UI Testing with Espresso

Use Matchers Effectively: Espresso offers a range of matchers for view interactions. Understanding and effectively using these matchers (like `withId`, `withText`, `isDisplayed`) can make your tests more robust.

For Composables

Each test uses the `composeTestRule` to set the Compose content and perform actions or assertions on the nodes.

- The tests rely on functions like `onNodeWithText`, `performClick`, `performTextInput`, and `performScrollTo` to interact with Composable elements in a way that mimics user behavior.
- Assertions like `assertIsDisplayed` are used to verify that the UI elements are in the expected state after the interactions.

Exercise 1: Testing Text Display

Prompt: Write an Espresso test to verify that a TextView with the ID `R.id.text_view` in an activity displays the text "Hello, Espresso!".

Expected Outcome: The test should pass if the TextView correctly displays "Hello, Espresso!". It fails if the text is different or the TextView is not found.

Answer

```
@Test
fun testTextViewDisplaysCorrectText() {
    onView(withId(R.id.text_view))
        .check(matches(withText("Hello, Espresso!")))
}
```

Exercise 2: Testing Button Click Behavior

Prompt: Write an Espresso test for an activity with a Button (`R.id.button_click`) that, when clicked, changes the text of a TextView (`R.id.text_result`) to "Button clicked".

Expected Outcome: The test should pass if the TextView's text changes to "Button clicked" after the button is clicked.

Answer

```
@Test
fun testButtonClickUpdatesTextView() {
    onView(withId(R.id.button_click))
        .perform(click())

    onView(withId(R.id.text_result))
        .check(matches(withText("Button clicked")))
}
```

Exercise 3: Verifying Text Display

Prompt: Write an Espresso test for a Composable function that test a `Text` with the string "Welcome to Compose"

Expected Outcome: The test should pass if the `Text` composable correctly displays "Welcome to Compose"

Composable Sample.

```
@Composable
fun WelcomeTextComposable() {
    Text("Welcome to Compose")
}
```

Answer

```
@Test
fun testTextDisplaysCorrectly() {
    composeTestRule.setContent {
        Text("Welcome to Compose")
    }

    composeTestRule.onNodeWithText("Welcome to
Compose").assertIsDisplayed()
}
```

Exercise 4: Testing Button Click

Prompt: Create a test with a `Button` and a `Text`. Initially, the `Text` should display "Not Clicked". When the button is clicked, the text changes to "Clicked".

Expected Outcome: The test passes if the text changes to "Clicked" after the button is clicked.

Composable Sample

```
@Composable
fun ClickableTextComposable() {
    var text by remember { mutableStateOf("Not Clicked") }
    Column {
        Button(onClick = { text = "Clicked" }) {
            Text("Click Me")
        }
        Text(text)
    }
}
```

Answer

```
@Test
fun testButtonClickUpdatesText() {
    composeTestRule.setContent {
        var text by remember { mutableStateOf("Not Clicked") }
        Button(onClick = { text = "Clicked" }) {
            Text("Click Me")
        }
        Text(text)
    }

    composeTestRule.onNodeWithText("Click Me").performClick()
    composeTestRule.onNodeWithText("Clicked").assertIsDisplayed()
}
```

Exercise 5: Testing TextField Input

Prompt: Write a test for a Composable containing a `TextField` and a `Text`. The `Text` should display whatever is typed into the `TextField`.

Expected Outcome: The test should pass if the text entered in the `TextField` is correctly displayed in the `Text` composable

Composable Sample

```
@Composable
fun InputFieldComposable() {
    var text by remember { mutableStateOf("") }
    Column {
        TextField(
            value = text,
            onValueChange = { text = it },
            label = { Text("Enter text") }
        )
        Text("You entered: $text")
    }
}
```

Answer

```
@Test
fun testTextFieldInputReflectsInText() {
    composeTestRule.setContent {
        var text by remember { mutableStateOf("") }
        Column {
            TextField(value = text, onValueChange = { text = it })
            Text(text)
        }
    }

    val inputText = "Hello Compose"

composeTestRule.onNode(hasSetTextAction()).performTextInput(inputT
ext)
    composeTestRule.onNodeWithText(inputText).assertIsDisplayed()
}
```

Exercise 6: Testing LazyColumn Scroll

Prompt: Test a Composable with a `LazyColumn` containing a list of text items. Verify that a specific item is visible after scrolling.

Expected Outcome: The test passes if "Item 50" is visible after scrolling the `LazyColumn`.
https://developer.android.com/training/testing/espresso/lists#recycler-view-list-items
https://developer.android.com/jetpack/compose/testing

Composable Sample

```
@Composable
fun LazyColumnComposable() {
    LazyColumn {
        items(100) { index ->
            Text("Item $index")
        }
    }
}
```

Answer

```
@Test
fun testLazyColumnScrolling() {
    composeTestRule.setContent {
        LazyColumn {
            items(100) { index ->
                Text("Item $index")
            }
        }
    }

    composeTestRule.onNodeWithText("Item
50").performScrollTo()
    composeTestRule.onNodeWithText("Item
50").assertIsDisplayed()
}
```

Exercise 7: Testing Toggle State

Prompt: Create a test for a Composable with a `Switch` and a `Text`. The `Text` should display "ON" when the switch is on and "OFF" when it is off.

Expected Outcome: The test should pass if the text changes to "ON" when the switch is toggled.

Composable Sample

```
@Composable
fun ToggleSwitchComposable() {
    var isChecked by remember { mutableStateOf(false) }
    Column {
        Switch(
            checked = isChecked,
            onCheckedChange = { isChecked = it }
        )
        Text(if (isChecked) "ON" else "OFF")
    }
}
```

Answer

```
@Test
fun testSwitchToggleChangesText() {
    composeTestRule.setContent {
        var isChecked by remember { mutableStateOf(false) }
        Column {
            Switch(checked = isChecked, onCheckedChange = {
isChecked = it })
            Text(if (isChecked) "ON" else "OFF")
        }
    }


composeTestRule.onNode(hasContentDescription("Switch")).perfo
rmClick()
    composeTestRule.onNodeWithText("ON").assertIsDisplayed()
}
```