Android project: Wikipedia Functional GUI Testing Using Android Espresso Framework

1. Project Goal

Implement scalable Automated Functional Test code for Android Project Using Android Espresso Framework.

2. Notable Features of Implementation

2.1 Define Wrapper Class of Espresso Framework for Better Implementation and Faster Development

With default auto generated code form Espresso Framework we need to insert Thread Sleep code after each action. This bottleneck for a development as we need to manually insert Thread Sleep code after each action and it is not possible to maintain configurable sleep time. In my implementation, I have wrapped following classes:

```
android.support.test.espresso.action.ViewActions
android.support.test.espresso.Espresso
```

Inside wrapper class I have inserted Sleep code as follows:

```
public static ViewAction click() {
SleepUtil.sleep(Config.delayTime);
return android.support.test.espresso.action.ViewActions.click();

}
```

For using wrapper action class we just need to change include statement of the test class so that it uses warrper action classes rather than Espresso action class.

2.2 Maintain Test Class Stability by Cleaning Up Previous Application Data

After running each test case app generates some that like landing page or history and this states can make instability of test outcome if we run single and multiple test cases several times. To handle this issue I am cleaning app data before running each test case as follows:

```
@Before
public void setUp() {
    CleanUpUtil.cleanUpData();
    mActivityTestRule.launchActivity(new Intent());
}


public static void cleanUpData()
{
    File root =
InstrumentationRegistry.getTargetContext().getFilesDir().getParentFile();
    String[] sharedPreferencesFileNames = new File(root,
"shared_prefs").list();
    for (String fileName : sharedPreferencesFileNames) {

InstrumentationRegistry.getTargetContext().getSharedPreferences(fileName.repl
ace(".xml", ""), Context.MODE_PRIVATE).edit().clear().commit();
    }

    clearApplicationData();
}
```

2.3 Clear Up Idle Resources After Running Each Test Case

In each test case Espresso Instruments different application resources and after running each test case we need to clear up app idle resources for efficient test case execution. So after running each test case I cleaned up idle resources with following code:

```
@After
public void tearDown() {
    List<IdlingResource> idlingResourceList = getIdlingResources();
    if (idlingResourceList != null) {
        for (int i = 0; i < idlingResourceList.size(); i++) {
            unregisterIdlingResources(idlingResourceList.get(i));
        }
    }

}
```

2. Implemented Test Cases:

Following test cases are implemented for functional GUI testing:

1. Change Language from Setting(File: ChangeLanguageTest)
2. Change Language from Opened WikiPage(File: ChangeLanguageFromPageTest)

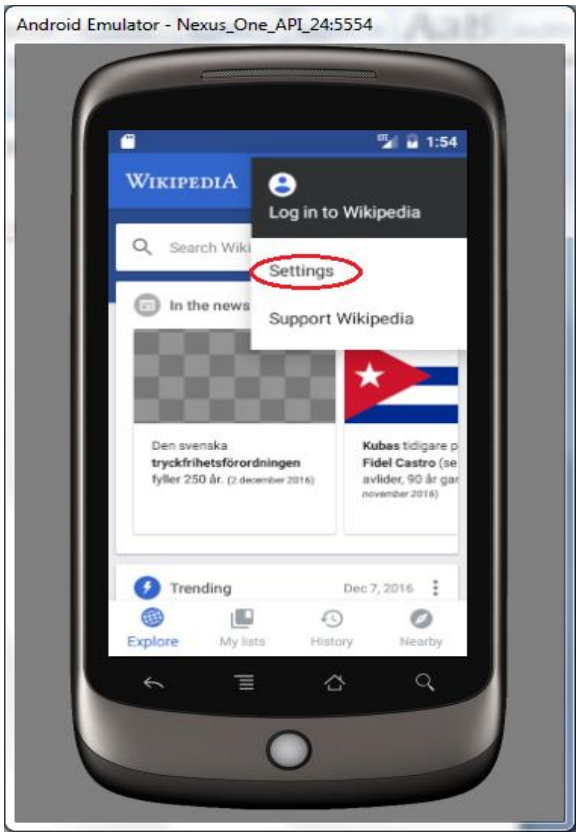
3. History Generation and Cleanup(File: HistoryTest)
4. Jump to a specific section in Wiki Page(File: JumpingSectionTest)
5. Log in and Log out as Wikipedia User(File: LogInTest)
6. Reading List Manage Test(File: ReadingListMngTest)
7. Read Wiki page and search in Wiki page(File: ReadSearchTest)

