**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**INGENIERÍA DEL SOFTWARE**

**Generador de pruebas de interfaz de usuario reparables para Android**

**Realizado por**

**Luis Pardo López**

**Dirigido por**

**José Antonio Parejo Maestre**

**Departamento**

**Lenguajes y sistemas informáticos**

**Sevilla, FECHA AQUÍ**

A mis padres, por toda una vida de cariño y apoyo, y a Félix, por ayudarme con tanto a lo largo de la carrera.

ÍNDICES AQUÍ

# Introducción

En la actualidad, existen más de 6.5 billones de smartphones en el mundo. Esto significa que aproximadamente el 84% de personas tiene un teléfono móvil inteligente [1]. El hecho de que haya tantas personas distintas con tantos dispositivos distintos da lugar, naturalmente, a la existencia de muchas aplicaciones.

Gráfico, Gráfico de barras

Descripción generada automáticamente

Figura 1 - Gráfica representando el número de usuarios de smartphones a lo largo de los últimos 5 años

Estas aplicaciones, antes de ser lanzadas al público, pasan una serie de pruebas y tests que comprueban que su funcionamiento sea correcto. De todas formas, al haber tantos dispositivos inteligentes, también hay una gran diversidad de modelos diferentes. Estos modelos tienen distintos diseños tanto internos como externos, desde sistemas operativos distintos hasta una gran variedad de tamaños y resoluciones de pantalla.

Esta cantidad tan ingente de modelos distintos implica una gran dificultad a la hora de testear aplicaciones móviles. El funcionamiento interno de la aplicación puede ser el mismo para varios modelos distintos, pero las pruebas de interfaz de usuario varían mucho de un modelo a otro, puesto que simplemente el tamaño de la pantalla del dispositivo puede influir en el resultado de la ejecución de la prueba. Además, el simple hecho de apaisar la pantalla ya cambiaría totalmente la disposición de los elementos de la aplicación, en caso de que esta esté diseñada para adaptarse a los modos vertical y horizontal.

Para este proyecto, nos centraremos específicamente en dispositivos Android, ya que la amplia gama de modelos que funcionan con sistemas operativos Android es más que suficiente para abordar el problema de la ejecución de pruebas de interfaz de usuario. La solución más evidente al problema indicado anteriormente es automatizar estas pruebas, diseñando un sistema que sirva para ejecutar pruebas en cualquier dispositivo, a pesar de las especificaciones de este. Otros compañeros y alumnos de este grado han estado trabajando en este sistema (creado inicialmente por Adrián Cantón) a lo largo de estos últimos años, y yo trabajaré sobre él con el fin de mejorar su eficacia.

## Objetivos del proyecto

Este proyecto consta de tres objetivos. El primero de ellos es mejorar el sistema de generación de valores aleatorios para las distintas acciones que se pueden realizar en una aplicación. El sistema de generación, aunque es aleatorio, también es limitado, puesto que en caso de que tenga que generar cadenas de texto para introducir en campos de texto, estas cadenas son únicamente palabras aleatorias extraídas de un diccionario. Se mejorará este sistema añadiendo más tipos de generadores, como uno que devuelva números entre un mínimo y un máximo predefinidos, uno que devuelva cadenas basándose en expresiones regulares, u otro que genere objetos de un tipo determinado que tengan probabilidades de ser escogidos asignadas aleatoriamente.

El segundo objetivo es el de mejorar el sistema de aserciones. Una aserción es una comprobación que se hace al final de la ejecución de una prueba. Actualmente, las aserciones son obligatorias. La mejora consistirá en hacer que se comprueben sólo en caso de que existan, y en mejorar el mismo sistema de aserciones también, dado que de momento sólo son comprobaciones visuales en pantalla al final de la ejecución de la prueba. Durante el desarrollo del proyecto se hará posible el uso de aserciones personalizadas que el usuario podrá escribir. Además, se dará apoyo al uso de estados intermedios de las pruebas para realizar dichas aserciones personalizadas.

El tercer objetivo es el de mejorar el sistema de parseo de las pruebas generadas por Espresso Test Recorder [2], un sistema de grabación y generación de pruebas automáticas de interfaz de usuario. El sistema convierte las pruebas de Espresso en ficheros con comandos que resultan más naturales de leer, de cara a poder recorrerlas con lenguaje java para ejecutarlas, así posibilitando su automatización. Actualmente, el sistema es incapaz de traducir algunas de las acciones de algunas aplicaciones a ese lenguaje. En concreto, es incapaz de traducir la acción de pulsar el botón de retroceder en el dispositivo móvil, por lo cual la mejora consiste en posibilitar el parseo de esa acción. También se añadirá la opción de hacer capturas de pantalla y guardarlas en el dispositivo móvil simulado del entorno.

## Estructura de la memoria

La estructura del contenido del presente documento es la siguiente:

● Capítulo 1: Introducción

En el primer capítulo se introducirá en el contexto del problema, analizando las herramientas existentes en la actualidad y detallando qué ofrecen y qué no.

● Capítulo 2: Análisis tecnológico

En este capítulo nos centraremos en explicar en detalle los diferentes lenguajes, frameworks y herramientas que se han usado durante el proyecto.

● Capítulo 3: Metodología y planificación

En esta parte hablaremos sobre la metodología de desarrollo usada y cómo se ha llevado a cabo la planificación de las diferentes tareas, así como los costes del proyecto.

● Capítulo 4: Análisis

En este capítulo nos centraremos en detallar los requisitos de la herramienta que se plantea en el proyecto.

● Capítulo 5: Diseño

En este capítulo se presentará la estructura del proyecto mediante diagramas.

● Capítulo 6: Implementación

En este capítulo se detalla todo el trabajo realizado durante el proyecto.

● Capítulo 7: Manual de usuario

En esta sección, se explicará en detalle cómo usar cada una de las funcionalidades de la aplicación, haciendo uso de capturas de pantallas para facilitar su comprensión.

● Capítulo 8: Pruebas

En este capítulo se mostrarán las pruebas realizadas para la validación del progreso.

● Capítulo 9: Conclusiones

En el capítulo final se compartirán las conclusiones obtenidas durante el desarrollo del proyecto.

# Análisis tecnológico

Para el desarrollo del proyecto hemos utilizado Java como lenguaje principal de programación, para mejorar las funcionalidades del proyecto.

## Lenguajes y plataformas

Se ha usado Java como lenguaje de programación debido a que la lógica de las aplicaciones Android nativas usa dicho lenguaje. Además, la librería UIAutomator está diseñada para este lenguaje, y las pruebas la requieren.

### Java

Java es un lenguaje de programación de propósito general orientado a objetos desarrollado por originalmente por Sun Microsystems y mantenido en la actualidad por la comunidad de software libre y Oracle. También se puede decir que Java es una tecnología que no sólo se reduce al lenguaje, sino que además provee de una máquina virtual Java que permite ejecutar código compilado Java, sea cual sea la plataforma que exista por debajo. Esta plataforma puede ser tanto hardware como software.

Las características de este lenguaje según Sun son:

● Simple.

● Orientado a Objetos.

● Tipado estáticamente.

● Distribuido.

● Interpretado.

● Robusto.

● Seguro.

● de Arquitectura Neutral.

● Multihilo.

● Con Recolector de basura (Garbage Collector).

● Portátil.

● De Alto Rendimiento: sobre todo con la aparición de hardware especializado y mejor software.

● Dinámico.

Algunas de las características específicas que nos han llevado a decidir utilizar dicho lenguaje son:

● Existencia de múltiples herramientas de desarrollo y librerías creadas por terceros disponibles para usar.

● Extensa documentación. Java tiene una excelente API.

● Las aplicaciones Android nativas utilizan este lenguaje para el desarrollo de su lógica.

## Librerías y frameworks

Durante la elaboración del proyecto se han usado los siguientes frameworks y librerías: UIAutomator y Junit para crear pruebas sobre la interfaz de usuario, EXTJwnl como generador de texto y Spring para evaluar los casos de pruebas durante la reparación de los tests.

## UIAutomator

UIAutomator es una librería Java que posee APIs para realizar tests de interfaz de usuario funcionales interaccionando con las aplicaciones tanto las instaladas como las del sistema. Es adecuado para desarrollar pruebas de caja negra automatizadas, donde el código de prueba no se basa en los detalles de implementación interna de la aplicación a probar. UIAutomator incluye las siguientes herramientas desarrolladas en Java:

● UIAutomator Viewer: Visor que captura la pantalla del dispositivo en un determinado momento y analiza los componentes que se muestran. Se puede inspeccionar la jerarquía de diseño y ver las propiedades de dichos componentes.

● Una clase UIDevice para acceder al dispositivo y realizar operaciones en él.

● Unas APIs para capturar y manipular los componentes de la interfaz de usuario de las aplicaciones del dispositivo.

Estas últimas APIs utilizadas para generar las pruebas para su posterior análisis presentan los siguientes objetos, los cuales se han necesitado para cada escenario:

● UiObject: representa un elemento de interfaz de usuario que es visible en el dispositivo.

● UiCollection: representa una colección de elementos de la interfaz de usuario (colección de UiObject).

● UiScrollable: proporciona compatibilidad con la búsqueda de elementos en un contenedor de interfaz de usuario desplazable.

● UiSelector: representa una consulta para encontrar uno o más elementos de interfaz de usuario en un dispositivo.

## JUnit

JUnit se trata de un Framework Open Source para la automatización de las pruebas (tanto unitarias, como de integración) en los proyectos Software. El framework provee al usuario de herramientas, clases y métodos que le facilitan la tarea de realizar pruebas en su sistema y así asegurar su consistencia y funcionalidad. En este proyecto, JUnit se utiliza para evaluar las distintas acciones que se ejecutan en la interfaz de usuario, para saber si se están ejecutando de la manera esperada.

## Espresso

Espresso es un framework creado por Google para Android que nos permite escribir tests sobre la interfaz de usuario. Dispone de una API fácil de adaptar a nuestras necesidades y elimina la complejidad de la gestión de distintos hilos de ejecución. Espresso UI Test funciona en tres bloques:

• ViewMatchers: permite encontrar un elemento en la vista.

• ViewActions: permite ejecutar acciones sobre los elementos.

• ViewAssertions: valida un estado de la vista.

Además, se ha utilizado la herramienta Espresso Test Recorder, que se encuentra integrada en Android Studio, para grabar automáticamente las acciones de interfaz de usuario mediante la navegación en propio dispositivo y guardarlas con la sintaxis de Espresso.

Captura de pantalla de un celular

Descripción generada automáticamente

Figura - Grabadora de tests Espresso Test Recorder

## JavaParser

JavaParser es una librería del lenguaje Java que permite parsear, analizar y transformar estéticamente el código fuente de un programa Java, proporcionando un árbol de sintaxis abstracto del mismo. De esta forma, se pueden identificar fácilmente los métodos de una clase y sus expresiones dentro de estos, así como las variables declaradas.

