CA400
Test Documentation

Workload Measurement System for Nurses

Project Supervisor: Dr. David Sinclair

Ruth Leavey 17323886
Harley Martin 17401932

Date completed: 07/05/2021

1. Introduction

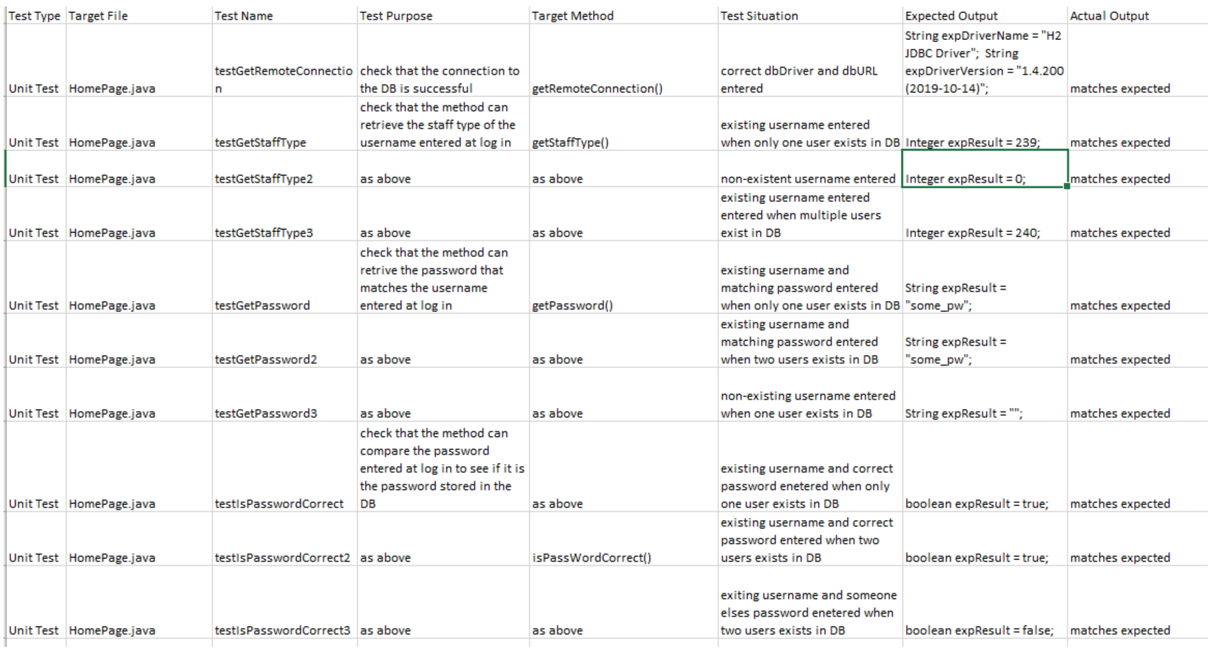
For this final year project we will be undertaking the following testing:

- Unit testing
- GUI testing
- Integration testing
- User testing

2. Unit Testing

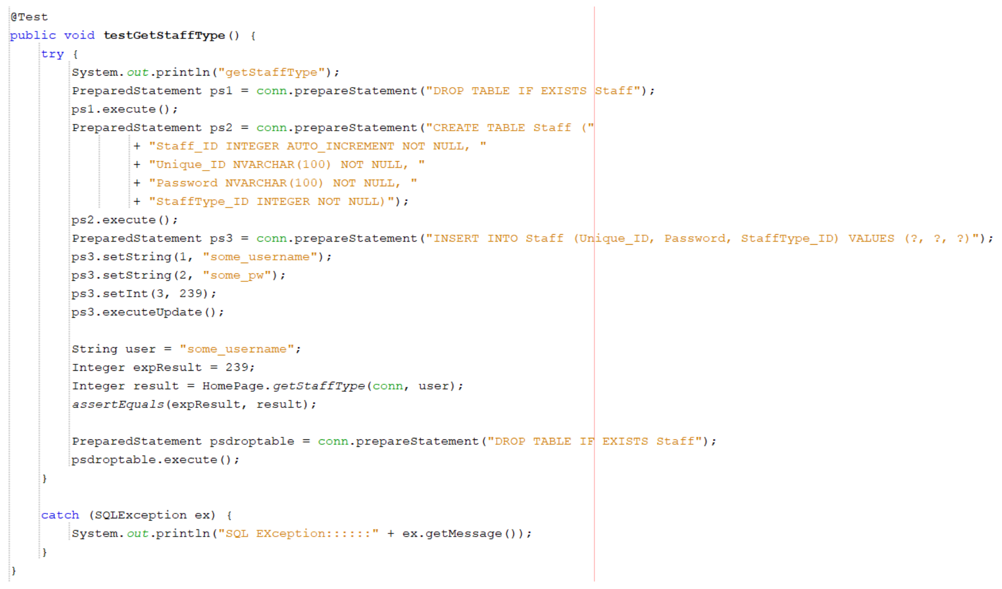
Unit testing was implemented using JUnit throughout this project.

