UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

GARANTIZANDO LA CONSISTENCIA: IMPLEMENTACIÓN DE BROWSERSTACK PARA PRUEBAS END-TO-END EN ADA+

MARC CABERO ARMENGOL

Director/a
FEDERICO PÉREZ MARINA (PUNTO-FA, SL)

Ponente: MATÍAS SEBASTIÁN MARTÍNEZ MARTÍNEZ (Departamento de Ingenieria de Servicios y Sistemas de Información)

Titulación
Grado en Ingeniería Informática (Ingeniería del Software)

Memoria del trabajo de fin de grado

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

27/06/2024

Resumen

Este proyecto se centra en la implementación de pruebas automatizadas end-to-end para ADA+, la aplicación de retail de moda utilizada en las tiendas Mango. Las pruebas cubren varias funcionalidades clave de ADA+, incluyendo inicio de sesión, búsqueda de productos, modificación de productos en exhibición, cambio de idioma y configuración de usuario. El objetivo es garantizar que ADA+ funciona correctamente ante cualquier cambio, proporcionando una capa de seguridad adicional antes del despliegue. Los archivos de prueba están organizados en paquetes separados para cada funcionalidad y utilizan un enfoque basado en robots para simular las interacciones del usuario. Este proyecto está diseñado para ser fácilmente adaptable y útil para los desarrolladores de Android. Las pruebas se realizan utilizando la herramienta de integración continua Jenkins y la plataforma de pruebas en la nube BrowserStack.

Resum

Aquest projecte es centra en la implementació de proves automatitzades end-to-end per a ADA+, l'aplicació de retail de moda utilitzada a les botigues Mango. Les proves cobreixen diverses funcionalitats clau de ADA+, incloent l'inici de sessió, la cerca de productes, la modificació de productes en exhibició, el canvi d'idioma i la configuració d'usuari. L'objectiu és garantir que ADA+ funciona correctament davant qualsevol canvi, proporcionant una capa de seguretat addicional abans del desplegament. Els arxius de prova estan organitzats en paquets separats per a cada funcionalitat i utilitzen un enfocament basat en robots per a simular les interaccions de l'usuari. Aquest projecte està dissenyat per ser fàcilment adaptable i útil per als desenvolupadors d'Android. Les proves es realitzen utilitzant l'eina d'integració contínua Jenkins i la plataforma de proves en el núvol BrowserStack.

Abstract

This project focuses on the implementation of end-to-end automated tests for ADA+, the fashion retail application used in Mango stores. The tests cover several key functionalities of ADA+, including login, product search, modification of products on display, language change, and user settings. The aim is to ensure that ADA+ functions correctly in the face of any changes, providing an additional layer of security before deployment. The test files are organized into separate packages for each functionality and use a robot-based approach to simulate user interactions. This project is designed to be easily adaptable and useful for Android developers. The tests are performed using the Jenkins continuous integration tool and the BrowserStack cloud testing platform.

Índice

1. Contexto....7
1.1 Introducción....7
1.2 Identificación del problema....8
1.3 Definiciones....9
1.4 Stakeholders....9
2. Justificación....11
2.1 Soluciones existentes / competidores....11
3. Alcance y obstáculos....12
3.1 Alcance: Objetivos y subobjetivos....12
3.2 Obstáculos....13
3.3 Riesgos....13
4. Diseño....14
4.1 Arquitectura....14
4.1.1 Tecnologías usadas....15
4.1.1 Funcionalidades testeadas de ADA+....16
4.1.2 Framework de BrowserStack....21
4.2 Modelo arquitectura física....22
4.2.1 local....22
4.2.2 ordenador-browser....22
4.2.3 job jenkins-browser....23
4.2.4 pull request develop....24
4.3 Modelo UML....25
4.3.1 Diagramas uml con patrón Robot....25
4.3.2 Diagramas de secuencia de los Test sobre BrowserStack....28
5. Metodología y herramientas de seguimiento....41
5.1 Metodología....41
5.2 Seguimiento....42
5.3 Cambios de metodología....42
6. Descripción de las tareas....43
6.1 Introducción....43
6.2 Tareas....43
6.2.1 Tareas de Documentación....43
6.2.2 Tareas de Planificación....44
6.2.3 Tareas de Desarrollo....44
6.3 Recursos....47
7. Estimaciones y Gantt....48
8. Ejecución del proyecto....51
8.1 Reuniones semanales....51
8.2 Retrospectiva de los sprints....51
8.2.1 Retrospectiva del primer sprint....51
8.2.2 Retrospectiva del segundo sprint....52
8.2.3 Retrospectiva del tercer sprint....53
8.3 Línea temporal real....54
9. Integración de los conocimientos....55
10. Legislación y cumplimiento regulatorio....56
10.1 Marco legal....56

10.2 Cumplimiento regulatorio.... 56
10.3 Cumplimiento regulatorio.... 56
10.4 Responsabilidad y obligaciones.... 57
11. Conocimientos técnicos del proyecto.... 57
12. Ejecución práctica del proyecto.... 58
12.1 Ejecución local.... 58
12.2 Ejecución en browserstack.... 58
12.3 Ejecución desde job de jenkins.... 58
12.4 Ejecución desde bitbucket.... 58
13. Implementación de los test end to end.... 59
13.1 Calidad de código.... 59
13.1 Jenkins.... 61
13.2 Tests sobre funcionalidades.... 62
13.2.1 LogIn.... 63
13.2.2 SearchBar and SearchTopics.... 64
13.2.3 Settings.... 69
13.2.4 Product.... 71
13.2.5 Cambiar Idioma.... 73
13.2.6 Procesos.... 74
14. Gestión de riesgo: Planes alternativos y obstáculos.... 76
15. Presupuesto.... 77
15.1 Identificación de los costes.... 77
15.1.1 Recursos humanos.... 77
15.1.2 Hardware.... 80
15.1.3 Software.... 81
15.1.4 Gastos generales.... 82
15.2 Estimación de los costes.... 83
15.3 Control de gestión.... 84
16. Informe de sostenibilidad.... 85
16.1 Autoevaluación.... 85
16.2 Dimensión Económica.... 85
16.3 Dimensión Ambiental.... 86
16.4 Dimensión Social.... 87
17. Puesta a prueba.... 88
17.1 Contexto.... 88
17.2 Primera puesta a prueba.... 89
17.3 Extracción de resultados.... 90
18. Desviación Real.... 92
19. Conclusiones y trabajo futuro.... 93
19.1 Competencias técnicas trabajadas.... 93
19.2 Relación del TFG con el grado y la especialidad.... 93
19.3 Conclusiones personales.... 94
19.4 Trabajo futuro.... 95
Glosario.... 96
Bibliografía.... 100

Índice de figuras y tablas

Figura 4.1.1.1: Login y menú principal de ADA+. Fuente: Propia.....14
Figura 4.1.1.2: Proceso de confirmación de cajas en ADA+. Fuente: Propia.....15
Figura 4.1.1.3: Proceso de inventarios de tiendas en ADA+. Fuente: Propia.....16
Figura 4.1.1.4: Proceso de traspasos de cajas en ADA+. Fuente: Propia.....16
Figura 4.1.1.7: Compra de prendas para clientes en ADA+. Fuente: Propia.....18
Figura 4.1.2.1 BrowserStack App Live. Fuente: Propia..... 19
Figura 4.1.2.2: BrowserStack App Automate. Fuente: Propia..... 19
Figura 4.2.1.1: Ejecución Local. Fuente: Propia..... 20
Figura 4.2.2.1:Ejecución a BrowserStack. Fuente: Propia.....21
Figura 4.2.3.1: Ejecución Jenkins. Fuente: Propia.....21
Figura 4.2.4.1:Ejecución al generar una pull request a Develop. Fuente: Propia.....22
Figura 4.3.1 Diagrama de UML de los Test. Fuente: Propia.....23
Figura 4.3.1.1 Diagrama uml con el patrón Robot..... 25
Figura 4.3.2.1: Diagrama de secuencia del test Cambiar Idioma. Fuente: Propia.....26
Figura 4.3.2.2: Diagrama de secuencia del test Acceder a una caja no confirmada. Fuente: Propia..... 27
Figura 4.3.2.3: Diagrama de secuencia del test Empezar un Inventario creado. Fuente: Propia.....28
Figura 4.3.2.4: Diagrama de secuencia del test Buscar ropa de Mujer por número de referencia. Fuente: Propia.....29
Figura 7.1: Diagrama de Gantt para las tareas propuestas para el desarrollo del proyecto (obviando las reuniones). Fuente: Propia.....48
Figura 8.3.1: Ejecución real en un diagrama de Gantt de las tareas a desarrollar del proyecto (obviando las reuniones). Fuente: Propia..... 52
Figura 13.1.1: Captura del archivo de configuración de Job de Jenkins. Fuente: Propia.....59
Figura 13.2.1.1: Captura del archivo LogInTest. Fuente: Propia.....61
Figura 13.2.1.2 Captura del archivo LogInElements. Fuente: Propia..... 61
Figura 13.2.2.1: Capturas del archivo SearchBarTests. Fuente: Propia.....64
Figura 13.2.2.3: Capturas del archivo SearchRobots. Fuente: Propia.....66
Figura 13.2.3.1: Captura del archivo SettingsAndHelpTests. Fuente: Propia.....67
Figura 13.2.3.2: Captura del archivo SettingsAndHelpElements. Fuente: Propia..... 67
Figura 13.2.3.3: Captura del archivo SettingsAndHelpRobots. Fuente: Propia.....68
Figura 13.2.4.1: Captura del archivo ProductTests. Fuente: Propia..... 69
Figura 13.2.4.3: Capturas del archivo ProductRobots. Fuente: Propia.....71
Tabla 13.2.4.3: Tabla con la representación en porcentaje cuantas funcionalidades están testeadas. Fuente: Propia.....72
Tabla 15.1.1.1: Tabla con los costes estimados por rol y hora. Fuente: Elaboración propia..... 75
Tabla 15.1.1.2: Tabla con los costes estimados del proyecto por tarea y rol. Fuente: Elaboración propia.. 77
Tabla 15.1.2.1: Tabla con los costes de los productos Hardware. Fuente: Elaboración propia.....78
Tabla 15.1.3.1: Tabla con los costes de los productos Software. Fuente: Elaboración propia..... 79
Tabla 15.5: Tabla con los costes generales. Fuente: Elaboración propia..... 80
Tabla 15.2.1: Tabla con los costes para imprevistos. Fuente: Elaboración propia.....81
Tabla 15.2.2: Tabla con los costes estimados del proyecto. Fuente: Elaboración propia.....81
Tabla 17.1.1: Tabla con incidencias desde 25 julio a 23 diciembre. Fuente: Documento Resumen de Sprint.....86
Tabla 17.1.2: Tabla con incidencias desde 23 octubre a 24 febrero. Fuente: Documento Resumen de Sprint.....86

Tabla 17.1.3: Tabla con incidencias desde 23 Octubre a 24 abril. Fuente: Documento Resumen de Sprint.....87
Figura 17.2.1: Resultado de la ejecución de tests. Fuente: BrowserStack app automate.....87
Tabla 17.2.2: Tabla con incidencias desde 23 Octubre a 24 abril. Fuente: Documento Resumen de Sprint.....88
Figura 17.2.3: Figura con incidencias y tiendas abiertas desde el enero del 2023. Fuente: Documento Resumen de Sprint..... 88

1. Contexto

1.1 Introducción

En el mundo actual, en constante evolución hacia la digitalización, la calidad del software toma un papel vital en:

- La demanda de los clientes y el cumplimiento de las expectativas.
- Garantizar un ahorro en costes y solucionar errores antes de que estos puedan llegar a ser más problemáticos.
- Para cualquier compañía disponer de un departamento de QA supone tener rentabilidad y crecimiento.
- La capacidad de proporcionar aplicaciones y servicios en línea que sean fiables, seguros y funcionales resulta crítico para mantener la satisfacción del cliente y la competitividad en el mercado. [1]

En base a lo anterior, mencionado en el artículo de Inlogic, las plataformas de pruebas y los entornos de testing han adquirido una relevancia protagonista, ya que han permitido a los programadores evaluar el comportamiento y rendimiento de sus aplicaciones en tantos dispositivos, plataformas o sistemas operativos como se quiera.

Este trabajo de final de grado se basa en explorar el efecto que pueden llegar a causar estos test en un proyecto desarrollado sobre BrowserStack, una plataforma de pruebas de software basada en la nube. BrowserStack proporciona a los desarrolladores un entorno de prueba robusto y escalable que les permite realizar pruebas automatizadas y manuales en una amplia variedad de dispositivos, sistemas operativos y navegadores web reales.

Estos tipos de plataforma se han convertido en una necesidad para equipos de desarrollo de software de todos los ámbitos, facilitando la detección temprana de errores, la controlación de errores habituales, la optimización del rendimiento y la mejora de la experiencia del usuario final.

Para definir el marco de este proyecto, se analizará BrowserStack como plataforma para diseñar test automatizados *end to end*. Empezando por que plataforma usar y que pros y contras debemos tener en cuenta una vez escogida la plataforma, la configuración del proyecto será otra fase crítica para el éxito de este, ya que si el primer paso ya es en falso el resto será en vano, la siguiente fase serán la definición y creación de test en función de factores como, historias de usuarios concurrentes, aleatoriedad en el orden de la ejecución o otros factores. También consta la metodología de trabajo y las herramientas de seguimiento utilizadas en Mango para llevar a cabo la investigación y el desarrollo de este proyecto.

También mencionar que estos test se elaborarán sobre una aplicación existente de Mango, llamada “ADA +”, esta aplicación es muy importante ya que tiene un impacto directo y muy próximo a las ventas de productos. Esta App de android permite a los trabajadores de tiendas de todas las líneas (Woman, Man, Teen, Kids, Home), tener control absoluto sobre el *stock* que disponen, ejecutar procesos para transferir productos o recibir ventas *online* para entregar a los clientes.

A través de este estudio, se espera proporcionar una visión detallada y completa de los test y proyectos desarrollados sobre BrowserStack, resaltando su importancia en el proceso de desarrollo de *software* y

su contribución a la mejora de la calidad y fiabilidad de las aplicaciones web y móviles en el panorama actual de la tecnología.

1.2 Identificación del problema

El mayor punto a cubrir y más importante para determinar el éxito del proyecto radica en poder garantizar la calidad, el correcto funcionamiento de la aplicación en los dispositivos móviles usados en tienda y versión de sistema operativo, en este caso versión de Android.

Actualmente en Mango para poder asegurar el funcionamiento correcto de ADA+ hay unas reuniones llamadas “Kombats”, que se celebran cada final de *sprint* donde su objetivo es poner a todos los desarrolladores *Frontend* y *Backend,* testeando a mano todos los casos posibles para buscar errores sobre la nueva versión de las nuevas funcionalidades o de las funcionalidades modificadas, y además luego un par de desarrolladores ejecutan a mano más de 130 test a mano probando casuísticas habituales de las tiendas Mango, por lo que de media se suele perder un día laboral entero de todos los desarrolladores, ya que si luego se encuentra que algo no va bien se cancela la subida de la nueva versión a producción

Un gran problema actual es poder asegurar y garantizar el continuo funcionamiento de aplicaciones móviles después de actualizar o modificar el código de este, actualmente se gastan muchos puntos de esfuerzo en ejecutar test de forma manual sencillas y repetitivas.

La identificación de problemas de compatibilidad antes de sacar la nueva versión a tienda es fundamental para evitar tener que repetir tareas en los sprints y evitar posibles bugs o caídas de funcionamiento en la aplicación. Sin embargo, la falta de herramientas adecuadas y enfoques efectivos para realizar pruebas exhaustivas de compatibilidad puede llevar a la liberación de aplicaciones con errores no detectados ni controlados, lo que resulta en una disminución de la satisfacción de los trabajadores de tienda, reducción de ingresos en ventas de tiendas físicas y daños a la reputación de la empresa.

Por lo tanto, este proyecto se propone abordar este problema mediante la exploración y utilización de herramientas avanzadas de prueba. Al comprender mejor las funcionalidades y el potencial de esta plataforma, BrowserStack, se espera mejorar significativamente la eficacia y la eficiencia de las pruebas de compatibilidad, reduciendo así los riesgos asociados con la liberación de aplicaciones defectuosas en el mercado.

Para añadir datos a la identificación del problema, durante los sprints anteriores de media Mango tenía una media de 2.2 incidencias con la aplicación, que significa que cada tres semanas habían dos incidencias relacionadas con la aplicación en las que una funcionalidad o algún sector de la aplicación dejaba de funcionar, gran parte de estas incidencias podrían reducirse drásticamente con test end to end, ya que estos test aseguran el flujo completo y funcionamiento correcto.

Para concluir este apartado, la identificación del problema se basa en la necesidad de implementar un sello más de garantía sobre el funcionamiento de los tests, asegurando así la compatibilidad de versiones de Android, diferentes dispositivos móviles.

1.3 Definiciones

A. ADA+

ADA+ es una aplicación de tienda desarrollada por Mango. Esta aplicación está diseñada para mejorar la eficiencia de las operaciones de la tienda y proporcionar una mejor experiencia de usuario. ADA+ facilita la gestión de inventario, las transacciones de ventas y otras funciones esenciales de la tienda.

B. Browserstack

Browserstack es una plataforma de pruebas en la nube que permite a los desarrolladores probar sus sitios web y aplicaciones móviles en diferentes sistemas operativos y navegadores. Proporciona un entorno de pruebas en tiempo real con acceso a una amplia gama de dispositivos y navegadores.

C. Transfer Request

En el contexto de una tienda, una petición de traspaso es una solicitud para mover mercancía de una ubicación a otra. Esto puede ser entre diferentes tiendas o entre un almacén y una tienda. En ADA+, los empleados pueden usar esta función para solicitar traspasos de inventario según sea necesario.

D. Pala

En el contexto de Mango, "pala" se refiere a un dispositivo externo utilizado en las tiendas para leer las etiquetas RFID de los productos. Este dispositivo se conecta al sistema a través de Bluetooth.

E. Impresora

En este contexto, una impresora se refiere a un dispositivo que se utiliza para imprimir etiquetas de productos en las tiendas Mango. Estas impresoras suelen ser portátiles y se conectan al sistema a través de Bluetooth.

F. Androids

En el contexto de Mango, el equipo Androids de ADA+ se refiere al grupo de desarrolladores y técnicos responsables de mantener y mejorar la aplicación ADA+ para dispositivos Android.

G. Voft

Voft es un término utilizado en Mango para referirse a la opción de comprar online y recoger en tienda física. Esto permite a los clientes comprar productos en la web de Mango y recogerlos en su tienda Mango más cercana, proporcionando una experiencia de compra más flexible y conveniente.

1.4 Stakeholders

Equipo software Engineering: Responsable de desarrollar y mantener la aplicación. Su influencia radica en la implementación efectiva de las estrategias de prueba y la corrección de los problemas identificados. Cómo beneficio de el desarrollo de este proyecto, los developers podrán realizar código de manera más segura, ya que su código será testeado por este proyecto

Usuarios finales de tienda: Aquellos usuarios que usarán la aplicación una vez se publique, en este caso la aplicación de la que hablamos es la usada en tienda por los trabajadores. Como beneficio de este proyecto podrán funcionar prácticamente el 100% del tiempo con la aplicación y sin dar problemas de ningún tipo.

Browserstack y otros proveedores de Herramientas de Prueba: Proveedores de herramientas y servicios de prueba en la nube. Su influencia radica en la eficacia y disponibilidad de las soluciones tecnológicas utilizadas para abordar los problemas de compatibilidad.

Director: Profesor de la UPC que ejercerá como director del proyecto, tiene la responsabilidad de guiar el proyecto y su desarrollo

Ponente: Su influencia radica en la capacidad de proporcionar información valiosa y consejos prácticos que enriquezcan la comprensión y la implementación efectiva de las estrategias de prueba en el proyecto.

Desarrollador (Yo): El desarrollador tendrá la responsabilidad de programar código para investigar y elaborar el proyecto para garantizar el funcionamiento de la aplicación *ADA+*

2. Justificación

Las aplicaciones de hoy en día, cada vez son más exigentes por parte de los usuarios por lo que es lo mismo que decir que cada vez es más importante la calidad y la compatibilidad en el máximo de dispositivos posibles.

La prueba de compatibilidad de una aplicación sobre varios dispositivos móviles es fundamental para garantizar un rendimiento óptimo. Si se genera esta satisfacción en el cliente genera un mejor rendimiento por parte de los trabajadores, facilita procesos costosos y genera mejor reputación sobre la marca.

En este apartado se analizan soluciones ya existentes para el problema anterior, en el que queremos garantizar la calidad del código. En este caso en Mango se tiene acceso a la plataforma de BrowserStack, por lo que se estudiarán otros competidores actuales.

2.1 Soluciones existentes / competidores

En el mercado actual, existen varias soluciones y herramientas diseñadas para abordar los desafíos de prueba de compatibilidad en el desarrollo de software. Entre las soluciones más relevantes se encuentran:

- BrowserStack [2]: Es una web en la nube y plataforma para testear sobre móviles o ordenadores reales, que permite a los desarrolladores testear sus sitios web y sus aplicaciones móviles tanto Android como IOS, a través de navegadores web, sistemas operativos y dispositivos móviles, sin requerir usuarios para instalar o mantener un entorno de desarrollo interno formado por máquinas virtuales, dispositivos y/o emuladores.
  Tienen cuatro productos principales
  - Live
  - App Live
  - Automate
  - App Automate

- Espresso [3]: Espresso está dirigido a desarrolladores, que creen que las pruebas automatizadas son una parte integral del ciclo de vida del desarrollo. Si bien Espresso se puede usar para pruebas de caja negra, quienes están familiarizados con la base de código bajo prueba pueden desbloquear toda su potencia.

- Selenium[4]: Plataforma para crear pruebas y suites de automatización de regresión sólidas basadas en navegador, escalar y distribuir scripts en muchos entornos, entonces desea utilizar Selenium WebDriver, una colección de enlaces específicos de lenguaje para controlar un navegador, como debe ser impulsado.

- Appium[5]: Appium tiene como objetivo admitir la automatización de la interfaz de usuario de muchas plataformas diferentes (móvil, web, escritorio, etc.). No sólo eso, sino que también pretende admitir código de automatización escrito en diferentes lenguajes (JS, Java, Python, etc.).

Esta lista que se nombran las plataformas más grandes que se dedican facilitar proyectos de pruebas de compatibilidad, son parecidos entre ellos pero tienen algunas diferencias que hacen decantarse por una o la otra, lo que destaca la importancia de evaluar cuidadosamente las opciones disponibles y seleccionar la solución más adecuada para las necesidades específicas de cada proyecto.

3. Alcance y obstáculos

3.1 Alcance: Objetivos y subobjetivos

Antes de mencionar los objetivos y subobjetivos de este proyecto quiero explicar cuál es la intención final de este proyecto que puede ser no poder finalizarse en estos meses de desarrollo, además que está sujeto a cambios de requisitos funcionales y no funcionales.
Este proyecto quiere llegar a conseguir ser un sello de garantía para la aplicación ADA+, asegurar el funcionamiento y sin errores. Estos tests tienen largos tiempos de ejecución, pero es lo que conlleva simular de forma real varios escenarios, por lo que para que afecte lo menos posible se quiere ejecutar estos test siempre que en el proyecto de ADA+ se haga una *"Pull Request"* hacia "*Develop"*, de esta manera se puede asegurar que esta rama estará siempre funcionando correctamente

El alcance del proyecto se define mediante la especificación de objetivos principales y subobjetivos que se pretenden conseguir [6]. Los objetivos principales son:

Revisión de las soluciones existentes: Se realizará un estudio en profundidad de las diferentes plataformas de test end to end [7], centrándose en sus características, funcionalidades y facilidad para generar test con varias versiones de Android y varios dispositivos diferentes.

Análisis comparativo de las plataformas de prueba: Se realizará un análisis detallado de las plataformas mencionadas anteriormente. Este análisis permitirá generar una tabla comparativa donde podamos al final tener nuestra elección en función de qué plataforma cumple con los máximos requerimientos

Identificación de recomendaciones para mejorar el proceso de prueba de compatibilidad: Con los hallazgos del análisis comparativo, se elaborarán recomendaciones específicas para mejorar el proceso de prueba de compatibilidad o optimización en el desarrollo de software.
Estas recomendaciones podrían incluir la adopción de nuevas herramientas, la implementación o modificación de prácticas o la optimización de los flujos de trabajo existentes.

Generar un proyecto con test end to end sobre la aplicación ADA +: Este objetivo consiste en la elaboración de test en kotlin en el que se quiere cubrir todas las funcionalidades existentes y las que se vayan añadiendo a lo largo del proyecto, también se considerará que este objetivo se ha cumplido si se el proyecto acaba siendo fácil integrar nuevas test sobres las futuras nuevas funcionalidades.

Los subobjetivos abarcan actividades específicas que contribuirán a alcanzar los objetivos principales, como:

Recopilación de datos sobre las diferentes plataformas de prueba.
Realización de pruebas comparativas utilizando casos de uso reales.
Elaboración de informes detallados que resuman los hallazgos y recomendaciones.

3.2 Obstáculos

Durante la ejecución del proyecto, pueden surgir varios obstáculos que afecten su progreso y resultados [8]. Entre los posibles obstáculos se incluyen:

- Limitaciones de tiempo: Los plazos ajustados pueden dificultar la realización de todas las actividades planificadas dentro del tiempo establecido, además al ser un proyecto nuevo en el departamento puede que sea más difícil de dar los primeros pasos.

- Limitaciones de recursos: La disponibilidad limitada de recursos humanos, financieros o tecnológicos puede afectar la capacidad del equipo para llevar a cabo el proyecto de manera efectiva. Empezando porque este proyecto se va a desarrollar de forma sobre todo individual, eso sí, siempre se va a tener acceso a ayuda.

- Complejidad técnica: La complejidad técnica de algunas plataformas de prueba puede requerir tiempo adicional para comprender y evaluar adecuadamente su funcionamiento.

Además, la variabilidad en los entornos de desarrollo y las diferencias en los requisitos de prueba como idioma de los test, marco de pruebas o plataforma sobre la que emular los test, pueden plantear desafíos adicionales que deben abordarse de manera efectiva para garantizar el éxito del proyecto.

3.3 Riesgos

En este proyecto existen varios riesgos a tener en cuenta para el desarrollo:

- La falta de disponibilidad de datos podría ser un riesgo, puesto que se necesitan datos muy completos o elaborados para generar test precisos en los casos con valores frontera [9], también conocido como la caja negra, o estar informado de todos los tipos de dispositivos y versiones que se usan.

- La resistencia al cambio por parte de los equipos de desarrollo puede dificultar la implementación de recomendaciones o la generación de los mismos test para mejorar el proceso de prueba de compatibilidad y eficiencia. Este es un riesgo a tener en cuenta ya que la idea es dejar estos test a manos de los desarrolladores Android, por lo que tendrán que aprender un nuevo lenguaje de programación.

- La dependencia excesiva de una sola herramienta, este es un riesgo crítico que se tendrá que ver cómo gestionarlo puesto que este proyecto debería poder garantizar siempre el funcionamiento de la aplicación que se quiera testear de prueba puede aumentar el riesgo de fallos si la herramienta experimenta problemas de fiabilidad o no cumple con las expectativas del equipo.

Estos riesgos se deberían intentar gestionar desde un principio para que una vez aparezcan ya tener un plan de mitigación.

4. Diseño

4.1 Arquitectura

La arquitectura de este proyecto de prueba sobre “ADA+” ha sido diseñada utilizando una arquitectura de tres capas con un patrón específico creado por Android, denominado "Robots", que permite una mayor mantenibilidad y facilita la comprensión de las pruebas. A diferencia de un proyecto típico, las capas en este caso son las siguientes:

- Elementos: Esta capa contendrá los elementos que interactúan con la aplicación, como los identificadores de elementos como botones, casillas de verificación, cuadros de texto, imágenes, etc. Esta capa solo debe ser consumida por la capa de Robots para evitar ciclos de dependencia.

- Pruebas: Como su nombre indica, esta capa contiene las pruebas que evalúan si la aplicación reacciona correctamente. Aquí se definen los escenarios de prueba y se verifican los resultados esperados.

- Robots: Esta capa es la encargada de ejecutar las acciones de cada prueba utilizando los elementos de la capa Elementos. Los "Robots" actúan como intermediarios entre las pruebas y los elementos de la interfaz de usuario, permitiendo una mayor modularidad y reutilización de código.

La arquitectura de tres capas con el patrón "Robots" proporciona una estructura clara y eficiente para el desarrollo y la ejecución de pruebas en la aplicación ADA+. Esta organización facilita la mantenibilidad del proyecto y mejora la comprensión de las pruebas realizadas. Además, al separar claramente las responsabilidades entre las capas, se promueve un diseño más escalable y fácil de mantener a medida que el proyecto crece y evoluciona.

Ventajas de usar este diseño:

- Mantenibilidad: La arquitectura de tres capas con el patrón "Robots" facilita la mantenibilidad del proyecto al separar claramente las responsabilidades de cada capa, lo que permite realizar cambios y actualizaciones de manera más eficiente.

- Modularidad: La separación de las capas Elementos, Pruebas y Robots promueve la modularidad del código, lo que facilita la reutilización de componentes y la incorporación de nuevas funcionalidades.

