IMPLEMENTACIÓN DE PRUEBAS FUNCIONALES AUTOMATIZADAS CON SELENIUM, APPIUM Y REST-ASSURED EN LA EMPRESA VALID COLOMBIA

PRESENTADO POR:

Juana Andrea Orozco Díaz

Código:
2010114074

PRESENTADO A:

Ing. Luis Alberto Pava Carmona
Tutor de Prácticas Profesionales

José Consuegra
Tutor Empresarial

Universidad del Magdalena
Facultad de Ingeniería
Ingeniería de Sistemas
Santa Marta D.T.C.H
30/12/2021

Tabla de Contenido

Tabla de Ilustraciones

1. PRESENTACIÓN

En este documento se describen las actividades principales que se desarrollaron para la empresa Valid Colombia, donde se busca implementar un nuevo proceso enfocado al área de calidad de software. El objetivo principal en el desarrollo de este proyecto es establecer un nuevo modelo de trabajo que le permita a la compañía optimizar los tiempos de prueba funcionales en los distintos proyectos de software, tales como DaviPlata, Super App Davivienda y Super App Cívica, que juntos conforman los grupos de trabajo más grandes y demandantes en la actualidad.

Gracias al trabajo realizado se logró establecer una ruta para implantar un nuevo proceso dentro del área de desarrollo que abarca: un proceso completo en el que se establecen las distintas tareas que se deben ejecutar para la creación de scripts automatizados y los lineamientos para la creación los mismos. Asimismo, se logró establecer el proceso adecuado para la integración de las pruebas funcionales automatizadas de manera remota utilizando granjas de dispositivos alojados en AWS.

2. OBJETIVOS Y/O FUNCIONES

2.1. Objetivo General:

Implementación de pruebas funcionales automatizadas con Selenium, Appium y Rest-Assured.

2.2. Objetivos Específicos:

1. Investigar sobre el uso de frameworks para el desarrollo de casos de prueba funcionales automatizados
2. Crear lineamientos de automatización de pruebas de software para Selenium, Appium y Rest-Assured.
3. Diseñar proyectos base para pruebas de interfaz automatizadas
4. Generar un diagrama del proceso de automatización de pruebas funcionales
5. Implementar las pruebas funcionales automatizadas en los proyectos más críticos

2.3. Funciones del practicante en la organización:

- Liderar el proceso de automatización de pruebas para la compañía
- Actualizarse en nuevas tecnologías y mejores prácticas de testing.
- Tener un conocimiento especializado en pruebas para desarrollo de software
- Liderar el equipo de QA y asegurar el cumplimiento del proceso de QA.
- Apoyar a los involucrados en el Proyecto en preguntas técnicas y ofertas.
- Asegurar que el equipo cumpla con las buenas prácticas de pruebas establecidas por el área.
- Liderar la interacción y colaboración con otros equipos dentro del área de desarrollo.
- Apoyar en la creación de estimaciones de desarrollo de alto nivel.

- Conducir reuniones regulares con el equipo de QA para monitoreo y capacitación.
- Capacitar al equipo para que sus pruebas sean robustas, con un alto desempeño modular y escalables.
- Realizar el entrenamiento al cargo de los nuevos miembros de equipo y capacitar a los actuales cuando un proyecto requiere una tecnología que el equipo no conoce.
- Mantener el contacto directo con el equipo fortaleciendo el sentido de pertenencia y la comunicación de necesidades de estos
- Liderar el apoyo técnico en los procesos de selección que se requieran dentro del área.

3. JUSTIFICACIÓN:

Durante varios años dentro de la compañía se vienen llevando a cabo procesos de desarrollo de software que implican tanto el desarrollo como pruebas. Con el paso del tiempo las pruebas de software han ido aumentando a medida que crecen los aplicativos y los distintos requerimientos, por lo que se surgió la necesidad de implementar un proceso que ayude a optimizar el tiempo de las pruebas manuales funcionales, esto implica pruebas de regresión, pruebas de humo y pruebas de aceptación del usuario. Dentro de los proyectos más grandes un desarrollo nuevo puede estar costando alrededor de 2 meses, de los cuales 1 está enfocado en la certificación del aplicativo, esto es costoso tanto para el equipo de calidad de la compañía como para el cliente. Actualmente el proceso de pruebas se ejecuta prácticamente al final del desarrollo en algunos de estos proyectos, por lo que surgen muchos errores que implican corrección, este es el momento en el que se sufre debido a las muchas pruebas de regresión y de humo que deben ejecutarse.

