Contenido

[1. Autores del trabajo, planificación y entrega 2](#_Toc513497068)

[1.1 Autores 2](#_Toc513497069)

[1.2 Planificación 2](#_Toc513497070)

[1.3 Entrega 2](#_Toc513497071)

[2. Requisitos del prototipo a implementar 3](#_Toc513497072)

[2.1 Requisitos funcionales 3](#_Toc513497073)

[2.2 Otros requisitos 3](#_Toc513497074)

[3. Criterios de comparación en la implementación 4](#_Toc513497075)

[4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Appium 6](#_Toc513497076)

[4.1 Documentación de diseño 6](#_Toc513497077)

[4.2 Documentación de construcción 7](#_Toc513497078)

[4.3 Documentación de pruebas 9](#_Toc513497079)

[4.4 Documentación de instalación 9](#_Toc513497080)

[4.5 Manual de usuario 11](#_Toc513497081)

[5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Espresso 16](#_Toc513497082)

[5.1 Documentación de diseño 16](#_Toc513497083)

[5.2 Documentación de construcción 16](#_Toc513497084)

[5.3 Documentación de pruebas 16](#_Toc513497085)

[5.4 Documentación de instalación 19](#_Toc513497086)

[5.5 Manual de usuario 22](#_Toc513497087)

[6. Comparación de las dos implementaciones 27](#_Toc513497088)

[6.1 Evaluación de los criterios en la implementación usando la tecnología A 27](#_Toc513497089)

[6.2Evaluación de los criterios en la implementación usando la tecnología B 30](#_Toc513497090)

[7. Comparación de la implementación de las tecnologías 30](#_Toc513497091)

[8. Conclusiones 33](#_Toc513497092)

# 1. Autores del trabajo, planificación y entrega

## 1.1 Autores

Grupo M1

Amir Sholkami Berube - Amir95sb(coordinador)

Javier Ballesteros García - JavierBallesteros5

Ismael Sainz Maza Jiménez - Ismaeel19

Stephany Susan Vela Vellachich - stephanyvela

## 1.2 Planificación



Ya que este trabajo cuenta 3 Puntos de la nota final de la asignatura, cada uno ha sido asignado trabajos equivalentes o mayores a 45 horas totales.

https://app.teamweek.com/#pg/1g82G63vww4N07cj0sN0WsaZLnDfTiuz

## 1.3 Entrega

https://github.com/Amir95sb/TG3

# 2. Requisitos del prototipo a implementar

## 2.1 Requisitos funcionales

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| RF01 | Testing de Botones |
| RF02 | Testing de introducción de texto |
| RF03 | Guardar Procedimientos Test |

## 2.2 Otros requisitos

| **REQ.** | **DESCRIPCIÓN** |
| --- | --- |
| R01 | Testing de interfaz de usuario |
| R02 | Velocidad de pruebas >1 minuto |
|  |  |

# 3. Criterios de comparación en la implementación

***3.1 Interfaz de Usuario***

Nombre del criterio: Interfaz de Usuario.

Descripción: En este criterio comparamos la facilidad de la interfaz de usuario con respecto a otras aplicaciones. Más tarde veremos esa comparación de nuestros dos softwares frente a frente

Tipo de valor: Texto.

***3.2 Tiempo de Aprendizaje***

Nombre del criterio: Tiempo de Aprendizaje.

Descripción: En este criterio valoramos el tiempo que hemos tardado desde que nosotros arrancamos desde el momento que arrancamos nuestro software y podemos empezar a realizar ese proyecto. También introducimos el tiempo que le hemos dedicado a la adquisición de conocimientos de aprendizaje del software.

Tipo de valor: Numérico (horas).

***3.3 Tiempo de Configuración***

Nombre del criterio: Tiempo de Configuración.

Descripción: Incluimos el tiempo que hemos empleado en la configuración de nuestro software para utilizarlo. Aquí debemos incluir la instalación.

Tipo de valor: Numérico (horas).

***3.4 SS.OO Útiles***

Nombre del criterio: SS.OO Útiles.

Descripción: Diremos los SS.OO que hemos usado en la instalación y realización del proyecto. En este punto introducimos también la dificultad que hemos tenido.

Tipo de valor: Texto.

***3.5 Horas Empleadas en el Desarrollo***

Nombre del criterio: Horas Empleadas en el Desarrollo.

Descripción: Decimos el número de horas que se hemos tardado desde que empezamos en su momento el desarrollo hasta tener un proyecto viable al menos mínimamente.

Tipo de valor: Numérico (horas).

***3.6 Herramientas Utilizadas en el Desarrollo***

Nombre del criterio: Herramientas Utilizadas en el Desarrollo.

Descripción: Vemos las herramientas que hemos usado pues nos ayudaban a realizar ese mínimo de nuestro proyecto.

Tipo de valor: Texto.

***3.7 Costes del software***

Nombre del criterio: Costes del software

Descripción: Decimos cuanto a costado el software con todas las herramientas que también usamos para hacerlo funcionar.

Tipo de valor: Valor económico.

***3.8 Requisitos en el ordenador***

Nombre del criterio: Requisitos en el ordenador

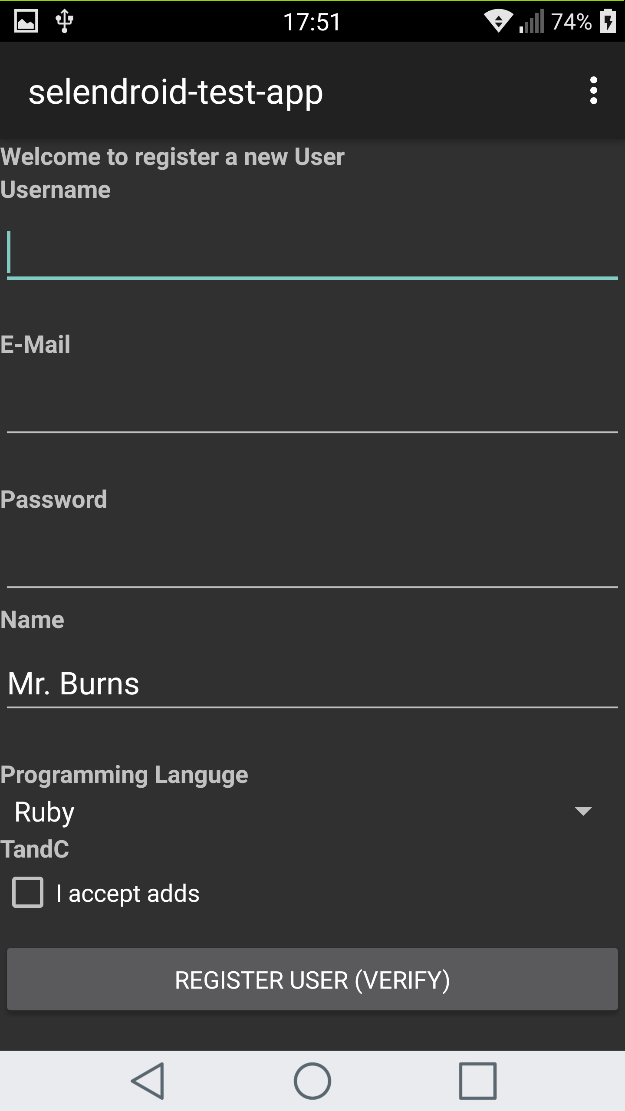
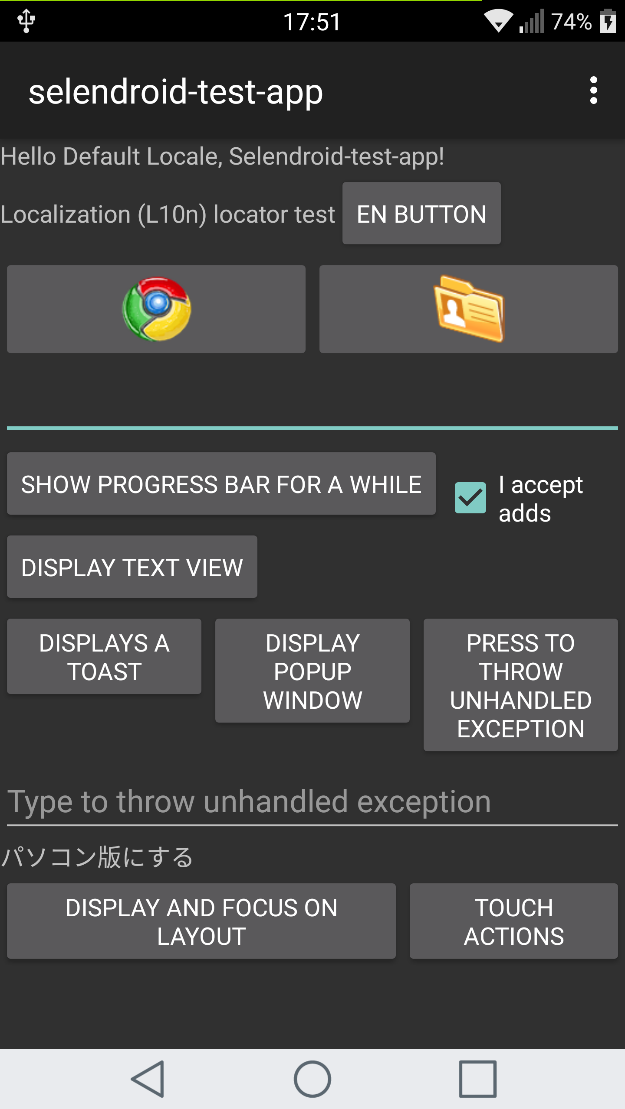
Descripción: Decimos cuales son los requisitos para nuestro ordenador o dispositivo para que nuestro software pueda funcionar a pleno rendimiento.

