

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

¡Bienvenidos! En esta guía aprenderemos a implementar nuestra primera tarea.

Aprenderemos cómo interactúa el actor con las tareas (tasks) sobre la interfaz de usuario (userinterface) para el desarrollo de cada paso descrito en la historia de usuario. Y aprenderemos cómo identificar los objetos en nuestro page.

Contenido

Implementación del Given.....	2
Configuración de Appium	5
Creación de nuestra task y nuestra userinterface.....	13
Implementación del @When.....	15



ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Implementación del Given.

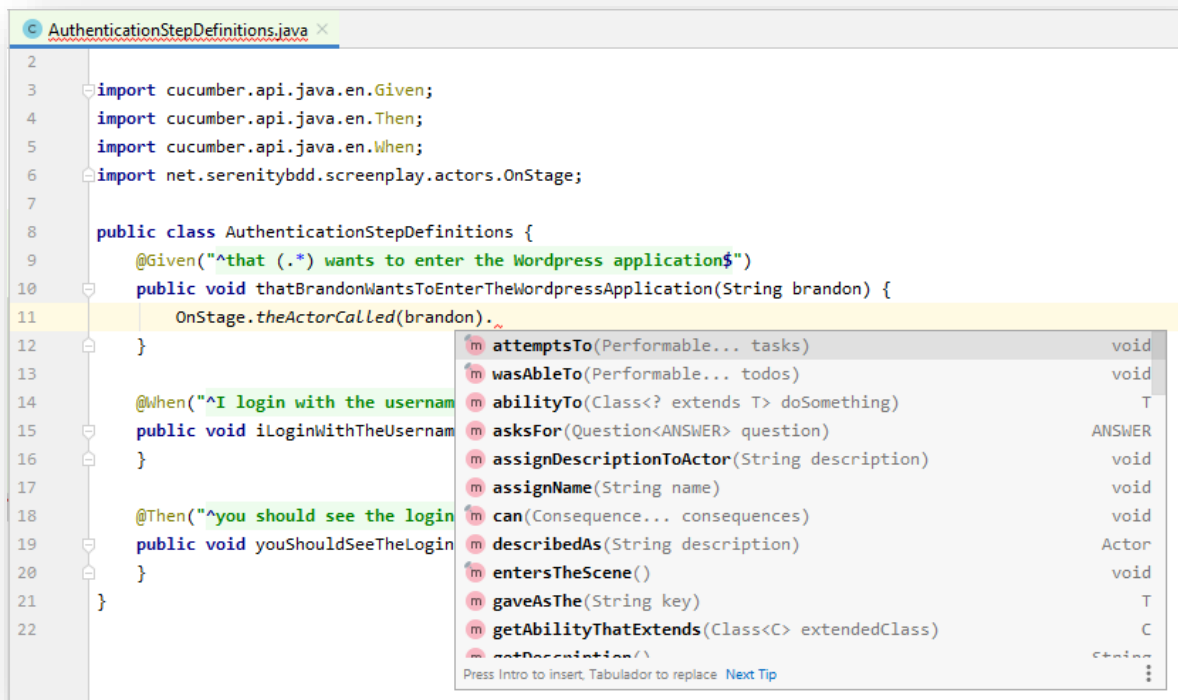
Retomando donde terminamos la guía pasada, vayamos a nuestra clase AuthenticationStepDefinitions y empecemos la implementación del paso:

Given that Brandon wants to enter the Wordpress application

Dicha implementación la haremos en el método llamado:

public void thatBrandonWantsToEnterTheWordpressApplication()

Lo primero que haremos será escribir el nombre de nuestro actor en nuestra clase stepdefinitions. En el ejemplo seguido por esta guía, nuestro actor se llama “brandon”, una vez declaremos el actor, escribiremos el carácter punto, “.”. Observaremos que nos mostrará todos los métodos que nuestro actor brandon puede ejecutar. y además vamos a modificar la información de la etiqueta @Given con el fin de encapsular el nombre de el actor en una variable de tipo String



```
2
3 import cucumber.api.java.en.Given;
4 import cucumber.api.java.en.Then;
5 import cucumber.api.java.en.When;
6 import net.serenitybdd.screenplay.actors.OnStage;
7
8 public class AuthenticationStepDefinitions {
9     @Given("^that (.*) wants to enter the Wordpress application$")
10    public void thatBrandonWantsToEnterTheWordpressApplication(String brandon) {
11        OnStage.theActorCalled(brandon).
12    }
13
14    @When("^I login with the username")
15    public void iLoginWithTheUsernam
16    }
17
18    @Then("^you should see the login")
19    public void youShouldSeeTheLogin
20    }
21 }
22
```

Methods available for actor 'brandon':

- attemptsTo(Performable... tasks) void
- wasAbleTo(Performable... todos) void
- abilityTo(Class<? extends T> doSomething() T
- asksFor(Question<ANSWER> question) ANSWER
- assignDescriptionToActor(String description) void
- assignName(String name) void
- can(Consequence... consequences) void
- describedAs(String description) Actor
- entersTheScene() void
- gaveAsThe(String key) T
- getAbilityThatExtends(Class<C> extendedClass) C
- getDescription() String



ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Continuando con la implementación del método given utilizaremos `wasAbleTo` debido a que el `given` es una precondición y por ende se debe hablar en pasado.

```
@Given("^that (.*) wants to enter the Wordpress application$")
public void thatBrandonWantsToEnterTheWordpressApplication(String brandon) {
    OnStage.theActorCalled(brandon).wasAbleTo(OpenThe.wordpressApp());
}
```

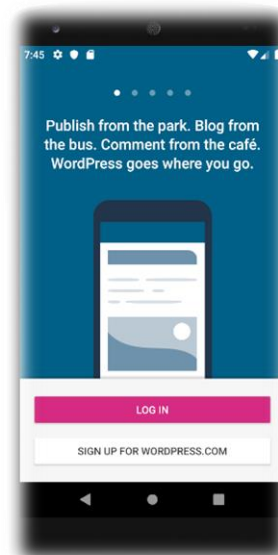
Además, nuestra tarea se llamará `OpenThe` y nuestro método `WordPress()`, estos dos serán los encargados de permitir el acceso al apartado de logueo de la app.