Por esta razón, se encontró que una buena práctica sería la implementación de pruebas automatizadas, ya que reemplazarían los procesos repetitivos, tardíos y dispendiosos de las pruebas manuales, de esta forma el equipo de testing podría enfocarse en otras actividades que representen un mayor valor para la compañía. El desarrollo de las pruebas automatizadas busca brindar un retorno económico a largo plazo, ya que se suprimirían tiempos extensos de regresión y pruebas complejas durante los entregables.

4. GENERALIDADES DE LA EMPRESA:

Valid es una empresa multinacional de origen brasilero, cuenta con presencia en distintos países de América, Europa y África. Actualmente la sede en Colombia se encarga principalmente de brindar soluciones tecnológicas a nivel pago, móviles, soluciones de identidad y datos, así como de Marketing Digital y certificación digital. De igual forma se desarrollan soluciones digitales que permiten a las empresas, agencias gubernamentales, consumidores y ciudadanos en general identificarse a sí mismos, comunicar y realizar transacciones financieras o de datos totalmente seguras. La sede en Colombia se encuentra ubicada en la ciudad de Bogotá, la compañía a nivel global cuenta con más de 6.000 empleados.

Historia.

- Valid inicia sus operaciones en Brasil, bajo el nombre de Thomas de La Rue, como filial de una compañía inglesa que presta servicios de papel de alta seguridad y tecnología de impresión en el año 1957.

- Después de tres décadas (1993), American Banknote compra Thomas de La Rue, lo que supone el comienzo de una etapa de gran expansión. La compañía pronto se convierte en un referente en Brasil para la emisión segura de documentos.

- En el año 2006 La compañía lanza una oferta pública inicial con el 57% de su capital social, consiguiendo un total de 480 millones de reales brasileños.

- En 2007 La cartera de soluciones de Medios de Pago ya están disponibles en Argentina.

- En el año 2010 La empresa cambia de nombre convirtiéndose en Valid, una compañía muy importante que empieza a operar en Europa, estableciendo las bases para una nueva era.

- Para el año 2012 y con presencia en Brasil, Argentina y España, Valid empieza a consolidar su presencia en Estados Unidos, ofreciendo soluciones de Data, Pago y Mobile.

- Valid amplía su cartera de soluciones en Estados Unidos ofreciendo la solución de Identificación Segura en el 2013.

- En el 2015 Las nuevas adquisiciones refuerzan la presencia de Valid en Estados Unidos, Europa, África, Asia y Oriente Medio, expandiendo su presencia en los cinco continentes.

- En 2017 Valid se une al universo IoT a través de su inversión en Cubic Telecom.

Visión.

Establecer con cada cliente relaciones a largo plazo, basadas en la confianza mutua.

Misión.

Proporcionamos servicios seguros y personalizados para la identificación de personas, objetos y transacciones.

Valores corporativos.

- Foco en el cliente
- Integridad y Confianza
- Resultados y Excelencia
- Simplicidad y Creatividad

Organigrama

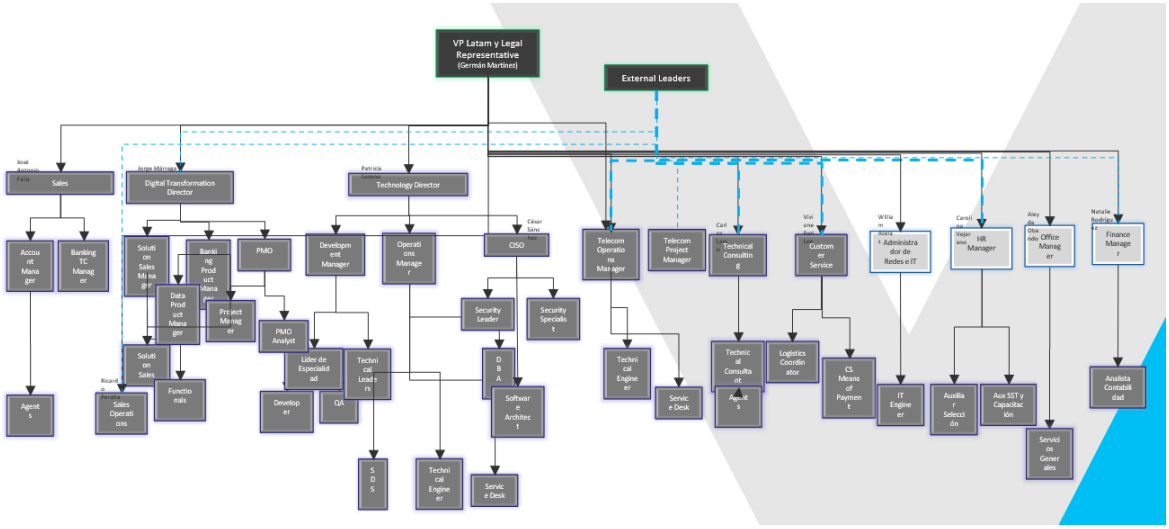

Ilustración 1. Organigrama Valid

Proceso de soluciones.