Test plans for these unit tests were created during the development of the system and depending on the functionality. An example includes the following:

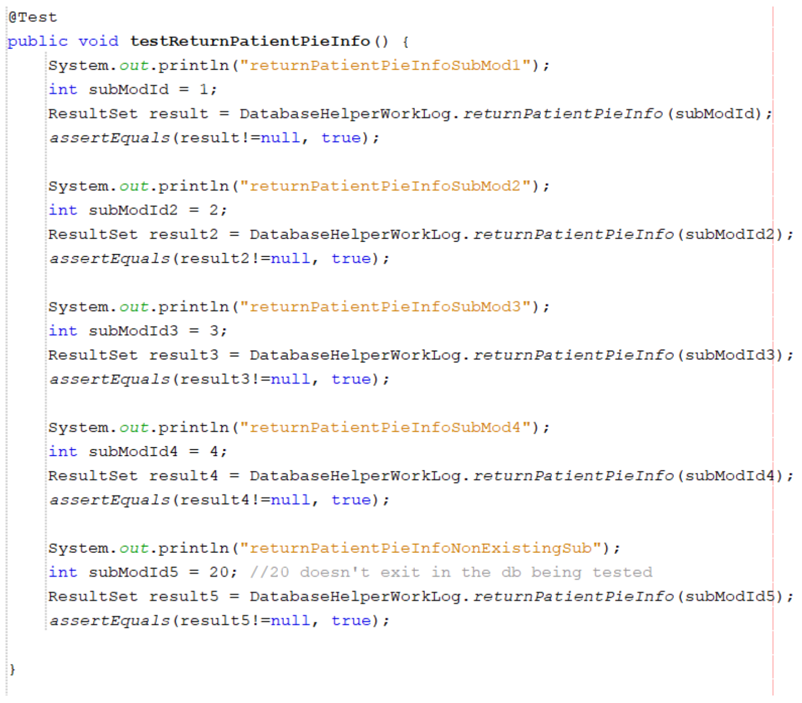


Test Type	Target File	Test Name	Test Purpose	Target Method	Test Situation	Expected Output	Actual Output
Unit Test	HomePage.java	testGetRemoteConnection	check that the connection to the DB is successful	getRemoteConnection()	correct dbDriver and dbURL entered	String expDriverName = "H2 JDBC Driver"; String expDriverVersion = "1.4.200 (2019-10-14)";	matches expected
Unit Test	HomePage.java	testGetStaffType	check that the method can retrieve the staff type of the username entered at log in	getStaffType()	existing username entered when only one user exists in DB	Integer expResult = 239;	matches expected
Unit Test	HomePage.java	testGetStaffType2	as above	as above	non-existent username entered	Integer expResult = 0;	matches expected
Unit Test	HomePage.java	testGetStaffType3	as above	as above	existing username entered entered when multiple users exist in DB	Integer expResult = 240;	matches expected
Unit Test	HomePage.java	testGetPassword	check that the method can retrive the password that matches the username entered at log in	getPassword()	existing username and matching password entered when only one user exists in DB	String expResult = "some_pw";	matches expected
Unit Test	HomePage.java	testGetPassword2	as above	as above	existing username and matching password entered when two users exists in DB	String expResult = "some_pw";	matches expected
Unit Test	HomePage.java	testGetPassword3	as above	as above	non-existing username entered when one user exists in DB	String expResult = "";	matches expected
Unit Test	HomePage.java	testIsPasswordCorrect	check that the method can compare the password entered at log in to see if it is the password stored in the DB	as above	existing username and correct password enetered when only one user exists in DB	boolean expResult = true;	matches expected
Unit Test	HomePage.java	testIsPasswordCorrect2	as above	isPassWordCorrect()	existing username and correct password entered when two users exists in DB	boolean expResult = true;	matches expected
Unit Test	HomePage.java	testIsPasswordCorrect3	as above	as above	exiting username and someone elses password enetered when two users exists in DB	boolean expResult = false;	matches expected

