victoria Stacey |  |  |

# **Describe and evaluate the development model used**

The development of this product did not rigidly adhere to a single development model, rather, combined the benefits of a number of popular software development methodologies, including, but not limited to, Agile Development, eXtreem Programming (XP) and The Waterfall Model.

Due to the limited resources at our disposal for this project we initially decided to closely follow the Waterfall Model. At the outset, this provided our project with structure and a definite end date, which was key to our team completing the project before the submission deadline. Despite the obvious benefits of this model, we decided that using this development methodology in isolation was limiting to our project. This was because as the software progressed, the access to the client became more valuable – allowing for more interesting queries and an end-product that was hopefully even closer to the client’s expectations.

Although we did adopt a general Agile development practice for our project – towards the end of the software’s implementation we did lean towards the XP methodology (Beck 1996). This benefited us greatly because of the constantly changing approach to aspects of our system, were we found ourselves creating regular increments of the software, reviewing and improving the code as we went along.

# **Design**

# **3.1. Database Schema**

To reduce redundancy and minimise chances of loss of data, we decided to produce a schema to model our database from. Also some of the tables chosen in the schema are designed are for the propose of expanding the system. The reason for this is to create a system that is more flexible and that will be able to cope with future modifications. The entities and relationships in the database schema are shown in Appendix 7.1.

# **3.2. UML Diagram**

Please see Appendix 7.2 and 7.3.

# **3.3. User Interface Design**

A user interface was created using Java FX and Scene Builder 2.0. The user interface has various pages which we have designed to be consistent in colors and theme. The layout of the pages is quite simple as we intended the system to be intuitive and uncomplicated for users.

The user interface consists of the following pages:

**Login** - To be accessed by doctors, nurses and receptionist. A login screen which the user will use to enter the credentials which are their staff number and password.

**Queue System** - To be accessed by the triage nurse. This screen has a drop down box which allows the user to modify the patient’s position in the queue by assigning them a new triage priority. There is another drop down box which allows the user to select which treatment room to assign to the patient.

**Reception Layout** - To be accessed by the receptionist. The screen allows the user to search for the details of patients who have entered the A&E department. There are four fields where patient’s details can be entered. These are first name, last name, postcode and NHS number. There is a table of search results of patients in the database whose details match those inputted. The patient’s information that is displayed is title, first name, last name, street, city, postcode and phone number. The user can select the patient from the list and add them to triage.

**Treatment Room** - To be accessed by doctors and nurses working in the treatment rooms. The first name, last name blood type and known allergies of the current patient being treated are displayed on the screen. The time the treatment began is also shown on the screen. The user has the option to extend the treatment time and there is a text box for the user to enter a summary of the treatment which they can save in the system.

**Triage** - To be accessed by the triage nurse. This screen shows a table including the patients NHS number, first name, surname, allergies and condition. There is a drop down box which is used to assign a triage category to the patient which can then be saved into the system.

# **Implementation**

# **4.1.User Interface (UI)**

# Designed using JavaFX Scene Builder 2.0 – our UI plays a key role in the overall delivery of our system to the client. Therefore, it was important we created a friendly and easy to use UI. Scene Builder afforded us the opportunity to create such a design with and array of built-in features such as labels, buttons, and frames among other visually impressive utilities.

# We created our UI by firstly adding a new FXML file from within Eclipse – setting up the basic elements of the screen, ensuring to reference the corresponding Java Controller class in the project for each FXML file. We then opened this file in Scene Builder to add features and to make more precise layout edits to the file.

# **4.2. Database**

A SQL database was implemented for a number of reasons, the most obvious being for the storage of patientdetails in order to be searched for and entered into the queue. Client side validation and user input format corrections were incorporated before running the connection method in case of erroneous input in the “Login” and “Patient Search” sections.

Within the “patient” table of the database we created 2 patients called “Unknown Female” and “Unknown Male”, this allowed for patients with no known details to still be entered into the PAS. This would allow for further expansion of the system if details regarding at the time “Unknown” patients are later found out and can then be inputted to the patient database table.