Details of each test case is following sub sections:

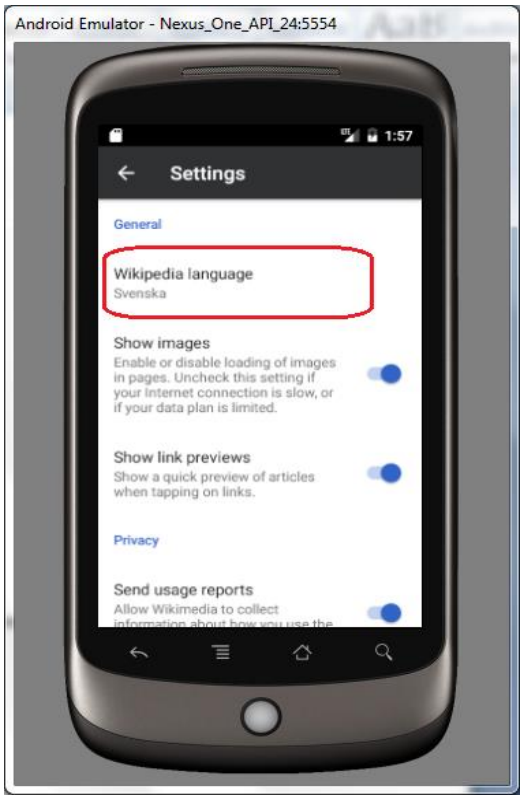
2.1 Change Language from Setting

Steps of the Test Case:

1. Click into App Settings Button:

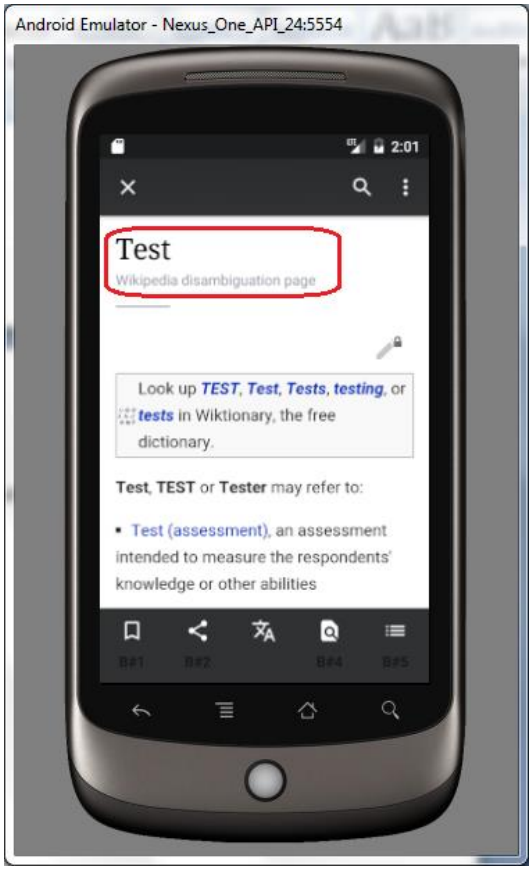


2. From Settings Window Change Language to "Svenska" for Swedish and Close Settings Window.



3. Search page with topic "Test" and Validate that for Test Wiki page header is written in Swedish language that is "Test Grensida".

