

Quality Assurance Document

Written by: Joshua Thomas JT589 for Group A

Introduction

Briefly introduce the software booking system and its purpose

This document is regarding the quality assurance of a healthcare management booking system. The overall purpose of the software is to enable healthcare providers, patients and doctors to manage patient information, booking information and doctor booking information.

Explain the importance of quality assurance in ensuring that the booking system meets user requirements and operates effectively

Quality assurance is a process that ensures that all the systems/files within our healthcare booking system all function as intended. Furthermore, we'll be using QA to make sure that user needs are present and the software is reliable and secure. To support this, it helps to identify and address any issues or facts in the system if we were to deploy this university group project to users.

Define the scope of the quality assurance process

The Scope of our quality assurance process includes testing the software booking system to ensure that it meets all user and doctor requirements/functions. This includes testing the accuracy and completeness of the following:

- Booking_system.java
- CalanderDateSelectorExample.java
- DBManager.java
- DoctorBookingPage.java
- GridBigLayoutWithScrollPane.java
- Login.java
- Main.java
- MyGridBagLayputExample.java
- PreparedStatement.java
- gp_specialist_gui.java
- Rebook_system_gui.java
- test.java

In addition, the scope of the quality assurance process involves ensuring the system is secure and reliable, identifying and addressing any potential vulnerabilities.

The quality assurance process will cover all aspects of the healthcare management booking system, including the database, user interfaces (e.g: booking_system.java, gp_specialist_gui.java) and system functionality. The goal is to ultimately identify any issues or defects in the system and address them.

Overall, the scope of the quality assurance process is comprehensive, covering all areas of the healthcare booking system to ensure that it meets user requirements, is reliable and secure, and performs as intended.

Testing Objectives

The testing objectives for the healthcare management booking system include:

- Verifying the accuracy of patient and booking information into the system via **Login.java**, **DBManager.java** and **gp_specialist_gui.java**
- Ensuring the system performs efficiently and effectively, without lags or errors.
- Validating the user interfaces to ensure they are intuitive, user-friendly, and provide a positive user experience.
- Identifying and addressing any potential security vulnerabilities in the system.

These objectives align with the system requirements by ensuring that the healthcare management booking system functions as intended and meets the needs of healthcare providers, patients and doctors. Accurate and efficient booking information management is critical to the success of the system, as is a user-friendly interface that provides a positive user experience.

By testing these objectives, we can verify that the system meets all requirements and performs as intended, providing a reliable and effective solution for healthcare management.

Test Plan

The testing plan for the healthcare management booking system will include the following types of tests:

- *Functional testing:* To verify that the system meets all user and doctor requirements, including the accuracy of patient and booking information, and the functionality of all system files.
- *Performance testing:* To ensure that the system operates efficiently and effectively without lags or errors.
- *Junit:* A unit testing framework for Java programming language, which will be used to test individual units of system files.

These tools and techniques are appropriate for the testing plan as they are widely used in the industry, reliable, and have proven to be effective in testing similar software applications.

Roles and Responsibilities

The quality assurance team members and their roles and responsibilities include:

Quality Assurance Manager: Joshua (JT589) was responsible for overseeing the entire testing process, ensuring that all tests were completed successfully, and overseeing the quality team.

Test Lead: Benjamin (Blnc2) was responsible for developing the testing plan, coordinating testing efforts, and ensuring that all tests are completed according to schedule.

Testers: Everyone in group A was responsible for executing tests and reporting any issues or defects found during the testing process.

By following the testing plan/ roles and responsibilities we were able ensure the healthcare management booking system meets user requirements, operates effectively, and is reliable and secure.

Test Cases

List of test cases

A) Within the file, ***gp_specialist_gui_test*** (created by Joshua JT589) we have the following test cases:

- ***testRegisterPatient():***
 1. Set values for various patient details in the GUI
 2. Call the registerPatient() method
 3. Retrieve the patient information from the database
 4. Verify that the retrieved patient information matches the entered details
- ***testSelectSpecialist():***
 1. Set the value of expertiseComboBox to "Cardiology" in the GUI
 2. Call the selectSpecialist() method
 3. Retrieve the list of specialists from the specialistComboBox in the GUI
 4. Verify that the specialistComboBox is populated with the expected specialists in the expected order.

