UNIVERSIDAD POLITÉCNICA SALESIANA ECUADOR

POSGRADOS

MAESTRÍA EN

SOFTWARE CON MENCIÓN EN DISEÑO DE ARQUITECTURA DE SISTEMAS

RPC-SO-34-NO.778-2021

OPCIÓN DE TITULACIÓN:

PROYECTO DE TITULACIÓN CON COMPONENTES DE INVESTIGACIÓN APLICADA Y/O DE DESARROLLO

TEMA:

ANÁLISIS COMPARATIVO EN PRUEBAS DE REGRESIÓN CON INTEGRACIÓN CONTINUA ENTRE SELENIUM Y APPIUM EN LA APLICACIÓN JAMOVIL DE LA COOPERATIVA AHORRO Y CREDITO JARDÍN AZUAYO

AUTORES:

SANTIAGO DAVID CORDERO CRESPO
CRISTIAN IVÁN IDROVO TAPIA

DIRECTOR:

MAXIMO GIOVANI TANDAZO ESPINOZA

CUENCA – ECUADOR
2024

Autores:

Santiago David Cordero Crespo
Ingeniero en Sistemas Mención Gestión de Aplicaciones.
Candidato a Magíster en Software con mención en Diseño de Arquitectura de Sistemas por la Universidad Politécnica Salesiana – SedeCuenca.
scordero@est.ups.edu.ec

Cristian Iván Idrovo Tapia
Ingeniero en Sistemas Mención Gestión de Aplicaciones.
Candidato a Magíster en Software con mención en Diseño de Arquitectura de Sistemas por la Universidad Politécnica Salesiana – SedeCuenca.
cidrovo@est.ups.edu.ec|

Dirigido por:

Máximo Giovani Tandazo Espinoza
Ingeniero en Sistemas Computacionales.
Magister en Administración de Empresas con Mención en Sistemas de Información Gerencial.
mtandazo@ups.edu.ec

Análisis comparativo en pruebas de regresión con integración continua entre SELENIUM y APPIUM en la aplicación JAMOVIL de la cooperativa ahorro y credito Jardín Azuayo

DEDICATORIA

El presente trabajo de tesis lo dedico a mi familia, por brindarme su amor incondicional y sus sacrificios constantes que me permitieron seguir adelante, gracias por creer siempre en mí y en mis capacidades para alcanzar cada una de las metas que me he propuesto en mi vida.

A mi madre por estar pendiente de mi en todo momento y ser mi más grande inspiración y fortaleza.

A mis suegros por apoyarme siempre y más durante este tiempo de estudio, estuvieron pendientes de mí y me brindaron el respaldo que necesitaba aun en momentos complicados.

Finalmente, a todos aquellos que, de alguna manera, contribuyeron a la realización de este trabajo. Sus aportes, muchas veces silenciosos, fueron de gran valor.

Cristian Ivan Idrovo Tapia

AGRADECIMIENTO

El presente trabajo de tesis es el resultado del esmero y compromiso que hemos puesto mi compañero Santiago Cordero y el mío propio.

Quiero agradecer en primer lugar a Dios por darme la fortaleza y sabiduría necesarias para completar este proyecto.

A mi lugar de trabajo COAC Jardín Azuayo por brindarme la oportunidad de forjar mi carrera profesional y ser parte de mi superación personal.

Al Msc. Máximo Tandazo Espinoza por habernos apoyado durante la elaboración de este tema de tesis, sus aportes fueron de gran ayuda para cumplir con este objetivo.

Finalmente, un agradecimiento muy especial a mi esposa, mi madre y mis suegros quienes me apoyaron desde el momento en que decidí realizar mi estudio de maestría, estuvieron siempre pendientes de mí.

Cristian Ivan Idrovo Tapia

DEDICATORIA

A mi esposa y a mis hijas por su apoyo incondicional durante esta etapa académica que me ha permitido crecer profesionalmente. A mis padres, quienes con orgullo me han visto escalar nuevos peldaños y han sido mi fuente constante de inspiración y fortaleza.

A mis compañeros de la COAC Jardín Azuayo, por su comprensión, flexibilidad y apoyo constante. Gracias a su colaboración, he podido equilibrar mis responsabilidades laborales y académicas de manera efectiva.

Finalmente, a todas las personas que han contribuido con sus ideas y mejoras a este trabajo de titulación. Su ayuda ha sido fundamental para la realización de este proyecto.

Santiago David Cordero Crespo

AGREDECIMIENTO

En primer lugar, agradezco a Dios por la salud que ha brindado a mi familia y a mí, permitiéndonos avanzar con fortaleza y determinación. A mi esposa, por su apoyo incondicional y por estar a mi lado durante todo este proceso académico, brindándome el ánimo necesario para completar mis estudios.

A mis compañeros, colegas y amigos de la COAC Jardín Azuayo, especialmente a Cristian Idrovo, con quien he compartido conocimientos y experiencias enriquecedoras que han sido fundamentales para mi desarrollo profesional y personal.

Al Msc. Máximo Tandazo Espinoza, por su valioso acompañamiento durante la elaboración de este trabajo de titulación. Su orientación y apoyo han sido cruciales para la culminación exitosa de este proyecto.

Finalmente, a mi madre, que ha estado siempre pendiente de mi bienestar y progreso, brindándome su amor y aliento inquebrantables en cada paso de este camino.

Santiago David Cordero Crespo

Tabla de Contenido

ANÁLISIS COMPARATIVO EN PRUEBAS DE REGRESIÓN CON INTEGRACIÓN CONTINUA ENTRE SELENIUM Y APPIUM EN LA APLICACIÓN JAMOVIL DE LA COOPERATIVA AHORRO Y CREDITO JARDÍN AZUAYO

AUTOR(ES):

SANTIAGO DAVID CORDERO CRESPO Y
CRISTIAN IVAN IDROVO TAPIA

Resumen

El propósito de este análisis es abordar la importancia crítica de las pruebas de regresión en el mantenimiento de la estabilidad y rendimiento a largo plazo de la aplicación móvil JAMOVIL perteneciente a la Cooperativa de Ahorro y Crédito Jardín Azuayo. Las pruebas de regresión son esenciales para identificar posibles efectos secundarios no deseados que pudieran surgir con la introducción de nuevas funcionalidades o cambios en el código de la aplicación.

En cuanto a las herramientas seleccionadas, Selenium y Appium son reconocidas por su eficacia en la automatización de pruebas, cada una especializada en su respectivo entorno: Selenium para aplicaciones web/móviles y Appium para aplicaciones móviles. La elección entre ellas no solo se basará en la popularidad general, sino en una evaluación detallada de las necesidades específicas del proyecto, considerando la naturaleza de la aplicación y la compatibilidad con la plataforma de desarrollo.

La integración continua, es un aspecto crucial del proceso de desarrollo de software moderno, permite la ejecución automatizada de cada iteración del desarrollo, que facilita la identificación temprana de errores y mejora la eficiencia en la entrega continua de nuevas funcionalidades.

La aplicación JAMOVIL será el eje central del análisis comparativo, donde se evaluará cada herramienta en términos de compatibilidad con la aplicación móvil, ventajas y desventajas en pruebas de regresión. Los resultados obtenidos proporcionarán información valiosa para tomar decisiones informadas sobre la selección de la herramienta que mejor se adapte a los requisitos del proyecto y las características de la aplicación móvil.

Palabras clave:

Pruebas de regresión, integración continua, aplicación móvil, análisis

Abstract

The purpose of this analysis is to address the critical importance of regression testing in maintaining the long-term stability and performance of the JAMOVIL mobile application, which belongs to the Jardín Azuayo Savings and Credit Cooperative. Regression testing is essential to identify potential unwanted side effects that may arise with the introduction of new functionality or changes to the application code.

Regarding the selected tools, Selenium and Appium are recognized for their effectiveness in test automation, each specialized in their respective environment: Selenium for web/mobile applications and Appium for mobile applications. The choice between them will not only be based on general popularity but also on a detailed evaluation of the specific needs of the project, considering the nature of the application and compatibility with the development platform.

Continuous integration, a crucial aspect of the modern software development process, allows for the automated execution of each development iteration, facilitating early identification of errors and improving efficiency in the continuous delivery of new functionality.

The JAMOVIL application will be the central focus of the comparative analysis, where each tool will be evaluated in terms of compatibility with the mobile application, advantages, and disadvantages in regression tests. The results obtained will provide valuable information to make informed decisions about selecting the tool that best suits the project requirements and the characteristics of the mobile application.

Keywords:

Regression testing, Continuous integration, mobile application, analysis

1. Introducción

Las pruebas de regresión se remontan a los primeros días del desarrollo de software. Estas pruebas se han vuelto fundamentales con la constante evolución que los sistemas implementan para adaptarse a los cambios buscando garantizar que la aplicación no introduzca nuevos errores o provoquen la regresión de características previamente funcionales (Gutierrez Zapata, 2021).

En el desarrollo de software tradicional las pruebas se realizaban de forma manual, los equipos de estabilización funcional realizaban pruebas exhaustivas después de cada cambio en el código, lo que consumía tiempo y recursos. A medida del alto crecimiento de las aplicaciones móviles, se volvió evidente la necesidad de encontrar métodos más eficaces para garantizar la calidad del software. (Maida, E. G., & Pacienzia, J., 2015)

Con la automatización de pruebas emergieron herramientas y marcos de trabajo que facilitaron la ejecución de esta labor, esto ha permitido a los equipos de desarrollo ejecutar pruebas repetitivas de manera efectiva y rápida. No obstante, todavía era crucial determinar qué pruebas debían llevarse a cabo tras cada modificación en el código.

La implementación de enfoques ágiles, como Scrum y DevOps, resultó en un incremento significativo en la eficiencia del desarrollo. Los equipos comenzaron a introducir cambios de código generando así una demanda creciente de asegurar la estabilidad y la calidad del software a lo largo del tiempo. (Masache Romero, 2021)

Esta evolución generó una demanda altísima de realización de pruebas de regresión para verificar que las funcionalidades existentes no se vieran afectadas por nuevos cambios en el código.

El término "pruebas de regresión" se popularizó y se convirtió en una parte esencial de los flujos de trabajo de desarrollo.

Este análisis comparativo busca identificar cuál es la herramienta más adecuada para abordar los desafíos específicos de la aplicación móvil financiera **“Ja móvil”** de la COAC Jardín Azuayo, considerando aspectos clave como calidad, eficiencia, cobertura y integración continúa CI/CD.

Se espera que los resultados de este análisis no solo beneficien a la cooperativa en cuestión, sino que contribuyan al conocimiento en el campo de las pruebas de regresión en el ámbito de las aplicaciones móviles financieras, estableciendo mejores prácticas y orientación para futuros desarrollos.

Entre las principales investigaciones y estudios que aportarán a los análisis de las herramientas están "la implementación de una plataforma de automatización de procesos usando “Selenium Web Driver” para optimizar las pruebas de regresión" (Gutierrez Zapata, 2021), también esta (Garg, 2016) que indica el funcionamiento de Appium en pruebas de testeo móvil. Por otro lado, está “el análisis de la funcionalidad de las pruebas de Regresión mediante el Uso de Patrones de Diseño". (Leticia, et al.)

2. Determinación del Problema

2.1 Descripción del Problema

La Cooperativa Jardín Azuayo enfrenta un desafío significativo en el desarrollo y gestión continua de su aplicación móvil "JAMOVIL", donde la percepción del cliente se ha convertido en un factor crítico cuando actualiza a una nueva versión. La presencia de fallas en la aplicación ha generado preocupación entre los usuarios, afectando directamente su experiencia y confianza.

Republicar la aplicación en las tiendas virtuales después cada actualización no es inmediata, problema que se agrava debido a las mejoras continúas implementadas en la aplicación aumentando así la complejidad del proceso de pruebas manuales, consumiendo valiosos recursos y tiempo del equipo de desarrollo. A pesar del crecimiento constante de la aplicación, con la incorporación de nuevas funcionalidades y actualizaciones para satisfacer las cambiantes necesidades del mercado, la insatisfacción del cliente debido a los fallos recurrentes representa un desafío crítico que debe abordarse de manera efectiva.

2.2 Formulación del Problema

¿Cómo validar que las funcionalidades preexistentes en el aplicativo no se vean comprometidas cuando se realizan nuevas incorporaciones o cambios en el aplicativo JAMOVIL evitando la republicación del aplicativo en las tiendas virtuales?

2.3 Justificación del Problema

Desde el punto de visto teórico se realiza esta investigación a pesar de que existen estudios sobre pruebas de regresión, no se han encontrado estudios comparativos entre Selenium y Appium orientada al aplicativo JAMOVIL, con la creciente complejidad y las actualizaciones frecuentes surge la necesidad de garantizar no

solo la eficacia y funcionalidad de las nuevas características, sino también la estabilidad general del sistema. La frecuente ocurrencia de fallos en la aplicación afecta directamente la percepción del cliente.

La calidad y confiabilidad de la aplicación son aspectos fundamentales para la satisfacción del usuario, los fallos recurrentes generan preocupación y desconfianza, poniendo en peligro la relación con los usuarios y la reputación de la Cooperativa. La demora en la publicación de la nueva versión de la aplicación en las tiendas virtuales de Apple, Android ha destacado la oportunidad de investigar soluciones que optimicen el proceso de pruebas. Esto permitiría al equipo de desarrollo abordar de manera más ágil y eficiente los desafíos asociados con la calidad y la entrega continua de la aplicación

2.4 Delimitación del Problema

Este análisis se propone explorar en detalle las características técnicas y la eficacia de Selenium y Appium en el contexto específico de pruebas de regresión en la aplicación "JAMOVIL", también se busca comprender cómo estas herramientas se integran con el enfoque ágil y las metodologías de desarrollo utilizadas por la Cooperativa en la implementación de las aplicaciones móviles. Finalmente, se buscará evaluar la compatibilidad con la integración continua (CI) y la entrega continua (CD), elementos cruciales para mantener la agilidad y la estabilidad en el ciclo de desarrollo.

2.5 Objetivos

2.5.1 Objetivo General

Comparar y Evaluar el proceso de Integración Continua CI/CD al realizar pruebas de regresión en la aplicación móvil financiera "JAMOVIL", utilizando las herramientas de automatización Selenium y Appium, para optimizar el proceso de pruebas mejorando la calidad del software.

2.5.2 Objetivos Específicos

a) Llevar a cabo un análisis completo de la literatura científica y técnica relacionada con pruebas de regresión en aplicaciones móviles financieras, enfocándose en estudios e investigaciones para determinar cuál de las dos herramientas puede brindar resultados de pruebas más rápidos y exhaustivos.

b) Definir el conjunto de requerimientos y criterios de evaluación utilizando herramientas Selenium y Appium en las pruebas de regresión de la aplicación móvil financiera "JAMOVIL" para evaluar y comparar la eficiencia, calidad, cobertura de casos de prueba e integración Continua CI/CD.

c) Definir la automatización de pruebas con Selenium y Appium, asegurando la configuración adecuada en el ambiente de pruebas y dispositivos móviles, para recolectar información que será utilizada en el proceso de análisis y resultados.

d) Medir y comparar las pruebas de regresión obtenida con Selenium y Appium, considerando el número de casos de prueba ejecutados y la profundidad de la cobertura en diferentes escenarios de prueba, para determinar cuál de las dos herramientas ofrece una mayor cobertura de pruebas.

e) Evaluar la calidad de las pruebas de regresión realizadas con Selenium y Appium en la aplicación móvil 'JAMOVIL', mediante la medición de métricas de calidad de código, tasa de defectos detectados y tiempo de ejecución, con el propósito de reconocer las ventajas y desventajas de cada herramienta en relación con su calidad.

3. Marco teórico referencial

A lo largo de los años, hemos sido testigos de un notable progreso en el ámbito del desarrollo de software, con un creciente enfoque en la eficiencia y la calidad. En este contexto, han surgido herramientas específicas para la automatización, entre las cuales destacan Selenium y Appium. Estas herramientas han posibilitado la automatización de una variedad de pruebas, adaptándose a las necesidades específicas de cada proyecto y al tipo de aplicación que se esté desarrollando. (Gundecha, 2015) Además, han emergido metodologías y prácticas para gestionar y ejecutar pruebas de regresión de manera efectiva, como la integración continúa (CI) y la entrega continua (CD).

La expansión constante de las tecnologías móviles ha dado lugar a una creciente dependencia de las aplicaciones móviles en varios aspectos de nuestras vidas (Ted Schadler, 2014), incluyendo las operaciones bancarias y financieras. Por consiguiente, las aplicaciones móviles desempeñan un papel crucial para las instituciones financieras, permitiendo la interacción positiva y segura con los usuarios. Una de estas instituciones es la COAC Jardín Azuayo, cuya aplicación móvil "JAMOVIL" brinda una experiencia de usuario fluida.

A medida que las dinámicas de negocio cambian o se incrementan, las aplicaciones evolucionan y se actualizan para brindar constantes mejoras, en consecuencia, aumenta la complejidad de su desarrollo y mantenimiento. La introducción de cambios en una aplicación móvil puede conllevar a la aparición de errores no deseados, lo que pone en riesgo la confiabilidad y la experiencia del usuario (Murthy, 2019).

Por tanto, para garantizar que la aplicación cumpla con altos estándares de calidad, es necesario contar con un proceso de pruebas sólido que se enfoque en aspectos clave como la calidad del software, evaluación de cumplimiento de requisitos (Ramesh, Srinivasan Desikan y Gopalaswamy, 2016). Además, se debe lograr una amplia cobertura de pruebas para asegurar que todas las partes de la aplicación

estén probadas exhaustivamente. Finalmente, la integración continua (CI) y la entrega continua (CD) son esenciales para asegurar que las pruebas se realicen de manera automática y constante en el ciclo de desarrollo (Farley, 2010).

En este contexto, la automatización de pruebas de regresión ha emergido como un enfoque crucial para mantener la integridad funcional de las aplicaciones móviles en evolución. Así pues, las herramientas de automatización de pruebas, como Selenium y Appium, se han convertido en opciones populares para este propósito, permitiendo a los equipos de desarrollo y pruebas garantizar la estabilidad de la aplicación a lo largo del tiempo y a través de diferentes dispositivos y sistemas operativos.

Las pruebas de regresión en aplicativos móviles desempeñan un papel esencial en el proceso de desarrollo de software para dispositivos móviles (Hans, 2015). En el contexto del ciclo de vida del desarrollo, estas pruebas se sitúan como un componente crítico que busca garantizar la estabilidad y la integridad de una aplicación a lo largo del tiempo.

Cuando se realiza el desarrollo de una aplicación móvil, es común que se realicen modificaciones en el código fuente para agregar nuevas funciones, corregir errores o mejorar el rendimiento. Estos cambios, aunque destinados a mejorar la aplicación, también pueden introducir errores no deseados o afectar inadvertidamente las funcionalidades existentes. Aquí entran en juego las pruebas de regresión.

El propósito fundamental de las pruebas de regresión radica en garantizar la estabilidad y el correcto funcionamiento de las funciones existentes de una aplicación después de realizar cambios o actualizaciones en su código fuente. En esencia, estas pruebas están diseñadas para verificar que las modificaciones introducidas no hayan generado efectos adversos o errores en las funcionalidades previamente establecidas (Ramesh, Srinivasan Desikan y Gopalaswamy, 2016).

Cuando se desarrollan nuevas características o se realizan ajustes en el código de una aplicación, existe el riesgo potencial de que estos cambios impacten

negativamente en otras partes del sistema, generando errores no previstos. Las pruebas de regresión buscan mitigar este riesgo al asegurarse de que las funciones existentes, que antes operaban de manera correcta, sigan haciéndolo incluso después de las modificaciones realizadas.

En términos más simples, estas pruebas actúan como un mecanismo de verificación que busca preservar la integridad y el rendimiento general de la aplicación (Pham, 2003). Su objetivo es identificar cualquier inconveniente o disfunción que pueda surgir como consecuencia de las actualizaciones en el código, permitiendo a los desarrolladores corregir problemas potenciales antes de que afecten la experiencia del usuario o el funcionamiento global de la aplicación.

3.1 Antecedentes Internacionales

3.1.1 Pruebas de Regresión

(Vega Llobell, 2018) Desarrolló una tesis titulada “Pruebas funcionales automatizadas para aplicaciones Web: Usando Selenium para aplicar pruebas de regresión automatizadas” en la Universitat Politècnica de València. Se enfoca en implementar un proceso completo de pruebas automáticas para aplicaciones web en el ciclo de desarrollo empresarial, utilizando Selenium. Incluye el diseño y desarrollo de una herramienta junto con una batería de pruebas para una aplicación específica de la empresa, con el fin de mejorar la eficiencia y calidad del desarrollo de software.

(Orozco Díaz, 2022) Desarrollo una tesis titulada “Implementación de pruebas funcionales automatizadas con Selenium, Appium y Rest-Assured en la empresa Valid Colombia” en la Universidad del Magdalena Colombia. En la cual detalla el proceso desarrollado dentro de la empresa Valid Colombia, indicando como se logró implementar de manera exitosa la automatización de pruebas funcionales haciendo uso de tecnologías como Selenium para la automatización en web, Appium para móviles y Rest-Assured para API's.

(DAVILA NICANOR, et al., 2015) Publicaron una revista de investigación titulada "Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño" en el Instituto Tecnológico de Orizaba. Mencionan que, en el desarrollo de software, la funcionalidad es crucial, pero los cambios en los requisitos pueden resultar costosos. La fase de pruebas busca verificar la funcionalidad del sistema para mejorar el producto. Se propone la automatización de la generación de casos de prueba utilizando la matriz de trazabilidad y patrones de diseño, logrando eficiencia y confiabilidad en la evaluación de la relación entre requisitos y componentes en la arquitectura del software.

(Gojare, S., et al., 2015) Publicó un artículo científico llamado "Analysis and Design of Selenium WebDriver Automation Testing Framework" en el que menciona que el objetivo de las pruebas de software es detectar defectos tempranamente, siendo estas pruebas responsables del 30-60% de los costos del ciclo de vida. Con la prevalencia de aplicaciones web, la calidad de estas se vuelve crucial. Para mejorar la calidad del software en un entorno de desarrollo acelerado, se recurre a las pruebas automatizadas. Este enfoque, utilizando herramientas como Selenium, busca automatizar el diseño y la ejecución de scripts de prueba, reduciendo la intervención humana.

(Morales, et al., 2017) Publico un artículo llamado "Desarrollo de un modelo de pruebas funcionales de software basado en la herramienta SELENIUM" en la Universidad Nacional Mayor de San Marcos Perú. En el cual presentan un modelo de referencia utilizando herramientas existentes para automatizar las pruebas funcionales durante el ciclo de revisión de la calidad del software desarrollado. Indican como la automatización incluye la evaluación de herramientas de gestión de las pruebas y la automatización de pruebas funcionales. Evalúan herramientas y metodologías para la automatización de pruebas, destacando la importancia de soluciones que apoyen la detección temprana de defectos, una cobertura más amplia de funcionalidades durante las pruebas y la ejecución de pruebas eficientes en términos de costo y tiempo.

3.1.2 Integración Continua (CI)

(Herrera Pérez, 2017) Desarrollo una tesis titulada “Análisis e Implementación de la Integración Continua en empresas de software”. En la que realiza un estudio de las tareas para implementar esta técnica, identificando herramientas básicas para la integración continua y los mecanismos de integración entre ellas

(Fradejas Pascual, 2019) Desarrollo una tesis titulada “Sistema de integración continua para el desarrollo del software y hardware en servidores y estaciones de trabajo de empresa”. En el que se destaca la importancia del tiempo de despliegue, la frecuencia de cambios en el código y la práctica de integración continua.

3.1.3 Despliegue Continuo (CD)

(Cusco Mejía, 2022) Tesis titulada “Desarrollo e implementación de una arquitectura DevOps para un sistema web basado en microservicios en infraestructuras basadas en código” destaca que el despliegue continuo minimiza el tiempo de entrega garantizando la calidad y estabilidad del software, acelera procesos solventando problemas tempranos ayudándonos a identificar los cambios para una mejora continua.

