Candidate Name:MD ATIQUL ISLAM.

Email- atik.cmttiu1001@gmail.com

Qustions of the Assessment:

#Mobile Applications

1. Write 10 Test Cases CRUD Operation.

2. You need to learn about Appium clients.

3. Automate these 10 Test Cases by using selenium/WebdriverIO.

4. Generate an Html report

1.Answer: Sure, here are 10 test cases for CRUD (Create, Read, Update, Delete) operations in a mobile application:

1. *Create - Add a New Item:*

   - Test that a new item can be successfully added with all required fields filled in.

   - Verify that the newly added item is displayed correctly in the list.

2. *Create - Empty Fields Validation:*

   - Test that the application prevents the creation of an item with empty or invalid fields.

   - Confirm that appropriate error messages are displayed for missing or invalid data.

3. *Read - View Item Details:*

   - Select an existing item and verify that its details are displayed accurately.

   - Ensure that all relevant information, such as name, description, and date, is shown correctly.

4. *Update - Modify Item Information:*

   - Edit an existing item and save the changes.

- Confirm that the item's details are updated correctly, including any changes made to its data.

5. *Update - Cancel Editing:*
   - Start editing an item's details but then cancel the operation.
   - Ensure that the item's original data remains unchanged.

6. *Delete - Remove an Item:*
   - Delete an item and confirm that it is removed from the list.
   - Verify that the item can no longer be accessed.

7. *Delete - Cancel Deletion:*
   - Attempt to delete an item but then cancel the deletion process.
   - Ensure that the item remains in the list and retains all its data.

8. *Create - Duplicate Prevention:*
   - Try to create a new item with the same unique identifier (e.g., name) as an existing item.
   - Verify that the application prevents the creation of duplicate items.

9. *Read - Empty List Handling:*
   - Delete all items from the list and confirm that the application handles an empty list gracefully.
   - Ensure it provides a user-friendly message or guidance.

10. *Update - Validation on Edit:*
    - Attempt to update an item with invalid data (e.g., a date in the past).
    - Confirm that the application prevents the update and provides appropriate error feedback.

These test cases cover the basic CRUD operations and common scenarios you'd encounter when working with data in a mobile application. Be sure to adapt them to your specific application and its requirements.

2.Answer: Appium is an open-source automation tool for testing mobile applications on various platforms like Android and iOS. To interact with Appium, you need to use client libraries in your preferred programming language. These client libraries provide the necessary functions and methods to automate mobile app testing. Some popular Appium client libraries include:

1. *Java*: Appium Java client is commonly used for Android and iOS app automation in Java. You can use it with tools like JUnit or TestNG for test management.

2. *Python*: Appium Python client is popular among Python developers. It allows you to write test scripts in Python for mobile app testing.

3. *JavaScript (Node.js)*: Appium provides a Node.js client, which is suitable for JavaScript developers. You can write test scripts in JavaScript to automate mobile apps.

4. *C#*: If you prefer C#, there's an Appium C# client that integrates well with the .NET ecosystem. It's used for automating apps on Windows platforms, Android, and iOS.

5. *Ruby*: Appium also has a Ruby client, making it convenient for Ruby developers to automate mobile app testing.

3.Answer: This is an example of how you can automate 10 test cases for a mobile application using Selenium and WebdriverIO. Please note that this is a high-level overview, and you would need to adapt it to your specific application and test scenarios.

javascript

```
const { remote } = require('webdriverio');

async function runTests() {
```

```
const opts = {
    port: 4723, // Appium server port
    capabilities: {
        platformName: 'Android', // or 'iOS'
        deviceName: 'YourDeviceName',
        app: 'path/to/your/app.apk', // Path to your mobile app
        automationName: 'UiAutomator2', // Use UiAutomator2 for Android
    },
};
const driver = await remote(opts);
try {
    // Test Case 1: Launch the app
    await driver.launchApp();

    // Test Case 2: Perform some actions (e.g., clicking buttons, entering text)
    await driver.element('locator').click();
    await driver.element('locator').setValue('text');

    // Test Case 3: Verify elements
    const element = await driver.element('locator');
    assert(await element.isDisplayed());

    // Test Case 4: Navigate to another screen
    await driver.element('locator').click();

    // Test Case 5: Perform actions on the new screen
```

```
        // ...

        // Test Case 6: Go back to the previous screen
        await driver.back();

        // Test Case 7: Verify something else
        // ...

        // Test Case 8: Close the app
        await driver.closeApp();

        // Test Case 9: Re-launch the app
        await driver.launchApp();

        // Test Case 10: Perform final verification
        // ...
    } catch (error) {
        console.error('Test failed:', error);
    } finally {
        await driver.deleteSession(); // Close the session
    }
}

runTests();
```

In this example, you would need to replace `'locator'` with the appropriate locator strategy and value for the elements you want to interact with in your app. You should also replace `'YourDeviceName'` with the name of your target device and `'path/to/your/app.apk'` with the actual path to your mobile app's APK file.

4.Answer: Generating an HTML report for mobile application test automation can be achieved using various testing frameworks and reporting libraries. One popular choice is to use Mocha as the test framework along with a reporter like "mochawesome" for generating HTML reports. Here's a step-by-step guide on how to generate an HTML report for your mobile application tests using Selenium and WebdriverIO with Mocha and mochawesome:

1. *Install Dependencies*:

First, ensure you have the necessary dependencies installed. You can install them using npm (Node Package Manager):

bash

npm install webdriverio mocha mochawesome

2. *Create Test Files*:

Organize your mobile application test cases into separate JavaScript files, each containing individual test cases. For example, create `testcase1.js`, `testcase2.js`, and so on.

3. *Write Test Scripts*:

In each test script (e.g., `testcase1.js`), write your mobile application test cases using WebdriverIO and Mocha. Make sure to include assertions to validate the test results.

4. *Create a Test Suite*:

Create a test suite file (e.g., `testsuite.js`) to run all your test scripts together. This file will orchestrate the execution of your test cases.

```
javascript

// testsuite.js

describe('Mobile App Test Suite', () => {

    require('./testcase1.js');

    require('./testcase2.js');

    // Include more test cases if needed

});
```

5. *Generate HTML Report*:

    To generate an HTML report using mochawesome, you can use the following command:

```
bash

npx mocha testsuite.js --reporter mochawesome
```

    This command runs your test suite and generates an HTML report named `mochawesome-report` in the project directory.

6. *View the Report*:

    You can view the HTML report by opening the generated `mochawesome-report/mochawesome.html` file in a web browser.

That's it! You'll have an HTML report that summarizes the results of your mobile application test automation. You can customize the appearance and behavior of the report by configuring mochawesome options in your test suite or via the configuration file.

Remember to adapt your test scripts to include appropriate assertions and error handling to ensure that the report accurately reflects the test outcomes.