## Herramientas

En esta sección se describirán las herramientas usadas para la realización del proyecto.

## Android Studio

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android tras ser anunciado en 2013. La primera versión estable fue publicada en diciembre de 2014.

Está basado en el software intelliJ IDEA de JetBrains y fue publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas GNU/Linux, macOS, Microsoft Windows y Chrome OS. Está diseñado específicamente para el desarrollo de Android. Las funciones que ofrece son las siguientes:

• Soporte para construcción basada en Gradle.

• Refactorización específica de Android y arreglos rápidos.

• Herramientas Lint para detectar problemas de rendimiento, usabilidad, compatibilidad de versiones y otros problemas.

• Integración de ProGuard y funciones de firma de aplicaciones.

• Plantillas para crear diseños comunes de Android y otros componentes.

• Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.

• Soporte para programar aplicaciones para Android Wear.

• Soporte integrado para Google Cloud Platform, que permite la integración con Firebase Cloud Messaging y Google App Engine.

• Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.

• Renderizado en tiempo real.

• Consola de desarrollador: consejos de optimización, ayuda para la traducción, estadísticas de uso.

## Android SDK

El SDK (kit de desarrollo software) de Android nos permite principalmente crear aplicaciones para Android. Desde las más sencillas hasta cualquiera que nuestros conocimientos de programación nos permitan. Las APIs de Android facilitan la tarea y la propia herramienta de Android Studio nos deja descargar decenas apps de código abierto para que veamos cómo funciona todo.

Existen también multitud de plantillas de código, enlaces a Github, tutoriales y frameworks de prueba. Una cantidad enorme de herramientas para adentrarse en el mundo de la programación Android.

## Gradle

Gradle es una herramienta de automatización de la construcción de nuestro código que bebe de las aportaciones que han realizado herramientas como Apache Ant y Apache Maven, pero intenta llevarlo todo un paso más allá.

Se apoya en Groovy y en un DSL (Domain Specific Language) para trabajar con un lenguaje sencillo y claro a la hora de construir el build comparado con Maven. Por otro lado, dispone de una gran flexibilidad que permite trabajar con ella utilizando otros lenguajes y no solo Java. También presenta un sistema de gestión de dependencias sólido.

## Github

Github es una herramienta que nos permite tener nuestros repositorios de Git en la nube. Esto nos permite centralizar el contenido del repositorio para poder colaborar con otros miembros de nuestra organización.

También permite la integración con otras herramientas de testeo de software y la incorporación de extensiones que aportan aún más funcionalidades a la herramienta.

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

Ilustración 1 - Interfaz de Github

## Clockify

Clockify es una página web que permite crear proyectos con el fin de medir cuánto tiempo se dedica a cada una de las tareas individuales de cada uno. Con esta herramienta se puede ver el total de horas trabajadas en el proyecto, en cada tarea, diariamente e incluso semanalmente.

Esto nos permite no solamente seguir el progreso temporal del proyecto, sino también da mucha información sobre lo que se ha tardado más y menos en hacer. Además, permite la sincronización con GitHub para poner el nombre automáticamente a las tareas basándose en el nombre de las issues creadas en el proyecto de GitHub.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ilustración 2 - Interfaz de Clockify

## Google Drive

Google Drive es un servicio de almacenamiento de datos que son guardados en la nube (plataforma en línea a la que se accede desde cualquier dispositivo con conexión a Internet). El servicio de almacenamiento de Google Drive ofrece 15 GB de capacidad sin costo para cada usuario de Gmail. Además, ofrece planes de pago que disponen de mayor espacio en la nube, entre otros beneficios.

Google Drive permite copiar archivos desde el ordenador para que sean guardados en la nube. En caso de que el usuario quiera editar esos documentos, debe descargarlos en su ordenador. También, permite crear documentos y carpetas directamente desde la plataforma a través de los documentos de Google Docs que presentan funciones muy similares a los programas del paquete Microsoft Office.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteEsta herramienta se ha utilizado para almacenar toda la información que no tenga que ver con el código fuente de la aplicación. Esto es, la memoria, las actas de reuniones, la planificación de horas y el backlog de tareas, y los enlaces de interés.

Ilustración 3 - Interfaz de Google Drive

# Metodología y planificación

En esta sección se explicará la metodología utilizada para el desarrollo del proyecto, su justificación y la planificación que se ha seguido para realizar el trabajo.

## Metodología usada

En este proyecto se ha optado por utilizar la metodología Scrum, ya que al intervenir solo dos actores (profesor y alumno), se puede adaptar de tal forma que cada actor tenga más de un rol. Además, para el seguimiento del proyecto es más adecuado que se lleven a cabo incrementos cortos con requisitos cambiantes, ya que, al ser Scrum una metodología iterativa, en cada Sprint se visitan todas las fases de desarrollo.

Comenzaremos definiendo los roles implicados en el desarrollo del proyecto usando dicha metodología:

● Product Owner: es el responsable del Product Backlog y de optimizar el trabajo del equipo.

● Scrum Master: vela por el correcto cumplimiento de la metodología Scrum y ayuda al equipo a crear productos de valor.

● Equipo de desarrollo.

En este proyecto se ha establecido que el alumno lleve el rol del único miembro del equipo de desarrollo y el profesor que dirige el proyecto sea Product Owner y Scrum Master, además de llevar a cabo las funciones de cliente. Los artefactos que establece Scrum para el manejo de tareas es el siguiente:

● Product Backlog: es un tablero donde se describen las tareas a realizar durante todo el proyecto. Estas tareas están priorizadas de tal forma que se realizan antes las que se consideren de mayor necesidad.

● Sprint Backlog: es un tablero donde se describen las tareas a realizar durante el Sprint actual. Todas las tareas están igualmente priorizadas.

Scrum propone varios tipos de reuniones para el seguimiento del proyecto. Estas presentan una periodicidad, debido a la existencia de varios Sprints:

● Sprint Planning Meeting: reunión al principio de cada Sprint donde se planifica qué tareas se van a realizar en este y sus estimaciones, obteniéndose así el Sprint Backlog.

● Daily Scrum: reunión breve diaria donde cada miembro del equipo ofrece una actualización sobre el estado de sus tareas y los problemas encontrados si hubiese.

● Sprint Review: reunión al final de cada Sprint donde se lleva un recorrido por el número de tareas del Sprint Backlog completadas y el estado actual del proyecto.

● Sprint Retrospective: reunión al final de cada Sprint donde se lleva a cabo un análisis cualitativo del proyecto. Se discute sobre los aspectos bien llevados y los aspectos que necesitan cambios, con el fin de mejorar en los siguientes Sprints.

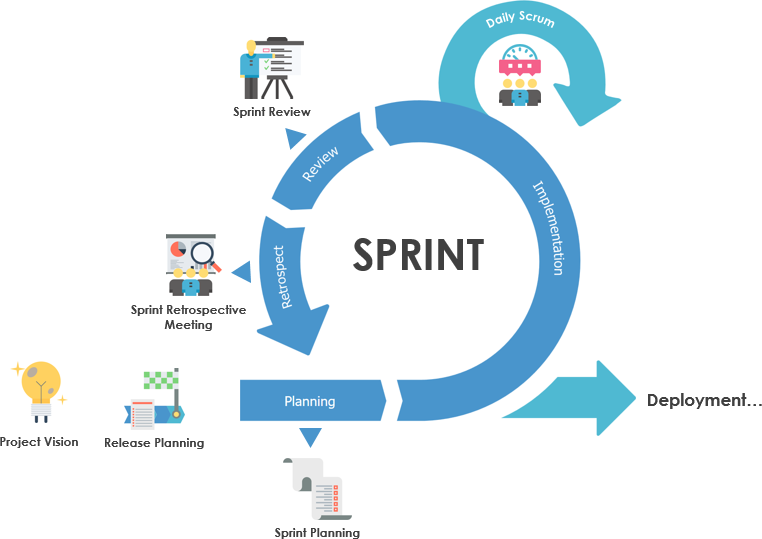


Ilustración 4 - Ciclo de vida de un Sprint de la metodología Scrum

## Modificaciones para este proyecto

Dada la naturaleza de este proyecto, como sólo hay dos participantes, hemos se ha decidido eliminar las reuniones diarias y prescindir de la plantilla de retrospectiva. Además, se han decidido realizar de forma secuencial las reuniones de revisión, retrospectiva y planificación del siguiente sprint en una misma reunión, la cual ha sido invocada en todos los casos por el alumno.

Las tareas han ido siendo asignadas en las mismas reuniones tras haber acabado las anteriores, puesto que por razones de tiempo era posible que algunas de las mejoras a aplicar en el proyecto no diesen tiempo de ser desarrolladas.

## Planificación

# Análisis

Para llevar a cabo el desarrollo del proyecto se han establecido unos requisitos a cumplir. A continuación, se nombran dichos requisitos en una tabla y en las siguientes secciones se detalla cada uno de ellos.

|  |  |
| --- | --- |
| ID | Requisito |
| R-01 | Establecer jerarquía de generadores variados para crear entradas de texto |
| R-02 | Implementar generadores en el proyecto para que funcionen |
| R-03 | Implementar el uso de un archivo de configuración para los generadores |
| R-04 | Implementar la acción de pulsar el botón de retroceder en tests parseados |
| R-05 | Implementar sistema de aserciones personalizadas |
| R-06 | Añadir opción de tomar capturas de pantalla |

Tabla 1 - Requisitos del proyecto

## Establecer jerarquía de generadores variados para crear entradas de texto

El primer requisito consiste en crear una serie de generadores que, al ser invocados, devuelvan una serie de valores generados aleatoriamente. Esto se hace con el fin de poder elegir en un futuro qué tipo de valor ha de generarse para los distintos campos de texto que se pueden encontrar en las aplicaciones que se están probando. Los distintos generadores son los siguientes:

• RandomIntegerGenerator: Generador de números enteros aleatorios entre un mínimo y un máximo.

• IntegerListGenerator: Generador de números enteros elegidos aleatoriamente de una lista determinada.

• ProbabilityGenerator: Generador de números enteros elegidos aleatoriamente, con probabilidades asignadas a cada valor, de una lista determinada.

• GivenValueGenerator: Generador de cadenas de texto que devuelve el valor proporcionado sin modificarlo.

• IncrementDoubleGenerator: Generador de números decimales que devuelve un incremento desde 0 a 1 sobre un número entero dado.

• RandomRegexGenerator: Generador de cadenas de texto aleatorias basadas en una expresión regular.

• ReflectionGenerator: Generador de objetos con atributos generados aleatoriamente usando los demás generadores.