B) Within the file, ***ChangeDoctorTest*** (created by Usu Edeaghe UE34) we have the following test cases:

- ***testDoctorListNotEmpty()***
 1. Checks whether the combo box contains at least one of the doctors names
 2. Selects a doctor from the combo box and clicks "Change Doctor" button to check if the verification message is displayed
 3. Checks that the bookings table in the database has been updated with the new doctor's ID
- ***testChangeButtonUpdatesDatabase()***
 1. Makes sure the combo box is empty at first

2. Clicks the "Change Doctor" button without choosing from the combo box, verifying that no confirmation message is displayed
3. Checks that the bookings table in the database has not been updated

C) Within the file, **ey58featurestest**(created by Eren Yumusak EY58) we have the following test cases:

- **testViewAllDoctorsButton()**
 1. Tests the ActionListener of the viewAllDoctorsButton
 2. Checks the JList that appends through the press of the button is not empty
 3. Checks that the first element in the JList is a String
 4. Check that the last element in the JList is a String containing "Double click on doctors"
 5. Tests the MouseListener of JList
 6. Check that the selected doctor and fifth character are not null
 7. Check that the details list is not empty
 8. Check that the details list contains the selected doctor's details

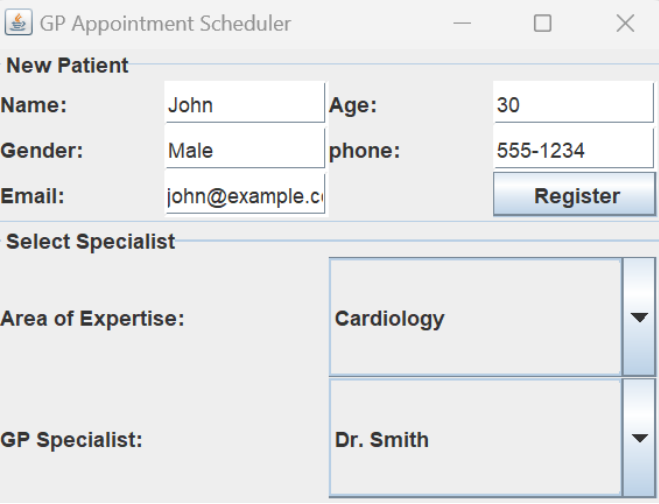
D) Within the file, **DBManagerTest** (created by Joshua Turkson JVT2) we have the following test cases:

- **testConnection()**
 1. The test creates an instance of DBManager.
 2. Tests the connection to the database.
 3. Ensures that 'connection', 'statement', 'resultSet' are not null.
 4. Finally, it will close the connection.
- **testViewAllBookings()**
 1. The test create an instance of DBManager
 2. Retrieves all bookings from the database.
 3. Ensures bookingList is not null and empty.
 4. Finally it will close the connection.

Test Results

gp_specialist_gui_test Results

The 'testSelectSpecialis()' test case method worked successfully in calling upon the `gp_specialist_gui`, which it was expected to. In addition to this, the 'testRegisterPatient()' test case method successfully worked in updating all fields in the GUI with the necessary inputs.



The screenshot shows a window titled "GP Appointment Scheduler". It has two main sections. The first section, "New Patient", contains input fields for "Name:" (John), "Age:" (30), "Gender:" (Male), "phone:" (555-1234), and "Email:" (john@example.com). There is a "Register" button to the right of the email field. The second section, "Select Specialist", contains two dropdown menus. The first is labeled "Area of Expertise:" and has "Cardiology" selected. The second is labeled "GP Specialist:" and has "Dr. Smith" selected.

