Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

**Лабораторная работа №3**

**Дисциплина «Тестирование программного обеспечения»**

**Вариант 19**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3317

**Преподаватель:**

Исаев Илья Владимирович

Санкт-Петербург

2019

**Задание**

Цель работы: изучить методику функционального тестирования Android приложения, используя Appium.

Необходимо взять приложение согласно варианту, установить эмулятор Genymotion (использовать андроид 6.0 и выше), настроить Appium (http://appium.io) и написать 15 тестовых сценариев. Сценарий установки виджета на экран обязателен. Проверка регистрации обязательна.

**Genymotion**

Genymotion – проприетарный эмулятор Android, который включает в себя настроенные образы Android (x86 с аппаратным ускорением OpenGL), работающий на VirtualBox. Genymotion имеет бесплатную версию со множеством возможностей, но также доступны платные версии для больших компаний, которым нужна совместная работа с Genymotion.

**Appium**

Appium — это HTTP-сервер, написанный на Node.js, который создает и обрабатывает WebDriver-сессии. Appium придерживается того же подхода, что и Selenium WebDriver, который получает HTTP-запросы в формате JSON от клиентов и преобразует их в зависимости от платформы, на которой он работает.

На Android-устройстве Appium использует UIAutomator, чтобы автоматизировать приложение. UIAutomator — фреймворк, созданный командой разработчиков Android для тестирования пользовательского интерфейса.

**Функциональные требования**

|  |  |  |
| --- | --- | --- |
| ID | Название | Описание |
| 1 | Регистрация с некорректным email | Система не должна допускать регистрацию с некорректным адресом электронной почты |
| 2 | Регистрация с некорректным номером телефона | Система не должна допускать регистрацию с некорректным номером телефона |
| 3 | Вход с неверным логином | Система не должна допускать вход с неверным логином |
| 4 | Вход с неверным паролем | Система не должна допускать вход с неверным паролем |
| 5 | Скрыть/показать пароль | Система должна позволять отображать и скрывать введённый пароль по нажатию кнопки |
| 6 | Отправка твитов | Система должна позволять писать твиты |
| 7 | Удаление твитов | Система дожна позволять удалять твиты |
| 8 | Поиск по твитам | Система должна позволять искать твиты |
| 9 | Поиск по людям | Система должна позволять искать людей |
| 10 | Подписка | Система должна позволять подписываться на людей |
| 11 | Отписка | Система должна позволять отписываться от людей |
| 12 | Игнорирование | Система должна позволять игнорировать людей |
| 13 | Отмена игнорирования | Система должна позволять удалять людей из списка игнорируемых |
| 14 | Чёрный список | Система должна позволять добавлять людей в чёрный список |
| 15 | Удаление из чёрного списка | Система должна позволять удалять людей из чёрного списка |

**Код**

Класс PlatformTouchAction используется вместо стандартного TouchAction для обхода ошибки, связанной с выводом типов в Kotlin:

package dmitry.tpo3  
  
import io.appium.java\_client.**PerformsTouchActions**  
import io.appium.java\_client.**TouchAction**  
  
class **PlatformTouchAction**(performsTouchActions: **PerformsTouchActions**) : TouchAction<**PlatformTouchAction**>(performsTouchActions)

Объект Matchers содержит матчер для Kotlintest:

package dmitry.tpo3  
  
import io.appium.java\_client.**MobileElement**  
import io.kotlintest.**Matcher**  
import io.kotlintest.**MatcherResult**  
import io.kotlintest.should  
import org.openqa.selenium.**StaleElementReferenceException**  
  
object **Matchers** {  
 fun beStale() = object : **Matcher**<**MobileElement**> {  
 override fun test(element: **MobileElement**): **MatcherResult** {  
 val isStale = try {  
 element.*isDisplayed*  
false  
 } catch(e: **StaleElementReferenceException**) {  
 true  
 }  
  
 return **MatcherResult**(isStale,  
 "Element **$**element should be stale",  
 "Element **$**element should not be stale")  
 }  
 }  
}

