

Testing Documentation

Testing: Login Page:

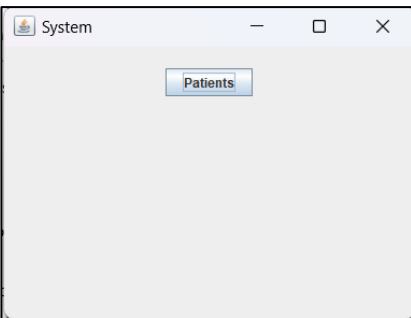
Successful Logins:

We will verify the effectiveness of login procedures by testing successful logins. Similar to standard login processes encountered on websites or software, users will be required to input their username and password. For security reasons, the password will be obscured.

One of the successful login details to get into the program is:

- Username = mworb0001
- Password = password123

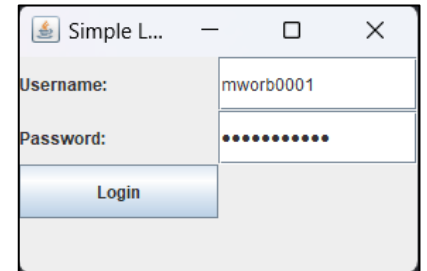
For testing purposes, the password is simple and not complex so that you can test the program efficiently.



As you can see, we have inputted the correct details for username and password. When the user clicks on the button 'Login' it will direct them to a new window which will be the main menu and this is what that looks like on the left-hand side.

This therefore shows that the login page has functionality and will allow correct user details to enter the system.

This is very case-sensitive therefore the user must use correct grammar and syntax when logging in or else it will be an unsuccessful login.



The code to the right is a function that tests the successful login. It automatically puts the correct details for the username and the password. This is done by this function 'setText' which will enter the values into the desired variable.

Now when the details of the user are entered, the test will automatically click the login button with the function 'doClick()' which will then send the user to the main menu once passed through authentication.

A message will also be put into the terminal saying 'Login Successful' just to clarify that it wasn't an accidental login.

```
@Test
void testSuccessfulLogin() {
    // Mocked LoginTool always returns 1 for UserID, indicating a successful login
    LoginTool loginTool = new LoginTool() {
        @Override
        public int login(String username, String password) {
            return 1; // Mock successful login
        }
    };

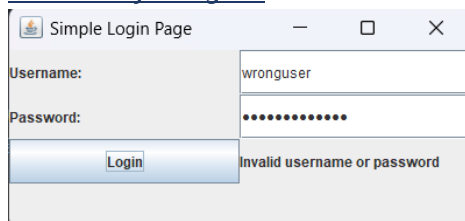
    // Create the LoginPage with the mocked LoginTool
    LoginPage loginPage = new LoginPage(loginTool);

    // Simulate user input
    loginPage.usernameField.setText("mworb0001");
    loginPage.passwordField.setText("password123");

    // Simulate button click
    loginPage.loginButton.doClick();

    // Verify the result
    assertEquals("Login Successful", loginPage.messageLabel.getText());
}
```

Unsuccessful Logins:



We will assess the robustness of login procedures by conducting tests on unsuccessful login attempts. Similar to typical login processes encountered on websites or software, users will need to enter their username and password. To uphold security measures, the password will remain obscured. As you can see access was denied and it says invalid username and password.

On the right we can see the tester code for testing against incorrect logins. For the sake of testing, the input used for username and password are 'wronguser' and 'wrongpassword' respectively.

When the test runs this, it will be notified as a successful test for an unsuccessful login as '-1' will be returned. There will also be a message stating, 'Invalid username or password'.

