KING'S College LONDON

Software Engineering Major Group Project

Project 1: Blood Test Diary

Client:

Dr. Marianne Samyn, King's College Hospital

Team Lyrebird:

Bonian Hu - k1764139

Zhenjie (Jeff) Jiang - k1764072

Yilei (Oscar) Liang - k1764097

Tao Lin - k1763808

Swapnil Paul - k1763878

Yeshvanth Prabakar - k1763461

Patrick Whyte - k1763560

Introduction:

Patients of the pediatric liver service at King's College Hospital needs to undergo regular blood tests to allow the doctors to monitor their condition. Our client regularly needs to track when specific patient's tests are due, remind patients of their upcoming blood tests, identify patients overdue for tests and request test results from laboratories. Currently our client uses a system that is time-consuming, prone to errors and does not scale well with the number of patients. Their current system also makes use of different applications and spreadsheets to accomplish this task. This request for a more clean an integrated solution. Our team came up with a new system with a simple and flat user interface that also automates much of their current processes which usually take up most of our client's time. A web application was developed using Spring Boot (MVC architectural layout) for the backend and MaterializeCSS on the front end. Our solution combines all their different platforms into one web application which is a much more cleaner and efficient solution to meet their daily needs.

Outcome:

Web application usable from any platform including phone and tablets by using a flexible design.

The application allows for scheduling and tracking of patients due for their blood test appointments presented in a tabular form:

- CRUD (create, read, update and delete) function on patient information
- Patients can be filtered based on severity (Critical, Urgent, Monitor, OPA).
- Patients can be searched by name.
- Automatic email notification sent to patients to notify them on upcoming appointments.
- Manual reminder can be set by clinician for patient appointments.

A test can be attributed to each patient where:

- CRUD function on test information
- Every missed test from the patient is recorded and if he/she miss more than 2 test the severity of the patient became Critical
- Blood test appointments are rescheduled automatically (1 week later) if the appointment is missed but can also be scheduled manually for any date by the user.

Blood test results can be sent to the appropriate GP automatically or manually for each of the patients on the patient page by clicking the 'email icon'.

- Easy upload of test results (can attach additional PDF or any other format prefered) and send results to patient in a auto-mail sending format that is prewritten.

The application includes a hospital page that maps the location of each blood test of a patient. This allows for the clinicians to quickly browse and contact (phone number or even address for posting/finding locality):

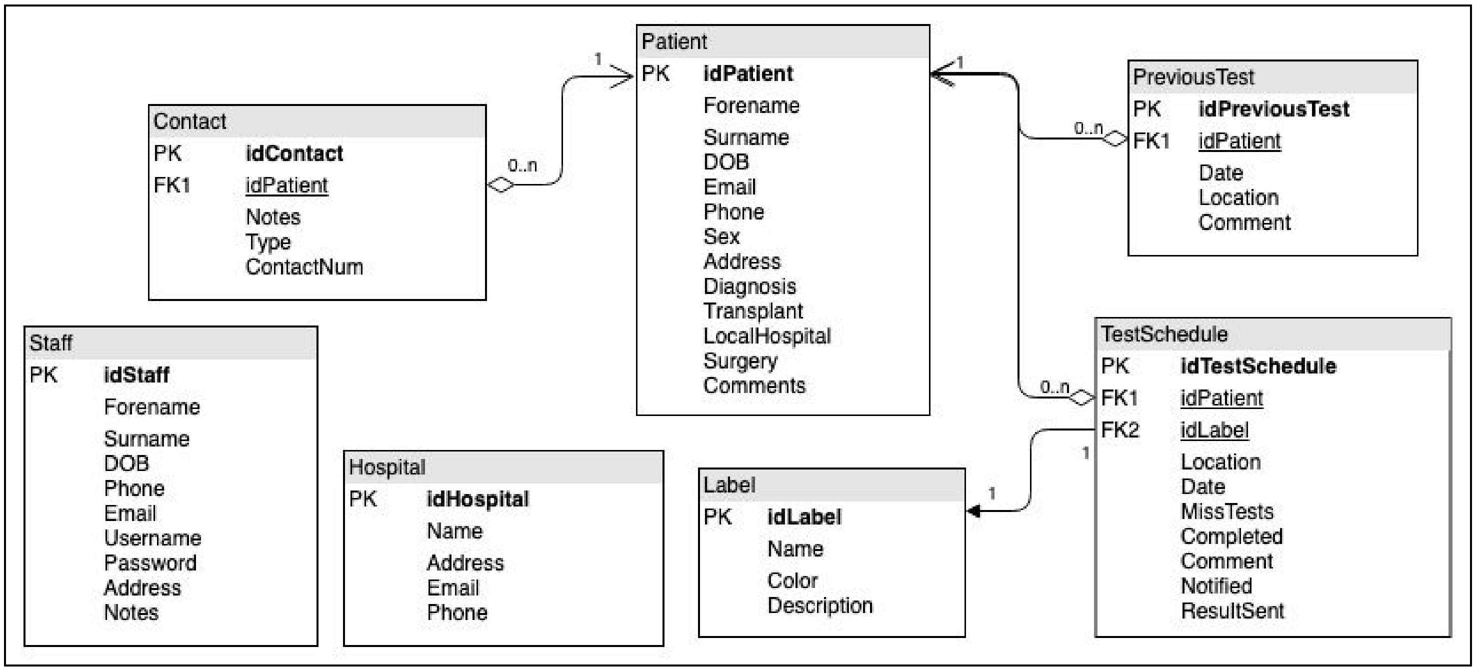
- CRUD function on hospital information
- A list of hospitals that correspond to each patient.
- Showing the hospital in Google Maps API which integrated to the web app, or retrieve manually by searching the address.
- Search hospitals by name.