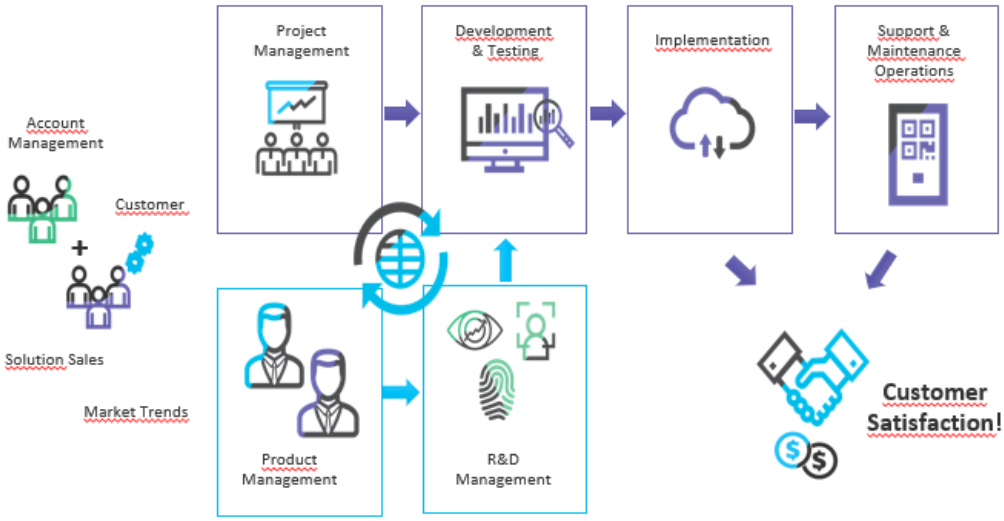

Ilustración 2. Proceso de Soluciones Tecnológicas

Proceso de desarrollo de software.

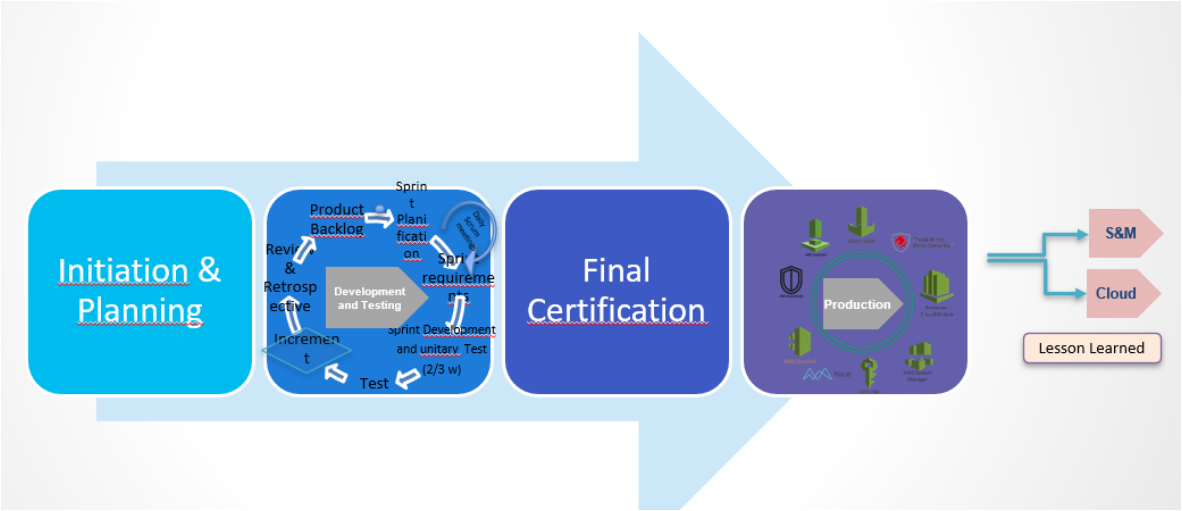

Ilustración 3. Proceso de desarrollo de software

El enfoque principal del trabajo será dentro del área de desarrollo. Esta área se encuentra a cargo del líder de desarrollo (Jose Consuegra) y al mismo tiempo bajo la dirección de Tecnología (Patricia Lozano). Dentro del área de desarrollo se