Antes de continuar con la creación de la clase `OpenThe`, vamos a preparar el escenario para el actor. Para esta parte vamos a hacer uso del paquete **utils**, donde crearemos una clase que llamaremos **Hook** la cual contendrá el siguiente método:

```
@Before
public void prepareStage() {
    OnStage.setTheStage(new OnlineCast());
}
```

Nuestra clase debe verse así:

```
Hook.java x
1 package co.com.choucair.automation.android.utils;
2
3 import cucumber.api.java.Before;
4 import net.serenitybdd.screenplay.actors.OnStage;
5 import net.serenitybdd.screenplay.actors.OnlineCast;
6
7 public class Hook {
8     @Before
9     public void prepareStage() {
10         OnStage.setTheStage(new OnlineCast());
11     }
12 }
```



ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

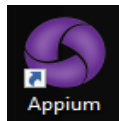
Sin embargo, nos hace falta un paso para terminar de preparar la escena para el actor, y esta parte la vamos a realizar en la clase **RunnerTags** en la variable llamada **glue** donde agregaremos la ruta del paquete utils y nuestra clase RunnerTags debe quedar así:

```
RunnerTags.java x
1 package co.com.choucair.automation.android.runners;
2
3 import cucumber.api.CucumberOptions;
4 import cucumber.api.SnippetType;
5 import net.serenitybdd.cucumber.CucumberWithSerenity;
6 import org.junit.runner.RunWith;
7
8 @RunWith(CucumberWithSerenity.class)
9 @CucumberOptions(features = "src/test/resources/features/wordpress.feature",
10                  glue = {"co.com.choucair.automation.android.stepdefinitions", "co.com.choucair.automation.android.utils"},
11                  snippets = SnippetType.CAMELCASE)
12 public class RunnerTags {
13 }
14
```

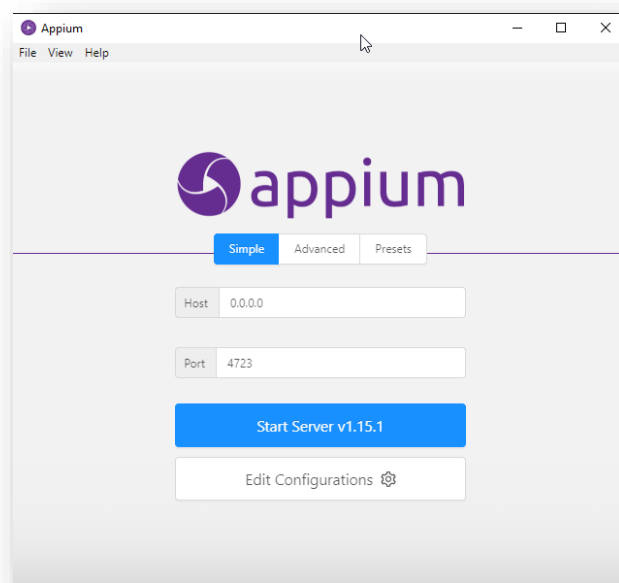


ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

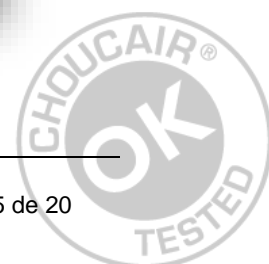
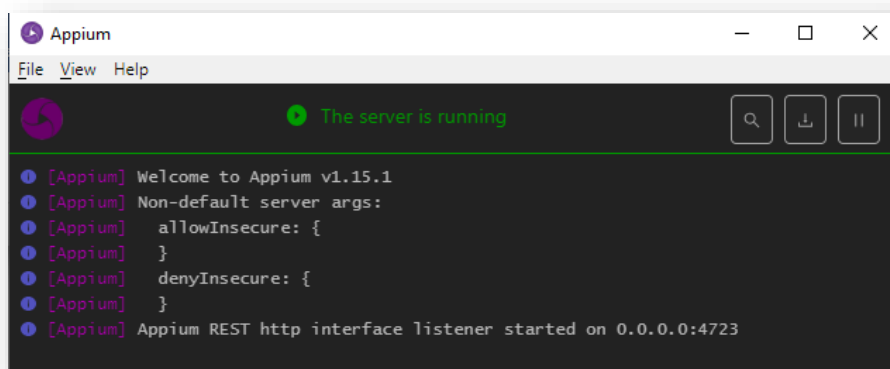
Configuración de Appium