4. Then though App Settings window change Language to "English" and for English Test Wiki page shows header in English language.

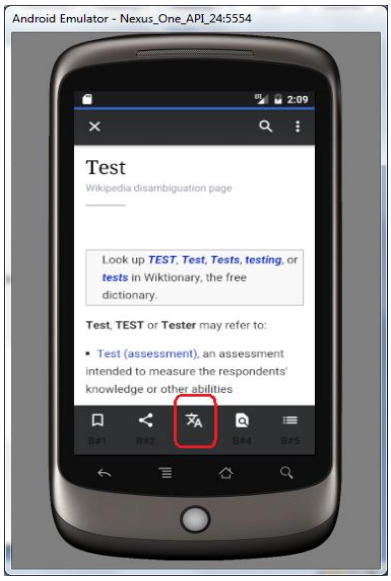


2.2 Change Language from Opened Wiki Page

Steps of the Test Case:

1. Search topic "test" and open the wiki page.

2. Through Wiki Page change button change language "Svenska" for Swedish.



2. Validate that topic header for Test is changed in Swedish language.

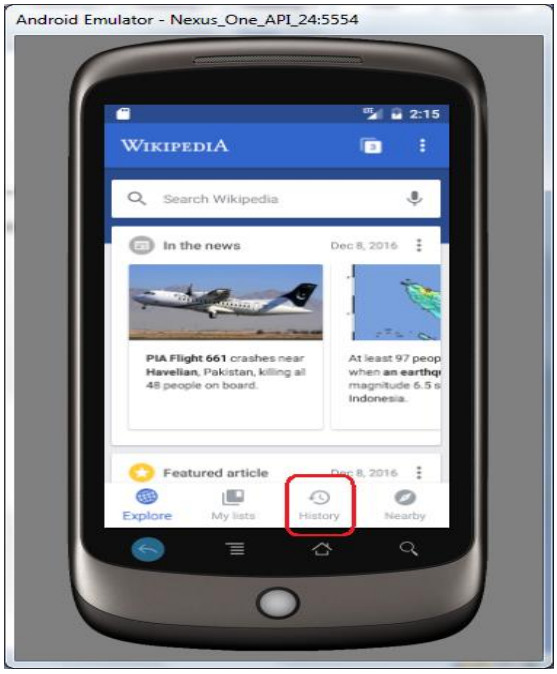
3. Similarly change language to English and validate that language in wiki page changed accordingly.

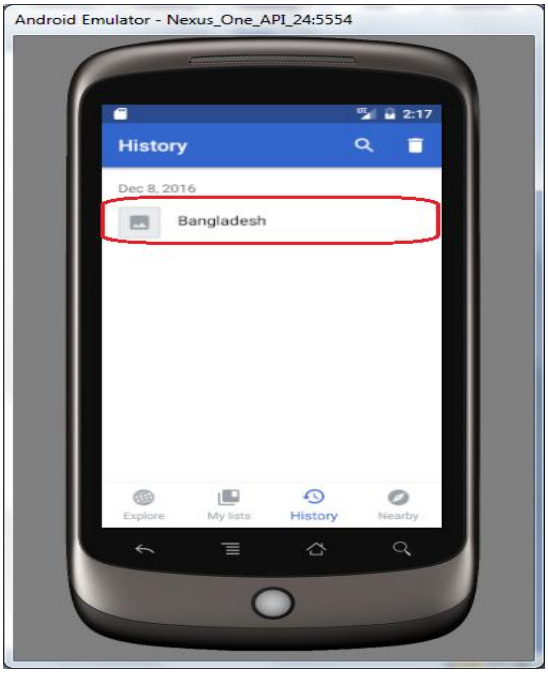
2.3 History Generation and Cleanup

Steps of the test case.