encuentran líderes de equipo y líderes de tecnologías. El líder de equipo es quien toma el mando de uno o varios proyectos y se encuentra en contacto directo con todo el equipo de desarrollo (desarrolladores y QA). El líder de tecnología gestiona y supervisa el trabajo de equipos divididos por tecnología, en este caso se cuenta con un líder para desarrolladores de móviles, uno para desarrolladores backend y otro para el equipo de calidad (QA).

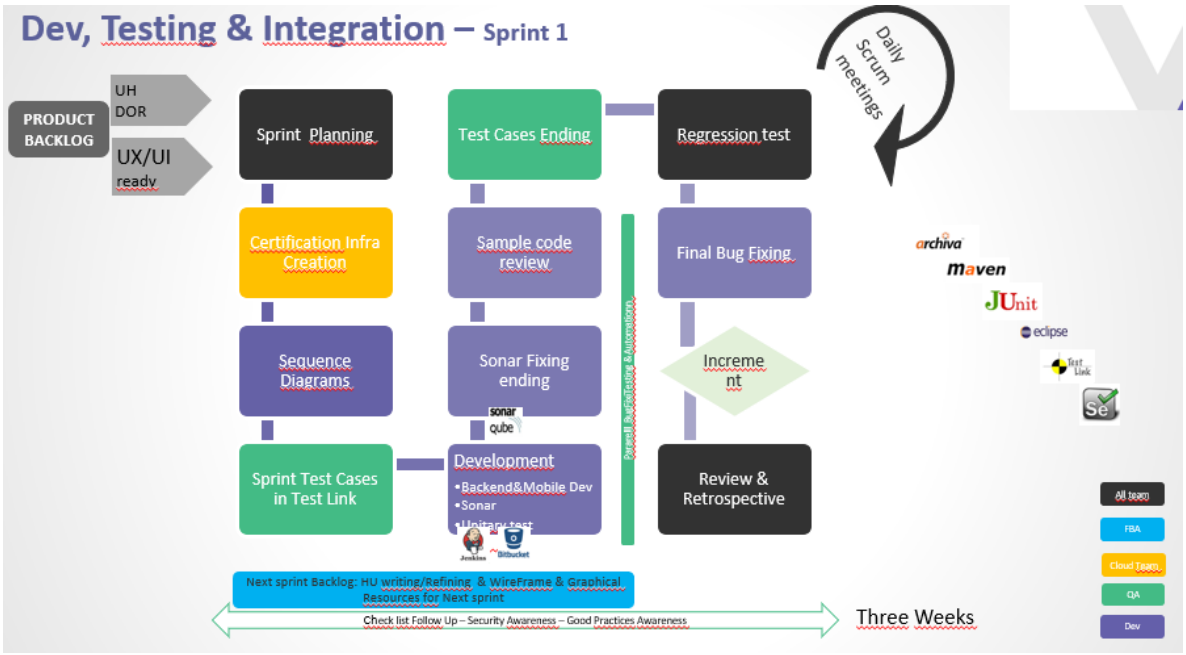

Ilustración 4. Actividades de desarrollo

Con el apoyo de cada uno de los miembros de los equipos se asignan proyectos que se desarrollan basados en metodologías ágiles, uno de los principales enfoques de la compañía es el uso de Scrum como metodología de desarrollo de software, aunque aún existen proyectos que no han adoptado esta forma de trabajo.

5. SITUACIÓN ACTUAL

El ritmo acelerado que se lleva actualmente dentro del desarrollo de software cada vez se va haciendo más necesaria la implementación de procesos que permitan agilizar los tiempos de entrega, actualmente uno de los procesos más costosos para la compañía es el tiempo de pruebas. En los últimos dos años se han visto afectados varios proyectos tanto en tiempos como en costos debido a la gran cantidad de tiempo que es requerido para culminar las pruebas de regresión. Los nuevos desarrollos para algunos proyectos de gran escala implican entregas continuas cada 15 o 20 días. Dentro de los proyectos más grandes que se tienen, una regresión puede llegar a costar 15 días, y este tiempo solo se incrementa más y más a medida que se agregan funcionalidades y requerimientos. Es por esto, que para algunos casos la entrega consume más tiempo de pruebas que del propio desarrollo.