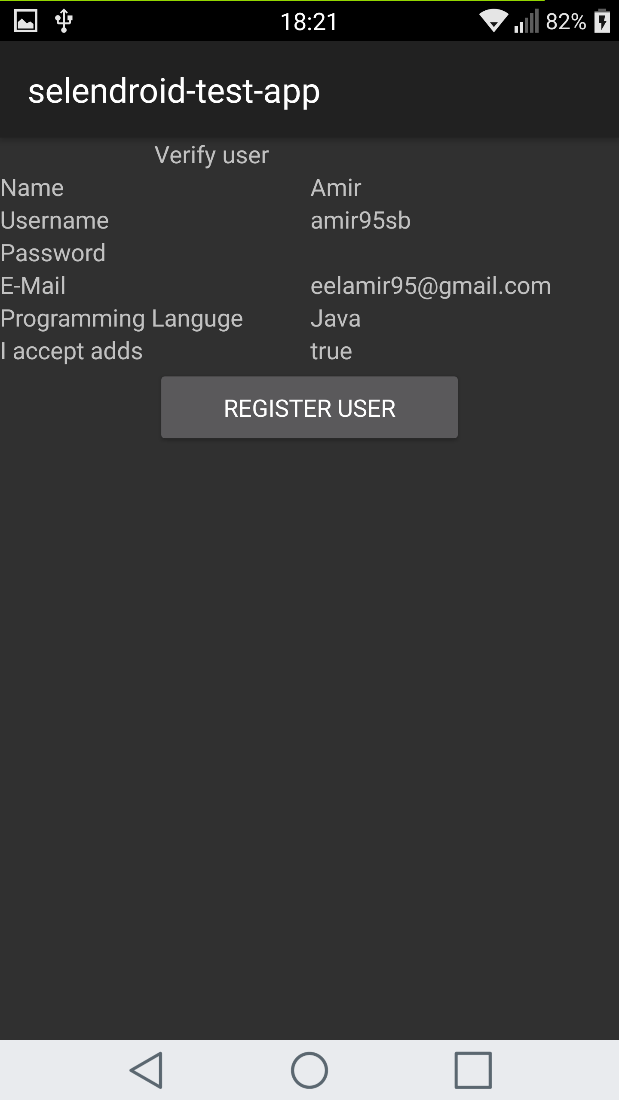
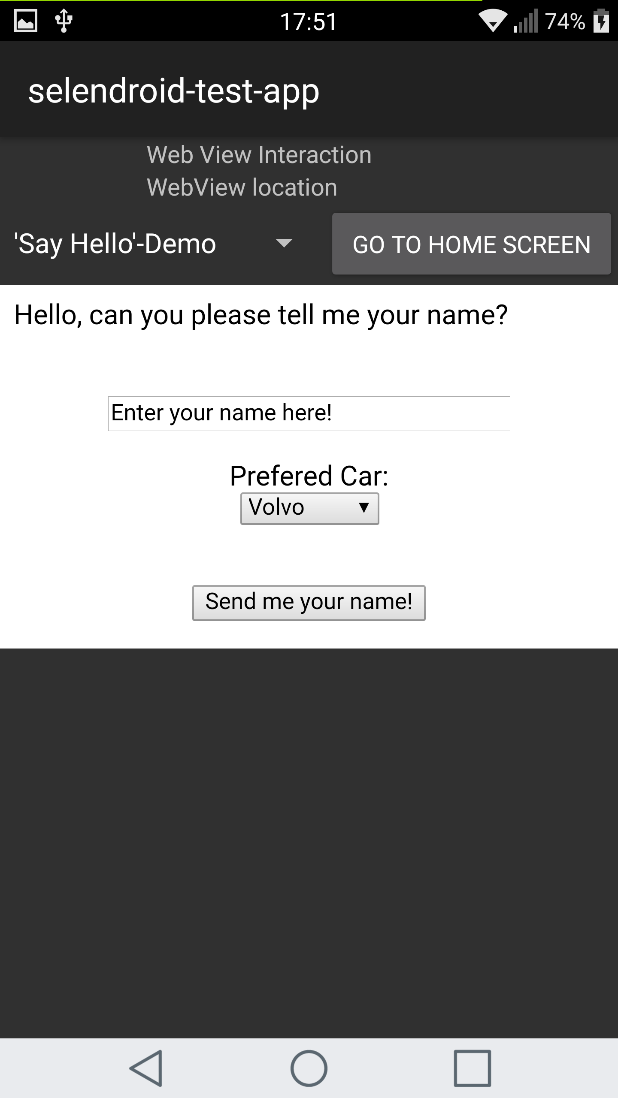
Tipo de valor: Texto.

# 4. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Appium

## 4.1 Documentación de diseño

El prototipo usado en este documento es una aplicación sencilla pero completa, llamada Selendroid test app, la cual ha sido diseñada para ejecutar y ver los resultados a pruebas desde cualquier herramienta de testing.





La interfaz del usuario es muy básica, ya que solo queremos probar la funcionalidad de la app, y las animaciones han sido desactivadas para evitar errores.

Las miniaturas usadas se pueden encontrar en la carpeta del código fuente llamada:

\selendroid-master\selendroid-test-app\res

## 4.2 Documentación de construcción

La construcción de la app es básica, como mostramos en el siguiente código fuente:

1. **public** **class** UserRegistrationTest **extends** BaseAndroidTest {
2. @Test
3. **public** void assertUserAccountCanRegistered() throws Exception {
4. UserDO user =
5. **new** UserDO("u**$erNAme**", "me@myserver.com", "mySecret", "Selendroid User",
6. PreferedProgrammingLanguage.JAVA\_SCRIPT);
7. registerUser(user);
8. verifyUser(user);
9. }
11. **private** void registerUser(UserDO user) throws Exception {
12. WebElement button = driver().findElement(By.id("startUserRegistration"));
14. button.click();
16. WebElement username = driver().findElement(By.id("inputUsername"));
18. username.sendKeys(user.getUsername());
19. WebElement nameInput = driver().findElement(By.id("inputName"));
20. [Assert](http://www.php.net/assert).assertEquals(nameInput.[getText](http://www.php.net/gettext)(), "Mr. Burns");
21. nameInput.clear();
22. nameInput.sendKeys(user.getName());
24. driver().findElement(By.id("inputEmail")).sendKeys(user.getEmail());
25. driver().findElement(By.id("inputPassword")).sendKeys(user.getPassword()); //se introduce la contraseña del usuario
26. try {
27. nameInput.submit();
28. [Assert](http://www.php.net/assert).fail("submit is not supported by SelendroidNativeDriver");
29. } catch (WebDriverException e) {
30. // expected behavior
31. }

Este código es el del formulario, en el cual se rellenan los datos del usuario como el ID, email, o contraseña.

1. <resources>
3. <string name="hello">Hello Default Locale, Selendroid-test-app!</string>
4. <string name="button">EN Button</string>
5. <string name="app\_name">Selendroid-test-app</string>
6. <string name="search\_hint">Search for users</string>
7. <string name="welcomeRegisterUser">Welcome to register a **new** User</string>
8. <string name="verifyNewUser">Register User (verify)</string>
9. <string name="label\_password">Password</string>
10. <string name="label\_username">Username</string>
11. <string name="label\_email">E-Mail</string>
12. <string name="label\_name">Name</string>
13. <string name="label\_preferedProgrammingLanguage">Programming Languge</string>
14. <string name="prompt\_preferedProgrammingLanguage">Programming Languge</string>
15. <string name="prompt\_webdriverHtmlTestFile">Webdriver Test File</string>
16. <string name="label\_acceptAdds">I accept adds</string>
17. <string name="label\_conditions">TandC</string>
19. <string-[array](http://www.php.net/array) name="arrays\_preferedProgrammingLanguage">
20. <item>Ruby</item>
21. <item>PHP</item>
22. <item>Scala</item>
23. <item>Python</item>
24. <item>Javascript</item>
25. <item>Java</item>
26. <item>C++</item>
27. <item>C*#</item>*
28. </string-array>
30. </resources>

En este código están incluidos todos los “strings” usados por la app.

Estarán subidos tanto el código fuente del proyecto como la .APK en el repositorio GitHub.

## 4.3 *Documentación de pruebas*

Hemos realizado tres casos de prueba.

- En el primero, realizamos acciones correctamente. Probamos los botones, la introducción de texto y después guardamos los procedimientos, es decir, los requisitos funcionales. Todo es válido. La interfaz de usuario funciona correctamente y al automatizar las pruebas están se recrean en menos de un minuto.

- En el segundo caso, realizamos una excepción pulsando el botón indicado para ello. Este para la aplicación y queda registrado en las acciones. Cuando automatizamos el test, nos indica que todo está correcto porque esa era la función del botón, pero nosotros queríamos que diera error.

- En el tercer caso, probamos otra forma de realizar la excepción. En el segundo cuadro de texto debe darse una al escribir. De nuevo, la aplicación se para rápidamente. Sin embargo, cuando automatizamos el test nos indica que todo está correcto porque esa era la función del cuadro, pero nosotros deseábamos que la excepción se reflejara de forma negativa en los resultados.

En el punto 4.5 vienen las acciones realizadas paso a paso con capturas de pantalla.

## 4.4 Documentación de instalación

Para que Appium funcione correctamente en Windows debemos tener instalado NodeJs, Java Development Kit (JDK), Paquetes de Android SDK y ADB, además de los drivers de los dispositivos que usaremos, ya que Appium los necesita para conectarse y probar la aplicación. En caso de no poseer un dispositivo se requiere un emulador.