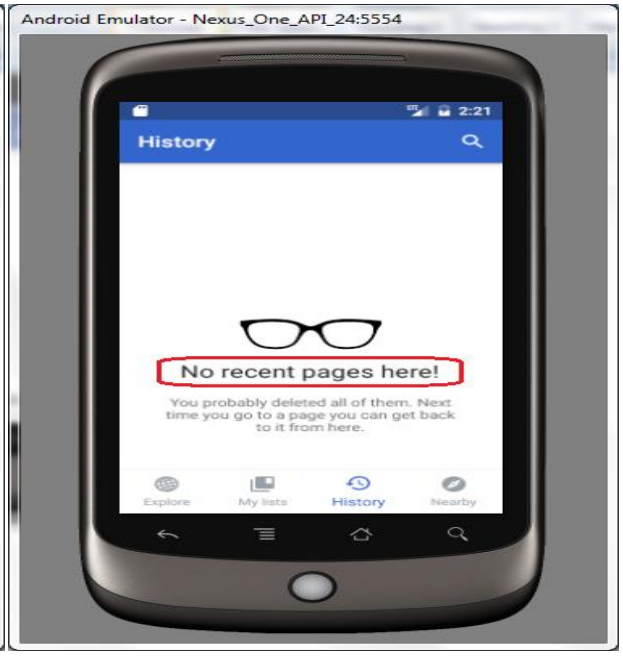
1. Launch application and search for topic "Bangladesh" and open the wiki page.