3.2 Importancia de las Pruebas de Regresión en Aplicativos

3.2.1 Diversidad de Dispositivos y Plataformas:

La diversidad de dispositivos y plataformas en el entorno móvil presenta un desafío significativo para el desarrollo y prueba de aplicaciones. Las aplicaciones móviles deben funcionar de manera consistente en una amplia variedad de dispositivos, desde teléfonos inteligentes hasta tabletas, y en diferentes sistemas operativos como Android e iOS (Zein, et al., 2016)

3.2.2 Complejidad de las Pruebas:

Cada dispositivo y sistema operativo puede tener peculiaridades únicas. Por lo tanto, las pruebas de regresión deben abordar esta diversidad para garantizar que la aplicación se comporte de manera coherente en todos los escenarios, independientemente del dispositivo o plataforma en el que se ejecute.

3.2.3 Garantía de Compatibilidad:

Las pruebas de regresión son esenciales para verificar la compatibilidad de la aplicación con la amplia gama de dispositivos y sistemas operativos, asegurando una experiencia uniforme para todos los usuarios.

3.2.4 Actualizaciones Continuas:

Las aplicaciones móviles tienden a recibir actualizaciones frecuentes para mantenerse al día con las expectativas del usuario, introducir nuevas características o corregir errores. Esta naturaleza dinámica destaca la importancia de las pruebas de regresión eficientes.

3.2.5 Prevención de Regresiones:

Cada actualización puede introducir cambios en el código, y las pruebas de regresión son cruciales para garantizar que estas modificaciones no afecten negativamente las funcionalidades existentes. De esta manera, se previenen regresiones no deseadas.

3.2.6 Garantía de Calidad Continua:

La realización de pruebas de regresión de manera constante, especialmente después de cada actualización, asegura la calidad continua de la aplicación a lo largo del tiempo. Esto es esencial para mantener la confianza del usuario y la reputación de la aplicación en el mercado.

3.3 Estrategias de Pruebas de Regresión en Aplicativos Móviles

3.3.1 Selección de Casos de Prueba:

La estrategia de selección de casos de prueba en las pruebas de regresión implica identificar y escoger cuidadosamente aquellos casos que son cruciales y representativos para la funcionalidad de la aplicación. El objetivo es optimizar el uso del tiempo y los recursos, centrándose en las áreas críticas de la aplicación que son más propensas a ser afectadas por cambios en el código.

3.3.2 Identificación de Casos Críticos:

Se buscan casos de prueba que abarquen las funcionalidades clave de la aplicación y aquellos escenarios que históricamente han sido propensos a errores o cambios.

3.3.3 Optimización de Recursos:

Al seleccionar casos de prueba de manera estratégica, se maximiza la cobertura de las áreas importantes de la aplicación sin necesidad de probar exhaustivamente cada aspecto, lo que resulta en un uso más eficiente de recursos.

3.3.4 Automatización de Pruebas:

La automatización de pruebas se presenta como una piedra angular en las estrategias de pruebas de regresión para aplicativos móviles. Este enfoque implica el uso de herramientas y scripts automatizados para realizar pruebas de manera sistemática y repetitiva.

3.3.5 Manejo de Diversidad de Dispositivos:

Dada la amplia variedad de dispositivos y sistemas operativos en el entorno móvil, la automatización permite ejecutar pruebas en diferentes configuraciones de manera rápida y consistente.

3.3.6 Eficiencia en Pruebas de Regresión:

La automatización acelera el proceso de pruebas de regresión, permitiendo la

ejecución de pruebas con mayor frecuencia y de manera más rápida que las pruebas manuales. Esto es crucial para mantener la integridad de la aplicación en entornos de desarrollo ágiles.

3.4 Pruebas de Regresión

Las pruebas de regresión ayudan a reducir la probabilidad de introducir errores en versiones anteriores del producto durante el mantenimiento correctivo o adaptativo, así como en las etapas de desarrollo. Este enfoque, según (Rothermel, G., & Harrold, M. J., 1993) contribuye a mantener la calidad del software al identificar posibles problemas que podrían surgir durante modificaciones o actualizaciones del software.

Al ser aplicadas en versiones previas del producto, respaldan la mejora de la confiabilidad en el software producido o mantenido, según lo expuesto por (Morales, et al., 2021). Este método asegura que las alteraciones realizadas no afecten negativamente la estabilidad del producto, fortaleciendo, de esta manera, la entrega de un software más fiable y libre de problemas.

En la literatura, se han explorado varios enfoques de pruebas de regresión. Tanto (Rothermel, et al., 1994) como (Engström, et al., 2010) proponen evaluaciones de técnicas clásicas.

(Pressman, 2010) Indica que cada vez que se incorpora un módulo adicional como parte de pruebas de regresión, el software experimenta modificaciones generando nuevas trayectorias de flujos de información, surgen operaciones de entrada/salidas adicionales e involucran lógica de control recién implementada. Estas alteraciones pueden ocasionar problemas en las funciones que operaban con normalidad, las pruebas de regresión conllevan la ejecución de un conjunto específico de pruebas que ya se han probadas.

En los últimos tiempos, ha crecido el interés en la aplicación de contribuciones provenientes de la inteligencia artificial a los métodos de pruebas de regresión de software. Se han tomado en cuenta algoritmos de aprendizaje máquina, redes neuronales como parte integral de la evolución tecnológica aplicada a este campo, como se discute en estudios de (Kumar, M., Sharma, A., & Kumar, R., 2014) y (Parsa, S., & Khalilian, A. , 2009). Estos reflejan la adaptación de enfoques avanzados con el propósito de mejorar las prácticas de pruebas mediante el aprovechamiento de las capacidades de la inteligencia artificial.

(Kanglin Li, 2006) menciona que las pruebas de regresión aseguran que las modificaciones de código, correcciones de errores y cualquier actividad posterior a la producción no hayan introducido errores adicionales en el código previamente probado. Esta prueba a menudo reutiliza los scripts de prueba creados para las pruebas unitarias e de integración. Las herramientas de prueba de software ofrecen entornos para gestionar estos scripts de prueba y programar la prueba de regresión.

4. MATERIALES Y METODOLOGÍA

4.1 Metodología

Diseño de la Investigación

En la presente tesis se aplica la investigación cuantitativa estableciendo métricas para evaluar y comparar experimentalmente cual es la aplicación más efectiva en pruebas de regresión.

Modalidad de la Investigación

(Babativa Novoa, 2017), menciona lo siguiente con respecto a la investigación cuantitativa:

La investigación cuantitativa nace del deseo de adquirir conocimiento científico y se distingue por explorar la realidad de diversos fenómenos sociales, que pueden ser comprendidos a través del intelecto humano. Los datos obtenidos son el resultado de mediciones. Cuando la información son números (o bien la información recolectada es transformada en escalas numéricas) estamos ante una investigación con datos cuantitativos.

Ejecución de la Metodología Investigación

En la presente tesis se utilizó la metodología de investigación cuantitativa, a través de la definición de variables y métricas necesarias para obtener datos numéricos que sustenten cual es la mejor herramienta para el proceso de pruebas de regresión sobre la aplicación Jamovil.

Se identificaron las variables dependientes en el cual se incluyó la compatibilidad del software de pruebas y el aplicativo móvil, el tiempo empleado de pruebas, la cobertura de código y la detección de errores. Las variables independientes

fueron las herramientas de Selenium WebDriver con su implementación Selendroid y Appium

Basándonos en la recomendación (Esquivel, A. L. E. , 2023), establecimos la relación entre las variables independientes (herramientas de Selenium WebDriver con su implementación Selendroid y Appium) y las variables dependientes (compatibilidad software, tiempo de ejecución de pruebas, cobertura de código y detección de errores).

Se diseño el experimento definiendo el escenario de pruebas en los cuales se creó un conjunto de casos de pruebas representativos de las funcionalidades como la creación del perfil, logueo de la aplicación, consulta estados de cuenta, realización de estados de cuenta, adicional se realizó la instalación y configuración de cada uno de los ambientes.

Durante la instalación y configuración de la herramienta Selenium con Selendroid se detectó errores de compatibilidad con las versiones que necesita el aplicativo Jamovil para funcionar. Por ellos las pruebas con la primera herramienta, Selendroid, fue imposible debido a problemas de compatibilidad con la API. Selendroid está diseñado para ser compatible con versiones de API que van desde la 10 hasta la 19. Sin embargo, la aplicación Jamovil que estamos probando requiere versiones de API superiores a la 29 para ejecutarse. Por lo tanto, debido a esta discrepancia en la compatibilidad de la API, no fue posible llevar a cabo las pruebas utilizando Selendroid.

Con la segunda herramienta, Appium, logramos exitosamente ejecutar los casos de pruebas propuestos. Cuando se logró ejecutar las pruebas de forma manual se configuro la herramienta gitlab runner para ejecutar de manera automática la ejecución de los scripts luego de cada cambio que se dé al código.

Dado que enfrentamos problemas de incompatibilidad con Selendroid, no se pudo realizar las pruebas con esta herramienta. Sin embargo, logramos ejecutar exitosamente las pruebas utilizando Appium. Con los resultados recolectados de las pruebas realizadas con Appium, generamos un registro detallado que incluyó

información sobre el tiempo de ejecución, los casos de éxito y los fallos, así como una evaluación de la cobertura del código y una descripción detallada de los errores encontrados.

Se identificaron tendencias específicas que revelaron patrones en los datos, señalando las ventajas y desventajas de Appium. También se llevó a cabo un estudio exhaustivo de la naturaleza y frecuencia de los errores detectados utilizando esta herramienta. Finalmente, se resumieron los hallazgos cuantitativos en relación con las métricas establecidas, brindando recomendaciones fundamentadas en datos numéricos para la elección de la herramienta más idónea para las pruebas de regresión en la aplicación Jamovil.

4.2 Herramientas de Automatización de Pruebas en Aplicativos Móviles: Appium y Selenium con Selendroid

La calidad en un proyecto de software se ve reflejada en la capacidad del sistema para satisfacer las necesidades del usuario final. Un proyecto con una baja calidad trae múltiples consecuencias, desde la perdida de dinero, tiempo, reputación, clientes entre otros. Con el fin de evitar esto las pruebas de software (tanto automatizadas como manuales) buscan asegurar la calidad del producto y disminuir el riesgo de tener errores cuando este se encuentra en operación como lo menciona (Mena Barrera, 2021).

Las herramientas de automatización ejercen un papel crucial en el proceso de pruebas de aplicativos móviles, permitiendo a los desarrolladores y equipos de calidad garantizar el rendimiento, la funcionalidad y la compatibilidad en una diversidad de dispositivos y plataformas. Dos de las herramientas más destacadas en este ámbito son Appium y Selenium WebDriver con Selendroid.

Selenium

Definición

Selenium es una herramienta que facilita la creación rápida de scripts de pruebas a través de una funcionalidad de grabación que registra todas las acciones hechas en la interfaz de usuario, generando automáticamente comandos de script. Esta herramienta opera como un complemento de Firefox, permitiendo la grabación y reproducción sencilla de interacciones con el navegador (Selenium Developers Group, 2016).

WebDriver

Herramienta que forma parte de una funcionalidad de Selenium, brinda una potente interfaz que permite interactuar de forma automática en un navegador Web, actúa como puente entre el código de prueba y el navegador, mediante este código que genera el desarrollador se generan acciones específicas simulando interacciones humanas (García Gutiérrez, et al., 2021).

Se puede simular un clic, completar formularios, verificar dinámicamente el contenido de una página gracias a que se puede generar un script claro y conciso, esta capacidad de automatización no solo ahorra tiempo, sino que también mejora la consistencia de las pruebas, reduciendo los errores humanos asociados con las pruebas manuales repetitivas (Pérez, et al., 2022).

Podemos destacar que la característica que tiene esta herramienta es el soporte a la gran variedad de navegadores Web, su propósito es ofrecer una API orientada a objetos, bien estructurada, que ofrece un soporte mejorado para los problemas avanzados actuales en las pruebas de aplicaciones web (Rivera Martínez, C. A., 2018).

La arquitectura de Selenium WebDriver consiste en un cliente de Selenium, un protocolo de comunicación JSON Wire, un servidor de Selenium, y una API de WebDriver. El cliente es el que se encarga de escribir y ejecutar los scripts de prueba, mientras que el servidor gestiona la comunicación entre el cliente y los

navegadores web. El protocolo define cómo se transmiten los mensajes entre el cliente y el servidor, y la API proporciona métodos para interactuar con los navegadores. En conjunto, esta arquitectura permite la automatización de pruebas en diversos navegadores y plataformas de manera eficiente y flexible (Pérez, et al., 2022).

A continuación, el diagrama de la implementación de Selenium WebDriver:

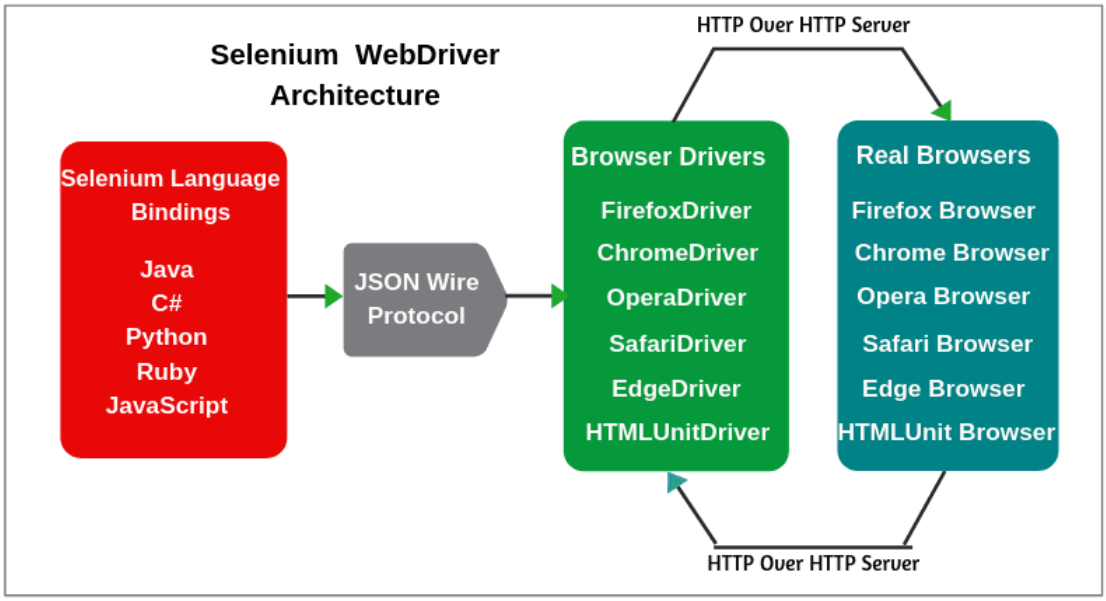


Figura 1. Arquitectura de Selenium WebDriver (Easy, 2020)

SELENDROID

Selendroid es una herramienta de automatización de pruebas para aplicaciones nativas e híbridas de Android, es un complemento de Selenium WebDriver. Elimina la interfaz de usuario y utiliza la API del cliente Selenium 2 para escribir y ejecutar pruebas. (Selendroid Selenium for android, s.f)

Selenium utiliza el complemento Selendroid para realizar pruebas en un dispositivo móvil que es otro framework para automatizar pruebas en aplicaciones Android (Mena Barrera, M. F. , 2021)

Selendroid se basa en el marco de instrumentación de Android y se limita a probar una aplicación a la vez. Tiene cuatro componentes principales:

Selendroid-Client (una biblioteca del cliente Java basada en Selenium), Selendroid-Server (que se ejecuta junto a la aplicación en el dispositivo Android), AndroidDriver-App (una aplicación de vista web del controlador Android integrada para pruebas de web móvil) y Selendroid-Standalone (que administra diferentes dispositivos Android instalando el servidor Selendroid y la aplicación bajo prueba). (Selendroid Selenium for android, s.f)

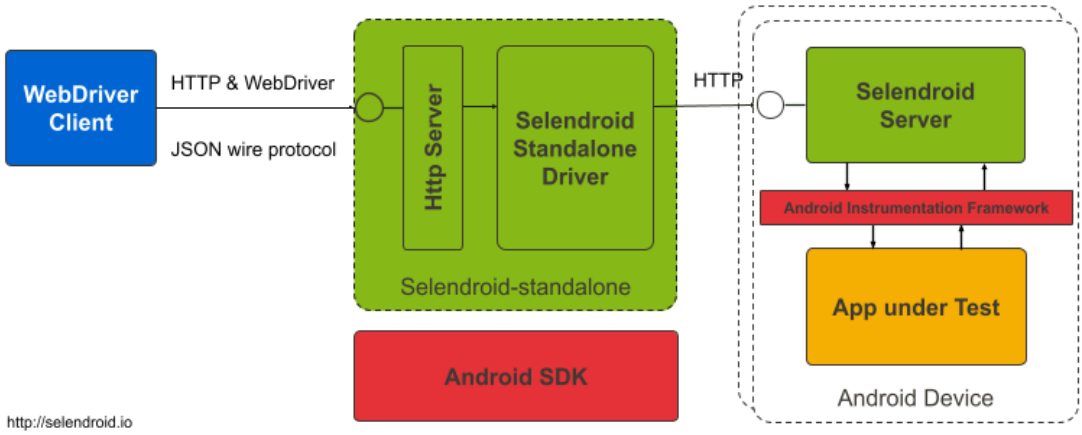


Figura 2. Arquitectura de Selendroid (Selendroid Selenium for android, s.f)

Ventajas

- Es software libre.
- Permite realizar pruebas en aplicaciones Web y Móviles (Selendroid)
- Flexibilidad para adaptarse a varios lenguajes de programación
- Compatibilidad con Navegadores y Android
- Integración con herramientas de informes como TestNG y JUnit
- Automatización de Flujos Críticos
- Integración con Herramientas de Integración Continua
- Compatibilidad total con JSON Wire Protocol /Selenium 3 Ready.
- No se requiere ninguna modificación de la aplicación bajo prueba para automatizarla.
- Prueba de la web móvil utilizando la aplicación webview del controlador Android integrada
- Mismo concepto para automatizar aplicaciones nativas o híbridas

Desventajas

- Limitaciones de compatibilidad con versiones más recientes de Android.
- El proyecto Selendroid no está siendo mantenido o actualizado como otras herramientas de automatización de pruebas para Android.
- Complejidad de configuración requiere un mayor tiempo y esfuerzo en comparación con otras herramientas de automatización.

Appium

Definición

Herramienta de automatización de pruebas multiplataforma de código abierto para aplicaciones web, escritorio, hibridas y móviles que permite crear pruebas en diversas plataformas utilizando la misma API lo que permite la reutilización de código entre varios casos de prueba definidos en cualquier plataforma(Paños Medina, 2021), de tal manera que es considerado un ecosistema de software relacionado, diseñado para facilitar la automatización de la interfaz de usuario multiplataforma (Appium.io, 2024).

Funcionalidad

(Appium.io, 2014) Se utiliza principalmente en el área de pruebas de automatización de software para garantizar que la aplicación funcione

correctamente tanto en un ambiente de pruebas como en un productivo manteniendo su velocidad, escalabilidad y funcionalidad. Appium expone una API REST estandarizada que permite la conexión del cliente en múltiples plataformas.

Esta estructura abre un mundo de posibilidades ya que permite escribir las pruebas en cualquier lenguaje que tenga una API cliente HTTP, aunque resulta más fácil utilizar una de las librerías cliente de Appium (Paños Medina, 2021).

Arquitectura

Appium es un servidor web basado en Node.js que recibe las conexiones de sus clientes mediante objetos JSON en formato HTTP, inicia una sesión para escuchar los comandos emitidos para luego ejecutarlos y devolver el estado de su sesión mediante un ID (Verma, 2017).

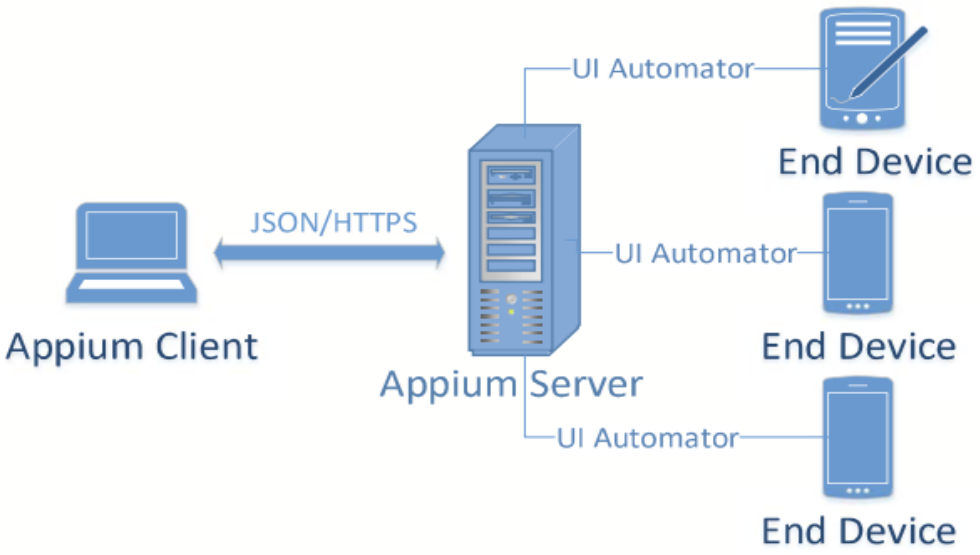


Figura 3. Arquitectura Appium (Tran, 2023)

Cliente Appium

El cliente es un código script que se escribe en cualquier lenguaje de programación como Java, Python, etc. Este script contiene los detalles de configuración de los dispositivos finales y las aplicaciones. Aquí también se escribe el código para ejecutar los casos de prueba de las aplicaciones. (Tran, 2023)

Servidor Appium

El servidor de Appium recibe solicitudes del cliente de Appium en formato JSON y ejecuta esos comandos en los dispositivos de prueba. Crea sesiones para interactuar con los dispositivos y luego reenvía las solicitudes del cliente a los dispositivos. (Tran, 2023)

Ui Automator

Proporciona un conjunto de API para construir interfaces de usuario. Estas interfaces permiten a los usuarios realizar operaciones como abrir el menú de configuración o el iniciador de aplicaciones en los dispositivos finales. (Tran, 2023)

Flujo de pruebas

La investigación de (Tran, 2023) indica los pasos que se deben seguir para realizar las pruebas son:

- Prepara los Scripts y casos de prueba que se van a realizar.
- Los Scripts se convierten en formatos JSON para luego ser enviados al Servidor Appium.
- El servidor Appium interpreta los comandos enviados por el cliente para luego ser transmitidos a un emulador o dispositivo físico para activar la ejecución de los casos de prueba.

- Los dispositivos finales responden al servidor mediante Http para que verifique las respuestas y pueda definir si están correctas o tienen algún fallo.

EL servidor genera el resultado de la prueba proporcionando el registro de todas las acciones realizadas.

Ventajas

- Software Libre
- Posee un API creado por un solo código homologado para cualquier plataforma.
- Fácil acceso a la API desde cualquier lenguaje de programación.
- Soporta tanto emuladores como dispositivos físicos, códigos reutilizables.
- Mejora constante en la documentación oficial.
- Proporciona herramientas para permitir el desarrollo comunitario.

Desventajas

- No existe una empresa que brinde soporte directo. No es compatible para Android inferior a 17.
- No se puede ejecutar pruebas en varios dispositivos de forma simultánea.
- Si durante las pruebas se necesita instalar o desinstalar aplicaciones se necesita librerías proporcionadas por Appium.

5. Resultados y discusión

A continuación, se llevará a cabo la comparación de las dos herramientas de automatización de pruebas propuestas.

5.1 Tabla comparativa entre Selenium WebDriver con Selendroid y Appium

CRITERIOS	**SELENIUM WEBDRIVER CON SELENDROID**	**APPIUM**
Plataforma	Soporta Aplicaciones Android.	Soporta multiplataforma (Android e iOS)
Pruebas de Aplicaciones Nativas e Híbridas	Admite aplicaciones web móviles, híbridas y nativas	Admite aplicaciones web móviles, híbridas y nativas
Lenguaje de Automatización	Soporta Java	Soporta Multilenguaje de programación (Java, Python, Ruby, C#, etc.)
Ejecución de Pruebas	Ejecuta las pruebas en Dispositivos reales con Android	Ejecuta las pruebas en emuladores y hardware real (IOS, Android)
Actualización Continua	Desarrollo ralentizado, limita su compatibilidad con las nuevas funciones y dispositivos Android.	Actualización periódicamente, garantiza la compatibilidad con las últimas plataformas y tecnologías móviles
Ecosistema y Comunidad	Base de usuarios y una comunidad limitada lo que genera menos recursos disponibles cuando se busca apoyo de la comunidad.	Cuenta con una comunidad de usuarios grande y activa, que ofrece una amplia cantidad de recursos, tutoriales y soporte

Tabla 1. Comparación entre Selendroid y Appium

5.2 Casos de prueba

Para realizar la comparación se detalla los casos de prueba que van a ser validados con las herramientas.