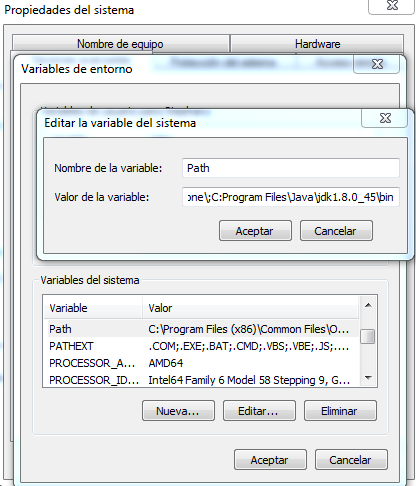
Para saber la versión de node.js (se recomienda tener al menos la v4.4.7):

node -v

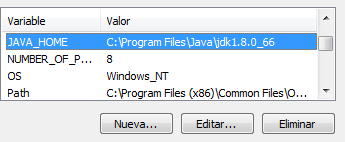
Para la versión de java (se recomienda tener a partir de la v1.8.0):

java –version

En caso de que desde la cmd no se permita ver la versión de java, deberemos indicar el camino. Para ello iremos a Panel de Control > Sistema y Seguridad > Sistema y entraremos a la pestaña de Configuración avanzada del sistema.

Desde opciones avanzadas daremos click a Variables de entorno y añadiremos a Path el camino al bin de nuestro software Java, pero sin quitar los demás ya que son necesarios.

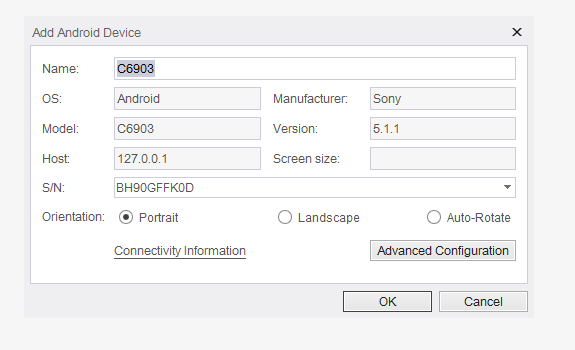
Además, añadiremos la variable JAVA\_HOME:

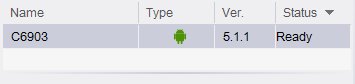


Tras instalar todos los drivers y softwares adecuados podemos concluir la instalación.

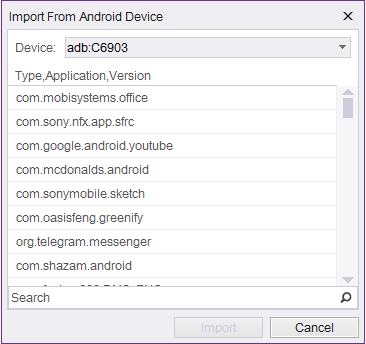
## 4.*5 Manual de usuario*

1. Se añade un dispositivo haciendo click en “Añadir nuevo dispositivo” y configuramos, en este caso, el móvil.





2. Importamos la aplicación deseada haciendo click en “Import”. Escogemos importar desde el dispositivo y seleccionamos la aplicación deseada.



3. La propia interfaz nos pide que añadamos comandos. Primero añadiremos el comando Launch. Después seleccionaremos la app que vamos a probar. Es importante añadir que no se debe interactuar desde el móvil. Debe hacerse desde la ventana que abre appium o el test no funcionará.

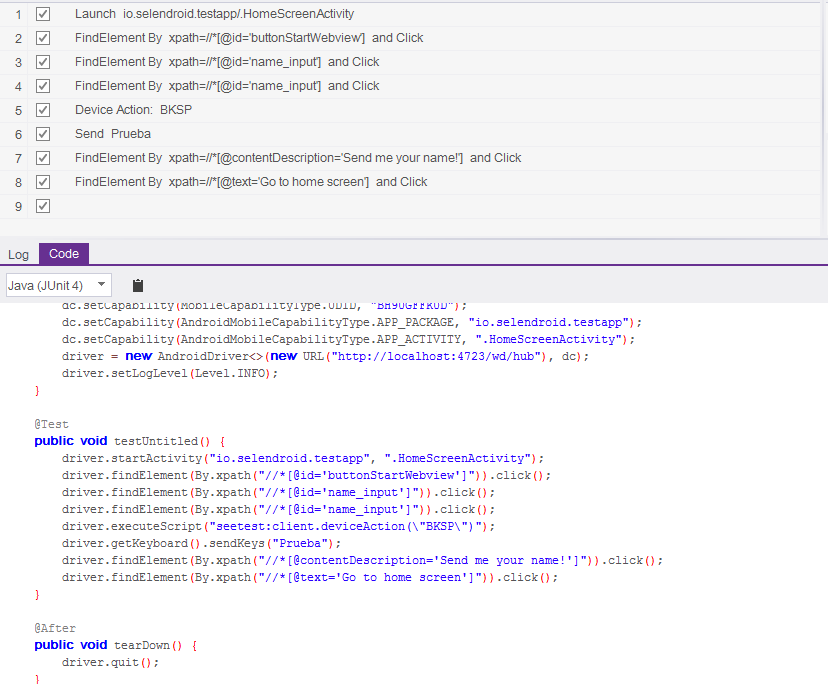


4. A continuación hacemos click en el botón rojo para grabar las acciones a probar:



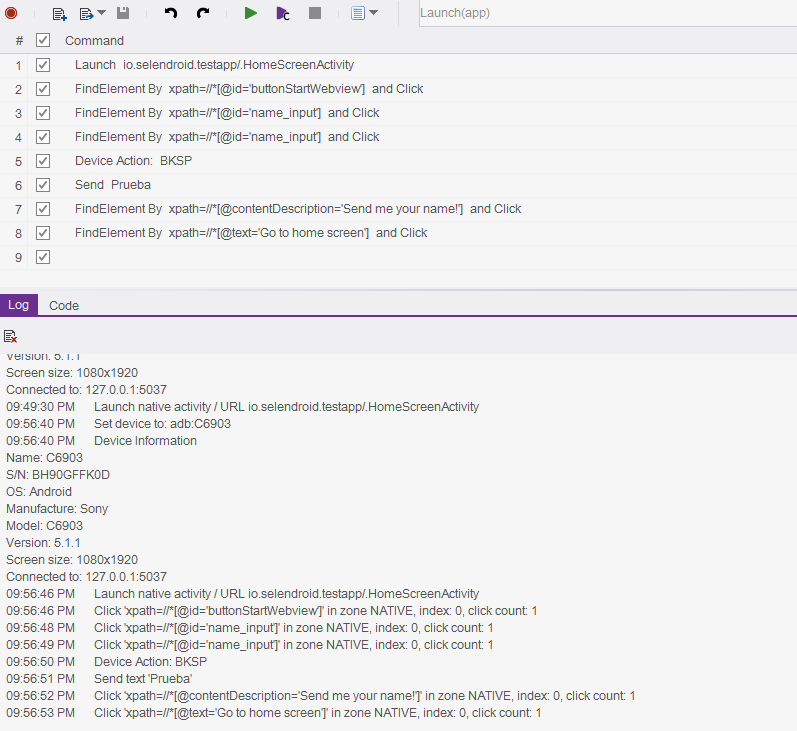
Se nos abrirá una ventana con nuestro dispositivo y podremos interactuar con él. En nuestro caso probamos unos botones y la entrada del texto.

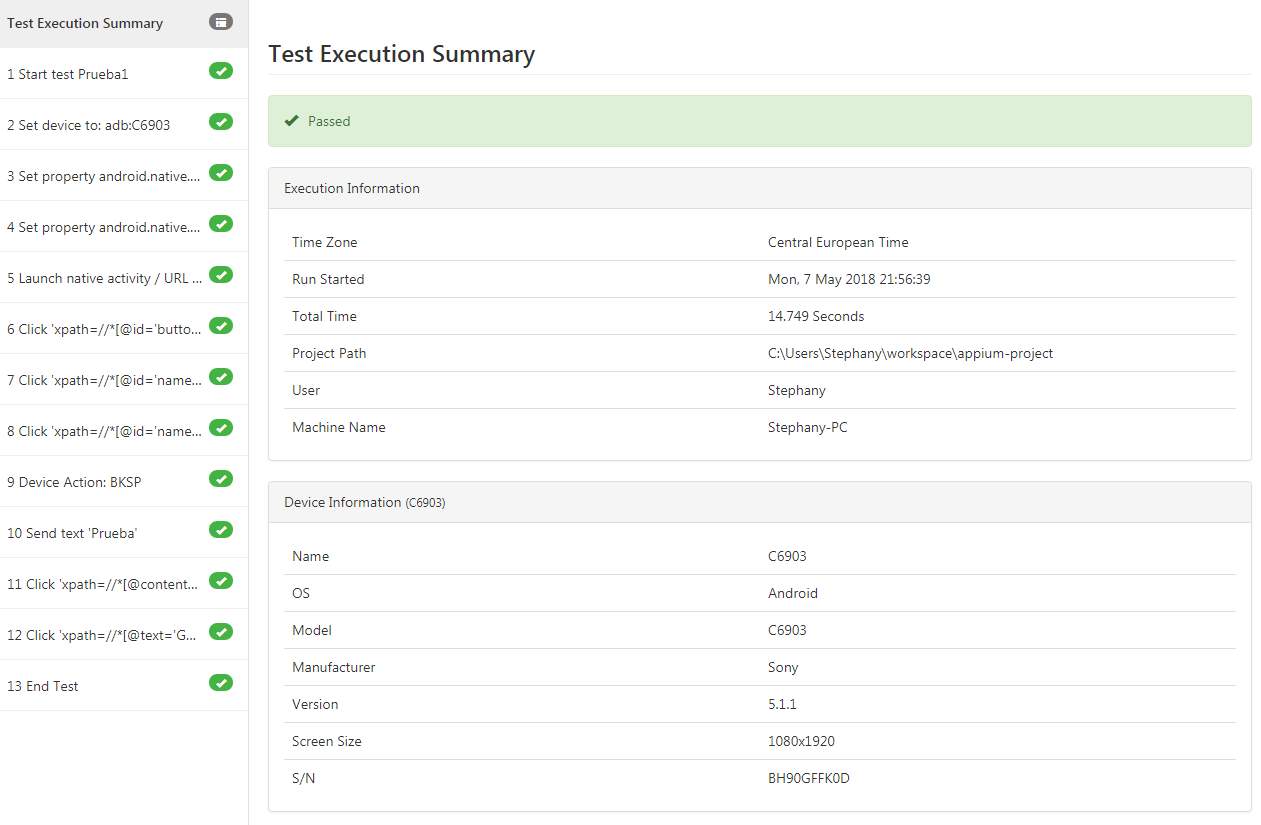
Después, donde antes había un botón rojo, habrá uno azul. Damos click para parar la grabación, y obtendremos nuestro código y las acciones.