# **4.3. Queue System**

Contained within QueueController.java class, our queue provides some of the most functionally important aspects of our system. The queue is displayed using the Eclipse console instead of the UI. Given that displaying the queue to UI was not a requirement of the client we decided to prioritise our workload appropriately which meant putting this function aside for future development.

Our queue is mainly supported in the console using System.out.print function of Eclipse to display the information as refreshed from the runnable thread every four seconds using an infinite loop. All of the code needed to display and sort the data, check if patients are in treatment rooms etc.; are kept within this main thread. Patient, TreatmentRoom, TreatmentRoom thread and Patient thread are all classes used by this queue controller class. Instances of new Patients are created and passed into the queue controller class from variables passed across using the TriageController.java. Linked lists as part of the Java Collections Framework are used to store patients in the queue and the Treatment Rooms as part of their own unique linked list.

Patients within the queue are sorted using ‘Collections.sort,’ that is called for every refresh of the queue. Each Patient object and Treatment Room object have assigned their own threads called on creation of the unique object, were each created new thread is run in the background using a runnable thread in the queue controller class, to allow the main thread to run without interruption (i.e. a thread within a thread). Boolean variables are used to check whether Patients are in rooms, and the use of Treatment Room count threads to check if they were to be removed from the treatment room.

# **Testing**

The majority of the testing in the project was undertaken as the code was first being written. Black box testing was used to check that the system functioned as expected. As the code became more complex we began to use Integration testing to test how separate methods and classes functioned together.

Close the end of the project when the majority of the code had been completed JUnit tests were used to validate the smaller components of the system to ensure they could handle user input and outputs correctly. Individual classes were tested separately to verify that they work in expected, boundary, and negative cases. White box testing was used to demonstrate how the code was working and helped to locate exactly where errors in the code were occurring.

Ideally as a team we would have liked to have JUint tested the entire project, but due to a technical road block with testing the fxml UI we were unable to. However, we continuously checked the functionality of the project for each new update to the code, using ‘ad hoc’ tests as necessary and reused test cases to ensure previous functioning code was unaffected by updates.

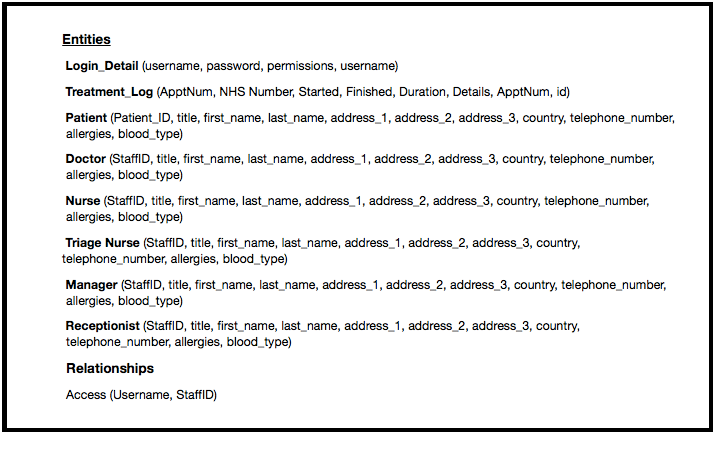
# **Evaluation of Project and Product Produced**

We believe that the quality of system we have produced to be reasonably high. It fulfils all of the requirements outlined in the problem description. We have also managed to include additional features that go above and beyond the requirements of the client; such as a database to store staff and patient information as well as a graphical user interface. We have agreed our product to be fit for use on a local scale but not national implementation. The reason being is that our system has been developed and tested based on sample database, which would not compare to the scale of an NHS database. That said our system did cope well and is available to be extended and expanded further.

