**FreeNow Demo Mobile Testing**

Test Scenarios and Test Cases with pretty and stable Android UI Tests with Espresso for the given android source code / app.

|  |
| --- |
| **Source Code:** <https://github.com/freenowtech/MobileAppAutomationTest/tree/master/AndroidDemo>  **APK File:** <https://drive.google.com/open?id=1kAAufGX0c_uQm5aHsgIlhiPQ0_o3yrYG> |

**Task 1:** **Login case, the up‑to‑date credential (username-“crazydog335” & password” “venture”)**

**Test Analysis**: Prior to start writing test cases for a login page, firstly you need to think about, how many maximum controls can be available on a login page?

There can be a username, password, ‘Log In’ button, Cancel Button, and Forgot Password link. There can be one more control which is a checkbox named ‘Remember me’ to remember the login details on a particular machine.

In our case, we have total three controls. username, password and ‘Log In’ button.

**Test Cases Login Page.**

Following is the possible list of functional and non-functional test cases for a login page:

**However we are wring Test classes with espresso for first 3 test cases.**

Functional Test Cases:

| **Sr. No.** | **Functional Test Cases** | **Type- Negative/ Positive Test Case** |
| --- | --- | --- |
| 1 | Verify if a user will be able to login with a valid username and valid password. | Positive |
| 2 | Verify if a user cannot login with a valid username and an invalid password. | Negative |
| 3 | Verify the login page for both, when the field is blank and Submit button is clicked. | Negative |
| 4 | Verify the messages for invalid login. | Positive |
| 5 | Verify if the data in password field is either visible as asterisk or bullet signs. | Positive |
| 6 | Verify if a user is able to login with a new password only after he/she has changed the password. | Positive |
| 7 | Verify if the login page allows to log in simultaneously with different credentials in a different Device. | Positive |
| 8 | Verify if the ‘Enter’ key of the keyboard is working correctly on the login page. | Positive |
|  | Other Test Cases | |
| 9 | Verify the time taken to log in with a valid username and password. | Performance & Positive Testing |
| 10 | Verify if the font, text color, and color coding of the Login page is as per the standard. | UI Testing & Positive Testing |
| 11 | Verify if there is a ‘Login’ button available to erase the entered text. | Usability Testing |
| 12 | Verify the login page and all its controls in different Devices. | Browser Compatibility & Positive Testing. |

Non Functional Security Test Cases:

| **Sr. No.** | **Functional Test Cases** | **Type- Negative/ Positive Test Case** |
| --- | --- | --- |
| 1 | Verify if a user cannot enter the characters more than the specified range in each field (Username and Password). | Negative |
| 2 | Verify if a user cannot enter the characters more than the specified range in each field (Username and Password). | Positive |
| 3 | Verify the login page by pressing ‘Back button’ of the browser. It should not allow you to enter into the system once you log out. | Negative |
| 4 | Verify the timeout functionality of the login session. | Positive |
| 5 | Verify if a user should not be allowed to log in with different credentials from the same browser at the same time. | Negative |

We can write our test classes on more granular way to test each and every control in detail as form basic to complex test sets below:

|  |
| --- |
| **Test cases/scenarios for Text boxes in our case for Username and Password controls:**   1. Check for the alpha characters or alpha numeric characters 2. Check for alphabet is accepted in upper and lower case both 3. Check the mandatory condition for the textbox if given 4. Check the textbox is blank or not by default 5. Check Max-Min character limit of text box 6. Check for numeric characters only 7. Check with the requirement whether it will accept numeric and special characters 8. Check by entering “Spaces” in the prefix and suffix of the entered character-Trimming of space 9. Check by entering “single quotes and double quotes” 10. Check Copy and Paste texts from word or notepad 11. Check the height and alignment of text boxes are same throughout the site 12. Check text box except only spaces as input |

**Conclusion:**

While writing test cases for login page write the test cases for all the fields. There should be a combination of both positive and negative test cases. Try to cover the performance, security, and functional scenarios.

