Candidate Name:MD ATIQUL ISLAM.

Email- atik.cmttiu1001@gmail.com

Qustions of the Assessment:

2nd Assessment:

Application URL: https://hishabee.business/

#Mobile Applications

1. Write 20 Test Cases From the (products) module CRUD Operation.

2. You need to learn about Appium clients.

3. Automate these 20 Test Cases by using selenium/WebdriverIO.

4. Generate an Html report

1.Answer: This is the CRUD (Create, Read, Update, Delete) operations in the "Products" module of a mobile application. Here are 20 test cases:

Create (C) Test Cases:

1. Verify that a new product can be created with valid data.

2. Ensure that mandatory fields (e.g., product name, price) are required for product creation.

3. Test creating a product with special characters in the name.

4. Test creating a product with a very long name.

5. Verify that product creation fails if required fields are missing.

6. Test creating a product with a negative price.

7. Test creating a product with a very high price.

8. Verify that the system handles concurrent product creation requests correctly.

9. Test creating a product with duplicate information.

10. Check if the system displays appropriate error messages for failed creations.

Read (R) Test Cases:

11. Verify that all products are displayed in the product list.

12. Check if the product details page displays the correct information.

13. Test searching for a product by name and ensure the correct product is displayed.

14. Verify that pagination works correctly if there are many products.

15. Test sorting products by different criteria (e.g., price, name).

16. Ensure that the product list is updated in real-time when a new product is added.

Update (U) Test Cases:

17. Test updating the name of an existing product.

18. Verify that updating the price of a product works as expected.

19. Test updating other attributes (e.g., description, image) of a product.

20. Check if the system prevents updating with invalid data (e.g., empty name).

Delete (D) Test Cases:

21. Verify that a product can be deleted successfully.

22. Check if deleting a product removes it from the product list.

23. Test deleting a product that is associated with orders, ensuring data integrity.

24. Verify that the system prompts for confirmation before deleting a product.

These test cases cover various scenarios to ensure that the CRUD operations in the "Products" module of the mobile application are thoroughly tested. You can adapt them as needed for your specific application and requirements.

2.Answer: This is an overview of how to write code using Appium client libraries to automate mobile

app testing. However, it's important to note that the specific code syntax and usage may vary depending on the programming language you choose (e.g., Java, Python, JavaScript, C#, Ruby). Below is a general outline of what your code might look like:

1. *Import Appium Library:*

   - In your chosen programming language, import the Appium client library to access its functions and classes.

2. *Set Desired Capabilities:*

   - Create an instance of `DesiredCapabilities` to specify the configuration for your mobile app test. These capabilities include details like the platform name, device name, app package, app activity, and automation name.

3. *Initialize Appium Driver:*

   - Initialize an Appium driver by passing the desired capabilities to the driver constructor. This driver will be used to interact with the mobile app.

4. *Locate Elements:*

   - Use methods provided by the Appium library to locate elements within the mobile app. Common methods include `findElementById`, `findElementByXPath`, `findElementByClassName`, etc.

5. *Perform Actions:*

   - Once you've located elements, you can perform actions such as tapping, swiping, entering text, clicking buttons, etc., using methods provided by the Appium library. For example:

```
python

driver.find_element_by_id("elementId").click()

driver.find_element_by_xpath("//android.widget.Button[@text='Submit']").click()

driver.find_element_by_id("inputField").send_keys("Hello, Appium!")
```

6. *Assertions:*

- Implement assertions to verify that the app behaves as expected. For example, you can check if certain elements are displayed or if text matches expectations.

7. *Error Handling:*

- Implement error handling to handle exceptions that may occur during test execution. Common exceptions include `NoSuchElementException` and `TimeoutException`.

8. *Waits and Synchronization:*

- Use explicit and implicit waits to handle synchronization issues in your tests. Waiting for elements to become visible or clickable is crucial to avoid race conditions.

9. *Test Cleanup:*

- After each test, make sure to perform any necessary cleanup, such as closing the driver or resetting the app to its initial state.

10. *Reporting:*

- Depending on your testing framework, you can integrate reporting tools to generate detailed test reports.