## Implementar generadores en el proyecto para que funcionen

El segundo requisito consiste en unir la jerarquía de generadores al resto del proyecto de modo que funcionen correctamente y se puedan invocar para usarlos en los tests. Los pasos para ello son los siguientes:

• Crear AbstractGenerator, un generador padre que contiene un método llamado “generate”, que usan todos los demás generadores, los cuales extienden a esta clase abstracta.

• Probar a invocar los generadores creados en lugar del que ya había en el proyecto, DictionaryBasedValueGenerator, y comprobar que su funcionamiento es correcto

## Implementar el uso de un archivo de configuración para los generadores

El tercer requisito consiste en crear un archivo de configuración que permita indicar al sistema qué generador utilizar para cada campo de texto a rellenar. Los pasos son los siguientes:

• Diseñar un formato de archivo de configuración.

• Implementar un sistema de lectura del archivo de configuración.

• Preparar el sistema para que, por cada campo de texto en un test, use una línea del archivo de configuración.

## Implementar la acción de pulsar el botón de retroceder en tests parseados

El cuarto requisito consiste en añadir la opción de pulsar el botón de retroceder del dispositivo móvil simulado a los tests parseados. Esta acción funciona al grabar tests de Espresso pero no al parsearlos. Los pasos son los siguientes:

• Crear una nueva acción que consiste en retroceder pulsando el botón de atrás en el dispositivo.

• Añadir el proceso de reconocimiento de la acción de retroceder en la clase que parsea los tests.

• Añadir semántica para la acción nueva en la clase que parsea los tests.

## Implementar sistema de aserciones personalizadas

El quinto requisito consiste en modificar el proyecto de modo que permita al usuario añadir aserciones personalizadas al final de los tests generados por Espresso. Los pasos son los siguientes:

• Crear clase abstracta llamada “AssertionChecker” que contenga el método abstracto “checkAssertion” y los métodos “setInitialLabels” y “setFinalLabels”. Además contendrá las variables “initialLabels” y “finalLabels”. Esto permitirá guardar el estado inicial y el estado final del test, para poder usarlos en las aserciones personalizadas.

• Añadir método llamado “assertionCheck” manualmente en la clase test generada por Espresso, donde se comprueben condiciones relevantes al test, y la invocación del mismo dentro del método principal de la clase test.

• Añadir código de parseo que permita al sistema reconocer clases test con aserciones personalizadas y añadir tales aserciones al fichero generado en lenguaje natural.

• Añadir método de evaluación de aserciones personalizadas.

## Añadir opción de tomar capturas de pantalla

El sexto y último requisito consiste en añadir la funcionalidad de generar capturas de pantalla en el dispositivo móvil simulado a cualquier altura de la ejecución del test. La captura de pantalla, además de generar la imagen que muestra el contenido de la pantalla en el momento de tomarla, también generará una lista de etiquetas de tipo String correspondiente a las etiquetas mostradas en pantalla en ese instante. Los pasos para ello son los siguientes:

•Añadir de forma manual la orden de tomar una captura de pantalla al test generado en lenguaje natural.

•Añadir una acción llamada “ScreenshotAction”, la cual genere una captura de pantalla y una lista de etiquetas llamada “screenshotLabels” en la clase “AssertionChecker”, la cual fue creada para el requisito anterior. Esto permitirá guardar el estado de la prueba en el momento de la generación de la imagen.

•Añadir código a la clase que recorre los tests en lenguaje natural para reconocer la orden de captura de pantalla.

# Diseño

En este capítulo, por un lado, se adjuntan diagramas correspondientes a detalles técnicos del desarrollo del proyecto. Por otro lado, se detallan los patrones de diseño utilizados.

## Diagramas

El primero de los diagramas es un UML que muestra la jerarquía de generadores.

Diagrama

Descripción generada automáticamente

Ilustración 5 - Diagrama UML de la jerarquía de generadores

## Patrones de diseño

A lo largo del desarrollo del proyecto se ha hecho uso de un patrón de diseño, que proporciona mecanismos de creación de objetos con el fin de incrementar la flexibilidad y la reutilización del código.

## Patrón Factory Method

Este patrón sirve para separar la clase que crea los objetos de la jerarquía de objetos a instanciar, es decir, proporciona una interfaz para crear objetos en una superclase, mientras que permite a las subclases alterar el tipo de objetos que se crearán. El uso de este patrón aporta centralización en la creación de objetos y escalabilidad en el sistema.

Gracias a este patrón han sido creados los generadores de valores aleatorios.

# Implementación

En esta sección se detallan las mejoras aplicadas al proyecto. Se explicará el proceso de definición de una jerarquía de generadores que devuelven distintos tipos de valores según lo que se indique en un fichero de texto de configuración, y también el proceso en el cual se han añadido más acciones al parseo de los tests para permitir tests más complejos. Toda la implementación se encuentra en el siguiente repositorio de GitHub: <https://github.com/luiparlop1/espresso-parsing>. El proyecto está funcional y listo para iniciar en el momento en el que se clona.

## Jerarquía de generadores

Durante el proceso de definición de la jerarquía de generadores, se han creado varios distintos para devolver valores variados a la hora de rellenar campos de texto. El resto del sistema se ha dejado intacto, por lo cual funciona exactamente como funcionaba en el proyecto anterior, de Iván. Solo se ha hecho una pequeña mejora que permite grabar tests de Espresso sin aserciones y luego parsearlos sin que el sistema devuelva error. Esto se debe a que se considera que se pueden realizar tests sin necesidad de añadir una aserción en cada uno, ya sea por comodidad del programador o porque el test sea tan simple que ni siquiera la necesite. Esta mejora permite crear tests tanto con aserciones como sin ellas, sin que devuelva error al crearlos sin ellas.

Para implementar esta mejora, se ha accedido a la clase TestCase, donde se genera el test con la información de la clase test autogenerada por Espresso, y se ha añadido una condición en dos lugares clave. La condición permite saltarse la aserción final del test si no hay ninguna. A continuación, se muestran los cambios a los métodos “evaluate” y “setPredicate” de la clase TestCase.

|  |
| --- |
| public boolean evaluate(){  if(predicate!=null){  return predicate.evaluate(this);  }  return true; } |

Listado 1 - Método evaluate de la clase TestCase

|  |
| --- |
| public void setPredicate(String predicate){  if(predicate!=null)  this.predicate = new TestPredicate(predicate); } |

Listado 2 - Método setPredicate de la clase TestCase

Como se puede observar en el código, en el método “evaluate”, se ha especificado que devuelva “true” siempre, a no ser que la aserción (llamada “predicate”) exista. En el método “setPredicate” simplemente se ha añadido la condición de que, si la aserción existe, se cree. Esto se tuvo que hacer para que no devolviese un error.

A continuación, y antes de entrar en detalle con cada uno de los generadores, se explicarán los cambios hechos al proyecto para añadirlos al sistema.

En la clase TextInputGenerator, había un método llamado generateInput. Este método llamaba a otro llamado “generate” que guardaba el valor predeterminado que se le pasaba como parámetro de entrada. En caso de que la semilla (que también recibía como parámetro de entrada) fuese negativa, simplemente devolvía ese valor. En caso contrario, usaría esa semilla para generar una o más palabras que obtenía de un diccionario para devolverlas en un resultado de tipo String.

|  |
| --- |
| public String generateInput(UiObject object) throws UiObjectNotFoundException {  String value = getDefaultValue();   DictionaryBasedValueGenerator dictionary = new DictionaryBasedValueGenerator(1, getSeed());  try {  if(getSeed() > 0 || defaultValue == null)  value = dictionary.generate().toString();  Log.*d*("TFG", value);  object.setText(value);  } catch (JWNLException e) {  e.printStackTrace();  }    return value; } |

Listado 3 - Método generateInput original de la clase TextInputGenerator

En este método sólo se utiliza un generador, llamado DictionaryBasedValueGenerator. Para añadir la posibilidad de usar otros generadores, se ha cambiado enormemente el interior del método.

El nuevo método, de mismo nombre, hace lo mismo cuando la semilla es negativa. De todas formas, cuando es positiva, entra en un Switch que recorre distintos casos. Los casos dependen de una variable de tipo String llamada “type”, que recibe del objeto de tipo TextInputGenerator que se crea justo antes de invocar el método. Dependiendo del contenido de la variable “type”, se crea un generador distinto cada vez que el método es invocado.

|  |
| --- |
| public String generateInput(UiObject object) throws UiObjectNotFoundException{  String res = getDefaultValue();  List<Integer> integerList = new ArrayList<>();   if(type.equals("numberFromList") || type.equals("numberFromProbabilityList")){  String[] numbers = cond1.split(",");  for(String number: numbers){  integerList.add(Integer.*parseInt*(number));  }  }   try {  if (getSeed() > 0 || defaultValue == null) {  switch (type) {  case "numberFromList":  IntegerListGenerator integerRes = new IntegerListGenerator(integerList);  res = integerRes.generate().toString();  break;  case "numberFromProbabilityList":  ProbabilityGenerator integerProbabilityRes = new ProbabilityGenerator(integerList);  res = integerProbabilityRes.generate().toString();  break;  case "number":  int min=Integer.*parseInt*(cond1);  int max=Integer.*parseInt*(cond2);  RandomIntegerGenerator numberRes = new RandomIntegerGenerator(min, max);  res = numberRes.generate().toString();  break;  case "regex":  RandomRegexGenerator regexRes = new RandomRegexGenerator(cond1);  res = regexRes.generate();  break;  case "word":  int numberOfWords = Integer.*parseInt*(cond1);  DictionaryBasedValueGenerator dictionaryRes = new DictionaryBasedValueGenerator(numberOfWords, Math.*abs*(new Random().nextLong()));  res = dictionaryRes.generate();  break;  case "given":  String givenWord = cond1;  GivenValueGenerator givenRes = new GivenValueGenerator(givenWord);  res = givenRes.generate();  break;  case "increment":  int givenNumber = Integer.*parseInt*(cond1);  IncrementDoubleGenerator incrementRes = new IncrementDoubleGenerator(givenNumber);  res = incrementRes.generate().toString();  break;  case "reflection":  ReflectionGenerator reflectionRes = new ReflectionGenerator(cond1);  res = reflectionRes.generate();  break;  }  }  }catch (JWNLException e) {  e.printStackTrace();  }  object.setText(res);  return res; } |

Listado 4 - Método generateInput nuevo de la clase TextInputGenerator

Los generadores nuevos suman un total de siete. Serán listados a continuación.