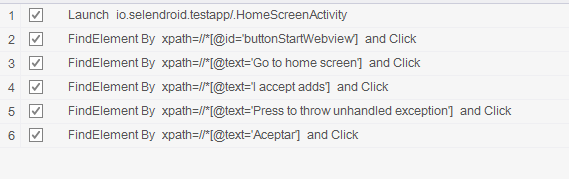
5. Hacemos click en play (botón verde) para ver como el programa realiza la prueba automáticamente. Se habré nuevamente la pestaña con nuestro dispositivo y realiza las acciones anteriores. Tras eso, nos indica el log en el propio programa.

Además, nos abre una pestaña en nuestro navegador con las actividades realizadas y si se han dado con éxito.

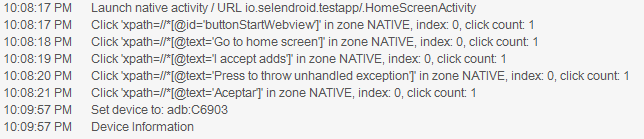




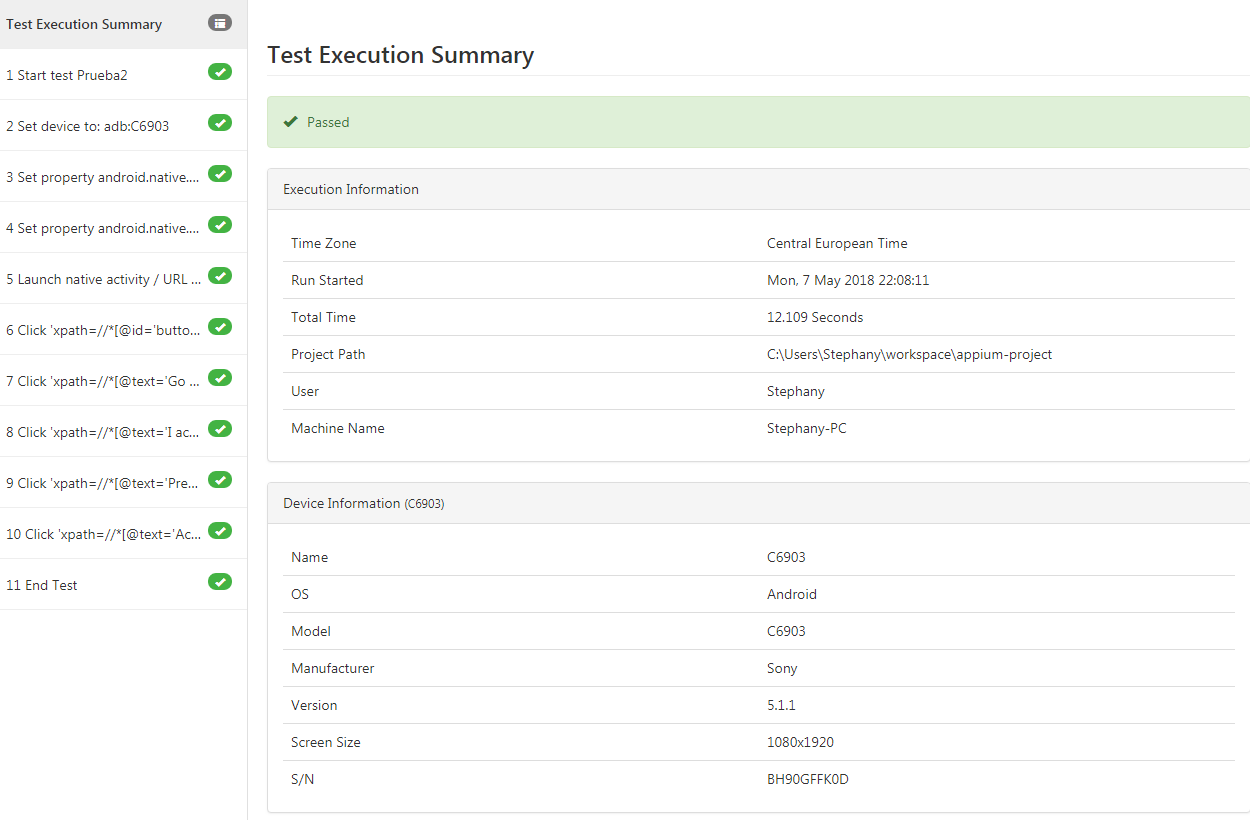
6. Repetimos los pasos anteriores lanzando una excepción.



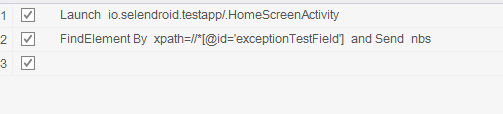
Y al probar el test nos da los siguientes resultados:



El botón está preparado para dar una excepción por lo que nos dice que la prueba es un éxito.

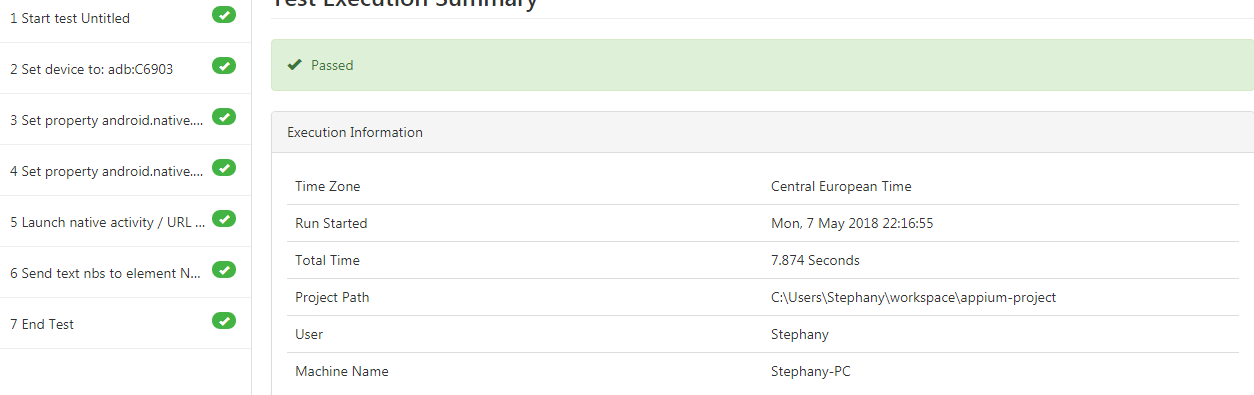


7. Repetimos los pasos anteriores lanzando una excepción de forma diferente. En el segundo recuadro de escritura, al escribir debe pararse el programar por una excepción. Al realizar la prueba, se escribe en Appium con éxito.



Pero cuando le pedimos que reproduzca la prueba, nos pasa como en el ejemplo anterior. Como el campo a cumplido su función (lanzar una excepción), nos lo toma como bueno.





# 5. Proyecto de implementación de un prototipo del sistema utilizando la tecnología Espresso

## 5.1 Documentación de diseño

El prototipo usado es exactamente el mismo que ha sido usado en la tecnología A (Appium)



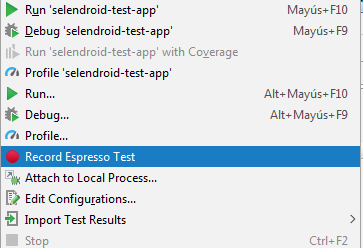
## 5.2 Documentación de construcción

El código fuente, ya está explicado en el apartado 4.2, pero cabe decir que ha sido modificado con Android Studio para su correcto funcionamiento, ya que algunos módulos estaban desactualizados, y al actualizarlos, salieron a la superficie problemas debido a funciones no admitidas en los módulos actualizados nuevos.

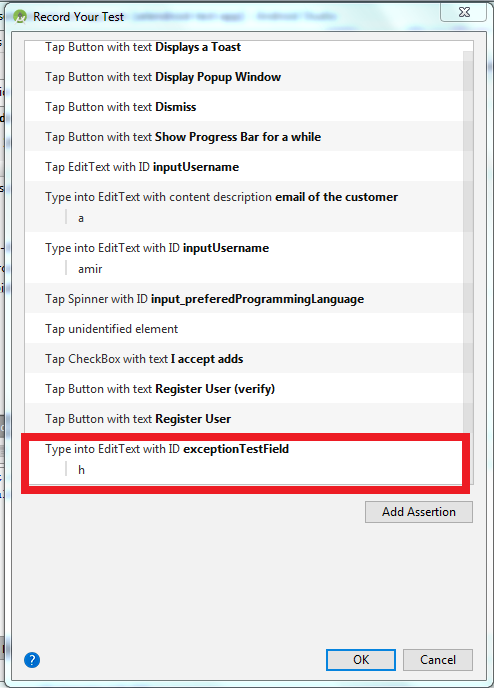
## 5.3 Documentación de pruebas

Se han realizado dos casos de prueba, un caso de prueba fallido:

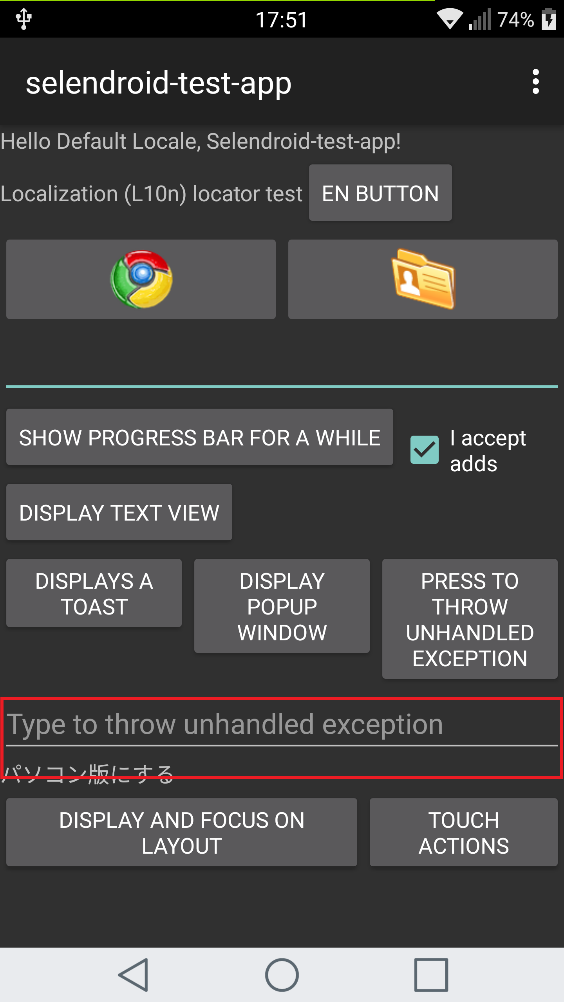
Para demostrar el correcto funcionamiento de Espresso, hemos realizado una prueba fallida, no porque la aplicación este mal programada, sino porque se incluye una función que lanza un error, para demostrar que es lo que sucede ante estos casos.

Para crear la prueba, se ha utilizado una herramienta llamada incluida en Android Studio: 

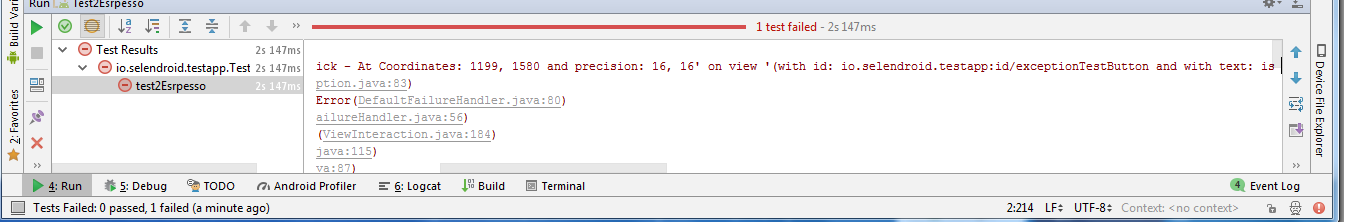
Con la cual la aplicación se inicia en el dispositivo seleccionado, y graba la serie de acciones que nosotros realizamos en la aplicación desde el dispositivo, transformándolo directamente en código de test:



Como se aprecia en la imagen, la última acción grabada es en el campo llamado “exceptionTestField” el cual lanza una excepción al ser escrito cualquier carácter en el campo.



Al realizar la prueba, el resultado es fallido, indicando que el fallo está en la línea de código la cual lanza la excepción. Los resultados están incluidos en el zip llamado PrototipoTecnologiaB\_final.rar



Para realizar una prueba Válida, es decir, sin fallos, hemos realizado los mismos pasos, excepto sin apretar el botón llamado “Press to throw unhandled exception” ni escribir en el campo “Type to throw unhandled exception” dando lugar a esta secuencia de acciones:

-(R.id.***showToastButton***), *withText*(**"Displays a Toast"**)

*-allOf*(*withId*(R.id.***topLevelElementTest***), *withText*(**"Display and focus on layout"**)

*-allOf*(*withId*(R.id.***visibleButtonTest***), *withText*(**"Display text view"**)

*-allOf*(*withId*(R.id.***waitingButtonTest***), *withText*(**"Show Progress Bar for a while"**)

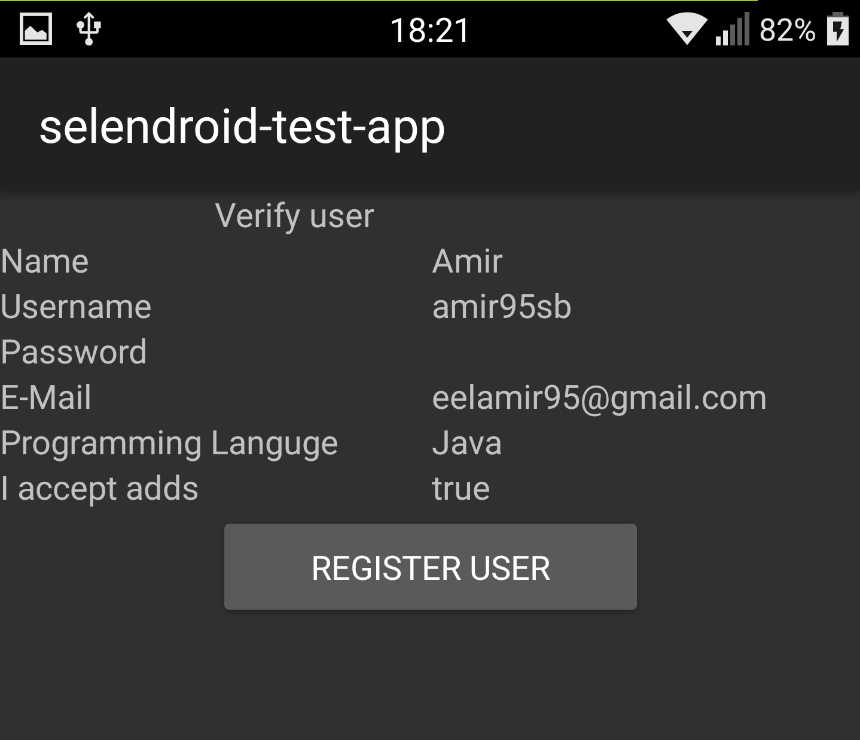
-editText.perform(*scrollTo*(), *replaceText*(**"amir"**)

-editText2.perform(*scrollTo*(), *replaceText*(**"amir95sb@gmail.com"**)

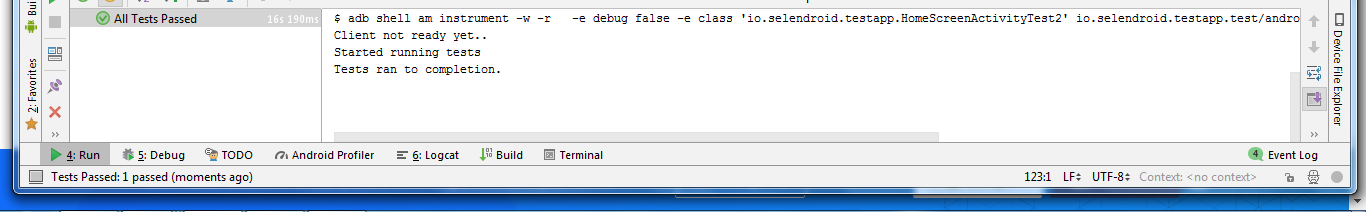
-*allOf*(*withId*(R.id.***btnRegisterUser***), *withText*(**"Register User (verify)"**)

-*allOf*(*withId*(R.id.***buttonRegisterUser***), *withText*(**"Register User"**)

Dando lugar a la siguiente pantalla:



A la hora de realizar el test, el resultado es positivo:



## 5.4 Documentación de instalación

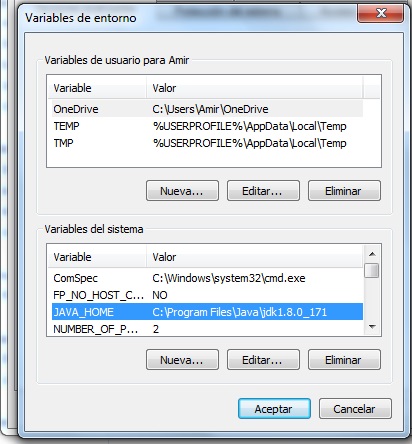
La instalación de Espresso es relativamente sencilla, ya que está integrada con la aplicación Android Studio, y el proceso es totalmente automatizado.

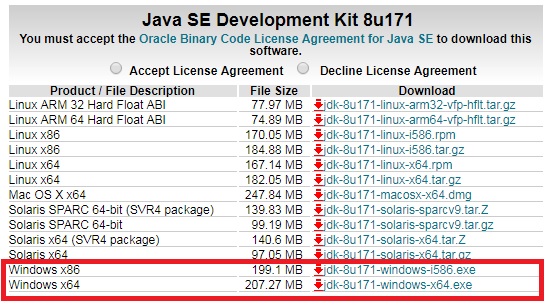
Las herramientas a descargar son las siguientes:

-Android Studio: <https://developer.android.com/studio/?hl=es-419>

-Java JDK: <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Primero sería recomendable descargar las herramientas Java JDK, ya que luego Android Studio las localizara automáticamente. De lo contrario habrá que incluir en las variables del entorno una variable llamada JAVA\_HOME con la dirección de la carpeta de instalación:

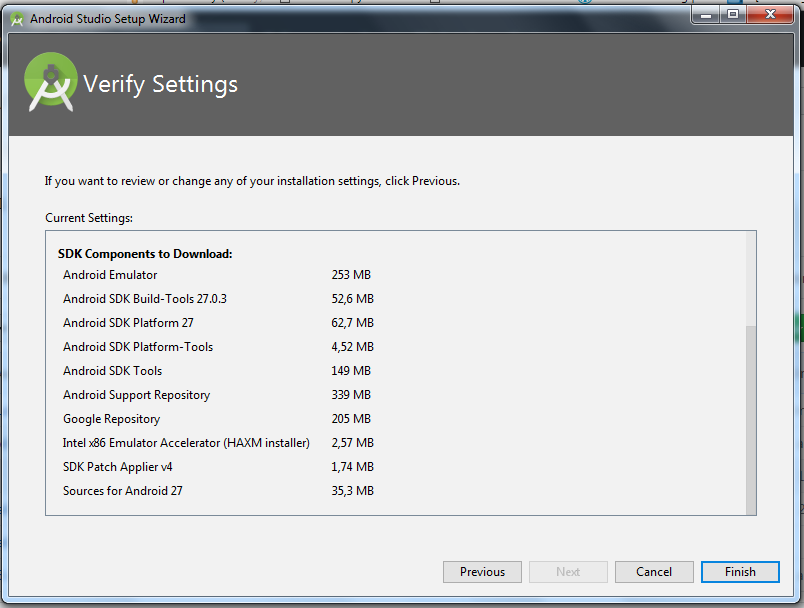




En nuestro caso seleccionamos las herramientas para Windows x64, porque es el sistema que usamos.