5.2.1 Registro de Usuario

REGISTRO DE USUARIO	CP001	
	¿Prueba de despliegue?	SI
Descripción: Registro de usuario en el aplicativo móvil para tener acceso al sistema		
Prerrequisitos • Debe ser socio activo. • Debe tener activo el servicio en la Web transaccional. • Debe tener instalado el aplicativo Jamovil		
Pasos: • Usuario y Clave: Para el registro deberá heredar el perfil Web transaccional • Captcha: Presenta de forma aleatoria código • Ingreso Captcha: Valida que corresponda al presentado por el sistema • Envío OTP: Si continúa enviará OTP por correo y SMS del usuario • Nombre de Dispositivo: Se deberá presentar en una nueva pantalla un campo que permita asignar un nombre al Dispositivo de hasta 10 caracteres. • Configuración PIN: (Nombre visual CLAVE) de cuatro dígitos con el que ingresará posteriormente • Ingreso de PIN: Permitirá el ingreso de Cuatro dígitos • Confirmación de PIN: Permitirá el ingreso de cuatro dígitos		

- Términos y Condiciones: Deberá presentar en pantalla: nombres y apellidos del socio, número de teléfono, correo electrónico y campo de aceptación de términos y condiciones.

Resultado esperado:

Crear el registro del usuario en la aplicación JAMOVIL que le permita tener al socio una clave por la cual pueda hacer el proceso de inicio de sesión.

Caso de Uso

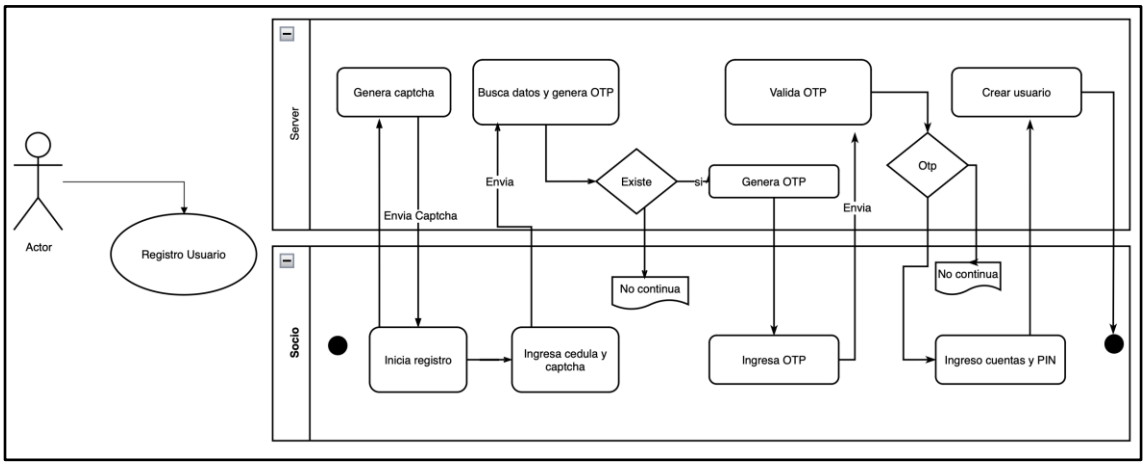


Figura 4. Caso de uso de registro de usuario

5.2.2 Consultar estados de cuenta

CONSULTAR ESTADOS DE CUENTA	CP002	
	¿Prueba de despliegue?	SI

Descripción:
Consultar los estados de cuenta de ahorros en un determinado rango de fechas

Prerrequisitos

- Tener registrado el usuario en el sistema.
- Tener el aplicativo Jamovil.
- Debe realizar el proceso de login.

Pasos:

- Deberá poder consultar las 5 últimas transacciones de un combo de cuentas de ahorro activas de la base de datos de producción
- Al momento de selección de cuenta deberá aparecer el saldo Total-Bloqueado y Disponible de la cuenta seleccionada
- Contendrá los siguientes datos: Fecha, Transacción, Total, Detalle (Transacción, valor, fecha y hora y campo observación)

Resultado esperado:

Crear el registro del usuario en la aplicación JAMOVIL que le permita tener al socio una clave por la cual pueda hacer el proceso de inicio de sesión.

Casos de Uso

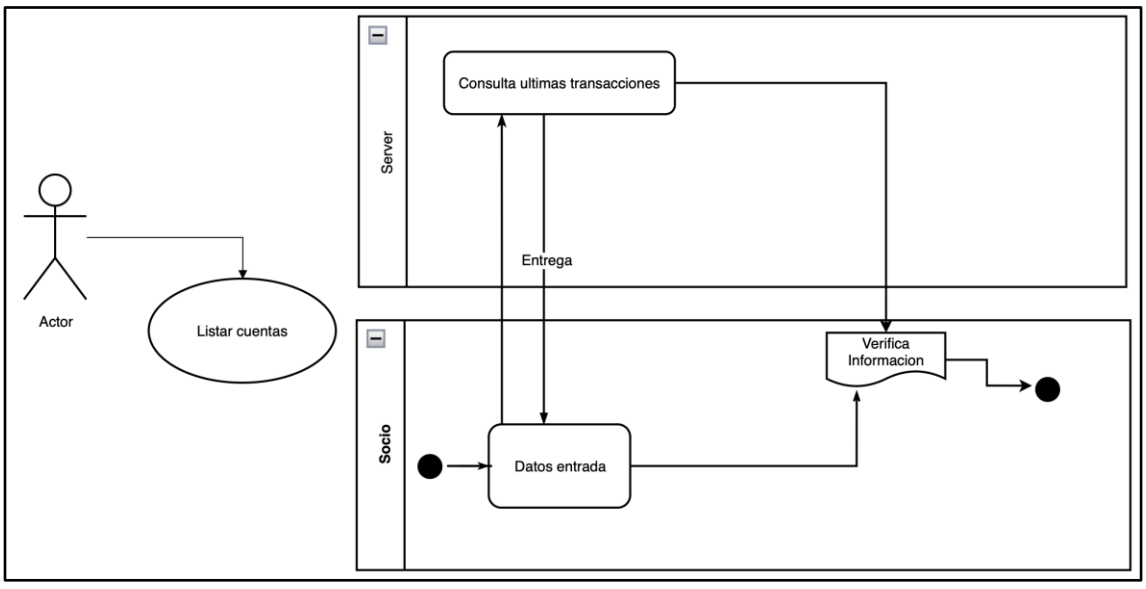


Figura 5. Caso de uso consulta de estado de cuenta

5.2.3 Transferencia entre cuentas internas

TRANFERENCIA CUENTAS INTERNAS	CP003	
	¿Prueba de despliegue?	SI

Descripción:
Realizar la transferencia entre 2 cuentas de la cooperativa, debemos escoger la cuenta origen y destino, colocar el valor a transferir seguido de un código Otp que confirme la transacción.

Prerrequisitos

- Tener registrado el usuario en el sistema.
- Tener el aplicativo Jamovil.
- Debe realizar el proceso de login.
- Tener registrado una cuenta origen con salgo > 0.
- Tener registrar la cuenta destino, debe ser una cuenta activa.

Pasos:

- Ingresar a la aplicación
- Opción transferencias
- Escoger la cuenta origen
- Escoger la cuenta destino
- Colocar el valor
- Clic en transferir
- Colocar el Otp de confirmación
- Transferir
- Aceptar

Resultado esperado: Se ejecute el proceso de forma satisfactoria, podemos revisar nuestro estado de cuenta

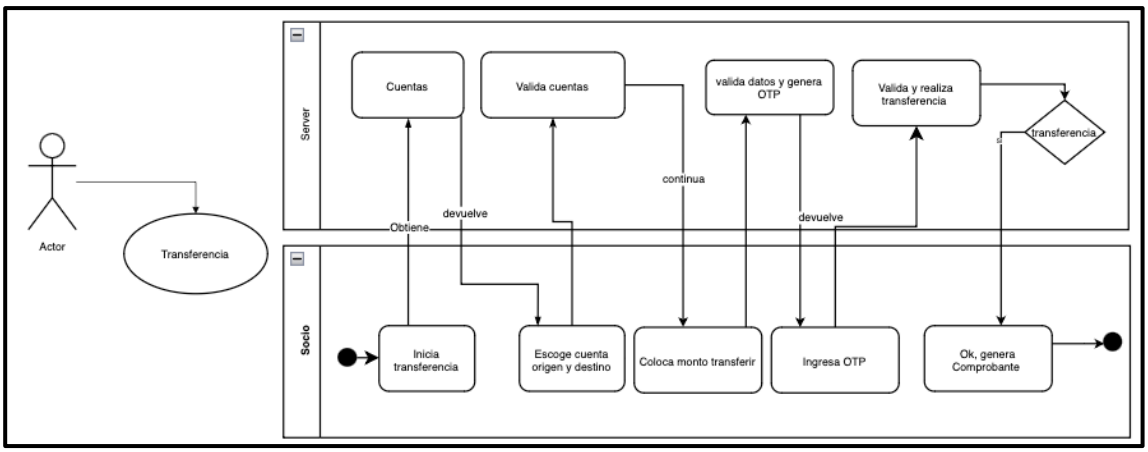


Figura 6. Caso de uso transferencia entre cuentas

5.2.4 Bloquear Libreta

CONSULTAR ESTADOS DE CUENTA	CP004	
	¿Prueba de despliegue?	SI
Descripción: Tener la opción para en caso de pérdida de libreta poder bloquear la cuenta		
Prerrequisitos • Tener registrado el usuario en el sistema. • Tener el aplicativo Jamovil. • Debe realizar el proceso de login.		
Pasos: • Cuentas de Ahorro: deberá tener la opción de bloquear los saldos de las cuentas de ahorro activas (no selecciona la cuenta, bloquea todas). • Deberá bloquearse los saldos de todas las cuentas de ahorro que posea el socio y verificarse en el detalle de Bloqueo de la Cuenta.		

- Se deberá bloquear todas las cuentas que se encuentren en estado activa, en caso de que ya se encuentren bloqueadas deberá mostrar mensaje: “Estimado Socio, sus cuentas ya han sido bloqueadas anteriormente”.

Resultado esperado: Luego del proceso de bloqueo no se puede realizar ninguna transacción de debito de las cuentas del socio.

Casos de Uso

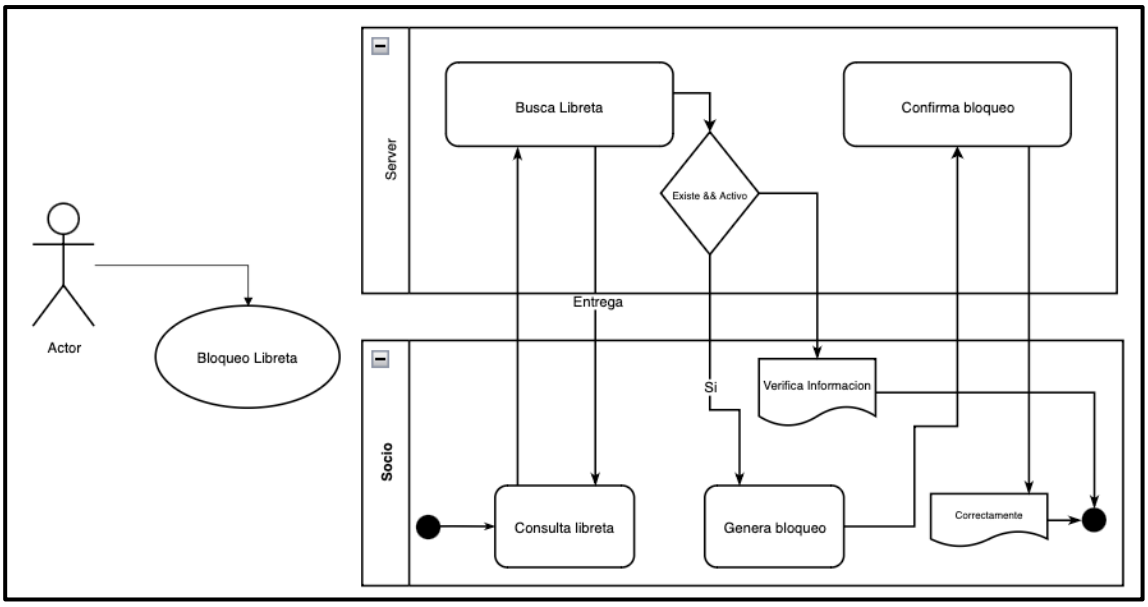


Figura 7. Caso de uso bloquear libreta

5.3 Configuración de entornos de pruebas

Previo a discutir los resultados particulares de las pruebas de regresión, es esencial contextualizar el entorno de pruebas. Se crea un ambiente controlado donde se configura cada una de las herramientas y scripts automatizados para Selenium y Appium, emulando las acciones que el usuario debe realizar en la aplicación Jamovil para pasar los casos de prueba establecidos en el apartado de casos de uso.

5.4 INSTALACIÓN Y CONFIGURACIÓN DE SELENIUM WEBDRIVER CON IMPLEMENTACIÓN SELENDROID

Para utilizar Selendroid es necesario contar con una versión de Java SDK (mínimo 1.6) y configurar JAVA_HOME.

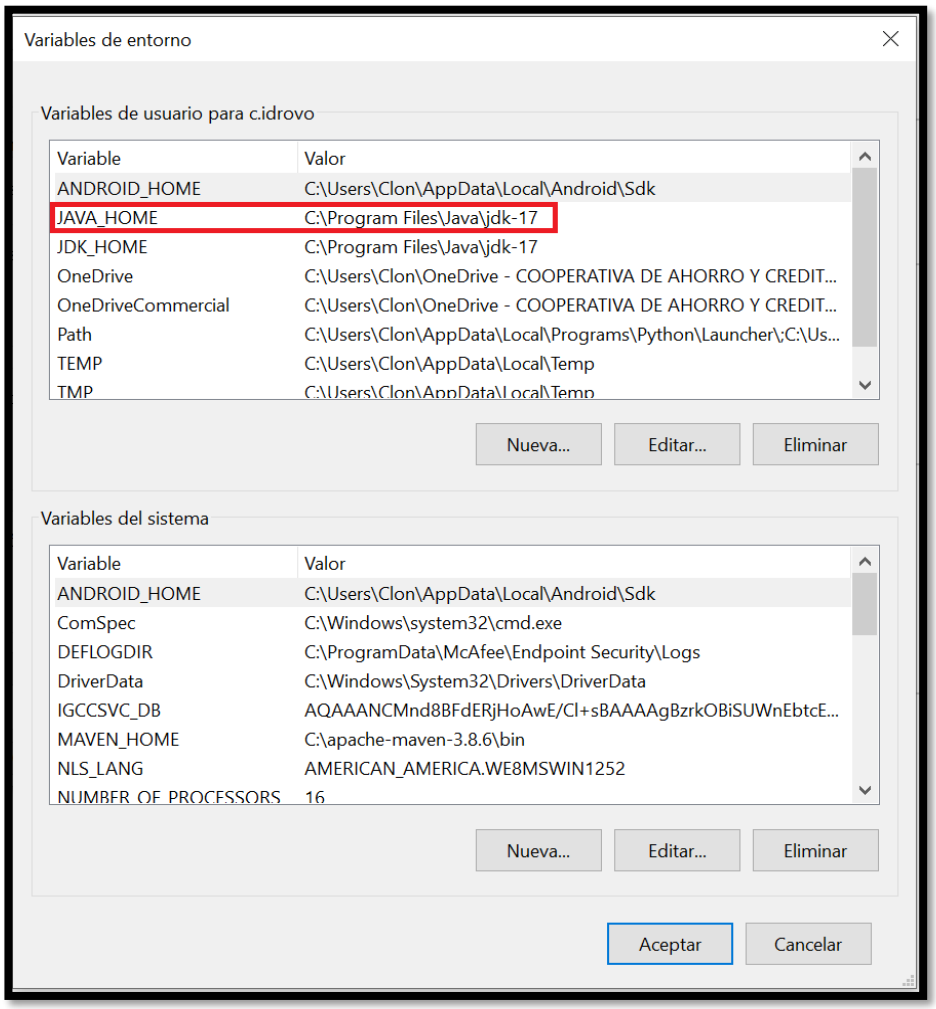


Figura 8. Configuración de JAVA_HOME en las variables de entorno

Instalar el último SDK de Android y configurar *ANDROID_HOME*.

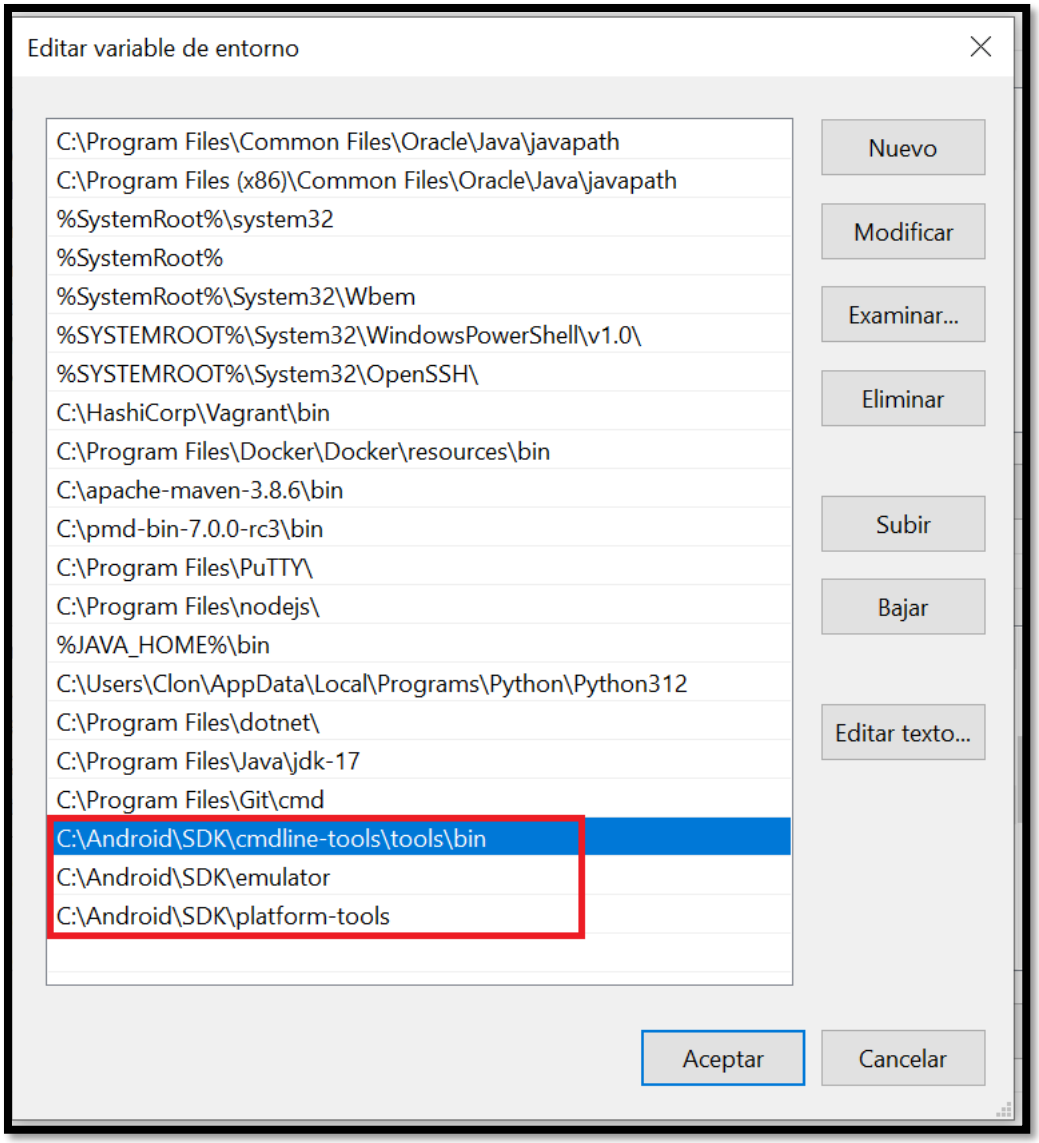


Figura 9. Configuración de SDK en las variables de entorno

Verificar el correcto funcionamiento del sdkmanager

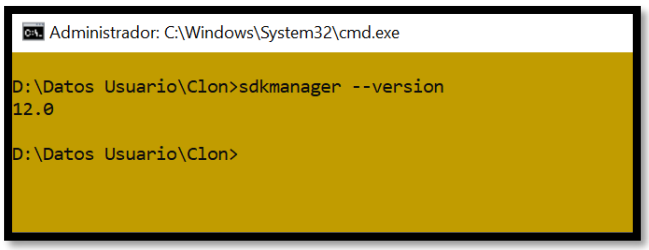


Figura 10. Validación de sdkmanager desde la consola de cmd

Descargar un apk de prueba para probar el ambiente.

Figura 11. Archivo .apk de prueba para validar Selendroid

Descargar .jar que sirve como lanzador del servicio de Selendroid

Figura. 12. Archivo .jar que lanza el servidor de Selendroid

Ingresar comandos para levantar el servidor de Selendroid

Figura 13. Comandos para ejecutar test de Selendroid

Por problemas de versiones, el sdkmanager que actualmente está disponible no soporta las versiones obsoletas de Selendroid, por lo que da el siguiente error.

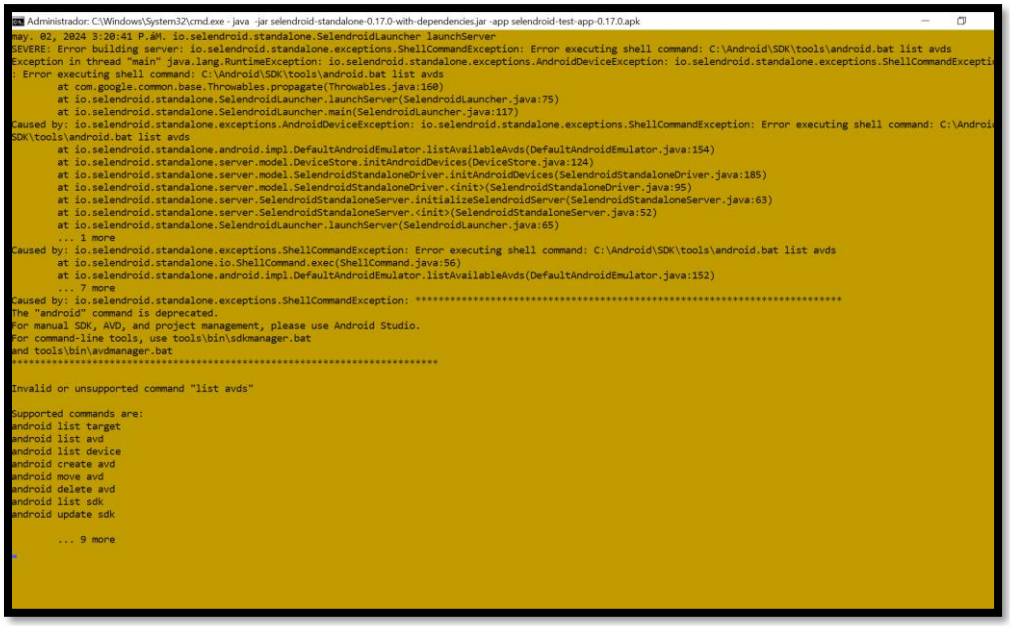


Figura 14. Log de error de compatibilidad de Selendroid

Comparación de herramientas necesarias para compilar Selendroid y la aplicación Jamovil.