## AbstractGenerator

AbstractGenerator es simplemente la clase abstracta que contiene el método llamado “generate”, y todos los demás generadores extienden a esta clase.

|  |
| --- |
| public abstract class AbstractGenerator<T> {   public abstract T generate() throws UiObjectNotFoundException, JWNLException, IllegalAccessException, NoSuchFieldException, ClassNotFoundException, InstantiationException;  } |

Listado - Clase AbstractGenerator

## RandomIntegerGenerator

RandomIntegerGenerator es un generador que en su método “generate” genera un número aleatorio entre un mínimo y un máximo. Estos dos parámetros los recibe como entrada cuando es invocado desde TextInputGenerator.

|  |
| --- |
| public Integer generate() throws JWNLException {  Integer randomValue = (int)(Math.*random*()\*(*max*-*min*+1)+*min*);  return randomValue; } |

Listado 6 - Método generate de la clase RandomIntegerGenerator

## IntegerListGenerator

IntegerListGenerator es un generador que, a diferencia de RandomIntegerGenerator, escoge un número entero aleatorio de una lista que recibe como parámetro de entrada.

|  |
| --- |
| public Integer generate() throws JWNLException{   Integer min = 0;  Integer max = valueList.size()-1;  Integer chosenIndex = (int)(Math.*random*()\*(max-min+1)+min);   Integer randomValue = valueList.get(chosenIndex);   return randomValue; } |

Listado 7 - Método generate de la clase IntegerListGenerator

## ProbabilityGenerator

ProbabilityGenerator está muy ligado a IntegerListGenerator. De hecho, lo extiende. Su método “generate” es considerablemente más complejo que el de IntegerListGenerator, de todas formas. Este método recibe la misma lista de números como entrada, pero en lugar de elegir uno aleatoriamente, le asigna a cada valor de la lista un valor numérico aleatorio, el cual usa luego para determinar cuántas probabilidades tiene el número de ser escogido. Esto añade aún más aleatoriedad a la selección.

|  |
| --- |
| public Integer generate() throws JWNLException {   List<Integer> probabilities = new ArrayList<>();  Integer index = 0;   for (Integer i: integerList){  Integer prob = (int)(Math.*random*()\*(100-1+1)+1);  probabilities.add(prob);  probabilityMap.put(i,probabilities.get(index));  index++;  }   Integer min = 0;  Integer max = 0;  Integer randomValue = 0;  for(int value: probabilityMap.values())  max = max+value;   Integer chosenNumber = (int)(Math.*random*()\*(max-min+1)+min);  Integer acc = 0;  for (Map.Entry<Integer, Integer> entry : probabilityMap.entrySet()) {   if(chosenNumber >= acc && chosenNumber < acc+entry.getValue()){  randomValue = entry.getKey();  }  acc = acc+entry.getValue();  }  return randomValue; } |

Listado 8 - Método generate de la clase ProbabilityGenerator

## IncrementDoubleGenerator

IncrementDoubleGenerator recibe un número entero como entrada y su método “generate” le aplica un incremento o decremento de entre 0 y 1, devolviendo un número de tipo Double.

|  |
| --- |
| public Double generate() throws JWNLException {   Double min = 0.0;  Double max = 1.0;  Double increment = ThreadLocalRandom.*current*().nextDouble(min, max);  Double randomValue = givenNumber + increment;  return randomValue; } |

Listado 9 - Método generate de la clase IncrementDoubleGenerator

## GivenValueGenerator

GivenValueGenerator simplemente devuelve el valor que recibe.

|  |
| --- |
| public String generate() throws UiObjectNotFoundException {  return defaultValue; } |

Listado 10 - Método generate de la clase GivenValueGenerator

## RandomRegexGenerator

RandomRegexGenerator devuelve un valor de tipo String que crea basándose en una expresión regular. Dependiendo del valor de la variable “type” que recibe como entrada, utiliza una expresión regular que genera un email o una que genera un número de teléfono.

|  |
| --- |
| public String generate() throws JWNLException, UiObjectNotFoundException {   String randomValue = null;   if(type.equals("email")) {  String regex = "[bcdfghjklmnpqrstvwxy][aeiou][bcdfghjklmnpqrstvwxy][aeiou][bcdfghjklmnpqrstvwxy][aeiou][bcdfghjklmnpqrstvwxy][aeiou][123456789][123456789](\\@)(gmail|hotmail|yahoo)(\\.)(com)";  Xeger generator = new Xeger(regex);  randomValue = generator.generate();  assert randomValue.matches(regex);  }else if(type.equals("phone")){  String regex = (95)[0123456789]()[0123456789][0123456789][0123456789]()[0123456789]  [0123456789][0123456789])";  Xeger generator = new Xeger(regex);  randomValue = generator.generate();  assert randomValue.matches(regex);  }  return randomValue; } |

Listado 11 - Método generate de la clase RandomRegexGenerator

## Archivo de configuración

El archivo de configuración es un fichero de texto plano que detalla qué generador utilizar y con qué valores hacerlo, cada vez que haya que aportar valores textuales para un campo de texto dentro de una aplicación en la que se estén haciendo pruebas.

A continuación, se mostrará un ejemplo de un archivo de configuración y se procederá a su posterior análisis, explicando sus componentes.

Texto

Descripción generada automáticamente con confianza media

La estructura del texto es la siguiente:

• Cada línea del fichero es una entrada distinta que indica qué generador usar y los valores de entrada para generar los resultados.

• Dentro de cada línea, la primera palabra indica qué generador utilizar. Hay 8 opciones: number, numberFromList, numberFromProbabilityList, word, regex, given, reflection e increment.

• Después de la primera palabra de cada línea, hay un guion. Esto sirve para separar el tipo de generador de los valores de entrada de dicho generador.

• Para separar los valores de entrada de cada generador, se hace uso de la barra “/”. En algunos casos, los generadores no requieren más de un valor, por lo cual el segundo se muestra vacío. Como máximo, son necesarios dos valores. En caso de que la opción sea numberFromList o numberFromProbabilityList, el primer valor es una lista de números separados por comas.

## Integración de los generadores y archivo de configuración

Para que el sistema funcione con los generadores nuevos y el archivo de configuración, ha habido que hacer varios cambios al sistema. Empezaremos explicando los cambios que se han hecho en la clase “ReadUtil” (dentro de la carpeta util).

En el método “generateTestCase”, se ha hecho lo siguiente:

|  |
| --- |
| Integer actionsSize = new Integer(lines[2]); String action = ""; String generatorType = ""; String cond1 = ""; String cond2 = ""; Integer textInputCounter = 0; ReadUtil ru = new ReadUtil("Download/config.txt"); String configFile = ru.readText(); String[] configLines = configFile.split("\n");  String predicate = null; //Inicializo predicate a null porque no siempre tiene por qué haberlo for(int i = 3; i<= actionsSize + 2; i++){  action = lines[i];  if(action.contains("TEXT")) {  String configLine = configLines[textInputCounter];  textInputCounter++;  String[] splitConfigLine = configLine.split("-");  generatorType = splitConfigLine[0];   String conditions = splitConfigLine[1];  String[] splitConditions = conditions.split("/");  cond1 = splitConditions[0];  if(!conditions.endsWith("/")) {  cond2 = splitConditions[1];  }  } |

Listado 12 – método generateTestCase modificado de la clase ReadUtil

Comenzando en la línea 91, se inicializa la variable “generatorType”. Luego se inicializan las dos condiciones “cond1” y “cond2”. Estas corresponden a los valores de entrada separados por “/” en el archivo de configuración. También se inicializa a cero una variable llamada “textInputCounter”, que será nuestro contador de acciones de tipo texto que hay en un test. Luego, se lee el archivo de configuración, llamado “config.txt”. Tras esto, en el bucle for que recorre todas las acciones, una condición if evalúa si la acción es de tipo texto. Cuando encuentra una, coge la línea correspondiente a la acción (esto se hace usando el valor de la variable “textInputCounter”, que se incrementa cada vez que se entra en el bucle) y la disecciona, sacando cada elemento de la línea.

En el caso de la segunda condición, un if la inicializa como el valor a la derecha de la barra “/” en caso de que haya algo en esa posición. Cabe indicar que la línea, haya o no condición, no debe acabar con un espacio o algo detrás de esta condición, pues el método incluirá esto en la condición y puede causar errores.

Este código añadido al método permite leer y usar el archivo de configuración. Bajo esa sección del código, se encuentra otra que también ha tenido que ser modificada.

|  |
| --- |
| if (seed < 0){  if(!action.startsWith("CUSTOM ASSERTION"))  testActions.add(generateActionFromString(action, seed, generatorType,cond1,cond2)); }else{  if(!action.startsWith("CUSTOM ASSERTION"))  testActions.add(generateActionFromString(action, seed, generatorType,cond1,cond2));  if(i == actionsSize+2){  break; } |

Listado 13 - Arreglo de un error en el que se pasaba una semilla errónea al método

Al crear las acciones de la lista ”testActions”, en el proyecto de Iván (último alumno que estuvo trabajando en el proyecto) se pasaba al método “generateActionFromString” un número aleatorio que podía ser positivo o negativo a modo de semilla. Esto causaba problemas a la hora de generar valores aleatorios porque, a veces, al ser la semilla negativa, el sistema no generaba valor alguno y devolvía el valor entrante, debido a la condición que le indica que las semillas negativas significan que no hay que generar nada aleatoriamente. Tras corregir eso, el sistema funciona correctamente de nuevo y se crea la acción. Para crearla, se le pasa la acción, la semilla, el tipo de generador a utilizar para esa acción y las condiciones necesarias. Una condición antes de generar las acciones comprueba que no sean del tipo “CUSTOM ASSERTION”, porque si lo son, no es necesario crear acciones nuevas. Esto forma parte de las aserciones personalizadas, y se entrará más en detalle al respecto de estas más adelante en el documento.

Por último, otro cambio que se tuvo que hacer con respecto al proyecto de Iván fue el de las siguientes líneas del método.

|  |
| --- |
| long numberOfLines = Arrays.*stream*(lines).count();actionsSize = new Integer(lines[2]);  if (numberOfLines != actionsSize+3) {  predicate = lines[actionsSize + 3]; |

Listado 14 - Modificación del código que previene errores cuando un test no contiene aserciones

Los comentarios incluidos en el código explican el cambio. Resumiendo, se ha modificado el método para que no falle, aunque no haya aserción en el test. Se ha considerado que los tests no tienen por qué tener aserciones, por lo cual no deberían fallar en caso de que no hubiera.

Además de los cambios en la clase “ReadUtil”, se han realizado otros en la clase “TextInputGenerator”, concretamente dentro del método “generateInput”.