The login page is the page with fewer controls, so even though it is looking simple for testing, it should not be considered as an easy task.

Also many a time it is the first impression of an application, so it should be perfect for user interface and usability.

**Source Code for Test cases for first 3 Test Scenarios for Login Activity:**

1. Verify if a user will be able to login with a valid username and valid password.

|  |
| --- |
| @Test **public void** testSuccessfulLOGIN() **throws** IOException,NullPointerException {  AuthenticationActivity act=**new** AuthenticationActivity();  **mEditTextUsername** = **"crazydog335"**;  **mEditTextPassword** = **"venture"**;   *// Input Correct Username as 'crazydog335'  onView*(*withId*((R.id.***edt\_username***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextUsername**));  *// Input Correct Password as 'venture'  onView*(*withId*((R.id.***edt\_password***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextPassword**));  *// Click Login Button  onView*(*withId*(R.id.***btn\_login***)).perform(*click*());  *// Assert if User Logged in.  assertTrue*(**mAuthenticationTestRule**.getActivity().isFinishing());  *intended*(*hasComponent*(MainActivity.**class**.getName()));  Espresso.*onView*(ViewMatchers.*withId*(R.id.***textSearch***))  .check(ViewAssertions.*matches*((*isDisplayed*())));   } |

1. Verify if a user cannot login with a valid username and an invalid password.

|  |
| --- |
| @Test **public void** testLOGIN\_InvalidPassword() **throws** IOException,NullPointerException {  AuthenticationActivity act=**new** AuthenticationActivity();  **mEditTextUsername** = **"crazydog335"**;  **mEditTextPassword** = **"venturee"**;  *// Input Correct Username as 'crazydog335'  onView*(*withId*((R.id.***edt\_username***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextUsername**));  *// Input Invalid Password as 'venturee'  onView*(*withId*((R.id.***edt\_password***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextPassword**));  *// Click Login Button  onView*(*withId*(R.id.***btn\_login***)).perform(*click*());  *// Assert User not Logged in.  assertFalse*(**mAuthenticationTestRule**.getActivity().isFinishing());  *intended*(*not*(*hasComponent*(MainActivity.**class**.getName())));  Espresso.*onView*(ViewMatchers.*withId*(R.id.***textSearch***))  .check(ViewAssertions.*matches*(*not*(*isDisplayed*()))); } |

1. Verify the login page for both, when the field is blank and Submit button is clicked.

|  |
| --- |
| @Test **public void** testLOGIN\_BlankUserPassword() **throws** IOException,NullPointerException{  AuthenticationActivity act=**new** AuthenticationActivity();  **mEditTextUsername** = **""**;  **mEditTextPassword** = **""**;  *// Input Correct Username as 'crazydog335'  onView*(*withId*((R.id.***edt\_username***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextUsername**));  *// Input Invalid Password as 'venturee'  onView*(*withId*((R.id.***edt\_password***))).perform(ViewActions.*clearText*()).perform(*typeText*(**mEditTextPassword**));  *// Click Login Button  onView*(*withId*(R.id.***btn\_login***)).perform(*click*());  *// Assert User not Logged in  assertFalse*(**mAuthenticationTestRule**.getActivity().isFinishing());  *intended*(*not*(*hasComponent*(MainActivity.**class**.getName())));  Espresso.*onView*(ViewMatchers.*withId*(R.id.***textSearch***))  .check(ViewAssertions.*matches*(*not*(*isDisplayed*())));  } } |

Test Class file:

|  |
| --- |
| Github link: <https://github.com/deejha017/FreeNowDemoTes/blob/master/AuthenticationActivityTest.java> |

**Task 2:** **Login Search for “sa”, select the 2nd result (via the name, not the index) from the list, then click the call button.**

**Test Analysis**: Search box is the common and most essential field of any mobile application. In Our case we need to search for drivers in the search bar by typing “sa” and selecting the 2nd search result by name.