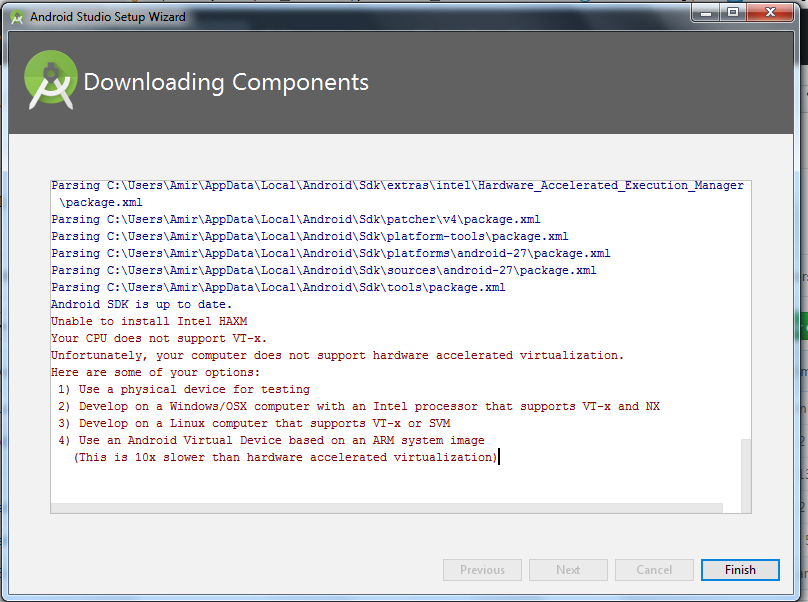
Se instala siguiendo las opciones por defecto que viene el instalador, sin tener que elegir ninguna opción.

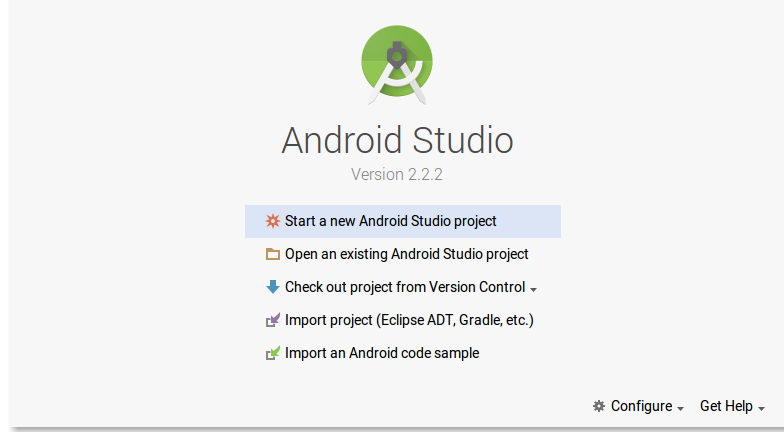
Seguimos con Android Studio, el cual también tenemos que seguir las opciones por defecto, incluidas las Android SDK.



Estas son las opciones por defecto que se deberían instalar.

En el ordenador en concreto que hemos usado, nuestra CPU no soporta dispositivos emulados, aunque no importa porque en nuestro caso hemos usado dispositivos reales.



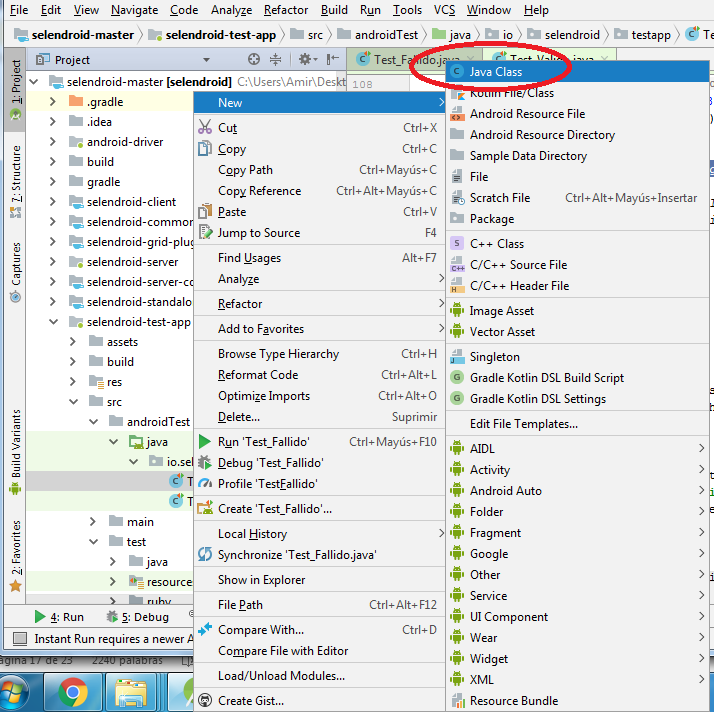


Y ya estaría todo preparado para crear tu propia aplicación, o empezar a testear tu aplicación sobre el código fuente. Se pueden usar archivos APK para crear tus test en Espresso, pero no se podrán ejecutar a no ser que tengas el código fuente.

## 5.5 Manual de usuario

Para comenzar necesitas abrir el proyecto en Android Studio en el cual este el código fuente de la aplicación en cuestión:

Una vez añadido, después de haber arreglado los errores correspondientes, podemos empezar a crear una clase Java, en la cual se escribirá el procedimiento para el test:



Para poder ejecutar pruebas en Espresso, es necesario añadir las siguientes líneas al código en el archivo gradle.build:

dependencies {

compile fileTree(dir: 'libs', include: ['\*.jar'])

testCompile 'junit:junit:4.12'

*// Android runner and rules support*

androidTestCompile 'com.android.support.test:runner:0.5'

androidTestCompile 'com.android.support.test:rules:0.5'

*// Espresso support*

androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {

exclude group: 'com.android.support', module: 'support-annotations'

})

*// add this for intent mocking support*

androidTestCompile 'com.android.support.test.espresso:espresso-intents:2.2.2'

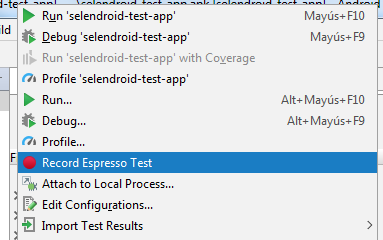
*// add this for webview testing support*

androidTestCompile 'com.android.support.test.espresso:espresso-web:2.2.2'

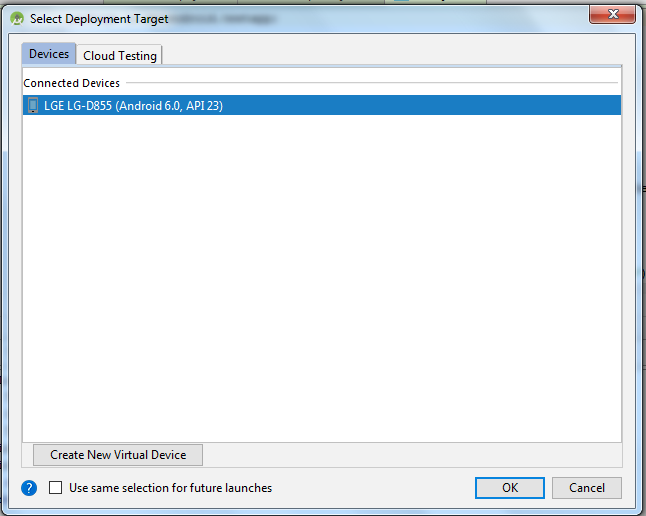
}

O usaremos la herramienta especial para grabar un test sobre el dispositivo y crear el código automáticamente:

En la pestaña RUN seleccionamos: “record Espresso test”:



Luego seleccionamos el módulo de la aplicación a grabar, en caso de haber algún fallo o incoherencia, te pedirá que lo arregles. En caso satisfactorio, se abrirá un cuadro, en el cual se activará el ADB “Android debug bridge” y seleccionaras el dispositivo en el cual se desean hacer las grabaciones de las pruebas:



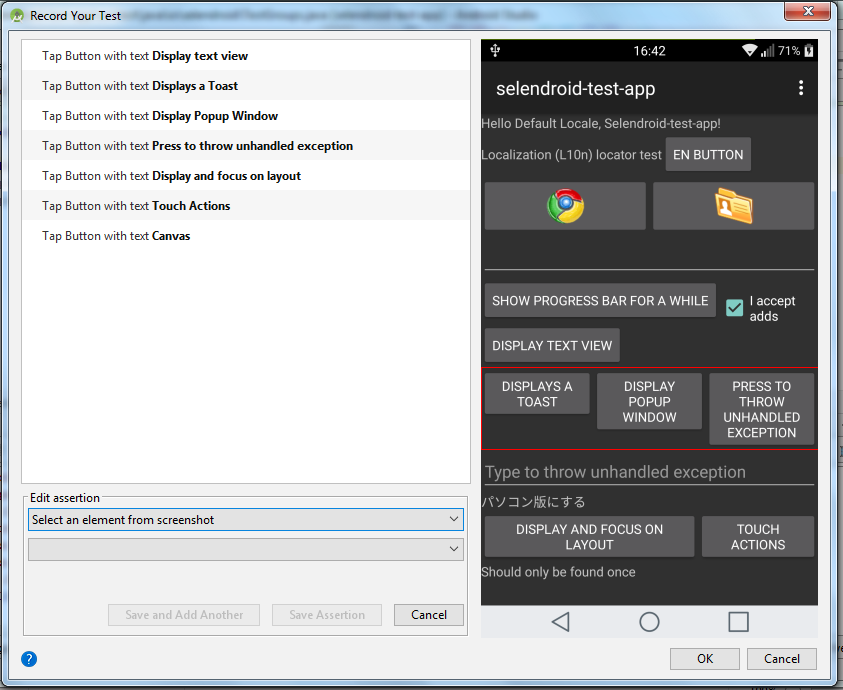
En nuestro caso usaremos un LG G3 d855, con la versión de Android 6.0.