2. Now select "History" button and check that in History page it shows recent wiki page for "Bangladesh".





3. Now through Delete button clear History and validate that there is not recent history.

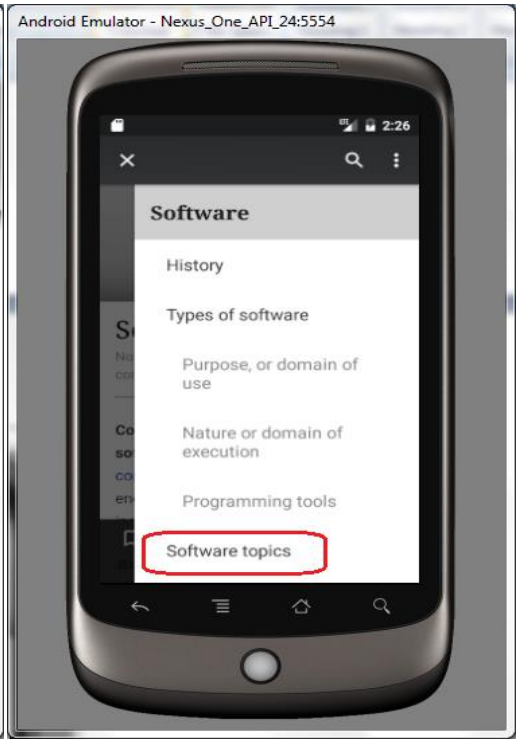


2.4 Jump to a specific section in Wiki Page

Steps of the test case

1. Launch application search for topic "Software" and open wiki page for "Software".

2. Through "Section Selection" button select section "Software Topics" .

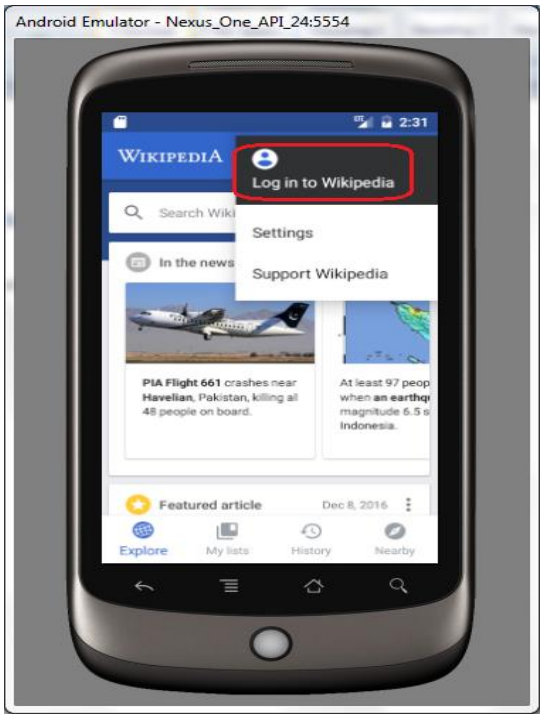


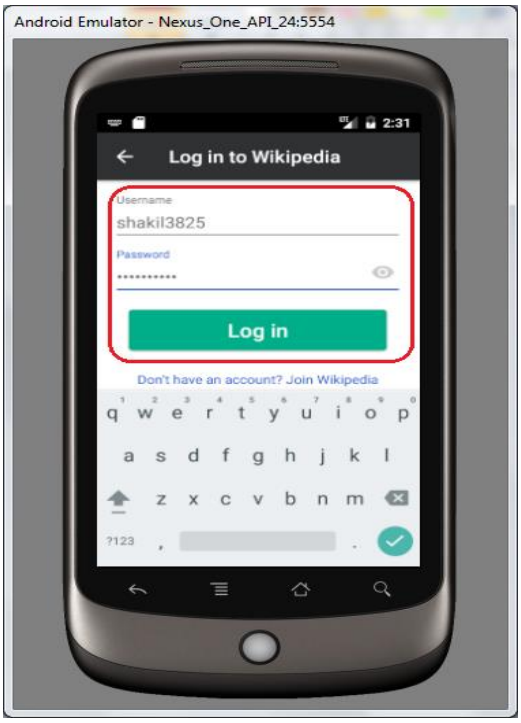
3. Validate the wiki page jumped to specific section.

2.5 Log in and Log out as Wikipedia User

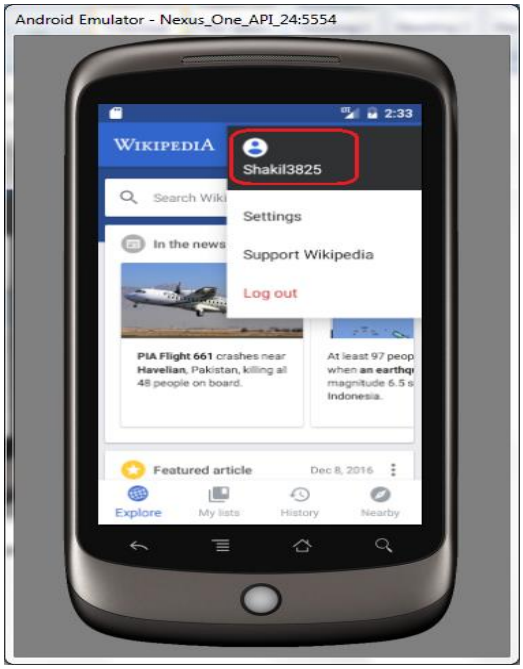
1. Launch application.

