**theScore Take-home Assignment**

Created by: Sean Xu

This assignment automated a scenario of theScore mobile application.

**Please navigate to the 4th page for the rationale of the test approach.**

**Scenario:**

1. Open a league, team, or player page of your choice (bonus points for using a data driven

or parameterized approach).

2. Verify that the expected page opens correctly.

3. Tap on a sub-tab of your choice, eg: league table / standings / leaders, or stats tab of the

league, team, or player.

4. Verify that you are on the correct tab and that the data is displayed correctly and

corresponds to the league, team, or player from step 1.

5. Verify that back navigation returns you to the previous page correctly.

**Cucumber Steps:** to be found in .src/test/java/features/theScoreDemo.feature

**Environment Requirements:**

1. IntelliJ IDEA

2. Cucumber and Gherkin plugin to be downloaded in IntelliJ IDEA

3. Appium desktop

4. Android Studio, Android Emulator with API version 29 or 30

5. Maven - version 3.5.4

- project pom.xml file is in root directory of the project

6. JAVA jdk1.8

7. theScore v21.2.0

**Environment Setup:**

1. Config Maven, Android Studio, Java environment path first

2. Import this project using IntelliJ IDEA as a Maven project, then set up the maven environment:

a. Maven path: to your maven folder

b. Maven user setting file: to your maven directory and which is in \apache-maven-3.5.4\conf\settings.xml

c. Set up your maven local repository

3. Install node.js and use command "npm install appium-doctor -g" to install appium-doctor, then "appium-doctor" to

check the environment

4. Launch your Android-Studio and download sdk. Create a Virtual Device (Android Emulator) with either API 29 or API 30 (API 30 preferred),

then launch the Emulator then using the theScore apk file in project root directory under apk folder to install theScore application

5. Launch your Appium desktop, set the host to 0.0.0.0 and port 4723 if you are using Windows. Change the host to 127.0.0.1,

if you are using Mac OS. Do remember to change the host in java file "CommonFunctions" in line 47, to make it as the same as the host in Appium

desktop. Then Start the server.

6. Config your Appium desktop for the setting below:

{

"platformName": "Android",

"deviceName": "Android Emulator",

"app": "D:/theScore/theScoreAssignment/apk/theScore Live Sports Scores News Stats Videos\_v21.2.0\_apkpure.com.apk",

"appPackage": "com.fivemobile.thescore",

"appActivity": "com.fivemobile.thescore.ui.MainActivity",

"noReset": false,

"automationName": "UiAutomator2"

}

Please do change "app" directory to where you have your apk file.

And try to use your Appium desktop to launch theScore application and see if it can be successfully launched

7. If theScore application can be launched using Appium desktop then go back to IntelliJ IDEA, navigate to

the java file "PropertiesClass", for line 13 and 14, please make sure the directory on your computer is correct.

Then go to java file "RunWithCucumber", line 9, make sure the directory is correct as well.

8. Then now, we can start executing the scripts by running the java file "RunWithCucumber".

9. You can see it is a parameterized test in theScoreDemo.feature file as I have set up some parameters for some steps at the bottom.

10. After the execution, you are able to find the test report/specification under /target/cucumber-reports. Use chrome browser to

open the html file.

Note: A pdf version of the previous execution cucumber-report has been placed under the root directory for your reference.

**Structure of the project:**

Under /src/test/java:

* Features
* StepDefinitions
* theScoreDemo

Features: contains .feature file which is the Cucumber file

StepDefinitions: contains StepDefinitions.java which implements the steps for the scenarios

theScoreDemo: contains java files:

* CalendarDate (Return recent 4 days for the verification in league details page)
* CommonFunctions (Common functions to be used in StepDefinitions.java, such as click, sendKey)
* NameForShort (Shorten the league name or team name to its abbreviation when it is too long for verification)
* PrintToConsole (Print whatever the elements the program is currently is looking for, and whether it find it or not)
* RunWithCucumber (Run the program using Cucumber, and set up the Cucumber report)

Under /src/test/resource:

* Properties

Properties: android.properties contains element locators, application.properties contains texts to be used for verification

Under target/cucumber-reports:

* cucumber.html

Under root directory:

theScore Take-Home Assignment Description.doc

This document contains environment setup, project description, description for test approach and automation coverage assessment etc.

Cucumber Test Specification Report.pdf

This document contains a Cucumber Test Specification Report for Previous Execution

Pom.xml – contains maven dependencies

**Short Description of the Rationale of Test Approach:**

I used Junit, AssertJ, Cucumber, Gherkin, Appium to automate the test. As Cucumber is the language that Business can understand and it can easily organized different test cases and scenarios and parameterized the tests as well. Testers are able to understand what the scenario is doing, what the test steps are and change parameters for the test in .feature file.

I used .properties file to store the element locators and some texts for verification and assertion since it is easy and better for us the maintain. Also, JVM does not need to re-compile the project whenever the change has been made in .properties file.

The program also prints elements that it is currently looking for, the way that it locates the elements (eg. Using id or xpath), and return true if it has found the elements.

The test automation scripts covered all the static check points that the requirements asked to verify. All the steps in the scenario have been automated and all checkable points are checked. But at the page of NHL Hockey, there are some news and scores are lively updated and dynamic, which are not be automated and verified by my automation scripts. If we calculate the automation coverage by dividing automated coverage by total coverage, it would be around 90% automation coverage as mentioned above there are some dynamic points which are hard to be verified.