Los cambios en esta clase permiten la invocación de los distintos tipos de generadores dependiendo de las entradas del archivo de configuración. Al inicio de la clase, se inicializan las variables “seed”, “defaultValue”, “type”, “cond1” y “cond2”. Al crear el objecto de tipo TextInputGenerator en la clase “ReadUtil”, se crean estos campos con los valores correspondientes. También se ha modificado el método “generateInput”. Con tal de evitar la repetición, puede verse el código de este método en el apartado [6.1. Jerarquía de generadores.](#_Jerarquía_de_generadores)

Al principio del método, se obtiene el valor por defecto de la entrada de texto. Esto se hace por si es necesario utilizarlo, cuando la semilla que se le pasa al método es negativa. Luego, se inicializa una lista de números enteros, que sólo se utilizará si la entrada de texto a generar es de tipo “numberFromList” o “numberFromProbabilityList”. En esos casos, esa lista de números se rellenará con los valores que se le pasen en el archivo “config.txt”.

Tras esto, si la semilla es mayor a cero o defaultValue es nulo, se pasa a recorrer los distintos casos. Dependiendo del valor de la variable “type”, se crean objetos distintos correspondientes a los distintos tipos de generador a utilizar, y se utilizan las condiciones correspondientes a cada tipo de generador para condicionar la salida generada. Finalmente, se devuelve el resultado.

## Acción de retroceder durante un test

Para posibilitar la acción de pulsar el botón de “atrás” durante la ejecución de un test, ha habido que hacer un cambio en la clase de parseo.

|  |
| --- |
| if(mc.toString().equals("pressBack()")){  objectTypes.add("GO\_BACK");  selectors.add("backButton");  texts.add("Go back"); } |

Listado 15 - Código añadido a la clase TextualUITestGenerator para dar apoyo a la acción de retroceder

Ahora, si la línea del test que se está analizando es “pressBack()”, lo cual indicaría que es la acción de pulsar el botón de retroceder, añade una acción de tipo “GO\_BACK”, la cual estaba ya presente en el proyecto pero no en uso. Esto añade una línea al test parseado como la siguiente:

|  |
| --- |
| GO\_BACK, UiSelector[backButton], Go back |

Listado 16 - Acción de retroceder representada en un test parseado

Los valores textuales de “backButton” y “Go back” tienen una función más visual para el usuario que funcional para el sistema. Una vez se analice esta línea del test parseado, el sistema pulsará el botón de retroceder del dispositivo móvil.

## Aserciones Personalizadas

Para capturar y almacenar los estados inicial y final del test de manera independiente al sistema de aserciones ya existente, se hace uso de la clase abstracta “AssertionChecker”.

|  |
| --- |
| public abstract class AssertionChecker {   public static List<String> *initialLabels*;  public static List<String> *finalLabels*;  public static List<String> *screenshotLabels*;   public abstract void assertionCheck();   public static void setInitialLabels(List<String> labels){  *initialLabels* = labels;  }   public static void setFinalLabels(List<String> labels){  *finalLabels* = labels;  }   public static void setScreenshotLabels(List<String> labels){  *screenshotLabels* = labels;  } } |

Listado 17 - Clase AssertionChecker

Esta clase contiene los atributos initialLabels y finalLabels y los métodos correspondientes para establecer sus valores. Estos métodos reciben las etiquetas de la pantalla al principio y al final del test, y las asignan a las listas. Las etiquetas se obtienen usando el método ya existente en el proyecto llamado “labelsDetection”.

Además, contiene el método abstracto “assertionCheck”. Este método es abstracto porque se define en cada clase test generada por Espresso en la que se quieran añadir aserciones personalizadas. Para añadir este método en la clase test, simplemente hay que hacer que la clase seleccionada extienda a “AssertionChecker” y, al final del método principal de la clase, añadir una llamada al método “assertionCheck”. Tras eso, se define el método en la misma clase, con aserciones propias que el usuario quiera hacer.

|  |
| --- |
| appCompatButton2.perform(*scrollTo*(), *click*());   assertionCheck();  }  @Override  public void assertionCheck(){  *assertTrue*(*finalLabels*.contains("Hola"));  *assertTrue*(*finalLabels*.size()>*initialLabels*.size());  List<String> editButtons = new ArrayList<String>();  for(String s: *finalLabels*){  if(s.equals("EDIT")){  editButtons.add(s);  }  }  *assertTrue*(editButtons.size()==2);  } |

Listado 18 - Llamada al método assertionCheck en el test de Espresso, seguida del método en sí

En la imagen superior se puede apreciar la llamada al método “assertionCheck” dentro del método principal de la clase test. Debajo, el método, que contiene tres aserciones que comprueban que se haya creado una nueva nota con el título “Hola” en la aplicación de MyNotes. Este método comprobará que se haya creado la nota correctamente.

Esto es suficiente para poder incluir estas aserciones personalizadas en el test de Espresso, pero no es suficiente para incluirlas en el test una vez parseado. Por lo cual, hay que realizar el siguiente cambio en la clase de parseo, llamada “TextualUITestGenerator”.

Para hacer que la clase reconozca el método “assertionCheck” y lo incluya en el documento de texto del test parseado, en la línea 165 de la clase TextualUITestGenerator, tenemos el siguiente código:

|  |
| --- |
| if(mc.getName().toString().startsWith("assertTrue")){  if(!objectTypes.contains("CUSTOM ASSERTION")){  objectTypes.add("CUSTOM ASSERTION");  selectors.add("onClass="+*FILE*);  texts.add("Check custom assertion");  }  } |

Listado 19 - Código añadido a TextualUITestGenerator para dar apoyo a las aserciones personalizadas

Esta sección de Código comprueba si hay una orden “assertTrue” en el test. Si lo hay, añade una línea al test parseado con la siguiente apariencia:

|  |
| --- |
| CUSTOM ASSERTION, UiSelector[onClass=NombreDeLaClaseTest], Check custom assertion |

Listado 20 - Aserción personalizada representada en un test parseado

Cuando esta línea ha sido incluida, no la incluirá de nuevo en caso de que haya más de una orden "assertTrue” dentro del método.

Una vez el sistema reconoce el método de aserciones personalizadas, hay que hacer que el sistema compruebe dichas aserciones. Para ello, se deben hacer algunos cambios en distintas clases del sistema.

Los primeros de estos cambios están en la clase “ReadTestCase”, dentro del método “generateTestCase”. A la hora de analizar las acciones del test, si alguna de ellas comienza por “CUSTOM ASSERTION”, a la variable “predicate” del TestCase que se está generando se le asigna el valor de esa línea del test parseado.

|  |
| --- |
| if(action.startsWith("CUSTOM ASSERTION")){  predicate = action; } |

Listado 21 - Código que asigna una aserción personalizada a la variable predicate

Luego, a la hora de generar cada acción, no se generará si se trata de una acción de aserción personalizada.

|  |
| --- |
| if (seed < 0) {  if(!action.startsWith("CUSTOM ASSERTION"))  testActions.add(generateActionFromString(action, seed, generatorType, cond1, cond2)); }else {  if(!action.startsWith("CUSTOM ASSERTION"))  testActions.add(generateActionFromString(action, seed, generatorType, cond1, cond2)); |

Listado 22 - Código que evita crear una acción si se trata de una aserción personalizada

Otro cambio a realizar está en la clase ReadTestCase, donde se añaden dos líneas de código que guardan el estado inicial y final del test, llamando a los métodos “setInitialLabels” y “setFinalLabels” de la clase “AssertionChecker”. Esto no interfiere con el resto del método, por lo cual se siguen pudiendo generar tests con aserciones del Espresso Test Recorder.

|  |
| --- |
| testCase.executeBefore(); List<String> initialState = *labelsDetection*(); AssertionChecker.*setInitialLabels*(initialState); testCase.setInitialState(initialState); testCase.executeTest(); List<String> finalState = *labelsDetection*(); testCase.setFinalState(finalState); AssertionChecker.*setFinalLabels*(finalState); Boolean eval = true; if(testCase.getPredicate() != null &&  testCase.getPredicate().toString().startsWith("CUSTOM ASSERTION")){  testCase.customEvaluate(); }else {  eval = testCase.evaluate(); } |

Listado 23 - Código del método read de la clase ReadTestCase con adiciones que permiten aserciones personalizadas

Tras esto, se inicializa la variable “eval” a “true”. Esto se hace por si el test no contiene aserciones, para que no devuelva errores en ese caso. Si el “predicate” del test existe u comienza por “CUSTOM ASSERTION” (esto indicaría que se están usando aserciones personalizadas), se llama al método “customEvaluate”, un método creado específicamente para aserciones personalizadas. En caso contrario, se llama al método ya existente que comprueba las aserciones.

El método “customEvaluate” utiliza reflexión para invocar al método “assertionCheck” de la clase test de Espresso. Si el método se ejecuta sin errores, se considerará que las aserciones son correctas. El código del método es el siguiente:

|  |
| --- |
| public void customEvaluate() throws NoSuchMethodException, InvocationTargetException, IllegalAccessException{  String predicateString = predicate.toString();  String target = predicateString.substring(predicateString.indexOf("=")+1,predicateString.indexOf("]"));  Class<?> targetClass = null;  try {  targetClass = Class.*forName*("esadrcanfer.us.alumno.autotesting."+target);  } catch (ClassNotFoundException e) {  e.printStackTrace();  }  Object test = null;  Constructor constructor = null;   try {  constructor = targetClass.getConstructor();  } catch (NoSuchMethodException e) {  e.printStackTrace();  }  try {  test = constructor.newInstance();  } catch (IllegalAccessException e) {  e.printStackTrace();  } catch (InstantiationException | InvocationTargetException e) {  e.printStackTrace();  }   if(predicate!=null) {  String methodName = "assertionCheck";  targetClass.getMethod(methodName).invoke(test, null);  } } |

Listado 24 - Método customEvaluate de la clase TestCase

Lo primero que hace el método es extraer el nombre de la clase, que se pasa en la línea del test parseado con el propósito de poder usarlo en el método. Usando el nombre, se inicializa un objeto de esa clase y se utiliza para invocar al método “assertionCheck”.

## Capturas de pantalla

Para dar soporte a capturas de pantalla, se ha optado por hacerlo posible sólo a partir de los tests parseados. Esta decisión se ha tomado porque Espresso Test Recorder es la única manera de generar tests para parsearlos posteriormente, y este no registra la acción de capturar la pantalla.

En la clase Action se ha añadido un nuevo tipo de acción, llamada ScreenshotAction. Esta, al igual que cualquier otra acción, contiene un método llamado “perform”. Dentro de este método encontramos el siguiente código:

|  |
| --- |
| @Override public void perform() throws UiObjectNotFoundException {  List<String> currentState = *labelsDetection*();  AssertionChecker.*setScreenshotLabels*(currentState);  File dir = new File("storage/emulated/0/Download/Screenshots");  if (!dir.exists()) {  dir.mkdirs();  }  WriterUtil writerUtil = new WriterUtil("Screenshot", "storage/emulated/0/Download/Screenshots");  File screenShot = writerUtil.getLogFile();  device.takeScreenshot(screenShot); } |

Listado 25 - Método perform de la clase ScreenshotAction

Este método hace dos cosas. La primera es guardar las etiquetas presentes en la pantalla en el momento de tomar la captura. Con el método “labelsDetection”, guarda las etiquetas en la lista llamada “currentState”. Luego, usa esa lista en una llamada al método setScreenshotLabels de la clase ”AssertionChecker”. Ese método simplemente pasa la lista “currentState” a la lista “screenshotLabels” de esta clase. Con esto, ya tenemos guardado en una variable el estado actual del test.