CRITERIOS	**SELENDROID**	**JAMOVIL**
Sdkmanager	Android 4.4 (Api 19)	Android 14.0 (Api 34)

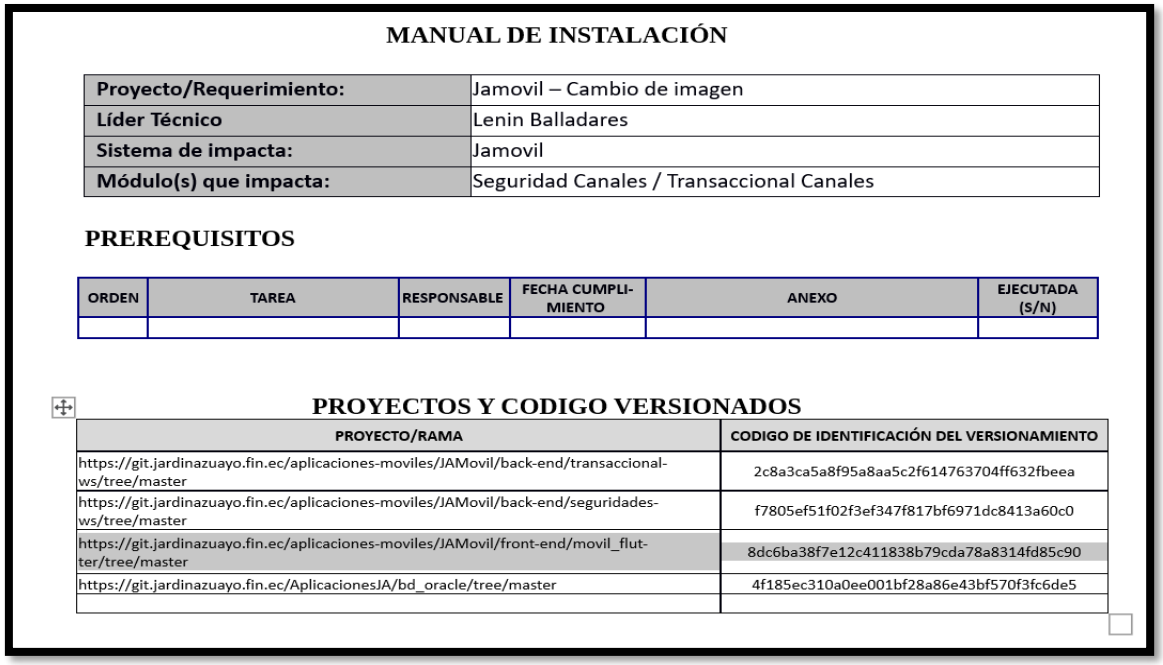
Tabla 2. Comparación entre SDK mánager Selendroid y Jamovil

5.5 Resultados de las pruebas de regresión con Selendroid

Durante el inicio del proceso de ejecución de pruebas de regresión utilizando Selenium WebDriver con Selendroid en la aplicación Jamovil, nos encontramos con desafíos significativos derivados de problemas que surgieron debido a discrepancias entre las versiones de la aplicación Jamovil y el servidor de Selendroid, el cual generó errores durante el proceso de instalación y levantamiento del servidor de Selendroid. A pesar de los esfuerzos para resolver estos problemas, incluyendo la búsqueda de actualizaciones de las versiones de Selendroid y la revisión de la configuración de las pruebas, no se logró una solución para llevar a cabo el análisis planteado dentro del alcance de este estudio.

La causa de la incompatibilidad del aplicativo con Selenium se da por varios factores entre los que están:

La aplicación Jamovil que pertenece a la cooperativa Jardín Azuayo sufrió una actualización por temas de normativas internas que exige la SEPS (Superintendencia de economía popular y Solidaria), misma que obligo a colocar nuevas funcionalidades que requerían de componentes actuales para su optimo funcionamiento.



MANUAL DE INSTALACIÓN

Proyecto/Requerimiento:	Jamovil – Cambio de imagen
Líder Técnico	Lenin Balladares
Sistema de impacta:	Jamovil
Módulo(s) que impacta:	Seguridad Canales / Transaccional Canales

PREREQUISITOS

ORDEN	TAREA	RESPONSABLE	FECHA CUMPLI-MIENTO	ANEXO	EJECUTADA (S/N)

PROYECTOS Y CODIGO VERSIONADOS

PROYECTO/RAMA	CODIGO DE IDENTIFICACIÓN DEL VERSIONAMIENTO
https://git.jardinazuayo.fin.ec/aplicaciones-moviles/JAMovil/back-end/transaccional-ws/tree/master	2c8a3ca5a8f95a8aa5c2f614763704ff632fbeea
https://git.jardinazuayo.fin.ec/aplicaciones-moviles/JAMovil/back-end/seguridades-ws/tree/master	f7805ef51f02f3ef347f817bf6971dc8413a60c0
https://git.jardinazuayo.fin.ec/aplicaciones-moviles/JAMovil/front-end/movil_flutter/tree/master	8dc6ba38f7e12c411838b79cda78a8314fd85c90
https://git.jardinazuayo.fin.ec/AplicacionesJA/bd_oracle/tree/master	4f185ec310a0ee001bf28a86e43bf570f3fc6de5

Figura 15. Manual de pase a producción Aplicación Jamovil.

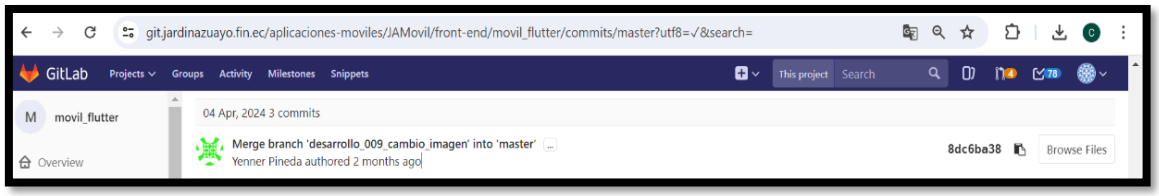


Figura 16. Commit realizado en el repositorio Git

Otra de las causas fue el cambio de imagen de la aplicación que genero una actualización en las versiones del sdkmanager requerido.

Figura 17. Cambio de imágenes en el repositorio GIT

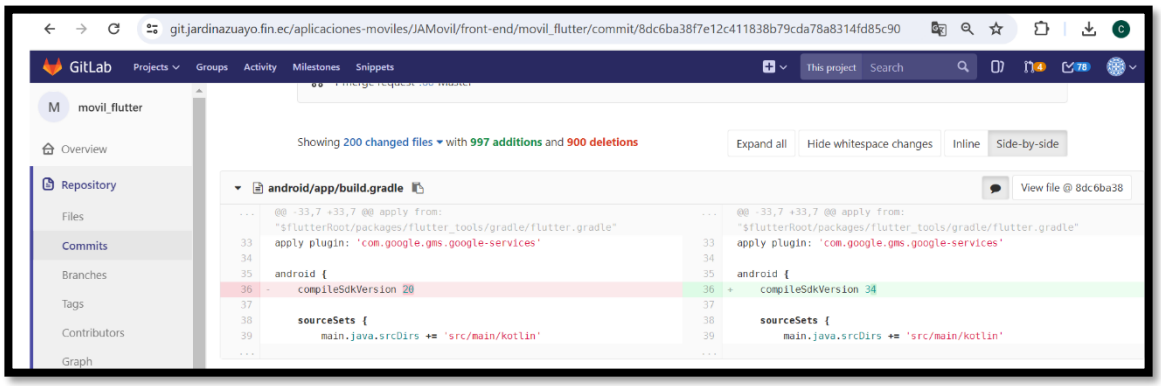


Figura 18. Actualización del SDK versión para compilar la aplicación.

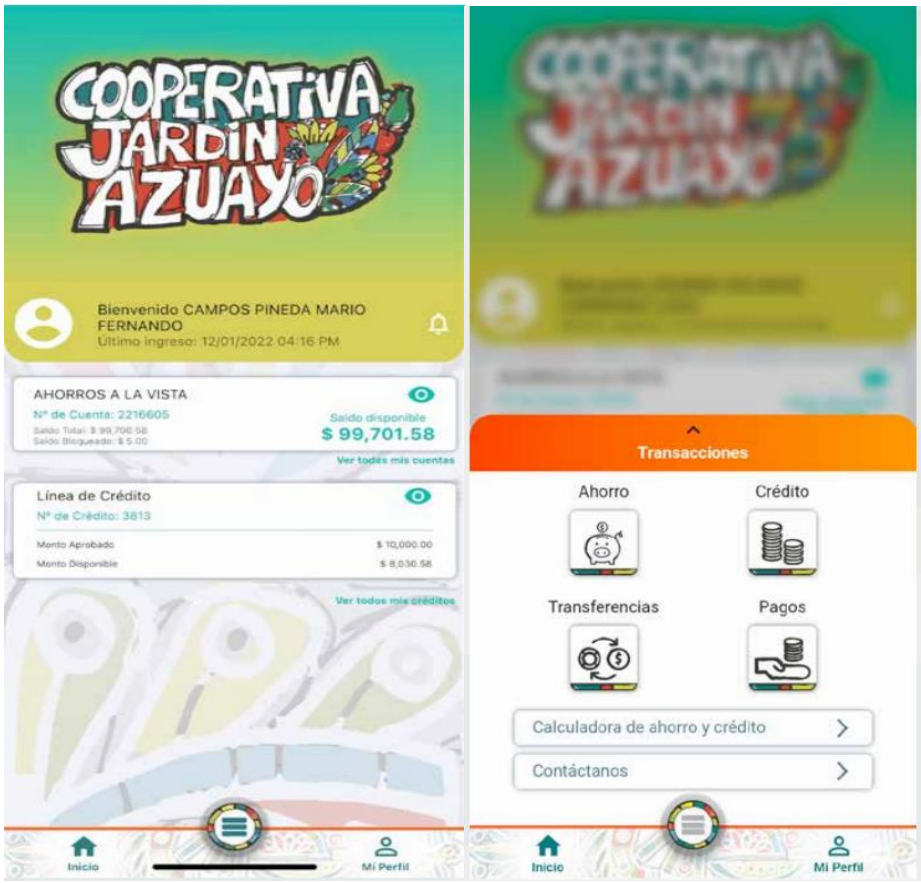


Figura 19. Home anterior de la aplicación Jamovil

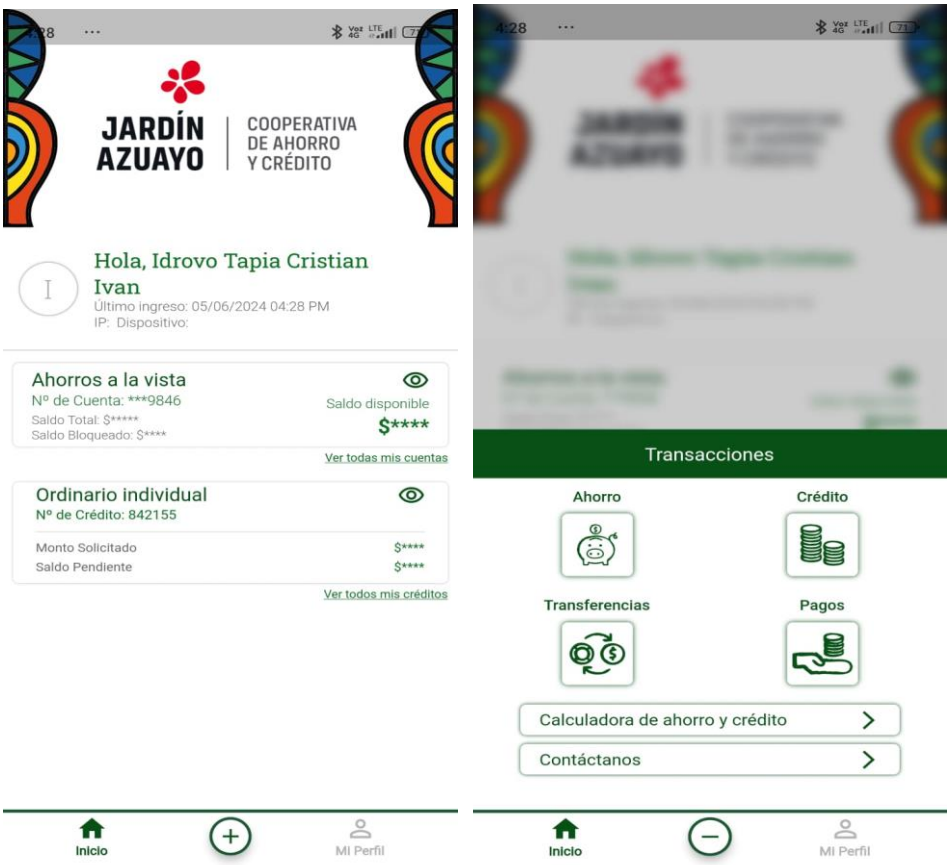


Figura 20. Home nuevo de la aplicación Jamovil

La imposibilidad de completar las pruebas con Selenium debido a problemas de compatibilidad subraya la importancia crucial de considerar la interoperabilidad entre las herramientas de automatización, el soporte continuo y las aplicaciones a probar antes de iniciar un proyecto de automatización de pruebas.

5.6 Instalación y configuración de Appium

La configuración se la realizo en un equipo Mac, la instalación se realizó mediante el gestor de paquetes NPM 9.7.2, para ejecutar de manera correcta la instalación es necesario tener configurado la variable de entorno Java mínimo el jdk 1.8 y Android.

Versiones:

- Node.js ^14.17.0
- NPM >= 8 (9.7.2)
- Java >= 8

Se realiza la configuración de la variable del entorno

```
s.corderoc@desarrollo pagos % vim ~/.zshrc
```

Figura 21. Comandos para realizar la configuración de variables de entorno en MacOS

Una vez abierto el archivo se procede a configurar cada una de las variables de entorno necesarias.

```
export ANDROID_HOME=~/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/platform-tools
export ANDROID_SDK_ROOT=~/Library/Android/sdk
export ANDROID_AVD_HOME=~/.android/avd
export PATH=${PATH}:$M2_HOME/bin:$$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
export GEM_HOME=$HOME/.gem
export PATH=$GEM_HOME/bin:$PATH
export PATH="$HOME/.jenv/bin:$PATH":/Users/s.corderoc/development/flutter/bin
```

Figura 21. Configuración de variables de entorno en MacOS

Para realizar el proceso e instalación de Appium es recomendable seguir la guía del sitio oficial del aplicativo: https://appium.io/docs/en/2.0/quickstart/install/

Una vez finalizada la instalación se procede con la configuración del ambiente para realizar las pruebas de regresión con esta herramienta.

Para efecto de las pruebas lo mantenemos el host y el puerto definido por defecto, luego iniciamos el servidor Appium pulsando el botón **starServer**

Figura 22. Pantalla inicial de arranque de Appium

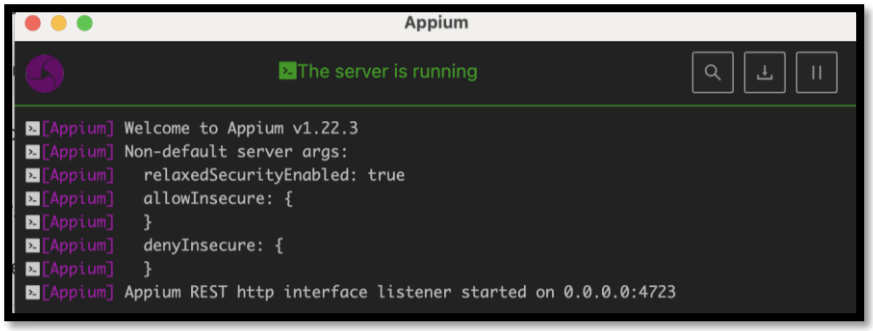


Figura 23. Pantalla que muestra el estado del servidor de Appium, la ip y puerto por el cual está escuchando.

Es necesario instalar la herramienta Appium inspector, la cual se va a usar para identificar los componentes del aplicativo al que vamos a realizar las pruebas.

En este apartado se define los Capabilities, que son un conjunto de claves y valores utilizados para especificar diversas propiedades y configuraciones, tales como la plataforma a utilizar, el dispositivo o emulador, el nombre de la aplicación y la versión del sistema operativo.

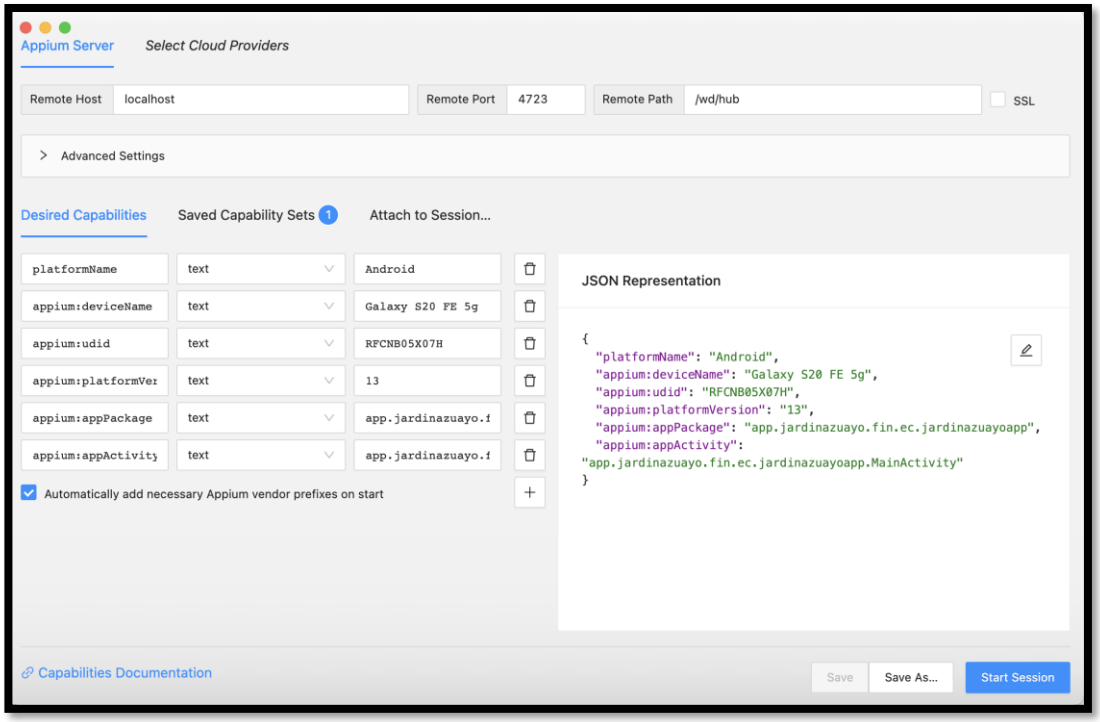


Figura 24. Pantalla que configura los Capabilities necesarios para que funcione Appium.

Las librerías utilizadas en java para definir los Capabilities son las siguiente:

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.remote.DesiredCapabilities;
```

Figura 25. Listado de importaciones de las librerías para ejecutar las pruebas.

Definimos los Capabilities que se van a usar en la prueba.