Therefore, successful and unsuccessful logins are both working.

```
@Test
void testUnsuccessfulLogin() {
    // Mocked LoginTool returns -1 for UserID, indicating an unsuccessful login
    LoginTool loginTool = new LoginTool() {
        @Override
        public int login(String username, String password) {
            return -1; // Mock unsuccessful login
        }
    };

    // Create the LoginPage with the mocked LoginTool
    LoginPage loginPage = new LoginPage(loginTool);

    // Simulate user input
    loginPage.usernameField.setText("wronguser");
    loginPage.passwordField.setText("wrongpassword");

    // Simulate button click
    loginPage.loginButton.doClick();

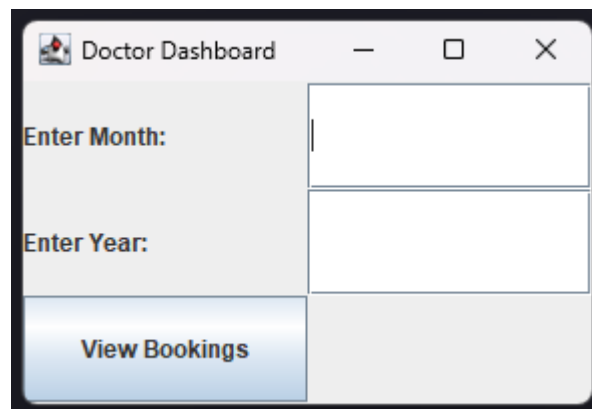
    // Verify the result
    assertEquals("Invalid username or password", loginPage.messageLabel.getText());
}
```


Testing: Doctor's Dashboard:

The doctor's dashboard has one intended functionality. As per the functionality required by the brief, it allows the user to input a month and a year.

The system will then display all appointments made during the provided month.

The image provided to the left shows the format of this data input.

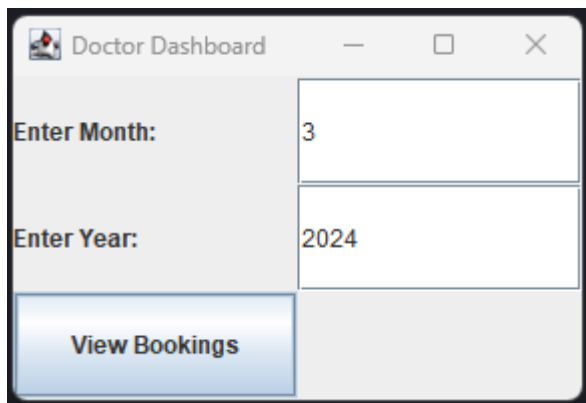


The screenshot shows a window titled "Doctor Dashboard". It contains two input fields: "Enter Month:" and "Enter Year:". Below these fields is a button labeled "View Bookings".

```
@Test
public void testGrabAppointments() {
    DoctorDashboard doctorDashboard = new DoctorDashboard();
    doctorDashboard.yearField.setText("2024");
    doctorDashboard.monthField.setText("03"); // There is one booking in the database in this month

    doctorDashboard.viewBookingsButton.doClick(); // Simulate pressing the button

    assertEquals("expected: 'Appointments in the given month:\nMax Worby has a booking with Dr Worby on 2024-03-10.'", doctorDashboard.messageLabel.getText());
}
```

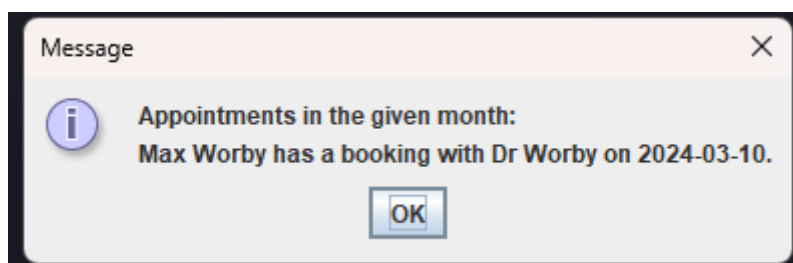


The screenshot shows the "Doctor Dashboard" window with the "Enter Month:" field containing "3" and the "Enter Year:" field containing "2024". The "View Bookings" button is still visible.

The code above shows the Test code written for this function. It tests if the result is as it should be given the inputs "03" and "2024". There is only one appointment for the given month, so the correct result is that one appointment (an appointment made between a doctor and a patient who are the same user, myself).

To be thorough, I tested this using the GUI as well.

The inputs I gave are shown to the right, and the result is shown below.



The screenshot shows a "Message" dialog box with an information icon. The text inside reads: "Appointments in the given month: Max Worby has a booking with Dr Worby on 2024-03-10." There is an "OK" button at the bottom right.