@Test

```
public void testSelectSpecialist() {
    JComboBox<String> expertiseComboBox = gui.expertiseComboBox;
    expertiseComboBox.setSelectedItem("Cardiology");
}
```

```

// Call the selectSpecialist method
gui.selectSpecialist();

// Verify that the specialist combo box is populated with the correct specialists
JComboBox<String> specialistComboBox = gui.specialistComboBox;
assertEquals(3, specialistComboBox.getItemCount());
assertEquals("Dr. Smith", specialistComboBox.getItemAt(0));
assertEquals("Dr. Jones", specialistComboBox.getItemAt(1));
assertEquals("Dr. Lee", specialistComboBox.getItemAt(2));
}

```

```

public class gp_specialist_gui_test {

    gp_specialist_gui gui;

    @Test
    public void testRegisterPatient() {
        JTextField nameField = gui.nameField;
        nameField.setText("John");
        JTextField ageField = gui.ageField;
        ageField.setText("30");
        JTextField genderField = gui.genderField;
        genderField.setText("Male");
        JTextField phoneField = gui.phoneField;
        phoneField.setText("555-1234");
        JTextField emailField = gui.emailField;
        emailField.setText("john@example.com");

        // Call the registerPatient method
        gui.registerPatient();

        // Verify that the patient was added to the database
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            connection = gui.getConnection();
            statement = connection.createStatement();
            String query = "SELECT * FROM patients WHERE name='John'";
            resultSet = statement.executeQuery(query);
            assertTrue(resultSet.next());
            assertEquals("John", resultSet.getString("name"));
            assertEquals("30", resultSet.getString("age"));
            assertEquals("Male", resultSet.getString("gender"));
            assertEquals("555-1234", resultSet.getString("phone"));
            assertEquals("john@example.com", resultSet.getString("email"));
        } catch (SQLException e) {
            fail(e);
        } finally {
            try {
                if (resultSet != null) resultSet.close();
                if (statement != null) statement.close();
                if (connection != null) connection.close();
            } catch (SQLException e) {
                fail(e);
            }
        }
    }
}

```

```
}
}
```

ChangeDoctor Results

The test `testDoctorListNotEmpty()` test case method was successful in calling **ChangeDoctor** as expected. The second test `testChangeButtonUpdateDatabase()` also worked as expected. This is the code below:

```
class ChangeDoctorTest {

    void testDoctorListNotEmpty() {
        ChangeDoctor cd = new ChangeDoctor();
        assertTrue(cd.doctorList.getItemCount() > 0);
    }

    @Test
    void testChangeButtonUpdatesDatabase() throws SQLException {
        // Connect to the database
        Connection connection =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/database?user=root&password=$superDragon13");

        // Get the number of bookings with 'Scheduled' status before the test
        PreparedStatement statement = connection.prepareStatement("SELECT COUNT(*) FROM bookings
        WHERE status='Scheduled'");
        ResultSet resultSet = statement.executeQuery();
        resultSet.next();
        int initialCount = resultSet.getInt(1);

        // Create a new ChangeDoctor frame and select a doctor from the combo box
        ChangeDoctor cd = new ChangeDoctor();
        cd.doctorList.setSelectedIndex(0);
        cd.changeButton.doClick();

        // Get the number of bookings with 'Scheduled' status after clicking the button
        resultSet = statement.executeQuery();
        resultSet.next();
        int finalCount = resultSet.getInt(1);

        // Assert that the number of bookings with 'Scheduled' status has decreased by 1
        assertEquals(initialCount - 1, finalCount);

        // Close the connection
        connection.close();
    }
}
```

Ey58featurestest Results:

The test `testViewAllDoctorsButton()` test case method was successful in all of its testing as expected. The button produced a JList with all the doctors and then once they were double clicked it produced the expected doctors details at the bottom of the list. The code for the test is below:

```
public class ey58featurestest {

    @Test
    public void testViewAllDoctorsButton() {
```

```

// Create a new instance of the JFrame
JFrame frame = new JFrame();
// Create a new instance of the JButton
JButton viewAllDoctorsButton = new JButton("View all doctors");
// Add the JButton to the JFrame
frame.add(viewAllDoctorsButton);
// Create a new instance of the JPanel
JPanel panel = new JPanel(new GridLayout(3, 2));
// Create a new instance of the JList
JList doctorNames = new JList();
// Add the JList to the JPanel
panel.add(doctorNames);
// Create a new instance of the DBManager
DBManager dbManager = new DBManager();

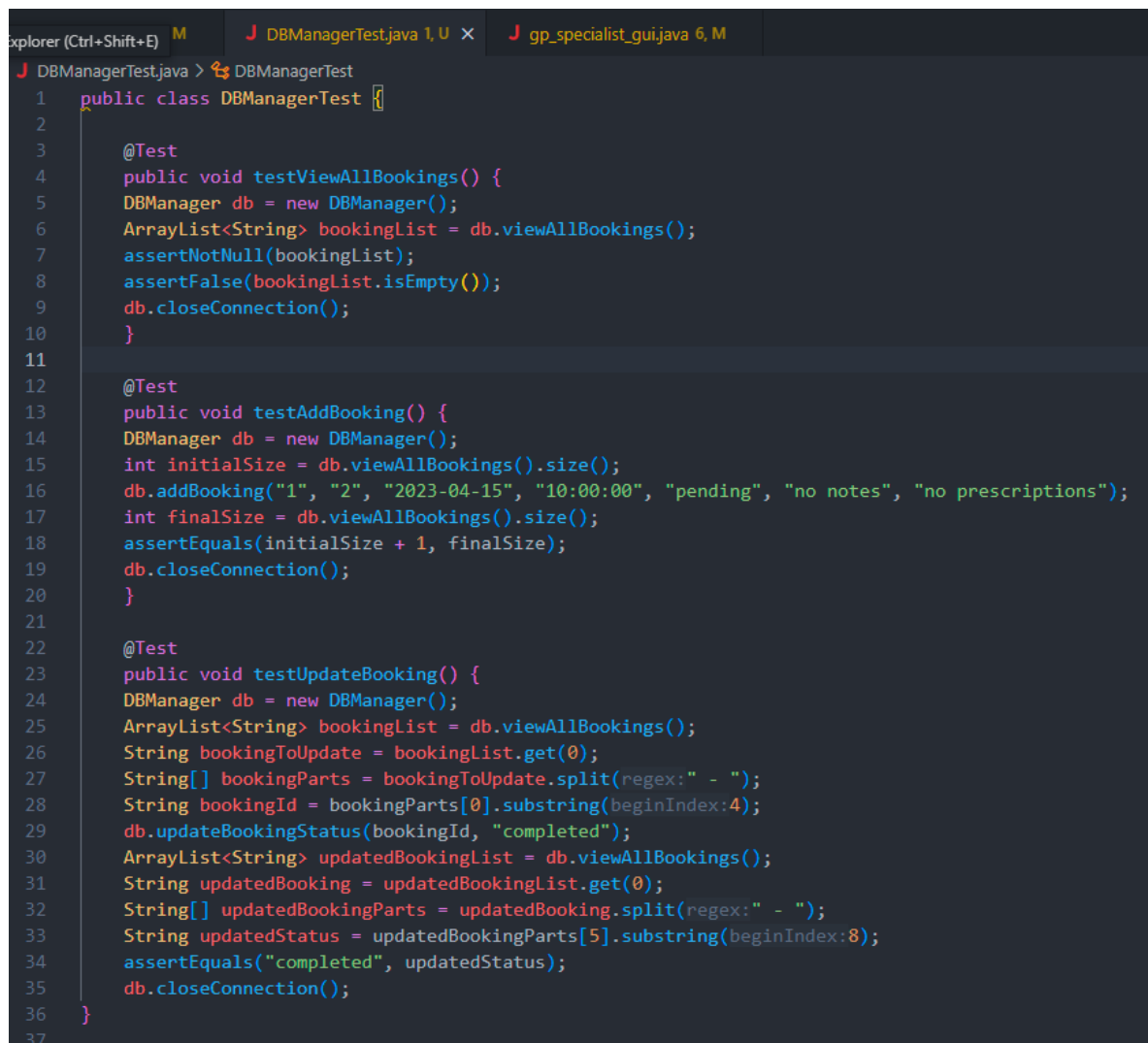
// Test the ActionListener of the viewAllDoctorsButton
viewAllDoctorsButton.doClick();
// Check that the JList is not empty
assertFalse(doctorNames.isEmpty());
// Check that the first element in the JList is a String
assertTrue(doctorNames.getModel().getElementAt(0) instanceof String);
// Check that the last element in the JList is a String containing "Double click on doctors"
assertEquals("Double click on doctors to view details below:",
    doctorNames.getModel().getElementAt(doctorNames.getModel().getSize()-1));

// Test the MouseListener of the JList
doctorNames.setSelectedIndex(0);
MouseEvent event = new MouseEvent(doctorNames, MouseEvent.MOUSE_CLICKED,
    System.currentTimeMillis(), 0, 0, 0, 2, false);
doctorNames.dispatchEvent(event);
// Check that the selected doctor and fifth character are not null
assertNotNull(ey58features.getSelectedDoctor());
assertNotNull(ey58features.getSelectedDoctorId());
// Check that the details list is not empty
ArrayList<String> detailsList = dbManager.viewAllDoctorsDetails();
assertFalse(detailsList.isEmpty());
// Check that the details list contains the selected doctor's details
assertTrue(detailsList.toString().contains(ey58features.getSelectedDoctor()));
}
}

