

Calidad de Software

Práctica Calificada 5

Universidad Nacional de Ingeniería

Julio 2023

Instalación de Appium

Para realizar la automatización de los test de aplicaciones móviles con Appium necesitaremos lo siguiente:

- NodeJS (npm)
- UiAutomator2 Driver
- Appium Inspector
- Android Studio (Android SDK, Java JDK)
- Python

Instalación de Appium

Para realizar la instalación de Appium debemos ejecutar los siguientes comandos:

- `npm i -g appium@next`
- `appium driver install uiautomator2`

Instalación de Appium

También es de utilidad descargar Appium Inspector, el cuál nos servirá para iniciar una sesión, utilizar la aplicación y recorrer sus elementos.

Instalación de Appium

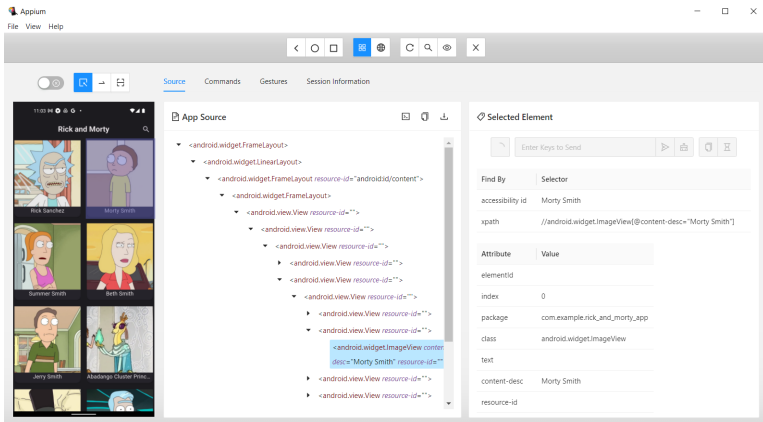


Figura: Appium Inspector ¹

¹<https://github.com/appium/appium-inspector>

Por último, es necesario declarar algunas variables de entorno ² del sistema tales como:

- ANDROID_HOME
- ANDROID_SDK_ROOT

²<https://appium.io/docs/en/2.0/>

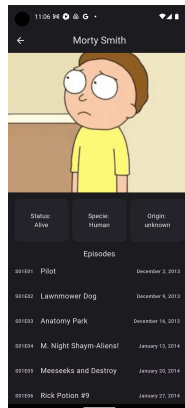
Aplicación utilizada

- Se utilizó una aplicación diseñada en Flutter.
- La aplicación consiste en una página de búsqueda de información sobre los personajes de la serie “Rick & Morty”.
- Para este fin, la aplicación se conecta a una API con la información de estos personajes.

Aplicación utilizada



(a) Pantalla principal



(b) Pantalla personaje

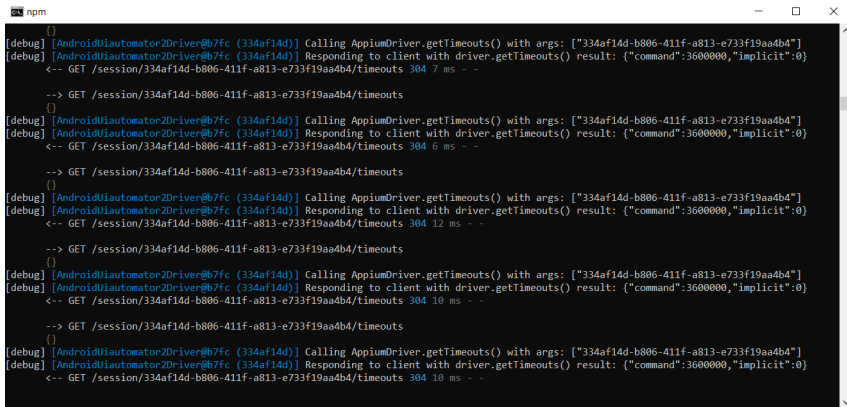
Figura: Aplicación móvil

Configuración del entorno

- Es necesario levantar la aplicación, con las dependencias que la misma requiera.
- Como opción alternativa, puede utilizarse solo la APK.
- Independientemente de la opción, la aplicación debe estar ejecutándose en un emulador o dispositivo físico.