- Claridad en la estructura: El diseño de tres capas con el patrón "Robots" proporciona una estructura clara y comprensible, lo que facilita la comprensión del código y la colaboración entre equipos de desarrollo y pruebas.

Desventajas de usar este diseño:

- Mayor complejidad inicial: La implementación de esta arquitectura puede requerir un mayor esfuerzo inicial para comprender y establecer las relaciones entre las capas, lo que puede resultar en un proceso de desarrollo más lento al principio.

- Posible sobrecarga de abstracción: Si no se gestiona adecuadamente, la introducción de capas adicionales como "Robots" puede llevar a una sobrecarga de abstracción, lo que dificulta la comprensión y el mantenimiento del código.

- Curva de aprendizaje: Los desarrolladores y probadores que no estén familiarizados con este tipo de arquitectura pueden requerir tiempo adicional para comprender y adaptarse a este enfoque, lo que puede afectar la productividad inicialmente.

En resumen, si bien la arquitectura de tres capas con el patrón "Robots" ofrece ventajas en términos de mantenibilidad, modularidad y claridad en la estructura, también puede presentar desafíos relacionados con la complejidad inicial, la posible sobrecarga de abstracción y la curva de aprendizaje para los equipos de desarrollo y pruebas.

4.1.1 Tecnologías usadas

Entre las tecnologías clave utilizadas se encuentran HTTP Request, Kotlin, Appium, JUnit 5 y JSON, cada una desempeñando un papel crucial en la comunicación con servidores remotos, la automatización de pruebas, la ejecución de pruebas unitarias y el intercambio de datos estructurados. A continuación, se detalla el papel de cada una de estas tecnologías en el contexto del proyecto de prueba sobre ADA+.

- HTTP Request: Se emplea para realizar solicitudes y recibir respuestas a través del protocolo HTTP, lo que permite la comunicación con servidores remotos para obtener o enviar datos, como por ejemplo, la interacción con APIs o servicios web, en nuestro caso para crear escenarios concretos sobre la aplicación, conectando con algunos “EndPoints”.

- Kotlin: Es el lenguaje de programación principal utilizado en el desarrollo de la aplicación ADA+. Kotlin es conocido por su concisión, seguridad y facilidad de interoperabilidad con el lenguaje de programación Java, lo que lo hace una elección popular para el desarrollo de aplicaciones Android.

- Appium: Se utiliza como framework de automatización de pruebas para aplicaciones móviles, permitiendo la realización de pruebas funcionales en dispositivos Android e iOS. Appium es conocido por su compatibilidad con múltiples lenguajes de programación, incluyendo Java y Kotlin.

- JUnit 5: Es un framework de pruebas unitarias para Java y Kotlin que proporciona un conjunto de anotaciones y aserciones para la creación y ejecución de pruebas. JUnit 5 es ampliamente utilizado en el desarrollo de aplicaciones Android para garantizar la calidad del código a través de pruebas automatizadas.

- JSON: Se utiliza para el intercambio de datos entre el cliente y el servidor, así como para el almacenamiento de información estructurada. En el contexto de la aplicación ADA+, es probable que se utilice JSON para el envío y recepción de datos entre la aplicación y los servicios web o APIs externas. En este caso se usará para alimentar con datos las HTPP Request hacia backend.

- Maven: Se utiliza como una herramienta de gestión de proyectos y construcción de software en el desarrollo de la aplicación ADA+. Maven simplifica el proceso de construcción, gestión de dependencias y generación de informes, lo que contribuye a una gestión eficiente del ciclo de vida del proyecto.

Estas tecnologías desempeñan un papel fundamental en el desarrollo y las pruebas de la aplicación ADA+, permitiendo la comunicación con servidores remotos, la automatización de pruebas, la ejecución de pruebas unitarias y la manipulación de datos estructurados.

4.1.1 Funcionalidades testeadas de ADA+

A continuación se muestran algunas pantallas de las funcionalidades sobre las que hay test de BrowserStack.

- **Login**

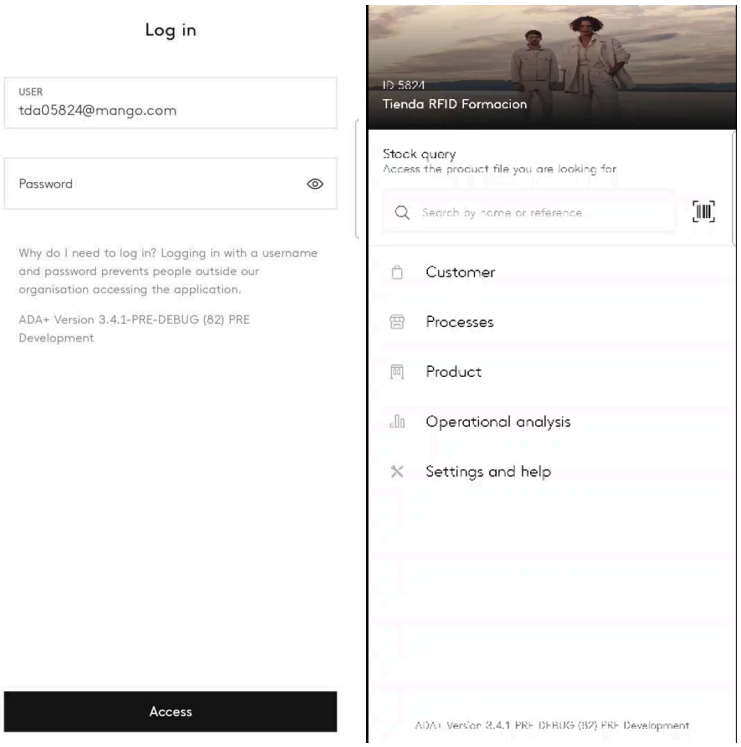


Figura 4.1.1.1: Login y menú principal de ADA+. Fuente: Propia.

- **Procesos**

  - **Confirmaciones**

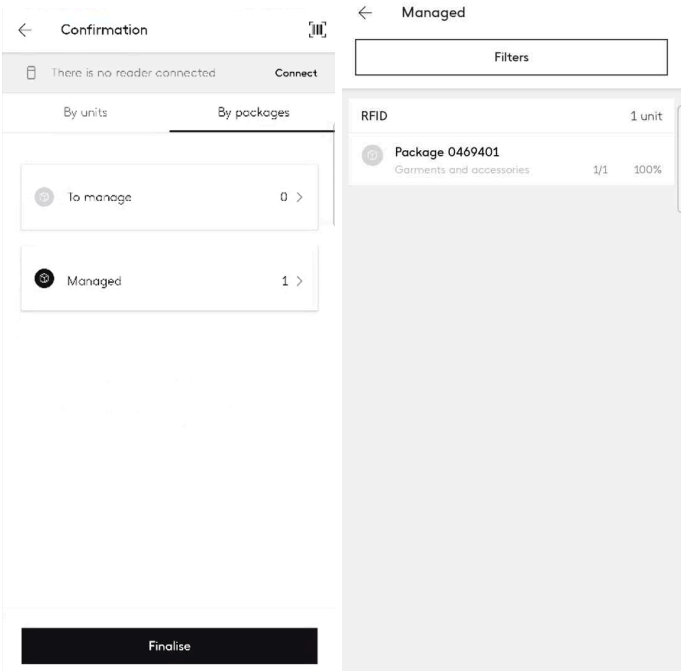


Figura 4.1.1.2: Proceso de confirmación de cajas en ADA+. Fuente: Propia.

  - **Inventario**

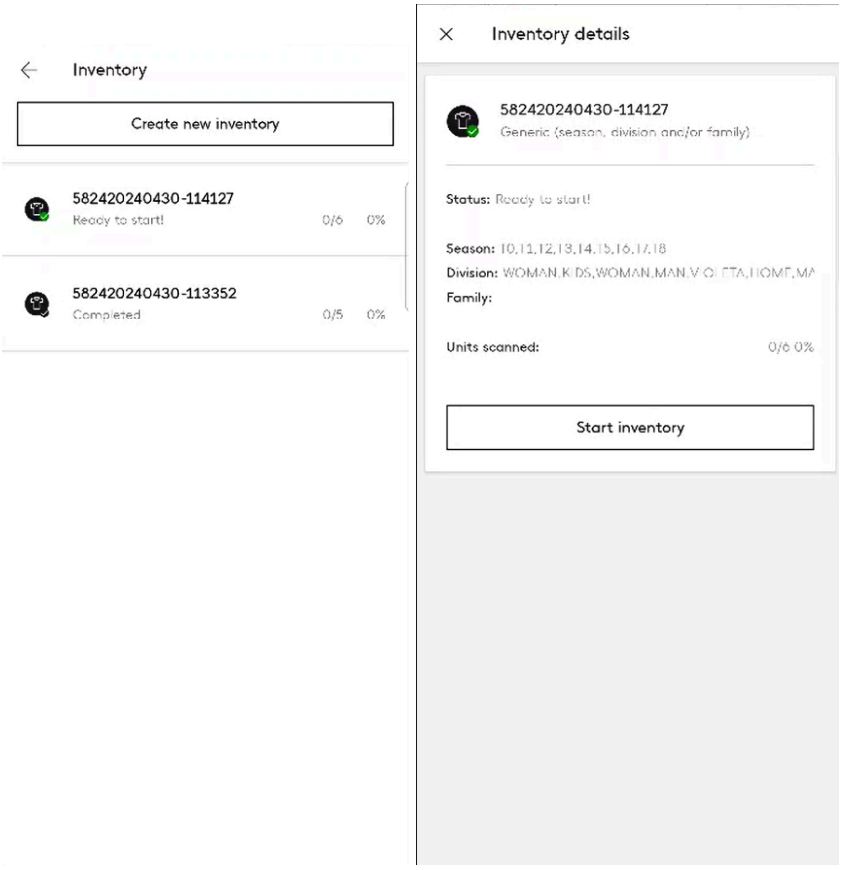


Figura 4.1.1.3: Proceso de inventarios de tiendas en ADA+. Fuente: Propia.

- **Traspasos**

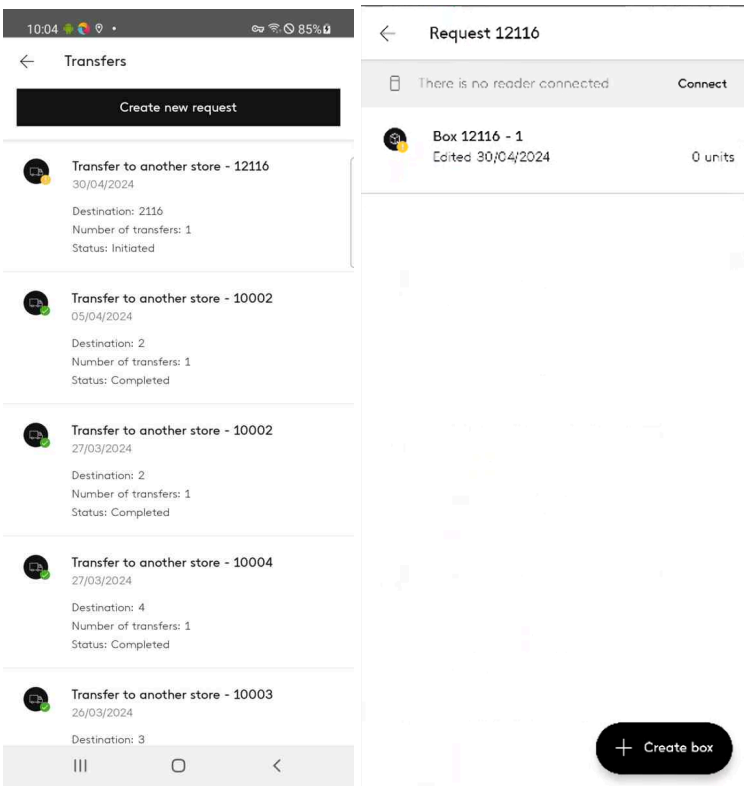


Figura 4.1.1.4: Proceso de traspasos de cajas en ADA+. Fuente: Propia.

- **Buscador**

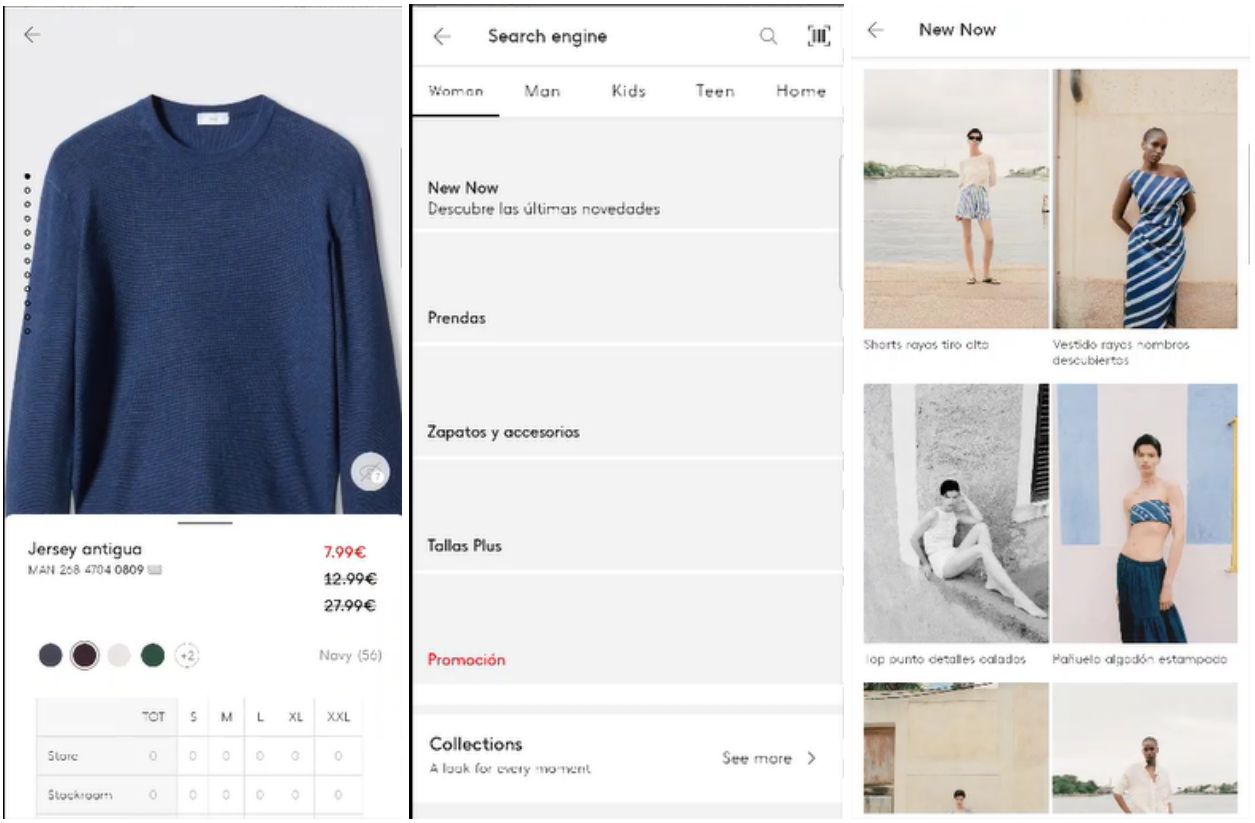


Figura 4.1.1.5: Buscador de prendas o colecciones en ADA+. Fuente: Propia.

- **Ajustes**

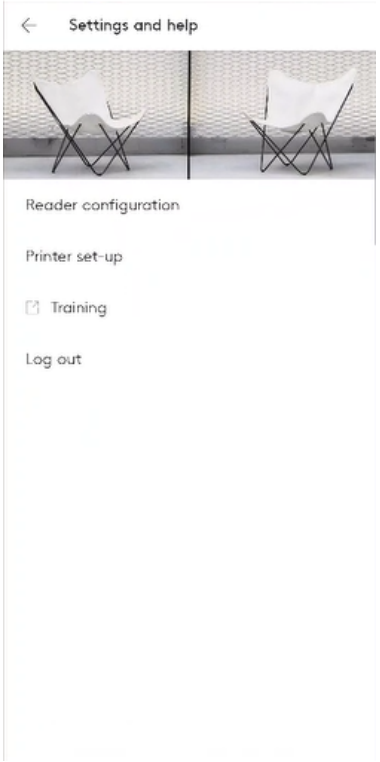
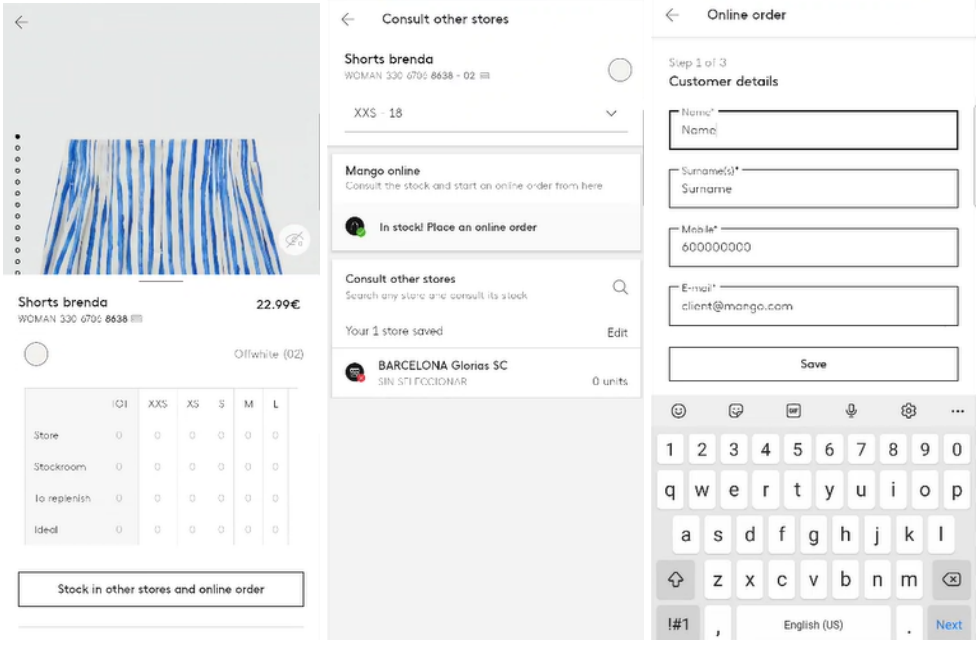


Figura 4.1.1.6: Ajustes en ADA+. Fuente: Propia.

- **Voft**



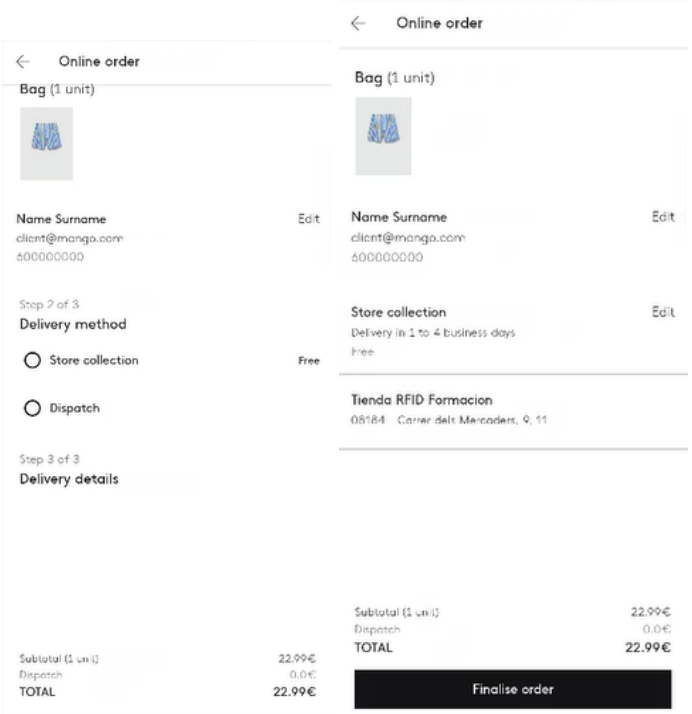


Figura 4.1.1.7: Compra de prendas para clientes en ADA+. Fuente: Propia.

4.1.2 Framework de BrowserStack

Los test se ejecutarán sobre la plataforma de BrowserStack, ya que esta empresa facilita la elaboración de los test *End2End* de forma sencilla.
Esta permite mucha flexibilidad sobre la arquitectura o tecnologías sobre la que se quieran hacer los test.
BrowserStack consiste en una plataforma que permite la ejecución automatizada de test sobre aplicaciones y la ejecución de la aplicación sobre dispositivos reales en línea.

Una de las mayores ventajas que ofrece BrowserStack es que no emulan dispositivos para los test, sino que son reales y esto hace posible poder comprobar mejor el rendimiento de una aplicación.
Ofrece una amplia gama de dispositivos, desde Iphones, Samsungs, Pixels, Oppo, Xiaomi, Huawei… entre otros.

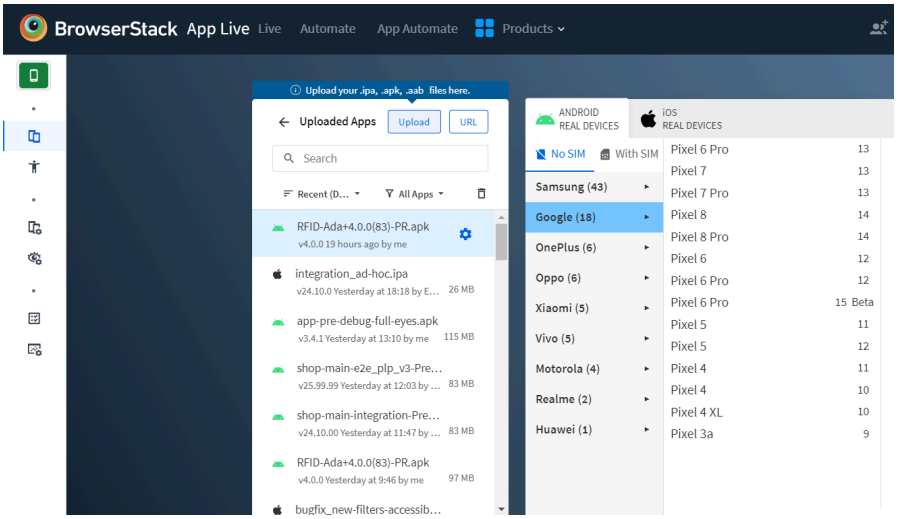


Figura 4.1.2.1 BrowserStack App Live. Fuente: Propia.

Esta funcionalidad, visible de la Figura 4.1.2.1, permite subir cualquier .apk o .ipa, y testear con el dispositivo y/o versión de Android sobre la que se quiera ver como funciona la aplicación.

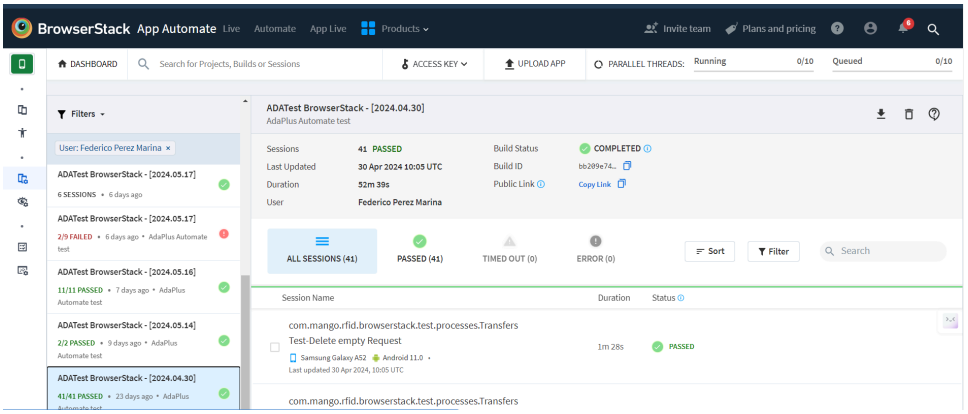


Figura 4.1.2.2: BrowserStack App Automate. Fuente: Propia.

Esta funcionalidad, visible de la Figura 4.1.2.2, permite ejecutar los test del proyecto sobre la plataforma, además permitiendo paralelizar los test, ahorrando muchísimo tiempo, ya que cada test, como son test End2End, todos los test empiezan desde el Login, por lo que la media de tiempo de ejecución de cada test es de 2 minutos, por lo que sí tenemos 40 test, tardan en ejecutarse uno por uno 80 minutos, pero como podemos paralelizar hasta 5 a la vez, la ejecución pasa a tardar 16 minutos.
Cada ejecución de un conjunto de test se guardan en sesiones, que están en la columna de la

izquierda, y dentro de cada sesión podemos observar las ejecuciones o test, desde el resultado de cada uno, video, reportes en texto y la posibilidad de generar un pdf con el resultado o compartir un link público para que cualquier persona que lo tenga, pueda ver los test

4.2 Modelo arquitectura física

A continuación, se explicarán los cuatro tipos de arquitectura física que existen para este proyecto, en función del tipo de ejecución que se realice. Estos son: la ejecución local, la ejecución sobre BrowserStack, la ejecución desde un trabajo de Jenkins, y la generación de una Pull Request (PR) a la rama "develop".

4.2.1 local

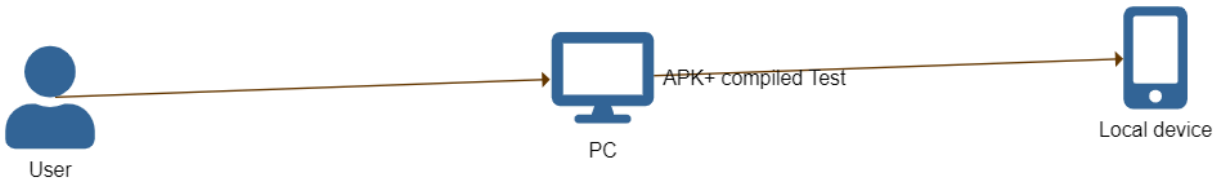


Figura 4.2.1.1: Ejecución Local. Fuente: Propia.

Este tipo de ejecución se realiza directamente desde el ordenador físico. Para ello, es necesario tener el repositorio clonado en el sistema local. El proceso consiste en compilar los tests junto con el archivo APK de la aplicación que se desea probar. Posteriormente, los tests se ejecutan en un dispositivo real que debe estar conectado al ordenador. Este método permite un control directo y una respuesta inmediata de los resultados de las pruebas.

4.2.2 ordenador-browser

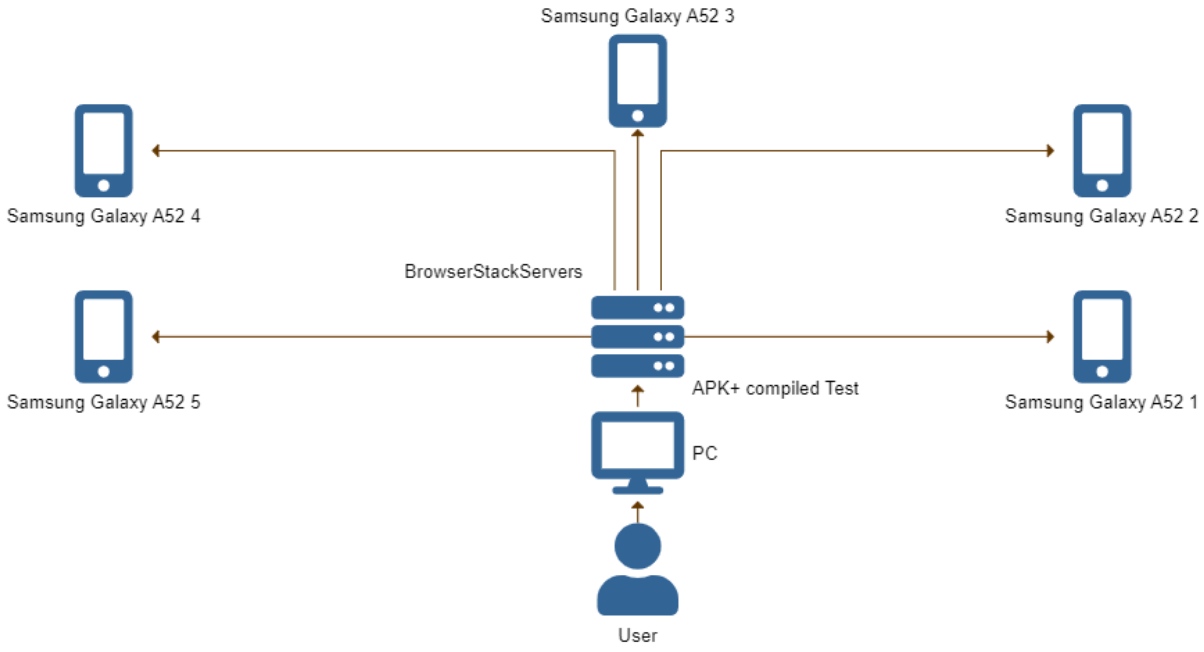


Figura 4.2.2.1:Ejecución a BrowserStack. Fuente: Propia.

En este caso, la ejecución también se realiza desde el ordenador físico y requiere tener el repositorio clonado. Sin embargo, en lugar de utilizar un dispositivo físico, se compilan los tests junto con la ID de la APK de la aplicación que se desea probar, la cual debe estar previamente subida a la plataforma de BrowserStack. Los tests se envían a los servidores de BrowserStack a través de Internet, por lo que es necesario contar con una conexión estable. Los servidores de BrowserStack se encargan de ejecutar los tests sobre la APK, paralelizando los tests en hasta 5 dispositivos simultáneamente.