Класс AbstractTwitterTestSet является базовым для наборов тестов:

package dmitry.tpo3.tests  
  
import dmitry.tpo3.**PlatformTouchAction**  
import io.appium.java\_client.**MobileBy**  
import io.appium.java\_client.**MobileElement**  
import io.appium.java\_client.android.**AndroidDriver**  
import io.appium.java\_client.android.nativekey.**AndroidKey**  
import io.appium.java\_client.android.nativekey.**KeyEvent**  
import io.appium.java\_client.remote.**AndroidMobileCapabilityType**  
import io.appium.java\_client.remote.**MobileCapabilityType**  
import io.appium.java\_client.touch.offset.**PointOption**  
import io.appium.java\_client.windows.**PressesKeyCode**  
import io.kotlintest.**IsolationMode**  
import io.kotlintest.**Spec**  
import io.kotlintest.**TestCaseOrder**  
import io.kotlintest.shouldThrow  
import io.kotlintest.specs.**WordSpec**  
import org.openqa.selenium.**By**  
import org.openqa.selenium.**NoSuchElementException**  
import org.openqa.selenium.**Point**  
import org.openqa.selenium.**StaleElementReferenceException**  
import org.openqa.selenium.remote.**DesiredCapabilities**  
import org.openqa.selenium.support.ui.**ExpectedConditions**  
import org.openqa.selenium.support.ui.**WebDriverWait**  
import java.net.**URL**  
import java.util.concurrent.**TimeUnit**  
  
abstract class **AbstractTwitterTestSet**() : WordSpec() {  
 companion object {  
 const val USERNAME = "Tester1234519"  
 const val PASSWORD = "qwerty123456"  
  
 const val LOGIN\_TEXT\_ID = "com.twitter.android:id/detail\_text"  
 const val USERNAME\_INPUT\_ID = "com.twitter.android:id/login\_identifier"  
 const val PASSWORD\_INPUT\_ID = "com.twitter.android:id/login\_password"  
 const val LOGIN\_BUTTON\_ID = "com.twitter.android:id/login\_login"  
  
 @JvmStatic protected val driver: **AndroidDriver**<**MobileElement**>  
 @JvmStatic protected val wait: **WebDriverWait**  
  
init {  
 val capabilityDict = *mapOf*(  
 **MobileCapabilityType**.*DEVICE\_NAME to* "emulator-5554",  
 **MobileCapabilityType**.*PLATFORM\_NAME to* "Android",  
 **MobileCapabilityType**.*PLATFORM\_VERSION to* "10",  
 **MobileCapabilityType**.*NO\_RESET to* "true",  
 **MobileCapabilityType**.*AUTOMATION\_NAME to* "UIAutomator2",  
 **AndroidMobileCapabilityType**.*APP\_PACKAGE to* "com.twitter.android",  
 **AndroidMobileCapabilityType**.*APP\_ACTIVITY to* "com.twitter.android.StartActivity"  
 )  
  
 val capabilities = DesiredCapabilities()  
 capabilityDict.*forEach* **{** (name, value) **->**  
capabilities.setCapability(name, value)  
 **}**  
  
val url = URL("http://127.0.0.1:4723/wd/hub")  
  
 driver = AndroidDriver<MobileElement>(url, capabilities)  
 wait = WebDriverWait(driver, 30)  
  
 driver.manage().timeouts().implicitlyWait(15, **TimeUnit**.SECONDS)  
 }  
 }  
  
 override fun testCaseOrder(): **TestCaseOrder**? = **TestCaseOrder**.Random  
  
 private fun find(locator: **By**): **MobileElement** {  
 wait.until **{**  
 **ExpectedConditions**.visibilityOfElementLocated(**locator**)  
 **}**  
return driver.findElement(**locator**)  
 }  
  
 protected fun findById(id: **String**): **MobileElement** = find(**MobileBy**.id(id))  
 protected fun findByXpath(xpath: **String**): **MobileElement** = find(**MobileBy**.xpath(xpath))

protected fun goBack() {  
 driver.pressKey( KeyEvent(**AndroidKey**.BACK) )  
 }  
  
 protected fun goBack(n: **Int**) {  
 for (i in 0 *until* n) {  
 driver.pressKey( KeyEvent(**AndroidKey**.BACK) )  
 }  
 }  
  
 protected fun clickAt(p: **Point**) {  
 PlatformTouchAction(driver).tap(**PointOption**.point(p)).perform()  
 }  
}