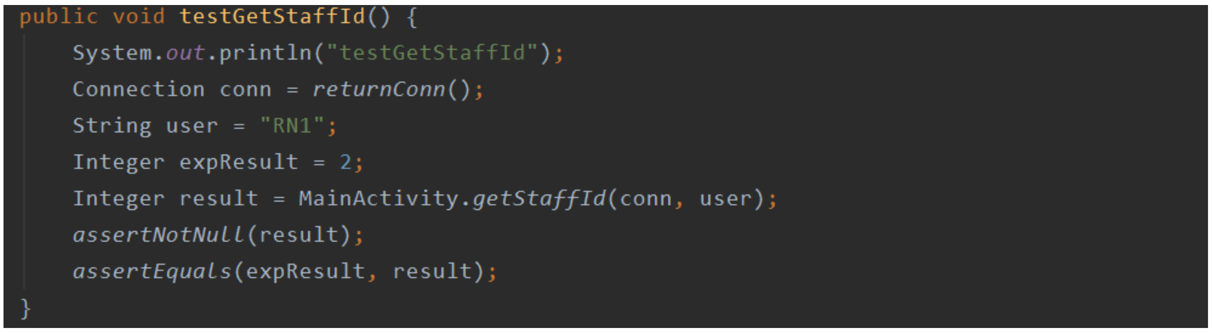
Below are some further code snippet examples of unit testing applied to our project:



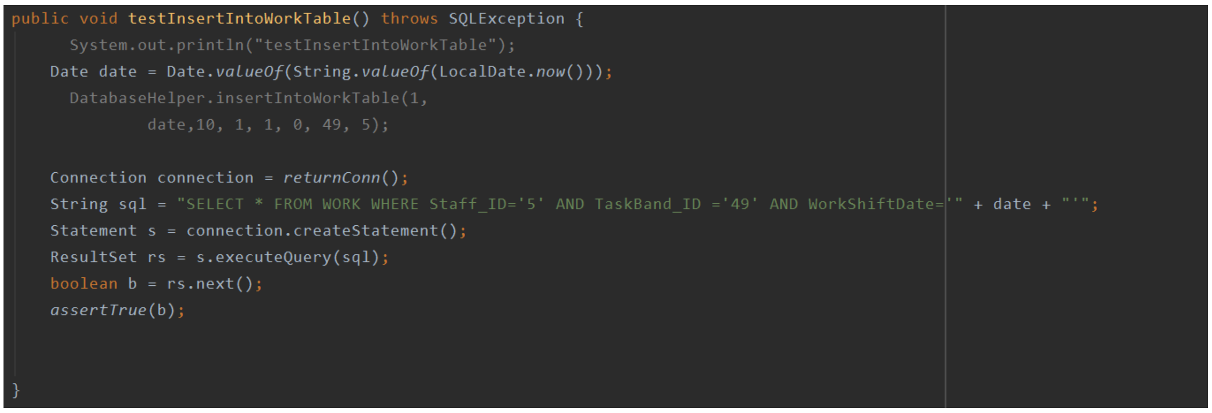
```java
@Test
public void testGetStaffType() {
    try {
        System.out.println("getStaffType");
        PreparedStatement ps1 = conn.prepareStatement("DROP TABLE IF EXISTS Staff");
        ps1.execute();
        PreparedStatement ps2 = conn.prepareStatement("CREATE TABLE Staff ("
                + "Staff_ID INTEGER AUTO_INCREMENT NOT NULL, "
                + "Unique_ID NVARCHAR(100) NOT NULL, "
                + "Password NVARCHAR(100) NOT NULL, "
                + "StaffType_ID INTEGER NOT NULL)");
        ps2.execute();
        PreparedStatement ps3 = conn.prepareStatement("INSERT INTO Staff (Unique_ID, Password, StaffType_ID) VALUES (?, ?, ?)");
        ps3.setString(1, "some_username");
        ps3.setString(2, "some_pw");
        ps3.setInt(3, 239);
        ps3.executeUpdate();

        String user = "some_username";
        Integer expResult = 239;
        Integer result = HomePage.getStaffType(conn, user);
        assertEquals(expResult, result);

        PreparedStatement psdroptable = conn.prepareStatement("DROP TABLE IF EXISTS Staff");
        psdroptable.execute();
    }

    catch (SQLException ex) {
        System.out.println("SQL EXception:::::" + ex.getMessage());
    }
}
```



```java
@Test
public void testReturnPatientPieInfo() {
    System.out.println("returnPatientPieInfoSubMod1");
    int subModId = 1;
    ResultSet result = DatabaseHelperWorkLog.returnPatientPieInfo(subModId);
    assertEquals(result!=null, true);

    System.out.println("returnPatientPieInfoSubMod2");
    int subModId2 = 2;
    ResultSet result2 = DatabaseHelperWorkLog.returnPatientPieInfo(subModId2);
    assertEquals(result2!=null, true);

    System.out.println("returnPatientPieInfoSubMod3");
    int subModId3 = 3;
    ResultSet result3 = DatabaseHelperWorkLog.returnPatientPieInfo(subModId3);
    assertEquals(result3!=null, true);

    System.out.println("returnPatientPieInfoSubMod4");
    int subModId4 = 4;
    ResultSet result4 = DatabaseHelperWorkLog.returnPatientPieInfo(subModId4);
    assertEquals(result4!=null, true);

    System.out.println("returnPatientPieInfoNonExistingSub");
    int subModId5 = 20; //20 doesn't exit in the db being tested
    ResultSet result5 = DatabaseHelperWorkLog.returnPatientPieInfo(subModId5);
    assertEquals(result5!=null, true);

}
```