1. Abrir la aplicación de Appium

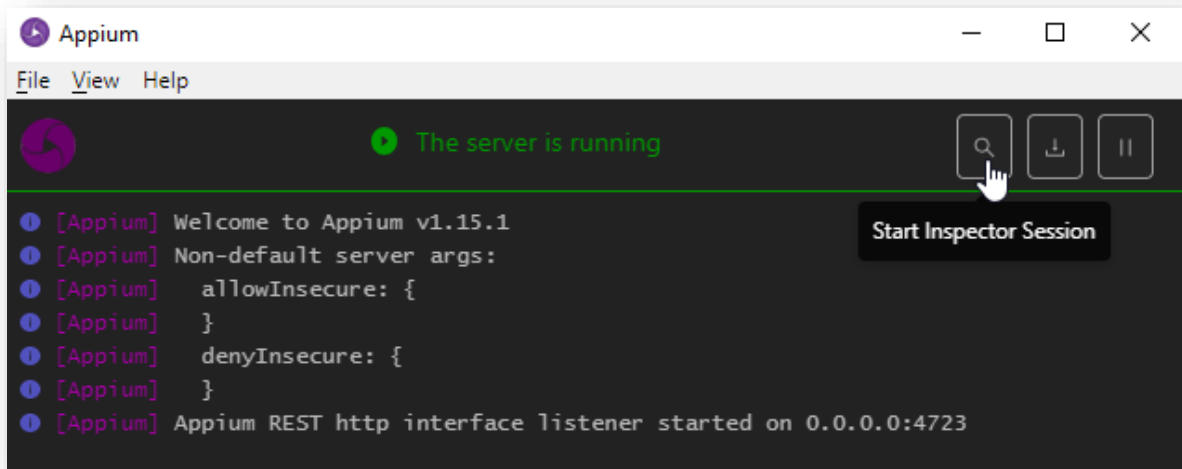


2. Una vez se cargue la pantalla de la aplicación nos abrirá en la pestaña simple donde el campo "Host" el valor por defecto es "0.0.0.0" o "127.0.0.1" y el campo "Port" dejaremos el default "4723" y dar clic en el botón "Start Server v1.15.1". Veremos el inicio del servidor.

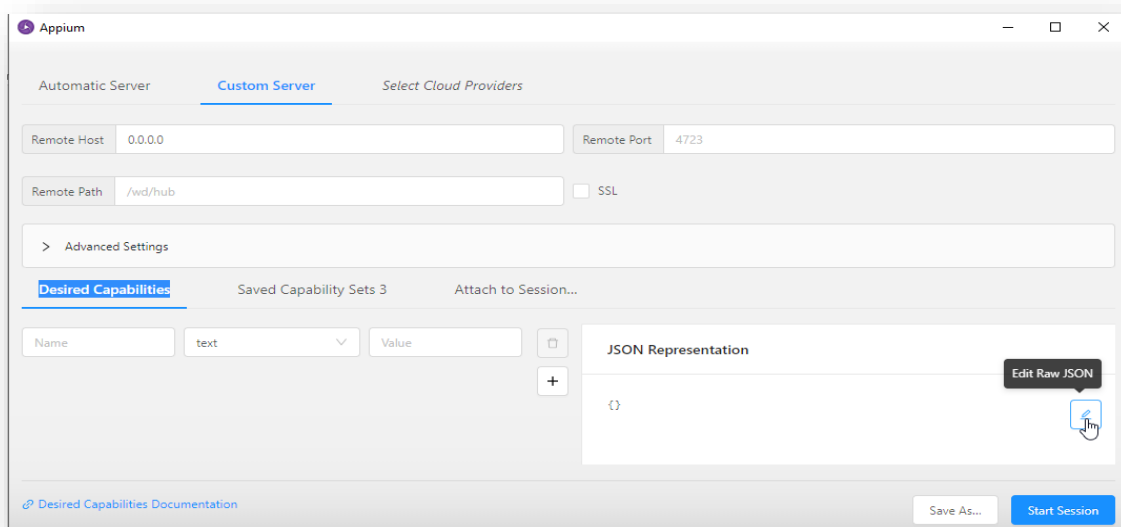


ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

- Continuaremos con la configuración del inspector de elementos. Para ello daremos clic al botón de la lupa que se llama “Start Inspector Session”



- Nos mostrará una nueva ventana donde configuraremos las capacidades deseadas, se debe dar clic en el botón “Edit Raw JSON” para parametrizar la sesión que nos permitirá inspeccionar cada uno de los objetos de nuestra APP.



ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

5. Agregamos las siguientes parametrizaciones al JSON Representation .

```
{
  "deviceName": "Nexus_5",
  "udid": "emulator-5554",
  "platformName": "Android",
  "platformVersion": "9",
  "appPackage": "org.wordpress.android",
  "appActivity": "org.wordpress.android.ui.WPLaunchActivity",
  "noReset": true,
  "app": "D:/Proyectos_Intellij_Android/co.com.choucair.automation.android/src/test/resources/app/wordpress.apk"
}
```

deviceName: Nombre que tiene nuestro dispositivo y que nos permite identificarlo.

udid: El Unique Device Identifier (UDID) es un código de 40 dígitos hexadecimales (alfa-númericos). Este código representa un único dispositivo.

platformName: Nombre Sistema Operativo del dispositivo celular (Android/iOS).

platformVersion: Version del sistema operativo del dispositivo.

appPackage: Paquete de una APP el cual contiene el desarrollo de la necesidad que suple la misma.