El problema es aún más notorio cuando se espera utilizar metodologías ágiles, pues se debe compensar el tiempo de entrega sacrificando parte de las pruebas, esto a su vez genera riesgos de que existan errores en desarrollos anteriormente testeados.

El flujo del proceso actual de QA se encuentra documentado dentro del diagrama de procesos de la empresa, como se puede observar, se dividen varias etapas del ciclo de vida del software, y es durante la etapa de diseño y desarrollo en donde se cuenta con mayor actividad por parte del equipo de QA.

Etapa de inicio.

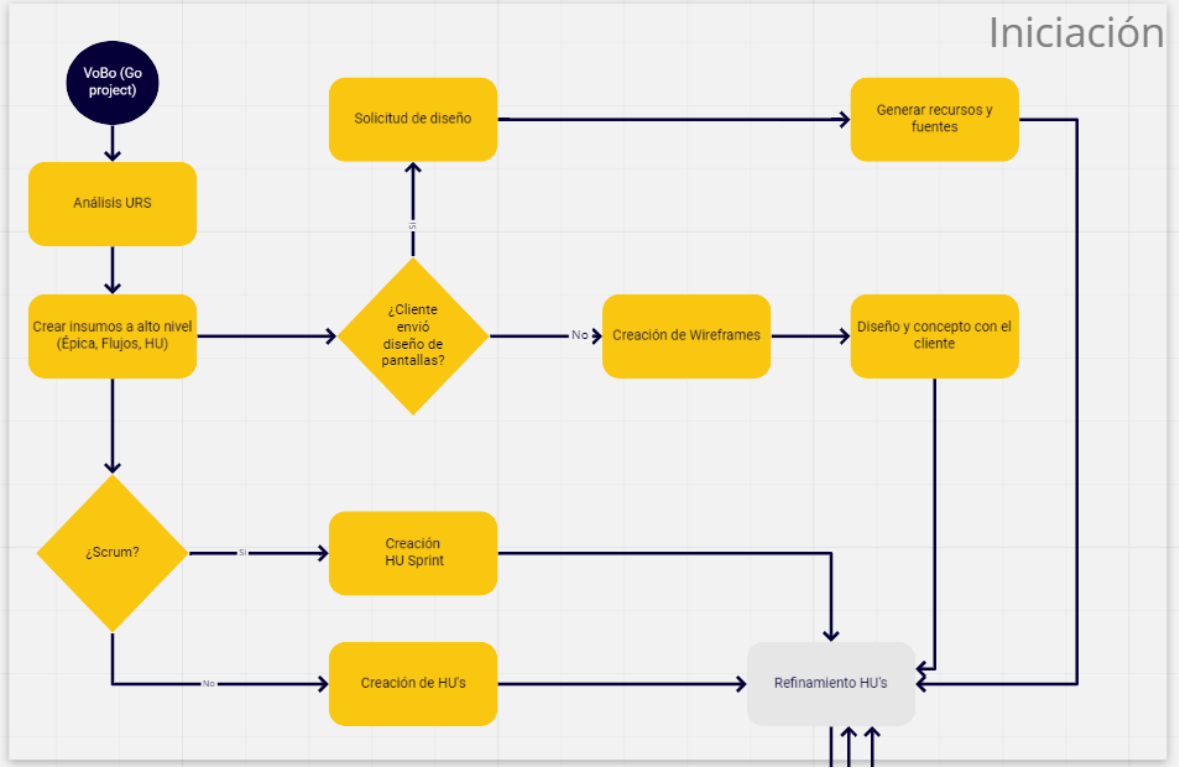

Ilustración 5. Etapa inicial del proyecto

Fase de diseño.