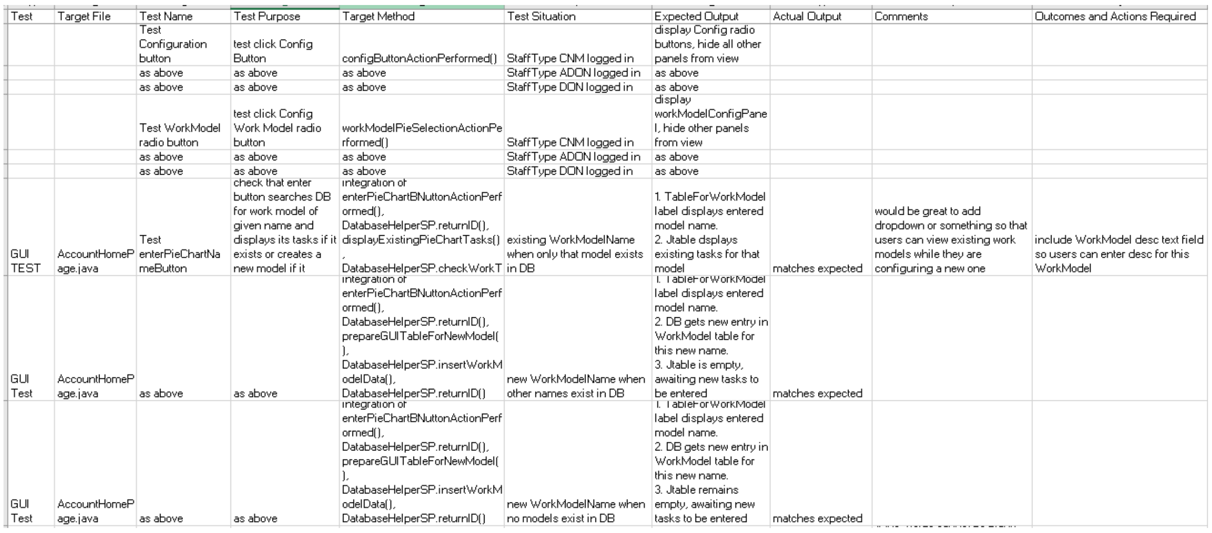
```java
public void testGetStaffId() {
    System.out.println("testGetStaffId");
    Connection conn = returnConn();
    String user = "RN1";
    Integer expResult = 2;
    Integer result = MainActivity.getStaffId(conn, user);
    assertNotNull(result);
    assertEquals(expResult, result);
}
```



```
public void testInsertIntoWorkTable() throws SQLException {
      System.out.println("testInsertIntoWorkTable");
    Date date = Date.valueOf(String.valueOf(LocalDate.now()));
      DatabaseHelper.insertIntoWorkTable(1,
              date,10, 1, 1, 0, 49, 5);

    Connection connection = returnConn();
    String sql = "SELECT * FROM WORK WHERE Staff_ID='5' AND TaskBand_ID ='49' AND WorkShiftDate='" + date + "'";
    Statement s = connection.createStatement();
    ResultSet rs = s.executeQuery(sql);
    boolean b = rs.next();
    assertTrue(b);


}
```