appActivity: Actividad de la APP

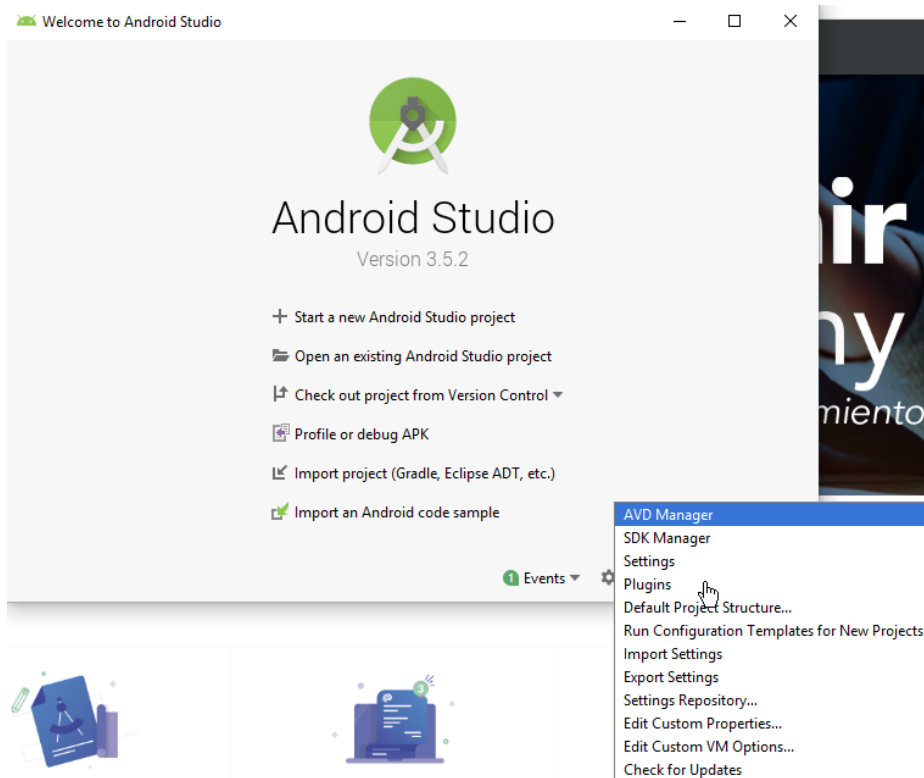
noReset: true : No restablezca el estado de la aplicación antes de esta sesión(true/false)

app: La ruta local absoluta o la URL http remota de la apk

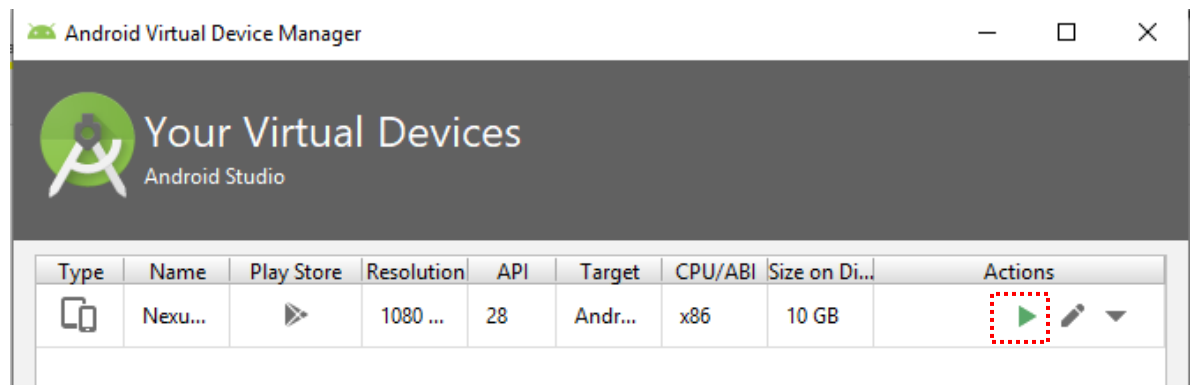
Para más información consulta [aquí](#).

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

6. Abrimos Android Studio >Configure>ADV Manager



7. Seleccionamos en Actions el boton Play.



ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

8. Y nos cargara el emulador del dispositivo que tengamos creado.



9. Para encontrar el dato del parámetro **udid** es necesario abrir el CMD de Windows e ingresar el comando **adb devices**

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.17763.652]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

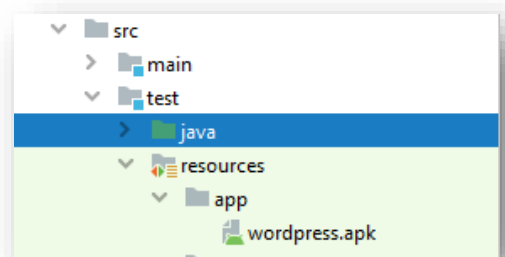
C:\Users\bquevedof>adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
emulator-5554    device
```

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

10. **appPackage** y **appActivity**: Estas opciones se pueden visualizar realizando la instalación de una APP en el celular llamada APK info la cual es gratuita o la puedes descargar [aquí](#). Para instalarla en la maquina **AVD manager** solo necesitas tener iniciado el dispositivo y arrastrar el apk hasta él y automáticamente se instala.



Una vez instalada accedemos a la misma y desde esta buscamos la aplicación a la cual queremos verificar este par de parámetros (esto quiere decir que de antemano debo instalar la aplicación **WordPress** en mi celular la cual se encuentra en la ruta **src>test>resources>app** de mi proyecto).



appPackage	appActivity

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

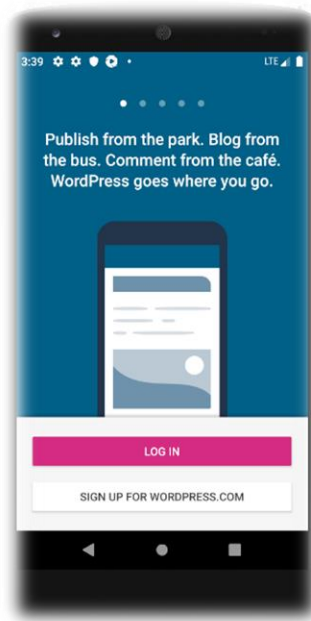
O también podemos adquirir los datos de **appPackage** y **appActivity** a través de la consola haciendo uso de dos comandos [adb Shell] y [dumpsys window windows | grep -E 'mCurrentFocus'], pero es necesario tener iniciada la aplicación en el dispositivo móvil.