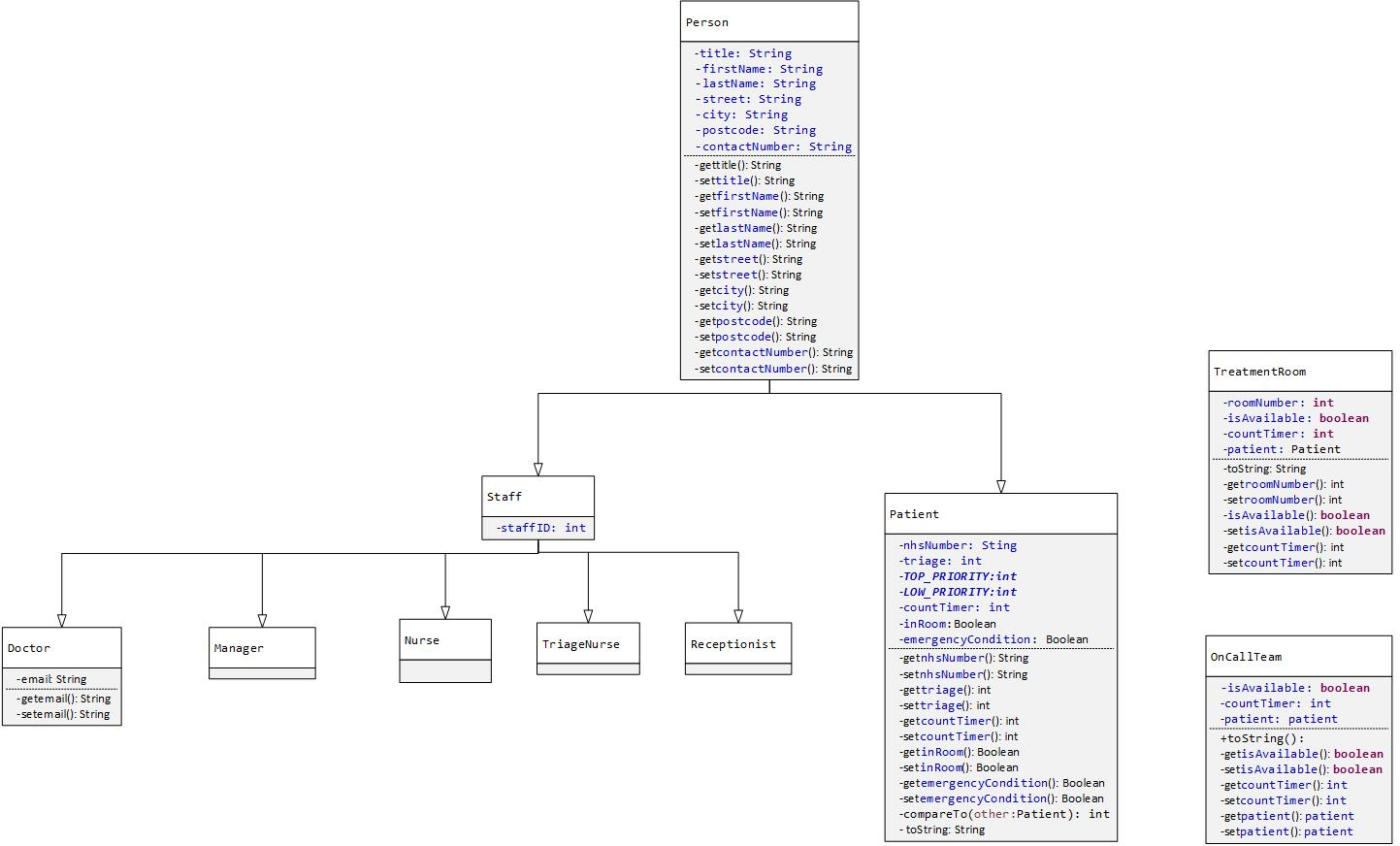
Reflecting on the project as a whole we agreed that we would prefer to take a more test driven approach whereby we would test as we develop based on the assumption that little testing can eliminate a few flaws, then a lot of testing can eliminate many more flaws.

# **Appendix**

# **Database Schema**



# **UML NHSystem**



# **UML Queue**