```
AndroidDriver<AndroidElement> driver;
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("appium:deviceName", "Galaxy S20 FE 5g");
capabilities.setCapability("appium:udid", "RFCNB05X07H");
capabilities.setCapability("appium:platformVersion", "13");
capabilities.setCapability("appium:appPackage", "app.jardinazuayo.fin.ec.jardinazuayoapp");
capabilities.setCapability("appium:appActivity", "app.jardinazuayo.fin.ec.jardinazuayoapp.MainActivity");
capabilities.setCapability("appium:ensureWebviewsHavePages", "true");
capabilities.setCapability("appium:nativeWebScreenshot", "true");
capabilities.setCapability("appium:connectHardwareKeyboard", true);
driver = new AndroidDriver( new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
```

Figura 26. Asignación de valores a cada Capabilities.

Para la obtención de los componentes que se necesitan colocar en el código es necesario utilizar la aplicación Appium Inspector, esta herramienta nos permite navegar en la aplicación e ir obteniendo los nombres de las pantallas, botones, imágenes, cuadros de texto, etc.

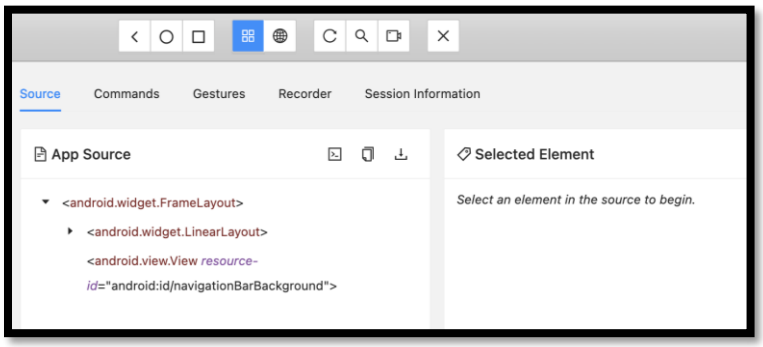


Figura 27. Sección de Appium inspector en el que se listan cada componente que interviene medida que se va realizan las acciones en la aplicación móvil.

5.7 Resultados de las pruebas usuario final (tester)

Para tener una idea del tiempo que emplea un usuario tester en realizar cada caso de uso de uno de los 4 planteados se realizó una medición para obtener la referencia del tiempo.

Herramienta Automatización	Caso Prueba	Numero de repeticiones	Tiempo Ejecución	Porcentaje Caso de prueba alcanzado	Observaciones
Tester	CP001	1	00:05:10	100%	Se ejecuta el registro del usuario
Tester	CP002	1	00:04:12	100%	Revisar el estado de cuenta del usuario, se realiza desde el registro de usuario
Tester	CP003	1	00:04:20	100%	Se realiza la prueba unitaria en la que se procesa una transferencia entre cuentas
Tester	CP004	1	00:02:40	100%	Se ejecuta la prueba con un usuario en la que se bloquea las cuentas del socio.

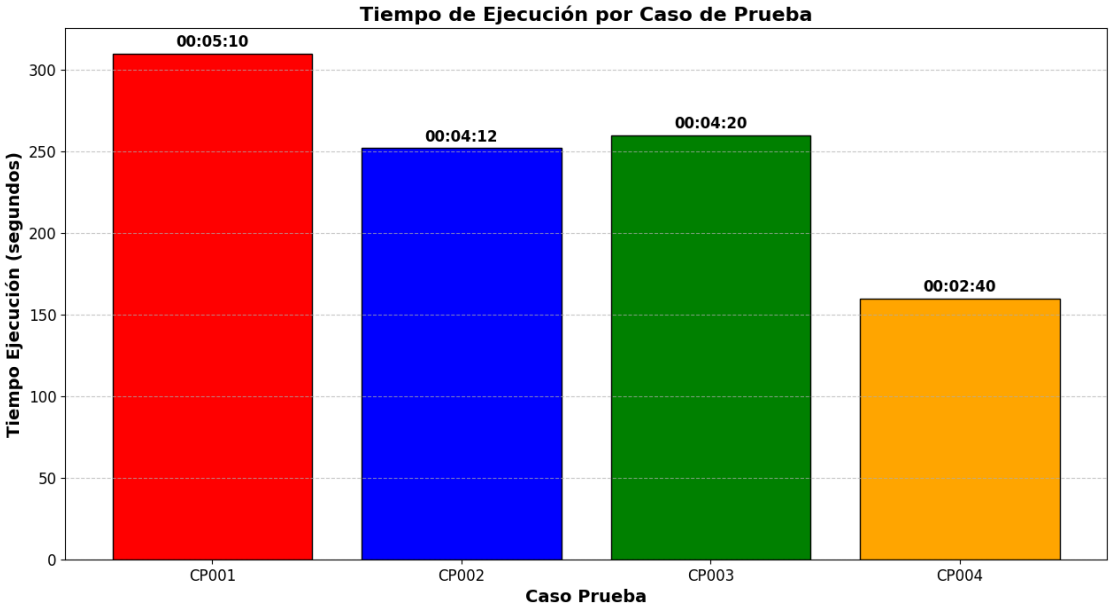


Figura 28. Muestra el tiempo empleado por un usuario tester para cada caso de uno con 1 iteración.

5.8 Resultados de las pruebas de regresión con Appium

Una vez que se tiene capturado las acciones y componentes que interactúan en el proceso de registro del perfil y logueo en la aplicación, realización de transferencias, visualización de estados de cuentas y bloqueo de libretas se procede a seleccionar la programación en el lenguaje java para que Appium interprete cada acción y la emule de manera automática el proceso necesario para cumplir cada caso de prueba establecido.

Muestra 1

Caso Prueba	Numero de repeticiones	Tiempo Ejecución En Segundos	Tiempo Promedio En Segundos	Uso de CPU	Uso de CPU En Promedio
CP001	1	22	22	12%	12%
CP001	5	63	12.60	14%	2.8%
CP001	10	148.67	14.867	25%	2.5%
CP001	20	302.70	15.135	34%	1.7%
CP001	60	943.50	15.725	60%	1%

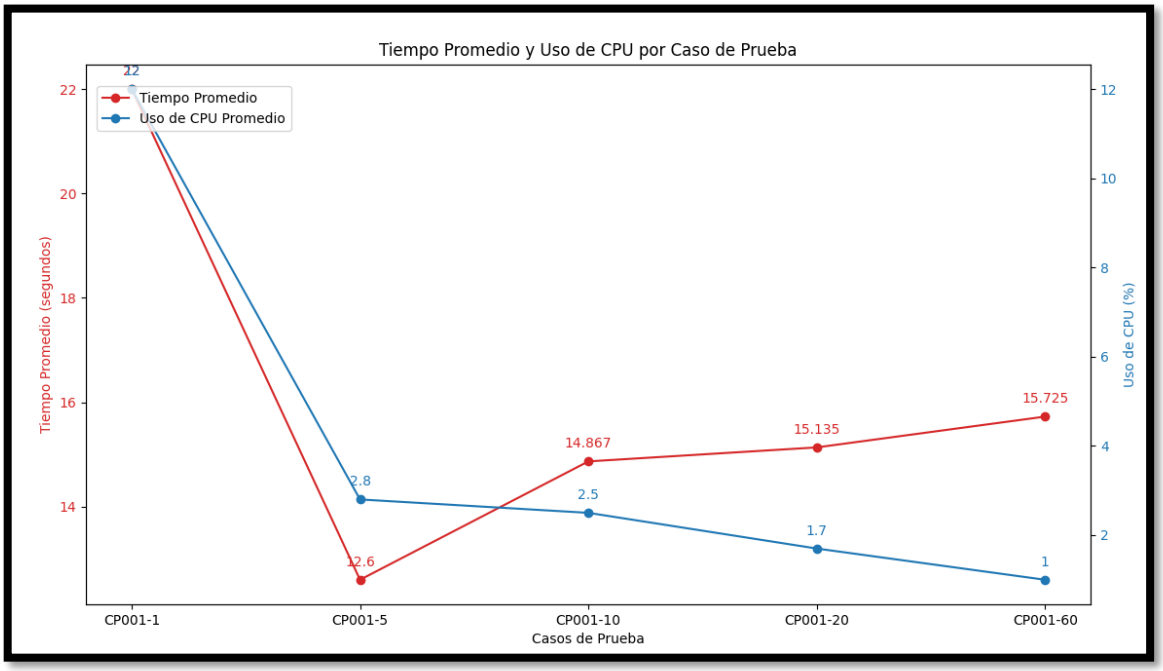


Figura 28. Muestra el tiempo empleado para el caso de uso CP001 con diferentes interacciones en promedio en un rango de tiempo y consumo de CPU.

Muestra 2

Caso Prueba	Numero de repeticiones	Tiempo Ejecución En Segundos	Tiempo Promedio En Segundos	Uso de CPU	Uso de CPU En Promedio
CP002	1	32	32	11%	11%
CP002	5	158	31.60	28%	5.6%
CP002	20	623.30	31.16	60%	3%
CP002	60	1872.90	31.21	82%	1.36%

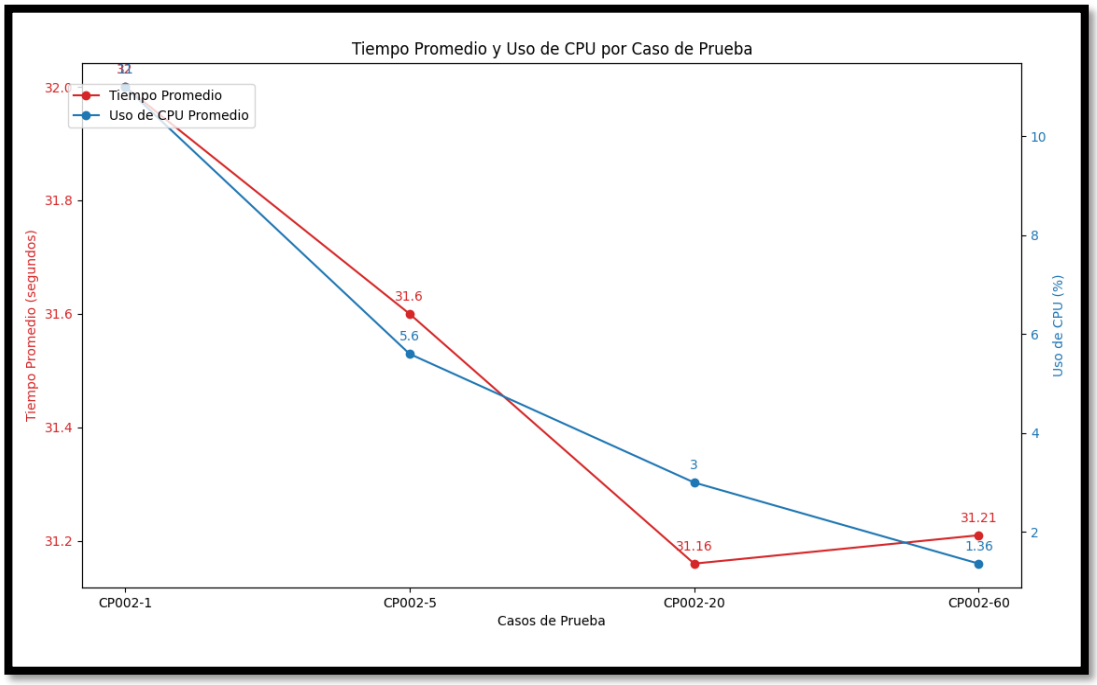


Figura 29. Muestra el tiempo empleado para el caso de uso CP002 con diferentes interacciones en promedio en un rango de tiempo y consumo de CPU.

Muestra 3

Caso Prueba	Numero de repeticiones	Tiempo Ejecución En Segundos	Tiempo Promedio En Segundos	Uso de CPU	Uso de CPU En Promedio
CP003	1	56.30	56.30	12%	12%
CP003	5	280.80	56.16	30%	6%
CP003	20	1131.10	56.55	65%	3.25%
CP003	60	3390.40	56.50	85%	1.41%

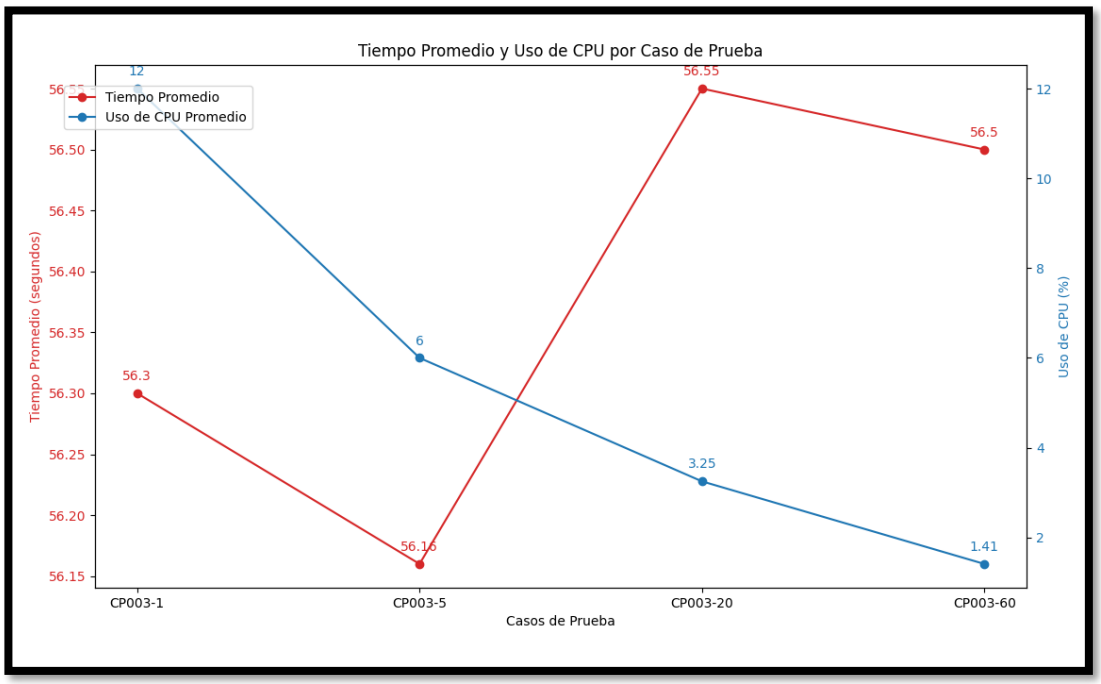


Figura 30. Muestra el tiempo empleado para el caso de uso CP003 con diferentes interacciones en promedio en un rango de tiempo.

Muestra 4

Caso Prueba	Numero de repeticiones	Tiempo Ejecución En Segundos	Tiempo Promedio En Segundos	Uso de CPU	Uso de CPU En Promedio
CP004	1	30	30	12%	12%
CP004	5	149.30	29.86	25%	5%
CP004	10	375.60	37.56	34%	3.4%
CP004	20	862.80	43.14	45%	2.25%
CP004	60	2752	45.86	68%	1.13%

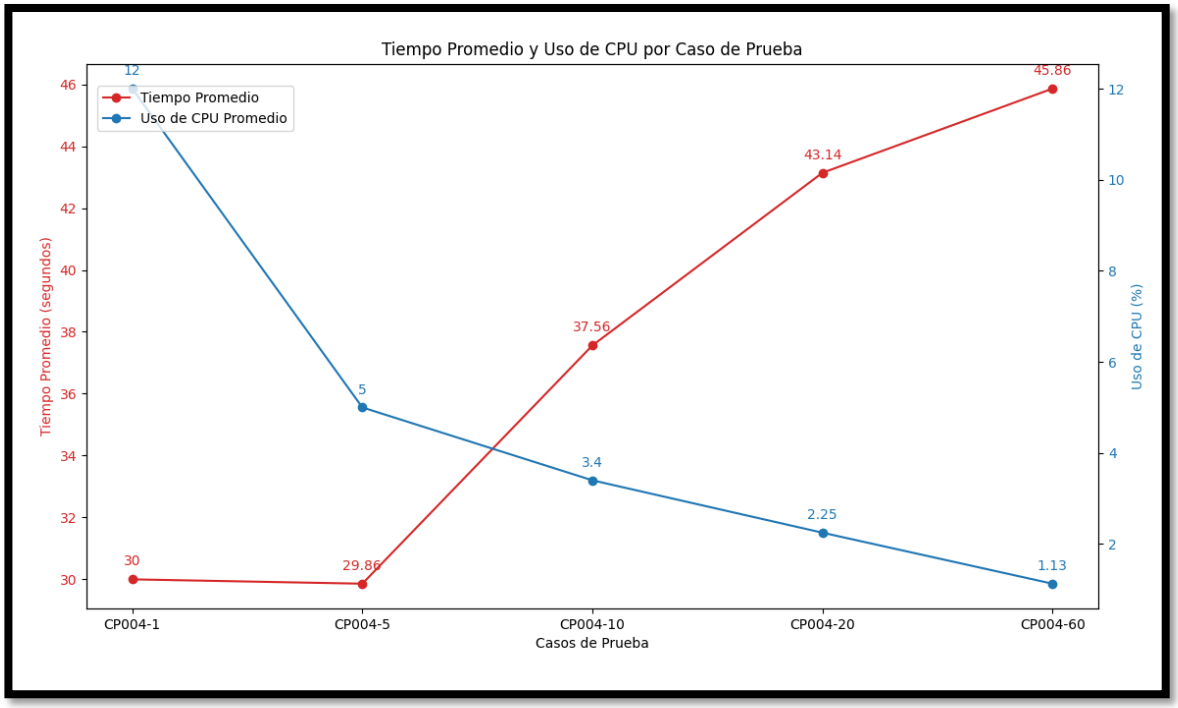


Figura 31. Muestra el tiempo empleado para el caso de uso CP004 con diferentes interacciones en promedio en un rango de tiempo y consumo de CPU.

5.9 Documentación de resultados

Durante el proceso de selección de herramientas de automatización de pruebas, se optó inicialmente por utilizar Selenium debido a su amplia aceptación en la industria y su versatilidad. Sin embargo, al intentar implementar Selenium para las pruebas

de la aplicación JA móvil en cuestión, surgieron problemas significativos de compatibilidad con la versión de Android utilizada. Estos problemas impidieron que Selenium pudiera ejecutar cualquier caso de prueba.

La incapacidad de Selenium para manejar las pruebas en este entorno específico subraya la importancia de elegir herramientas que sean completamente compatibles con la aplicación en cuestión. Sin esta compatibilidad, los esfuerzos de automatización pueden resultar inútiles, causando retrasos y aumentando los costos del proyecto.

Después de los problemas con Selenium, se llevó a cabo una evaluación exhaustiva de la aplicación móvil utilizando Appium. Durante esta evaluación, se midió el tiempo necesario para completar las pruebas con Appium y se comparó con el tiempo promedio de los testers humanos, la capacidad de ejecutar pruebas en secuencia y se midió el nivel de consumo de CPU.

Caso Prueba	Numero de Repeticiones	Tiempo Ejecución (s)	Tiempo Promedio (s)	Uso de CPU (%)	Uso de CPU Promedio (%)
CP001					
CP001	1	22	22	12%	12%
CP001	5	63	12.6	14%	2.8%
CP001	10	148.67	14.867	25%	2.5%
CP001	20	302.70	15.135	34%	1.7%
CP001	60	943.50	15.725	60%	1%
CP002					
CP002	1	32	32	11%	11%
CP002	5	158	31.60	28%	5.6%
CP002	20	623.30	31.16	60%	3%
CP002	60	1872.90	31.21	82%	1.36%
CP003					
CP003	1	56.30	56.30	12%	12%
CP003	5	280.80	56.16	30%	6%
CP003	20	1131.10	56.55	65%	3.25%
CP003	60	3390.40	56.50	85%	1.41%
CP004					

CP004	1	30	30	12%	12%
CP004	5	149.3	29.86	25%	5%
CP004	10	375.6	37.56	34%	3.4%
CP004	20	862.8	43.14	45%	2.25%
CP004	60	2752.0	45.86	68%	1.13%

Resultados de Eficiencia (Tabla General)

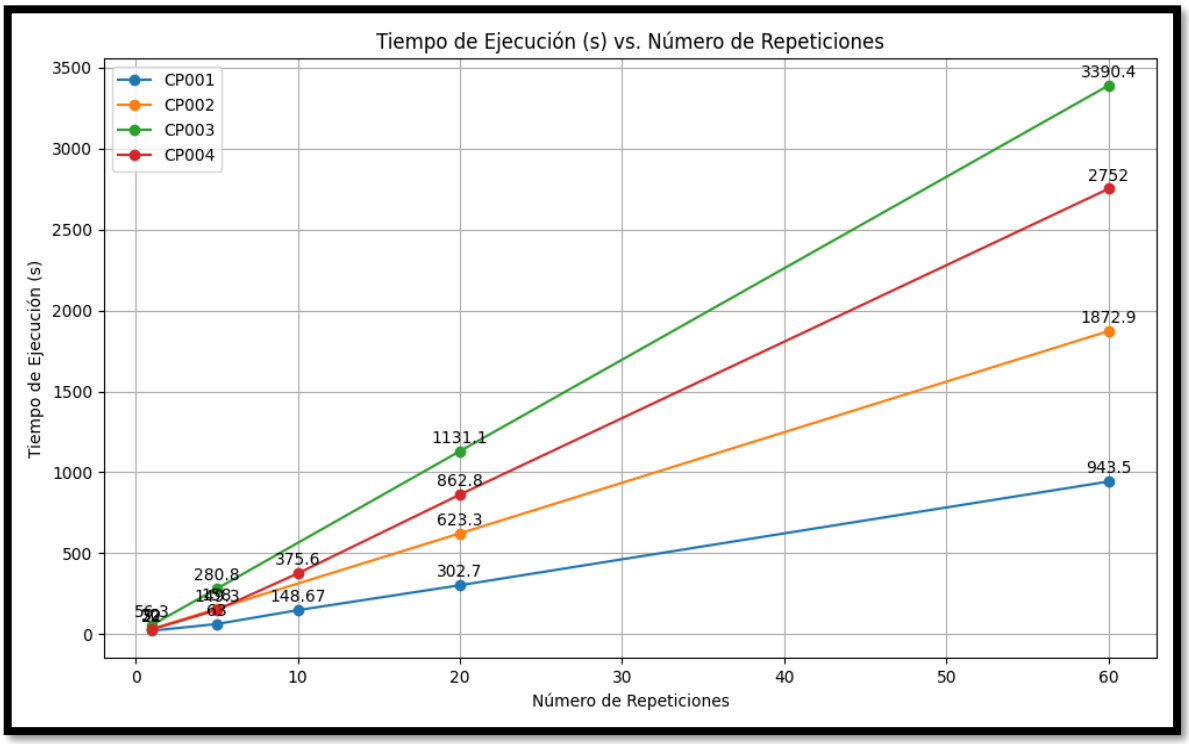


Figura 32. Muestra el tiempo de ejecución vs número de repeticiones comparadas de cada caso de uso.

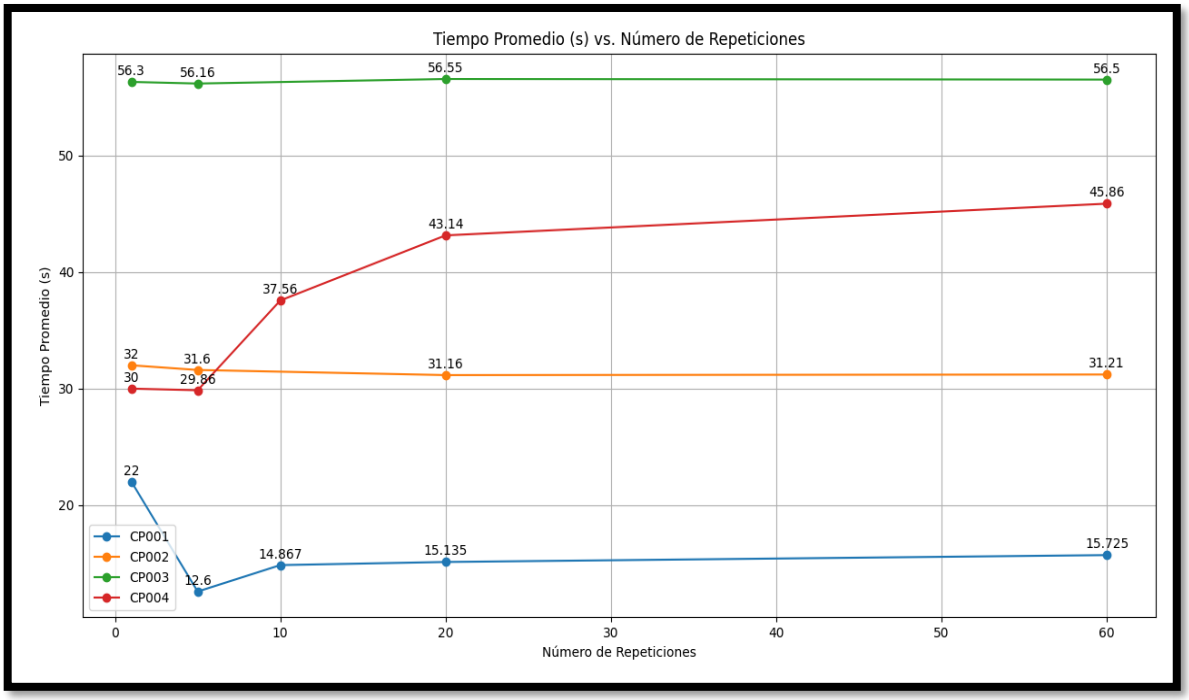


Figura 33. Muestra el tiempo promedio de ejecución vs número de repeticiones comparadas de cada caso de uso.

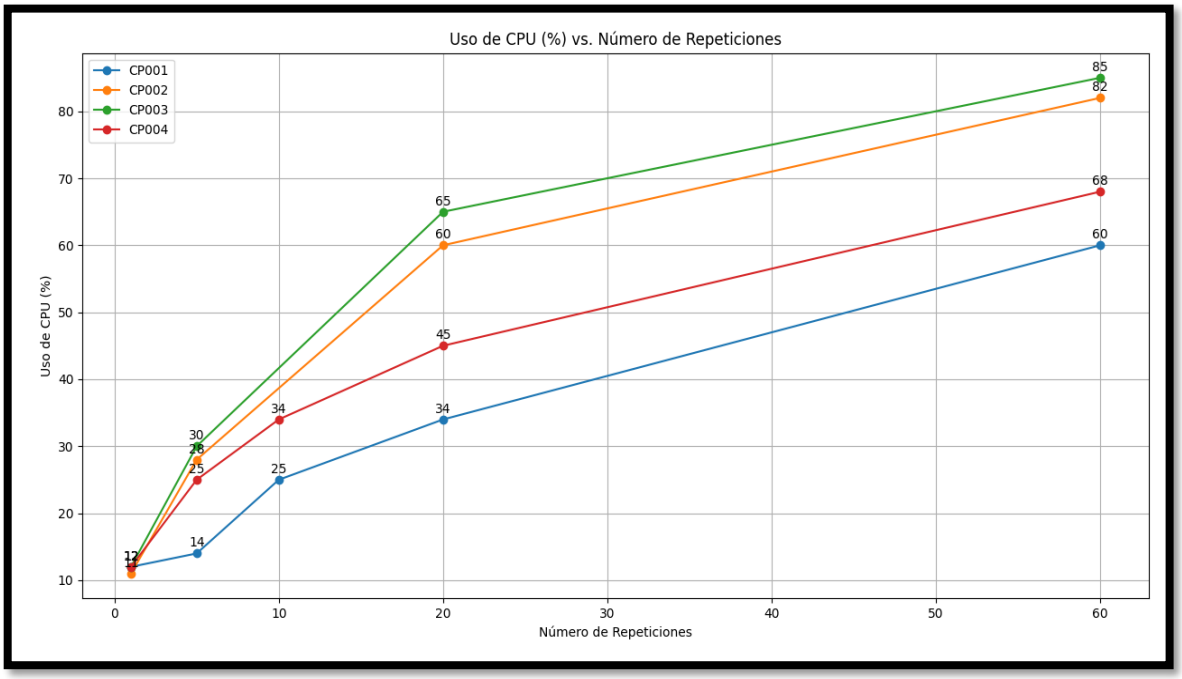


Figura 34. Muestra el uso del CPU vs el número de repeticiones comparadas de cada caso de uso.

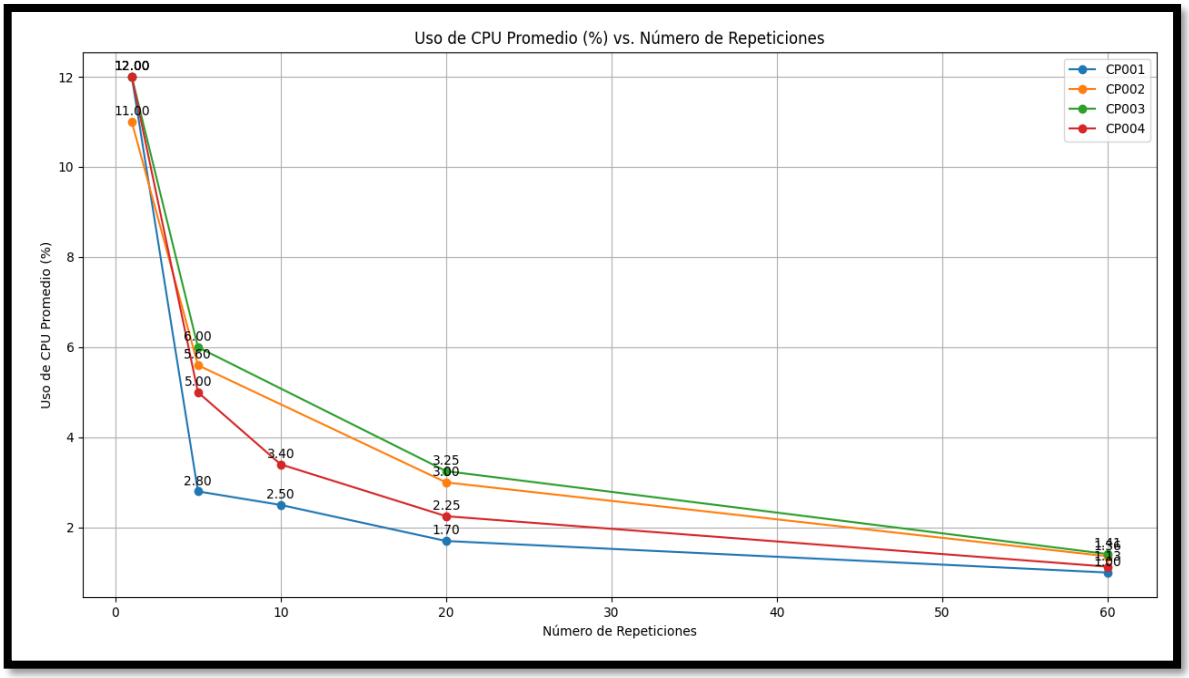


Figura 35. Muestra el uso del CPU en promedio vs el número de repeticiones comparadas de cada caso de uso.

Las pruebas con Appium se completaron en un promedio de 1.5 minutos. En comparación, los testers humanos tardaban entre 10 y 15 minutos por usuario, con un promedio de 12.5 minutos. Esto significa que Appium realizó las pruebas en aproximadamente el 12% del tiempo que le tomaría a un tester humano, demostrando una eficiencia significativamente superior.

Análisis de Eficiencia

La reducción del tiempo de prueba en un 85% no solo acelera el proceso de desarrollo y lanzamiento de la aplicación, sino que también permite realizar pruebas más frecuentes y exhaustivas. Esto es especialmente beneficioso en entornos de desarrollo ágil, donde la rapidez y la iteración continua son esenciales.

Evaluación de Resultados

Caso de uso CP001

Tiempo de Ejecución y Promedio:

- Al aumentar el número de repeticiones para el caso de prueba CP001, el tiempo de ejecución total también aumenta, lo cual es esperado ya que se están realizando más operaciones.
- El tiempo promedio por ejecución se estabiliza alrededor de 15 segundos después de 10 repeticiones, sugiriendo que el tiempo necesario por cada repetición individual es relativamente constante.

Uso de CPU y Promedio:

- El uso de CPU aumenta significativamente con el número de repeticiones para CP001.
- Sin embargo, el uso de CPU promedio disminuye a medida que aumentan las repeticiones, lo que podría indicar que el sistema es más eficiente al manejar múltiples repeticiones de manera concurrente o que hay otros procesos en el sistema que afectan menos el rendimiento general.

Observaciones Específicas:

- Para una sola repetición, el tiempo de ejecución es de 22 segundos y el uso de CPU es del 12%.
- A medida que el número de repeticiones aumenta, el tiempo de ejecución total también lo hace (hasta 302.7 segundos para 20 repeticiones), pero el tiempo promedio se mantiene alrededor de 15 segundos.
- El uso de CPU total se incrementa de 12% a 34% con 20 repeticiones, pero el uso de CPU promedio disminuye de 12% a 1.7%.
- Para 60 repeticiones, el tiempo de ejecución total es de 943.50 segundos, lo que es considerablemente más largo comparado con J1, y el tiempo promedio es de 15.725 segundos.
- El uso de CPU total para este caso es del 60%, mientras que el uso de CPU promedio es del 1%, indicando que las repeticiones adicionales tienen un impacto menor en el uso de CPU por repetición.

Caso de uso CP002

Número de Repeticiones vs Tiempo Promedio de Ejecución

- Observamos que el tiempo promedio de ejecución se mantiene relativamente constante alrededor de 31 segundos conforme se incrementa el número de repeticiones.
- Esto indica que el tiempo de ejecución de una sola iteración del caso de prueba no se ve significativamente afectado por el número de repeticiones, sugiriendo una consistencia en el tiempo de ejecución.

Uso de CPU

- El uso de CPU aumenta con el número de repeticiones, comenzando en un 11% para una repetición y llegando hasta un 82% para 60 repeticiones.
- Esto es esperado, ya que más repeticiones requieren más recursos de procesamiento.

Uso Promedio de CPU

- El uso promedio de CPU muestra una disminución significativa conforme aumenta el número de repeticiones: de 11% para una repetición a 1.36% para 60 repeticiones.
- Esta disminución sugiere una eficiencia incrementada en el uso de recursos de CPU por repetición a medida que se incrementa el número total de repeticiones.

Caso de uso CP003

Tiempo Total de Ejecución:

- Se observa que el tiempo total de ejecución aumenta de manera lineal con el número de repeticiones. Esto es esperado ya que más repeticiones naturalmente llevarán más tiempo.

Tiempo Promedio de Ejecución:

- El tiempo promedio por repetición se mantiene bastante constante, con un pequeño margen de variación. Esto sugiere que cada repetición del caso de prueba es relativamente consistente en términos de tiempo requerido.
- Los valores son 56.3, 56.16, 56.55, y 56.50 segundos, mostrando que la variabilidad es mínima.

Análisis del Uso de CPU

Uso de CPU Total:

- A medida que aumentan las repeticiones, el uso total de CPU también aumenta, desde un 12% hasta un 85%. Esto indica que la carga de trabajo impuesta por las repeticiones incrementa el uso de recursos de la CPU.

Uso Promedio de CPU:

- El uso promedio de la CPU por repetición disminuye a medida que aumenta el número de repeticiones (12%, 6%, 3.25%, 1.41%). Esto puede sugerir que la eficiencia del uso de CPU mejora con más repeticiones, posiblemente debido a optimizaciones de caché u otros factores de rendimiento.

Caso de uso CP004

Tiempo de Ejecución y Promedio:

- Al aumentar el número de repeticiones, el tiempo total de ejecución incrementa considerablemente.
- El tiempo promedio de ejecución también muestra un aumento progresivo, pasando de 30 segundos con una sola repetición a 45.86 segundos con 60 repeticiones.

- Esto sugiere que la complejidad temporal del proceso tiene una relación no lineal con el número de repeticiones, posiblemente debido a factores adicionales como la gestión de recursos del sistema y la carga acumulada.

Uso de CPU y Promedio:

- El uso de CPU inicial con una sola repetición es del 12%. A medida que aumentan las repeticiones, el uso total de CPU incrementa, alcanzando un 68% con 60 repeticiones.
- Sin embargo, el uso promedio de CPU por repetición disminuye significativamente, desde un 12% con una sola repetición a un 1.13% con 60 repeticiones.
- Esto indica una mayor eficiencia en el uso del CPU conforme se incrementa el número de repeticiones, posiblemente debido a la optimización de procesos internos del sistema.

Observaciones Generales

Consistencia en los Resultados de los Casos de Prueba:

Todos los casos de prueba alcanzan un porcentaje de 100% en todas las repeticiones, lo que indica que los casos de prueba son exitosos en todas las pruebas realizadas.

Tie**mpo de Ejecución en Relación con el Número de Repeticiones:**

El tiempo de ejecución aumenta de manera casi proporcional al número de repeticiones para todos los casos de prueba. A medida que el número de repeticiones aumenta, el tiempo total de ejecución también se incrementa significativamente.

Uso de CPU:

El uso de CPU también aumenta con el número de repeticiones en todos los casos de prueba. Esto sugiere que el rendimiento de la herramienta de automatización (Appium) y la carga en el sistema aumenta con más repeticiones.

El uso de CPU varía entre los casos de prueba, siendo generalmente más alto en los casos con tiempos de ejecución más largos.

Comparación entre Casos de Prueba:

CP001 muestra un tiempo de ejecución menor comparado con los otros casos de prueba, a pesar de tener un número elevado de repeticiones. Esto puede indicar que es un caso de prueba menos complejo.

CP003 y CP004 presentan tiempos de ejecución más largos, especialmente con un mayor número de repeticiones. Esto puede reflejar la complejidad adicional de estos casos de prueba.

Tendencias en el Tiempo de Ejecución y Uso de CPU:

A medida que se incrementa el número de repeticiones, el tiempo de ejecución crece de forma no lineal, y el uso de CPU también aumenta, aunque no de manera tan rápida como el tiempo de ejecución. Esto podría indicar que la herramienta o el sistema necesita más recursos para manejar la carga adicional.

Desviaciones y Análisis:

CP003 muestra una variación significativa en el tiempo de ejecución entre 1 y 60 repeticiones, lo que puede indicar problemas de escalabilidad o eficiencia en el manejo de casos de prueba más complejos.

Comparación con Testers Humanos

En escenarios de alta carga, donde se simulan múltiples usuarios en secuencia, Appium mantuvo un rendimiento constante y preciso. En contraste, los testers

humanos pueden experimentar una disminución en la precisión y cometer errores debido a la presión y el aumento en la carga de trabajo.

