ESPRESSO TESTING

LISTS, WEB, MULTIPROCESS, UI TESTING

ESPRESSO LISTS

- Espresso offers mechanisms to scroll to or act on a particular item for two types of lists: adapter views and recycler views.
- When dealing with lists, especially those created with a RecyclerView or an AdapterView object, the view that you're interested in might not even be on the screen because only a small number of children are displayed and are recycled as you scroll. The scrollTo() method can't be used in this case because it requires an existing view.

ADAPTER VIEWS

- AdapterView is a special kind of view specifically designed to render a collection of similar information like product list and user contacts fetched from an underlying data source using Adapter.
- The data source may be simple list to complex database entries. Some of the view derived from AdapterView are ListView, GridView and Spinner.
- AdapterView renders the user interface dynamically depending on the amount of data available in the underlying data source. In addition, AdapterView renders only the minimum necessary data, which can be rendered in the available visible area of the screen.
- AdapterView does this to conserve memory and to make the user interface look smooth even if the underlying data is large.

ADAPTER VIEW

- Upon analysis, the nature of the AdapterView architecture makes the onView option and its view matchers irrelevant because the particular view to be tested may not be rendered at all in the first place.
- Luckily, espresso provides a method, onData(), which accepts hamcrest matchers (relevant to the data type of the underlying data) to match the underlying data and returns object of type DataInteraction corresponding to the view o the matched data.
  - onData(allOf(is(instanceOf(String.class)), startsWith("Apple"))).perform(click())
- Here, onData() matches entry “Apple”, if it is available in the underlying data (array list) and returns DataInteraction object to interact with the matched view (TextView corresponding to “Apple” entry).

METHODS

DataInteraction provides the below methods to interact with the view,

- perform()
  - This accepts view actions and fires the passed in view actions.
  - onData(allOf(is(instanceOf(String.class)), startsWith("Apple"))).perform(click())
- check()
  - This accepts view assertions and checks the passed in view assertions.
  - onData(allOf(is(instanceOf(String.class)), startsWith("Apple")))
    .check(matches(withText("Apple")))

METHODS

- inAdapterView()
  - This accepts view matchers. It selects the particular AdapterView based on the passed in view matchers and returns DataInteraction object to interact with the matched AdapterView
  - onData(allOf())

    .inAdapterView(withId(R.id.adapter_view))

    .atPosition(5)

    .perform(click())

METHODS

- atPosition()
- This accepts an argument of type integer and refers the position of the item in the underlying data. It selects the view corresponding to the passed in positional value of the data and returns DataInteraction object to interact with the matched view. It will be useful, if we know the correct order of the underlying data.

```
onData(allOf())
    .inAdapterView(withId(R.id.adapter_view))
    .atPosition(5)
    .perform(click())
```

METHODS

- onChildView()
- This accepts view matchers and matches the view inside the specific child view. For example, we can interact with specific items like Buy button in a product list based AdapterView.

```
onData(allOf(is(instanceOf(String.class)), startsWith("Apple")))
  .onChildView(withId(R.id.buy_button))
  .perform(click())
```

dependencies

- testImplementation 'junit:junit:4.12'
- androidTestImplementation 'androidx.test:runner:1.1.1'
- androidTestImplementation 'androidx.test:rules:1.1.1'
- androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'

RecyclerView

- RecyclerView objects work differently than AdapterView objects, so onData() cannot be used to interact with them.
- To interact with RecyclerViews using Espresso, you can use the espresso-contrib package, which has a collection of RecyclerViewActions that can be used to scroll to positions or to perform actions on items:
  - scrollTo() - Scrolls to the matched View, if it exists.
  - scrollToHolder() - Scrolls to the matched View Holder, if it exists.
  - scrollToPosition() - Scrolls to a specific position.
  - actionOnHolderItem() - Performs a View Action on a matched View Holder.
  - actionOnItem() - Performs a View Action on a matched View.
  - actionOnItemAtPosition() - Performs a ViewAction on a view at a specific position.

WEBVIEW

- WebView is a special view provided by android to display web pages inside the application. WebView does not provide all the features of a full-fledged browser application like chrome and firefox.
- However, it provides complete control over the content to be shown and exposes all the android features to be invoked inside the web pages. It enables WebView and provides a special environment where the UI can be easily designed using HTML technology and native features like camera and dial a contact.
- This feature set enables a WebView to provide a new kind of application called Hybrid application, where the UI is done in HTML and business logic is done in either JavaScript or through an external API endpoint.

webview

- Normally, testing a WebView needs to be a challenge because it uses HTML technology for its user interface elements rather than native user interface/views. Espresso excels in this area by providing a new set to web matchers and web assertions, which is intentionally similar to native view matchers and view assertions. At the same time, it provides a wellbalanced approach by including a web technology based testing environment as well.
- Espresso web is built upon WebDriver Atom framework, which is used to find and manipulate web elements. Atom is similar to view actions. Atom will do all the interaction inside a web page. WebDriver exposes a predefined set of methods, like findElement(), getElement() to find web elements and returns the corresponding atoms (to do action in the web page).

- A standard web testing statement looks like the below code,
- onWebView()

  .withElement(Atom)

  .perform(Atom)

  .check(WebAssertion)

Here,

onWebView() – Similar to onView(), it exposes a set of API to test a WebView.

withElement() – One of the several methods used to locate web elements inside a web page using Atom and returns WebInteration object, which is similar to ViewInteraction.

perform() – Executes the action inside a web page using Atom and returns WebInteraction.

check() – This does the necessary assertion using WebAssertion.

- A sample web testing code is as follows,

  - onWebView()
    .withElement(findElement(Locator.ID, "apple"))
    .check(webMatches(getText(), containsString("Apple")))

Here,

- findElement() locate a element and returns an Atom
- webMatches is similar to matches method