La segunda cosa que hace el método es crear un repositorio donde guardará un archivo de imagen vacío, crea tal archivo y lo sobrescribe con una captura de pantalla. Para crear el directorio donde se guardará la imagen, simplemente se crea un objeto de tipo File y se le pasa la ruta del almacenamiento interno del dispositivo. Además, se indica que si el directorio no existe, se crea uno nuevo.

Para crear el archivo de imagen vacío, se ha tenido que crear un nuevo constructor en la clase “WriterUtil”. Este constructor recibe el nombre del fichero y la ruta donde se guardará, y le añade la fecha y hora a la que se toma la captura de pantalla y una extensión “.png”.

|  |
| --- |
| public WriterUtil(String fileName, String path) {  String timeLog = new SimpleDateFormat("yyyyMMdd\_HHmmss").format(Calendar.*getInstance*().getTime());  String filename = fileName + timeLog+ ".png";  File dir = new File(path);  this.logFile = new File(dir, filename); } |

Listado 26 - Constructor de la clase WriterUtil modificado para crear imágenes vacías

Por último, con las dos últimas líneas del método “perform” se obtiene la imagen que captura la pantalla y se escribe sobre el archivo.

|  |
| --- |
| File screenShot = writerUtil.getLogFile(); device.takeScreenshot(screenShot); |

Listado 27 - Código que captura la pantalla y asigna la imagen al archivo de imagen vacío

Gracias a este método, se generará una captura de la pantalla en la carpeta “Storage/Emulated/0/Download/Screenshots” del dispositivo simulado y se tendrá acceso a la lista de valores String con las etiquetas del test en ese momento, llamada “screenshotLabels”, para poder usarlas en aserciones personalizadas.

# Manual de usuario

A continuación, se explicarán los pasos a seguir para configurar el proyecto previamente a su uso y para utilizarlo de forma correcta, independientemente del sistema en el que se use. Primero, se explicará la configuración previa. Esto incluye las librerías a utilizar y el contenido del fichero build.gradle. Para descargar el proyecto, hay que acceder a su repositorio en GitHub y clonarlo. La dirección del repositorio es: <https://github.com/luiparlop1/espresso-parsing.git>. La rama más avanzada es “master”, por lo cual se recomienda estar en esa rama para ver el proyecto en su estado más reciente.

## Configuración previa

En este proyecto, las dependencias deberían estar incluidas en el archivo build.gradle. En caso de que no, a continuación estarán listadas para poder incluirlas. Durante el desarrollo del trabajo, surgieron algunos errores dado a la incorrecta configuración previa del anterior proyecto. De todas formas, esas dependencias están incluidas tanto en el proyecto como en esta sección.

En primer lugar, para poder acceder a los archivos de configuración del proyecto, el proyecto deberá visualizarse en la pestaña “Android” de la barra de navegación llamada “Project” que se sitúa en la parte superior izquierda de la pantalla.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 6 - Selección del modo de vista del proyecto

Una vez hecho esto, se podrán encontrar dos archivos llamados build.gradle en l carpeta “Gradle Scripts”. Uno de ellos pertenece al proyecto MyNotes y el otro al módulo del mismo nombre.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 7 - Carpeta de archivos Gradle

Accediendo al archivo del proyecto, veremos el siguiente contenido:

|  |
| --- |
| // Top-level build file where you can add configuration options common to all sub-projects/modules.  buildscript **{** repositories **{** google()  jcenter()    **}** dependencies **{** classpath 'com.android.tools.build:gradle:4.1.2'    // NOTE: Do not place your application dependencies here; they belong  // in the individual module build.gradle files  **} }** allprojects **{** repositories **{** google()  jcenter()    **} }** task clean(type: Delete) **{** delete rootProject.buildDir **}** |

Listado 28 - Archivo build.gradle del proyecto

Esto no es necesario modificarlo, puesto que es la configuración del proyecto base y no hizo falta cambiarla en ningún momento durante el desarrollo del trabajo. En el otro archivo, el perteneciente al módulo individual MyNotes, encontramos este contenido:

|  |
| --- |
| apply plugin: 'com.android.application'  android **{** compileSdkVersion 30  defaultConfig **{** applicationId "esadrcanfer.us.alumno.autotesting"  minSdkVersion 18  targetSdkVersion 30  versionCode 1  versionName "1.0"  testInstrumentationRunner 'androidx.test.runner.AndroidJUnitRunner'  multiDexEnabled true  **}** buildTypes **{** release **{** minifyEnabled false  proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  **}  }** testOptions **{** unitTests.returnDefaultValues = true  **}** compileOptions **{** sourceCompatibility JavaVersion.*VERSION\_1\_8* targetCompatibility JavaVersion.*VERSION\_1\_8* **}** packagingOptions **{** exclude 'META-INF/spring.tooling'  exclude 'META-INF/spring.handlers'  exclude 'META-INF/spring.schemas'  exclude 'META-INF/license.txt'  exclude 'META-INF/notice.txt'  exclude 'META-INF/LICENSE'  exclude 'META-INF/LICENSE.txt'  exclude 'META-INF/NOTICE'  exclude 'META-INF/NOTICE.txt'  exclude 'META-INF/ASL2.0'  exclude 'META-INF/DEPENDENCIES'  exclude("META-INF/\*.kotlin\_module")  **}** sourceSets **{** *main* **{** java.srcDirs = ['src/main/java'] **}** test **{** java.srcDirs = ['src/test/java'] **}  } }** dependencies **{** implementation 'androidx.multidex:multidex:2.0.1'  implementation 'net.sf.extjwnl:extjwnl:2.0.1'  implementation 'net.sf.extjwnl:extjwnl-data-wn31:1.2'  implementation fileTree(include: ['\*.jar'], dir: 'libs')  implementation 'androidx.legacy:legacy-support-v4:1.0.0'  testImplementation 'org.junit.jupiter:junit-jupiter-api:5.2.0'  testImplementation 'junit:junit:4.13.2'  androidTestImplementation 'androidx.test.ext:junit:1.1.2'  androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  androidTestImplementation 'androidx.test.uiautomator:uiautomator:2.2.0'  implementation "org.springframework:spring-core:4.2.0.RELEASE"  implementation "org.springframework:spring-expression:4.2.0.RELEASE"  implementation "org.springframework:spring-context:4.2.0.RELEASE"  implementation 'org.apache.commons:commons-math3:3.6.1'    implementation 'com.github.javaparser:javaparser-symbol-solver-core:3.18.0'  implementation 'com.github.javaparser:javaparser-core-serialization:3.18.0'   implementation 'com.github.krraghavan:xeger:1.0.0-RELEASE'   androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  androidTestImplementation 'androidx.test:rules:1.4.0-alpha05'  implementation 'com.google.android.material:material:1.3.0'  implementation 'androidx.navigation:navigation-fragment-ktx:2.3.5'  implementation 'androidx.navigation:navigation-ui-ktx:2.3.5'  implementation 'androidx.constraintlayout:constraintlayout:2.0.4'   testImplementation 'androidx.test.uiautomator:uiautomator:2.2.0'  testImplementation 'androidx.test.ext:junit:1.1.2'   testImplementation 'androidx.test.espresso:espresso-core:3.3.0'  **}** |

Listado 29 - Archivo build.gradle del módulo MyNotes

Todas esas líneas de código se tratan de especificaciones para que el proyecto funcione correctamente con sus dependencias y librerías. Para el desarrollo de este proyecto, además de las librerías que había incluidas, ha habido que utilizar otras, concretamente la de Xeger para crear expresiones regulares y la de Apache Commons para hacer uso de Math, con el fin de efectuar algunas operaciones sobre valores numéricos. En caso de que el archivo no sea idéntico a este, ha de copiarse el código de esta memoria y pegarse en el archivo para evitar errores.

Una vez comprobado que el estado de los archivos “build.gradle” es correcto y contienen todas las dependencias necesarias, se puede considerar que el proyecto está configurado correctamente. Bastaría con hacer una sincronización con los archivos Gradle.

## Creación de tests con el archivo de configuración

Para crear tests con un archivo de configuración, se siguen los siguientes pasos:

-Primero, se crea un test de Espresso. Esto se hace en el menú “Run”, situado en la barra de herramientas. En el menú desplegable, se elige la opción “Record Espresso Test”. Esto abrirá el software de Espresso para grabar tests automatizados.

Captura de pantalla de computadora

Descripción generada automáticamente

Ilustración 8 - Botón para comenzar a grabar un test de Espresso

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Ilustración 9 - Proceso de grabación de un test de Espresso

Tras esto, basta con realizar algunas acciones que el software de grabación registrará y convertirá en tests automáticamente cuando se pulse el botón de “OK”. En este proyecto no es necesario añadir una aserción al test, dado que ha sido modificado para que no considere erróneos los tests que no tengan aserciones.

-Una vez generada la clase java con el código del test (la cual se puede encontrar en la carpeta de nombre “esadrcanfer.us.alumno.autotesting (androidTest)”, basta con acceder a la clase llamada “TestConverter”, situada en l carpeta de nombre “esadrcanfer.us.alumno.autotesting (test)”. Dentro de esta clase, hay que cambiar el nombre del test a parsear por el nombre del test que se acaba de crear y ejecutar la clase.

|  |
| --- |
| @Test public void crearNotaTest() throws IOException {  TextualUITestGenerator parse = new TextualUITestGenerator();  parse.textualUITestGenerator("NombreDeTuTest","src/androidTest/java/esadrcanfer/us/alumno/autotesting/", BuildConfig.*APPLICATION\_ID*); } |

Listado 30 - Método crearNotaTest de la clase TestConverter

Esto iniciará el proceso de parseo del test a lenguaje textual. Una vez hecho esto, se habrá generado el test en formato textual en la carpeta “assets”.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ilustración 10 - Carpeta assets del proyecto

Como se puede observar, hay un fichero llamado “configu” en esta carpeta. Se trata del fichero de configuración y viene incluido en el proyecto.

-Luego, para ejecutar el test textual, tanto este como el fichero de configuración deben ser descargados e introducidos en la memoria interna del dispositivo simulado. Para ello, el dispositivo debe estar encendido. Para acceder a la memoria interna, basta con desplegar el menú “Device File Explorer”, situado en la esquina inferior derecha de la interfaz de Android Studio. En la carpeta Storage>Emulated>0>Download se hace click derecho, se elige la opción “Upload” y se escogen los dos archivos (el test generado por Espresso Test Recorder y el archivo de configuración).