4.2.3 job jenkins-browser

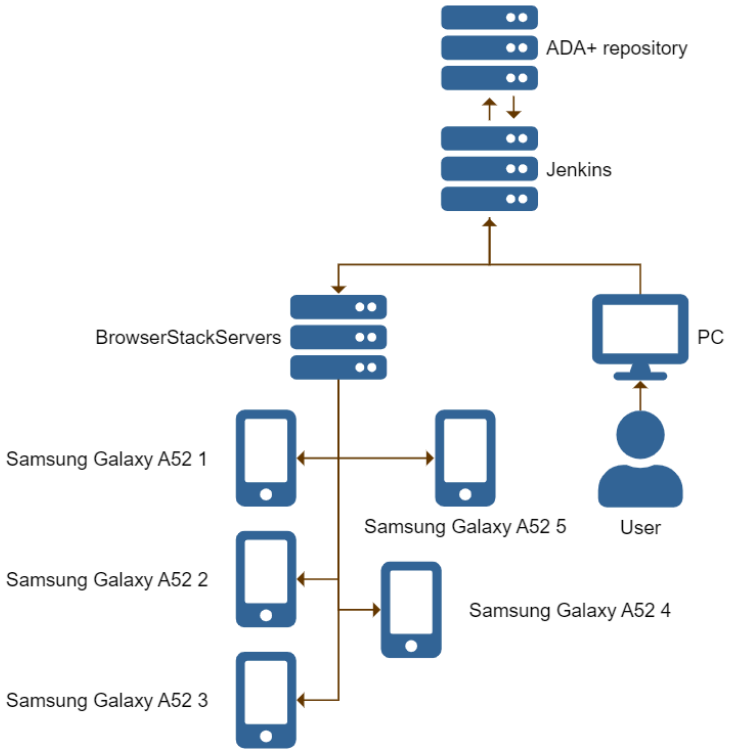


Figura 4.2.3.1: Ejecución Jenkins. Fuente: Propia.

Este tipo de ejecución se realiza desde el ordenador físico, pero a diferencia de las anteriores, no requiere tener clonado el repositorio. Solo se necesita acceder a los servidores de Jenkins y ejecutar el trabajo de BrowserStack, específicamente el test End2End. Este trabajo genera una APK de la rama "develop" y ejecuta los tests sobre la misma rama en BrowserStack. Al igual que en el método anterior, los servidores de BrowserStack se encargan de ejecutar los tests sobre la APK, paralelizando los tests en hasta 5 dispositivos simultáneamente.

4.2.4 pull request develop

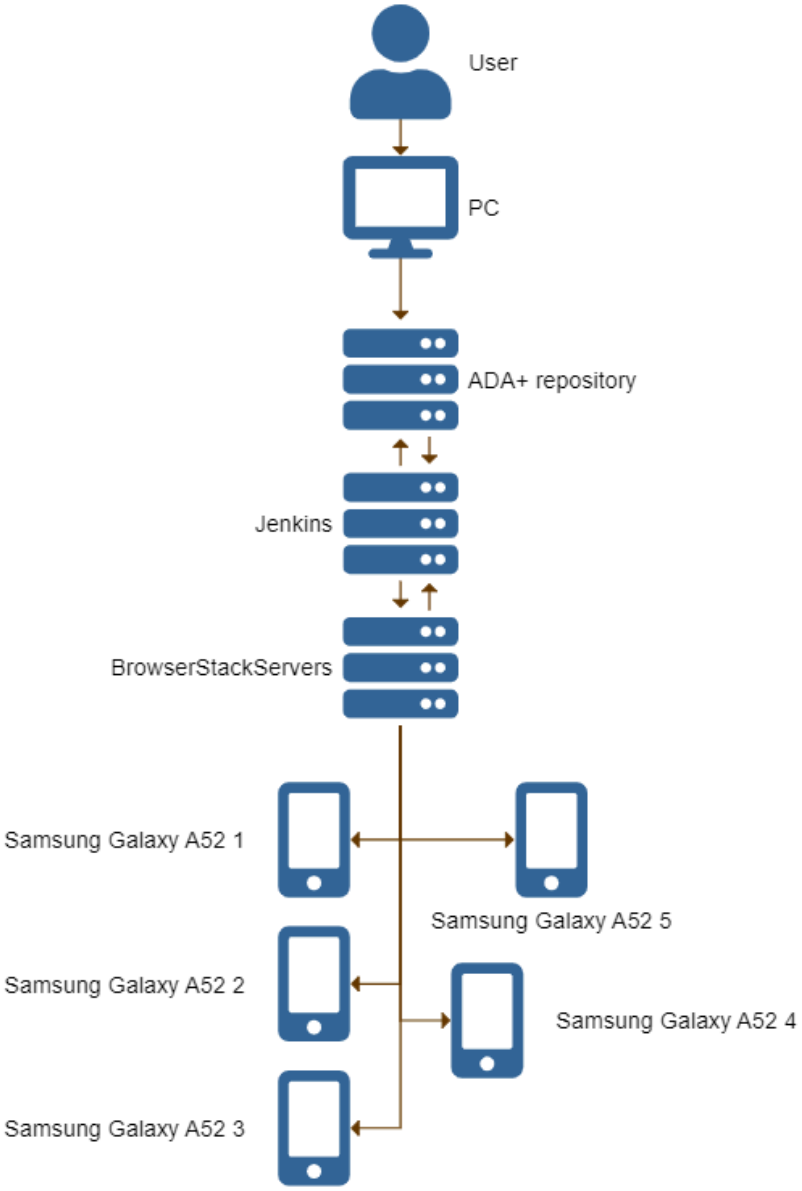


Figura 4.2.4.1:Ejecución al generar una pull request a Develop. Fuente: Propia.

Este tipo de ejecución se realiza siempre que alguien desde su ordenador genera una solicitud de extracción (pull request) del repositorio de ADA+ hacia la rama "develop". Al igual que en la ejecución Job Jenkins-Browser, no requiere tener clonado el repositorio. Solo se necesita acceder a los servidores de Jenkins y ejecutar el trabajo de BrowserStack, test End2End. Este trabajo genera una APK de la rama "develop" y ejecuta los tests sobre la misma rama en BrowserStack. Los servidores de BrowserStack se encargan de ejecutar los tests sobre la APK, paralelizando los tests en hasta 5 dispositivos simultáneamente.

4.3 Modelo UML

Los test en cuestión, están separados por clases en función de la funcionalidad a la que pertenecen, ya que la aplicación sobre la que los test dependen, está dividida en funcionalidades, como por ejemplo, "*Inventarios*", "*Traspasos*", "*Confirmaciones*", "*LogIn*" etc…
El patrón aplicado para estos es, Robot, este patrón, recomendado por muchas fuentes, como Google o Android, ayuda a la generación de estos, haciéndolos más fáciles de crear y entender. El patrón robot consiste en factorizar el código, juntando código por funciones comunes, como puede ser el Login, ya que esta función la ejecutarán todos los test, y en cambio de repetir varias líneas de código se crea una clase en un package llamado, "robots" y dentro de este una clase pública con el método "loginAndOnboarding()".
Todos los test dependen de una clase padre llamada BrowserStackJUnitTest.kt, esta contiene la creación del driver que necesitan el resto de test para interactuar sobre la aplicación, y una función que se ejecutará una vez terminado el test para matar el driver.

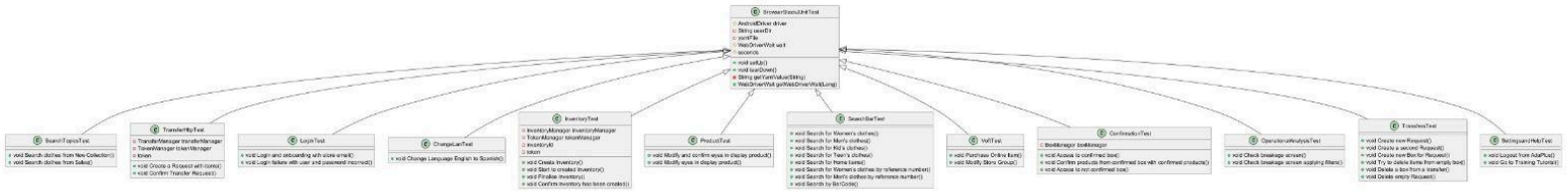


Figura 4.3.1 Diagrama de UML de los Test. Fuente: Propia.

Cada una de estas clases de Test están formadas por funciones, estas son los test, y como podemos observar cada clase o funcionalidad de ADA+, tiene varios test ya que cada funcionalidad tiene varias casuísticas a testear.

4.3.1 Diagramas uml con patrón Robot

A continuación, se introduce un diagrama de clases en formato UML que explica y representa la estructura del proyecto. Se ha decidido omitir las clases que establecen conexión con el backend para la creación de escenarios específicos para los tests, dado que no tienen relevancia directa sobre el patrón Robot. En cuanto a estas clases que establecen conexión con el servidor backend de ADA+, se ha optado por seguir una estructura sencilla compuesta por una interfaz y su correspondiente implementación. Esta elección facilita la adaptación a cualquier cambio que pueda surgir en la base de datos.

En la siguiente página, se proporcionará el diagrama UML para ayudar al lector a comprender mejor la estructura del proyecto.

Es importante destacar que existe una clase principal de la que todas las demás dependen para su ejecución. Esta clase, denominada BrowserStackJUnitTest, es responsable de configurar el driver y establecer la conexión con BrowserStack para la ejecución de los tests. En un segundo nivel, encontramos todas las clases de los tests, las cuales están separadas en función de las diferentes funcionalidades. Se ha elegido esta forma de fragmentación porque permite organizar el proyecto de manera eficiente sin granularizar en exceso, como podría suceder si se dividiera por pantallas, un enfoque extremo que resultaría poco eficiente.

En un tercer nivel, encontramos todos los robots, que son los encargados de ejecutar las funciones pertinentes de cada test. Esta estructura permite que los demás tests puedan acceder a otros robots sin necesidad de duplicar código, optimizando así la eficiencia y la mantenibilidad del código.

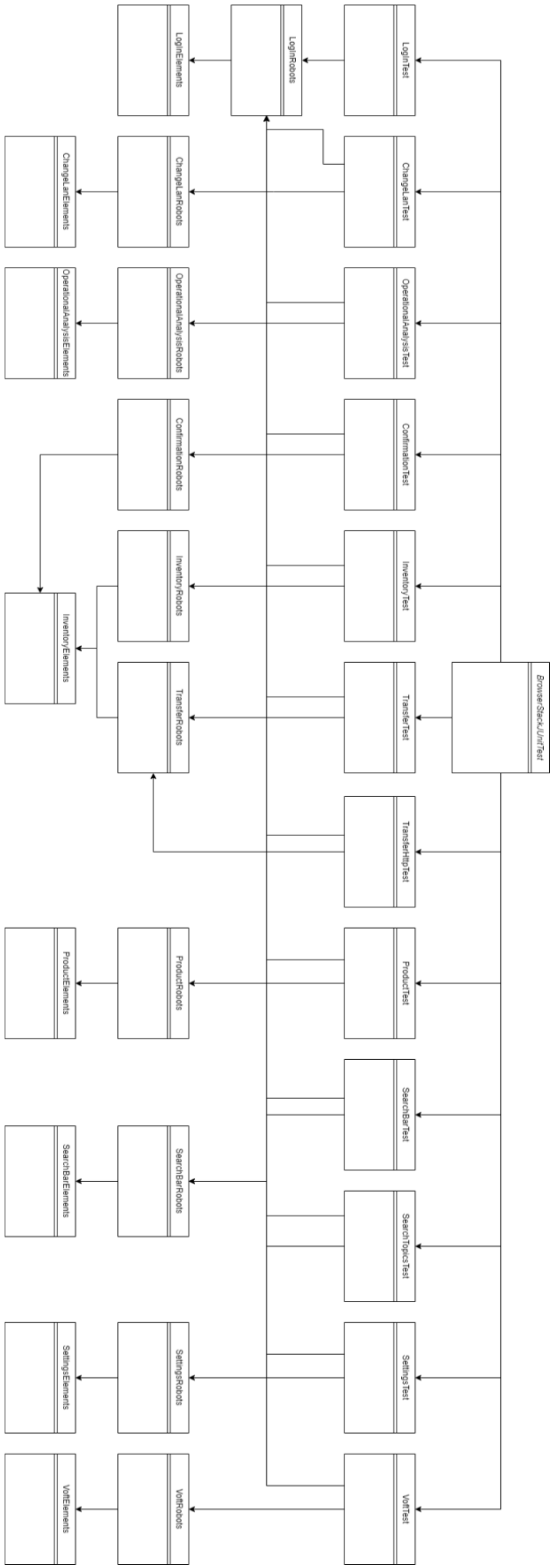


Figura 4.3.1.1 Diagrama uml con el patrón Robot

Podemos observar como todos los test por ejemplo dependen del robot de LoginRobots, ya que esta clase tiene la función que ejecuta el login sobre la aplicación y recordemos que cada test empieza desde la primera pantalla de la aplicación, porque cada test, es una sesión diferente, por lo que significa que son diferentes drivers o dispositivos móviles.

4.3.2 Diagramas de secuencia de los Test sobre BrowserStack

A continuación se adjuntan algunos diagramas de secuencia sobre los test.

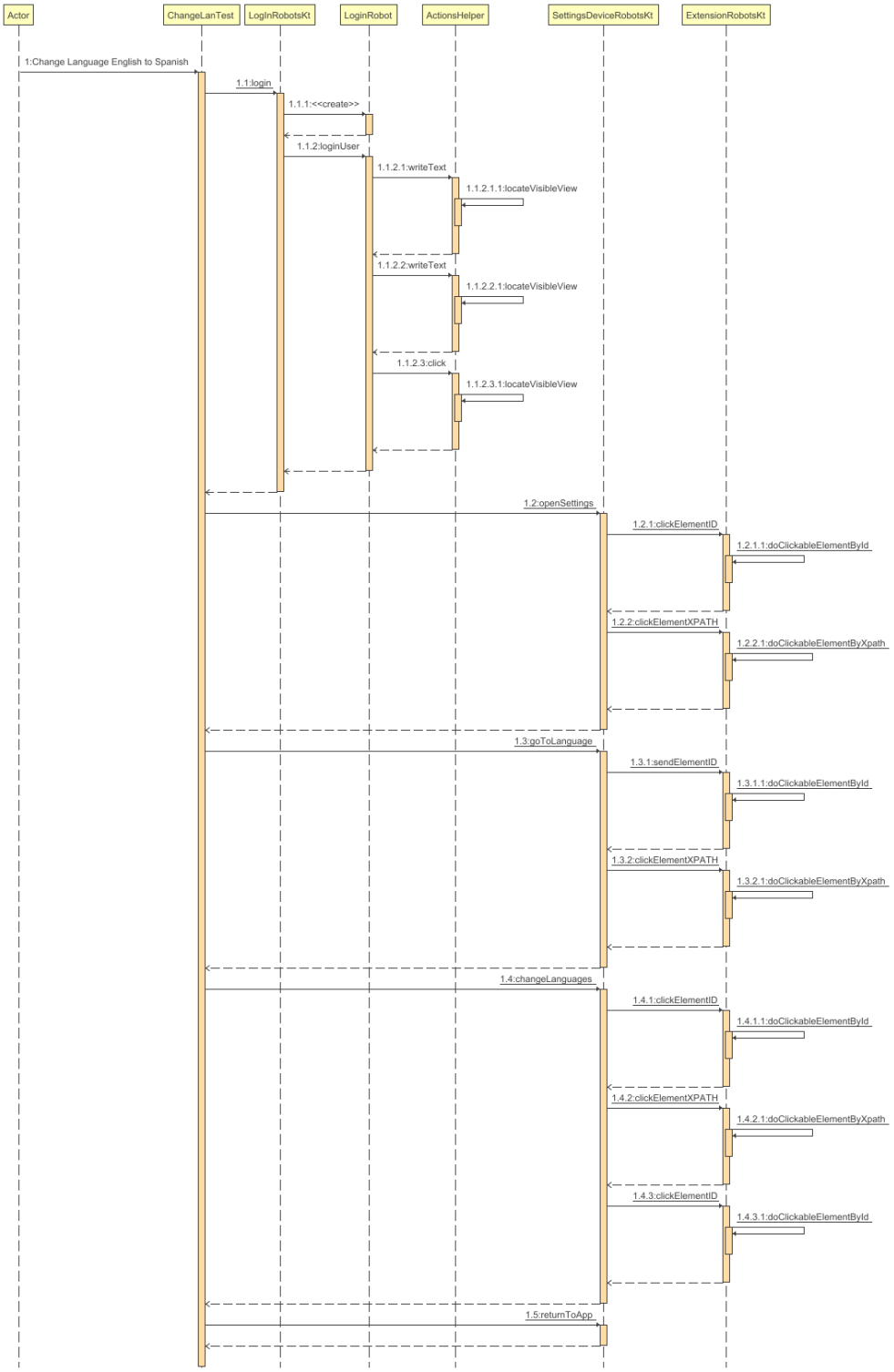


Figura 4.3.2.1: Diagrama de secuencia del test Cambiar Idioma. Fuente: Propia.

Este diagrama de secuencia, figura 4.3.2.1, es la ejecución del test para cambiar idioma del dispositivo

y comprobar que cambia correctamente, nos podemos fijar como ChangeLanTest solo se comunica con Robots, en este caso LogInRobots y SettingDeviceRobots, y estos se encargan de ejecutar las funciones que interactúan sobre el dispositivo móvil.

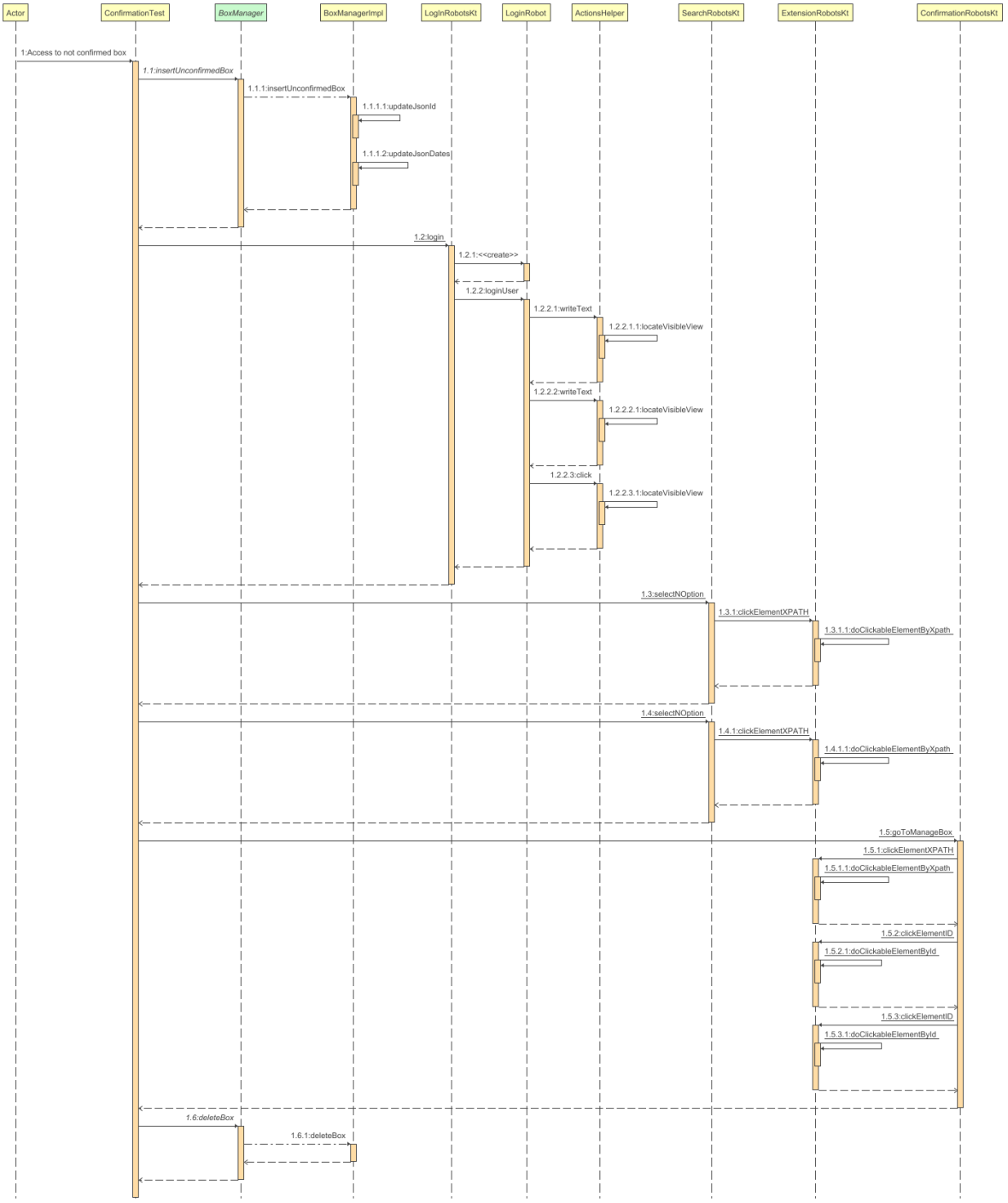


Figura 4.3.2.2: Diagrama de secuencia del test Acceder a una caja no confirmada. Fuente: Propia.

Este diagrama de secuencia, figura 4.3.2.2, es la ejecución del test para acceder a una caja no confirmada del proceso confirmaciones y comprobar que cambia correctamente el estado del inventario de creado ha iniciado, nos podemos fijar como ConfirmationTest solo se comunica con Robots, en este caso LogInRobots, SearchRobots y ConfirmationRobots.

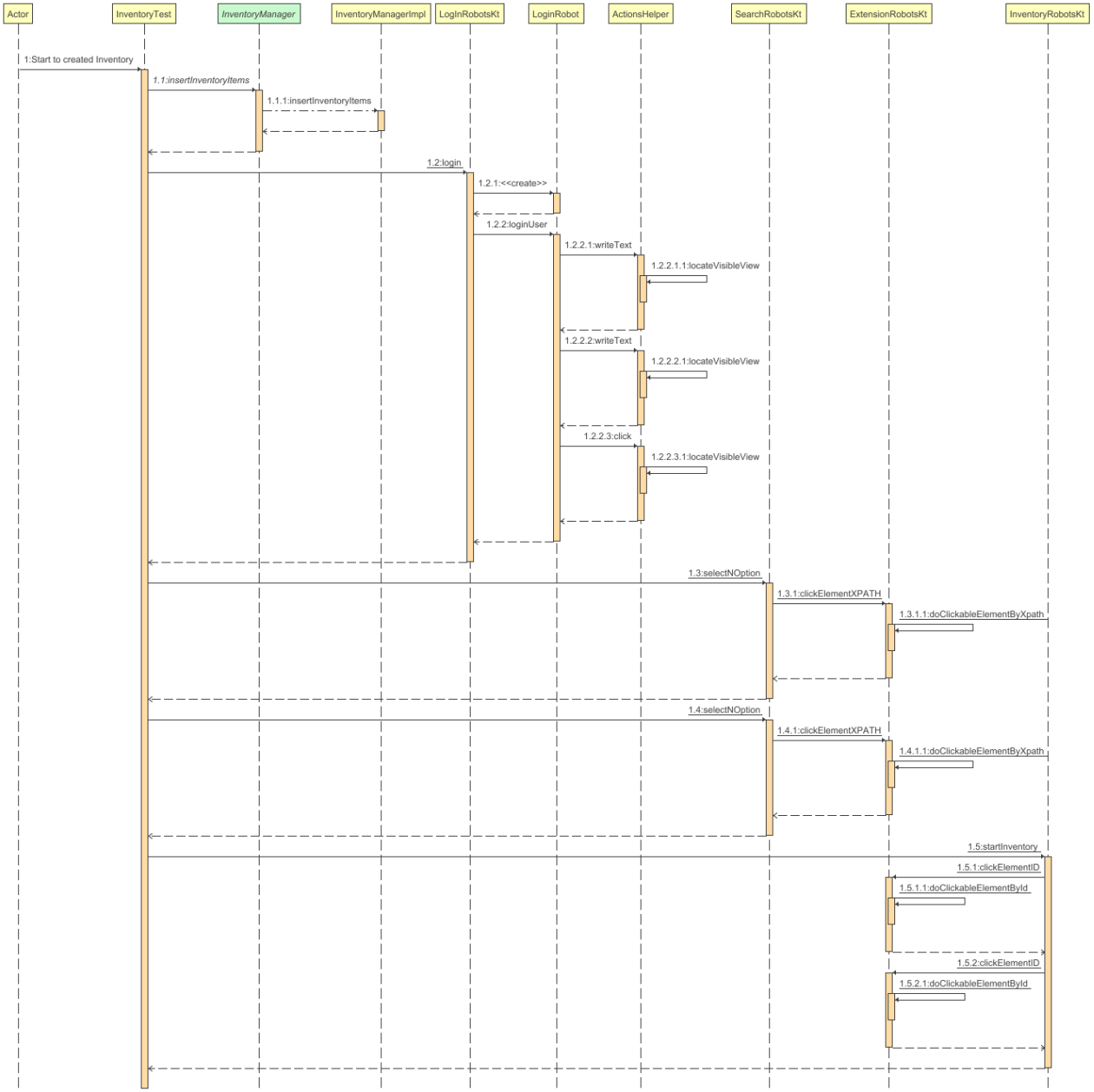


Figura 4.3.2.3: Diagrama de secuencia del test Empezar un Inventario creado. Fuente: Propia.

Este diagrama de secuencia, figura 4.3.2.3, es la ejecución del test para empezar un inventario creado y comprobar que se muestra correctamente, nos podemos fijar como las primeras funciones llaman a InventoryImpl, que se encarga de emular la funcionalidad de la pala, creando previamente con una llamada HTTP Request el objeto deseado. Después vuelve a ser como el resto de test, donde se llama a los distintos Robots requeridos para completar la ejecución del test.

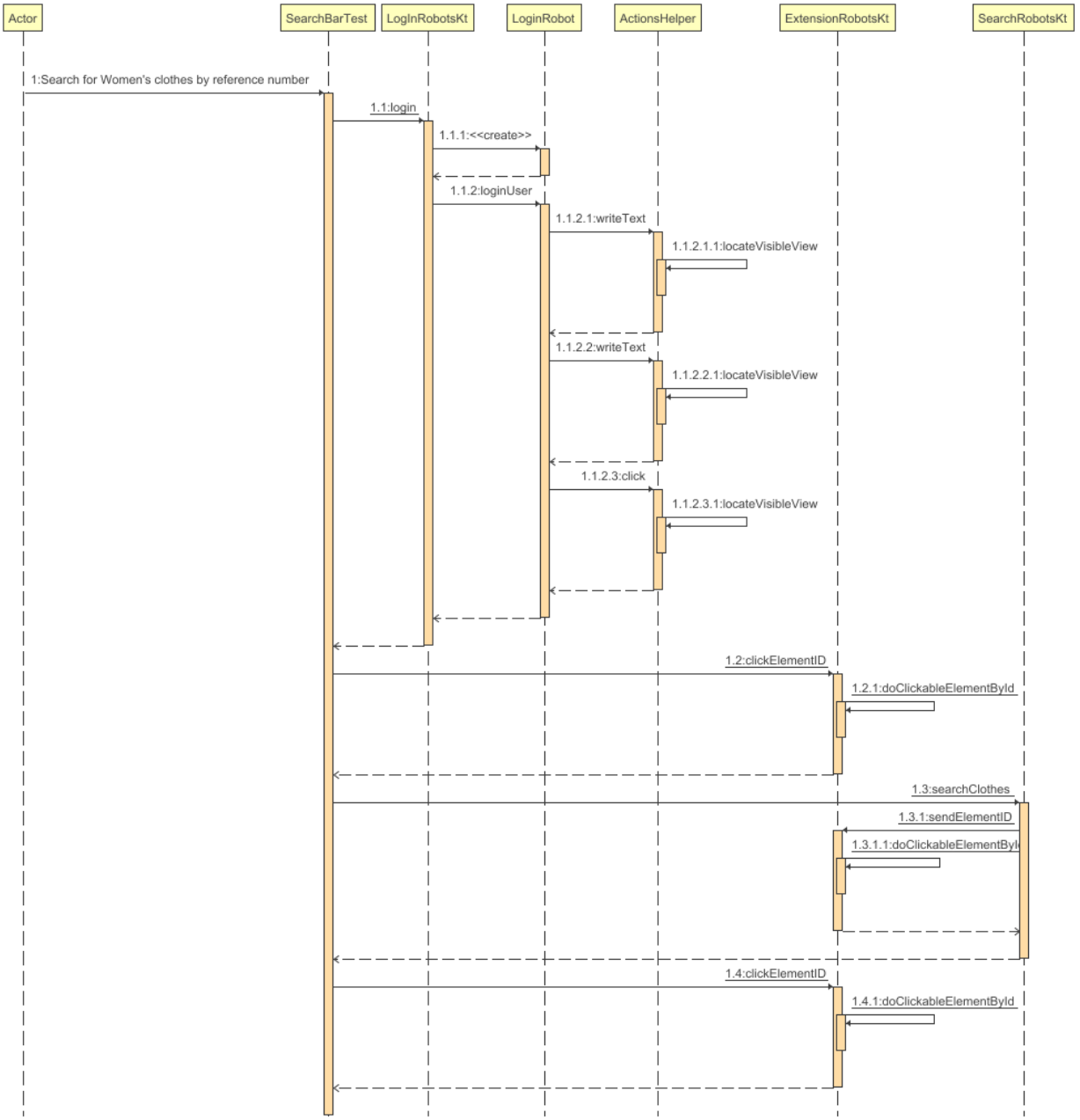


Figura 4.3.2.4: Diagrama de secuencia del test Buscar ropa de Mujer por número de referencia. Fuente: Propia.