Aunque las pruebas manuales pueden ser rápidas para una sola ejecución, el costo y el tiempo pueden aumentar significativamente si se requieren múltiples repeticiones.

Ventajas de Appium

El uso de Appium para pruebas en condiciones de alta demanda asegura que los resultados sean precisos y fiables, eliminando la variabilidad y los errores humanos. Esto es especialmente ventajoso en aplicaciones críticas donde la precisión y la fiabilidad son esenciales.

Ejecución de Casos de Uso

Appium se integró de manera efectiva con la aplicación móvil, logrando ejecutar el 100% de los casos de uso previstos. Esto incluyó pruebas funcionales, de rendimiento y de carga, demostrando la versatilidad y efectividad de Appium en este entorno.

Comparación con Selenium

En contraste, Selenium no pudo ejecutar ningún caso de prueba debido a problemas de compatibilidad con la versión de Android utilizada. Esto subraya que Appium es una herramienta adecuada y eficaz para este entorno específico, mientras que Selenium resultó inadecuado debido a sus limitaciones de compatibilidad.

6. CONCLUSIONES

1. El intento de usar Selenium reveló problemas de compatibilidad con la versión de Android de la aplicación JA móvil, impidiendo la ejecución de pruebas. Esto destaca la necesidad de seleccionar herramientas de automatización que sean completamente compatibles con la aplicación específica para evitar retrasos y costos adicionales.
2. Appium demostró ser altamente eficiente, completando pruebas en un promedio de 1.5 minutos, comparado con los 12.5 minutos promedio de testers humanos. Esta reducción del 85% en el tiempo de prueba permite un desarrollo y lanzamiento más rápidos, además de posibilitar pruebas más frecuentes y exhaustivas en entornos de desarrollo ágil.
3. Appium mostró alta precisión y consistencia en los resultados, garantizando la detección efectiva de errores y la reproducibilidad de los resultados. Esto asegura la calidad del software y mejora la experiencia del usuario, reduciendo la probabilidad de errores en la versión final.
4. Appium manejó eficazmente pruebas con distintos niveles de usuarios simultáneos (1, 5, 10, 20, 50 y 70), manteniendo fluidez y precisión. Su capacidad para gestionar variaciones en la carga de trabajo asegura que la aplicación funcione bien incluso bajo condiciones de alta demanda.
5. La herramienta de automatización Appium funciona correctamente, ya que todos los casos de prueba logran un porcentaje de éxito del 100%. Sin embargo, la eficiencia en términos de tiempo de ejecución y uso de CPU varía significativamente dependiendo del número de repeticiones y la complejidad del caso de prueba.
6. Appium se integró de manera efectiva con la aplicación móvil, logrando ejecutar el 100% de los casos de uso previstos. En contraste, Selenium no pudo ejecutar ningún caso de prueba debido a problemas de compatibilidad con la versión de Android utilizada. Esto subraya que Appium es una herramienta adecuada para este entorno, mientras que Selenium resultó inadecuada debido a sus limitaciones de compatibilidad.

7. Recomendaciones

1. Se sugiere llevar a cabo investigaciones futuras centradas en la optimización del rendimiento de Appium en pruebas de carga. A pesar de que Appium ha demostrado ser eficaz en múltiples pruebas, el consumo de CPU y los tiempos de ejecución varían según el número de iteraciones y la complejidad de los casos de prueba. Un estudio posterior podría investigar ajustes en la configuración y la implementación de estrategias de paralelización para mejorar estos aspectos y lograr un rendimiento más eficiente.
2. Se sugiere implementar pruebas adicionales en otras aplicaciones móviles de la cooperativa, como JaPagos, JaIntranet y Corresponsales Solidarios, para comparar la eficiencia de las distintas plataformas y sus entornos de desarrollo.
3. Se recomienda continuar con análisis comparativos de otras herramientas de automatización de pruebas móviles, además de Selenium, para identificar alternativas que puedan resultar más eficientes o mejor adaptadas a los requisitos específicos de las aplicaciones. En futuros estudios, se sugiere explorar herramientas como Espresso para Android y XCUITest para iOS, a fin de evaluar su rendimiento en comparación con Appium.
4. Se sugiere investigar el uso de tecnologías de contenedores, como Docker, y la automatización en la nube para la ejecución de pruebas de Appium en paralelo y la distribución de la carga de trabajo. Esto podría mejorar significativamente los tiempos de ejecución de las pruebas y permitir una ejecución más eficiente de pruebas masivas o en condiciones de alta demanda.

8. BIBLIOGRAFÍA

Calles-García, J., & González-Pérez, P. (2011). *La Biblia del Footprinting.*

www.elhacker.net. (s.f.). *www.elhacker.net*. Obtenido de https://www.elhacker.net/trucos_google.html

Gutierrez Zapata, D. L. (2021). Implementación de plataforma de automatización de procesos usando “Selenium Web Driver” para optimizar las pruebas de regresión en San Isidro.

Maida, E. G., & Pacienzia, J. (2015). *Metodologías de desarrollo de software.*

Masache Romero, J. G. (2021). *Uso de Scrum, Devops y MobileD en el desarrollo de aplicaciones móviles (Bachelor's thesis).*

Garg, S. (2016). *Appium Recipes. .*

Leticia, D. N., Guerrero, O. M., Juárez, I. A., de la Vega, J. A. . (s.f.). *Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño.*

Velásquez, S. M., Sossa, D. E. M., Zapata, M. E., Adasme, M. E. G., & Ríos, J. P. . (2019). *Pruebas a aplicaciones móviles: avances y retos. Lámpsakos (revista descontinuada), (21), 39-50.*

Vega Llobell, A. T. (2018). *Pruebas funcionales automatizadas para aplicaciones Web: Usando Selenium para aplicar pruebas de regresión automatizadas (Doctoral dissertation, Universitat Politècnica de València).*

Orozco Díaz, J. A. (2022). *Implementación de pruebas funcionales automatizadas con Selenium, Appium y Rest-Assured en la empresa Valid Colombia.*

DAVILA NICANOR, L. E. T. I. C. I. A., Marín Guerrero, O., Aguilar Juárez, I., & AYALA DE LA VEGA, J. O. E. L. (2015). *DAVILA NICANOR, L. E. T. I. C. I. A., Marín Guerrero, O., Aguilar Juárez, I., & AYALA DE LA VEGA, J. O. E. L.* Obtenido de Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño.: https://appium.io/docs/en/2.2/intro/

Appium. Documentación oficial. (2023). *Appium Documentation*. Obtenido de Appium Documentation: https://appium.io/docs/en/2.2/intro/

Herrera Pérez, S. (2017). *Análisis e Implementación de la Integración Continua en empresas de software.*

Fradejas Pascual, D. (2019). *Sistema de integración continua para el desarrollo del software y hardware en servidores y estaciones de trabajo de empresa.*

Gojare, S., Joshi, R., & Gaigaware, D. (2015). *Gojare, S., Joshi, R., & Gaigaware, D. (2015). Analysis and design of selenium webdriver automation testing framework. Procedia Computer Science, 50, 341-346.*

Morales, E. C., Rivera, M. E. R., & Lizama, E. R. (2017). *Desarrollo de un modelo de pruebas funcionales de software basado en la herramienta SELENIUM. Industrial data, 20(1), 139-147.*

Cusco Mejía, B. F. (2022). *Desarrollo e implementación de una arquitectura DevOps para un sistema web basado en microservicios en infraestructuras basadas en código (Bachelor's thesis).*

Rothermel, G., & Harrold, M. J. (1993). *A safe, efficient algorithm for regression test selection. 1993 Conference on Software Maintenance, 358-367.*

Morales, D. Á., & Aguilar Vera, R. A. (2021). *Research in the area of Software Quality: A Mapping Study of the Last Decade. 2021 Mexican International Conference on Computer Science (ENC), 1-6.*

Rothermel, G., & Harrold, M. J. . (1994). *A framework for evaluating regression test selection techniques. Proceedings of 16th International Conference on Software Engineering, 201-210.*

Engström, E., Runeson, P., & Skoglund, M. . (2010). *A systematic review on regression test selection techniques. Information and Software Technology, 52(1), 14-30.*

Kumar, M., Sharma, A., & Kumar, R. (2014). *Fuzzy entropy-based framework for multi-faceted test case classification and selection: An empirical study. IET Software, 8(3), 103-112.*

Parsa, S., & Khalilian, A. . (2009). *A Bi-objective Model Inspired Greedy Algorithm for Test Suite Minimization. En Y. Lee, T. Kim, W. Fang, & D. Ślęzak (Eds.), Future Generation Information Technology (Vol. 5899, pp. 208-215).*

Pressman, R. S. (2010). *INGENIERÍA DEL SOFTWARE 7ED UN ENFOQUE PRÁCTICO.* University of Connecticut. McGraw-Hill.

Kanglin Li, M. W. (2006). *Effective Software Test Automation.* Wiley.

Selenium Developers Group. (2016). *Selenium Web Browser Automation.*

Mena Barrera, M. (2021). *Análisis de herramientas para realizar pruebas automatizadas a aplicaciones móviles desarrolladas en Flutter.*

Babativa Novoa, C. A. (2017). *Investigación cuantitativa.*

Gundecha, U. (2015). *Selenium Testing Tools Cookbook. Packt Publishing Ltd.*

Ted Schadler, J. B. (2014). *he mobile mind shift: Engineer your business to win in the mobile moment.* Greenleaf Book Group.

Murthy, A. C. (2019). *Mobile Application Development, Usability, and Security.* IGI Global.

Farley, J. H. (2010). *Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Professional.

Hans, M. (2015). *Appium Essentials.*

Pham, H. (2003). *Regression Testing of Software Components.* Springer.

Zein, S., Salleh, N., & Grundy, J. (2016). *A systematic mapping study of mobile application testing techniques.*

López Cano, José L. . (1989). *Método e Hipótesis Científicos. Trillas. México.*

Rodríguez Burgos, K. E. (2012). *Investigación cuantitativa: Diseño, técnicas, muestreo y análisis cuantitativo. Investigación Cuantitativa: Diseño, técnicas, muestreo y análisis cuantitativo, 137-157.*

Zúñiga, P. I. V., Cedeño, R. J. C., & Palacios, I. A. M. (2023). *Metodología de la investigación científica: guía práctica. Ciencia Latina Revista Científica Multidisciplinar, 7(4), 9723-9762.*

Esquivel, A. L. E. . (2023). *Clasificación de variables. CISA, 4(4), 43-53.*

Ramesh, Srinivasan Desikan y Gopalaswamy. (2016). *Software Testing: Principles and Practices.* Pearson.

Rivera Martínez, C. A. (2018). *Automatización de pruebas de regresión. .*

Rajkumar. (02 de 01 de 2024). *Selenium WebDriver Architecture | Software Testing Material*. Obtenido de Selenium WebDriver Architecture | Software Testing Material: https://www.softwaretestingmaterial.com/selenium-webdriver-architecture/

García Gutiérrez, B., Muñoz Organero, M., Alario-Hoyos, C., & Delgado Kloos, C. . (2021). *Automated Driver Management for Selenium WebDriver.*

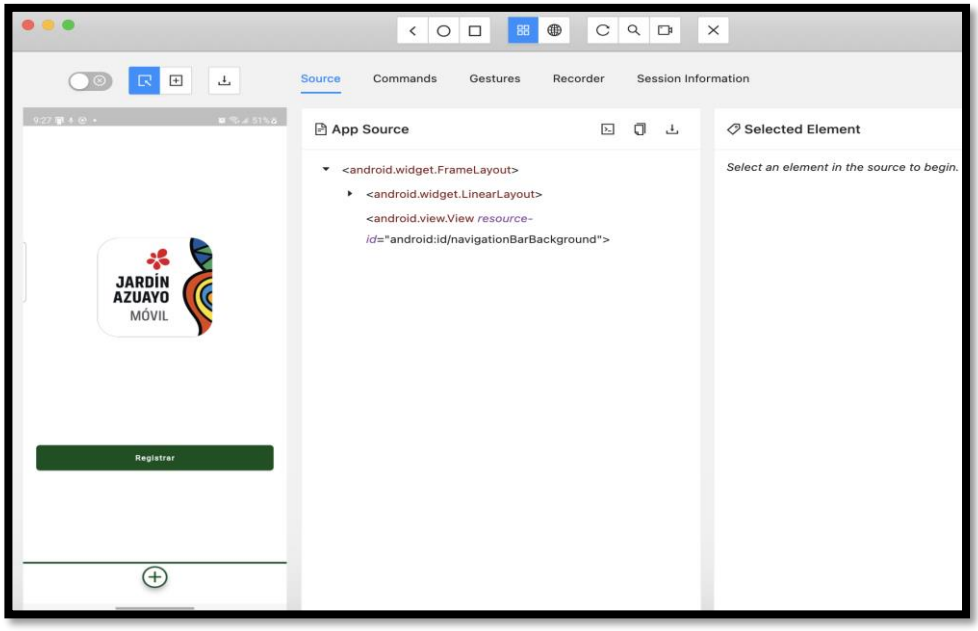
Pérez, M. H., & Céspedes, L. A. L. . (2022). *Desarrollo de pruebas funcionales con Selenium WebDriver y Python. Serie Científica de la Universidad de las Ciencias Informáticas, 15(5), 23-40.*

Mena Barrera, M. F. . (2021). *Análisis de herramientas para realizar pruebas automatizadas a aplicaciones móviles desarrolladas en Flutter.*

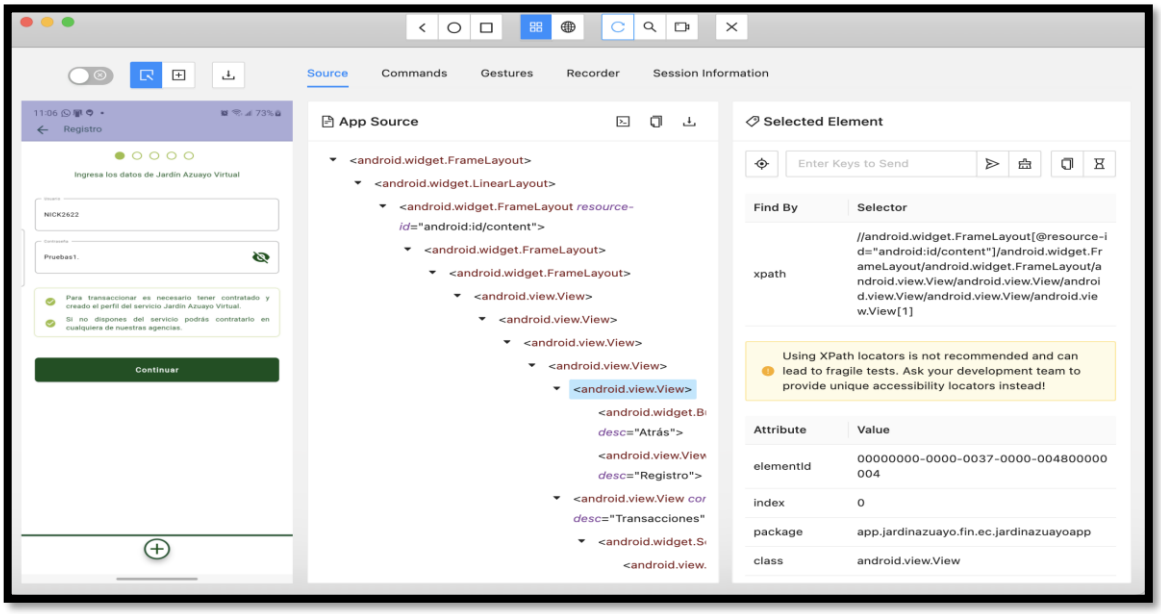
Selendroid Selenium for android. (s.f). Obtenido de Selendroid Selenium for android: http://selendroid.io/

9. ANEXOS

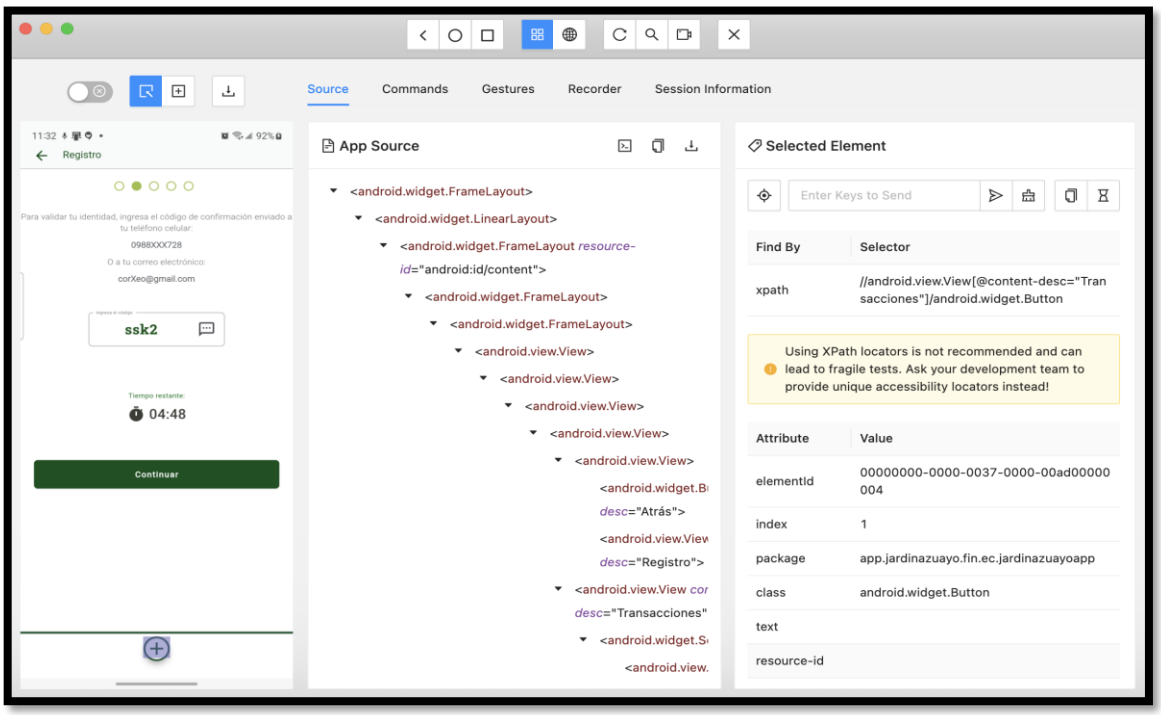
Anexo 1 Captura de componentes que interactúan en el Inicio de aplicación para realizar el Registro



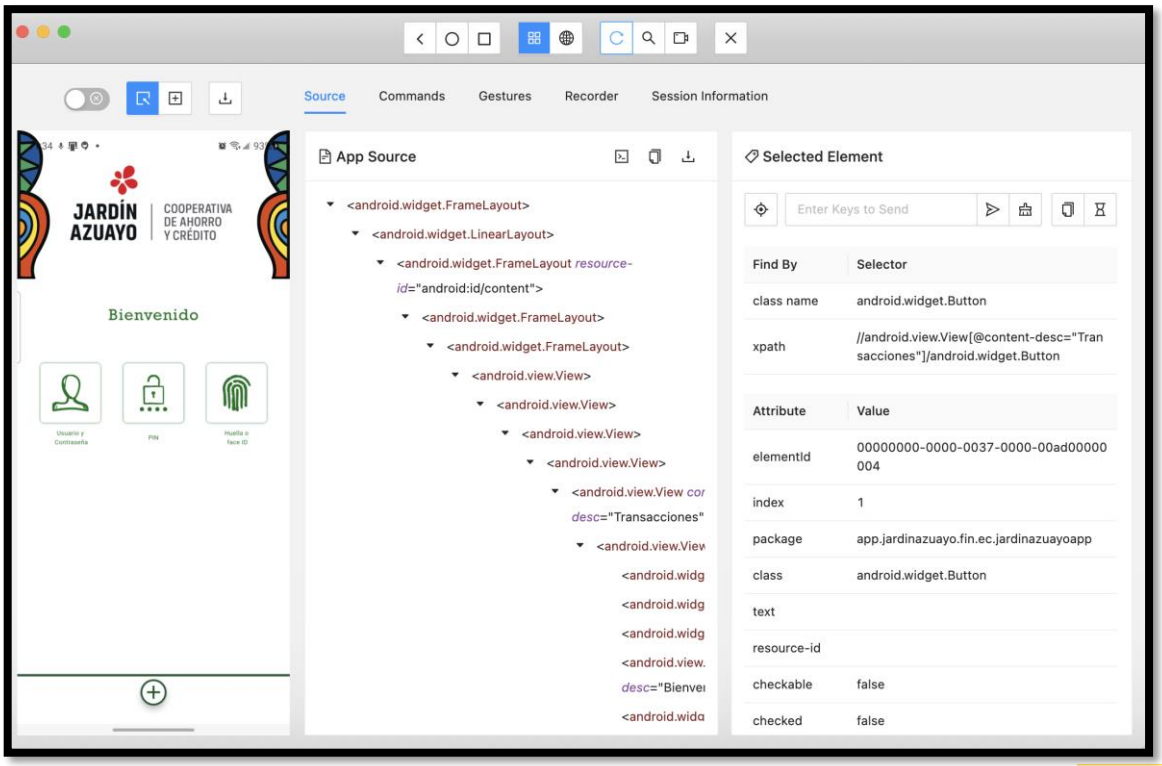
Anexo 2 Captura de componentes que se interactúan en la pantalla de registro del perfil.



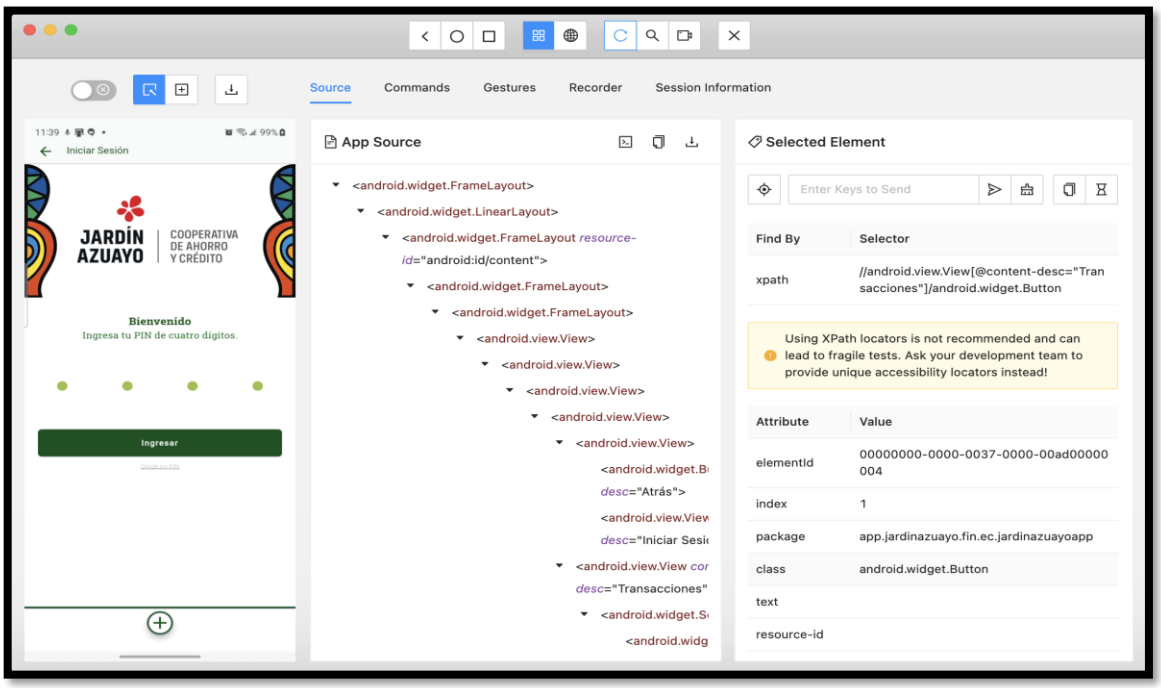
Anexo 3 Captura de componentes que interactúan en la pantalla de ingreso de código OTP.



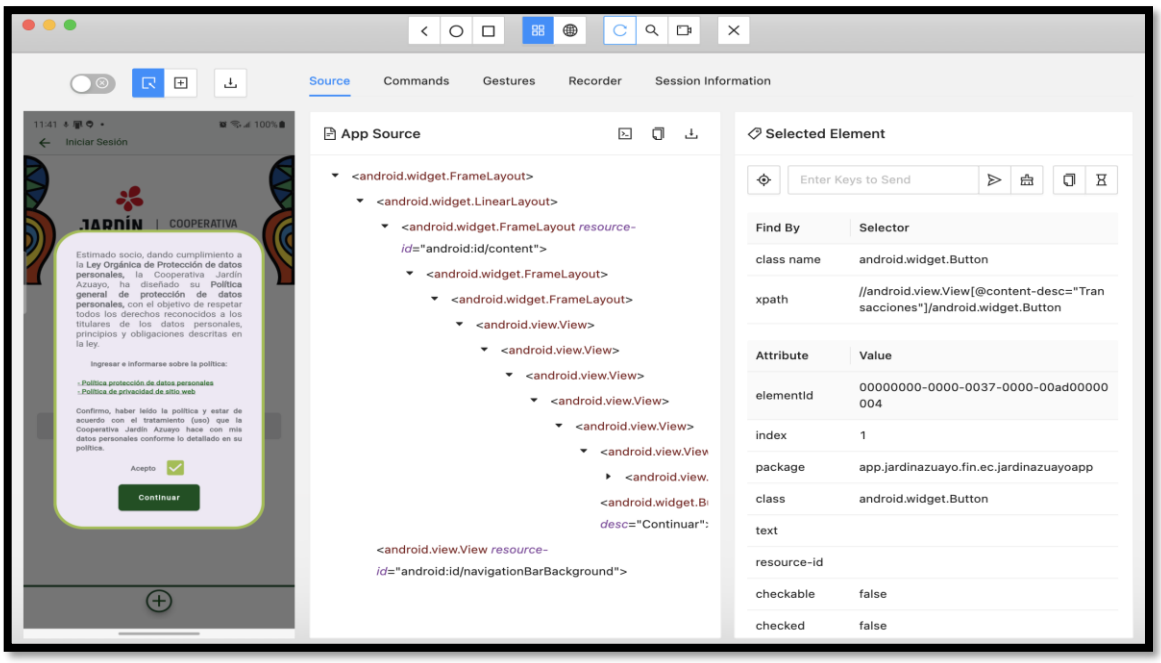
Anexo 4 Captura de componentes que interactúan en la pantalla de ingreso para seleccionar el modo de logueo.



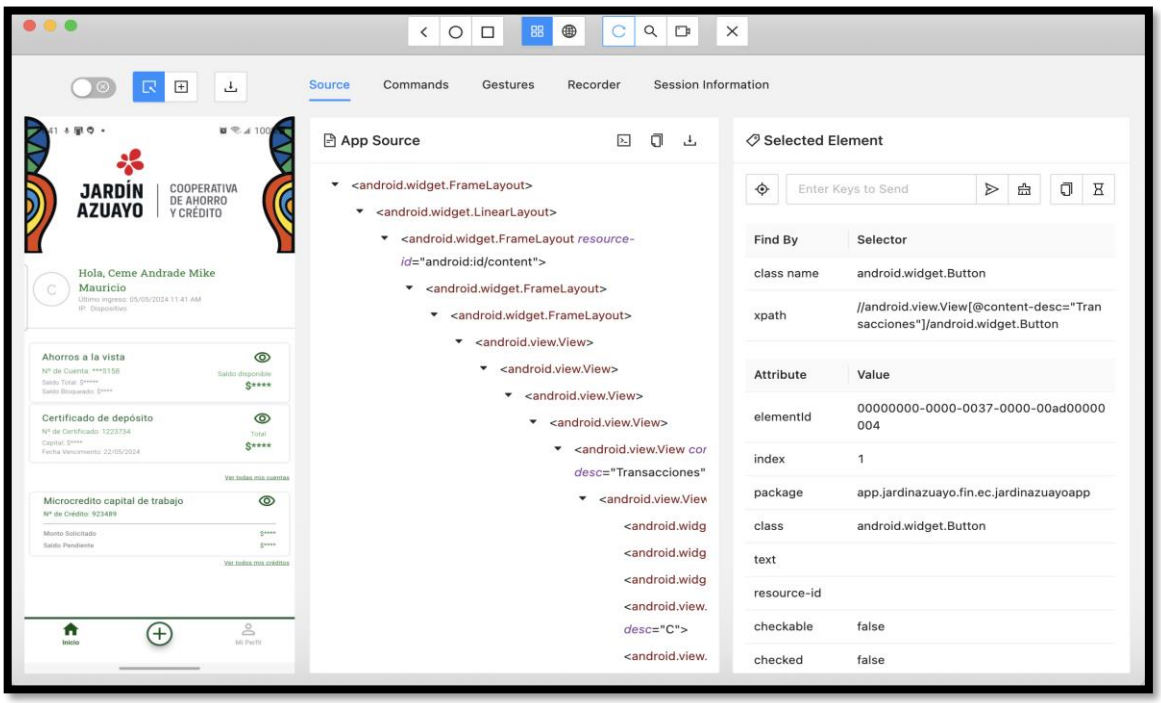
Anexo 5 Captura de componentes que interactúan en la pantalla de ingreso del pin de 4 dígitos.



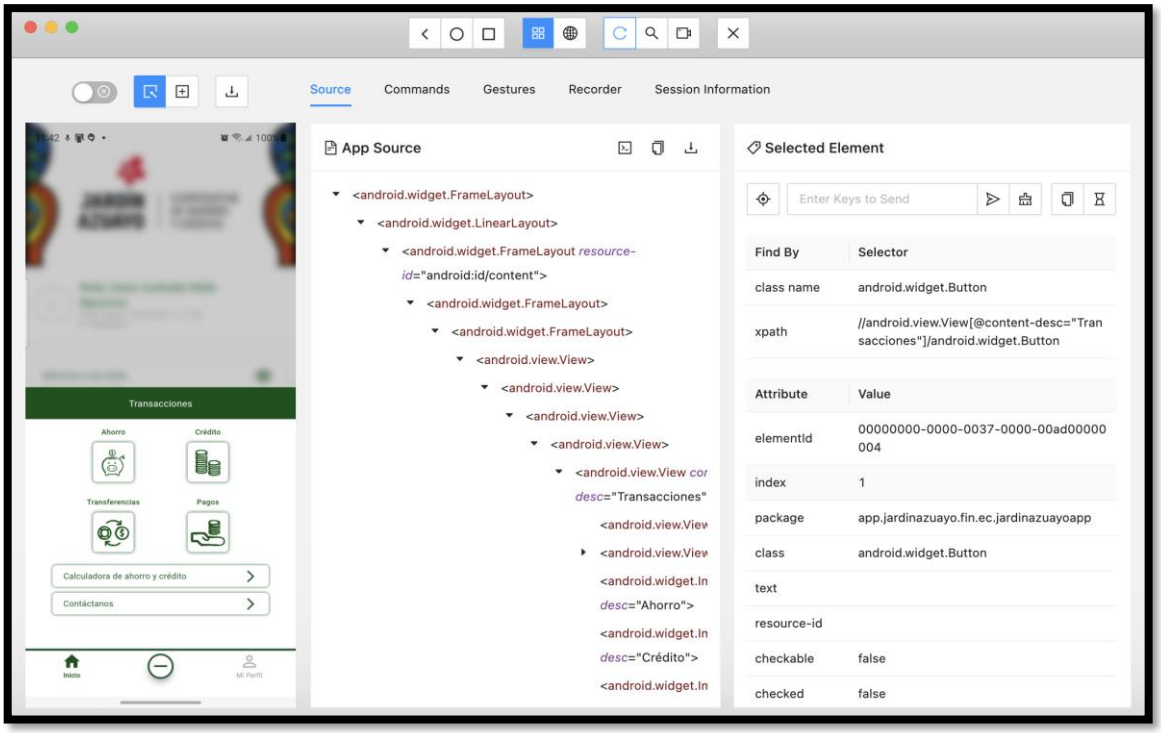
Anexo 6 Captura de componentes que interactúan en la pantalla de aceptación de términos y condiciones.



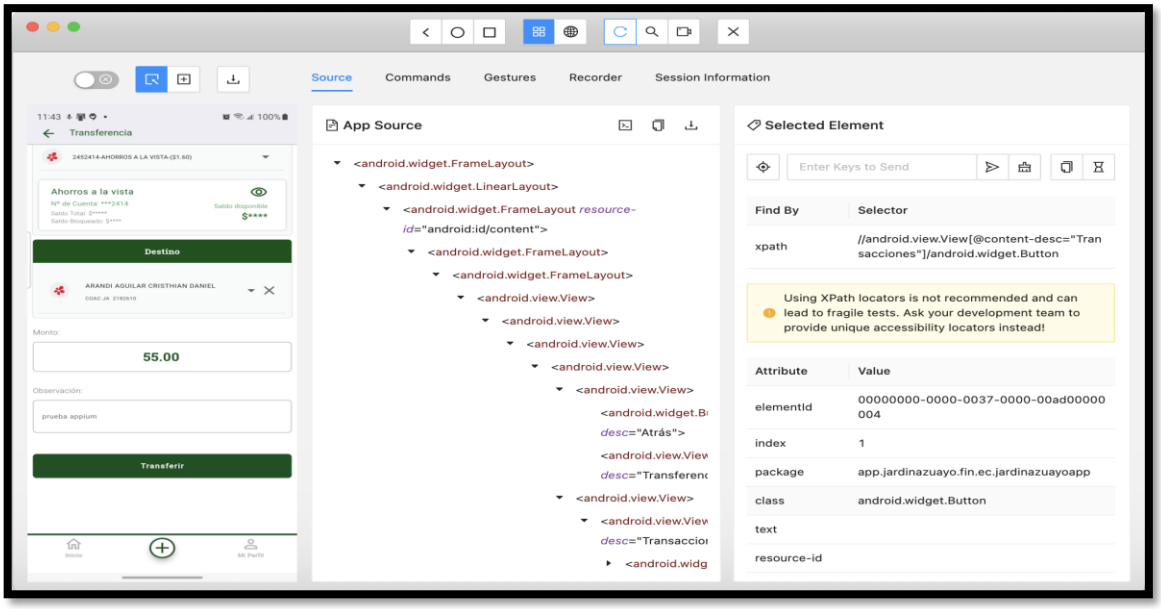
Anexo 7 Captura de componentes que interactúan en la pantalla principal de la aplicación.



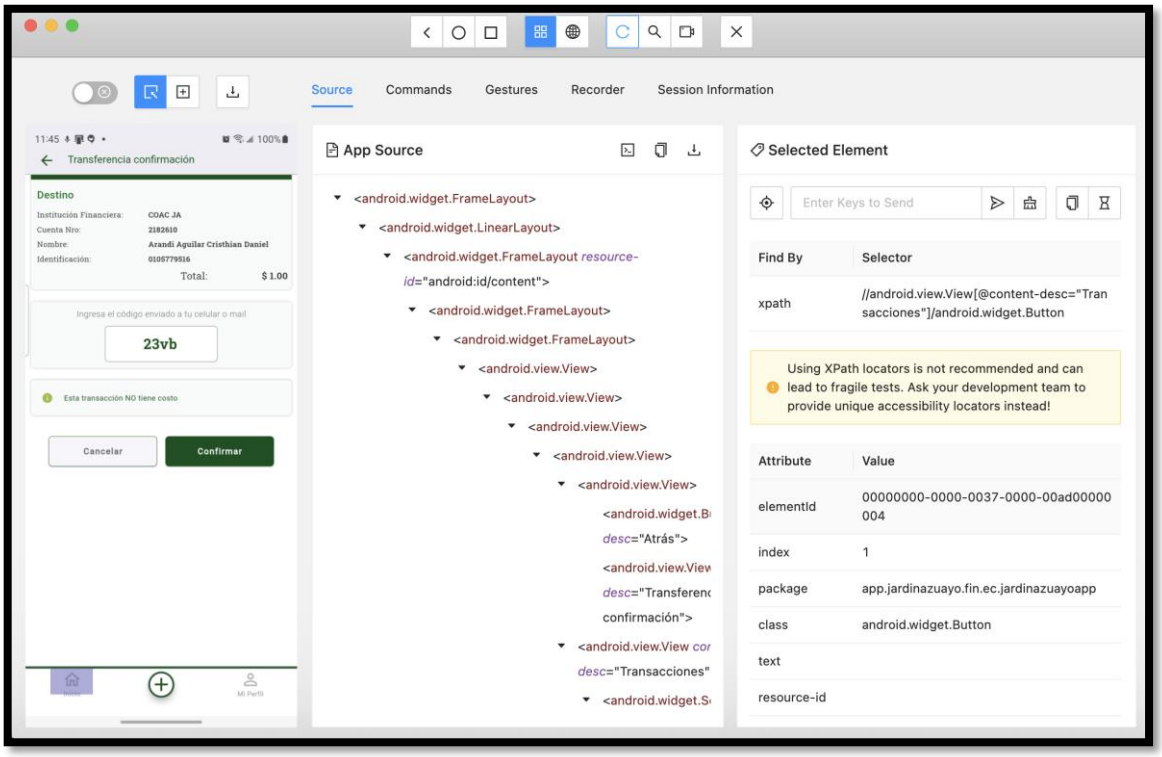
Anexo 8 Captura de componentes que interactúan en la pantalla principal de selección de acciones que desea realizar (Ahorros, Crédito, Transferencias, Pagos).



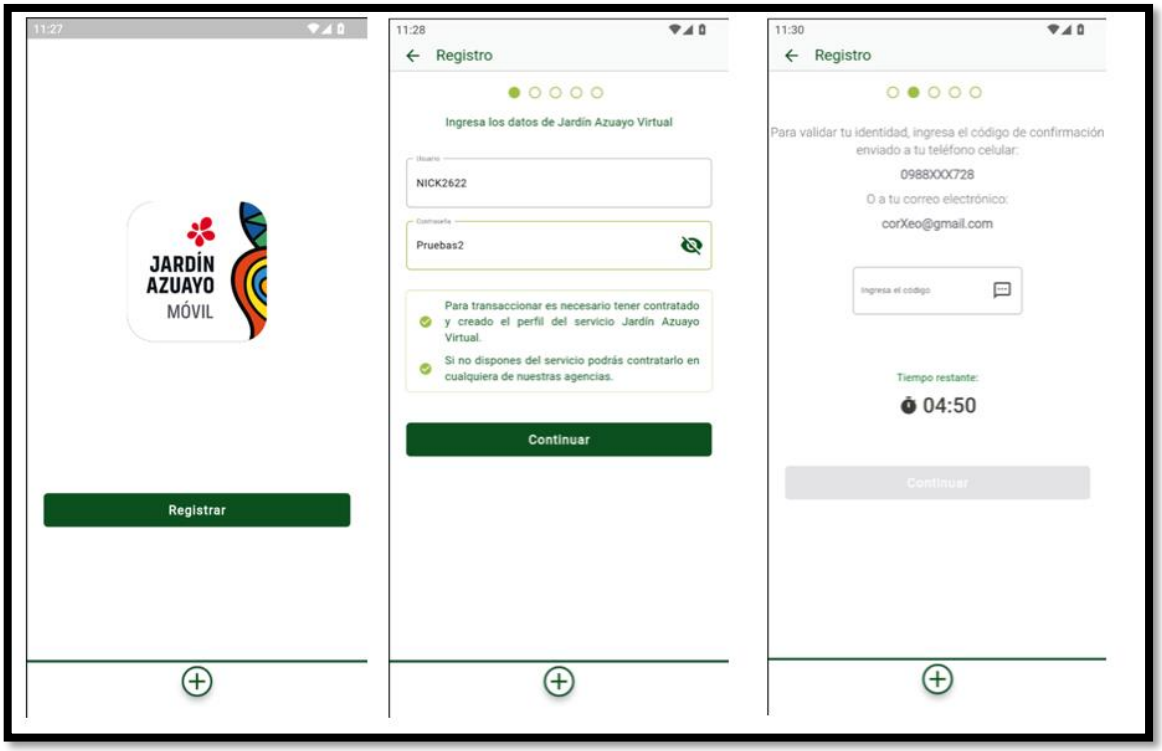
Anexo 9 Captura de componentes que interactúan en la pantalla de selección para realización de la transferencia (Cuenta Origen, Cuenta Destino, Valor de transferencia)



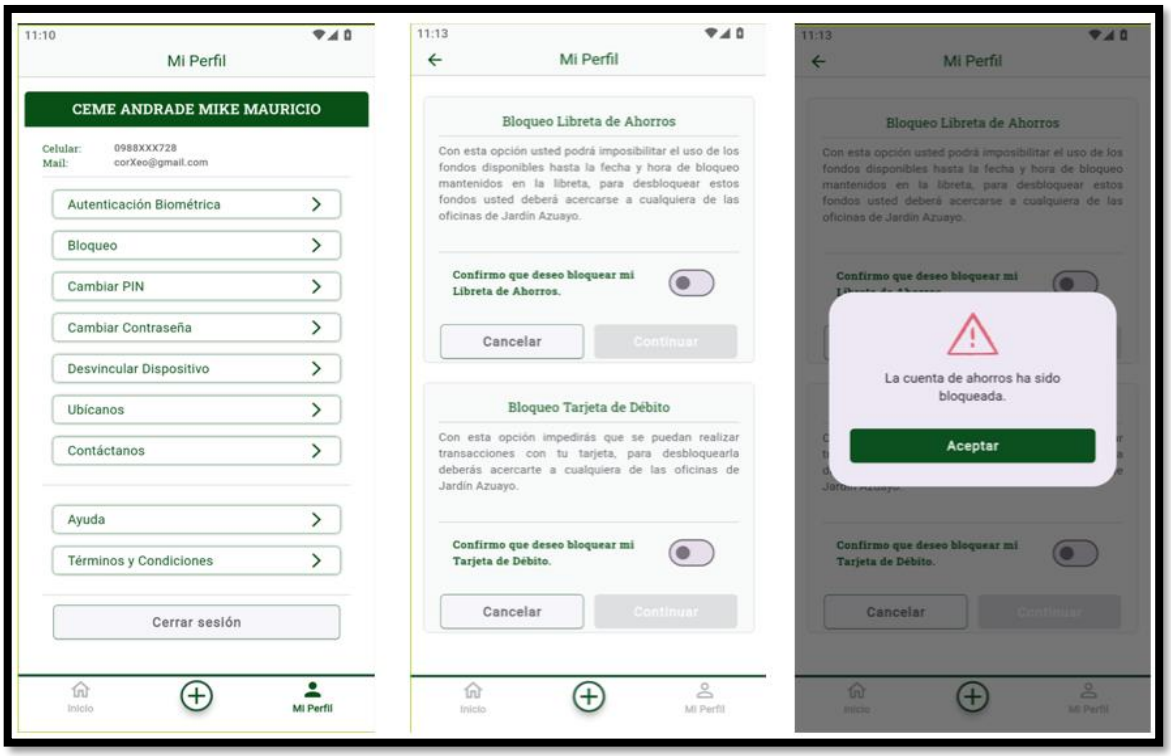
Anexo 10 Captura de componentes que interactúan en la pantalla de ingreso de OTP para realización de la transferencia



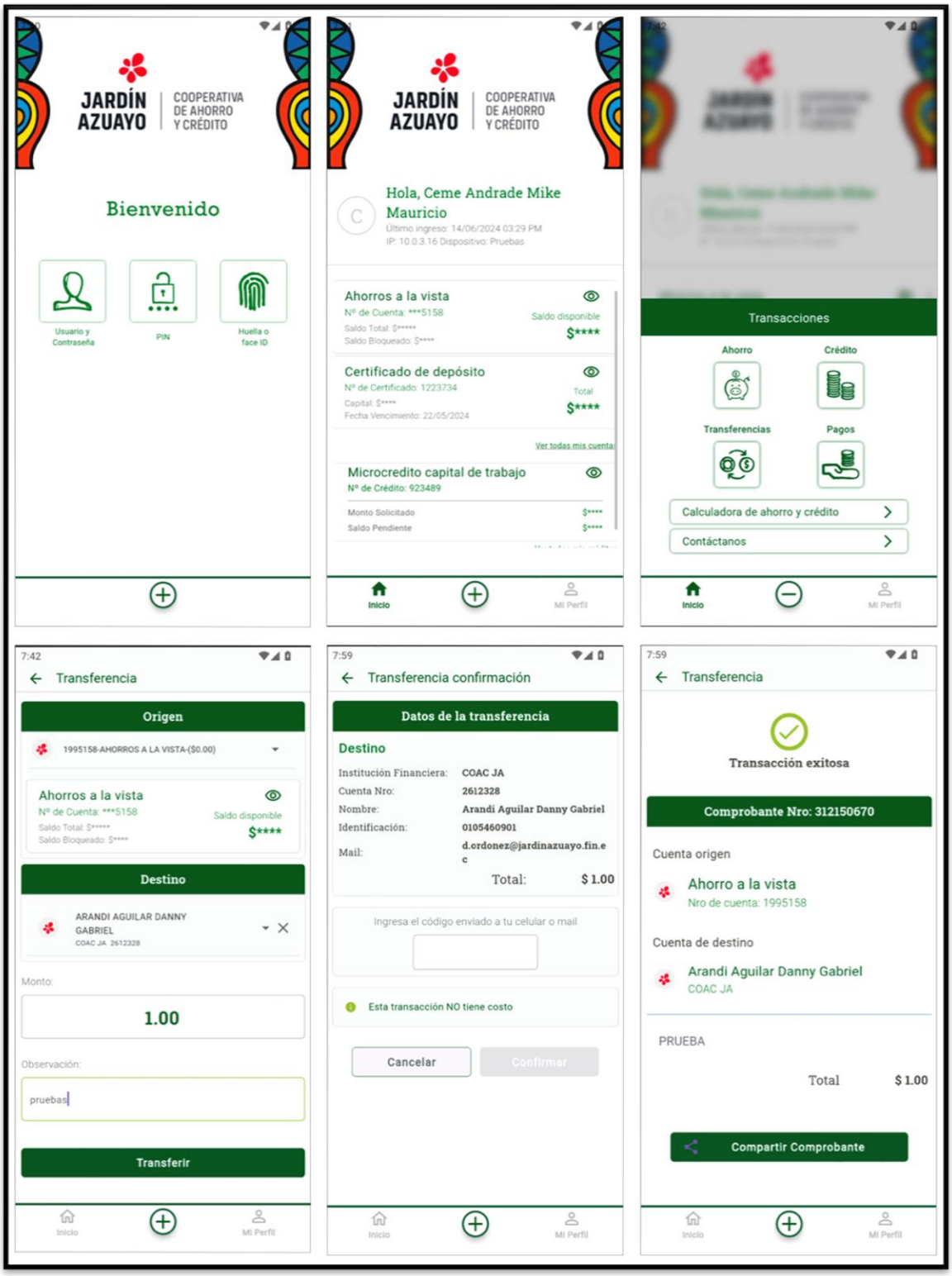
Anexo 11 Flujo de pantallas para realizar el registro de un usuario en el aplicativo



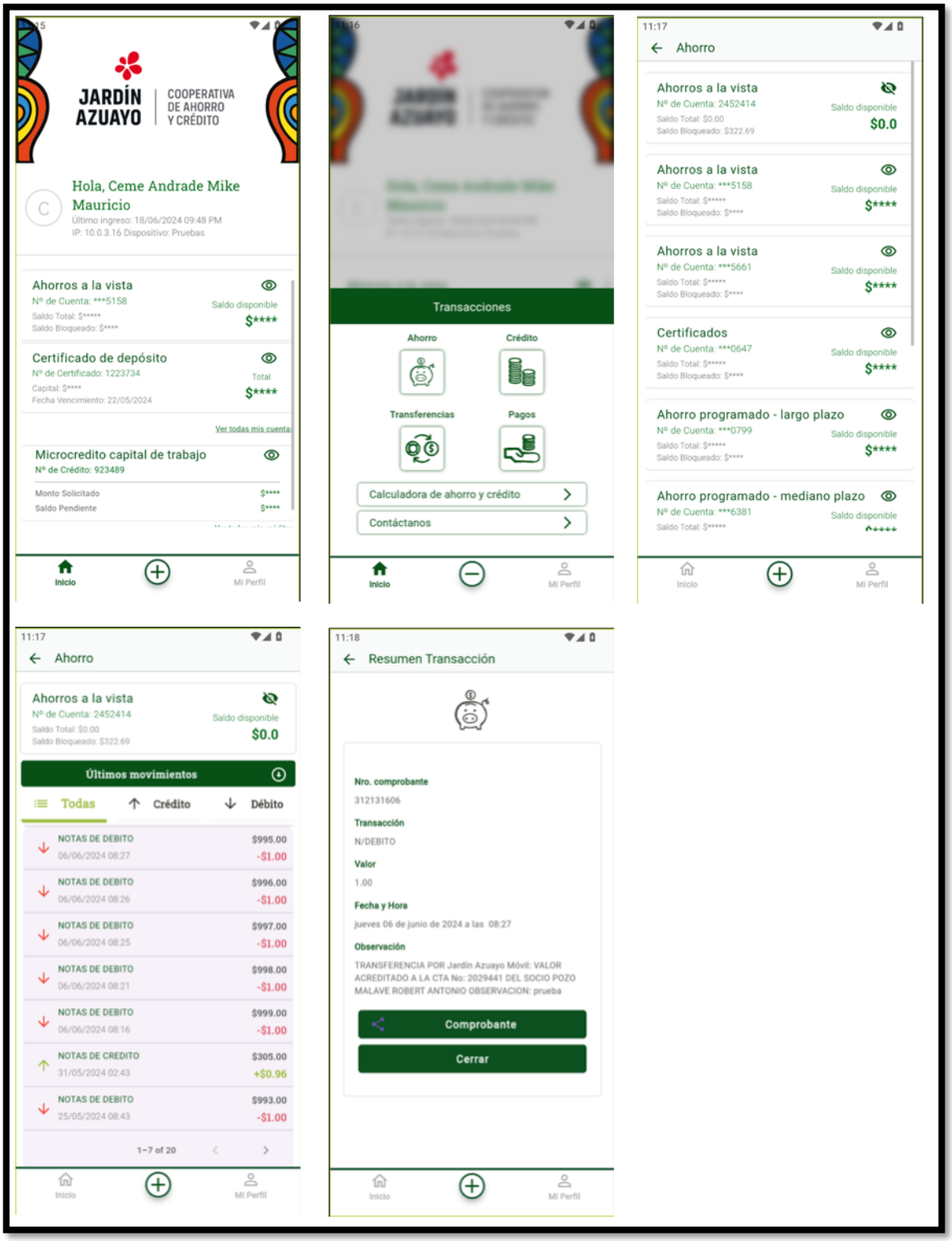
Anexo 12 Flujos de pantallas para realizar el bloqueo de una libreta



Anexo 13 Flujo de pantallas para realizar el ingreso al aplicativo y secuencia para realizar una transferencia entre cuentas.



Anexo 14 Flujo de pantallas para realizar la revisión de estados de cuentas del usuario.



Anexo 15 Captura de lenguaje de programación que identifica las propiedades del emulador en el que va a ejecutar las pruebas

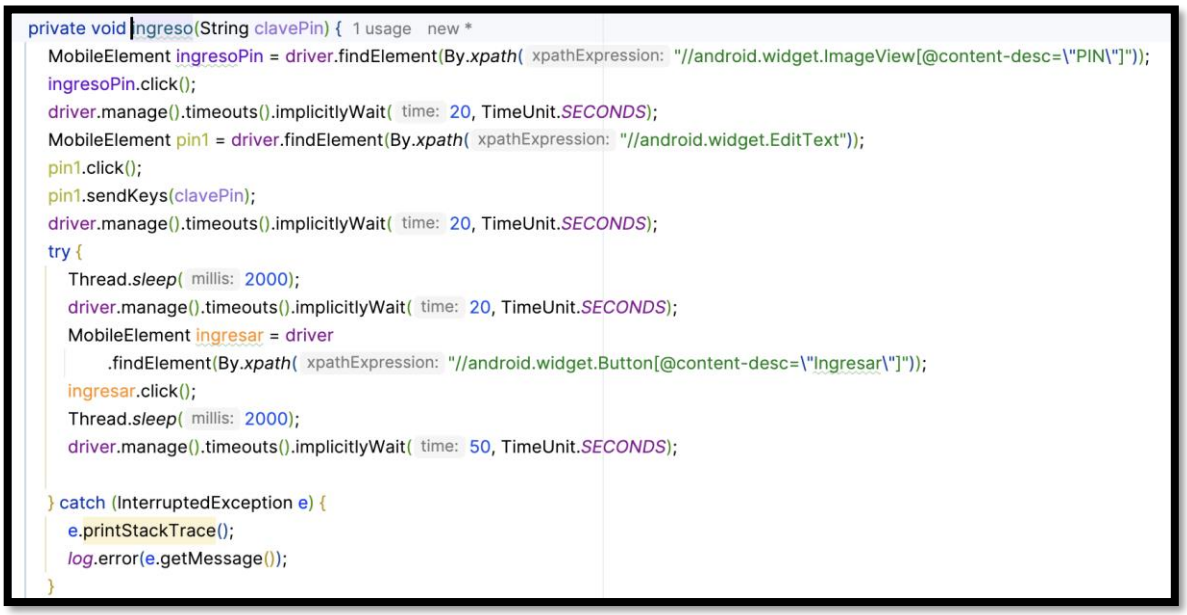
```
private void agregaParametros() throws Exception {  1 usage   s.corderoc *
    try {
        DesiredCapabilities capabilities = new DesiredCapabilities();
        capabilities.setCapability( capabilityName: "platformName", value: "Android");
        capabilities.setCapability( capabilityName: "appium:deviceName", value: "Galaxy S20 FE 5g");
        capabilities.setCapability( capabilityName: "appium:udid", value: "127.0.0.1:6555");
        capabilities.setCapability( capabilityName: "appium:platformVersion", value: "13");
        capabilities.setCapability( capabilityName: "appium:appPackage", value: "app.jardinazuayo.fin.ec.jardinazuayoapp");
        capabilities.setCapability( capabilityName: "appium:appActivity", value: "app.jardinazuayo.fin.ec.jardinazuayoapp.MainActivity");
        capabilities.setCapability( capabilityName: "appium:ensureWebviewsHavePages", value: "true");
        capabilities.setCapability( capabilityName: "appium:nativeWebScreenshot", value: "true");
        capabilities.setCapability( capabilityName: "appium:connectHardwareKeyboard", value: true);
        URL appiumServerURL = new URL( spec: "http://127.0.0.1:4723/wd/hub");
        driver = new AndroidDriver<>(appiumServerURL, capabilities);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
    } catch (Exception e) {
        log.error("No se pudo conectar al servidor" + e.getMessage());

