



DEMOP

AREA:

"MULTIPLATFORM SOFTWARE DEVELOPMENT"

PROJECT:

"U3 ACT 1"

PRESENT:

MARCO AURELIO VALADEZ GUZMAN

TEACHER:

ANTONIO MOISES GONZAGA

GRADE & GROUP:

10 "B"

CD. REYNOSA, TAMAULIPAS

MAR 2023.

Contenido

Software testing for mobile applications.....	1
Functional testing	1
Example with code.....	1
Usability testing	2
Example with code.....	2
Performance testing.....	3
Example with code.....	3

Software testing for mobile applications

Functional testing

This type of testing is used to ensure that the application's features and functions work as intended. It includes testing user interfaces, navigation, data input and output, and overall application behavior.

Example with code

In this example, we are testing the login functionality of a mobile application. We start the Appium server and create a driver instance. Then, we find the login button and click it to open the login screen. We find the username and password fields and input test data. We find the submit button and click it to submit the form. Finally, we verify that the dashboard page is displayed, and close the driver and stop the Appium server.

This example uses Appium to automate the testing process, and Java as the programming language. It demonstrates how functional testing can be performed to ensure that the application's features and functions work as intended.

```
java Copy code

@Test
public void testLogin() {
    // Start the Appium server and create a driver instance
    AppiumDriver<MobileElement> driver = startAppiumServerAndCreateDriver();

    // Find the login button and click it
    MobileElement loginButton = driver.findElement(By.id("login_button"));
    loginButton.click();

    // Find the username and password fields, and input test data
    MobileElement usernameField = driver.findElement(By.id("username_field"));
    usernameField.sendKeys("testuser");
    MobileElement passwordField = driver.findElement(By.id("password_field"));
    passwordField.sendKeys("testpass");

    // Find the submit button and click it
    MobileElement submitButton = driver.findElement(By.id("submit_button"));
    submitButton.click();

    // Verify that the dashboard page is displayed
    MobileElement dashboardTitle = driver.findElement(By.id("dashboard_title"));
    Assert.assertEquals("Dashboard", dashboardTitle.getText());

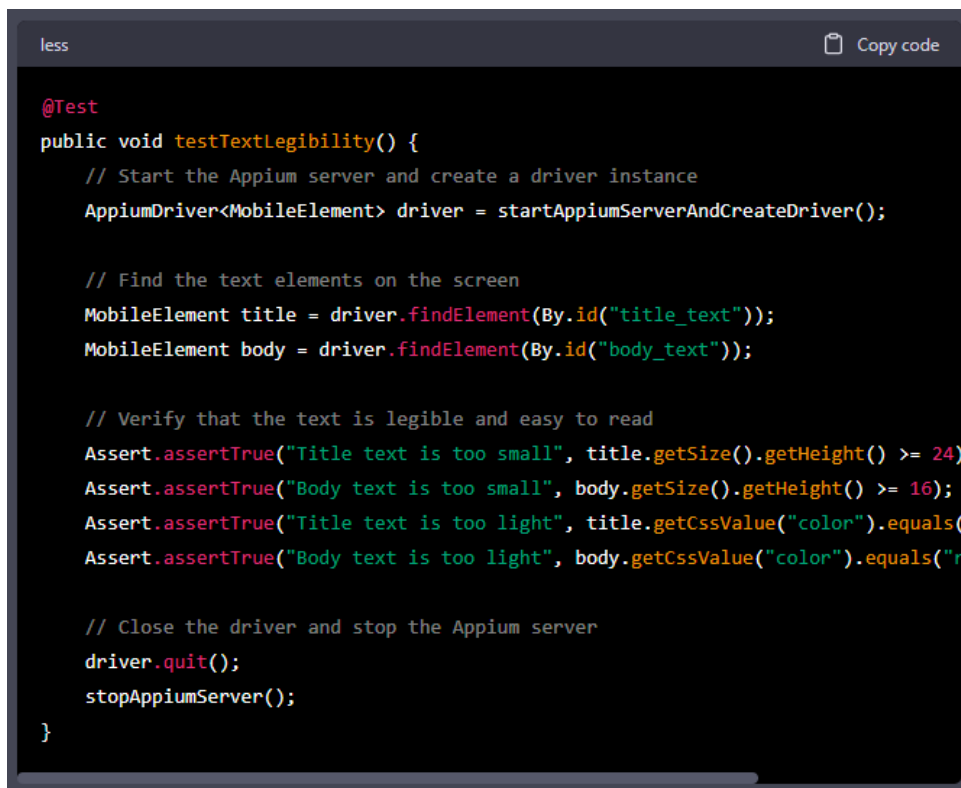
    // Close the driver and stop the Appium server
    driver.quit();
    stopAppiumServer();
}
```

Usability testing

This type of testing is used to evaluate the application's user experience and user interface design. It includes testing user interactions, accessibility, readability, and overall ease of use.