Here are some scenarios which may help to test the search box field.

| **Sr. No.** | **Functional Test Cases** | **Type- Negative/ Positive Test Case** |
| --- | --- | --- |
| 1 | Verify that user is able to enter alphabetic, numeric and special characters in Search field. | Positive |
| 2 | The text in the input field **disappears** when you enter the first character, and **appears** again, if the input field is empty. | Positive |
| 3 | Search results displayed should be relevant to search keyword | Positive |
| 4 | Application should not crash if user inserted % in search field | Negative |
| 5 | When user start typing word in text box it should suggest words that matches typed keyword | Positive |
| 6 | Validate search rules defined for “Exact Match” with the search key word | Positive |
| 7 | Validate search rules defined for “Similar Match” with the search key word | Positive |
| 8 | Validate search rules defined to search with a set of keywords | Positive |

In our task, we have 3 sub tasks as below:

1. Type “sr” in the search text field and check if results are displaying. The test code would be as below.

|  |
| --- |
| *//Type "sr" in the search field* Espresso.*onView*(*withId*(R.id.***textSearch***)).perform(*typeText*(**"sr"**));  *//Check if the results are available;* Espresso.*onView*(*with Text*(**"Sara Christensen"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .check(*matches*(*isDisplayed*())); Espresso.*onView*(*withText*(**"Sarah Scott"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .check(*matches*(*isDisplayed*())); |

1. Select 2nd result by name. The test code would be as below.

|  |
| --- |
| *//Select 2nd driver from the search result by name* Espresso.*onView*(*withText*(**"Sarah Scott"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .perform(*click*());  *//assert*  *intended*(*hasComponent*(DriverProfileActivity.**class**.getName())); Espresso.*onView*(*withId*(R.id.***textViewDriverName***))  .check(*matches*(*withText*(**"Sarah Scott"**))); |

1. Press the call button. The test code would be as below.

|  |
| --- |
| *//Click on call button* **id** = R.layout.***content\_driver\_profile***; Espresso.*onView*(*withId*(R.id.***fab***)).perform(*click*());  *// assert  assertTrue*(**mDriverProfileActivityTestRule**.getActivity().isFinishing()); |

The complete test class will look like:

|  |
| --- |
| **package** com.freenow.android\_demo.activities;  **import** android.content.res.Resources; **import** android.support.test.espresso.Espresso; **import** android.support.test.rule.ActivityTestRule; **import** com.freenow.android\_demo.R; **import** org.junit.After; **import** org.junit.Before; **import** org.junit.Rule; **import** org.junit.Test; **import static** android.support.test.espresso.Espresso.*onView*; **import static** android.support.test.espresso.action.ViewActions.*click*; **import static** android.support.test.espresso.action.ViewActions.*typeText*; **import static** android.support.test.espresso.assertion.ViewAssertions.*matches*; **import static** android.support.test.espresso.matcher.RootMatchers.*withDecorView*; **import static** android.support.test.espresso.matcher.ViewMatchers.*isDisplayed*; **import static** android.support.test.espresso.matcher.ViewMatchers.*withId*; **import static** android.support.test.espresso.matcher.ViewMatchers.*withText*; **import static** org.hamcrest.CoreMatchers.*is*; **import static** org.hamcrest.CoreMatchers.*not*; **import static** org.junit.Assert.\*;  **public class** DriverProfileActivityTest {   **private** DriverProfileActivity **mActivity** = **null**;  **private** String **mEditTextUsername**;  **private** String **mEditTextPassword**;  @Rule  **public** ActivityTestRule<DriverProfileActivity> **mDriverProfileActivityTestRule**=**new** ActivityTestRule<DriverProfileActivity>(DriverProfileActivity.**class**);  **private int id**;  @Before  **private void** setUp() **throws** Exception {  **mActivity** = **mDriverProfileActivityTestRule**.getActivity();  }   @After  **public void** tearDown() **throws** Exception {  }  @Test  **public void** searchDriver() **throws** IOException{  *//Type "sr" in the search field* Espresso.*onView*(*withId*(R.id.***textSearch***)).perform(*typeText*(**"sr"**));  *//Check if the results are available;* Espresso.*onView*(*withText*(**"Sara Christensen"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .check(*matches*(*isDisplayed*()));  Espresso.*onView*(*withText*(**"Sarah Scott"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .check(*matches*(*isDisplayed*()));  *//Select 2nd driver from the search result by name* Espresso.*onView*(*withText*(**"Sarah Scott"**))  .inRoot(*withDecorView*(*not*(*is*(**mActivity**.getWindow().getDecorView()))))  .perform(*click*());  *//assert* Espresso.*onView*(*withId*(R.id.***textViewDriverName***))  .check(*matches*(*withText*(**"Sarah Scott"**)));  *//Click on call button* **id** = R.layout.***content\_driver\_profile***;  Espresso.*onView*(*withId*(R.id.***fab***)).perform(*click*());  *// assert  assertTrue*(**mDriverProfileActivityTestRule**.getActivity().isFinishing());    } } |