Este diagrama de secuencia, figura 4.3.2.4, es la ejecución del test para buscar a través de un número de referencia de una renda, se espera llegar a la ficha de producto de la línea de Woman nos podemos fijar como SearchBarTest solo se comunica con Robots, en este caso LogInRobots y SearchRobots.

4.4 Diseño de Tests

El nombre de los test siempre se definen en inglés para permitir en un futuro la mayor comprensión de lo que hace el test para todo el mundo.

Test	#1 Login and onboarding with store email
Funcionalidad a testear	LogIn
Precondición	● No estar previamente con la sesión empezada ● Sea la primera vez en abrir y iniciar sesión en el dispositivo móvil ● Conexión internet
Escenario principal de éxito	
En caso de que se ejecute correctamente el login, luego aparecerá una serie de pantallas para realizar el *onboarding* de la aplicación, este solo se tiene que hacer la primera vez que se inicia sesión en un dispositivo. Para comprobar el éxito del test se espera llegar al menú principal, en caso contrario dará *Failed*	

Test	#2 Login failure with incorrect credentials
Funcionalidad a testear	LogIn
Precondición	● Conexión internet
Escenario principal de éxito	
El escenario de éxito para este test es que no se realice correctamente el login y de un mensaje de error conforme las credenciales son incorrectas, y se usan estas para comprobar si la aplicación responde correctamente a credenciales erróneas	

Test	#3 Logout from AdaPlus
Funcionalidad a testear	Ajustes y Ayuda
Precondición	● Conexión internet ● El dispositivo debe haber acabado de sincronizar todos los datos de la tienda con el servidor
Escenario principal de éxito	
Para comprobar que se ha ejecutado bien se esperará acabar en la pantalla de login	

Test	#4 Go to training tutorial
Funcionalidad a testear	Ajustes y Ayuda
Precondición	• Conexión internet
Escenario principal de éxito	
Para comprobar el éxito de este test se esperará llegar a la página web a la que te redirige la opción, donde se comprobará que sea la página correcta	

Test	#5 Change Language English to Spanish
Funcionalidad a testear	Cambiar Idioma (Lokalise)
Precondición	• Conexión internet
Escenario principal de éxito	
Se ejecutará la función de Login, una vez hecho, se irá a ajustes para cambiar el idioma del dispositivo y comprobar que se ha cambiado con éxito, comprobará que el título del menú principal pasa de estar en inglés a estar en español. De este cambio de idioma se encarga la tecnología Lokalize, es la que usamos para gestionar los idiomas.	

Test	#6 Check breakage screen
Funcionalidad a testear	Rotura
Precondición	• Conexión internet
Escenario principal de éxito	
Para validar esta funcionalidad se ejecuta el login y el onboarding, una vez en el menú se accede a la opción de roturas, se comprueba que se muestran los valores de rotura de la tienda	

Test	#7 Check breakage screen applying filters
Funcionalidad a testear	Rotura
Precondición	• Conexión internet
Escenario principal de éxito	
Para validar esta funcionalidad se ejecuta el login y el onboarding, una vez en el menú se accede a la opción de roturas, se comprueba que se muestran los valores de rotura de la tienda, entonces aplicar filtro para comprobar que funciona a la perfección toda la funcionalidad.	

Test	#8 Modify and confirm eyes in display product
Funcionalidad a testear	Producto
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca una prenda aleatoria a la cual cambiar los ojos (los ojos representan cuantas prendas deben estar expuestas en tienda y que tallas), y confirmar el cambio con el código de empleado. Una vez confirmado el cambio se espera llegar a la pantalla que confirme el cambio exitoso de los ojos de la prenda	

Test	#9 Modify eyes in display product
Funcionalidad a testear	Producto
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca una prenda aleatoria a la cual cambiar los ojos, pero esta no confirma el cambio y comprobar que no se ha efectuado el cambio.	

Test	#10 Purchase Online Item
Funcionalidad a testear	VOFT (compra online desde tienda física)
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca una prenda aleatoria, una vez se acceda a la ficha de producto, generar una petición VOFT, es decir, generar una petición a central para comprar la prenda online, rellenar todos los datos necesarios, tanto del cliente como la prenda solicitada y “comprar” la prenda.	

Test	#11 Modify Store Group
Funcionalidad a testear	VOFT (compra online desde tienda física)
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca una prenda aleatoria, una vez se acceda a la ficha de producto, y consultar las tiendas que están configuradas en “Store Group”, tiendas para consultar el stock de la prenda seleccionada, modificar la lista de tiendas y comprobar que se actualice la lista y el estado de si existe esa prenda en las tiendas añadidas.	

Test	#12 Search clothes from New Collection
Funcionalidad a testear	Buscar por tópicos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador por la sección de New Collection, que siempre debe existir una, y comprobar que salgan prendas.	

Test	#13 Search clothes from Sales
Funcionalidad a testear	Buscar por tópicos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador por la sección de Sales (rebajas), que siempre debe existir una colección, y comprobar que salgan prendas con descuentos	

Test	#14 Search for Women's clothes
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador prendas de la línea de Woman	

Test	#15 Search for Men's clothes
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador prendas de la línea de Man	

Test	#16 Search for Kid's clothes
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador prendas de la línea de Kid	

Test	#17 Search for Teen's clothes
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador prendas de la línea de Teen	

Test	#18 Search for Home items
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador prendas de la línea de Home	

Test	#19 Search for Women's clothes by reference number
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador una prenda de la línea de Woman con el número de referencia de la prenda	

Test	#20 Search for Men's clothes by reference number
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador una prenda de la línea de Man con el número de referencia de la prenda	

Test	#21 Search by BarCode
Funcionalidad a testear	Buscador de productos
Precondición	• Conexión internet
Escenario principal de éxito	
Una vez ejecutado el login y onboarding se busca en el buscador una prenda por el número del código de barras, dos opciones para ejecutar esta función es escaneando la etiqueta con la cámara o bien escribiendo el número con el teclado numérico	

Test	#22 Access to not confirmed box
Funcionalidad a testear	Proceso - Confirmación
Precondición	• Conexión internet • Crear una caja no confirmada en el servidor de MongoDB
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a confirmación, dentro de procesos, y confirmar que se muestra la caja no confirmada.	

Test	#23 Confirm products from confirmed box with confirmed products
Funcionalidad a testear	Proceso - Confirmación
Precondición	• Conexión internet • Crear una caja confirmada con productos confirmados en el servidor de MongoDB • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a confirmación, dentro de procesos, y confirmar que se muestra la caja confirmada, acceder a los datos de esta caja y validar que se han confirmado todas las prendas de la caja.	

Test	#24 Access to confirmed box
Funcionalidad a testear	Proceso - Confirmación
Precondición	• Conexión internet • Crear una caja confirmada con productos confirmados en el servidor de MongoDB • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a confirmación, dentro de procesos, y confirmar que se muestra la caja confirmada, acceder a los datos de esta caja y validar que se han confirmado algunas de las prendas de la caja pero no en su totalidad.	

Test	#25 Create Inventory
Funcionalidad a testear	Proceso - Inventario
Precondición	• Conexión internet • No debe existir otro inventario sin finalizar en la misma tienda • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a inventario, dentro de procesos, crear un inventario y validar que se crea correctamente	

Test	#26 Start to created Inventory
Funcionalidad a testear	Proceso - Inventario
Precondición	• Conexión internet • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a inventario, dentro de procesos, validar que existe un inventario sin empezar, acceder a este y empezarlo. Con una llamada al backend introducir datos para emular la lectura de la *pala,* dispositivo bluetooth para leer tags RFID.	

Test	#27 Finalize Inventory
Funcionalidad a testear	Proceso - Inventario
Precondición	• Conexión internet • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a inventario, dentro de procesos, validar que existe un inventario empezado, acceder a este y finalizarlo. Validar que no existe ningún inventario no finalizado y el generado se ha finalizado correctamente usando su id. (cada inventario tiene un id, para diferenciarlos y guardar registro de todos ellos)	

Test	#28 Confirm Inventory has been created
Funcionalidad a testear	Proceso-Confirmación
Precondición	• Conexión internet • Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding con ayuda del buscador, buscar una prenda confirmada en el inventario finalizado y validar que en la ficha de producto aparece una prenda disponible en almacén	

Test	#29 Create a Request with items
Funcionalidad a testear	Proceso-Traspaso
Precondición	• Conexión internet • Debe finalizar la sincronización de datos de REALM • Añadir prendas a la request modificando de la base de datos
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos, crear una *request* para transferir prendas a otra tienda, en este caso una tienda virtual (usada para testear). Validar la creación de la request y añadir una caja con prendas.	

Test	#30 Confirm Transfer Request
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos, acceder a la request creada anteriormente y confirmar su traspaso.	

Test	#31 Create new Request
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos, crear una *request* para transferir prendas a otra tienda, en este caso una tienda virtual.	

Test	#32 Create a second Request
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos, crear una *request* para transferir prendas a otra tienda, en este caso una tienda virtual (usada para testear). Esta petición debe dar error ya que ya existe otra petición sin finalizar.	

Test	#33 Create new Box for Request
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos,acceder a la request con destino la tienda virtual y crear una caja vacía.	

Test	#34 Try to delete items from empty box
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos,acceder a la request con destino la tienda virtual y acceder a la caja vacía. Intentar borrar todos los ítems de la caja y obtener el mensaje de error al estar previamente sin prendas.	

Test	#35 Delete a box from a transfer
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos,acceder a la request con destino la tienda virtual. Una vez en la lista de cajas creadas, borrar las cajas exitosamente deslizándose a la izquierda.	

Test	#36 Delete empty Request
Funcionalidad a testear	Proceso-Traspaso
Precondición	● Conexión internet ● Debe finalizar la sincronización de datos de REALM
Escenario principal de éxito	
Una vez ejecutado el login y onboarding acceder a traspasos, dentro de procesos, entonces borrar el traspaso creado una vez ya no tiene cajas, deslizando la request a la izquierda.	

5. Metodología y herramientas de seguimiento

5.1 Metodología

La elección de la metodología adecuada es esencial para la gestión efectiva del proyecto. En este sentido, se implementará el marco de trabajo Scrum, una metodología ágil ampliamente reconocida por su capacidad para adaptarse a los cambios y promover la colaboración dentro del equipo. [10]

Scrum se caracteriza por dividir el desarrollo de un proyecto basados en tiempos de trabajo cortos llamados "sprints", en Mango se trabaja con sprints de 3 semanas

Cada sprint empieza un miércoles con una reunión de planificación donde se define el conjunto de tareas a realizar, estas tareas las decide el scrum manager y son escogidas del backlog en función de la urgencia, prioridad o importancia que tienen, y para finalizar cada sprint, se analiza cómo ha ido, si se han acabado todas las tareas pendientes, posibles factores de mejora y otros factores.

La ventaja de una metodología ágil permite una respuesta inmediata a los cambios de última hora de un proyecto, una mayor transparencia en el progreso y una entrega continua de valor al cliente haciendo que cada tres semanas tenga una nueva actualización disponible.

Además se realizan varias reuniones periódicamente.

La más habitual será una diaria, en la que solo se reúne el equipo para hablar en un máximo de 15 minutos sobre los avances del día y en que están o podrían estar bloqueados.

El segundo tipo de reunión periódica será semanal, donde dentro de esta categoría tendremos dos reuniones diferentes, la primera semana será una en la que todo el departamento se reúne y se presenta la evolución de cada equipo y la segunda es una en la que todos los Android Developers de varios equipos se reúnen para mantener coherencia y seguir las mismas normas a la hora de desarrollar código.

La plataforma que se usará para subir el código será Bitbucket [11], un repositorio muy parecido al funcionamiento de GitHub.

Las tareas serán controladas desde Jira [12] una plataforma que facilita la creación de pizarras, kanban… Básicamente una plataforma que ofrece poder trabajar con la metodología Scrum muy fácil y además ofrece una conexión con bitbucket que facilita la generación de tareas y monitorizar el estado de estas.

En este repositorio se mantendrá una estructura de ramas limpia y con una nomenclatura fijada desde un principio, es decir, existirá la rama “develop” y se ara una rama con el nombre de la tarea para luego una vez finalizada generar una pull request y poner de ojeadores a tres personas del equipo de Android developers.

5.2 Seguimiento

El seguimiento y la gestión del proyecto se llevarán a cabo utilizando un conjunto de herramientas y técnicas que facilitarán la comunicación, la colaboración y el control del progreso.

La herramienta principal a destacar se encuentra el software de gestión de proyectos Jira Cloud, que proporciona una plataforma centralizada para la planificación y seguimiento de las tareas, la asignación de recursos y la gestión de incidencias. Además que aporta una facilidad extrema para integrarlo con Bitbucket repositorio donde se encuentran todos los proyectos de Mango

Además, se establecen reuniones periódicas de seguimiento, como reuniones semanales de stand-up, donde cada equipo compartirá brevemente el progreso realizado, los desafíos encontrados y los próximos pasos que su equipo tomará.

Estas reuniones fomentan la transparencia y la comunicación entre los miembros del equipo, lo que facilitan la identificación prematura de problemas y la toma de decisiones oportunas. Asimismo, se emplearán herramientas de comunicación en línea, como Microsoft Teams[13], para facilitar la comunicación y la resolución de problemas de forma ágil y eficiente.

5.3 Cambios de metodología

A lo largo del desarrollo del proyecto, se ha mantenido la metodología Scrum propuesta inicialmente, adaptándola a las necesidades específicas del proyecto y del equipo. Esta metodología ágil ha demostrado ser efectiva para gestionar las tareas y responder a los cambios y desafíos que han surgido durante el proceso.

Además, se ha llevado a cabo un seguimiento semanal con el director del proyecto, Federico Pérez Marina. Estas reuniones de seguimiento han sido esenciales para revisar el progreso del proyecto, discutir cualquier problema o desafío y planificar los próximos pasos. El feedback y la orientación proporcionados durante estas reuniones han sido de gran valor para mantener el proyecto en el camino correcto y garantizar que se cumplan los objetivos establecidos.

En resumen, la metodología y las herramientas de seguimiento seleccionadas han demostrado ser efectivas para la gestión del proyecto, permitiendo un desarrollo fluido y eficiente del mismo. Aunque se han realizado ajustes a lo largo del camino, estos han sido en respuesta a las necesidades específicas del proyecto y han contribuido a su éxito.

6. Descripción de las tareas

6.1 Introducción

Este proyecto arrancó a finales de enero, por lo que significa que aún no ha finalizado y por ese motivo las tareas que se verán reflejadas son las que se han ido realizando desde el 1 de febrero hasta la semana de presentación de este proyecto. Este proyecto tendrá en total un desarrollo de 20h a la semana, por lo que al final tendrá un tamaño de 370h de desarrollo, más aparte las horas que necesite para documentar, preparar la documentación final, presentación y defensa, probablemente serán entre 150h y 200h de documentación, por lo que este proyecto costará entre 490 horas y 440h.

6.2 Tareas

Este proyecto se desarrollará por sprints, para definir las tareas a resultado más difícil de lo previsto, ya que estas se especifican cada tres semanas en una reunión llamada *sprint planning* y el problema está en que como este proyecto no tiene ningún antecedente ni nadie que supiera sobre esta tecnología se ha ido haciendo más bien historias de usuario o tareas que engloban a otras. Para intentar expresar las tareas más aproximadas a lo que serán en el final, mi *product manager* me ha ayudado a definir las tareas relacionadas con mi proyecto. También otro inconveniente es que las tareas realizadas a día de hoy se han perdido ya que Mango ha realizado una migración de las pizarras de Jira a Jira Cloud y se ha perdido la información de los anteriores sprints.
Cada tarea tiene su descripción, con una estimación de horas y su identificador. En caso que haya dependencias también se mostrará.

6.2.1 Tareas de Documentación

- TD1 → Alcance (30h): Documentación de la primera entrega. Esta entrega incluye la gestión básica del proyecto, su justificación, alcance y metodología que se va a usar.
  Sin dependencias.

- TD2 → Planificación (10h): Documentación de la segunda entrega. Esta entrega incluye planificación, definición de tareas, diagrama de Gantt y gestión de los riesgos observados y mencionados en la primera entrega.
  Dependencias: TD1.

- TD3 → Gestión económica y sostenibilidad (10h): Documentación de la tercera entrega. Esta entrega incluye los presupuestos y el informe de sostenibilidad.
  Dependencias: TD2.

- TD4 → Documentación de la fase inicial (15h): Recopilación de documentación de las tres tareas anteriores y se aplican las correcciones pertinentes.
  Dependencias: TD3.

- TD5 → Documentación de la fase de seguimiento (30h): Se prepara la entrega junto al ponente, en mi caso, Matias Martinez, para compartir y tener conocimiento del progreso y estado del proyecto
  Dependencias: TD4.

- TD6 → Documentación de la fase final (70h): Se redacta y se corrige lo restante del trabajo
  Dependencias: TD5.

- TD7 → Preparación de la presentación final (40h): Preparación de la presentación del trabajo y preparación de su defensa

Dependencias: TD6.

6.2.2 Tareas de Planificación

- TP1 → *Daily Performance* (30min): Cada mañana el equipo de *ADA+* performance se reúne para compartir la situación en la que están, si están avanzando sin problema o por el contrario están bloqueados por algún motivo.
  Sin dependencias.

- TP2 → Refinamiento (1h): Cada tres semanas se realiza esta reunión, justo después de finalizar cada sprint, se reúne el equipo para refinar historias de usuarios y tareas, ya que con esto obtenemos información para encaminar el siguiente *sprint planning.*
  Sin dependencias.

- TP3 → *Sprint planning* (1h): Cada tres semanas, se reúnen todos los equipos de *ADA+,* esta reunión se celebra con motivo de intentar planificar el siguiente *sprint* y definir las nuevas tareas de las historias de usuario.
  Sin dependencias.

- TP4 → *Android Coordination* (1h): Esta reunión se celebra cada semana entre todos los *Androids Developers* de ADA+, con motivo de coordinar tareas de android y acordar términos para que todos los programas tengan las misma estructuras, además esta reunión se usa para presentar nuevas tecnologías relacionadas que se pueden usar.
  Sin dependencias.

- TP5 → *Weekly Scrum* (1h): Esta reunión se celebra cada semana entre todos los miembros del departamento "Software Intern", está formada por varios equipos dentro del mismo departamento y en total somos unas 80 personas aproximadamente. Esto se realiza para compartir los avances constantes de todos, de esta forma todos estamos actualizados y al corriente de donde se están invirtiendo los puntos de esfuerzo de cada equipo.
  Sin dependencias.

6.2.3 Tareas de Desarrollo

Como punto preliminar antes de definir las tareas, en muchas de ellas, sobre todo las tareas más relacionadas con la elaboración y desarrollo de los tests, se tiene que tener en cuenta que antes se ha preguntado por los casos de uso más preguntar en varias tiendas como lo usan y en qué situaciones lo necesitan.

Configuración del proyecto

- TCP1 → Creación Proyecto y configuración de Marco de Test (50h): Esta tarea consiste no solo en la creación del proyecto sino también en la investigación en que lenguaje de programación de tests (JUnit 4, JUnit 5, Nunit, TestNG, selenium WebDriver), idioma de programación (Kotlin, Java, Python) y que forma de compilar usar (maven o gradle).
  Sin dependencias.

- TCP2 → Creación *Job* en *Jenkins* (20h): Esta tarea es la creación del *job* de *Jenkins* respetando las nomenclaturas ya usadas y la arquitectura creada en Mango. Para crear y configurar el proyecto para que se adecue más a las necesidades.
  Dependencias: TCP1.

- TCP3 → Investigar que *Capabilities* se adecuan más a ADA+ (15h): Esta tarea consiste en descubrir qué dispositivos se usan en tienda para simular los test en dispositivos lo mas parecidos o exactos posibles, que tipo de conexión a internet tienen (wifi, 4g, 5g, 3g), que horas

usan cada funcionalidad de la aplicación.
Dependencias: TCP2.

- TCP4 → Investigar Arquitectura (15h): Investigar qué arquitecturas hay para los test *End to End* y escoger cuál se adapta mejor.
  Sin dependencias.

Desarrollo de Test

- TDT1 → Diseñar y investigar que test crear para la aplicación de ADA+ (15h): Para realizar esta tarea se va hacer uso de *Firebase* ya que se reportan que procesos o actividades se han realizado durante ese dia, semana, mes o histórico, además se puede ordenar por tipo de dispositivos y país. Para asegurar también se acudira a varias tiendas para ver qué funcionalidades y uso hacen de la aplicación.
  Dependencias: TCP3.

- TDT2 → Crear test Log In (7h): Esta tarea implica la creación de pruebas para la funcionalidad de inicio de sesión en la aplicación. Se estima que tomará 7 horas completarla.
  Dependencias: TDT1.

- TDT3 → Crear test Log out (7h): En esta tarea, se crearán pruebas para la funcionalidad de cierre de sesión en la aplicación. Tiene una duración estimada de 7 horas.
  Dependencias: TDT1.

- TDT4 → Crear test buscador principal por nombre (10h): Se trata de crear pruebas para la funcionalidad de búsqueda principal por nombre de prendas y productos en la aplicación.
  Dependencias: TDT1.

- TDT5 → Crear test buscador principal por referencia (10h): Similar a la tarea anterior, pero enfocada en la búsqueda por referencia.
  Dependencias: TDT1.

- TDT6 → Crear test buscador principal por *barcode* (10h): Esta tarea implica crear pruebas para la búsqueda principal por código de barras en la aplicación.
  Dependencias: TDT1.

- TDT7 → Crear test Proceso Inventario (15h): Aquí se crean pruebas para el proceso de inventario en la aplicación, creación gestión y comprobar que entre dispositivos distintos se sincronizan correctamente los inventarios .
  Dependencias: TDT1.

- TDT8 → Crear test Proceso Traspasos (15h): Similar a la tarea anterior, pero enfocada en el proceso de traspasos, comprobando la creación de traspasos a otras tiendas, gestión de cada traspaso creado, eliminar y añadir prendas a cada traspaso y que se actualice correctamente.
  Dependencias: TDT1.

- TDT9 → Crear test acceder Training (7h): Esta tarea implica crear pruebas para acceder a la funcionalidad de entrenamiento en la aplicación, función que sirve para formar a los nuevos trabajadores de tiendas.
  Dependencias: TDT1.

- TDT10 → Crear test Compra Online (15h): Aquí se crearán pruebas para la funcionalidad de compra en línea de prendas como comprar online con stock en tienda o sin.
  Dependencias: TDT1.

- TDT11 → Crear test Cambiar Idioma (15h): Esta tarea implica crear pruebas para cambiar el idioma en la aplicación y asegurar que se cambia el idioma de la aplicación ya que es importante para aplicaciones de otros países.
  Dependencias: TDT1.

- TDT12 → Crear test Buscar por Tópicos (15h): Aquí se crean pruebas para la funcionalidad de búsqueda por tópicos en la aplicación como Nuevas Colecciones o Rebajas.
  Dependencias: TDT1.

- TDT13 → Crear test Operaciones y Análisis (12h): Esta tarea implica crear pruebas para las operaciones y análisis en la aplicación, aquí se muestra cuántas prendas se rompen, que rendimiento tiene cada producto.
  Dependencias: TDT1.

- TDT14 → Crear test Proceso Confirmaciones (15h): Similar a las tareas anteriores de proceso, pero enfocado en confirmaciones, que consiste en confirmar llegadas de camiones con paquetes.
  Dependencias: TDT1.

  En general, estas tareas implican la creación de pruebas para diferentes aspectos y funcionalidades de una aplicación, y todas dependen del trabajo realizado en la tarea TDT1.

Integración proyecto a los *Android Developers*

- TAD1 → Presentar Proyecto a los Android Developers de ADA+ (7h): Presentar la arquitectura y forma de desarrollar los tests a los *Android Developers*.
  Sin dependencias.

- TAD2 → Integrar Fase Inicial con *Android Developers de ADA+* (15h): Comenzar a integrar test de ADA+ con un integrante como mínimo de cada equipo relacionado con ADA+.
  Dependencias : TAD1.

- TAD3 → Integrar Fase final con *Android Developers de ADA+* (20h): Comenzar a integrar test de ADA+ con todos los integrantes de los equipos relacionados con ADA+.
  Dependencias : TAD2.

6.3 Recursos

A continuación, enumero los recursos necesarios para asegurar el éxito del proyecto.

Antes de enumerar los recursos necesarios también recordar que para este proyecto elaborado en Mango, se realizará en sus oficinas situadas en Palau Solità i Plegamans, para llegar a esas oficinas hay varios autobuses lanzaderas. También se podrá trabajar desde casa propia la mitad de la semana, pero la otra mitad tiene que ser presencial en las oficinas de Mango.

- **Recursos Humanos:** Yo [M] como parte del equipo de desarrollo. El equipo de Android Developers [AD] del que yo también formo parte. Necesitaremos también un Product Owner [PO], del equipo especializado en DevOps [DO], además habrán varias personas que me irán dando feedback de mi trabajo realizado en Mango, la primera es Federico Perez Marina [FPM] y Joan Martinez Garcia [JMG] este es quien dirige al equipo de Android Developers.

- **Recursos Materiales:** Los elementos esenciales para llevar a cabo el proyecto y las reuniones de equipo incluyen una serie de equipos físicos y software necesario. Para ello, tanto en casa como en la oficina, requeriré:

    - [IJ] IntelliJ IDEA: IDE para Java y Kotlin.

    - [GIT] Bitbucket: Herramienta de gestión de proyectos y repositorios.

    - [J] Jenkins: Servidor de automatización para *CI*.

    - [O] Office 365: Paquete de programas Office 365 como Outlook y Teams.

    - [K8S] Kubernetes: Sistema complementario a Jenkins para levantar un *Job.*

    - [AT] Atenea: Herramienta de trabajo para entregas o para sacar información para el desarrollo de trabajo.

    - [GC] Google Chrome: Herramienta para investigar en internet y conseguir información relevante al trabajo.

    - [BS] BrowserStack: Plataforma para generar test de ADA+.

    - [GANTT] Gantt Project: Herramienta para crear el diagrama de GANTT

7. Estimaciones y Gantt

ID	TAREA	HORAS	DEPENDENCIAS	RECURSOS
TD1	Alcance	30	-	[M][AD][PO][FPM][JMG][GC][O][AT]
TD2	Planificación	10	TD1	[M][AD][PO][FPM][JMG][GC][O][GANTT][AT]
TD3	Gestión económica y sostenibilidad	10	TD2	[M][AD][PO][FPM][JMG][GC][O][AT]
TD4	Documentación de la fase inicial	15	TD3	[M][AD][PO][FPM][JMG][GC][O][AT]
TD5	Documentación fase de seguimiento	30	TD4	[M][AD][PO][FPM][JMG][GC][O][AT]
TD6	Documentación de la fase final	70	TD5	[M][AD][PO][FPM][JMG][GC][O][AT]
TD7	Preparación de la presentación final	40	TD6	[M][AD][PO][FPM][JMG][GC][O][AT]
TP1	Daily Performance	0.5	-	[M][AD][JMG][DO][JMG][O]
TP2	Refinamiento	1	-	[M][AD][JMG][DO][PO][JMG][O]
TP3	Sprint planning	1	-	[M][AD][JMG][DO][PO][JMG][O]
TP4	Android Coordination	1	-	[M][AD][JMG][O]
TP5	Weekly Scrum	1	-	[M][AD][JMG][DO][PO][JMG][FPM]
TCP1	Creación Proyecto y configuración de Marco de Test	50	-	[M][IJ][JMG][BS][GIT]
TCP2	Creación Job en Jenkins	20	TCP1	[M][IJ][J][K8S][BS][GIT]
TCP3	Investigar que Capabilities se adecuan más a ADA+	15	TCP2	[M][IJ][JMG][BS][GIT][GC]
TCP4	Investigar Arquitectura	15	-	[M][IJ][JMG][BS][GIT]
TDT1	Diseñar y investigar que test crear para la aplicación de ADA+	15	TCP3	[M][IJ][JMG][BS][GIT]
TDT2	Crear test Log In	7	TDT1	[M][IJ][JMG][BS][GIT]
TDT3	Crear test Log out	7	TDT1	[M][IJ][JMG][BS][GIT]
TDT4	Crear test buscador principal por	10	TDT1	[M][IJ][JMG][BS][GIT]

	nombre			
TDT5	Crear test buscador principal por referencia	10	TDT1	[M][IJ][JMG][BS][GIT]
TDT6	Crear test buscador principal por barcode	10	TDT1	[M][IJ][JMG][BS][GIT]
TDT7	Crear test Proceso Inventario	15	TDT1	[M][IJ][JMG][BS][GIT]
TDT8	Crear test Proceso Traspasos	15	TDT1	[M][IJ][JMG][BS][GIT]
TDT9	Crear test acceder Training	7	TDT1	[M][IJ][JMG][BS][GIT]
TDT10	Crear test Compra Online	15	TDT1	[M][IJ][JMG][BS][GIT]
TDT11	Crear test Cambiar Idioma	15	TDT1	[M][IJ][JMG][BS][GIT]
TDT12	Crear test Buscar por Tópicos	15	TDT1	[M][IJ][JMG][BS][GIT]
TDT13	Crear test Operaciones y Análisis	12	TDT1	[M][IJ][JMG][BS][GIT]
TDT14	Crear test Proceso Confirmaciones	15	TDT1	[M][IJ][JMG][BS][GIT]
TAD1	Presentar Proyecto a los Android Developers de ADA+	7	-	[M][JMG][AD][IJ][BS][O][GC]
TAD2	Integrar Fase Inicial con Android Developers de ADA+	15	TAD1	[M][JMG][AD][IJ][BS][O][GC]
TAD3	Integrar Fase final con Android Developers de ADA+	20	TAD2	[M][JMG][AD][IJ][BS][O][GC]

Tabla 7.1: Tabla resumida de las tareas. Fuente: Propia

En este diagrama, diagrama 1, podemos ver cómo se vería reflejado el plan de ejecución reflejado en un diagrama de Gantt, cabe mencionar que en este diagrama se han excluido las reuniones diarias y semanales, para evitar llenar de información no relacionada con el plan de ejecución, estas tareas son las que más relacionadas están con el producto final. Las tareas no representadas son de la TP1 a la TP5.