Antes de este paso, deberás activar la Depuración USB en tu dispositivo Android, que se encuentra en la sección de herramientas de desarrollador, en los ajustes del móvil. En caso de no tenerlos activados, se deberá hacer “click” en la sección del número de construcción “build number” repetidas veces hasta que se active.

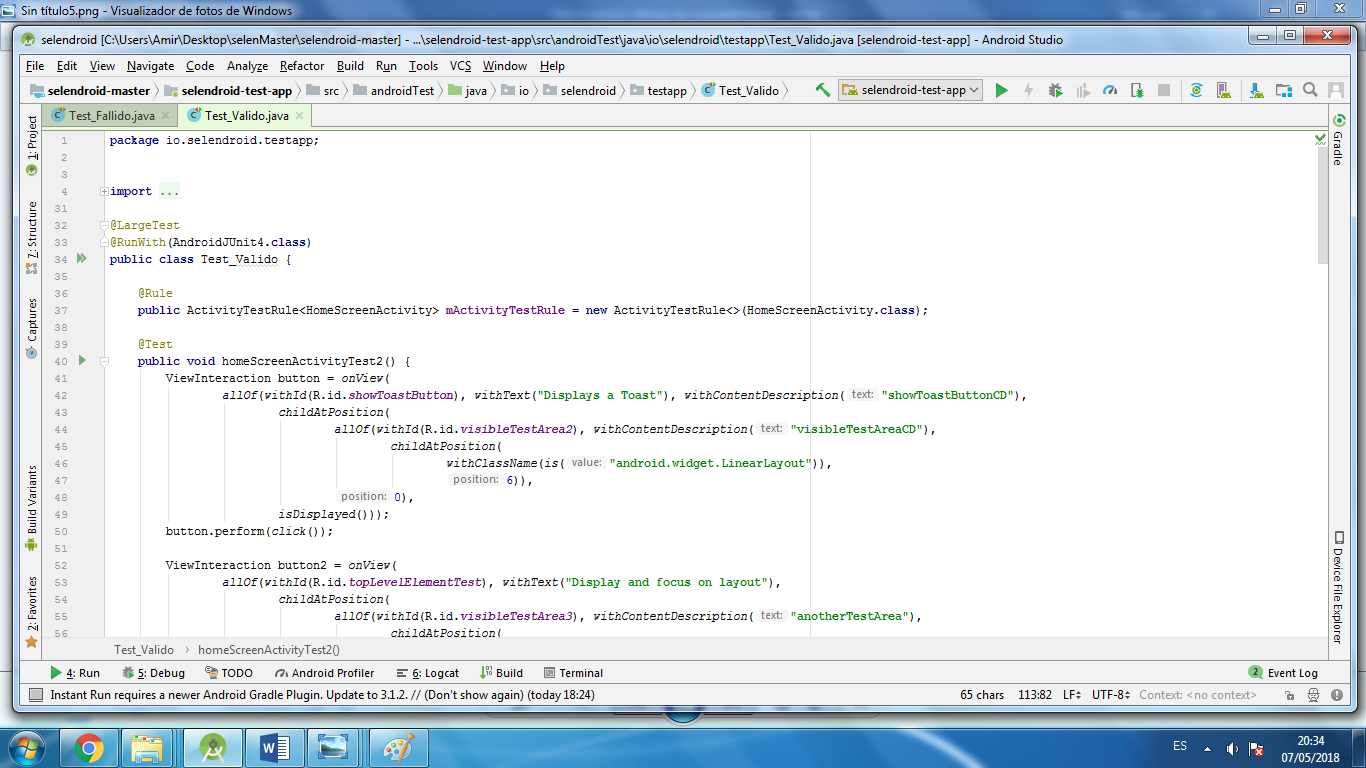
También deberás instalar los drivers de tu dispositivo Android en el ordenador.

Android studio compilara automáticamente la aplicación, generará un archivo .apk y lo instalara automáticamente en tu dispositivo. En cuanto esté instalado se abrirá directamente la aplicación, y esperara acciones para ser grabadas:

También se pueden grabar acciones desde el ordenador, es decir, al hacer click en el botón “add assertions” se mostrará por pantalla una captura de pantalla de la aplicación con todos los botones disponibles, y se grabará cualquier acción realizada, o seleccionada en el menú:



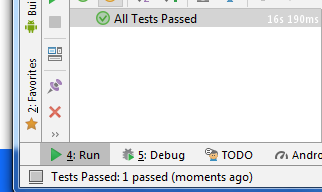
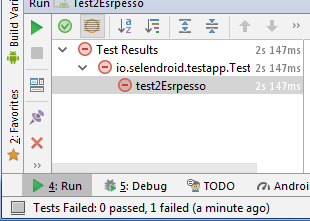
Una vez satisfecho con las acciones, se finaliza con el botón “ok”, se introduce el nombre del archivo test a crear, y se generara automáticamente el código en un archivo Java.



Android Studio añadirá los import necesarios automáticamente, y estará listo para pasar a la prueba.

Para empezar la prueba, se hace click con el botón derecho sobre el archivo de prueba generado, y sobre “Run”. Se pedirá otra vez el dispositivo al que se desea hacer la prueba, y se repetirá el proceso de compilar e instalar la app en el dispositivo móvil.

En este momento, se ejecutará la aplicación y Android studio realizará todas las acciones a ejecutar, y finalizará con un resultado positivo, o fallido:



En el log se indicarán los fallos ocurridos, y el tiempo de ejecución, además de las líneas de código en los cuales han sucedido fallos.

# 6. Comparación de las dos implementaciones

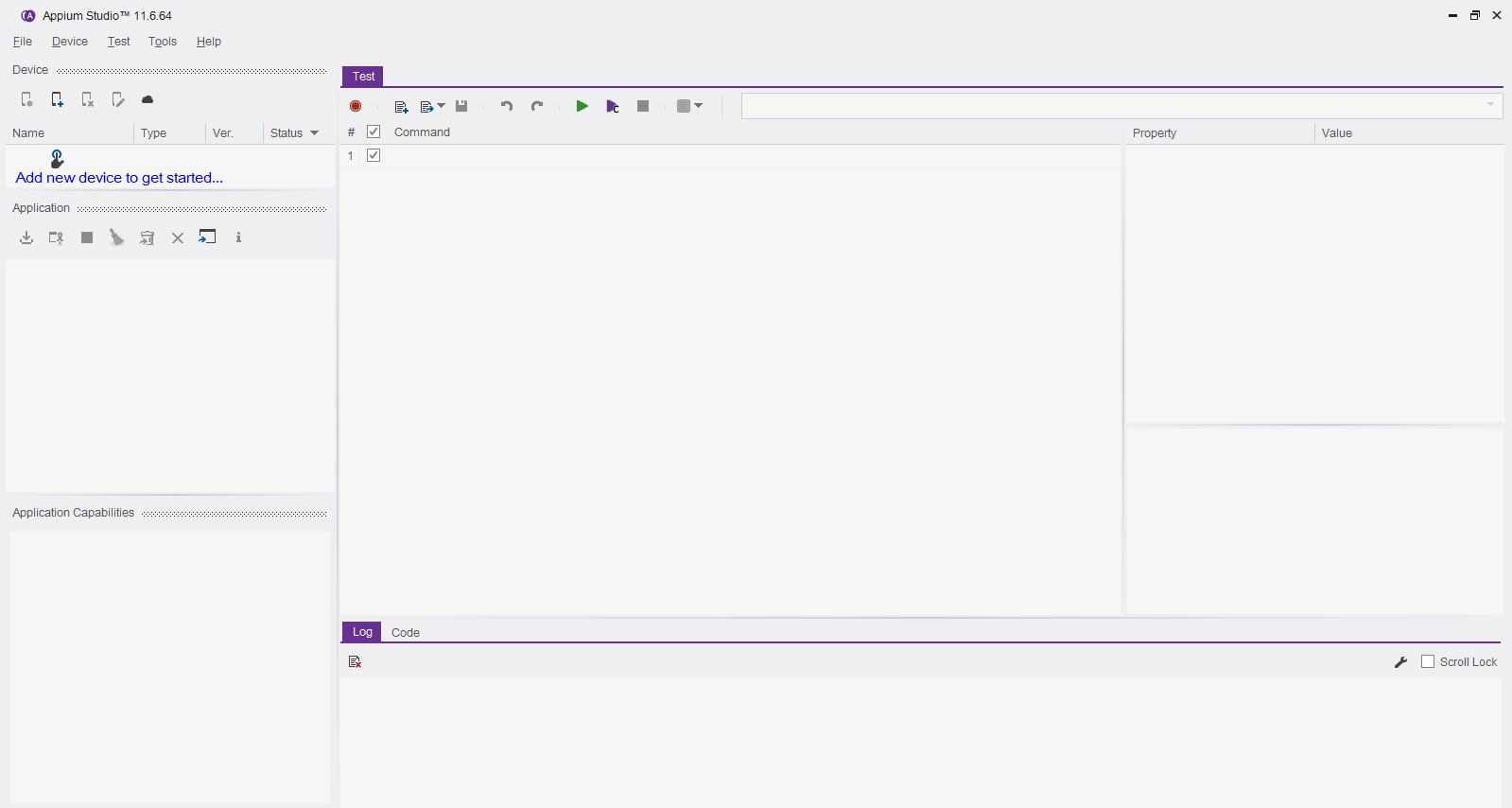
Se trata de dar valores a los criterios de comparación definidos en el apartado 3 sobre la implementación de cada uno de los prototipos.