    }
}
```

Anexo 16 Captura de lenguaje de programación que indica el proceso de registro de un usuario.

```
private void registro(String nick, String clave, String otpRegistro, String nombreEquipo) {  1 usage   s.corderoc *
    try {
        MobileElement registroNombre = driver.findElementByXPath( using: "//android.widget.EditText");
        String texto = registroNombre.getText();
        if (registroNombre.isSelected()) {
            registroNombre.click();
            registroNombre.isSelected();
            registroNombre.sendKeys( ...keysToSend: nombreEquipo);
            driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        }
    } catch (NoSuchElementException e) {
        log.info("Error en el proceso de Registro" + e.getMessage());
    }
    log.info("Ingreso a registro");
    MobileElement btonRegistro = driver.findElementByXPath( using: "//android.widget.Button[@content-desc=\"Registrar\"]");
    btonRegistro.click();
    driver.manage().timeouts().implicitlyWait( time: 50, TimeUnit.SECONDS);
    MobileElement txtNombreUsuario = driver
            .findElement(By.xpath( xpathExpression: "//android.widget.ScrollView/android.widget.EditText[1]"));
    txtNombreUsuario.click();
    txtNombreUsuario.sendKeys(nick);
    MobileElement txtClaveUsuario = driver
            .findElement(By.xpath( xpathExpression: "//android.widget.ScrollView/android.widget.EditText[2]"));
    txtClaveUsuario.click();
    txtClaveUsuario.sendKeys(clave);
    driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
```

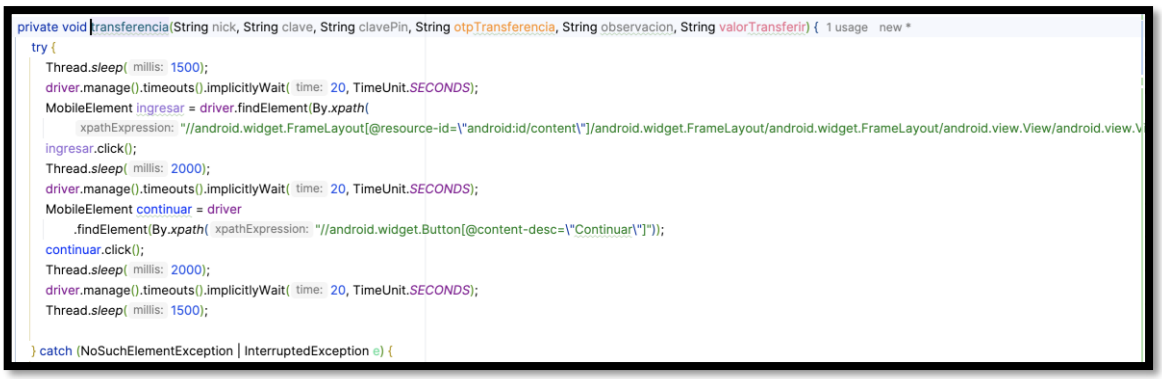
Anexo 17 Captura de lenguaje de programación que indica el proceso de ingreso al aplicativo