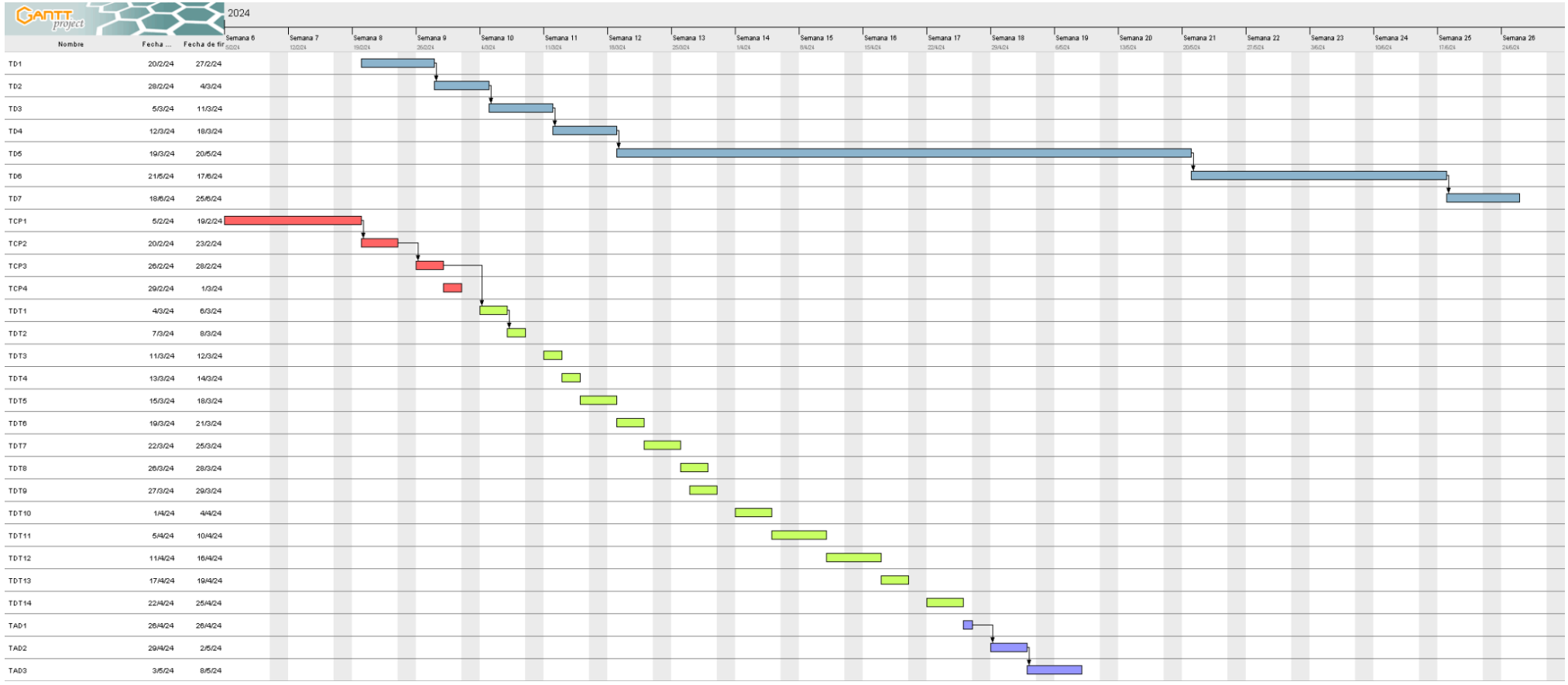


Figura 7.1: Diagrama de Gantt para las tareas propuestas para el desarrollo del proyecto (obviando las reuniones). Fuente: Propia.

8. Ejecución del proyecto

8.1 Reuniones semanales

Como parte del proceso de desarrollo y prueba de la aplicación ADA+, se llevan a cabo varias reuniones regulares para garantizar una comunicación efectiva y un progreso constante del proyecto.

Cada semana, se realiza una reunión con el propietario del producto de ADA+ para discutir y analizar los tests que se desean recrear. Durante estas reuniones, se priorizan las pruebas según su urgencia y se planifica su implementación. Este diálogo constante asegura que el equipo de desarrollo esté alineado con las expectativas y necesidades del propietario del producto.

Además, se llevan a cabo reuniones semanales con Federico Pérez Marina para presentar el avance del proyecto. Estas reuniones son una oportunidad para discutir cualquier bloqueo que pueda haber surgido, así como para presentar nuevas implementaciones que se hayan realizado. La retroalimentación de Federico es invaluable para mantener el proyecto en el camino correcto.

También se realizan reuniones semanales con el equipo de desarrollo para comenzar a introducir el proyecto a los desarrolladores de Android de ADA+. Estas sesiones son esenciales para garantizar que todos los miembros del equipo estén familiarizados con el proyecto y puedan contribuir de manera efectiva a su progreso.

Finalmente, se realizan reuniones mensuales presenciales con Matías para compartir el estado actual del proyecto. Durante estas reuniones, se discuten los "next steps" y se reciben consejos sobre cómo continuar con el desarrollo. Estas reuniones son cruciales para mantener una visión clara del proyecto y para asegurar que se esté avanzando de manera efectiva hacia los objetivos establecidos.

En resumen, estas reuniones regulares juegan un papel vital en el desarrollo y prueba de la aplicación ADA+, permitiendo una comunicación efectiva entre todos los involucrados y asegurando que el proyecto avance de manera constante y eficiente.

8.2 Retrospectiva de los sprints

8.2.1 Retrospectiva del primer sprint

El primer sprint de nuestro proyecto comenzó el 5 de febrero de 2024 y estaba programado para finalizar el 2 de marzo del mismo año. Sin embargo, debido a varios desafíos y bloqueos que surgieron durante el proceso, la fecha de finalización real se extendió hasta el 8 de marzo.

Uno de los principales desafíos que enfrentamos fue la creación y configuración del proyecto. Subestimamos el tiempo que requeriría la investigación sobre los lenguajes de programación, los frameworks, los patrones de diseño y los tipos de trabajos de Jenkins que deseábamos utilizar. Además, tuvimos que dedicar tiempo a diseñar las capacidades de las pruebas, lo que también contribuyó a la extensión del sprint.

Nos encontramos con un bloqueo significativo con la tarea TCP1, que requería una reunión con BrowserStack para solicitar ayuda y asesoramiento sobre nuestra situación. Esta reunión fue crucial para tomar decisiones importantes, como la elección entre Maven y Gradle para la gestión del proyecto. Finalmente, optamos por Maven, ya que BrowserStack solo brindaba soporte para Maven en ese momento.

También tuvimos que tomar una decisión entre utilizar Appium o Espresso para la automatización de pruebas. Optamos por Appium debido a su mayor flexibilidad entre plataformas y la posibilidad de extraer las pruebas end-to-end desde otro proyecto, algo que no era posible con Espresso ya que requiere que las pruebas estén en el mismo proyecto.

Dada la magnitud de las tareas y los desafíos que enfrentamos, decidimos extender el sprint una semana adicional para completar todas las tareas. Gracias a esta extensión, pudimos finalizar todas las tareas necesarias para configurar el proyecto. Aunque el sprint se alargó más de lo previsto, esta experiencia nos proporcionó valiosas lecciones para los futuros sprints y nos permitió establecer una base sólida para el proyecto.

8.2.2 Retrospectiva del segundo sprint

El segundo sprint de nuestro proyecto estaba programado para comenzar el 2 de marzo de 2024, pero debido al retraso en el primer sprint, el inicio real fue el 8 de marzo. La fecha de finalización teórica era el 26 de abril, pero al igual que en el primer sprint, tuvimos que extenderlo hasta el 3 de mayo.

La decisión de retrasar el inicio del segundo sprint se tomó en consulta con el propietario del producto, ya que no queríamos omitir ninguna prueba sobre la funcionalidad de la aplicación. Estas pruebas aportan tanto valor que decidimos alargar el sprint una semana más para asegurarnos de que todas se completaron correctamente.

Durante este sprint, generamos con éxito varios tests para cada funcionalidad, asegurando así todas las posibles historias de usuario típicas de ADA+. Comenzamos con los tests de inicio de sesión y continuamos en función de la importancia de cada funcionalidad y la dificultad de cada test.

Un desafío que enfrentamos fue la programación del "swipe" o deslizamiento, una función humana que realizamos deslizando el dedo sobre la pantalla. Esta función resultó ser un desafío ya que Appium está diseñado principalmente para ejecutar "clicks" o enviar textos a formularios. Sin embargo, logramos superar este obstáculo y completar con éxito todas las tareas de tests.

Además, a petición del propietario del producto, creamos más tests para abordar una incidencia real que ocurrió en la tienda: la creación de dos traspasos del mismo tipo hacia la misma tienda podía llegar a corromper datos. Por lo tanto, decidimos añadir un test para controlar esta casuística.

Durante la reunión con mi tutor, Matías Martínez, le mostré los tests creados y discutimos los próximos pasos. Reflexionamos sobre cómo implementar estos tests en la integración y despliegue continuos (CI/CD) y cómo asegurar el desarrollo de las nuevas funcionalidades de ADA+. Esta reunión fue crucial para planificar el camino a seguir y garantizar el éxito de los futuros sprints.

8.2.3 Retrospectiva del tercer sprint

El tercer sprint de nuestro proyecto estaba programado para comenzar el 26 de abril de 2024, pero debido a los retrasos anteriores, el inicio real fue el 3 de mayo. La fecha de finalización teórica era el 9 de mayo, pero al igual que en los sprints anteriores, tuvimos que extenderlo hasta el 24 de mayo.

Subestimamos este sprint pensando que sería más fácil generar y explicar la documentación. Además, no consideramos la necesidad de sincronizar con las entregas de nuevas funcionalidades de ADA+, especialmente teniendo en cuenta que se plantean grandes entregas antes del verano, una temporada de alta rentabilidad para Mango. Por lo tanto, las formaciones se realizaron cuando fue posible y, dado que mi papel era supervisar los tests, decidimos posponer estas tareas.

A pesar de los desafíos, durante este sprint pudimos ver el impacto positivo de los tests. El 23 de mayo de 2024, a las 00:00, se realizó una migración de Mongo 5.0 a Mongo 7.0. A pesar de los riesgos asociados, en lugar de tener a todo el equipo de Android despierto, decidimos ejecutar los tests desde el trabajo de Jenkins, evitando así tener a personas despiertas innecesariamente.

Este sprint concluyó con buenas noticias y comenzamos a ver el potencial de este proyecto. Decidimos que todas las solicitudes de extracción de nuevas funcionalidades de ADA+ que van a la rama "develop" ejecuten automáticamente los tests. Sin embargo, estos tests no son bloqueantes, ya que puede haber falsos positivos.

En resumen, aunque este sprint presentó desafíos y requirió una extensión, también nos permitió ver el valor y el potencial de los tests automatizados en nuestro proyecto. Estas lecciones aprendidas serán valiosas para los futuros sprints y para el desarrollo continuo de ADA+.

link público de los test ejecutados de la migración de prueba ejecutada en preproducción, y su resultado:
https://app-automate.browserstack.com/dashboard/v2/public-build/UjJIVnFVYlc1ZFZhWmJrSmszYWZMRFlsdUMzRVdFU285ZDFEeUtIK3FlRnhEam00VW1oZE1zSW55d2FCMTRvdDFqMVpmNHJ2ZHJjWmV6ajhMcXdLN1E9PS0tSU41NXZJczlBTzBkR3RxMmpwRTVvQT09--147610eefbe9e3d9c1e90415fa9422e180a4c8b3
Si consultamos el link, podemos ver que durante el 30 de abril, el equipo de backend de ADA+ Performance probó de migrar de mongo con la versión 5.0 a la versión 7.0.

8.3 Línea temporal real

A continuación se muestra en la figura 8.1 en formato de un diagrama de Gantt la ejecución real de las tareas y cómo estas se han ejecutado realmente. Esta figura plasma lo comentado en el apartado anterior de las retrospectivas, donde podemos comparar con el original y como el retrasado sufrido en el primer sprint a afectado al resto de tareas.

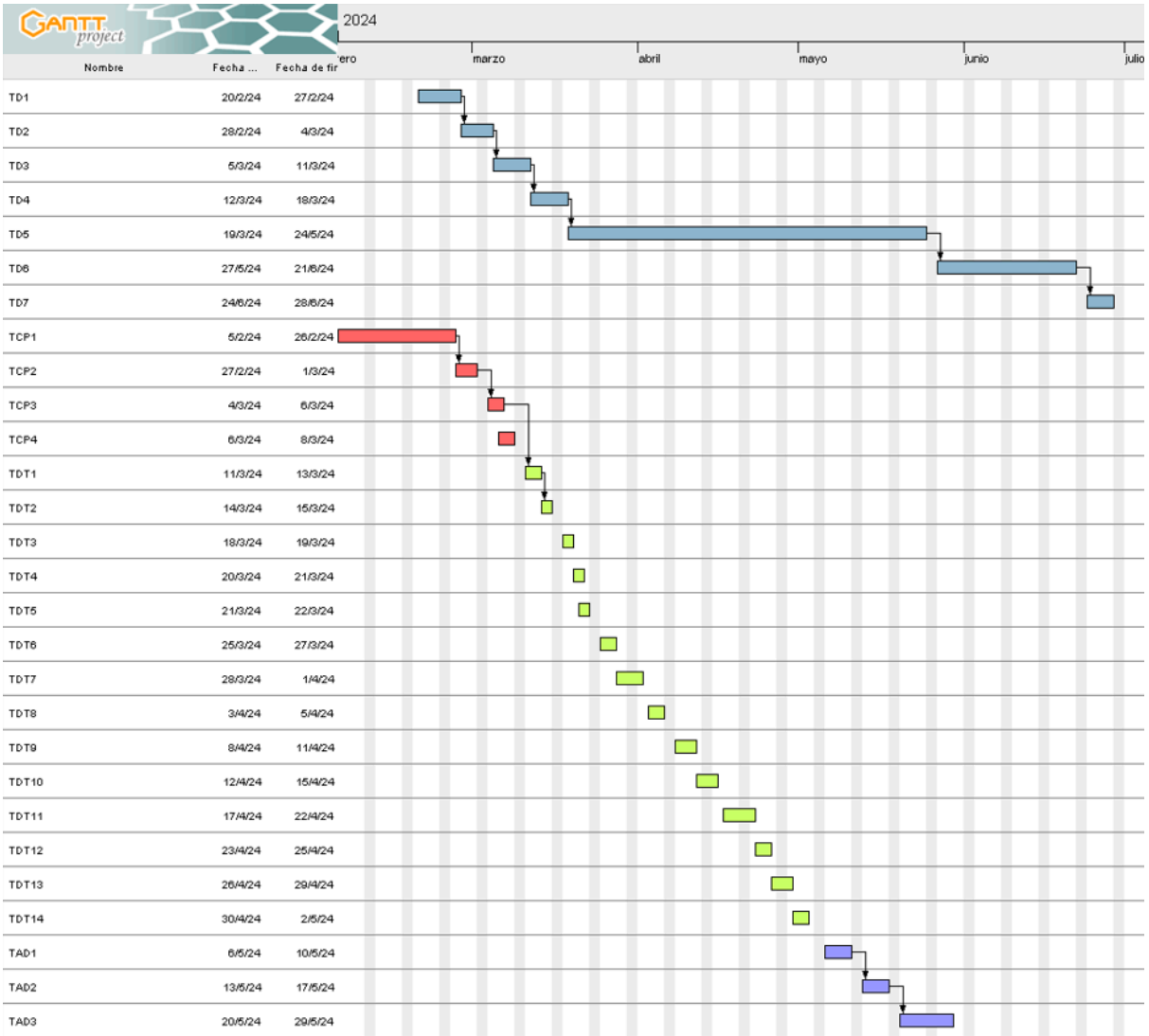


Figura 8.3.1: Ejecución real en un diagrama de Gantt de las tareas a desarrollar del proyecto (obviando las reuniones). Fuente: Propia.

9. Integración de los conocimientos

En este proyecto, hemos integrado una variedad de conocimientos y herramientas para desarrollar un sistema de pruebas sólido y eficiente. A continuación, se detallan algunos de los elementos clave:

Patrón Robot: El patrón Robot es una estrategia de diseño de pruebas que fomenta la reutilización del código y mejora la legibilidad de las pruebas. En este patrón, cada "robot" representa una pantalla o flujo de la aplicación y contiene métodos que replican las acciones del usuario. Esto nos permite escribir pruebas de alto nivel que son fáciles de leer y mantener, mejorando la eficiencia de nuestro proceso de pruebas.

Kotlin: Hemos utilizado Kotlin, un lenguaje de programación moderno y conciso, para desarrollar nuestras pruebas. Su interoperabilidad con Java y su soporte nativo para pruebas unitarias y de integración lo hacen ideal para nuestro proyecto. Kotlin nos ha permitido escribir código limpio y eficiente, lo que ha mejorado la calidad de nuestras pruebas y ha facilitado su mantenimiento.

Appium: Appium es una herramienta de pruebas de automatización de código abierto que nos permite escribir pruebas end-to-end para aplicaciones Android. Con Appium, podemos simular acciones del usuario en la aplicación real en un dispositivo o emulador. Esto nos ha permitido validar que nuestra aplicación funciona correctamente en condiciones de la vida real.

JUnit 5: Hemos utilizado JUnit 5, la última versión del popular marco de pruebas unitarias para Java, para estructurar nuestras pruebas. JUnit 5 ofrece aserciones potentes y flexibles, así como características como pruebas parametrizadas y soporte para pruebas concurrentes. Esto nos ha permitido escribir pruebas más robustas y completas.

Maven: Maven es una herramienta de gestión de proyectos de software que hemos utilizado para compilar y gestionar nuestro proyecto. Maven nos ha permitido gestionar las dependencias de nuestro proyecto, construir nuestro código de manera eficiente y distribuir nuestro software de manera efectiva.

Diseño de pruebas End-to-End: Las pruebas End-to-End son fundamentales para asegurar que todas las partes de nuestra aplicación trabajan juntas correctamente. Hemos diseñado nuestras pruebas para cubrir los flujos de usuario más importantes, asegurándonos de que la aplicación funciona como se espera en condiciones de la vida real.

Presentación del proyecto al equipo: Antes de comenzar el desarrollo, presentamos el proyecto al equipo para asegurarnos de que todos comprendieran los objetivos y las expectativas. Esta presentación incluyó una descripción detallada de las herramientas y técnicas que íbamos a utilizar, así como una discusión sobre los desafíos potenciales y cómo podríamos superarlos.

Investigación de los dispositivos usados en tienda: Como parte de nuestro proceso de pruebas, investigamos los dispositivos que se utilizan en las tiendas de Mango. Descubrimos que el Samsung A52 es uno de los dispositivos más utilizados, por lo que nos aseguramos de que nuestras pruebas se ejecutarán correctamente en este dispositivo. Esto nos ayudó a garantizar que nuestra aplicación funcionará correctamente en el entorno en el que se utilizará.

Conclusión: La integración de estos conocimientos y herramientas ha sido fundamental para

el éxito de nuestro proyecto. Hemos creado un conjunto de pruebas robusto y mantenible que garantiza la calidad de nuestra aplicación. A través de este proceso, hemos mejorado nuestra capacidad para trabajar en equipo, resolver problemas y adaptarnos a nuevos desafíos. Este proyecto ha sido una oportunidad valiosa para aplicar y profundizar nuestros conocimientos en un contexto práctico, y estamos orgullosos de los resultados que hemos logrado.

10. Legislación y cumplimiento regulatorio

El propósito de esta sección es establecer los aspectos legales y regulatorios que el proyecto de final de grado, realizado en colaboración con Mango, debe cumplir. Este marco legal y regulatorio es esencial para garantizar que el proyecto se adhiere a todas las leyes y regulaciones aplicables, minimizando así el riesgo de infracciones y sanciones.

10.1 Marco legal

El proyecto se desarrolla dentro de un marco legal específico que abarca diversas áreas, incluyendo la protección de datos, la propiedad intelectual y la seguridad de la información.
Este marco legal puede variar dependiendo de la jurisdicción en la que se implemente el proyecto, por lo que es crucial comprender y adherirse a las leyes y regulaciones locales e internacionales correspondientes.

10.2 Cumplimiento regulatorio

El cumplimiento regulatorio implica asegurar que el proyecto se adhiere a todas las leyes, regulaciones, directrices y especificaciones relevantes para su operación.

Esto puede incluir normativas específicas de la industria, como las normas de seguridad de datos para la industria de tarjetas de pago (PCI DSS) si el proyecto implica el procesamiento de pagos con tarjeta de crédito, o las regulaciones de privacidad y protección de datos como el Reglamento General de Protección de Datos (RGPD) de la Unión Europea si el proyecto implica el procesamiento de datos personales.

10.3 Cumplimiento regulatorio

La gestión de riesgos legales implica la identificación, evaluación y mitigación de los riesgos legales asociados con el proyecto.

Esto puede implicar la realización de auditorías legales y de cumplimiento, la implementación de políticas y procedimientos para garantizar el cumplimiento continuo, y la formación y educación del personal sobre sus responsabilidades legales y de cumplimiento.

10.4 Responsabilidad y obligaciones

Es importante que todas las partes involucradas en el proyecto comprendan y cumplan con sus responsabilidades y obligaciones legales y de cumplimiento. Esto incluye a los desarrolladores, los operadores del sistema, los usuarios finales y cualquier otra parte que interactúe con el proyecto.

En conclusión, la sección de legislación y cumplimiento regulatorio es un componente esencial del proyecto, diseñado para garantizar que todas las actividades se realicen de acuerdo con las leyes y regulaciones aplicables, y para minimizar el riesgo de infracciones legales y sanciones regulatorias.

11. Conocimientos técnicos del proyecto

El desarrollo de este proyecto ha requerido una amplia gama de conocimientos técnicos. A continuación, se detallan algunos de los más relevantes:

Kotlin: Kotlin es un lenguaje de programación moderno y conciso que hemos utilizado para desarrollar nuestras pruebas. Su interoperabilidad con Java y su soporte nativo para pruebas unitarias y de integración lo hacen ideal para nuestro proyecto. Kotlin nos ha permitido escribir código limpio y eficiente, lo que ha mejorado la calidad de nuestras pruebas y ha facilitado su mantenimiento.

Appium: Appium es una herramienta de pruebas de automatización de código abierto que nos permite escribir pruebas end-to-end para aplicaciones Android. Con Appium, podemos simular acciones del usuario en la aplicación real en un dispositivo o emulador. Esto nos ha permitido validar que nuestra aplicación funciona correctamente en condiciones de la vida real.

JUnit 5: Hemos utilizado JUnit 5, la última versión del popular marco de pruebas unitarias para Java, para estructurar nuestras pruebas. JUnit 5 ofrece aserciones potentes y flexibles, así como características como pruebas parametrizadas y soporte para pruebas concurrentes. Esto nos ha permitido escribir pruebas más robustas y completas.

Maven: Maven es una herramienta de gestión de proyectos de software que hemos utilizado para compilar y gestionar nuestro proyecto. Maven nos ha permitido gestionar las dependencias de nuestro proyecto, construir nuestro código de manera eficiente y distribuir nuestro software de manera efectiva.

Patrón Robot: El patrón Robot es una estrategia de diseño de pruebas que fomenta la reutilización del código y mejora la legibilidad de las pruebas. En este patrón, cada "robot" representa una pantalla o flujo de la aplicación y contiene métodos que replican las acciones del usuario. Esto nos permite escribir pruebas de alto nivel que son fáciles de leer y mantener, mejorando la eficiencia de nuestro proceso de pruebas.
Además de estos conocimientos técnicos, también hemos aplicado habilidades de gestión de proyectos, como la planificación y la gestión del tiempo, para asegurarnos de que el proyecto se completara de manera eficiente y efectiva. También hemos trabajado en equipo, colaborando y comunicándonos de manera efectiva para resolver problemas y tomar decisiones.

12. Ejecución práctica del proyecto

Durante el desarrollo de este proyecto, hemos implementado varias estrategias de ejecución de pruebas, cada una con sus propias ventajas y desventajas. Estas estrategias se seleccionaron para adaptarse a las necesidades cambiantes del proyecto en diferentes etapas.

12.1 Ejecución local

La ejecución local se refiere a la ejecución de pruebas en un dispositivo móvil que está físicamente conectado al ordenador. Esta estrategia es particularmente útil cuando se necesita ejecutar un número limitado de pruebas, ya que es más rápida que otras opciones. Sin embargo, tiene algunas limitaciones. Por un lado, depende de la disponibilidad de un dispositivo móvil específico, que puede no ser el mismo modelo que se utiliza en las tiendas de Mango. Además, la ejecución local no permite la paralelización de las pruebas, lo que significa que solo se puede ejecutar una prueba a la vez.

12.2 Ejecución en browserstack

La ejecución en BrowserStack implica ejecutar las pruebas en dispositivos móviles virtuales proporcionados por el servicio de BrowserStack. Esta opción tiene la ventaja de permitir la paralelización de las pruebas, lo que significa que se pueden ejecutar varias pruebas al mismo tiempo. Además, no requiere que el ordenador esté disponible durante la ejecución de las pruebas, lo que libera recursos para otras tareas. Sin embargo, esta opción puede ser más lenta que la ejecución local cuando solo se necesitan ejecutar unas pocas pruebas. Además, requiere una conexión a internet estable, lo que puede ser un problema en algunos entornos.

12.3 Ejecución desde job de jenkins

Esta estrategia implica ejecutar las pruebas desde un job de Jenkins, que se ejecuta dentro de un contenedor de Kubernetes. Esto permite ejecutar todas las pruebas sin necesidad de tener el proyecto instalado en el ordenador, lo que facilita la gestión de las dependencias y libera recursos del sistema. Al igual que la ejecución en BrowserStack, esta opción permite la paralelización de las pruebas y no requiere que el ordenador esté disponible durante la ejecución de las pruebas. Sin embargo, puede ser más lento que la ejecución local cuando solo se necesitan ejecutar unas pocas pruebas. Además, requiere una conexión a internet estable.

12.4 Ejecución desde bitbucket

La ejecución desde Bitbucket es una estrategia automatizada que se configura utilizando Jenkins. En este proceso, se genera una APK de desarrollo utilizando un script en Python, y luego se ejecutan las pruebas en esta APK, que se actualiza con los últimos commits de desarrollo. Esta estrategia tiene varias ventajas. Por un lado, permite un mayor control sobre el desarrollo de nuevo código, ya que se puede verificar que los cambios recientes no afecten a las funcionalidades existentes. Además, al igual que las otras estrategias, permite la paralelización de las pruebas y no requiere que el ordenador esté disponible durante la ejecución de las pruebas. Sin embargo, puede ser más lento que la ejecución local cuando solo se necesitan ejecutar unas pocas pruebas y también requiere una conexión a internet estable.

13. Implementación de los test end to end

La implementación de pruebas End2End (E2E) es un componente esencial en el desarrollo de software. Estas pruebas se realizan para asegurar que el flujo de un sistema funciona como se espera desde el inicio hasta el final, garantizando que todos los componentes integrados funcionen correctamente juntos. En Mango, nos esforzamos por mantener la más alta calidad y funcionalidad en todos nuestros sistemas y productos. Por lo tanto, la implementación de pruebas E2E es una parte integral de nuestras operaciones.

Las pruebas E2E implican simular escenarios de la vida real y verificar que el sistema completo funcione correctamente. Esto incluye la interacción con la base de datos, la comunicación entre diferentes servicios, el rendimiento de la interfaz de usuario y la experiencia general del usuario. Estas pruebas son fundamentales para garantizar que nuestros clientes tengan una experiencia fluida y sin problemas al utilizar nuestros productos y servicios.