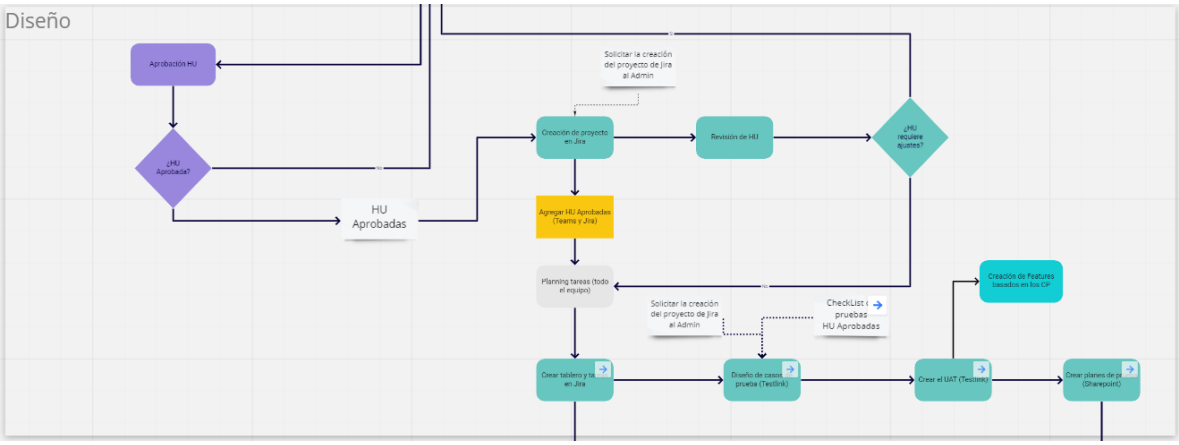

Ilustración 6. Etapa de diseño

Fase de desarrollo y testing.

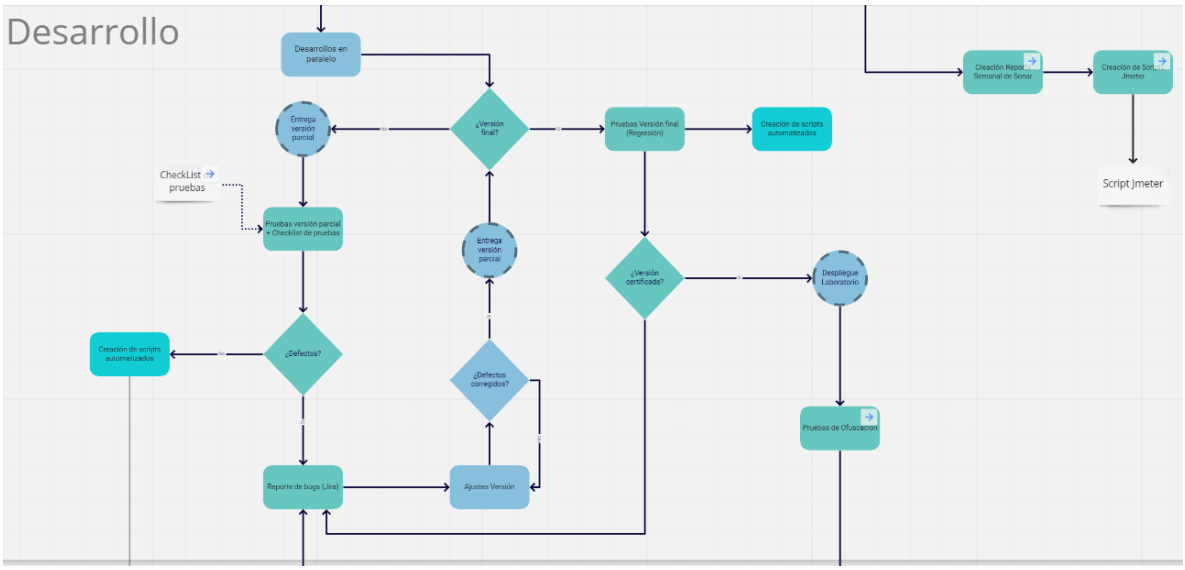

Ilustración 7. Etapa de desarrollo y testing

Fase de certificación de pruebas.