3. GUI Testing

Because a large amount of our java application involved the user interacting with the interface, GUI testing was carried out through trying out different GUI interactions and displaying the results. Examples of this can be seen in the following spreadsheet:



Test	Target File	Test Name	Test Purpose	Target Method	Test Situation	Expected Output	Actual Output	Comments	Outcomes and Actions Required
		Test Configuration button	test click Config Button	configButtonActionPerformed()	StaffType CNM logged in	display Config radio buttons, hide all other panels from view			
		as above	as above	as above	StaffType ADON logged in	as above			
		as above	as above	as above	StaffType DON logged in	as above			
		Test WorkModel radio button	test click Config Work Model radio button	workModelPieSelectionActionPerformed()	StaffType CNM logged in	display workModelConfigPanel, hide other panels from view			
		as above	as above	as above	StaffType ADON logged in	as above			
		as above	as above	as above	StaffType DON logged in	as above			
GUI TEST	AccountHomePage.java	Test enterPieChartNameButton	check that enter button searches DB for work model of given name and displays its tasks if it exists or creates a new model if it	integration of enterPieChartBNuttonActionPerformed(), DatabaseHelperSP.returnID(), displayExistingPieChartTasks() , DatabaseHelperSP.checkWorkT	existing WorkModelName when only that model exists in DB	1. TableForWorkModel label displays entered model name. 2. Jtable dsplays existing tasks for that model	matches expected	would be great to add dropdown or something so that users can view existing work models while they are configuring a new one	include WorkModel desc text field so users can enter desc for this WorkModel
GUI Test	AccountHomePage.java	as above	as above	integration of enterPieChartBNuttonActionPerformed(), DatabaseHelperSP.returnID(), prepareGUITableForNewModel( ), DatabaseHelperSP.insertWorkModelData(), DatabaseHelperSP.returnID()	new WorkModelName when other names exist in DB	1. TableForWorkModel label displays entered model name. 2. DB gets new entry in WorkModel table for this new name. 3. Jtable is empty, awaiting new tasks to be entered	matches expected		
GUI Test	AccountHomePage.java	as above	as above	integration of enterPieChartBNuttonActionPerformed(), DatabaseHelperSP.returnID(), prepareGUITableForNewModel( ), DatabaseHelperSP.insertWorkModelData(), DatabaseHelperSP.returnID()	new WorkModelName when no models exist in DB	1. TableForWorkModel label displays entered model name. 2. DB gets new entry in WorkModel table for this new name. 3. Jtable remains empty, awaiting new tasks to be entered	matches expected		

4. Integration Testing

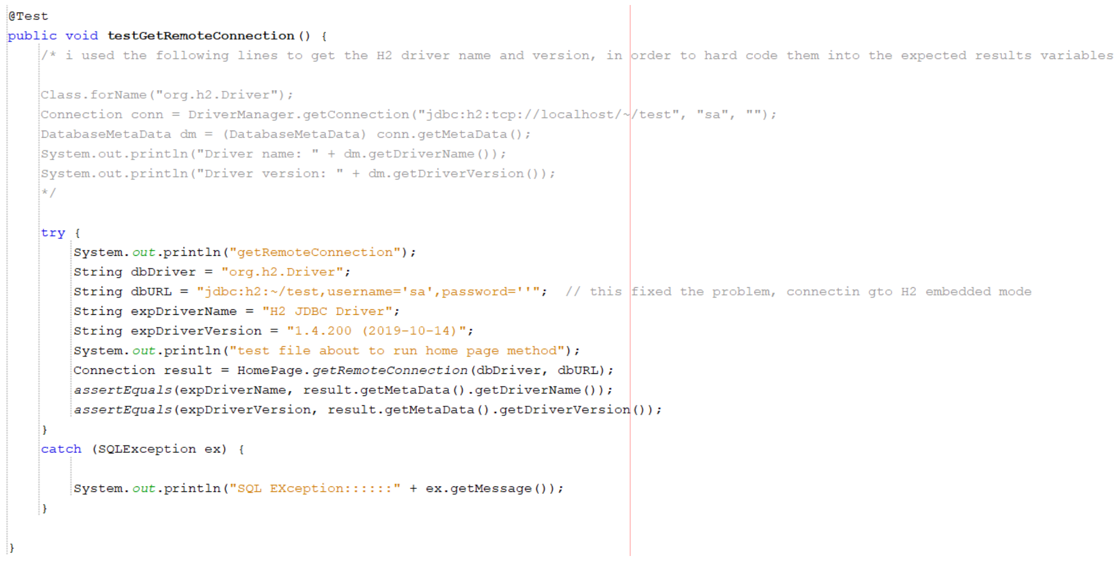
This project incorporates a desktop application, a mobile application and a database hosted on AWS.