```
private void ingreso(String clavePin) { 1 usage  new *
    MobileElement ingresoPin = driver.findElement(By.xpath( xpathExpression: "//android.widget.ImageView[@content-desc=\"PIN\"]"));
    ingresoPin.click();
    driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
    MobileElement pin1 = driver.findElement(By.xpath( xpathExpression: "//android.widget.EditText"));
    pin1.click();
    pin1.sendKeys(clavePin);
    driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
    try {
        Thread.sleep( millis: 2000);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        MobileElement ingresar = driver
                .findElement(By.xpath( xpathExpression: "//android.widget.Button[@content-desc=\"Ingresar\"]"));
        ingresar.click();
        Thread.sleep( millis: 2000);
        driver.manage().timeouts().implicitlyWait( time: 50, TimeUnit.SECONDS);

    } catch (InterruptedException e) {
        e.printStackTrace();
        log.error(e.getMessage());
    }
```

Anexo 18 Captura de lenguaje de programación que indica el proceso de registro de una transferencia en el aplicativo



```
private void transferencia(String nick, String clave, String clavePin, String otpTransferencia, String observacion, String valorTransferir) { 1 usage  new *
    try {
        Thread.sleep( millis: 1500);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        MobileElement ingresar = driver.findElement(By.xpath(
                xpathExpression: "//android.widget.FrameLayout[@resource-id=\"android:id/content\"]/android.widget.FrameLayout/android.widget.FrameLayout/android.view.View/android.view.V
        ingresar.click();
        Thread.sleep( millis: 2000);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        MobileElement continuar = driver
                .findElement(By.xpath( xpathExpression: "//android.widget.Button[@content-desc=\"Continuar\"]"));
        continuar.click();
        Thread.sleep( millis: 2000);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 1500);

    } catch (NoSuchElementException | InterruptedException e) {
```

Anexo 19 Captura de lenguaje de programación que indica que se escoge la opción de transferencia

```
void transferir() {  1 usage   new *
    try {
        MobileElement transferir = driver.findElement(By.xpath( xpathExpression: "//android.widget.Button"));
        transferir.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        MobileElement cuenta = driver
                .findElement(By.xpath( xpathExpression: "//android.widget.ImageView[@content-desc=\"Transferencias\"]"));
        cuenta.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
        log.error("Error al transferir" + e.getMessage();
    }
}
```

Anexo 20 Captura de lenguaje de programación que indica que se selecciona la cuenta origen de la transferencia

```
void chooseOriginAccount() {  1 usage   new *
    try {
        MobileElement cuenta = driver.findElement(By.xpath( xpathExpression: "//android.widget.Button[@content-desc=\"Selecciona una cuenta\"]"));
        cuenta.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        List<MobileElement> options = driver.findElements(By.xpath( xpathExpression: "//android.widget.ImageView"));
        if (!options.isEmpty()) {
            options.get(0).click();
        }
    } catch (Exception e) {
        e.printStackTrace();
        log.error("Error al seleccionar cuenta origen" + e.getMessage();
    }
}
```

Anexo 21 Captura de lenguaje de programación que indica que se selecciona la cuenta destino de la transferencia

```
void chooseDestinationAccount() {  1 usage   new *
    try {
        MobileElement cuenta = driver.findElement(By.xpath( xpathExpression: "//android.view.View[@content-desc=\"Selecciona un Beneficiario.\"]"));
        cuenta.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 2000);
        List<MobileElement> options = driver.findElements(By.xpath( xpathExpression: "//android.widget.EditText"));
        if (!options.isEmpty()) {
            options.get(0).click();
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
        log.error("Error al seleccionar cuenta destino" + e.getMessage());
    }
}
```

Anexo 22 Captura de lenguaje de programación que indica la colocación de los valores a transferir

```
void colocaValores(String valorTransferir, String otpTransferencia) { 1 usage  new *
    try {
        MobileElement valor = driver.findElement(By.xpath( xpathExpression: "//android.widget.ScrollView/android.widget.EditText[1]"));
        valor.click();
        valor.sendKeys(valorTransferir);
        Thread.sleep( millis: 5000);
        MobileElement observacion = driver.findElement(By.xpath( xpathExpression: "//android.widget.ScrollView/android.widget.EditText[2]"));
        observacion.click();
        observacion.sendKeys(otpTransferencia);
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 1500);
    } catch (InterruptedException e) {
        e.printStackTrace();
        log.error("Error al colocar valores" + e.getMessage());
    }
}
```

Anexo 23 Captura de lenguaje de programación que indica la ejecución la transferencia

```
void btnTransferir() { 1 usage  new *
    try {
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        MobileElement transferir = driver.findElement(By.xpath( xpathExpression: "//android.widget.Button[@content-desc=\"Transferir\"]"));
        transferir.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 1500);
        MobileElement otp = driver.findElement(By.xpath( xpathExpression: "//android.widget.EditText"));
        otp.click();
        otp.sendKeys( ...keysToSend: "123456");
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 1500);
        MobileElement confirmaTransferencia = driver.findElement(By.xpath( xpathExpression: "//android.widget.Button[@content-desc=\"Confirmar\"]"));
        confirmaTransferencia.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 3000);
        MobileElement confirmar = driver.findElement(By.xpath( xpathExpression: "//android.view.View[@content-desc=\"Inicio\"]"));
        confirmar.click();
        driver.manage().timeouts().implicitlyWait( time: 20, TimeUnit.SECONDS);
        Thread.sleep( millis: 3000);
    } catch (InterruptedException e) {
        e.printStackTrace();
        log.error("Error al transferir" + e.getMessage());
    }
}
```