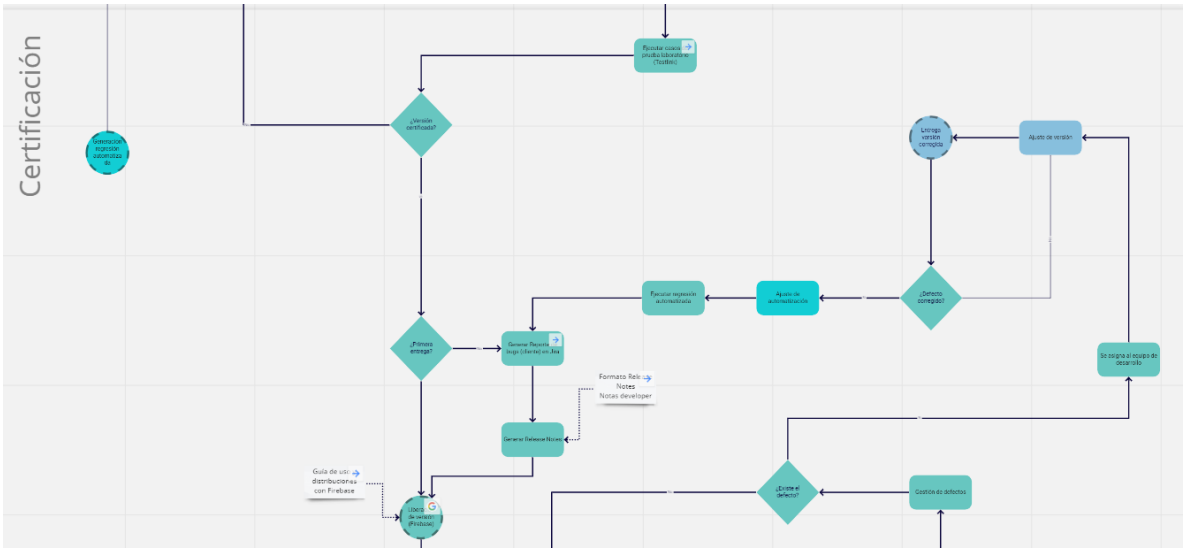

Ilustración 8. Etapa de certificación de software

Fase productiva.

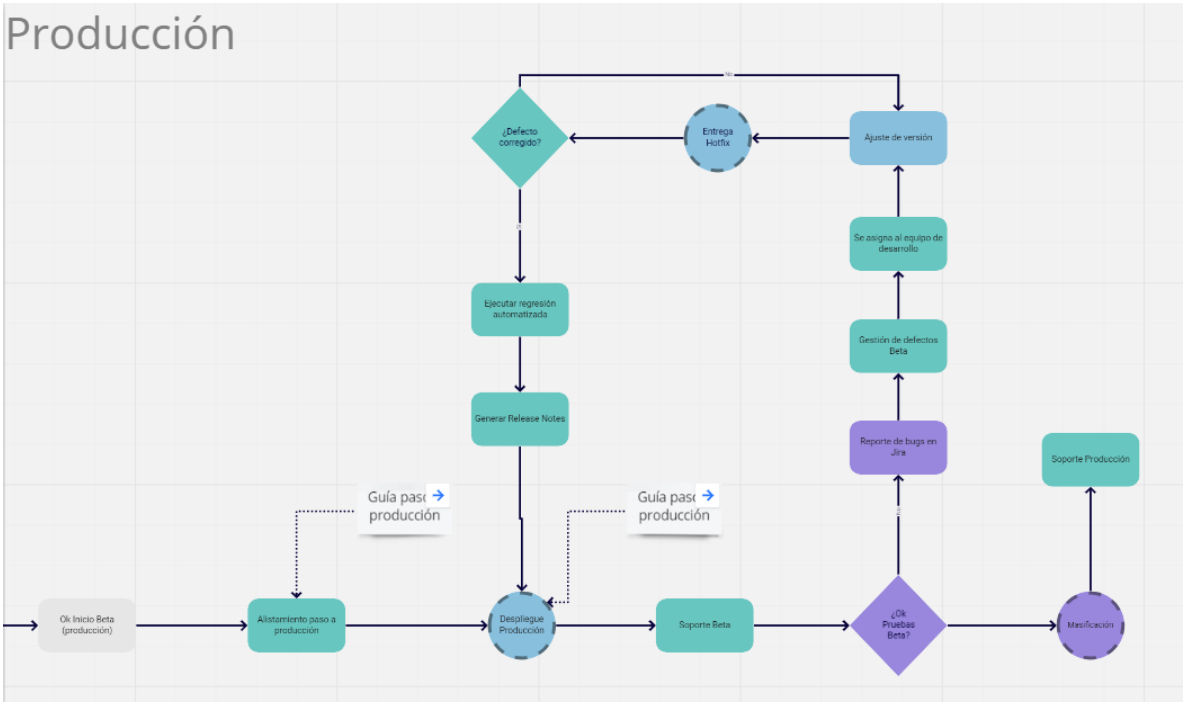

Imagen 1 Etapa de producción

6. BASES TEÓRICAS RELACIONADAS

Para el desarrollo de este proyecto fueron necesarias varias herramientas de desarrollo de software, ejecución de pruebas y conocimientos adquiridos durante y después de la carrera, tales como:

Appium.

Appium es un marco de automatización de pruebas de código abierto para usar con aplicaciones web nativas, híbridas y móviles. Controla aplicaciones de iOS, Android y Windows mediante el protocolo WebDriver.