A continuación, describiremos algunos de los archivos y herramientas que utilizamos en la implementación de nuestras pruebas E2E, detallando su funcionalidad, calidad y cómo contribuyen al éxito general de nuestras operaciones.

13.1 Calidad de código

Para garantizar la calidad del código en este proyecto, se han implementado varias prácticas y herramientas clave. Estas medidas han sido esenciales para asegurar que el código sea limpio, eficiente y mantenible a lo largo del desarrollo.

Uso de ESLint
ESLint ha sido una herramienta fundamental en el proceso de desarrollo, utilizada para identificar y corregir problemas en el código Kotlin. Al integrar ESLint en el flujo de trabajo, se han establecido reglas de codificación claras y consistentes, lo que ha permitido detectar errores comunes y malas prácticas de forma temprana. Esto no solo ha mejorado la calidad del código, sino que también ha facilitado su lectura y mantenimiento por parte de todo el equipo.

Implementación de Detekt
Para el código Kotlin, se ha utilizado Detekt, una herramienta de análisis estático que ha ayudado a identificar problemas potenciales y mejorar la calidad del código. Detekt ha sido configurado para seguir las mejores prácticas y estándares de codificación, permitiendo detectar inconsistencias, posibles errores y áreas de mejora. Gracias a Detekt, el código Kotlin del proyecto es más robusto y confiable.

Integración de SonarQube
SonarQube ha sido otra herramienta crucial en la estrategia de aseguramiento de la calidad del código. Esta plataforma de análisis de código ha proporcionado un análisis continuo de la base de código, identificando problemas de seguridad, bugs y deuda técnica en múltiples lenguajes de programación. SonarQube ha permitido obtener una visión integral de la calidad del código, generando informes detallados y métricas que han ayudado a priorizar y corregir problemas de manera efectiva.

Reuniones semanales de revisión de código
Además del uso de herramientas automatizadas, se han llevado a cabo reuniones semanales para discutir y revisar las prácticas de programación utilizadas en el proyecto.

Durante estas reuniones, los miembros del equipo han compartido sus experiencias, sugerencias y mejores prácticas, fomentando un ambiente de aprendizaje continuo y colaboración. Estas discusiones han sido cruciales para alinear al equipo en cuanto a los estándares de codificación y resolver cualquier duda o problema que haya surgido durante el desarrollo.

Estas reuniones también han servido para revisar el código de manera conjunta, lo que ha permitido identificar y corregir problemas de manera colaborativa. Este enfoque ha asegurado que todos los desarrolladores estén alineados en cuanto a las expectativas de calidad y las metodologías de trabajo, promoviendo un código más coherente y de alta calidad.

En resumen, la combinación del uso de ESLint, Detekt, SonarQube y las reuniones semanales de revisión de código ha sido instrumental para mantener y mejorar la calidad del código en este proyecto. Estas prácticas han asegurado que el código sea consistente, libre de errores y fácil de mantener, lo que a su vez ha contribuido significativamente al éxito del proyecto.

13.1 Jenkins

```
import com.mango.jenkins.main.PodContainer

pipeline {
     agent {
        label "k8s-maven-17"
     }

    tools {
        maven 'Maven'
    }

    stages {
        stage('Run tests'){
            steps {
                script{
                    container(PodContainer.maven) {
                        sh 'mvn -v'
                        sh 'java -version'
                        sh 'mvn clean test'
                    }
                }
            }
        }
    }

    post {
        always {
            cleanWs()
            dir("${workspace}@tmp") {
                deleteDir()
            }
        }
    }
```

Figura 13.1.1: Captura del archivo de configuración de Job de Jenkins. Fuente: Propia.

Este archivo es un script de Jenkins, una herramienta de integración continua y entrega continua (CI/CD), que se utiliza para automatizar diferentes fases del ciclo de vida del desarrollo de software.

El script define un pipeline, que es una serie de pasos que se deben seguir para llevar a cabo un proceso de construcción (build), prueba (test) y despliegue (deploy).

En este caso, el script está configurado para ejecutar pruebas en un entorno específico, identificado por la etiqueta "k8s-maven-17". Utiliza la herramienta Maven, una herramienta de gestión de proyectos y comprensión de software.

El pipeline consta de varias etapas. En la etapa 'Run tests', se ejecutan las pruebas utilizando Maven dentro de un contenedor específico (PodContainer.maven). Los comandos 'mvn -v' y 'java -version' se utilizan para verificar las versiones de Maven y Java, respectivamente. El comando 'mvn clean test' se utiliza para limpiar el proyecto, compilarlo, y luego ejecutar cualquier prueba unitaria definida.

Finalmente, la sección 'post' define acciones que siempre se deben realizar después de que se complete el pipeline, independientemente de si fue exitoso o no. En este caso, se limpia el espacio de trabajo (workspace) y se elimina el directorio temporal.

Este script es un ejemplo de cómo Jenkins puede automatizar el proceso de pruebas, lo que ayuda a mejorar la eficiencia y la calidad del software.

13.2 Tests sobre funcionalidades

Este apartado explicará por subapartados las diferentes funcionalidades testeadas y que tipos de test se han diseñado para estas.

13.2.1 LogIn

```
package com.mango.rfid.browserstack.test.login

import ...

03932096
class LogInTest : BrowserStackJUnitTest() {

    03932096
    @Test
    fun `Login and onboarding with store email`() {
        login(wait)
        onboarding(wait)
        val destinationTitle = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id(TITLEMENU)))
        Assertions.assertEquals(
            "Tienda RFID Formacion",
            destinationTitle.text,
            "El test no llegó a la pantalla de destino."
        )
    }

    03932096
    @Test
    fun `Login failure with user and password incorrect`() {
        login(wait,"usuario_incorrecto", "contraseña_incorrecta")
        val destinationTitle = wait.until(ExpectedConditions.visibilityOfElementLocated(By.ByXPath(TEXT_ERROR)))
        Assertions.assertEquals(
            "Incorrect username and/or password",
            destinationTitle.text,
            "El test no llegó a la pantalla de destino."
        )
    }
}
```

Figura 13.2.1.1: Captura del archivo LogInTest. Fuente: Propia.

A continuación, tenemos el archivo LogInTest. Este archivo define dos pruebas unitarias para la funcionalidad de inicio de sesión. La primera prueba, Login and onboarding with store email, verifica que un usuario pueda iniciar sesión y completar el proceso de incorporación con un correo electrónico de tienda válido. La segunda prueba, Login failure with user and password incorrect, verifica que el inicio de sesión falle cuando se proporciona un nombre de usuario y una contraseña incorrectos.

```
package com.mango.rfid.browserstack.test.login

import com.mango.rfid.browserstack.ENVIRONMENT

const val USERNAME_FIELD =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val PASSWORD_FIELD =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val TEXT_ERROR =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val LOGIN_BUTTON =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val CONTINUE_BUTTON =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val CONTINUE2_BUTTON =
        "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget
const val XPATHEXPRESSION = "//android.view.ViewGroup[.//*[contains(@text, Continue)]]"
val LOGIN_BUTTON_ID = "$ENVIRONMENT/login"
const val OneSecond :Long = 1000
const val FiveSeconds :Long = 5000
```

Figura 13.2.1.2 Captura del archivo LogInElements. Fuente: Propia.

El archivo LoginElements define constantes para varios elementos de la interfaz de usuario

en la pantalla de inicio de sesión, como los campos de nombre de usuario y contraseña, el botón de inicio de sesión y los mensajes de error. Estas constantes se utilizan para interactuar con estos elementos durante las pruebas.

```
1   package com.mango.rfid.browserstack.robots
2
3   import ...
7
    03932096
8   fun login(wait: WebDriverWait, username: String = SHOP_EMAIL, password: String = PASSWORD) {
9
10      wait.sendElementXPATH(USERNAME_FIELD,username)
11      wait.sendElementXPATH(PASSWORD_FIELD,password)
12      wait.clickElementID(LOGIN_BUTTON_ID)
13      Thread.sleep(FiveSeconds)
14  }
15
    03932096 *
16  fun onboarding(wait: WebDriverWait){
17      wait.clickElementXPATH(CONTINUE_BUTTON)
18      wait.clickElementXPATH(CONTINUE_BUTTON)
19      wait.clickElementXPATH(CONTINUE_BUTTON)
20      wait.clickElementXPATH(CONTINUE2_BUTTON)
21      wait.clickElementXPATH(CONTINUE_BUTTON)
22      wait.clickElementXPATH(CONTINUE_BUTTON)
23  }
24
    03932096
25  fun loginAndOnboarding (wait: WebDriverWait) {
26      login(wait)
27      try {
28          onboarding(wait)
29      } catch (e: Exception){
30          e.printStackTrace()
31      }
32  }
```

Figura 13.2.1.3: Captura del archivo LogInRobots. Fuente: Propia.

Finalmente, el archivo LogInRobots define varias funciones que simulan acciones del usuario, como iniciar sesión y completar el proceso de incorporación. Estas funciones utilizan las constantes definidas en LoginElements para interactuar con los elementos de la interfaz de usuario. También utilizan la clase WebDriverWait para pausar la ejecución de la prueba hasta que ciertas condiciones se cumplan, como la aparición de un elemento de la interfaz de usuario.

Estos archivos demuestran cómo se pueden automatizar las pruebas de la funcionalidad de inicio de sesión. Al automatizar estas pruebas, se puede aumentar la eficiencia del proceso de prueba y reducir la posibilidad de errores humanos. Además, estas pruebas proporcionan una forma de verificar que la funcionalidad de inicio de sesión continúa funcionando como se espera a medida que se realizan cambios en la aplicación.

13.2.2 SearchBar and SearchTopics

Estos archivos son parte de un conjunto de pruebas automatizadas para la funcionalidad de la barra de búsqueda de una aplicación. Se utilizan para validar que la barra de búsqueda funcione correctamente y devuelva los resultados esperados.

```
1   package com.mango.rfid.browserstack.test.searchBar
2
3   import ...
11
    03932096
12  class SearchBarTest : BrowserStackJUnitTest() {
13
        03932096
14      @Test
15      fun `Search for Women's clothes`() {
16          loginAndOnboarding(wait)
17          wait.clickElementID(SEARCH_FIELD)
18          searchClothes(driver, clothes: "Camisa")
19          val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
20          Assertions.assertTrue(searchResultsList.isDisplayed)
21      }
        03932096
22      @Test
23      fun `Search for Men's clothes`() {
24          loginAndOnboarding(wait)
25          wait.clickElementID(SEARCH_FIELD)
26          selectMan(wait)
27          wait.clickElementID(SEARCH_LEN)
28          searchClothes(driver, clothes: "Pantalón")
29          val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
30          Assertions.assertTrue(searchResultsList.isDisplayed)
31      }
```

```
        03932096
32      @Test
33      fun `Search for Kid's clothes`() {
34          loginAndOnboarding(wait)
35          wait.clickElementID(SEARCH_FIELD)
36          selectKid(wait)
37          wait.clickElementID(SEARCH_LEN)
38          searchClothes(driver, clothes: "Camisetas")
39          val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
40          Assertions.assertTrue(searchResultsList.isDisplayed)
41      }
        03932096
42      @Test
43      fun `Search for Teen's clothes`() {
44          loginAndOnboarding(wait)
45          wait.clickElementID(SEARCH_FIELD)
46          selectTeen(wait)
47          wait.clickElementID(SEARCH_LEN)
48          searchClothes(driver, clothes: "Camisetas")
49          val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
50          Assertions.assertTrue(searchResultsList.isDisplayed)
51      }
        03932096
52      @Test
53      fun `Search for Home items`() {
54          loginAndOnboarding(wait)
55          wait.clickElementID(SEARCH_FIELD)
56          selectHome(wait)
57          wait.clickElementID(SEARCH_LEN)
58          searchClothes(driver, clothes: "Cojín")
59          val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
60          Assertions.assertTrue(searchResultsList.isDisplayed)
61      }
```

```
03932096
@Test
fun `Search for Women's clothes by reference number`() {
    loginAndOnboarding(wait)
    wait.clickElementID(SEARCH_FIELD)
    searchClothes(driver, clothes: "27043265")
    wait.clickElementID(START_BUTTON)
    val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.id(PRODUCT_NAME)))
    Assertions.assertEquals(
        expected: "Jeans philipa",
        searchResultsList.text,
        message: "El test no llegó a la pantalla de destino."
    )
}
03932096
@Test
fun `Search for Men's clothes by reference number`() {
    loginAndOnboarding(wait)
    wait.clickElementID(SEARCH_FIELD)
    selectMan(wait)
    wait.clickElementID(SEARCH_LEN)
    searchClothes(driver, clothes: "47040809")
    wait.clickElementID(START_BUTTON)
    val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.id(PRODUCT_NAME)))
    Assertions.assertEquals(
        expected: "Jersey antigua",
        searchResultsList.text,
        message: "El test no llegó a la pantalla de destino."
    )
}
```

```
@Test
fun `Search by BarCode`(){
    loginAndOnboarding(wait)
    wait.clickElementID(CAMERA_BUTTON)
    wait.clickElementID(ALLOW_BUTTON)
    wait.clickElementID(ENTER_CODE_BUTTON)
    wait.sendElementID(ENTER_NUM_CODE_FIELD, key: "8445791741848")
    driver.pressKey(KeyEvent(AndroidKey.ENTER))
    wait.clickElementID(CONTINUE_BUTTON)
    val searchResultsList : WebElement! = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(RESULT_DISPLAY)))
    Assertions.assertTrue(searchResultsList.isDisplayed)
}
}
```

Figura 13.2.2.1: Capturas del archivo SearchBarTests. Fuente: Propia.

Este archivo define varias pruebas unitarias para la funcionalidad de la barra de búsqueda. Cada prueba simula una búsqueda de diferentes categorías de productos (ropa de mujer, ropa de hombre, ropa de niños, ropa de adolescentes, artículos para el hogar) y verifica que los resultados de la búsqueda se muestren correctamente. También hay pruebas que simulan la búsqueda de productos por número de referencia y por código de barras.

```
1   package com.mango.rfid.browserstack.test.searchBar
2
3   import com.mango.rfid.browserstack.ENVIRONMENT
4
5
6   val SEARCH_FIELD = "$ENVIRONMENT/edittext_search_model"
7   val SEARCH_LEN = "$ENVIRONMENT/menu_item_search"
8   const val SEARCH_LIST = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.F
9   val START_BUTTON = "$ENVIRONMENT/btn_open_nearby_stores_activity"
10  val PRODUCT_NAME = "$ENVIRONMENT/product_name"
11  val CAMERA_BUTTON = "$ENVIRONMENT/fbActionScanCamera"
12  const val ALLOW_BUTTON = "com.android.permissioncontroller:id/permission_allow_foreground_only_button"
13  val ENTER_CODE_BUTTON = "$ENVIRONMENT/contentMangoButton"
14  val ENTER_NUM_CODE_FIELD = "$ENVIRONMENT/barcodeInputText"
15  val CONTINUE_BUTTON = "$ENVIRONMENT/contentMangoButton"
16  const val RESULT_DISPLAY = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widge
17  val SEARCH_BAR = "$ENVIRONMENT/search_src_text"
18  const val MAN = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayo
19  const val KID = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayo
20  const val TEEN = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLay
21  const val HOME = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLay
```

Figura 13.2.2.2: Capturas del archivo SearchElements. Fuente: Propia.

Este archivo define constantes para varios elementos de la interfaz de usuario en la pantalla de búsqueda, como el campo de búsqueda, el botón de inicio, el nombre del producto, el botón de la cámara y los mensajes de error. Estas constantes se utilizan para interactuar con estos elementos durante las pruebas.

```
1   package com.mango.rfid.browserstack.robots
2
3   import ...
12
    03932096 *
13  fun searchClothes(driver: AndroidDriver, clothes: String) {
14      val wait = WebDriverWait(driver, Duration.ofSeconds(30))
15      wait.sendElementID(SEARCH_BAR,clothes)
16      driver.pressKey(KeyEvent(AndroidKey.ENTER))
17      try {
18          val okButton = wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//*[@text='OK']")))
19          okButton.click()
20      } catch (_: Exception) {
21      }
22  }
    03932096 *
23  fun selectWoman(wait: WebDriverWait){
24      wait.clickElementXPATH(BACK_PRODUCT)
25  }
    03932096 *
26  fun selectMan(wait: WebDriverWait){
27      wait.clickElementXPATH(BACK_PRODUCT)
28      wait.clickElementXPATH(MAN)
29  }
    03932096 *
30  fun selectKid(wait: WebDriverWait){
31      wait.clickElementXPATH(BACK_PRODUCT)
32      wait.clickElementXPATH(KID)
33  }
    03932096 *
34  fun selectTeen(wait: WebDriverWait){
35      wait.clickElementXPATH(BACK_PRODUCT)
36      wait.clickElementXPATH(TEEN)
37  }
```

Figura 13.2.2.3: Capturas del archivo SearchRobots. Fuente: Propia.

Este archivo define varias funciones que simulan acciones del usuario, como buscar ropa y seleccionar diferentes categorías de productos. Estas funciones utilizan las constantes definidas en SearchBarElements para interactuar con los elementos de la interfaz de usuario.

```
1   package com.mango.rfid.browserstack.test.searchBar
2
3   import ...
9
    03932096 *
10  class SearchTopicsTest : BrowserStackJUnitTest() {
        03932096 *
11      @Test
12      fun `Search clothes from New Collection`() {
13
14          loginAndOnboarding(wait)
15          wait.clickElementID(SEARCH_FIELD)
16          selectWoman(wait)
17          selectNOption(wait,3)
18
19          val searchResultsList = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
20          Assertions.assertTrue(searchResultsList.isDisplayed)
21      }
22
        03932096 *
23      @Test
24      fun `Search clothes from Sales`() {
25
26          loginAndOnboarding(wait)
27          wait.clickElementID(SEARCH_FIELD)
28          selectKid(wait)
29          selectNOption(wait,5)
30          selectNOption(wait,4)
31          selectNOption(wait,1)
32
33          val searchResultsList = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(SEARCH_LIST)))
34          Assertions.assertTrue(searchResultsList.isDisplayed)
35      }
36  }
```

Figura 13.2.2.4: Captura del archivo SearchTopicTests. Fuente: Propia.

Este archivo define pruebas unitarias que simulan la búsqueda de ropa de la nueva colección y de las ventas. Estas pruebas utilizan las funciones definidas en SearchRobots para simular las acciones del usuario y verifican que los resultados de la búsqueda se muestren correctamente.

Estos archivos demuestran cómo se pueden automatizar las pruebas de la funcionalidad de la barra de búsqueda. Al automatizar estas pruebas, se puede aumentar la eficiencia del proceso de prueba y reducir la posibilidad de errores humanos. Además, estas pruebas proporcionan una forma de verificar que la funcionalidad de la barra de búsqueda continúa funcionando como se espera a medida que se realizan cambios en la aplicación.

13.2.3 Settings

Estos archivos son parte de un conjunto de pruebas automatizadas para la funcionalidad de configuración de una aplicación. Se utilizan para validar que las opciones de configuración funcionen correctamente y produzcan los resultados esperados.

```
1   package com.mango.rfid.browserstack.test.settings
2
3   import ...
    03932096 *
10  class SettingsandHelpTest : BrowserStackJUnitTest() {
11
        03932096 *
12      @Test
13      fun `Logout from AdaPlus`() {
14
15          loginAndOnboarding(wait)
16          selectNOption(wait,5)
17          selectNOption(wait,4)
18          val destinationTitle = wait.until(ExpectedConditions.visibilityOfElementLocated(By.ByXPath(TITLE)))
19          Assertions.assertEquals(
20              "Log in",
21              destinationTitle.text
22          )
23      }
24
        03932096 *
25      @Test
26      fun `Go to Training Tutorial`() {
27
28          loginAndOnboarding(wait)
29          selectNOption(wait,5)
30          selectNOption(wait,3)
31          val destinationTitle = wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(TITLE_TRAINING)))
32          Assertions.assertEquals(
33              "MANUAL_IN_ ADA+",
34              destinationTitle.text
35          )
36      }
37  }
```

Figura 13.2.3.1: Captura del archivo SettingsAndHelpTests. Fuente: Propia.

Este archivo define dos pruebas unitarias para la funcionalidad de configuración. La primera prueba, `Log Out from AdaPlus`, verifica que un usuario pueda cerrar sesión correctamente. La segunda prueba, `Go to Training Tutorial`, verifica que un usuario pueda acceder al tutorial de formación desde la pantalla de configuración.

```
1   package com.mango.rfid.browserstack.test.settings
2
3   import com.mango.rfid.browserstack.ENVIRONMENT
4
5   const val TITLE = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayo
6   const val TITLE_TRAINING = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.
7
```

Figura 13.2.3.2: Captura del archivo SettingsAndHelpElements. Fuente: Propia.

Este archivo define constantes para varios elementos de la interfaz de usuario en la pantalla de configuración, como los títulos de las pantallas de inicio de sesión y formación. Estas constantes se utilizan para interactuar con estos elementos durante las pruebas.

```
1   package com.mango.rfid.browserstack.robots
2
3   import ...
9
    03932096 *
10  fun openSettings(driver: AndroidDriver, wait: WebDriverWait){
11      driver.openNotifications()
12      wait.clickElementID(SETTINGS_BUTTON)
13      wait.clickElementXPATH(SEARCH_SETTINGS_BAR)
14  }
15
    03932096
16  fun goToLanguage(driver: AndroidDriver, wait: WebDriverWait){
17      wait.sendElementID(SEARCH_SETTINGS_TEXT, "Add language")
18      Thread.sleep(FiveSeconds)
19      driver.pressKey(KeyEvent(AndroidKey.ENTER))
20      Thread.sleep(FiveSeconds)
21      wait.clickElementXPATH(LANGUAGE_SETTINGS_BUTTON)
22  }
23
    03932096 *
24  fun changeLanguages(wait: WebDriverWait){
25      wait.clickElementID(ADD_LANGUAGE_BUTTON)
26      wait.clickElementXPATH(ADD_SPANISH)
27      wait.clickElementID(SET_DEFAULT_BUTTON)
28  }
29
    03932096
30  fun returnToApp(driver: AndroidDriver){
31      driver.pressKey(KeyEvent(AndroidKey.BACK))
32      driver.pressKey(KeyEvent(AndroidKey.BACK))
33      driver.pressKey(KeyEvent(AndroidKey.BACK))
34      driver.pressKey(KeyEvent(AndroidKey.BACK))
35  }
```

Figura 13.2.3.3: Captura del archivo SettingsAndHelpRobots. Fuente: Propia.

Este archivo define varias funciones que simulan acciones del usuario, como abrir la pantalla de configuración, ir a la opción de idioma, cambiar los idiomas y volver a la aplicación. Estas funciones utilizan las constantes definidas en `SettingsElements` para interactuar con los elementos de la interfaz de usuario.

Estos archivos demuestran cómo se pueden automatizar las pruebas de la funcionalidad de configuración. Al automatizar estas pruebas, se puede aumentar la eficiencia del proceso de prueba y reducir la posibilidad de errores humanos. Además, estas pruebas proporcionan

una forma de verificar que la funcionalidad de configuración continúa funcionando como se espera a medida que se realizan cambios en la aplicación.

13.2.4 Product

Estos archivos son parte de un conjunto de pruebas automatizadas para la funcionalidad de producto de una aplicación. Se utilizan para validar que las operaciones de producto, como la modificación de la cantidad de productos en exhibición, funcionen correctamente y produzcan los resultados esperados.

```
1   package com.mango.rfid.browserstack.test.product
2   import ...
    03932096 *
8   class ProductTest : BrowserStackJUnitTest() {
9
        03932096 *
10      @Test
11      fun `Modify and confirm eyes in display product`() {
12          loginAndOnboarding(wait)
13          selectNOption(wait,3)
14          selectNOption(wait,1)
15          searchProductDisplay(driver,wait)
16          plusEyes(wait,1)
17          val resultEyes = wait.until(ExpectedConditions.elementToBeClickable(By.id(UNITS_TO_EXPOSE))).text
18          saveChanges(wait)
19          wait.clickElementID(TOUCH_OUTSIDE)
20          backToMenu(wait)
21          searchProductID(driver,wait)
22          val searchResultsNum = wait.until(ExpectedConditions.presenceOfElementLocated(By.xpath(RESULT_NUMBER_EYES)))
23          Assertions.assertEquals(resultEyes, searchResultsNum.text)
24      }
25
        03932096 *
26      @Test
27      fun `Modify eyes in display product`() {
28          loginAndOnboarding(wait)
29          selectNOption(wait,3)
30          selectNOption(wait,1)
31          searchProductDisplay(driver,wait)
32          plusEyes(wait,1)
33          saveChanges(wait)
34          val confirmSave = wait.until(ExpectedConditions.presenceOfElementLocated(By.id(SUCCESS_IMAGE)))
35          Assertions.assertTrue(confirmSave.isDisplayed)
36      }
37  }
```

Figura 13.2.4.1: Captura del archivo ProductTests. Fuente: Propia.

Este archivo define dos pruebas unitarias para la funcionalidad de producto. La primera prueba, "Modify and confirm eyes in display product", verifica que un usuario pueda modificar la cantidad de productos en exhibición y que esta modificación se refleje correctamente en la aplicación. La segunda prueba, "Modify eyes in display product", verifica que un usuario pueda modificar la cantidad de productos en exhibición y que se muestre una confirmación de que los cambios se han guardado correctamente.

```
1   package com.mango.rfid.browserstack.test.product
2
3   import com.mango.rfid.browserstack.ENVIRONMENT
4
5   val CAMERA_BUTTON = "$ENVIRONMENT/menu_menu_scanner"
6   const val ALLOW_BUTTON = "com.android.permissioncontroller:id/permission_allow_foreground_only_button"
7   val ENTER_CODE_BUTTON = "$ENVIRONMENT/btnCodeManually"
8   val BARCODE_NUMBER = "$ENVIRONMENT/barcodeInputText"
9   val UNITS_TO_EXPOSE = "$ENVIRONMENT/units_to_expose"
10  val SAVE_BUTTON = "$ENVIRONMENT/btnAction"
11  val EMPLOYEE_CODE_FIELD = "$ENVIRONMENT/code"
12  val APPLY_BUTTON = "$ENVIRONMENT/btn_apply"
13  val TOUCH_OUTSIDE = "$ENVIRONMENT/touch_outside"
14  const val BACK_PRODUCT = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.F
15  const val BACK_MENU = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.Fram
16  val SEARCH_BUTTON = "$ENVIRONMENT/fbActionScanCamera"
17  val ENTER_CODE = "$ENVIRONMENT/contentMangoButton"
18  val NUMBER_BAR_CODE = "$ENVIRONMENT/barcodeInputText"
19  val CONTINUE_BUTTON = "$ENVIRONMENT/contentMangoButton"
20  const val RESULT_NUMBER_EYES = "/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/android.widget.LinearLayout/android.wi
21  val SUCCESS_IMAGE = "$ENVIRONMENT/success_img"
22
```

Figura 13.2.4.2: Captura del archivo ProductElements. Fuente: Propia.

Este archivo define constantes para varios elementos de la interfaz de usuario en la pantalla de producto, como los botones de cámara, guardar y aplicar, el campo de código de empleado y los mensajes de éxito. Estas constantes se utilizan para interactuar con estos elementos durante las pruebas.

```
1   package com.mango.rfid.browserstack.robots
2   import ...
    03932096
12  fun plusEyes(wait: WebDriverWait, n: Int){
13      val ojoPlusBtn = wait.until(ExpectedConditions.elementToBeClickable(By.id("$ENVIRONMENT/iv_plus")))
14      for (i in 0 until n) {
15          ojoPlusBtn.click()
16      }
17  }
    03932096
18  fun minusEyes(wait: WebDriverWait, n: Int){
19      val ojoMinusBtn = wait.until(ExpectedConditions.elementToBeClickable(By.id("$ENVIRONMENT/iv_minus")))
20      for (i in 0 until n) {
21          ojoMinusBtn.click()
22      }
23  }
    03932096 *
24  fun searchProductDisplay(driver: AndroidDriver, wait: WebDriverWait) {
25      wait.clickElementID(CAMERA_BUTTON)
26      wait.clickElementID(ALLOW_BUTTON)
27      wait.clickElementID(ENTER_CODE_BUTTON)
28      wait.sendElementID(BARCODE_NUMBER,"8445791741848")
29      driver.pressKey(KeyEvent(AndroidKey.ENTER))
30  }
    03932096 *
31  fun saveChanges(wait: WebDriverWait){
32      wait.clickElementID(SAVE_BUTTON)
33      wait.sendElementID(EMPLOYEE_CODE_FIELD, EMPLOYEE_CODE)
34      wait.clickElementID(APPLY_BUTTON)
35  }
```

```
03932096 *
36 fun backToMenu(wait: WebDriverWait){
37     wait.clickElementXPATH(BACK_PRODUCT)
38     wait.clickElementXPATH(BACK_MENU)
39 }
03932096 *
40 fun searchProductID(driver: AndroidDriver, wait: WebDriverWait){
41     wait.clickElementID(SEARCH_BUTTON)
42     wait.clickElementID(ENTER_CODE)
43     wait.sendElementID(NUMBER_BAR_CODE,"8445791741848")
44     driver.pressKey(KeyEvent(AndroidKey.ENTER))
45     wait.clickElementID(CONTINUE_BUTTON)
46 }
```

Figura 13.2.4.3: Capturas del archivo ProductRobots. Fuente: Propia.