## 6.1 Evaluación de los criterios en la implementación usando la tecnología A

Debe incluir al menos una tabla con la siguiente estructura.

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Interfaz de Usuario | Al abrir la aplicación se nos presenta una interfaz sencilla y funcional. Podemos añadir nuevos dispositivos, abrir proyectos, etc. La interfaz es clara. \* |
| Tiempo de aprendizaje | 3 horas (aprox) |
| Tiempo de configuración | 5 horas (aprox)\*\* |
| Sistema Operativos útiles | Se ha usado desde Windows para probar una apk de Android. |
| Horas empleadas en el desarrollo | 5 horas |
| Herramientas utilizadas en el desarrollo | Appium Studio, Android SDK, NodeJs, drivers para Xperia Z1\*\*\* |
| Costes de software | 0 |
| Requisitos en el ordenador | iOS: Mac OSX  XCode 4.5+ y con las herramientas de línea de comandos  Android: Mac OSX 10.7+ o Windows 7+ o Linux  Android SDK ≥ 16 (SDK < 16 en el modo del Selendroid) |

\*Pantalla de la interfaz de usuario:



\*\* Se han tenido diversos problemas con la instalación de los drivers por lo que su configuración ha durado más de lo esperado.

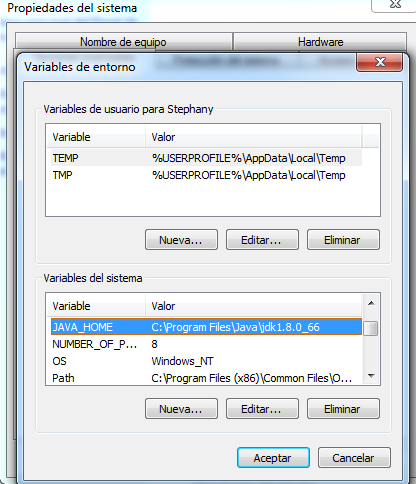
\*\*\*Al instalar Appium y querer hacer una prueba sobre Android debemos instalar varios complementos, como el Java Development Kit (JDK), Android SDK/ADB y los paquetes deseados, NodeJs antes de ponernos a trabajar. Estas instalaciones pueden llevar su tiempo, ya que son bastantes. Por ello, la habilidad a la hora de configurarlo es elevada, ya que además se requiere la preparación de los elementos como se muestra en las próximas capturas de pantalla.

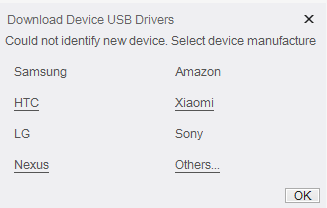
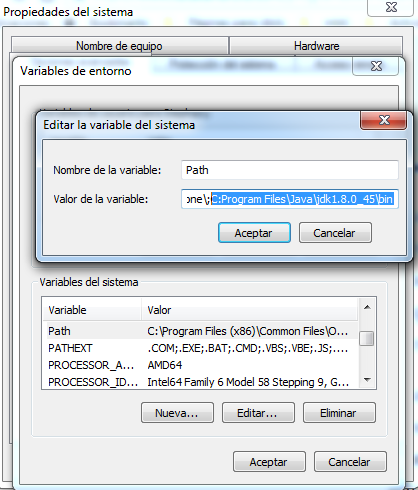
Además, para poder usar el software se nos pide que instalemos unos drivers, es decir, necesitamos tener un dispositivo o un emulador para poder probar el código. Sin dispositivo no se pueden hacer las pruebas.

En cuanto a los requisitos de hardware:

- Requisitos mínimos:

* Procesador Intel®  i5 or i7
* 1 GB de espacio en el disco duro.
* 8 GB RAM
* Un Puerto USB 2.0 y un cable USB





## 6.2Evaluación de los criterios en la implementación usando la tecnología B

| **CRITERIO** | **EVALUACIÓN** |
| --- | --- |
| Interfaz de Usuario | Es muy completa ya que al estar Espresso integrado a Android Studio, las opciones son infinitas. Puedes diseñar la aplicación directamente, y ejecutar las pruebas sin cambiar de aplicación. |
| Tiempo de aprendizaje | 10 horas (aprox) |
| Tiempo de configuración | 2 horas (aprox)\*\* |
| Sistema Operativos útiles | Se ha usado desde Windows para probar una apk de Android. |
| Horas empleadas en el desarrollo | 20 horas |
| Herramientas utilizadas en el desarrollo | Android Studio, Java JDK's y Android SDK's |
| Costes de software | 0 |
| Requisitos en el ordenador | Muchos. Requiere bastante capacidad de procesamiento.\* |

Requisitos en el ordenador.

- Requisitos mínimos:

* SSD
* 8 GB RAM
* Core i5

- Requisitos recomendados:

* SSD
* 16 GB RAM
* Core i7

Apuntar que, aunque estos son los requisitos recomendados, hemos probado la herramienta en un portátil con los requisitos mínimos y no lo hemos conseguido hacer funcionar. Al cambiar a un ordenador más potente de sobremesa hemos conseguido hacerlo funcionar, pero con problemas.

# 7. Comparación de la implementación de las tecnologías

| **CRITERIOS** | **TECNOLOGÍA A** | **TECNOLOGÍA B** | **COMENTARIOS** |
| --- | --- | --- | --- |
| Interfaz de Usuario | Al abrir la aplicación se nos presenta una interfaz sencilla y funcional. Podemos añadir nuevos dispositivos, abrir proyectos, etc. La interfaz es clara. \* | Es muy completa ya que al estar Espresso integrado a Android Studio, las opciones son infinitas. Puedes diseñar la aplicación directamente, y ejecutar las pruebas sin cambiar de aplicación. | Hablando de la interfaz de usuario, la interfaz de Appium está muy cuidada con pocas opciones y claras mientras que Espresso tiene más opciones con una interfaz algo más tosca. |
| Tiempo de aprendizaje | 8 horas (aprox) | 10 horas (aprox) | Estas horas son aproximadamente las necesarias para poder comenzar a utilizar estas aplicaciones. Si quisiéramos utilizarlas con un mayor potencial necesitaríamos muchas más horas.  También destacar que Appium solo se dedica al testing de aplicaciones mientras que Espresso hace un desarrollo al detalle de las aplicaciones por lo que es más complicado. También nos da más facilidades según vamos aprendiendo a hacer los test. |
| Tiempo de configuración | 5 horas (aprox)\*\* | 2 horas (aprox)\*\* | Espresso es una aplicación integrada a Android Studio, y las herramientas SDK se instalan junto con Android Studio, mientras que en Appium, se necesitan muchas más herramientas y el proceso no es automatizado. |
| Sistema Operativos útiles | Se ha usado desde Windows para probar una apk de Android.  Soporta: Mac OS a partir de X 10.7, Windows 7/8/10 32 o 64 y Linux. | Se ha usado desde Windows para probar una apk de Android.  Soporta: Mac OS a partir de X 10.10 hasta 10.13, Windows 7/8/10 32 o 64 y Linux. | Las dos soportan los tres sistemas operativos más importantes por lo que no hay mucho que comentar en este apartado. |
| Horas empleadas en el desarrollo | 5 horas | 20 horas | Son las horas transcurridas durante varios días de visualizaciones de tutoriales por internet y manuales. Creíamos que era más fácil poner a funcionar estas tecnologías, pero sin un conocimiento anterior son tecnologías que para alguien que no sepa nada antes se pueden hacer bastante complicadas.  Sobre todo, hablamos de la tecnología Espresso la cual nos llevó 3-4 días saber cómo utilizarla. |
| Herramientas utilizadas en el desarrollo | Appium Studio, Android SDK, NodeJs, drivers para Xperia Z1\*\* | Android Studio, Java JDK's y Android SDK's | Son las herramientas necesarias para poder hacer funcionar estas tecnologías. |
| Costes de software | 0 | 0 | Todo el software utilizado es gratis. |
| Requisitos en el ordenador | Se requieren diversos software complementarios y drivers, además de un dispositivo o emulador con el sistema operativo deseado. \*\* | Muchos. Requiere bastante capacidad de procesamiento.\* | Hay una notable diferencia en cuanto a recursos hardware se requieren ya que Espresso pide unos requisitos bastante elevados y aun así no funciona correctamente. Con Appium ningún problema.  Esta diferencia se ve reflejada en la potencia de cada tecnología, siendo Espresso mucha más rápida realizando pruebas. |

# 8. Conclusiones

Por último, terminamos con la conclusión habiendo probado las dos tecnologías.

Es difícil decidirse por una tecnología sin saber las necesidades de cada uno ya que son herramientas que en principio realizan la misma función, pero difieren mucho.

Si necesitamos una herramienta potente, con muchas posibilidades tanto de testeo como de desarrollo utilizaríamos Espresso ya que nos da estas opciones, pero ya sabemos que la configuración de esta herramienta y su puesta en marcha es costosa. A la larga nos va a ser muy rentable esta herramienta, pero necesitamos ordenadores potentes para poder hacerla funcionar, lo que nos permite realizar test a una increíble velocidad. Todo esto sabiendo que solo podemos utilizar esta tecnología con Android.

En cuanto a Appium, la escogeríamos si necesitáramos solo realizar pruebas y más sencillas, pero a la vez más versátiles por su capacidad de operar en distintos SO móviles y en muchos dispositivos a la vez. Además, es más fácil de utilizar en un primer momento.

Por esto es que no podemos elegir una de las dos tecnologías sin saber para que las queremos con más detalle.