2. Though application Settings menu Login as Wiki User with valid username and password.





3. After login open Settings menu again and validate application user name.



4. Then Log Out from application.

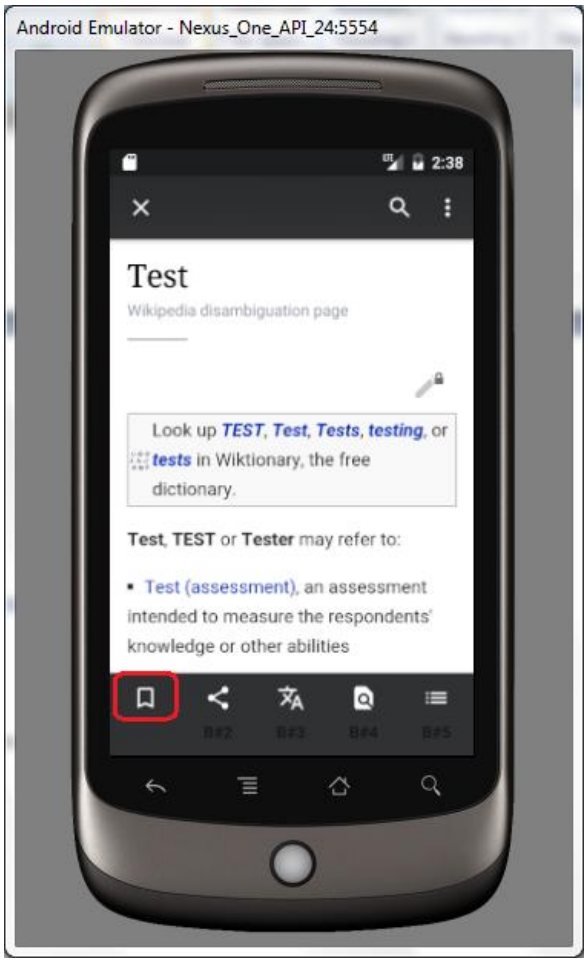
2.6 Reading List Manage Test

Steps of the test case:

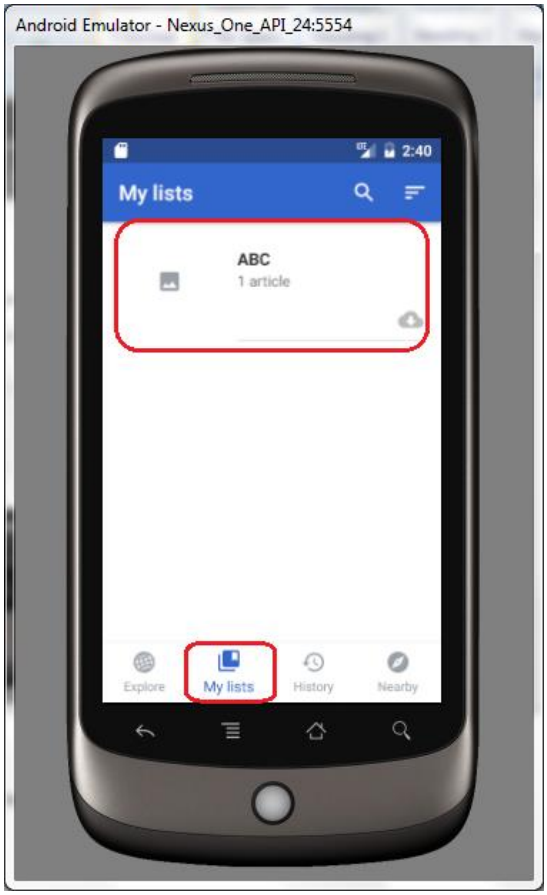
1. Launch application.

2. Search a topic "test" and open the corresponding wiki page.

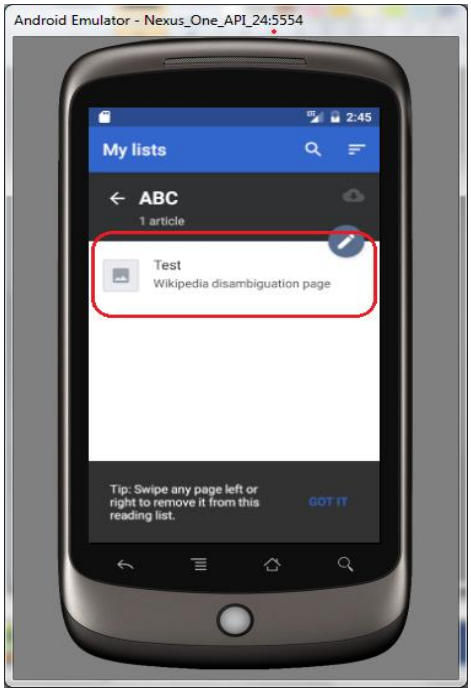
3. Add the page as Reading List ABC.