Este archivo define varias funciones que simulan acciones del usuario, como aumentar y disminuir la cantidad de productos en exhibición, buscar un producto por su código de barras, guardar cambios y volver al menú. Estas funciones utilizan las constantes definidas en `ProductElements` para interactuar con los elementos de la interfaz de usuario.

Estos archivos demuestran cómo se pueden automatizar las pruebas de la funcionalidad de producto. Al automatizar estas pruebas, se puede aumentar la eficiencia del proceso de prueba y reducir la posibilidad de errores humanos. Además, estas pruebas proporcionan una forma de verificar que la funcionalidad del producto continúa funcionando como se espera a medida que se realizan cambios en la aplicación.

13.2.5 Cambiar Idioma

Estos archivos son parte de un conjunto de pruebas automatizadas para la funcionalidad de cambio de idioma de una aplicación. Se utilizan para validar que el cambio de idioma funcione correctamente y produzca los resultados esperados.

El archivo ChangeLanTest define una prueba unitaria para la funcionalidad de cambio de idioma. La prueba, `Change Language English to Spanish`, verifica que un usuario pueda cambiar el idioma de la aplicación de inglés a español y que este cambio se refleje correctamente en la aplicación.

El archivo ChangeLanElements define constantes para varios elementos de la interfaz de usuario en la pantalla de configuración de idioma, como los botones de configuración, búsqueda y añadir idioma, y el campo de texto de búsqueda. Estas constantes se utilizan para interactuar con estos elementos durante las pruebas.

Se definen varias funciones que simulan acciones del usuario, como abrir la pantalla de configuración, ir a la opción de idioma, cambiar los idiomas y volver a la aplicación. Estas funciones utilizan las constantes definidas en `ChangeLanElements` para interactuar con los elementos de la interfaz de usuario.

Estos archivos demuestran cómo se pueden automatizar las pruebas de la funcionalidad de cambio de idioma. Al automatizar estas pruebas, se puede aumentar la eficiencia del proceso de prueba y reducir la posibilidad de errores humanos. Además, estas pruebas proporcionan una forma de verificar que la funcionalidad de cambio de idioma continúa funcionando como se espera a medida que se realizan cambios en la aplicación.

13.2.6 Procesos

En este apartado hay muchos test subdividido en muchas clases de test y para evitar poner código muy repetitivo solo se va a mencionar que funcionalidades se han testeado.
Dentro del menú de procesos de la aplicación ADA+ encontramos varias funcionalidades, que son:

- Recepción
- Etiquetado
- Inventario
- Traspasos
- Cambio de Zona
- Confirmación
- Reposición
    - General
    - Camión
    - Bultos
- Etiquetado

Próximamente se mostrará una tabla representando cuales de todas estas funcionalidades están cubiertas con test y que tipo de test, ya que muchos de los test que se pidieron eran difíciles de implementar y investigamos con el equipo un poco más allá

Recepción	10%
Etiquetado	10%
Inventario	95%
Traspasos	100%
Cambio de Zona	10%
Confirmación	95%
Etiquetado	50%

Tabla 13.2.4.3: Tabla con la representación en porcentaje cuantas funcionalidades están testeadas. Fuente: Propia.

A continuación se explicarán qué test se querían realizar en un principio y como se han conseguido o porqué no se han podido realizar.

Recepción: Estos tests son sobre la aplicación de tienda, ADA+. Esta funcionalidad requiere de un dispositivo externo, llamado pala dentro de Mango, que se conecta por Bluetooth al dispositivo móvil y sirve para leer las etiquetas RFID. Estos tests end-to-end se ejecutan sobre dispositivos reales, por lo que no se puede emular un dispositivo Bluetooth.

Esto ha supuesto una dificultad extra sobre los tests. Además, esta funcionalidad no es muy usada ni crítica para el funcionamiento de la tienda, así que decidimos con Ferran, Product Owner, y Federico, bajar la prioridad de los tests sobre esta funcionalidad. Solo se pidió testear que se mostrará la opción "Recepción" y poder acceder a esta, pero no hay tests de la funcionalidad en sí.

Etiquetado: Este proceso es similar al anterior, pero además se requiere de una impresora para imprimir una etiqueta. Por lo tanto, comprobar si imprime correctamente una etiqueta es prácticamente imposible, ya que estos tests se ejecutan sobre dispositivos reales de BrowserStack, ubicados en la India. Es imposible conectar por Bluetooth un dispositivo de Mango. Así que se decidió testear lo mismo que en la funcionalidad anterior.

Inventario: Este proceso es muy importante para la tienda ya que comprueba el stock de la tienda y revisa que todo esté en orden. Se conecta la pala con el dispositivo móvil. Al ser tan importante y no poder crear la conexión, se ha testeado todo excepto la lectura de etiquetas con la pala. Se decidió crear escenarios de casuísticas típicas de Mango, usando llamadas HTTP Request con datos mockeados. Por lo tanto, se han creado todos los escenarios diseñados por Ferrán, Product Owner. He valorado con un 95% de coverage puesto que no se comprueba que la pala lea correctamente las etiquetas. La funcionalidad de "Inventarios" se usa en todas las tiendas Mango del mundo, dos veces a la semana, en Europa-Asia-África martes y jueves, mientras que en Estados Unidos es lunes y miércoles.

Traspasos: Esta funcionalidad consiste en enviar o, como dice el nombre, traspasar prendas de una tienda a otra, a almacén general, retirar o restaurar prendas. Para esta funcionalidad no se requiere de un segundo dispositivo, por lo que se pudo realizar todos los tests pedidos desde un principio. Dentro de cada traspaso, puedes crear el número de cajas que se quiera y rellenarlo de tantas prendas como se diga. También hay tests que comprueban reglas de negocio como no permitir crear dos tipos de traspasos del mismo con el mismo destino.

Cambio de Zona: Esta funcionalidad consiste en cambiar el registro de una prenda en concreto de almacén a tienda o viceversa. Esta funcionalidad requiere de la pala para realizar el cambio. Es una funcionalidad sencilla y básica, por lo que no se dio prioridad al testeo de esta. Como las otras funcionalidades que no se testean a fondo, se ha testeado que se muestre la opción.

Confirmación: Para esta funcionalidad se requiere de la pala, pero esta funcionalidad es crítica para confirmar a Central que un camión ha traído las prendas correctas a la tienda. Por lo que se usa en todas las tiendas Mango del mundo, dos veces a la semana, en Europa-Asia-África lunes y miércoles, mientras que en Estados Unidos es martes y jueves. Se ha buscado diseñar varios escenarios para poder replicar el funcionamiento de la pala usando datos simulados creados artificialmente en la base de datos.

Etiquetado: Esto consiste en una opción para poder gestionar las etiquetas de las prendas, que se requiere de una impresora portátil que se tiene en todas las tiendas para imprimir las etiquetas. Esta funcionalidad tiene mucha importancia aunque sea relativamente sencilla en cuanto a programación. Esta opción permite al usuario de la aplicación, personal de tiendas, que den de baja etiquetas de prendas, creen para prendas que se les haya estropeado o roto la etiqueta y/o también para modificar el precio en caso de épocas de rebajas. Cómo comprobar que la impresora imprima correctamente o no, se ha testeado todo el proceso de la aplicación hasta la petición de impresión a la impresora. Por eso, con Ferrán, decidimos valorar que estaba testeada al 50% ya que sigue requiriendo de un test manual.

14. Gestión de riesgo: Planes alternativos y obstáculos

Riesgos:

1. **Posibles bloqueos por falta de acceso a herramientas o permisos necesarios:**
   Riesgo: Bajo **Probabilidad:** Alta
   La incapacidad para acceder a ciertas herramientas puede generar obstáculos en el progreso del proyecto. Resolver estos bloqueos implica abrir un ticket para obtener los permisos pertinentes, lo cual suele llevar unos tres días en solucionarse.

2. **Variaciones en la planificación debido a contratiempos en otros proyectos:**
   Riesgo: Medio **Probabilidad:** Baja
   Es posible que los planes establecidos se vean alterados por problemas surgidos en otros proyectos en curso. Para mitigar estas desviaciones, es esencial mantener un alto estándar de calidad en el código, evitando así errores y facilitando la gestión de cualquier cambio en el cronograma.

3. **Limitaciones en la realización de pruebas sobre todas las funcionalidades debido a problemas con capacidades externas:**
   Riesgo: Alto **Probabilidad:** Media
   Existe la posibilidad de que no se puedan realizar pruebas exhaustivas sobre todas las funcionalidades del proyecto debido a inconvenientes relacionados con capacidades externas, como la simulación de funciones Bluetooth o el acceso a la cámara. Esto podría comprometer la calidad del producto final y aumentar el riesgo de errores o mal funcionamiento en áreas no probadas.

Planes:

1. **Riesgo 1**: Establecer un procedimiento claro para solicitar y obtener los permisos necesarios para acceder a las herramientas requeridas, minimizando así el tiempo de bloqueo por esta causa.

2. **Riesgo 2**: Mantener una comunicación fluida con los equipos de otros proyectos para anticipar posibles problemas que puedan afectar la planificación y poder ajustarla de manera oportuna.

3. **Riesgo 3**: Explorar alternativas para probar las funcionalidades que dependen de capacidades externas, como buscar herramientas de simulación o encontrar soluciones temporales para acceder a dichas capacidades en entornos controlados. Esto garantizará una evaluación exhaustiva del proyecto y reducirá el riesgo de problemas no detectados antes de la entrega.

15. Presupuesto

15.1 Identificación de los costes

Para poder estimar el coste que tendría este proyecto para llevarlo a cabo, en los próximos apartados se va estudiar y a dividir los tipos de gasto que pueda tener este proyecto, en este caso los hemos dividido en 4 apartados: Recursos humanos (sueldos de personas que se necesitan para desarrollar este proyecto), hardware (material necesario como ordenador, ratón, teclado…), software (programas necesarios para el desarrollo, algunos de ellos son gratuitos como Google Chrome y otros de pago, como Microsoft Office 365), gastos generales (luz, internet, impuestos). Además se va a tener en cuenta imprevistos y posibles contingencias.

15.1.1 Recursos humanos

Esta sección trata los roles de empleados necesarios para este proyecto y desarrollar las diferentes tareas mencionadas en el diagrama de *Gantt.* Se usarán aproximaciones para calcular y planificar los costes por hora de los distintos roles. El Equipo de desarrollo se divide entre desarrolladores *frontend* y *backend* seniors, es decir, con desarrolladores con varios años de experiencia. La siguiente tabla muestra una tabla de los salarios por hora brutos. Las aproximaciones se han extraído de una página llamada *glassdoor* [14].

Rol	**Salario/hora (Bruto)**	**Salario/hora (Bruto) + Seguridad Social (x1.3)**
Desarrollador Android senior	20.75€	26.98€
Product Owner	21.53€	27.99€
Sistemas	25.56€	32.84€
Experience Designer	22.13€	28.13€
Desarrollador becario (Yo)	10€	13€

Tabla 15.1.1.1: Tabla con los costes estimados por rol y hora. Fuente: Elaboración propia

Una vez obtenidos los costes por rol en la tabla anterior, podemos determinar de manera más aproximada el coste de realizar cada tarea. En la siguiente tabla podemos observar el coste que supone cada tarea fragmentado en cada el coste de cada rol.

ID	Tarea	Horas Totales	Horas						Coste (€)
			M	JGM	AD	PO	FPM	DO	
TD1	Alcance	30	29						377
TD2	Planificación	10	9						117
TD3	Gestión económica y sostenibilidad	10	9						117
TD4	Documentación de la fase inicial	15	14						182
TD5	Documentación fase de seguimiento	30	29.5						15.5
TD6	Documentación de la fase final	70	69						897
TD7	Preparación de la presentación final	40	39						507
TP1	Daily Performance	0.5(x90)	0,5	0,5	0,5	0,5	0,5		130,92
TP2	Refinamiento	1(x6)	1	1	1	1	1		261,84
TP3	Sprint planning	1(x6)	1	1	1	1	1		261,84
TP4	Android Coordination	1(x6)	1	1	1	1	1		261,84
TP5	Weekly Scrum	1(x18)	1	1	1	1	1	1	293,97
TCP1	Creación Proyecto y configuración de Marco de Test	50	40	5	2		3		1074,63
TCP2	Creación Job en Jenkins	20	18	1				1	293,11
TCP3	Investigar que Capabilities se adecuan más a ADA+	15	10	2		2	1		271,93
TCP4	Investigar Arquitectura	15	14	1					208,98
TDT1	Diseñar y investigar que test crear para la aplicación de ADA+	15	12	1		1			210,97
TDT2	Crear test Log In	7	6		1				239,88
TDT3	Crear test Log out	7	6		1				239,88
TDT4	Crear test buscador principal por nombre	10	9		1				278,88
TDT5	Crear test buscador principal por referencia	10	9		1				278,88
TDT6	Crear test buscador principal por barcode	10	9		1				278,88
TDT7	Crear test Proceso Inventario	15	14		1				343,88
TDT8	Crear test Proceso Traspasos	15	14		1				343,88
TDT9	Crear test acceder Training	7	6		1				239,88
TDT10	Crear test Compra Online	15	14		1				343,88
TDT11	Crear test Cambiar Idioma	15	14		1				343,88
TDT12	Crear test Buscar por Tópicos	15	14		1				343,88
TDT13	Crear test Operaciones y Análisis	12	11		1				304,88

TDT14	Crear test Proceso Confirmaciones	15	14		1				343,88
TAD1	Presentar Proyecto a los Android Developers de ADA+	7	7		7				1224,16
TAD2	Integrar Fase Inicial con Android Developers de ADA+	15	15		15				2623,2
TAD3	Integrar Fase final con Android Developers de ADA+	20	20		15				2688,2
TOTAL		515							16543,29

Tabla 15.1.1.2: Tabla con los costes estimados del proyecto por tarea y rol. Fuente: Elaboración propia

15.1.2 Hardware

El desarrollo de este proyecto requiere de material hardware básico. Este material ya se listó en apartados anteriores. Este hardware es necesario y vamos a amortizar durante 4 años este material, tal y como indica hacienda que se puede hacer. Para hacer una aproximación del coste que nos va a suponer este hardware durante estos siguientes años, se va a suponer que este año tiene 2040 horas, ya que este año, 2024 tiene 255 días laborables, con este dato podemos calcular la amortización.

Fórmula ejecutada:
$(Coste\ Hardware(€)\ /(4\ años\ *\ 255\ (días\ laborables/año)\ *\ 5(h/día)))\ *\ 90\ (días\ desarrollo)\ *\ 5(h/día)$

En la siguiente tabla podemos observar la amortización de cada objeto con su coste y su amortización.

Hardware	**Coste (€)**	**Amortización (€)**
ThinkPad T 14 Gen 4 21HDCTO1WWES3 (Ordenador portátil [15])	2588.01	270.57
Logitech B100 (Ratón [16])	6.49	0.68
Trust ClassicLine (Teclado [17])	11.99	1.25
BenQ GW2280 (Monitor[18])	168.50	17.61
TOTAL		**290.11**

Tabla 15.1.2.1: Tabla con los costes de los productos Hardware. Fuente: Elaboración propia

15.1.3 Software

Continuando con los costos, a continuación se detallan los gastos generados por el software utilizado en el día a día para el desarrollo del proyecto y la comunicación entre colegas. En nuestro caso, al emplear herramientas con licencias anuales o mensuales, no consideramos la amortización para posibles cambios de software obsoleto, ya que la normativa fiscal permite efectuar modificaciones cada dos años, dado que con las licencias tenemos acceso a actualizaciones.
El cálculo para determinar las amortizaciones es el siguiente:
$(Coste\ Software(€)\ /(4\ años\ *\ 255\ (días\ laborables/año)\ *\ 5(h/día)))\ *\ 90\ (días\ desarrollo)\ *\ 5$
En la Tabla siguiente tabla se presentan los costos asociados al software.

Software	Coste (€)	Amortización (€)
Visual Studio Code	Gratuito	
IntelliJ IDEA	Gratuito	
Bitbucket	5602 anual / 100 usuarios	23.43
Git	Gratuito	
SourceTree	Gratuito	
Jenkins	Gratuito	
Office365	99 anual	10.35
Google Chrome	Gratuito	
BrowserStack	1999 anual	208.99
TOTAL		242.77

Tabla 15.1.3.1: Tabla con los costes de los productos Software. Fuente: Elaboración propia

15.1.4 Gastos generales

Para concluir con los costos, es importante tener en cuenta otros tipos de gastos, tales como el material de oficina necesario para llevar a cabo el proyecto, el consumo eléctrico, los gastos de Internet y el transporte hacia el lugar de trabajo, entre otros. En el caso del material de oficina, es posible amortizar en un período de 20 años, según lo estipulado por Hacienda [19].
Dado que actualmente asisto a la oficina la mitad de los días y la duración total del proyecto es de 90 días, se deben considerar 46 días de desplazamientos. Para calcular los gastos de electricidad, se debe tener en cuenta la potencia del portátil, aproximadamente 150W, y la de un fluorescente, alrededor de 18W, considerando un par de fluorescentes para el cálculo del costo.
La fórmula para obtener las amortizaciones es la siguiente:
$(C.G(€)\ /(20\ años\ *\ 255\ (días\ laborables/año)\ *\ 5(h/día)))\ *\ 90\ (días\ desarrollo)\ *\ 5(h/día)$

En la Tabla siguiente se presentan los costos asociados a estos recursos.

Recurso	Coste (€)	Amortización (€)
Mesa oficina [20]	139	2.90
Silla oficina [21]	179	3.74
Desplazamiento (Bus lanzadera + Gasolina)	4250 €/día	195.50
Electricidad	0.1232 €/kWh	10.54
Internet	33€/mes	165.60
TOTAL		378.28

Tabla 15.5: Tabla con los costes generales. Fuente: Elaboración propia

También se ha buscado locales o naves industriales parecidas a las instalaciones de Mango en Palau Solità i Plegamans, aunque no se ha encontrado nada parecido, ya que estas son una instalaciones muy grandes, se trata de una nave industrial de $26.740m^2$, las instalaciones que más parecidas he encontrado actualmente en venta o de alquiler por *idealista [22],* página de venta y compra o alquiler de viviendas, locales, naves industriales o oficinas. Una nave industrial [23] situada a menos de 5km, en sabadell, hay una nave industrial de $28.000m^2$ y la misma cantidad de pisos, en este caso 3, y tiene un coste de 189.00€/mes. Como este proyecto tiene una duración de 4 meses ya tendríamos el coste total de la nave, 756.000€.

15.2 Estimación de los costes

Para concluir con la evaluación de los costos, debemos considerar los gastos imprevistos y la contingencia, que en este sector es del 15%, lo que equivale a **2.621,68**€. En la tabla siguiente, podemos observar los costos relacionados con los posibles imprevistos que podríamos enfrentar durante la ejecución del proyecto.

Riesgo	Probabilidad (%)	Tiempo (horas)	Coste Estimado (€)	Coste (€)
Bugs	35%	25h	625.52	218.93
Bloqueos	20%	15h	405.28	113.48
Desviación del plan	5%	10h	487.67	24.38
Limitaciones en la realización de pruebas	35%	15h	371.92	130.17
Total				486.96

Tabla 15.2.1: Tabla con los costes para imprevistos. Fuente: Elaboración propia

	Coste (€)
Costes de Personal por Actividad	16543.29
Costes Generales	911.16
Contingencia	2621.68
Imprevistos	486.96
Total	20563.1

Tabla 15.2.2: Tabla con los costes estimados del proyecto. Fuente: Elaboración propia

15.3 Control de gestión

En esta sección, se detalla el seguimiento que se realizará para controlar las posibles variaciones en el presupuesto que podrían surgir durante el desarrollo del proyecto. Nuestra estrategia consiste en registrar el tiempo real que nos lleva completar cada tarea. De esta manera, podremos determinar si hay una desviación en nuestro presupuesto y si estamos cumpliendo con lo presupuestado inicialmente. Para lograr este control, hemos definido las siguientes métricas:

- **Desviación en coste laboral (variación en coste por tarifa)** = (coste estimado - coste real) * horas reales utilizadas

- **Desviación en consumo laboral (variación en eficiencia)** = (horas estimadas utilizadas - horas reales utilizadas) * coste estimado

Al concluir el período del proyecto que estamos llevando a cabo, realizaremos los cálculos necesarios para determinar las desviaciones. Esto nos permitirá analizar el presupuesto y decidir si necesitamos o no recurrir a la contingencia que se calculó al comienzo del proyecto.

16. Informe de sostenibilidad

16.1 Autoevaluación

Para realizar la autoevaluación se han primero contestado las preguntas y realizado la encuesta para poder tener mayor conocimiento sobre el estado y qué posibles consecuencias puede llegar a tener. La implementación del proyecto ha demostrado un enfoque sólido hacia la sostenibilidad en múltiples dimensiones. Desde el punto de vista económico, se ha reconocido la importancia de garantizar el funcionamiento eficiente de las operaciones comerciales, lo que contribuye directamente a la estabilidad financiera de la empresa. Además, se ha establecido una base sólida para futuras mejoras en la gestión de costos y la optimización de recursos, mediante la exploración de estrategias adicionales para minimizar gastos y aumentar la rentabilidad del proyecto.

En cuanto a la sostenibilidad ambiental, se ha demostrado un compromiso real al considerar cuidadosamente el impacto ambiental del proyecto. Sin embargo, para futuras implementaciones, sería beneficioso profundizar en la investigación y explorar aún más opciones para reducir la huella de carbono y promover prácticas ambientales más sostenibles en todas las etapas del proyecto, desde el desarrollo hasta la implementación y el mantenimiento.

Respecto a la sostenibilidad social, si bien se ha demostrado un compromiso con el bienestar y la calidad de vida de los empleados, es crucial continuar evaluando y abordando los desafíos relacionados con la carga de trabajo y el equilibrio entre vida laboral y personal. Implementar estrategias adicionales de apoyo y desarrollo profesional puede ayudar a garantizar un entorno laboral saludable y sostenible a largo plazo.

En resumen, si bien la implementación actual del proyecto muestra un compromiso firme con la sostenibilidad en todas sus dimensiones, también ofrece oportunidades para futuras mejoras y refinamientos. Al continuar enfocándose en la eficiencia económica, la responsabilidad ambiental y el bienestar social, el proyecto puede seguir avanzando hacia una mayor sostenibilidad y éxito a largo plazo.

16.2 Dimensión Económica

Respecto al Proyecto puesto en producción (PPP): Reflexión sobre el coste que has estimado para la realización del proyecto?

El costo estimado para la realización del proyecto ha resultado ser más elevado de lo inicialmente esperado. Se han identificado diversos aspectos que demandan consideración, como el consumo eléctrico, los desplazamientos, el equipo informático necesario, las plataformas de pruebas y el equipo humano requerido. No obstante, dado el vínculo directo de la aplicación con las ventas y la significativa generación de ingresos que representa, es fundamental asegurar su funcionamiento a toda costa. El costo adicional se justifica por la importancia estratégica de garantizar la integridad y eficiencia del sistema.

Respecto a la Vida Útil: ¿Cómo se resuelven actualmente los aspectos de costes del problema que quieres abordar (estado del arte)?

Los aspectos de costos del problema que se pretende resolver están siendo abordados de manera manual en la actualidad. Esto implica una pérdida considerable de tiempo y recursos humanos, ya que se ejecuta un proceso repetitivo que podría automatizarse. La automatización de las pruebas permitirá una optimización significativa en términos de eficiencia y recursos, al eliminar la necesidad de realizar pruebas manuales laboriosas y propensas a errores.

En que mejora económicamente tu solución respecto a las existentes?

La implementación de la solución propuesta conlleva mejoras sustanciales en términos económicos. Al asegurar el correcto funcionamiento de las ventas en las tiendas de Mango, se evitan posibles pérdidas financieras ocasionadas por errores humanos. La capacidad de identificar y corregir errores antes de que afecten a más de 2700 tiendas resulta crucial para evitar interrupciones en el servicio que podrían traducirse en pérdidas significativas de ingresos. La solución propuesta representa una inversión estratégica que garantiza la estabilidad y la rentabilidad a largo plazo del sistema.

16.3 Dimensión Ambiental

Respecto al Proyecto Puesto en Producción (PPP): ¿Has estimado el impacto ambiental que tendrá la realización del proyecto?

Al estimar el impacto ambiental del proyecto, se ha considerado cuidadosamente si la plataforma utilizada para generar las pruebas respeta el medio ambiente. En este caso, BrowserStack se destaca por su enfoque en el reciclaje de dispositivos sobre los cuales se ejecutan las pruebas. Además, se ha planificado ejecutar estas pruebas únicamente en momentos clave, como antes de lanzar una aplicación a producción, para evitar ejecuciones innecesarias. Esta estrategia contribuye a minimizar el consumo de recursos y la huella ambiental del proyecto, al optimizar el uso de los dispositivos disponibles y evitar el desperdicio de energía en pruebas redundantes.

Respecto al Proyecto Puesto en Producción (PPP): ¿Te has planteado minimizar su impacto, por ejemplo, reutilizando recursos?

En cuanto a la minimización del impacto del proyecto, se han explorado diversas opciones para reutilizar recursos y reducir el consumo energético. Se ha considerado la posibilidad de realizar pruebas más extensas para evitar encender y apagar repetidamente los dispositivos, así como el uso de simuladores y emuladores en lugar de dispositivos físicos. Sin embargo, ninguna de estas opciones garantiza el funcionamiento adecuado de la aplicación ni ofrece una representación precisa del entorno de ejecución real. Por lo tanto, se ha priorizado la calidad y la precisión de las pruebas sobre la minimización del impacto ambiental.

Respecto a la Vida Útil: ¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué aspectos mejorará ambientalmente tu solución respecto a las existentes?

Actualmente, el problema se aborda mediante pruebas manuales, lo que resulta en un uso ineficiente de recursos y tiempo. Cada programador ejecuta las mismas pruebas manualmente en dispositivos diferentes, lo que aumenta la carga de trabajo y la probabilidad de errores humanos. Esta metodología no solo es laboriosa, sino que también es poco sostenible desde el punto de vista ambiental, ya que implica un uso excesivo de

dispositivos y energía.

En comparación, la solución propuesta mejorará significativamente la eficiencia ambiental al automatizar las pruebas y minimizar el uso de recursos. Al utilizar una plataforma como BrowserStack, que recicla dispositivos para ejecutar pruebas, se reduce la necesidad de adquirir nuevos dispositivos y se optimiza el consumo de energía. Además, al eliminar las pruebas manuales redundantes, se reduce el desperdicio de recursos y se mejora la eficiencia operativa en términos económicos y ambientales.

16.4 Dimensión Social

Respecto al Proyecto Puesto en Producción (PPP): ¿Qué crees que te aportará a nivel personal la realización de este proyecto?

La realización de este proyecto promete un valioso crecimiento personal y profesional. A nivel personal, me brindará un profundo entendimiento de cómo operan las empresas internamente, especialmente en lo que respecta a los desafíos asociados con la implementación de actualizaciones de producción. Además, adquiriré conocimientos exhaustivos sobre diversos marcos de prueba, lo que ampliará mi conjunto de habilidades y me hará más versátil en el campo de la ingeniería de software. En términos de impacto, este proyecto me permitirá evaluar los efectos tanto positivos como negativos en el ámbito interno y externo de la empresa. Esto incluye entender cómo influirá en la eficiencia interna y en la experiencia del usuario final, lo que contribuirá a mi desarrollo como profesional y al éxito general del proyecto.

Respecto a la Vida Útil: ¿Cómo se resuelve actualmente el problema que quieres abordar (estado del arte)? ¿En qué mejorará socialmente (calidad de vida) tu solución respecto a las existentes?

Actualmente, el problema que intento abordar se resuelve principalmente mediante pruebas manuales, lo que consume una cantidad significativa de tiempo y recursos. La implementación de una solución automatizada no solo mejorará la calidad de vida de los desarrolladores al brindarles más tranquilidad y libertad para gestionar sus tareas, sino que también les permitirá dedicar más horas a tareas de mayor valor agregado en lugar de perder tiempo en ejecuciones manuales de pruebas. Esto, a su vez, puede tener un impacto social positivo al reducir el estrés laboral y mejorar la satisfacción en el trabajo. Sin embargo, es importante tener en cuenta que esta solución también implica una responsabilidad adicional para los desarrolladores, quienes deberán mantener y actualizar regularmente los tests automatizados para garantizar su efectividad a largo plazo.

Respecto a la Vida Útil: ¿Existe una necesidad real del proyecto?

Definitivamente existe una necesidad real para este proyecto. Hasta ahora, la falta de garantía sobre el correcto funcionamiento del proyecto antes de su implementación en producción ha sido un problema significativo, como ya se ha mencionado. La ejecución manual de pruebas no proporciona una cobertura exhaustiva ni garantiza la detección de todos los posibles errores, lo que puede dar lugar a problemas graves una vez que el proyecto está en producción. La implementación de un sistema automatizado de pruebas no solo abordará esta brecha en la calidad del software, sino que también mejorará significativamente la eficiencia y la confiabilidad del proceso de desarrollo, asegurando así un producto final de mayor calidad y satisfacción del cliente.