The application is secure:

- Passwords are encrypted with BCryptPasswordEncoder.
- Each user can change their password, only if they are register by authorised staff.
- Each login session is been secured, non authorized session can't access the main page and perform any type.
- SQL inputs have been sanitized preventing SQL injections corrupting the database and block attempts to retrieve sensible information.
- Session allow associating information with individual visitors

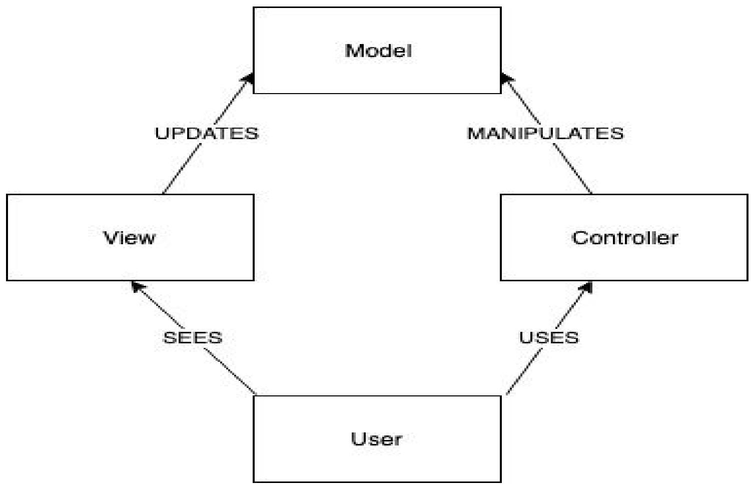
Design:

Figure 1 - Database UML Diagram:



The database for our application was developed using MySQL. The database allows our application to store information about patients, staff and hospitals which the user can read and update through the use of tables and forms built into the user interface. The database also allows our application to keep track of scheduled blood tests for each patient. This allowed us to automate many of the processes that our client used to do manually using their current system such as; rescheduling of missed blood tests, increasing the level of severity after missing multiple blood tests, notifying patients of up-coming blood test, etc. Using the framework Spring Boot helped mitigate the chance of any malicious SQL Injection attacks from causing damage to the database and secure the information contained within it.

Figure 2 - MVC Class Diagram:



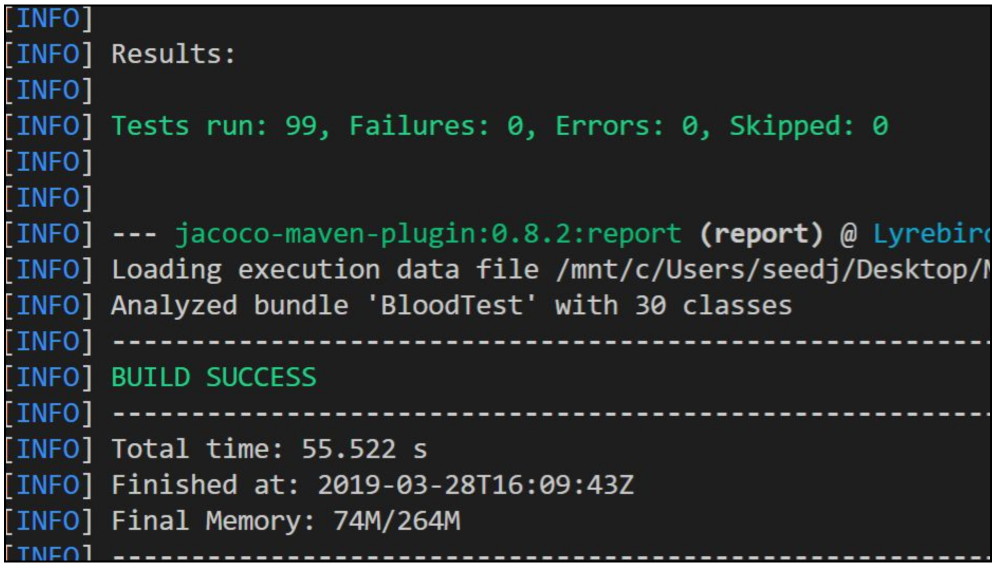
Using an MVC architectural pattern ensured that our project was well organized and modularized into three main components: the model (the data), the view (a way to modify the data) and the controller (operations that can be performed on the data); as seen in the aforementioned image. Model, VIew and Controller classes were created for each of the different elements such as Hospital, Patients etc. The flow of control is then passed amongst each of the respective elements such as:

Hospital Controller -> Hospital Model -> Hospital View.

The MVC architecture, a common architecture used on well designed software engineering projects, was achieved by using the Spring Boot Framework which is a java based platform and can be used with any Java application providing portability, wide range of libraries, security, great performance and robustness. Choosing Spring Boot Framework provide an efficient way to maintain the project in the future. Some other advantages are embed Tomcat (no need to deploy WAR files), provide POMs to simplify Maven configuration and automatic configuration.

Testing:

Testing was accomplished through the means of unit testing for each of the POJO classes to test the functionality of the back end classes for the pull and post requests (spring boot forms). Testing is automated with the help of the build automation tool maven and code coverage metrics were used to ensure testing was carried out to an appropriate degree. Testing for web layer was done by using mockmvc, however the web layer tests does require the mysql server to pass. Besides, many of the tests for services require the app to be booted up in order to pass. Junit tests can be found under “bloodTest/src/Test”. Additonally, due to the nature of the application (http requests, db functions, etc), a lot of the tests are done manually, where unit tests are not necessarily useful.



```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 99, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.2:report (report) @ Lyrebir
[INFO] Loading execution data file /mnt/c/Users/seedj/Desktop/
[INFO] Analyzed bundle 'BloodTest' with 30 classes
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 55.522 s
[INFO] Finished at: 2019-03-28T16:09:43Z
[INFO] Final Memory: 74M/264M
[INFO] ------------------------------------------------------------
```

Team Organisation:

1. Team coordination by Swapnil Paul (Back-end/Front-end)
    a. Team coordination: Back-end branch by Zhenjie (Jeff) Jiang
        i. Tao Lin
    b. Team coordination: Front-end branch by Yeshvanth Prabakar
        i. Bonian Hu
        ii. Yilei (OSCAR) Liang
        iii. Patrick Whyte

Further Subteams:	**Task:**
Yeshvanth	Front end UI design & Layout, Logo.
Yeshvanth, Bonian, Patrick	Front end development. Creation of all pages, their HTML components. Creation of main navbar, Mobile NavBar dropdown, footers as well as isolating them to separate file for reusability.
Yeshvanth, Swapnil	Changing patient severity levels by filtering via checkboxes.
Swapnil	MySQL database schema design and constraints. Insertion of data via SQL entries.
Yeshvanth, Patrick	JUnit testing for POJO classes.
Tao, Bonian	Creating email template, automatic sending of emails for patients and their reminders.
Patrick, Yeshvanth	Setting team deadlines, ensuring goals tasks were met on time and overall project planning. Played integral role as interface to liaise between team and client i.e requirements engineering and setting up meetings.
Oscar, Tao, Jeff, Swapnil	Linking the front end to the back end: pulling the data for the patient and hospital table, pre-displayed data for editing forms and writing to the database for entry for all ‘form’s.
Jeff	Logging in system, encryption of passwords and overall system security.
Jeff, Tao, Swapnil,	Mail System service back end
Oscar	Google MAP API
Jeff, Swapnil, Oscar	Junit Testing for Web layer, Controller and Services
Jeff	Functionality for managing missing_tests
Tao	Functionality for reschedule tests

Other Pertinent Information:

Main Client Requirements:

- Create, Read, Update and Delete (CRUD) patient information.
- Create, Read, Update and Delete (CRUD) hospital information.
- Send automated and manual reminders to patients.
- Send completed test result to the patient.
- Represent the current spreadsheets information in a efficient way in the new system.
- Represent the test carried over and current test in efficient view without wasting time.
- Chasing for patient and hospital information.
- Track overdue blood tests and send a notification.
- Filter the patient based on their conditions severity.
- Automatically change patients severity level if they missed several scheduled tests.