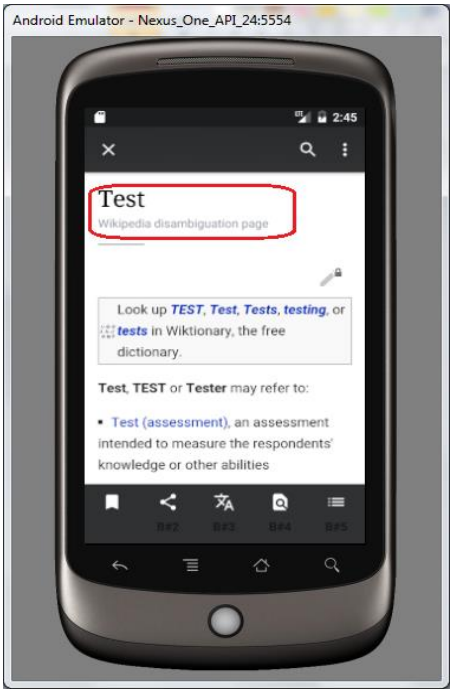


4. Now select "My lists" button and open the reading list page.

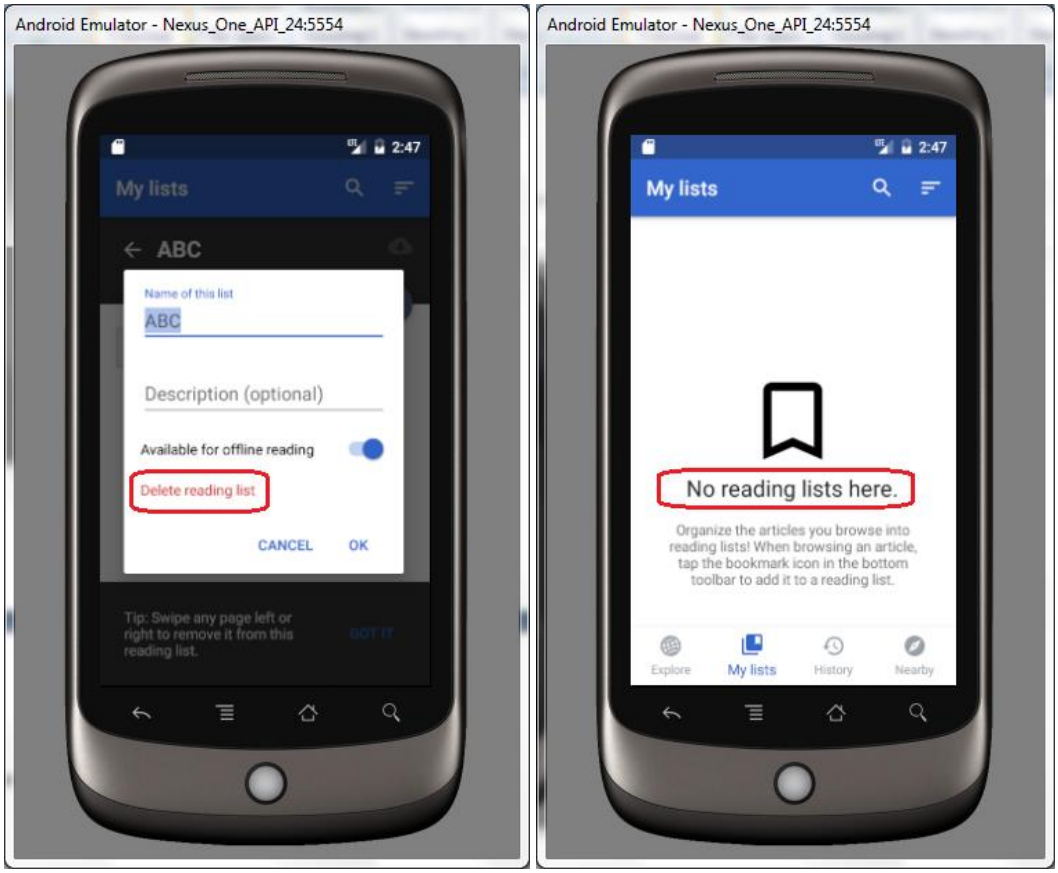


5. Expand the reading list item and select the reading list topic. Validate that proper page added as reading list item.





6. Delete the reading list and validate that there is no reading list item.

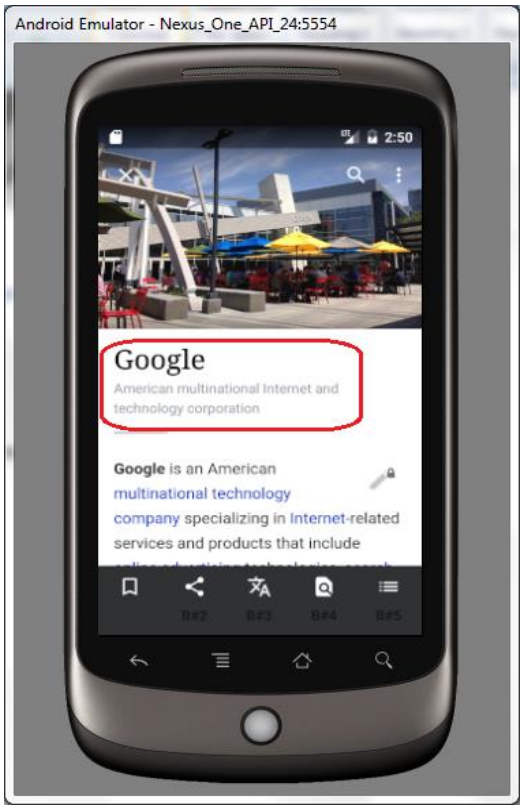


2.7 Read Wiki page and search in Wiki page

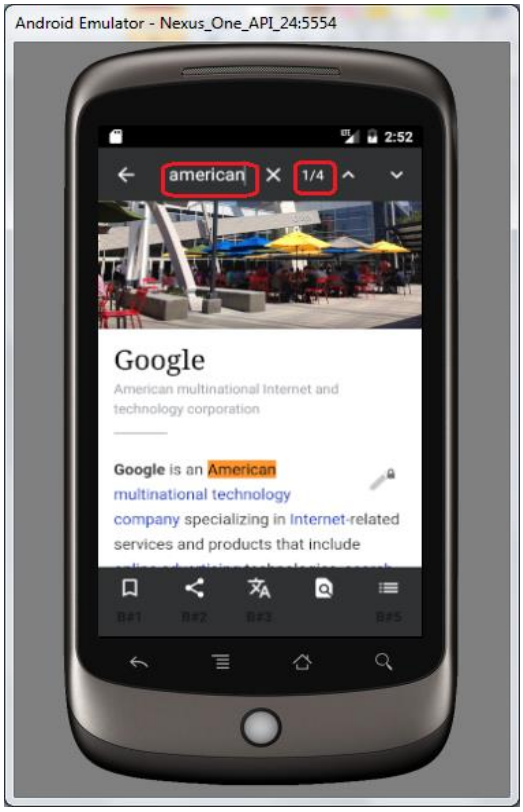
1. Launch application

2. Search with topic "Google"

3. In the wiki page validate header title.



4. Now in Wiki Page search with text "American" and validate that it finds the search text.



3. Challenge Faced During Test Automation

1. Espresso has limitation to generate proper test code. Needs manual debugging and solve the issue.
2. Espresso has some threading limitations and in some scenario is crashes for running large set of tests.
3. Difficulty with recognize GUI control with no ID.
4. After each test case change the app state in stable condition so that next test case can run smoothly.

~END~