17. Puesta a prueba

17.1 Contexto

En Mango, de media solían haber tres incidencias por sprint, relacionadas con el código interno la mayoría de veces, Estas incidencias, podrían ser mínimas, en las que no afectan al flujo o trabajo de la tienda o bien podía ser un error que dejara la aplicación fuera de servicio durante un par de horas. Estas incidencias son muy dañinas, tanto para el ritmo de trabajo del equipo, como para la confianza de los usuarios.
A continuación se adjuntan algunas gráficas que muestran las incidencias que han existido durante este año.

Tabla 17.1.1: Tabla con incidencias desde 25 julio a 23 diciembre. Fuente: Documento Resumen de Sprint

Tabla 17.1.2: Tabla con incidencias desde 23 octubre a 24 febrero. Fuente: Documento Resumen de Sprint

Tabla 17.1.3: Tabla con incidencias desde 23 Octubre a 24 abril. Fuente: Documento Resumen de Sprint

17.2 Primera puesta a prueba

Aprovechamos que teníamos la migración de la versión de MongoDB de 5.0 a 6.0 para testear qué utilidad tenía el proyecto. Antes de esta migración preparamos entre el equipo de Androids como el de Backends, un plan de restauración en caso de que alguno de los test fallase. Ejecutamos la migración a las 12 de la noche, ya que es la hora a la que menos tiendas afectamos, ya que solo afectamos a Estados Unidos (que son las 6:00 am en ese momento allí), que de momento es el continente con menos tiendas.
Una vez ejecutada la migración ejecutamos los test, usando al máximo permitido (5 dispositivos) para ejecutar los más rápido posible los test end2end.
Aqui link de la ejecución, como ya hace más de 30 días no se puede ver el video: link
En este link podremos ver la ejecución que se hizo ese día y como vemos todo fue según lo previsto.
Durante el sprint siguiente solo hubieron dos incidencias, no crítica, a causa de una actualización de SAP, por lo que podemos concluir que fue un completo éxito.

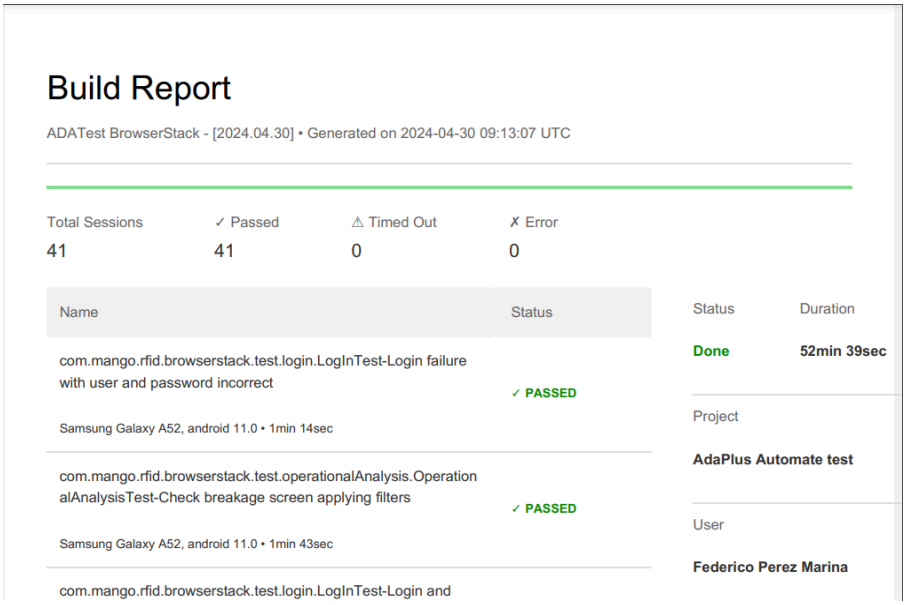


Figura 17.2.1: Resultado de la ejecución de tests. Fuente: BrowserStack app automate

Tabla 17.2.2: Tabla con incidencias desde 23 Octubre a 24 abril. Fuente: Documento Resumen de Sprint

Figura 17.2.3: Figura con incidencias y tiendas abiertas desde el enero del 2023. Fuente: Documento Resumen de Sprint

Como podemos ver la ejecución tardó casi 1 hora debido que se tuvo que repetir unos test para comprobar una funcionalidad, esos 52min, representan el tiempo transcurrido entre el primer test y el último. Así que como propuesta de mejora futura era conseguir reducir ese tiempo a 30 minutos.

17.3 Extracción de resultados

Al ver la cantidad de test que se tienen que ejecutar, si cada uno tarda 2-3 minutos por mucha paralelización, son muchos test y por lo tanto tarda mucho en ejecutar.
Así que analizando con el equipo aprovechamos para empezar a enseñar como realizar estos test y aplicar una nueva técnica para evitar la creación de test en masa.

Esta técnica consiste en fragmentar un solo test en “steps”, cada step puede ser un test, por lo que se pueden agrupar muchos test, en este caso lo implementamos en login para probarlo. Hicimos estos test y generamos test para una nueva funcionalidad llamada “Schedule eyes” o “Ojos programados”, aquí link de como se realizaron, link,

Como se puede ver, ahora con solo un test de login se puede testear lo que antes se necesitaban 3 test y ahora con solo 1 ya hemos testeado lo anterior. De la nueva funcionalidad anteriormente se hubiera necesitado la creación de más de 10 test, de esta forma con solo 3 podemos testear toda la funcionalidad.

18. Desviación Real

La desviación real de este proyecto solo ha supuesto en relación al tiempo, ya que se ha tardado dos semanas más en acabar el desarrollo de la aplicación, por lo que ha continuación calcularemos como ha afectado al cálculo inicial y que tanto por ciento se ha afectado al presupuesto inicial.

Podemos ver que el retraso se ha debido al primer y tercer sprint y la finalización de estas tres tareas, por lo que si calculamos la desviación de estas tareas tendremos el costo real del proyecto.

Primer Sprint: Como el retraso ha sido relacionado con las horas de [M], podemos calcular que 5 días, a 5h diarias con un coste de 11,3€/h, son 282,5€.

Segundo Sprint: Este sprint no tuvo ninguna desviación por lo que no hubo ningún coste extra.

Tercer Sprint: Este sprint tuvo un retraso de 5 días a causa de que el compañero Android hizo vacaciones esa semana para la finalización de la última tarea, por lo que tuvo un coste extra de 674,5€.

Para concluir podemos ver que este proyecto ha tenido un coste en cuanto a horas de personal de 674,5 + 282,5 = 957€

El costo real de este proyecto ha sido el siguiente:

$$Coste\ Total = Costes\ de\ Personal\ por\ Actividad + Costes\ Generales$$

$$Coste\ Total = (957 + 16543.29) + 911.16 = 18.411,45€$$

Como vemos y como no hemos tenido ningún bug ni imprevisto, no hemos gastado más dinero, por lo que hemos acabado gastando menos del calculado previamente, que eran 20563,1€. Aun así ha representado 957€ mas caro para pagar al personal que ha representado un 5.78% más caro de lo previsto

19. Conclusiones y trabajo futuro

19.1 Competencias técnicas trabajadas

A lo largo de mi periodo de prácticas en Mango, he trabajado en el desarrollo, mantenimiento y evaluación de sistemas y servicios de software complejos y críticos (CES1.1), con un enfoque particular en la aplicación de tienda ADA+. Hemos logrado una notable reducción de incidencias y mejorado la eficiencia del sistema, demostrando una solución efectiva a los problemas de integración (CES1.2).

Durante el proceso, hemos identificado, evaluado y gestionado los riesgos potenciales asociados a la construcción de software (CES1.3), lo que ha sido fundamental para garantizar la estabilidad y fiabilidad del sistema. Hemos desarrollado, mantenido y evaluado servicios y aplicaciones distribuidas con soporte de red (CES1.4), lo que ha permitido una mejor integración y comunicación entre los diferentes componentes del sistema.

En cuanto a la gestión de bases de datos, hemos especificado, diseñado, implementado y evaluado bases de datos (CES1.5) y administrado las mismas (CES1.6), lo que ha permitido un manejo eficiente de los datos y ha mejorado el rendimiento general del sistema.

El control de calidad y el diseño de pruebas en la producción de software (CES1.7) han sido una parte integral de nuestro trabajo, asegurando que cada funcionalidad cumple con los estándares de calidad y funciona como se espera.

Hemos desarrollado, mantenido y evaluado sistemas de control y de tiempo real (CES1.8), lo que ha permitido una respuesta rápida y precisa a las interacciones de los usuarios.

En cuanto a la gestión de requisitos, hemos definido y gestionado los requisitos de un sistema de software (CES2.1), y diseñado soluciones apropiadas que integren aspectos éticos, sociales, legales y económicos (CES2.2).

Hemos desarrollado servicios y aplicaciones multimedia (CES3.1), mejorando la experiencia del usuario y la interactividad de la aplicación. Aunque la gestión de un almacén de datos (CES3.2) y la comprensión en la gestión y gobierno de los sistemas de software (CES1.9) se han abordado en menor medida, hemos adquirido una comprensión básica de estos aspectos.

En resumen, considero que hemos logrado cumplir con los objetivos establecidos, contribuyendo significativamente al desarrollo y mejora de la aplicación ADA+.

19.2 Relación del TFG con el grado y la especialidad

PRO1: La asignatura PRO1 ha sido fundamental para adquirir una sólida comprensión de los conceptos básicos de la programación. Este conocimiento ha permitido abordar y resolver problemas de manera eficiente y sistemática, aplicando principios y técnicas fundamentales en la construcción de soluciones algorítmicas sólidas.

PRO2: Creación de proyectos y primeros tests unitarios
En PRO2, se ha profundizado en la creación de proyectos completos, incorporando

prácticas esenciales de desarrollo de software como los tests unitarios. Esta asignatura ha sido crucial para aprender a estructurar proyectos desde su inicio hasta su implementación, asegurando la calidad del código a través de pruebas sistemáticas.

PROP: La asignatura PROP ha proporcionado conocimientos y habilidades en la gestión de proyectos utilizando herramientas de control de versiones como GitLab, cuya funcionalidad es comparable a Bitbucket. Estas competencias son vitales para el trabajo en equipo y la gestión eficiente del código fuente, facilitando la colaboración y el seguimiento de los cambios en proyectos de desarrollo de software.

PAR: Ha sido esencial para comprender y aplicar técnicas de paralelización de procesos, optimizando así la eficiencia y el rendimiento de las aplicaciones. En este proyecto en particular, se ha enfocado en la paralelización de pruebas, lo que ha permitido ejecutar múltiples tests simultáneamente, reduciendo significativamente el tiempo de validación y asegurando un desarrollo más ágil y eficiente.

BD: La asignatura BD ha brindado una comprensión profunda sobre la gestión de bases de datos, con un enfoque especial en bases de datos no relacionales. Este conocimiento ha sido indispensable para el manejo eficiente de grandes volúmenes de datos, permitiendo diseñar y mantener sistemas de almacenamiento de datos robustos y escalables que soportan las necesidades del proyecto.

DBD: La asignatura DBD ha reforzado mis habilidades en la gestión eficiente de bases de datos, abarcando desde la planificación y diseño hasta la implementación y mantenimiento. Estas competencias son cruciales para asegurar que las bases de datos sean rápidas, seguras y escalables, aspectos fundamentales para el correcto funcionamiento del proyecto.

GCS: Ha sido esencial para aprender a proteger los datos utilizados en las pruebas, garantizando que no haya fugas de seguridad y evitando vulnerabilidades. Este conocimiento es vital para asegurar la integridad y confidencialidad de la información, especialmente en entornos donde la seguridad de los datos es crítica.

PES: La asignatura PES ha proporcionado un marco integral para la creación y gestión de proyectos de software, incluyendo la documentación, gestión de recursos y la implementación del método ágil. Estas habilidades han sido fundamentales para llevar a cabo una planificación y ejecución eficiente del proyecto, asegurando una entrega de alta calidad dentro de los plazos establecidos.

ER: Ha introducido los principios del método ágil y la metodología Scrum, proporcionando una base sólida para la gestión de proyectos de software. Estos enfoques han sido aplicados para mejorar la colaboración del equipo, la flexibilidad en la planificación y la entrega continua de incrementos de producto funcionales.

SIM: Los conocimientos adquiridos en la asignatura SIM han sido cruciales para diseñar y realizar tests que simulan escenarios reales. Aprender a crear pruebas que imiten situaciones del mundo real ha permitido asegurar que el software desarrollado se comporta de manera predecible y fiable en condiciones prácticas, mejorando así su calidad y robustez.

19.3 Conclusiones personales

Al iniciar mis prácticas en Mango, se me encomendó una tarea principal: desarrollar un proyecto que garantizara el correcto funcionamiento de la aplicación de tienda ADA+ a

través de pruebas end-to-end. En retrospectiva, creo que, junto con el equipo, hemos logrado este objetivo de manera efectiva. Hemos observado una notable reducción de incidencias durante las migraciones de bases de datos, las actualizaciones de la aplicación y las pruebas en nuevos dispositivos.

Además, otro objetivo era simplificar el proceso tanto como fuera posible para permitir que el proyecto continúe evolucionando bajo la supervisión de los desarrolladores de Android encargados de las nuevas funcionalidades. Este objetivo se ha logrado gracias a las sesiones de formación en equipo y al pair programming, que me han permitido compartir todos los conocimientos adquiridos a lo largo de este proyecto.

Considero que este proyecto representa un avance significativo y una valiosa contribución para Mango, dado que no existía ninguno similar anteriormente. En una empresa de gran envergadura como esta, asegurar el funcionamiento perfecto de un servicio el 100% del tiempo es absolutamente crítico. Sin embargo, aún existen aspectos que pueden mejorarse, como la generación de más pruebas y la adaptación completa de las pruebas sobre funcionalidades Bluetooth.

19.4 Trabajo futuro

Mirando hacia el futuro, veo un gran potencial para expandir y mejorar este proyecto. Aunque hemos logrado mucho, siempre hay espacio para la innovación y el crecimiento. Por ejemplo, la funcionalidad Bluetooth, que actualmente es un desafío debido a las limitaciones técnicas, podría ser un área de enfoque para futuras mejoras.

Además, a medida que la aplicación ADA+ continúa evolucionando y se introducen nuevas funcionalidades, el proyecto de pruebas end-to-end también debe adaptarse y crecer para mantenerse al día. Esto requerirá un compromiso continuo con la formación y el aprendizaje, así como una estrecha colaboración con los desarrolladores de Android.

En resumen, estoy emocionado por el futuro de este proyecto y confiado en que continuará aportando un valor significativo a Mango en los años venideros.

Glosario

- Quality Assurance (QA): QA es el proceso sistemático de garantizar que un producto o servicio cumpla con los estándares de calidad definidos. Involucra la planificación, ejecución y seguimiento de actividades de prueba para identificar y corregir defectos antes de que el producto final se entregue a los clientes.

- DevOps: DevOps es una cultura y metodología de desarrollo de software que promueve la colaboración estrecha entre los equipos de desarrollo (Dev) y operaciones (Ops). Su objetivo es mejorar la velocidad y la calidad del desarrollo de software a través de la automatización, la integración continua y la entrega continua.

- Test End to End: Las pruebas End-to-End son un tipo de prueba que evalúa el flujo completo de un sistema, desde el inicio hasta el final. Se centran en validar la funcionalidad y la integridad del sistema en su conjunto, simulando el comportamiento del usuario final en un entorno de producción.

- Mango: Mango es una reconocida cadena de tiendas de moda con presencia internacional. Ofrece una amplia gama de prendas de vestir y accesorios para hombres, mujeres y niños.

- ADA+: Es una aplicación móvil para dispositivos Android desarrollada para su uso en las tiendas de ropa de la marca Mango. Proporciona una experiencia de compra conveniente y personalizada para los clientes.

- Frontend: Se refiere a la parte de un sistema o aplicación en la que los usuarios interactúan directamente. Incluye la interfaz de usuario, los elementos visuales y la experiencia de usuario.

- Backend: Es la parte de un sistema o aplicación que está detrás de escena y que los usuarios no ven directamente. Incluye la lógica de negocio, la gestión de datos y la comunicación con el frontend y otras partes del sistema.

- Android Developer: Es un profesional especializado en la creación de aplicaciones móviles para dispositivos Android. Se encarga del diseño, desarrollo y pruebas de aplicaciones utilizando el lenguaje de programación Java o Kotlin y las herramientas de desarrollo de Android.

- Pull Request: Es una solicitud para fusionar cambios de código realizados en una rama de desarrollo a otra rama, generalmente la rama principal. Permite a los desarrolladores revisar y discutir los cambios antes de fusionarlos, promoviendo la colaboración y la calidad del código.

- Branches: Las Branches, o ramas, son versiones alternativas del código base de un proyecto. Se utilizan para desarrollar nuevas características, corregir errores o realizar experimentos sin afectar el código principal. Las ramas se pueden fusionar o eliminar una vez que se completen los cambios planeados.

- MongoDB: MongoDB es una base de datos NoSQL de código abierto que proporciona alto rendimiento, alta disponibilidad y fácil escalabilidad. Trabaja con documentos y colecciones en lugar de tablas y filas, lo que permite un esquema de datos más flexible.

- BrowserStack: BrowserStack es una plataforma de pruebas en la nube que permite a los desarrolladores probar sus aplicaciones web y móviles en diferentes sistemas operativos y dispositivos móviles. Proporciona acceso a un entorno de pruebas de navegador en tiempo real sin la necesidad de configurar y mantener una infraestructura de pruebas propia.

- API (Interfaz de Programación de Aplicaciones): Una API es un conjunto de reglas y protocolos que permite que diferentes programas de software interactúen entre sí. Esencialmente, es una forma de que diferentes programas de software 'hablen' entre sí.

- Agile: Agile es una metodología de desarrollo de software que se centra en la entrega continua de valor al cliente a través de iteraciones cortas y regulares. Promueve la colaboración, la adaptabilidad y la mejora continua.

- Scrum: Scrum es un marco de trabajo Agile que se utiliza para gestionar proyectos de desarrollo de software. Se centra en la entrega de valor en ciclos cortos llamados sprints, y promueve la colaboración, la transparencia y la inspección y adaptación regulares.

- Kotlin: Kotlin es un lenguaje de programación moderno que se utiliza para desarrollar aplicaciones para la plataforma Android. Es interoperable con Java y se centra en la seguridad, la claridad y la eficiencia del código.

- JUnit: JUnit es un marco de pruebas unitarias para el lenguaje de programación Java. Permite a los desarrolladores escribir y ejecutar pruebas para comprobar que partes individuales de su código funcionan como se espera.

- JSON (JavaScript Object Notation): JSON es un formato de intercambio de datos ligero y fácil de leer para humanos. Se utiliza para transmitir datos entre un servidor y una aplicación web, utilizando pares de nombre-valor para representar los datos.

- Maven: Maven es una herramienta de gestión de proyectos de software. Ayuda a los desarrolladores a compilar y gestionar sus proyectos, gestionando las dependencias, la construcción, la documentación y la distribución.

- HTTP Request: Una solicitud HTTP es un mensaje que un cliente (como un navegador web) envía a un servidor para solicitar una acción en un recurso específico. Las solicitudes HTTP pueden ser de varios tipos, incluyendo GET (para recuperar datos), POST (para enviar datos), y DELETE (para eliminar datos).

- UML (Unified Modeling Language): UML es un lenguaje de modelado estándar utilizado en el desarrollo de software para visualizar el diseño de un sistema. Proporciona un conjunto de diagramas que ayudan a representar diferentes aspectos de un sistema, como su estructura, comportamiento y arquitectura.

- DDS/Diagrama de Secuencia: Un Diagrama de Secuencia es un tipo de diagrama UML que muestra cómo los objetos interactúan en un proceso específico a lo largo del tiempo. Muestra la secuencia de interacciones entre los objetos y el orden en que ocurren.

- Arquitectura Física: La arquitectura física se refiere a la disposición y diseño de los componentes físicos de un sistema, incluyendo servidores, dispositivos y redes. En el contexto de un proyecto de software, puede referirse a cómo se distribuyen y organizan los componentes de hardware y software.

- Servidor: Un servidor es un sistema informático que proporciona servicios o recursos a otros sistemas, conocidos como clientes. En el desarrollo web, un servidor puede referirse a la máquina física que aloja un sitio web o a la aplicación de software que maneja las solicitudes HTTP.

- APK (Android Package): Un archivo APK es el formato de archivo que se utiliza para distribuir e instalar aplicaciones en el sistema operativo Android. Contiene todos los archivos necesarios para instalar una aplicación, incluyendo el código de la aplicación, los recursos y el manifiesto de la aplicación.

- Clonar (Proyecto): Clonar un proyecto se refiere al proceso de copiar un repositorio de código existente, generalmente desde un servicio de alojamiento de código como GitHub. Esto permite a los desarrolladores trabajar en el código de forma local en su propio entorno de desarrollo, sin afectar el repositorio original.

- Pala: En el contexto de este documento, "pala" se refiere a un dispositivo externo utilizado en las tiendas Mango para leer las etiquetas RFID de los productos. Este dispositivo se conecta al sistema a través de Bluetooth.

- Impresora: En este contexto, una impresora se refiere a un dispositivo que se utiliza para imprimir etiquetas de productos en las tiendas Mango. Estas impresoras suelen ser portátiles y se conectan al sistema a través de Bluetooth.

- Bluetooth: Bluetooth es una tecnología de comunicación inalámbrica de corto alcance que permite la transferencia de datos entre dispositivos electrónicos. En el contexto de este documento, se utiliza para conectar dispositivos externos, como la "pala" y la impresora, al sistema de la tienda.

- REALM: Realm es una base de datos móvil que es una alternativa a SQLite y Core Data. Gracias a su diseño, Realm es mucho más rápido y más fácil de usar que las bases de datos ORM convencionales. Realm permite a los desarrolladores de Android y iOS trabajar con datos en vivo, en tiempo real, directamente en el

dispositivo móvil, lo que facilita la creación de aplicaciones móviles interactivas y receptivas.

Bibliografía

[1] Marín, C. (2021, junio 21). *La importancia de QA (Quality Assurance) en la empresa - Parte I*. inlogiq. Recuperado el 23 de febrero de 2024 https://inlogiq.com/la-importancia-de-qa-quality-assurance-en-la-empresa-parte-i/

[2] *Documentation*. Browserstack.com. Recuperado el 23 de febrero de 2024, de https://www.browserstack.com/docs/

[3] Espresso. Android Developers. Recuperado el 23 de febrero de 2024, de https://developer.android.com/training/testing/espresso?hl=es-419

[4] About. Selenium. Recuperado el 23 de febrero de 2024, de https://www.selenium.dev/about/

[5] Appium in a nutshell. Appium.Io. Recuperado el 23 de febrero de 2024, de https://appium.io/docs/en/latest/intro/

[6] El Alcance de una Prueba de Software. Test.io. Recuperado el 25 de febrero de 2024, de https://academy.test.io/es/articles/2683803-el-alcance-de-una-prueba-de-software

[7] Pruebas End-To-End: definición, ejemplos y herramientas. Qalified.com. Recuperado el 25 de febrero de 2024, de https://qalified.com/es/blog/end-to-end-testing/

[8] Tes. (2021, abril 22). 3 obstáculos para las pruebas continuas y cómo eliminarlos. Parasoft. Recuperado el 25 de febrero de 2024, de https://es.parasoft.com/blog/3-obstacles-to-continuous-testing-how-to-remove-them/

[9] Gandarillas, A. Análisis de valores frontera. Glosario.es. Recuperado el 26 de febrero de 2024, de https://glosario.es/istqb/nivel-basico/analisis-de-valores-frontera/

[10] Kanban vs. Scrum. (2018, octubre 11). Deloitte Spain. Recuperado el 26 de febrero de 2024, de https://www2.deloitte.com/es/es/pages/technology/articles/kanban-vs-scrum.html

[11] Atlassian. Bitbucket. Bitbucket. Recuperado el 26 de febrero de 2024, de https://bitbucket.org/product?&aceid=&adposition=&adgroup=146041784351&campaign=18815940376&creative=632894031525&device=c&keyword=bitbucket&matchtype=e&network=g&placement=&ds_kids=p74116829994&ds_e=GOOGLE&ds_eid=700000001551985&ds_e1=GOOGLE&gad_source=1&gclid=CjwKCAiArfauBhApEiwAeoB7qA1cp9TjaxCJEGHdadKDUSxNjxVgSImcjul2UAEeb9XvntC79E9pmhoCMb4QAvD_BwE&gclsrc=aw.ds

[12] Atlassian. Jira & Bitbucket integration. Atlassian. Recuperado el 26 de febrero de 2024, de https://www.atlassian.com/software/jira/bitbucket-integration

[13] Microsoft.com. Recuperado el 27 de febrero de 2024, de https://www.microsoft.com/es-es/microsoft-teams/group-chat-software

[14] Glassdoor.es. Recuperado el 5 de marzo de 2024, de https://www.glassdoor.es/Empleo/index.htm

[15] ThinkPad T14 Gen 4. Lenovo.com. Recuperado el 6 de marzo de 2024, de https://www.lenovo.com/es/es/p/laptops/thinkpad/thinkpadt/thinkpad-t14-gen-4-(14-inch-intel)/21hdcto1wwes3?cid=es:sem:34bjix&gad_source=1&gclid=CjwKCAjwzN-vBhAkEiwAYiO7oI1z2Nw3VrBKumscuEIZpv5Mpc80TlEkdtO_hPaq5Enkp3Mgcz-f-BoC6ykQAvD_BwE&gclsrc=aw.ds

[16] Mediamarkt.es. Recuperado el 6 de marzo de 2024, de https://www.mediamarkt.es/es/product/_rat%C3%B3n-con-cable-logitech-m90-conexi%C3%B3n-usb-color-negro-1119673.html?utm_source=google&utm_medium=cpc&utm_campaign=rt_shopping_generic_nsp_na_MM-ES-S-G-CAT-PLA-PMAX.PH-PROMO.SECUNDARIOS-ALL-ALL&gad_source=1&gclid=CjwKCAjwzN-vBhAkEiwAYiO7oGlsUAla_ktJAf16PiLoYyGYo0yCya3o-5OaVUV8cVKKLfIQ5hCg-BoCNh0QAvD_BwE&gclsrc=aw.ds

[17] Trust Taro Teclado - Disposición QWERTY Español, Resistente a los Vertidos, Cable de 1.8m, Conexión USB, Windows, Mac, PC, Laptop, Ordenador Portátil, Negro. Recuperado el 6 de marzo de 2024, de https://www.amazon.es/Trust-Taro-Teclado-Disposici%C3%B3n-Resistente/dp/B08DG6LBRN/ref=sr_1_4?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=13OUS3WKP9B47&dib=eyJ2IjoiMSJ9.RftX4jHqjBZT2V-_OE53uOYjAvXTNeg6eR4i4Ty7Pvvtppt4HTwKHsKZmmCQsbPen7b6o-WB057LJvUSlfNiXmKHMaz2VDhsLcZAUxGHFWi13LR_INynJBO5tFSqRxapdpJMeROhxitDXRM4go4K0PriqcPT9HIuuCiVL84pxKipto-uAmxQuQYyIVxVMkxik2koq9g4Nh2VWKDGIsuaVkYWSVOZtpBRU7rtC8Wu7JR4YlpUVuHm58tZhOrgD_UHYexCtHn5s1XB2Vk1JzAg_Kp6G_hBYgjOnqx34Z0mBmE.mZElqPgLUHAWaNMt4qPK1k0wwKUP_ct8xYMtM99Yz1E&dib_tag=se&keywords=trust+classic+line+teclado&qid=1710791124&sprefix=trust+classic+line+teclado%2Caps%2C79&sr=8-4

[18] BenQ GW2280Monitor LCD de 22 pulgadas y 1080p para la oficina en casa. Recuperado el 6 de marzo de 2024, de https://www.amazon.es/BenQ-GW2280Monitor-pulgadas-1080p-oficina/dp/B07CQ3MYX3/ref=asc_df_B07CQ3MYX3/?tag=googshopes-21&linkCode=df0&hvadid=305031262934&hvpos=&hvnetw=g&hvrand=14715708855037064 60&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9181160&hvtargid=pla-537251884445&psc=1&mcid=bc9e0ea14b29358089bc58e0ce7ddcc3

[19] Agencia Tributaria: Inicio. Gob.es. Recuperado el 10 de marzo de 2024, de https://sede.agenciatributaria.gob.es/Sede/ayuda/manuales-videos-folletos/manuales-practicos/folleto-actividades-economicas/3-impuesto-sobre-renta-personas-fisicas/3_5-estimacion-directa-simplificada/3_5_4-tabla-amortizacion-simplificada.html

[20] Mesa de colectividad people 70. Ofiprix. Recuperado el 6 de marzo de 2024, de https://www.ofiprix.com/es/mesa-de-colectividad-people-70#/tamano-100x70_cm/colorvalid-gris_tormenta

[21] Silla de oficina Axel, espuma inyectada. Ofiprix. Recuperado el 6 de marzo de 2024, de https://www.ofiprix.com/es/silla-de-oficina-axel-espuma-inyectada#/colorvalid-negro

[22] Idealista.com. Recuperado el 16 de marzo de 2024, de https://www.idealista.com/

[23] Idealista.com. Recuperado el 16 de marzo de 2024, de https://www.idealista.com/inmueble/99318146/