Configuración del entorno

- Ejecutar appium en la consola.



```
npm
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Calling AppiumDriver.getTimeouts() with args: ["334af14d-b806-411f-a813-e733f19aa4b4"]
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Responding to client with driver.getTimeouts() result: {"command":3600000,"implicit":0}
<-- GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts 7 ms - -

--> GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Calling AppiumDriver.getTimeouts() with args: ["334af14d-b806-411f-a813-e733f19aa4b4"]
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Responding to client with driver.getTimeouts() result: {"command":3600000,"implicit":0}
<-- GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts 304 6 ms - -

--> GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Calling AppiumDriver.getTimeouts() with args: ["334af14d-b806-411f-a813-e733f19aa4b4"]
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Responding to client with driver.getTimeouts() result: {"command":3600000,"implicit":0}
<-- GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts 304 12 ms - -

--> GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Calling AppiumDriver.getTimeouts() with args: ["334af14d-b806-411f-a813-e733f19aa4b4"]
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Responding to client with driver.getTimeouts() result: {"command":3600000,"implicit":0}
<-- GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts 304 10 ms - -

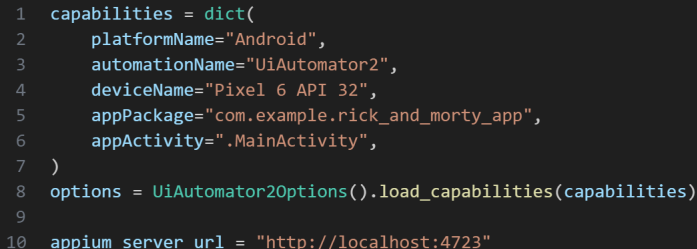
--> GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Calling AppiumDriver.getTimeouts() with args: ["334af14d-b806-411f-a813-e733f19aa4b4"]
[debug] [AndroidUiautomator2Driver@b7fc (334af14d)] Responding to client with driver.getTimeouts() result: {"command":3600000,"implicit":0}
<-- GET /session/334af14d-b806-411f-a813-e733f19aa4b4/timeouts 304 10 ms - -
```

Figura: Appium CMD

- Por último, ejecutar el script de Python donde se encuentran los tests.

Tests con Appium

- Para empezar con el script de Python, primero es necesario declarar una configuración.


A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a Python script for configuring Appium capabilities and options.

```
1 capabilities = dict(  
2     platformName="Android",  
3     automationName="UiAutomator2",  
4     deviceName="Pixel 6 API 32",  
5     appPackage="com.example.rick_and_morty_app",  
6     appActivity=".MainActivity",  
7 )  
8 options = UiAutomator2Options().load_capabilities(capabilities)  
9  
10 appium_server_url = "http://localhost:4723"
```

Figura: Configuración

Tests con Appium

- Luego, creamos una clase donde estarán incluidas funciones de levantamiento y fin, así como funciones donde se declaran los procesos para los tests.

A screenshot of a code editor with a dark background and light-colored text. The code defines a Python class named TestAppium that inherits from unittest.TestCase. It includes two methods: setUp, which initializes a webdriver.Remote object with appium_server_url and options, and tearDown, which calls quit() on the driver if it exists. The code is numbered from 1 to 7 on the left side of the editor.

```
1 class TestAppium(unittest.TestCase):
2     def setUp(self) -> None:
3         self.driver = webdriver.Remote(appium_server_url, options=options)
4
5     def tearDown(self) -> None:
6         if self.driver:
7             self.driver.quit()
```

Figura: Clase TestAppium

- Finalmente, se ejecutan todos los tests con la siguiente línea de código

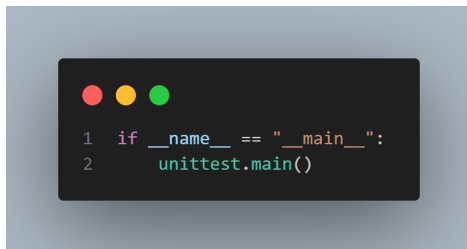
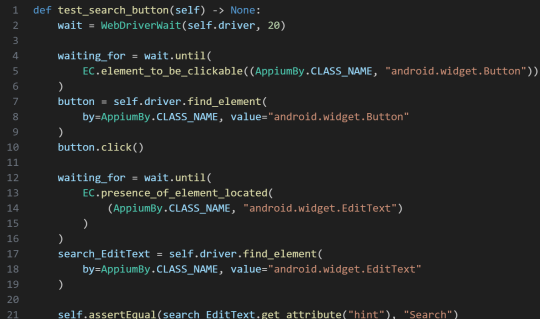


Figura: Ejecución de los tests

Los tests realizados en la aplicación fueron los siguientes:

- 1 Evalúa que el título de la aplicación esté correcto.
- 2 Evalúa que el título de una de las tarjetas coincida con el de la página redirigida.
- 3 Evalúa el correcto funcionamiento de la barra de búsqueda.
- 4 Realiza una búsqueda y evalúa que la página redirigida sea la correcta.
- 5 Realiza un scroll, ingresa a una de las tarjetas y evalúa que la pagina redirigida sea la correcta.

Para mostrar un ejemplo de código, se muestra el test número 3:

A screenshot of a code editor with a dark background and light-colored text. The code is a Python function named `test_search_button` that performs a series of actions on a web driver. It starts by waiting for a button to be clickable, then clicks it. Next, it waits for an edit text field to be present, finds it, and finally asserts that its hint attribute is "Search". The code is numbered from 1 to 21.

```
1 def test_search_button(self) -> None:
2     wait = WebDriverWait(self.driver, 20)
3
4     waiting_for = wait.until(
5         EC.element_to_be_clickable((AppiumBy.CLASS_NAME, "android.widget.Button"))
6     )
7     button = self.driver.find_element(
8         by=AppiumBy.CLASS_NAME, value="android.widget.Button"
9     )
10    button.click()
11
12    waiting_for = wait.until(
13        EC.presence_of_element_located(
14            (AppiumBy.CLASS_NAME, "android.widget.EditText")
15        )
16    )
17    search_EditText = self.driver.find_element(
18        by=AppiumBy.CLASS_NAME, value="android.widget.EditText"
19    )
20
21    self.assertEqual(search_EditText.get_attribute("hint"), "Search")
```

Figura: Test 3