Integration testing will be carried out:

- Between Mobile app and DB
- Between Desktop app and DB

Examples of integration testing carried out include the following:

For basic integration testing between the application and the database, a print statement was included in our code whenever a connection was generated between the application and the database. It would print “Method Name: Connected” if successful. We also used a catch to return any SQL exceptions that may have occurred and printed the exception to the output at run-time. A further unit test was written to check the connection when the application is initialised.



```java
@Test
public void testGetRemoteConnection() {
    /* i used the following lines to get the H2 driver name and version, in order to hard code them into the expected results variables

    Class.forName("org.h2.Driver");
    Connection conn = DriverManager.getConnection("jdbc:h2:tcp://localhost/~/test", "sa", "");
    DatabaseMetaData dm = (DatabaseMetaData) conn.getMetaData();
    System.out.println("Driver name: " + dm.getDriverName());
    System.out.println("Driver version: " + dm.getDriverVersion());
    */

    try {
        System.out.println("getRemoteConnection");
        String dbDriver = "org.h2.Driver";
        String dbURL = "jdbc:h2:~/test,username='sa',password=''";  // this fixed the problem, connectin gto H2 embedded mode
        String expDriverName = "H2 JDBC Driver";
        String expDriverVersion = "1.4.200 (2019-10-14)";
        System.out.println("test file about to run home page method");
        Connection result = HomePage.getRemoteConnection(dbDriver, dbURL);
        assertEquals(expDriverName, result.getMetaData().getDriverName());
        assertEquals(expDriverVersion, result.getMetaData().getDriverVersion());
    }
    catch (SQLException ex) {

        System.out.println("SQL EXception::::::" + ex.getMessage());
    }

}
```

Additionally to this, manual testing was undertaken frequently regarding the integration of our system. Print statements were written throughout the code to tell us when a sql query was being carried out successfully. Whenever we wrote code which involved the application displaying, inserting, updating or deleting data in our database, we would check the database to see if the database had updated successfully.

4. User Testing

Types of user testing to be carried out:

- Qualitative
- Quantitative

The following areas of the project will be user tested:

- GUI
- Pie chart use
- Ease of use
- Analysis

Both qualitative and quantitative user testing were carried out for this system.

For qualitative testing, we presented the different features of our applications to users and observed how they interacted with the system. Before we allowed them to access the app,

we described the concept of the feature of our project to them and provided an opportunity for them to ask any questions they might have. We then provided them access to the application and allowed them to navigate through it themselves. As they did this, we observed how they interacted with the application and took down any interesting observations seen. For some users, specific actions were asked of them (E.g. Access the home page, log a task, or configure a model). When this occurred we observed how they used the system and how long it took them to successfully carry out the task asked of them.
After they were finished interacting with the system, we asked them a series of questions regarding their experience. They were asked what they found most challenging about using the system, what they found the easiest, etc. It is important to note that the qualitative user testing was only carried out with family as Covid-19 restricted the nature of this testing with real-world users i.e. nurses.

Further qualitative analysis was undertaken for the analysis section of our project. This entailed presenting a series of different graphs representing the same data and asking the user to rank the graphs from most easy to understand to hardest to understand. We used this information to decide the structure of our analysis dashboard. Users were also asked to play with the analysis section and we observed their interaction with using different filters. After they were finished, we asked them to talk about their experience, what they liked, didnt like and how any suggestions they might have for improvements.

Qualitative user testing was also carried out. We provided a questionnaire to a series of nurses whom we knew. The concept of the system was first explained to them and they were asked their opinion on how beneficial the system would be to them. We included screenshots of different features of our app and asked them questions regarding each one. These questions included what their opinion was on the understandability, etc.