Класс RegistrationAndLoginTests содержит тесты регистрации и входа:

package dmitry.tpo3.tests  
  
import io.appium.java\_client.**MobileBy**  
import io.kotlintest.**IsolationMode**  
import io.kotlintest.shouldBe  
import io.kotlintest.shouldNotBe  
import org.openqa.selenium.support.ui.**ExpectedConditions**  
  
  
class **RegistrationAndLoginTests** : AbstractTwitterTestSet() {  
 companion object {  
 const val REGISTER\_BUTTON\_ID = "com.twitter.android:id/primary\_action"  
 const val PHONE\_EMAIL\_INPUT\_ID = "com.twitter.android:id/phone\_or\_email\_field"  
 const val USE\_EMAIL\_BUTTON\_ID = "com.twitter.android:id/secondary\_button"  
 const val NEXT\_BUTTON\_ID = "com.twitter.android:id/cta\_button"  
 }  
  
 override fun isolationMode(): **IsolationMode**? = **IsolationMode**.InstancePerLeaf  
  
 init {  
 "Registration" *should* **{**  
findById(REGISTER\_BUTTON\_ID).click()  
  
 val textField = findById(PHONE\_EMAIL\_INPUT\_ID)  
 val nextButton = findById(NEXT\_BUTTON\_ID)  
  
 "fail with a non-existing email" **{**  
 **textField**.click()  
  
 val useEmail = findById(USE\_EMAIL\_BUTTON\_ID)  
 useEmail.click()  
  
 **textField**.sendKeys("invalidmail@domain.null")  
 **nextButton**.*isEnabled shouldBe* false  
 **textField**.clear()  
  
 useEmail.click()  
  
 goBack()  
 **}**  
  
"fail with a non-existing phone number" **{**  
 **textField**.sendKeys("+1-202-555-0180")  
 **nextButton**.*isEnabled shouldBe* false  
 **textField**.clear()  
  
 goBack()  
 **}**  
 **}**  
  
"Log in" *should* **{**  
val loginText = findById(LOGIN\_TEXT\_ID)  
 *with*(loginText.*location*) **{**  
x += 600  
 clickAt(this)  
 **}**  
  
val usernameInput = findById(USERNAME\_INPUT\_ID)  
 val passwordInput = findById(PASSWORD\_INPUT\_ID)  
 val loginButton = findById(LOGIN\_BUTTON\_ID)  
  
 "fail with invalid username" **{**  
 **usernameInput**.sendKeys(USERNAME.*reversed*())  
 **passwordInput**.sendKeys(PASSWORD)  
 **loginButton**.click()  
  
 wait.until **{**  
 **ExpectedConditions**.invisibilityOfElementLocated(**MobileBy**.id("android:id/progress"))  
 **}**  
  
findById(USERNAME\_INPUT\_ID).*isDisplayed shouldBe* true  
  
 goBack()  
 **}**  
  
"fail with invalid password" **{**  
 **usernameInput**.sendKeys(USERNAME)  
 **passwordInput**.sendKeys(PASSWORD.*reversed*())  
 **loginButton**.click()  
  
 wait.until **{**  
 **ExpectedConditions**.invisibilityOfElementLocated(**MobileBy**.id("android:id/progress"))  
 **}**  
  
findById(USERNAME\_INPUT\_ID).*isDisplayed shouldBe* true  
  
 goBack()  
 **}**  
  
"show/hide password" **{**  
val buttonLocation = **passwordInput**.*location*  
buttonLocation.x += 950  
  
 **passwordInput**.sendKeys(PASSWORD)  
  
 clickAt(buttonLocation)  
 **passwordInput**.*text shouldBe* PASSWORD  
 clickAt(buttonLocation)  
 **passwordInput**.*text shouldNotBe* PASSWORD  
  
 goBack()  
 **}**  
 **}**  
}  
}

Класс AppTests содержит тесты самого приложения. Перед их запуском он выполняет вход в приложение, а после – выход:

package dmitry.tpo3.tests  
  
import io.appium.java\_client.**MobileBy**  
import io.kotlintest.**IsolationMode**  
import io.kotlintest.shouldBe  
import io.kotlintest.shouldNotBe  
import org.openqa.selenium.support.ui.**ExpectedConditions**  
  