Test Class file:

|  |
| --- |
| Github Link: <https://github.com/deejha017/FreeNowDemoTes/blob/master/DriverProfileActivityTest.java> |

**Task 3:** **Login Deploy the tests on CircleCI, and send us the link to the CircleCI builds overview page.**

It was really a great learning experience for me. I was completely new to all the terms like Espresso, Custom Configuration files, CircleCi, Builds, Workflows. When first I went through all this, I thought it is going to be just hardcore testing thing. But, later I explored many things.

One of the toughest tasks was to build and deploy on circleCI. Earlier I had no idea what exactly it’s all about. Slowly and gradually after researching on it and watching more than 20 videos, I figured it out that it’s all about just configuring the .yml file.

Still, it was not easy for me as integration was not proper. After researching more and applying fixes in each run, I did it. And trust me it was an amazing feeling when I saw the line “Build is successful”. And it happened at 54th run of the build job.

Though, I finished still I would say it might not be the exact and proper build in CircleCI and I don’t have any previous experiences. So being a tester, I focused on writing my Test classes with more readable test code and analyzing the source code to write all possible test scenarios that can help to assure the best quality of the application under observation.

Below is the link to CircleCI overview page.

|  |
| --- |
| CircleCI Link: <https://circleci.com/gh/deejha017/FreeNowTestClasses> |

**Summary:**

1. **For Task 1-** Created one test class name as “AuthenticationActivityTest.java” that consists total three test methods (testSuccessfulLOGIN(), testLOGIN\_InvalidPassword(), testLOGIN\_BlankUserPassword()) written for three different scenarios.

|  |
| --- |
| **Test class can be found at below location.**  <https://github.com/deejha017/FreeNowTestClasses/tree/master/MobileAppAutomationTest-master/AndroidDemo/app/src/androidTest/java/com/freenow/android_demo/activities> |

All other possible scenarios are also mention in the document above. Please refer.

1. **For Task 2-** Created one test class *“DriverProFileActivityTest.java”* that consists one test method as *“searchDriver()”* to test the given task activity.

|  |
| --- |
| **Test class can be found at below location.**  <https://github.com/deejha017/FreeNowTestClasses/tree/master/MobileAppAutomationTest-master/AndroidDemo/app/src/androidTest/java/com/freenow/android_demo/activities> |

All other possible scenarios are also mention in the document above. Please refer.

1. For Task 3- This one made my head vibrate like anything. I was feeling like I am on a mission and the mission was getting more serious every time I executed the build job and it got failed. However, the mission accomplished. Please find below the links for Github and CircleCI page.

Github:

<https://github.com/deejha017/FreeNowTestClasses>

CircleCI:

<https://circleci.com/gh/deejha017/FreeNowTestClasses>

In case the CircleCI link doesn’t open please use Login with Github using below access:

**User: deepak017**

**Password: D@novartis001.**