Una captura de pantalla de una computadora

Descripción generada automáticamente

Ilustración 11 - Carpeta del dispositivo móvil donde se suben los tests parseados

-Una vez hecho esto, para ejecutar el test, se accede a la clase “ParsedTestReader”, situada en la carpeta de nombre “esadrcanfer.us.alumno.autotesting (androidTest)”. Tras cambiar el nombre del test a ejecutar por el del test que se ha introducido en la memoria del dispositivo móvil, se ejecuta la clase y el test usará el archivo de configuración para generar los datos textuales indicados por el usuario. Es importante destacar que, en caso de que haya campos de texto a rellenar en el test, el archivo de configuración debe contener, como mínimo, tantas entradas como campos de texto a rellenar. En caso contrario, la ejecución de la prueba devolverá un error “IndexOutOfBoundsException”, el cual indica que no hay suficientes entradas en el archivo de configuración como para rellenar todos los campos de texto.

En adición al repertorio ya existente de acciones a realizar en los tests, tras aplicar las mejoras al proyecto, ahora se puede pulsar el botón de retroceder en el dispositivo durante la grabación de los tests de Espresso. No es necesario hacer nada en particular para que el sistema reconozca esa acción.

## Uso de aserciones personalizadas

Para usar aserciones personalizadas, es necesario seguir una serie de pasos específicos:

-Primero, se crea un test con la opción de “Record Espresso Test”, vista en el apartado anterior.

-Una vez creado el test, se accede a la clase y se hace que extienda a la clase “AssertionChecker”. Para ello, basta con añadir el texto “extends AssertionChecker” tras el nombre de la clase.

-Luego, se accede al método principal de la clase test, del mismo nombre que la clase, y se añade al final del mismo una simple llamada al método “assertionCheck”.

|  |
| --- |
| assertionCheck(); |

Listado 31 - Llamada al método assertionCheck que debe incluirse al final de un test de Espresso

-Finalmente, se añade el método “assertionCheck”, de tipo void, dentro del cual se añaden tantas comprobaciones como se crean oportunas. Es importante que el nombre del método sea este debido a que es el método abstracto de la clase AssertionChecker, a la cual extiende ahora nuestro test. El método es de tipo void porque realmente no devuelve nada, sólo da un error si alguna de las aserciones falla. Por ello, sólo deben utilizarse comprobaciones de tipo “assert” (como assertTrue o assertThat). A continuación se muestra un ejemplo del método que contiene algunas aserciones.

|  |
| --- |
| @Override  public void assertionCheck(){  *assertTrue*(*finalLabels*.contains("Hola"));  *assertTrue*(*finalLabels*.size()>*initialLabels*.size());  List<String> editButtons = new ArrayList<String>();  for(String s: *finalLabels*){  if(s.equals("EDIT")){  editButtons.add(s);  }  }  *assertTrue*(editButtons.size()==2); } |

Listado 32 - Ejemplo de un método assertionCheck con aserciones personalizadas

Como se puede observar, este método contiene tres aserciones. Una comprueba que la pantalla contenga el texto “Hola”, que correspondería al título de la nota que se ha creado en ese mismo test. La siguiente comprueba que la cantidad de etiquetas de la pantalla al final del test sea mayor a la cantidad al inicio. La tercera comprueba que haya más etiquetas de título “EDIT” en pantalla al final del test, ya que contando la del botón de editar la nota de ejemplo de la aplicación y la del mismo botón de la nota recientemente creada deberían ser las dos únicas.

## Capturas de pantalla

Las capturas de pantalla son muy simples de realizar. Se hace siguiendo los siguientes pasos.

-Primero, se añade el texto “SCREENSHOT” como una de las líneas del test una vez haya sido parseado.

-Como una captura de pantalla también se trata de una acción como cualquier otra, es necesario sumar uno al número que representa la cantidad de acciones del test en el archivo de texto. En la tercera línea del archivo, se cambia el número que haya por uno con un valor mayor por una unidad.

Si se desea, se pueden generar múltiples capturas de pantalla por test. De todas formas, solo la última guardará el estado de la pantalla en ese momento, debido a que el sistema actualmente sólo está optimizado para la toma de una captura de pantalla.

Texto

Descripción generada automáticamente con confianza media

Ilustración 12 - Captura de pantalla almacenada en el dispositivo móvil simulado

# Pruebas

Para comprobar el funcionamiento correcto de las mejoras aplicadas al proyecto, se ha realizado una serie de pruebas que cubren todos los casos de uso posibles de las funcionalidades nuevas. En este apartado se detallarán las pruebas realizadas.

En primer lugar, se han realizado pruebas en la aplicación MyNotes, utilizando el archivo de configuración “config.txt” para crear distintas notas con nombres diferentes generados por la jerarquía de generadores. Estos nombres serán de distinto tipo dependiendo de la entrada del archivo de configuración. Para estas pruebas, la clase test utilizada ha sido una directamente parseada, ya que los distintos generadores no tienen influencia alguna sobre clases generadas por Espresso.

Cada prueba consiste en la creación de una nota, poniéndole un título y pulsando el botón de guardar.

## Pruebas de la jerarquía de generadores con el archivo de configuración en MyNotes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo de generador | Entradas del archivo | Resultados | Éxito | ID de prueba |
| Number | number-1/5 | 1 | ✓ | P-1 |
| number-6/10 | 10 | P-2 |
| number-4/8 | 6 | P-3 |
| Number from List | numberFromList-1,3,5,7/ | 1 | ✓ | P-4 |
| numberFromList-100,300,500,700/ | 100 | P-5 |
| numberFromList-123,345,567,789/ | 567 | P-6 |
| Number from probability List | numberFromProbabilityList-1,3,5,7/ | 3 | ✓ | P-7 |
| numberFromProbabilityList-100,300,500,700/ | 700 | P-8 |
| numberFromProbabilityList-123,345,567,789/ | 789 | P-9 |
| Word | word-1/ | unsatisfactory | ✓ | P-10 |
| word-2/ | estimable unchivalrously | P-11 |
| word-3/ | flush insufficiently tougher | P-12 |
| Regex | regex-phone/ | 959 778 757 | ✓ | P-13 |
| regex-email/ | gogedefu72@hotmail.com | P-14 |
| Given | given-Hola/ | Hola | ✓ | P-15 |
| given-Prueba/ | Prueba | P-16 |
| given-Éxito/ | Éxito | P-17 |
| Increment | increment-1/ | 1.16100371 | ✓ | P-18 |
| increment-599/ | 599.45236082 | P-19 |
| increment-13/ | 12.07329640 | P-20 |
| Reflection | reflection-Pet/ | voice, interoceptive, chip: 6786 | ✓ | P-21 |
| reflection-Person/ | zero, inconsiderate | P-22 |

Todas las pruebas en la aplicación de MyNotes resultaron ser exitosas, consiguiendo salidas generadas aleatoriamente por cada tipo distinto de generador. La fecha de ejecución de todas estas pruebas fue el 25/04/2022.

## Pruebas de la jerarquía de generadores con el archivo de configuración en otras aplicaciones

Para asegurar que el sistema de generadores con el archivo de configuración no solo funciona en el contexto de la aplicación MyNotes, se realizaron pruebas en otras aplicaciones. En concreto, se probó el funcionamiento en algunas que contenían campos de texto donde poder introducir entradas aleatoriamente generadas, dándole uso a los generadores.

Se realizaron pruebas en la aplicación de Contactos, Reloj, Google Chrome y Gmail.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Aplicación | Entrada del archivo | Resultado | Fecha | Éxito | ID de prueba |
| Contactos | word-1/  given-Pérez/  number-2/5  numberFromList-111111111,222222222,333333333,444444444/  regex-email/ | Contacto creado con nombre “lispingly Pérez 3”, número de teléfono “111111111” y email “fawebiki11@gmail.com" | 26/04/2022 | ✓ | P-23 |
| Reloj | regex-email/ | Alarma editada con el nuevo nombre: “bavinusa97@gmail.com” | 28/04/2022 | ✓ | P-24 |
| Google Chrome | word-2/ | Búsqueda de Google realizada con: “dismount innatentive” | 28/04/2022 | ✓ | P-25 |
| Gmail | regex-email/  given-Probando a enviar un correo/  reflection-Pet/ | Correo guardado como borrador con destinatario “pificeke73@gmail.com”, asunto “Probando a enviar un correo” y cuerpo “misappropriated,woolly-headed, chip: 123123123” | 28/04/2022 | ✓ | P-26 |

Las pruebas ejecutadas en otras aplicaciones resultaron ser exitosas. Para probar las distintas aplicaciones, se modificaron pruebas textuales del proyecto desarrollado por Gonzalo Aguilar Hermoso. Estas pruebas tenían dos años de antigüedad, por lo cual tuvieron que ser modificadas para adaptarse al sistema más reciente. No obstante, tras ser modificadas, se ejecutaron exitosamente, y los generadores funcionaron como se esperaba.

## Pruebas de retroceder

Para probar la acción de retroceder, simplemente se ha utilizado una de las pruebas anteriores para cada aplicación y se le ha introducido una línea que consiste en la acción de retroceder. A continuación, los resultados.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aplicación | Éxito de parseo | Éxito en prueba textual | Fecha | ID de prueba |
| MyNotes | ✓ | ✓ | 29/04/2022 | P-27 |
| Contactos | ✓ | ✓ | 29/04/2022 | P-28 |
| Reloj | ✓ | ✓ | 29/04/2022 | P-29 |
| Google Chrome | ✓ | ✓ | 29/04/2022 | P-30 |
| Gmail | ✓ | ✓ | 29/04/2022 | P-31 |

Al ser una acción simple, no depende de ninguna aplicación específicamente, por lo que funciona en todas las pruebas en las que se ha introducido.

## Pruebas de aserciones personalizadas

Para estas pruebas se ha hecho lo mismo que para las pruebas de retroceder, incluyendo al final de cada una un método que contiene una aserción personalizada que comprueba que haya un elemento determinado en pantalla al final de la ejecución de la prueba.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aplicación | Éxito de parseo | Éxito en prueba textual | Fecha | ID de prueba |
| MyNotes | ✓ | ✓ | 29/04/2022 | P-32 |
| Contactos | ✓ | ✓ | 29/04/2022 | P-33 |
| Reloj | ✓ | ✓ | 29/04/2022 | P-34 |
| Google Chrome | ✓ | ✓ | 29/04/2022 | P-35 |
| Gmail | ✓ | ✓ | 29/04/2022 | P-36 |

Este sistema de aserciones personalizadas también se trata de uno universal, por lo cual puede utilizarse en cualquier aplicación.