  
class **RegistrationAndLoginTests** : AbstractTwitterTestSet() {  
 companion object {  
 const val REGISTER\_BUTTON\_ID = "com.twitter.android:id/primary\_action"  
 const val PHONE\_EMAIL\_INPUT\_ID = "com.twitter.android:id/phone\_or\_email\_field"  
 const val USE\_EMAIL\_BUTTON\_ID = "com.twitter.android:id/secondary\_button"  
 const val NEXT\_BUTTON\_ID = "com.twitter.android:id/cta\_button"  
 }  
  
 override fun isolationMode(): **IsolationMode**? = **IsolationMode**.InstancePerLeaf  
  
 init {  
 "Registration" *should* **{**  
findById(REGISTER\_BUTTON\_ID).click()  
  
 val textField = findById(PHONE\_EMAIL\_INPUT\_ID)  
 val nextButton = findById(NEXT\_BUTTON\_ID)  
  
 "fail with a non-existing email" **{**  
 **textField**.click()  
  
 val useEmail = findById(USE\_EMAIL\_BUTTON\_ID)  
 useEmail.click()  
  
 **textField**.sendKeys("invalidmail@domain.null")  
 **nextButton**.*isEnabled shouldBe* false  
 **textField**.clear()  
  
 useEmail.click()  
  
 goBack()  
 **}**  
  
"fail with a non-existing phone number" **{**  
 **textField**.sendKeys("+1-202-555-0180")  
 **nextButton**.*isEnabled shouldBe* false  
 **textField**.clear()  
  
 goBack()  
 **}**  
 **}**  
  
"Log in" *should* **{**  
val loginText = findById(LOGIN\_TEXT\_ID)  
 *with*(loginText.*location*) **{**  
x += 600  
 clickAt(this)  
 **}**  
  
val usernameInput = findById(USERNAME\_INPUT\_ID)  
 val passwordInput = findById(PASSWORD\_INPUT\_ID)  
 val loginButton = findById(LOGIN\_BUTTON\_ID)  
  
 "fail with invalid username" **{**  
 **usernameInput**.sendKeys(USERNAME.*reversed*())  
 **passwordInput**.sendKeys(PASSWORD)  
 **loginButton**.click()  
  
 wait.until **{**  
 **ExpectedConditions**.invisibilityOfElementLocated(**MobileBy**.id("android:id/progress"))  
 **}**  
  
findById(USERNAME\_INPUT\_ID).*isDisplayed shouldBe* true  
  
 goBack()  
 **}**  
  
"fail with invalid password" **{**  
 **usernameInput**.sendKeys(USERNAME)  
 **passwordInput**.sendKeys(PASSWORD.*reversed*())  
 **loginButton**.click()  
  
 wait.until **{**  
 **ExpectedConditions**.invisibilityOfElementLocated(**MobileBy**.id("android:id/progress"))  
 **}**  
  
findById(USERNAME\_INPUT\_ID).*isDisplayed shouldBe* true  
  
 goBack()  
 **}**  
  
"show/hide password" **{**  
val buttonLocation = **passwordInput**.*location*  
buttonLocation.x += 950  
  
 **passwordInput**.sendKeys(PASSWORD)  
  
 clickAt(buttonLocation)  
 **passwordInput**.*text shouldBe* PASSWORD  
 clickAt(buttonLocation)  
 **passwordInput**.*text shouldNotBe* PASSWORD  
  
 goBack()  
 **}**  
 **}**  
}  
}

**Выводы**

Appium построен на базе WebDriver и может быть использован для тестирования приложений как под iOS, так и под Android. При этом с помощью аннотаций можно задать локаторы для каждой платформы.

Тестирование серого ящика сложнее выполнять с мобильными приложениями, чем с веб-сайтами. Для получения структуры представления требуется использовать UIAutomatorViewer, тогда как структура HTML-страницы доступна всегда. Кроме того, в мобильных приложениях часто требуется использовать жесты, которые может быть тяжело реализовать программно.

Во время тестирования был выявлен незначительный дефект, который заключается в том, что поиск по пользователям иногда не срабатывает.