Here's a simple Python example of how a test script using the Appium Python client library might look:

python

```
from appium import webdriver

# Set desired capabilities
```

```
desired_caps = {
    'platformName': 'Android',
    'deviceName': 'your_device_name',
    'appPackage': 'com.example.myapp',
    'appActivity': '.MainActivity'
}

# Initialize Appium driver
driver = webdriver.Remote('http://localhost:4723/wd/hub', desired_caps)

# Locate and interact with elements
driver.find_element_by_id("elementId").click()
driver.find_element_by_id("inputField").send_keys("Hello, Appium!")

# Perform assertions
assert driver.find_element_by_id("resultText").text == "Expected Result"

# Close the driver
driver.quit()
```

This is a basic outline, and the actual code structure will depend on the specific requirements and test scenarios of your mobile application. Be sure to refer to the documentation of your chosen programming language's Appium client library for detailed usage instructions and examples._

3.Answer: Automating 20 test cases for a mobile application using Selenium and WebDriverIO typically requires a test automation framework and access to the mobile devices or emulators/simulators. Below, I'll provide you with a general outline of how to automate these test cases using WebDriverIO in JavaScript. Please note that you'll need to set up the WebDriverIO environment and configure your test

project accordingly.

Assuming you are automating for Android devices, you can use the Appium driver in WebDriverIO for this task. Here's a high-level example:

javascript

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

// Define desired capabilities for Android
const androidCapabilities = {
    platformName: 'Android',
    deviceName: 'your_device_name',
    appPackage: 'com.example.myapp',
    appActivity: '.MainActivity',
    automationName: 'UiAutomator2', // Use UiAutomator2 for Android
};

(async () => {
    const driver = await remote({
        capabilities: androidCapabilities,
        logLevel: 'error', // Set log level as needed
    });

    try {
        // Test Case 1: Verify that a new product can be created with valid data
        // Locate and interact with elements here using driver.
```

```
        // Assertions and test steps go here.

        // Test Case 2: Ensure that mandatory fields are required for product creation
        // ...

        // Continue automating the remaining test cases

    } catch (error) {
        console.error('Test failed:', error);
    } finally {
        await driver.deleteSession(); // Close the driver session when done
    }
})();
```

In this code:

- We import `remote` from WebDriverIO to create a WebDriverIO instance.

- Desired capabilities for Android are defined, including platform name, device name, app package, app activity, and automation name.

- Inside the async function, we create the WebDriverIO instance using `await remote()` with the specified capabilities.

- Test cases are automated inside the try block. You'll locate and interact with elements, perform assertions, and implement test steps for each case.

- Error handling is included to catch and log any failures.

- Finally, we delete the WebDriverIO session to close the driver after all test cases are completed.

You'll need to adapt this code for your specific mobile application, providing the correct element locators, test steps, and assertions for each test case. Additionally, ensure you have Appium and Appium server running with the appropriate setup for your Android device or emulator.

4.Answer: Generating an HTML report for test automation results typically involves using a test framework or reporting library. I'll provide you with a generic example of how to generate an HTML report using JavaScript. You can adapt this code to your specific project and test results.

You can use libraries like "mochawesome" or "jasmine-html-reporter" in combination with a test framework like Mocha or Jasmine for this purpose. Below is an example using "mochawesome" with Mocha for test reporting:

1. First, make sure you have Mocha and "mochawesome" installed in your project. You can install them via npm:

bash

```
npm install mocha mochawesome
```

2. Create your test script, e.g., `test.js`, and write your test cases using Mocha.

javascript

```
const assert = require('assert');

describe('Example Test Suite', function () {
  it('should pass this test', function () {
    assert.strictEqual(1 + 1, 2);
  });

  it('should fail this test', function () {
```

```
        assert.strictEqual(2 * 2, 5); // Intentional failure
    });
});
```

3. Run your tests using Mocha with "mochawesome" reporter enabled:

bash

```
mocha test.js --reporter mochawesome
```

4. After running the tests, an HTML report file named `mochawesome-report/mochawesome.html` should be generated in your project directory.

We can open this HTML report in a web browser to view the test results.