## Pruebas de captura de pantalla

Para las pruebas de captura de pantalla, al tratarse de un comando que se añade en el archivo de la prueba una vez parseado a lenguaje textual, no se comprueba que se parsee correctamente, dado que el parseo no forma parte del proceso. Al igual que con las pruebas de retroceder, se ha añadido una línea a los tests textuales con el comando de tomar una captura de pantalla y se ha comprobado que se haga una captura.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aplicación | Éxito de parseo | Éxito en prueba textual | Fecha | ID de prueba |
| MyNotes | ✓ | ✓ | 29/04/2022 | P-37 |
| Contactos | ✓ | ✓ | 29/04/2022 | P-38 |
| Reloj | ✓ | ✓ | 29/04/2022 | P-39 |
| Google Chrome | ✓ | ✓ | 29/04/2022 | P-40 |
| Gmail | ✓ | ✓ | 29/04/2022 | P-41 |

Por último, para cerrar la sección de pruebas, se mostrará la matriz de trazabilidad de las pruebas.

## Matriz de trazabilidad

La siguiente tabla muestra la matriz de trazabilidad del proyecto, donde se observa que todos los requisitos han sido validados por las pruebas realizadas.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | R-01 | R-02 | R-03 | R-04 | R-05 | R-06 |
| P-1 | ✓ | ✓ | ✓ | - | - | - |
| P-2 | ✓ | ✓ | ✓ | - | - | - |
| P-3 | ✓ | ✓ | ✓ | - | - | - |
| P-4 | ✓ | ✓ | ✓ | - | - | - |
| P-5 | ✓ | ✓ | ✓ | - | - | - |
| P-6 | ✓ | ✓ | ✓ | - | - | - |
| P-7 | ✓ | ✓ | ✓ | - | - | - |
| P-8 | ✓ | ✓ | ✓ | - | - | - |
| P-9 | ✓ | ✓ | ✓ | - | - | - |
| P-10 | ✓ | ✓ | ✓ | - | - | - |
| P-11 | ✓ | ✓ | ✓ | - | - | - |
| P-12 | ✓ | ✓ | ✓ | - | - | - |
| P-13 | ✓ | ✓ | ✓ | - | - | - |
| P-14 | ✓ | ✓ | ✓ | - | - | - |
| P-15 | ✓ | ✓ | ✓ | - | - | - |
| P-16 | ✓ | ✓ | ✓ | - | - | - |
| P-17 | ✓ | ✓ | ✓ | - | - | - |
| P-18 | ✓ | ✓ | ✓ | - | - | - |
| P-19 | ✓ | ✓ | ✓ | - | - | - |
| P-20 | ✓ | ✓ | ✓ | - | - | - |
| P-21 | ✓ | ✓ | ✓ | - | - | - |
| P-22 | ✓ | ✓ | ✓ | - | - | - |
| P-23 | ✓ | ✓ | ✓ | - | - | - |
| P-24 | ✓ | ✓ | ✓ | - | - | - |
| P-25 | ✓ | ✓ | ✓ | - | - | - |
| P-26 | ✓ | ✓ | ✓ | - | - | - |
| P-27 | ✓ | ✓ | ✓ | ✓ | - | - |
| P-28 | ✓ | ✓ | ✓ | ✓ | - | - |
| P-29 | ✓ | ✓ | ✓ | ✓ | - | - |
| P-30 | ✓ | ✓ | ✓ | ✓ | - | - |
| P-31 | ✓ | ✓ | ✓ | ✓ | - | - |
| P-32 | ✓ | ✓ | ✓ | - | ✓ | - |
| P-33 | ✓ | ✓ | ✓ | - | ✓ | - |
| P-34 | ✓ | ✓ | ✓ | - | ✓ | - |
| P-35 | ✓ | ✓ | ✓ | - | ✓ | - |
| P-36 | ✓ | ✓ | ✓ | - | ✓ | - |
| P-37 | ✓ | ✓ | ✓ | - | - | ✓ |
| P-38 | ✓ | ✓ | ✓ | - | - | ✓ |
| P-39 | ✓ | ✓ | ✓ | - | - | ✓ |
| P-40 | ✓ | ✓ | ✓ | - | - | ✓ |
| P-41 | ✓ | ✓ | ✓ | - | - | ✓ |

# Conclusiones

Tras finalizar el desarrollo de las funcionalidades propuestas al inicio del trabajo en el proyecto, se puede afirmar que se ha ampliado exitosamente el conjunto de acciones disponibles a la hora de realizar pruebas automatizadas.

En primer lugar, se ha desarrollado una variedad de generadores de texto con valores de distintos tipos, capaces de generar valores textuales completamente aleatorizados. Esto da la posibilidad al desarrollador de crear pruebas automatizadas con valores realistas, dependiendo de la información a rellenar en cada campo. Con la escritura de un simple archivo de configuración, se puede indicar al sistema el tipo de generador a utilizar para cada campo de texto para cada iteración de un test, simplificando mucho la tarea de desarrollo de tests, ya que ahora no sólo se pueden generar palabras aleatorias, sino que también se pueden generar números aleatorios, números escogidos aleatoriamente de una lista dada, números de teléfono y direcciones de correo electrónico generadas aleatoriamente a partir de expresiones regulares, o incluso información sobre personas y mascotas con nombres, apellidos, especies o chips de identidad generados aleatoriamente por reflexión de Java.

En segundo lugar, se ha implementado la opción de retroceder mediante la pulsación del botón de “atrás” del dispositivo móvil. Esto, aunque es una simple acción de retroceder, ayuda al desarrollador ya que en algunas aplicaciones, entrar en ciertas secciones donde es necesario ver la pantalla para determinar a qué campo de texto dirigirse, se abre automáticamente el teclado del dispositivo, imposibilitando la visualización de la pantalla. Esto causa un error al ejecutar el test. De todas formas, al poder retroceder, ahora es posible pulsar el botón para cerrar el teclado, dando lugar al resto de la prueba.

Además, se ha dado soporte a un sistema completo de aserciones personalizadas. Anteriormente, tan solo se podían realizar aserciones incluidas en Espresso Test Recorder. Esto limitaba las aserciones a estar al final de la prueba, y por lo cual solamente permitían comprobar el estado final de la pantalla, al concluir la prueba. Con el nuevo sistema, ahora el usuario es capaz de generar cualquier tipo de aserción mediante el uso de instrucciones de tipo “assert”. Basta con añadir un método al final de cada clase test con las aserciones que desee comprobar el usuario en su interior, y el sistema usará reflexión para invocar al método y comprobarlas. Dado que se guardan el estado inicial, el final e incluso uno intermedio, es posible comprobar muchas más aserciones que antes.

Por último, se ha añadido la opción de capturar la pantalla en cualquier momento de la ejecución del test. Esto genera una imagen de la pantalla en el momento de tomar la captura, dentro del almacenamiento interno del sistema. Además, haciendo referencia a la mejora anterior, se guarda el estado de la prueba en el momento de hacer la captura, para utilizarlo a la hora de comprobar aserciones personalizadas.

Abordar un proyecto grande como este no es tarea fácil. A lo largo de todas las horas que he dedicado a este proyecto, he llegado a varias conclusiones, las cuales listaré a continuación.

La primera conclusión y quizás la más importante es que, a pesar de ser un reto, los años que he dedicado a estudiar el grado de ingeniería informática han ayudado en gran medida. No sólo porque la programación ha sido realizada en lenguaje java, sino porque a lo largo del grado he desarrollado la habilidad de aprender a trabajar con software desconocido, en un entorno extraño y sobre un proyecto que nunca había abordado antes. Este ha sido el caso en más de una asignatura, con la única diferencia de que en esos casos disponía de compañeros de grupo para ayudarme con el desarrollo del proyecto. Aunque no lo haya hecho completamente sin ayuda (ya que el tutor José Antonio Parejo me ha ayudado en diversas ocasiones), desde luego ha sido un reto considerablemente mayor que cualquier otro que haya afrontado en estos años más recientes de mi vida. Tanto las horas trabajadas como el esfuerzo de investigación y aprendizaje han sido también mucho mayores.

La segunda conclusión es al respecto del trabajo de alumnos anteriores. Este proyecto ya ha pasado por las manos de otros tres alumnos distintos desde su comienzo. Debido a esto, durante mi aportación al sistema he encontrado una serie de errores. Las causas de estos errores, a la vez que a gravedad de los mismos, ha sido distinta en cada uno. Estos errores han sido mencionados en distintos apartados de la memoria. Algunos eran mínimos y podían ser ignorados, pero otros incluso llegaban a impedir el correcto funcionamiento del sistema, dando lugar a resultados erróneo de las pruebas. La conclusión a la que he llegado es que cualquier sistema informático, venga de donde venga, contiene errores. Si no se ha encontrado alguno es que no se ha buscado lo suficiente. El ser humano no es perfecto, por lo cual tampoco puede serlo ninguna de sus creaciones, al menos no por ahora. Es cierto que se han de esperar más errores de un proyecto elaborado por alumnos de un grado, ya que la falta de extensa experiencia al trabajar en sistemas informáticos puede dar lugar a un sistema de reconocimiento y resolución de errores menos eficiente que aquel de un equipo de trabajo profesional en una empresa, pero a día de hoy se siguen encontrando errores en software de empresas tan importantes como Google o Amazon.

La tercera conclusión ha sido extraída gracias a mi propio sistema de reparto de horas de trabajo. Inicialmente, creé un calendario desde el momento de comienzo del proyecto hasta la fecha de fin del mismo, con el número de horas que iba a dedicar a trabajar, pero llegó un momento en el que comencé a incumplir ese horario, ya que consideraba mi progreso en el proyecto “estancado” dado a que encontraba errores que consideraba que no me permitían seguir. Viendo que no iba a cumplir con el horario a partir de un momento dado, opté por racionalizar las horas restantes hasta el día de la entrega. Esto me permitió tener una cantidad de horas establecida para cada día de la semana, y esta cantidad no variaba de un día para otro. La conclusión es que al tener una cantidad fija de horas diarias para trabajar da estructura al día de una persona. Con la pandemia que hemos sufrido (y cuyos efectos aún están muy presentes), se ha desequilibrado hasta cierto punto la vida de todo el mundo. De todas formas, tener una obligación diaria que siempre trata de lo mismo devuelve cierta estabilidad al día a día de una persona.

Resumiendo, todas las tareas propuestas para el proyecto se han realizado exitosamente en el tiempo indicado, y mi experiencia con el trabajo ha sido positiva. Además, trabajar con un proyecto real, y no uno simulado con el fin de conseguir una puntuación para una asignatura, me ha dado una imagen más realista del mundo laboral y de los muchos retos que me depara el futuro.