Example with code

Usability testing is typically conducted by gathering feedback from real users, rather than through automation or code. However, here's an example of a test case that can be used to evaluate the readability of an application's user interface:

A screenshot of a code editor with a dark background. The editor shows a Java test method named `testTextLegibility()`. The code includes comments in green, annotations in pink, and Java code in white. The test method starts by starting the Appium server and creating a driver instance. It then finds the title and body text elements on the screen. Next, it verifies that the text is legible and easy to read by using assertions to check the font size and color of the text elements. Finally, it closes the driver and stops the Appium server.

```
less Copy code  
  
@Test  
public void testTextLegibility() {  
    // Start the Appium server and create a driver instance  
    AppiumDriver<MobileElement> driver = startAppiumServerAndCreateDriver();  
  
    // Find the text elements on the screen  
    MobileElement title = driver.findElement(By.id("title_text"));  
    MobileElement body = driver.findElement(By.id("body_text"));  
  
    // Verify that the text is legible and easy to read  
    Assert.assertTrue("Title text is too small", title.getSize().getHeight() >= 24);  
    Assert.assertTrue("Body text is too small", body.getSize().getHeight() >= 16);  
    Assert.assertTrue("Title text is too light", title.getCssValue("color").equals("r  
    Assert.assertTrue("Body text is too light", body.getCssValue("color").equals("r  
  
    // Close the driver and stop the Appium server  
    driver.quit();  
    stopAppiumServer();  
}
```

In this example, we are testing the legibility and readability of text elements on the screen. We start the Appium server and create a driver instance. Then, we find the title and body text elements and verify that they are legible and easy to read. We use assertions to check the font size and color of the text elements and ensure that they meet the minimum requirements for readability.

While this example doesn't necessarily capture the full scope of usability testing, it demonstrates how tests can be created to evaluate specific aspects of the user interface and user experience. Usability testing should

ideally involve real users, who can provide feedback on the application's overall ease of use and user-friendliness.

Performance testing

This type of testing is used to measure the application's speed, stability, and scalability. It includes testing the application's response time, memory usage, battery consumption, and network connectivity.

Example with code

Sure, here's an example of performance testing with code using JMeter:

```
java Copy code  
  
import org.apache.jmeter.engine.StandardJMeterEngine;  
import org.apache.jmeter.protocol.http.sampler.HTTPSampler;  
import org.apache.jmeter.reporters.ResultCollector;  
import org.apache.jmeter.reporters.Summariser;  
import org.apache.jmeter.testelement.TestPlan;  
import org.apache.jmeter.threads.ThreadGroup;  
import org.apache.jmeter.util.JMeterUtils;  
import org.junit.Test;  
  
public class PerformanceTest {  
    @Test  
    public void testWebsiteLoadTime() {  
        // Set up the JMeter engine and test plan  
        JMeterUtils.setJMeterHome("/path/to/jmeter");  
        JMeterUtils.loadJMeterProperties("/path/to/jmeter.properties");  
        StandardJMeterEngine engine = new StandardJMeterEngine();  
        TestPlan testPlan = new TestPlan();  
        testPlan.setName("Website Load Time Test");  
  
        // Set up the HTTP sampler and thread group  
        HTTPSampler httpSampler = new HTTPSampler();  
        httpSampler.setDomain("www.example.com");  
        httpSampler.setPath("/");  
        ThreadGroup threadGroup = new ThreadGroup();  
        threadGroup.setNumThreads(10);  
        threadGroup.setRampUp(5);  
        threadGroup.setSamplerController(httpSampler);  
    }  
}
```

```

        // Set up the result collector and summariser
        Summariser summariser = new Summariser();
        ResultCollector resultCollector = new ResultCollector(summariser);

        // Add the test plan elements to the JMeter engine
        testPlan.addThreadGroup(threadGroup);
        testPlan.addTestElement(httpSampler);
        testPlan.addTestElement(resultCollector);

        // Configure and run the test
        engine.configure(testPlan);
        engine.run();

        // Get the test results and print them to the console
        System.out.println("Total requests: " + summariser.getTotal());
        System.out.println("Average response time: " + summariser.getAverage());
        System.out.println("Max response time: " + summariser.getMax());
        System.out.println("Min response time: " + summariser.getMin());
        System.out.println("Standard deviation: " + summariser.getStandardDeviation());
    }
}

```

In this example, we are testing the load time of a website using JMeter. We set up the JMeter engine and test plan, and configure an HTTP sampler and thread group. We also set up a result collector and summariser to collect performance data.

We then add the test plan elements to the JMeter engine and run the test. After the test has completed, we retrieve the test results from the summarizer and print them to the console. The results include the total number of requests, average response time, maximum response time, minimum response time, and standard deviation.

This example demonstrates how performance testing can be performed using JMeter, a popular open-source tool for load testing and performance testing. By measuring and analyzing performance metrics, we can identify performance bottlenecks and optimize the application's performance.