```
C:\WINDOWS\system32\cmd.exe - adb shell
Microsoft Windows [Versión 10.0.17763.652]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\bquevedof>adb shell
generic_x86_arm:/ $ dumsys window windows | grep -E 'mCurrentFocus'
mCurrentFocus=Window{bdd1e8e u0:org.wordpress.android/org.wordpress.android.ui.accounts.LoginActivity}
```

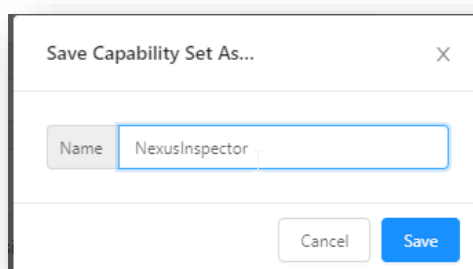
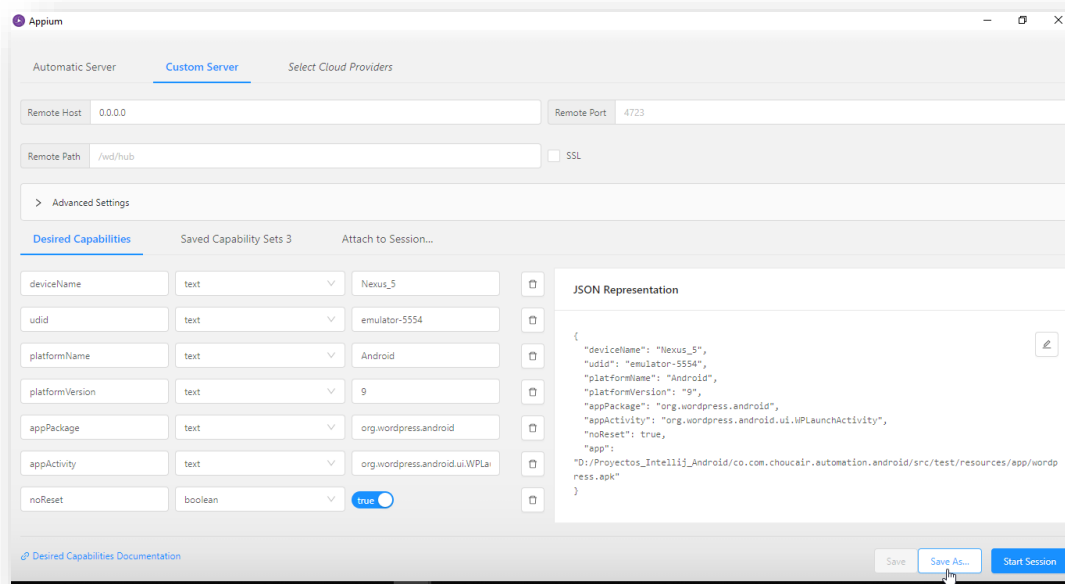
appPackage

appActivity



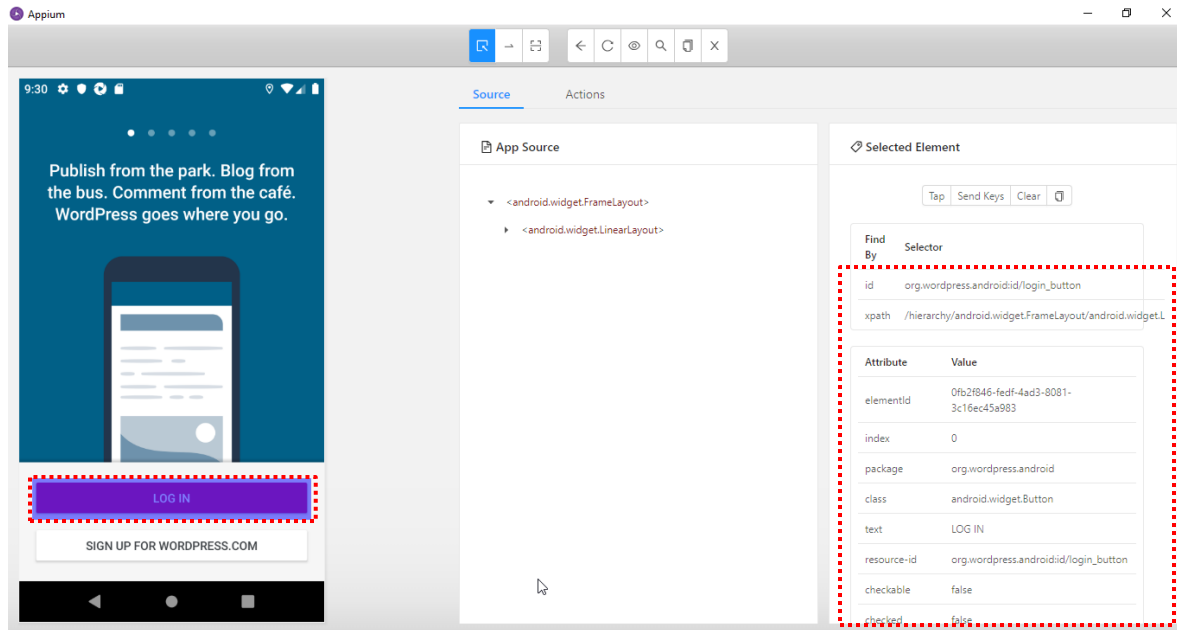
ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

11. Una vez tenemos ya nuestra información recolectada y escrita en el campo JSON Representation, procedemos a dar clic en el botón “Save As...” y asignamos un nombre a nuestra “Save Capability Set As...”. Y damos clic en “save”.



Y finalmente damos clic en el botón “Start Session” para comenzar la inspección de los objetos de nuestra APP. Esta inspección se realiza dando clic en los objetos que se visualizan en pantalla e inmediatamente en la sección de pantalla “Selected Element” se puede observar las distintas propiedades del mismo.

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium



Creación de nuestra task y nuestra userinterface

Ahora si procedemos a crear nuestra clase **OpenThe** en el paquete **tasks**, dicha clase debe implementar de la clase **Task** e implementar su método. Además, se debe crear el método **wordpressApp()**, nuestra clase de ir así:

```
1 package co.com.choucair.automation.android.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6
7 public class OpenThe implements Task {
8     public static OpenThe wordpressApp() {
9         return Tasks.instrumented(OpenThe.class);
10    }
11    @Override
12    public <T extends Actor> void performAs(T actor) {
13    }
14 }
15
16
```

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

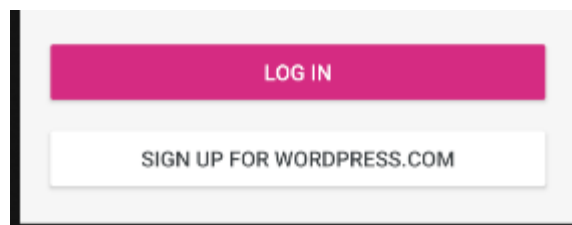
Para la creación de nuestra userinterface que será donde ira el mapeo de los objetos inspeccionados con Appium.

Crea una clase en el paquete userinterface en nuestro caso la llamo **ApplicationHomepage**

```
ApplicationHomepage.java x
1 package co.com.choucair.automation.android.userinterfaces;
2
3 public class ApplicationHomepage {
4 }
5
```

Mapearemos el objeto, es decir el botón LOG IN

Con el inspector de Appium, declararemos una variable de tipo Target estática y la localizaremos por el id que buscamos con la herramienta.



Tu clase **ApplicationHomepage** debe verse así:

```
ApplicationHomepage.java x
1 package co.com.choucair.automation.android.userinterfaces;
2
3 import net.serenitybdd.screenplay.targets.Target;
4 import org.openqa.selenium.By;
5
6 public class ApplicationHomepage {
7     public static final Target LOGIN_BUTTON = Target.the("login button")
8         .located(By.id("org.wordpress.android:id/login_button"));
9 }
```

Y para nuestra clase **OpenThe** tenemos que hacer la acción a través del actor que para este caso será un clic sobre el objeto **LOGIN_BUTTON**.

```
OpenThe.java x
1 package co.com.choucair.automation.android.tasks;
2
3 import net.serenitybdd.screenplay.Actor;
4 import net.serenitybdd.screenplay.Task;
5 import net.serenitybdd.screenplay.Tasks;
6 import net.serenitybdd.screenplay.actions.Click;
7
8 import static co.com.choucair.automation.android.userinterfaces.ApplicationHomepage.LOGIN_BUTTON;
9
10 public class OpenThe implements Task {
11     public static OpenThe wordpressApp() { return Tasks.instrumented(OpenThe.class); }
12
13     @Override
14     public <T extends Actor> void performAs(T actor) {
15         actor.attemptsTo(Click.on(LOGIN_BUTTON));
16     }
17 }
18
```

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Implementación del @When

Volvemos a nuestra clase AuthenticationStepDefinitions y empezamos la implementación del paso del when:

When I login with the username "pruebaappappium@gmail.com" and the password "pruebaapp99"

Dicha implementación la haremos en el método llamado:

```
public void iLoginWithTheUsernameAndThePassword(String arg1, String arg2)
```

Lo primero que haremos será llamar al actor a la escena para que este atento a iniciar sesión con usuario y contraseña

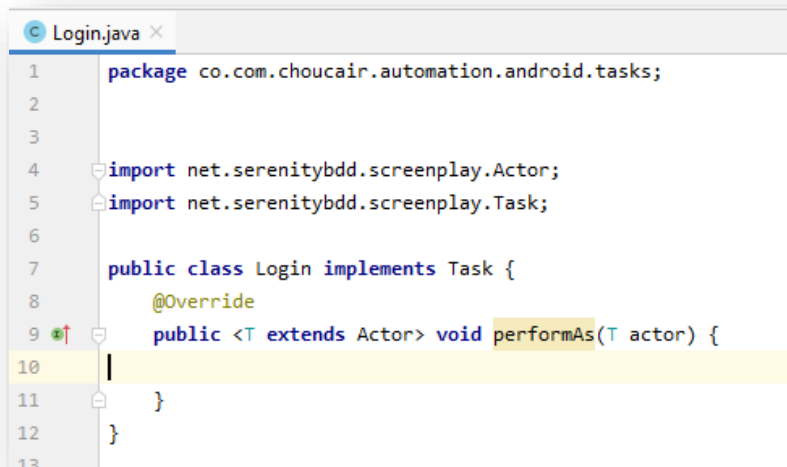
```
@When("^I login with the username \"([^\"]*)\" and the password \"([^\"]*)\"$")  
public void iLoginWithTheUsernameAndThePassword(String arg1, String arg2) {  
    OnStage.theActorInTheSpotlight().attemptsTo(Login.with(arg1,arg2));  
}
```

Además, modificaremos el nombre de la variable recomendadas por cucumber, arg1 la llamaremos "user" y arg2 la llamaremos "password".

```
@When("^I login with the username \"([^\"]*)\" and the password \"([^\"]*)\"$")  
public void iLoginWithTheUsernameAndThePassword(String user, String password) {  
    OnStage.theActorInTheSpotlight().attemptsTo(Login.with(user,password));  
}
```

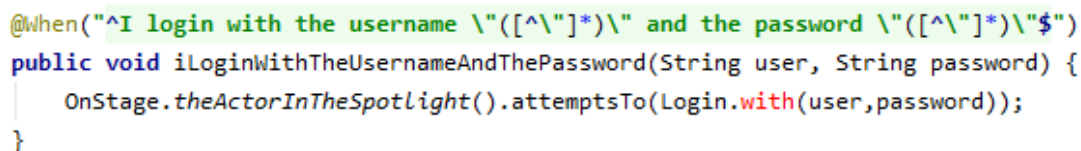
ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Paso a seguir será crear la clase **Login** y el método **with**. La clase la crearemos en el paquete **tasks** e implementaremos **Task** de la librería de **serenitybdd**.



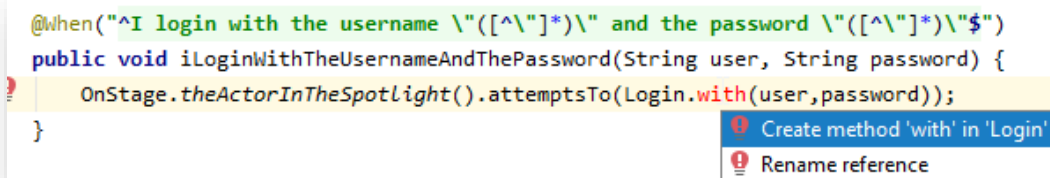
```
1 package co.com.choucair.automation.android.tasks;
2
3
4 import net.serenitybdd.screenplay.Actor;
5 import net.serenitybdd.screenplay.Task;
6
7 public class Login implements Task {
8     @Override
9     public <T extends Actor> void performAs(T actor) {
10
11     }
12 }
13
```

Si volvemos a nuestro stepdefinitions tendremos algo así:



```
@When("^I login with the username \"([^\"]*)\" and the password \"([^\"]*)\"$")
public void iLoginWithTheUsernameAndThePassword(String user, String password) {
    OnStage.theActorInTheSpotlight().attemptsTo(Login.with(user,password));
}
```

Nos posicionamos sobre el método **with** y usamos la siguiente combinación de teclas (Alt + Enter) y seleccionamos la opción **Create method 'with' in 'Login'**



```
@When("^I login with the username \"([^\"]*)\" and the password \"([^\"]*)\"$")
public void iLoginWithTheUsernameAndThePassword(String user, String password) {
    OnStage.theActorInTheSpotlight().attemptsTo(Login.with(user,password));
}
```

Create method 'with' in 'Login'
Rename reference

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Automáticamente el editor nos creara en método.

```
public class Login implements Task {  
  
    public static Performable with(String user, String password) {  
    }  
  
    @Override  
    public <T extends Actor> void performAs(T actor) {
```

El cual vamos a modificar la interface Performable y vamos a poner el nombre de la clase.