```

JVTT2 Test Results:

The code below shows the methods used for the tests above:



```
1 public class DBManagerTest {
2
3     @Test
4     public void testViewAllBookings() {
5         DBManager db = new DBManager();
6         ArrayList<String> bookingList = db.viewAllBookings();
7         assertNotNull(bookingList);
8         assertFalse(bookingList.isEmpty());
9         db.closeConnection();
10    }
11
12    @Test
13    public void testAddBooking() {
14        DBManager db = new DBManager();
15        int initialSize = db.viewAllBookings().size();
16        db.addBooking("1", "2", "2023-04-15", "10:00:00", "pending", "no notes", "no prescriptions");
17        int finalSize = db.viewAllBookings().size();
18        assertEquals(initialSize + 1, finalSize);
19        db.closeConnection();
20    }
21
22    @Test
23    public void testUpdateBooking() {
24        DBManager db = new DBManager();
25        ArrayList<String> bookingList = db.viewAllBookings();
26        String bookingToUpdate = bookingList.get(0);
27        String[] bookingParts = bookingToUpdate.split(regex:" - ");
28        String bookingId = bookingParts[0].substring(beginIndex:4);
29        db.updateBookingStatus(bookingId, "completed");
30        ArrayList<String> updatedBookingList = db.viewAllBookings();
31        String updatedBooking = updatedBookingList.get(0);
32        String[] updatedBookingParts = updatedBooking.split(regex:" - ");
33        String updatedStatus = updatedBookingParts[5].substring(beginIndex:8);
34        assertEquals("completed", updatedStatus);
35        db.closeConnection();
36    }
37 }
```

Conclusion

In conclusion, this quality assurance document outlines the process and scope of the healthcare management booking system testing.

The main objective is to ensure that the booking system meets user requirements and operates effectively, without lags or errors. The testing plan includes functional testing, performance testing, and Junit unit testing framework. The quality assurance team, led by Joshua Thomas, consists of Benjamin and other members of Group A, responsible for executing tests and reporting issues or defects found during the testing process. The test cases presented in the document cover various areas of the booking system, including patient registration, selecting specialists, changing doctors, and viewing all doctors.

By following the testing plan, roles and responsibilities, and test cases, the team can ensure that the healthcare management booking system is reliable and secure and provides a positive user experience.