Major changes in requirements:

- The client had mentioned how they wanted to run their project internally within the hospital network (intranet), but had little suggestions for security due to their lack of technical knowledge.
    - We came up security features such as password encryption and user session management.

- The client also mentioned that some patients are not able to receive emails therefore they wanted to send the reminders as SMS text messages instead of emails. However, we were unable to find any open-source API for SMS messaging (as they are paid services). And since the NHS cannot provide any funds for remind the patient by SMS, the cheapest way and also the most efficient way was to send all the notification by email.
    - If the client require a text message notifications to notify their clients, thus an API can be purchased and the application can be extended upon.
    - An additional requirement of using the Google Maps API was added by us, in coordination with the client.

(Unexpected) risks and how they were tackled:

- Since the team was divided into two teams (front end and back end) integrating the two proved to be a challenge.
    - Members had to develop knowledge of the opposite teams work and pair programming, were appropriate, was carried out to ensure that the two team members could work effectively and the front end/back-end was connected properly.

- There were quite a few technical issues that arose during the project.

- Initially, there was a learning curve to the spring boot framework as only two teammates were well versed with it. This also applied to using maven as a build automation tool despite familiarity with gradle as there were issues in automating the tests each time when used in VS-Code and specifying the maven build.
- The whole project was run on VS-code using a tightly managed directory of files. However despite the whole project being rather autonomous and easy to manage there were still issues with the database as the SQL scripts had to be reloaded into the database (server) each time modifications were made to the bloodtestdiary.sql file.
- There were issues in the front end when MaterializeCSS was used. Classes and IDs were used to specify buttons, tables and div elements accordingly to apply the materializeCSS effects such as *<a id="menu" class="waves-effect waves-light btn btn-floating" ><i class="material-icons">menu</i></a>* run. This interfered with the naming convention making it difficult to refer to these elements throughout the remainder of the code as they had to be referred using this exact aforementioned class name. Overriding them for a more 'customized' layout was a further problem. This by far was the biggest bottleneck during the development of the front end.
- To reuse existing code (such as the footer and header) we tried using javascript to call and insert 'div elements' but this did not work due to some compatibility issues with spring boot. Instead the ThymeLeafStandardLayout system had to be used throughout the project to for example to insert elements (th:insert, th:replace functions) to wedge page components into the final view. This was an unexpected issue that emerged during the early phases of development.
- There was also a problem on setting-up the sender email, by choosing a domain the mail was not verified. At first all the email sent to the sender's address were automatically inboxed as a spam. We resolved this problem by creating a validated Microsoft account where the sender's email is set up in application.properties
- At first the database was not compatible on the application because of the timezone. This was resolved by adding the following:

```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/bloodtestdiary?useUnico
de=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&se
rverTimezone=UTC
```

- Another unexpected risk that materialised during the project was the scheduling and tests for the patients. There was too much patient information to be fitted into one page.

- This was overcome by dividing the information in two, the patient search (Homepage) and a tests page for each patient, where there is quick navigation from each page to another to add, modify or delete tests and patients.
- During the latter half of the project the tests section was further divided into current tests and previous 'missed' tests as the team had felt itd better fit with the concept of a blood test schedule.

- Requirements engineering proved to be a challenge during the task. Extracting the requirements to be included in our project was a challenging task despite the client giving us a 'dumbed down' version of sample data.

- Sitting down with the group and thinking with a 'practical mindset' of how to best implement this was our approach to successfully tackling this problem and contacting the client where when doubts where present on whether to include a 'requirement' or not proved to be crucial.

- Furthermore, we found that there were persistent changes to the requirements too during the development of the project. An example would be when the option to 'label' a patient based on their priority on the homepage.

- Initially, a patient was flagged as 'serious' or 'safe' but after conversing with the client they had requested that there be four labels instead: 'critical', 'urgent', 'monitor' and OPA.
- Other such situations arose during the project with the notification of emails, and sending test results. The sending of results issue was resolved by providing an attach PDF option before an email is set on the application to allow clinicians to upload and send patient results.

Changes to these projects were made to the clients liking and preference where available. Overall, working in advance and setting strict deadlines was the solution to these challenges.

Appendix:

Appendix A - How to Setup the Project

- Setup Mysql server, load bloodtestdiary.sql from "bloodTest/src/main/rescources", in the application.properties, change the mysql information according to the server setup.
- Install maven, go into the bloodTest directory, run "mvn clean package"
- Using command line, inside the "bloodTest/target", run "java -jar Lyrebird-0.0.1-SNAPSHOT.java" command line
- On any browser(prefer chrome), go to http://localhost:8080, default username is "user" with password "password"

To change the main setting of the application such as:

- MySql database
- Sender email address

Can be found on /bloodTest/src/main/resources/application.properties