```
public static Login with(String user, String password) {  
}
```

Nos presenta error porque necesitamos retornar algo, en este caso será la clase y los atributos de la misma, además declararemos dos variables de tipo private y le generamos su respectivo constructor.

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

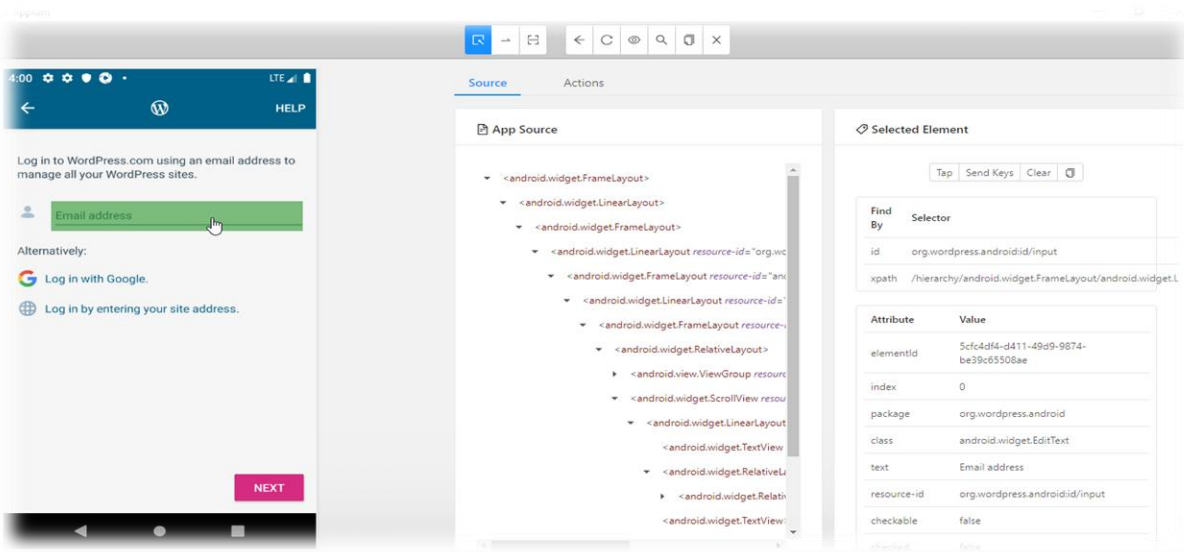
```
1 Login.java x
4 import net.serenitybdd.screenplay.Actor;
5 import net.serenitybdd.screenplay.Task;
6 import net.serenitybdd.screenplay.Tasks;
7
8 public class Login implements Task {
9     private String user;
10    private String password;
11
12    public Login(String user, String password) {
13        this.user = user;
14        this.password = password;
15    }
16
17    public static Login with(String user, String password) {
18        return Tasks.instrumented(Login.class, user, password);
19    }
20
21    @Override
22    public <T extends Actor> void performAs(T actor) {
23
24    }
25 }
26
```

Y nuestro stepdefinitions en la parte del when debe verse así:

```
@When("^I login with the username \"([^\"]*)\" and the password \"([^\"]*)\"$")
public void iLoginWithTheUsernameAndThePassword(String user, String password) {
    OnStage.theActorInTheSpotlight().attemptsTo(Login.with(user,password));
}
```

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Continuamos con el mapeo de los objetos al inicio de sesión de la app de wordpress, lo hacemos con la herramienta de inspeccionar de Appium, además crearemos una clase **LoginPage** en el paquete **userinterfaces**, donde tendremos todo el mapeo de todos los objetos como elementos de tipo Target.



```
LoginPage.java
1 package co.com.choucair.automation.android.userinterfaces;
2
3 import net.serenitybdd.screenplay.targets.Target;
4 import org.openqa.selenium.By;
5
6 public class LoginPage {
7     public static final Target EMAIL_INPUT = Target.the( targetElementName: "input for write email")
8         .located(By.id("org.wordpress.android:id/input"));
9     public static final Target EMAIL_BUTTON = Target.the( targetElementName: "button next email")
10        .located(By.id("org.wordpress.android:id/primary_button"));
11     public static final Target PASSWORD_LINK = Target.the( targetElementName: "Link for enter your password")
12        .located(By.id("org.wordpress.android:id/login_enter_password"));
13     public static final Target PASSWORD_INPUT = Target.the( targetElementName: "input for write password")
14        .located(By.id("org.wordpress.android:id/input"));
15     public static final Target PASSWORD_BUTTON = Target.the( targetElementName: "button next password")
16        .located(By.id("org.wordpress.android:id/primary_button"));
17     public static final Target TEXT_LABEL = Target.the( targetElementName: "text label for validations")
18        .located(By.id("org.wordpress.android:id/logged_in_as_heading"));
19 }
20
```

ScreenPlay +BDD+Appium+Android 3 - Implementar nuestra primera tarea con Appium

Ahora volvemos a la clase Login a implementar las acciones que el actor realizara a los objetos. Dichas acciones para este ejercicio serán dar clic e ingresar información un tiempo de espera.

```
1 package co.com.choucair.automation.android.tasks;
2 import net.serenitybdd.screenplay.Actor;
3 import net.serenitybdd.screenplay.Task;
4 import net.serenitybdd.screenplay.Tasks;
5 import net.serenitybdd.screenplay.actions.Click;
6 import net.serenitybdd.screenplay.actions.Enter;
7 import net.serenitybdd.screenplay.waits.WaitUntil;
8
9 import static co.com.choucair.automation.android.userinterfaces.LoginPage.*;
10 import static net.serenitybdd.screenplay.matchers.WebElementMatchers.isPresent;
11
12 public class Login implements Task {
13     private String user;
14     private String password;
15
16     public Login(String user, String password) {
17         this.user = user;
18         this.password = password;
19     }
20
21     public static Login with(String user, String password) { return Tasks.instrumented(Login.class, user, password); }
22
23     @Override
24     public <T extends Actor> void performAs(T actor) {
25         actor.attemptsTo(
26             Enter.theValue(user).into(EMAIL_INPUT),
27             Click.on(EMAIL_BUTTON),
28             Click.on(PASSWORD_LINK),
29             Enter.theValue(password).into(PASSWORD_INPUT),
30             Click.on(PASSWORD_BUTTON),
31             WaitUntil.the(TEXT_LABEL, isPresent()).forNoMoreThan( amount: 10).seconds()
32         );
33     }
34 }
35
36 }
```

Nos vemos en la próxima Guía...