Appium emplea el *WebDriver* de Selenium, que es un entorno de pruebas para aplicaciones web y que especifica un protocolo cliente-servidor conocido como *JSON Wire Protocol*. Esto permite al cliente el uso de marcos de pruebas escritos en cualquier lenguaje, enviando las solicitudes HTTP apropiadas al servidor.

Actualmente es utilizado para llevar a cabo la automatización de pruebas para móviles, ya que es el fuerte del desarrollo en la compañía.

AWS Device Farm.

AWS Device Farm es un servicio de prueba de aplicaciones que permite realizar pruebas automatizadas sobre aplicaciones web y móviles al presentar un amplio rango de navegadores de escritorio y dispositivos móviles reales, sin tener que aprovisionar y administrar ninguna infraestructura de pruebas. El servicio permite ejecutar las pruebas de forma concurrente en varios navegadores de escritorio y dispositivos reales. Además, genera videos y registros durante la ejecución de las pruebas.

El servicio fue adquirido para la ejecución de pruebas automatizadas de manera remota, por lo que se realizó una integración de los scripts automatizados y la granja de dispositivos de Amazon.

Bitbucket.

Bitbucket Cloud es una herramienta de alojamiento de código y colaboración basada en Git. Permite integrar distintas herramientas de colaboración en el desarrollo de software, así como generar un flujo completo de CI/CD para los equipos.

Dentro de la compañía es utilizado principalmente para el manejo del código fuente de las aplicaciones y para la compilación automática realizada a través de la herramienta de Jenkins.

Cucumber.

Cucumber es una de las herramientas que se utiliza para automatizar pruebas en BDD. Cucumber permite ejecutar descripciones funcionales en texto plano como pruebas de software automatizadas.

Fue creada en 2008 por Aslak Hellesoy y está escrito en Ruby, aunque tiene implementaciones para casi cualquier lenguaje de programación: JRuby (usando Cucumber-JVM), Java, Groovy, JavaScript, JavaScript (usando Cucumber-JVM y Rhino), Clojure, Gosu, Lua, .NET (usando SpecFlow), PHP (usando Behat), Jython, C++ o Tcl.

Cucumber es utilizado principalmente para la redacción de los escenarios de prueba o casos de prueba dentro del código. A través de estos escenarios es posible ejecutar las pruebas automatizadas siguiendo el paso a paso descrito.

Ingeniería del software.

La ingeniería del software es una disciplina que implica el uso de estructuras, herramientas y técnicas para construir programas informáticos.

Asimismo, incluye el análisis previo de la situación, la redacción del proyecto, la creación del software y las pruebas necesarias para garantizar su correcto funcionamiento antes de que el sistema esté operativo.

Esta ingeniería aborda todas las fases del ciclo de vida de desarrollo de cualquier tipo de sistema de información y es aplicable a una amplia gama de ámbitos de la informática y la ciencia de los ordenadores, como el diseño de compiladores, sistemas operativos y tecnologías de Intranet/Internet: la empresa, la investigación científica, la medicina, la fabricación, la logística, la banca, el control del tráfico y la meteorología son solo algunos de los campos en los que puede trabajar.

Jenkins

Jenkins es un servidor open source para la integración continua. Es una herramienta que se utiliza para compilar y probar proyectos de software de forma continua, lo que facilita a los desarrolladores integrar cambios en un proyecto y entregar nuevas versiones a los usuarios. Escrito en Java, es multiplataforma y accesible mediante interfaz web. Es el software más utilizado en la actualidad para este propósito.

Con Jenkins, las organizaciones aceleran el proceso de desarrollo y entrega de software a través de la automatización. Mediante sus centenares de plugins, se puede implementar en diferentes etapas del ciclo de vida del desarrollo, como la compilación, la documentación, el testeo o el despliegue.

Lenguaje de programación Java

Java es una plataforma informática y a su vez un lenguaje de programación creado en 1995 por la empresa Sun Microsystem. El objetivo de este lenguaje es que los programadores sólo tuvieran que escribir el código de un programa una vez, y que éste, pudiese ejecutarse en cualquier dispositivo. Esto es posible gracias a la Máquina Virtual de Java (JVM), que brinda esa portabilidad necesaria.

Con Java se pueden crear programas en una gran variedad de dispositivos, permitiendo ejecutar la misma aplicación en diversos sistemas operativos.

El nombre inicial con el que se iba a denominar era Oak, pero al estar la marca registrada se optó por Java.

Java es un lenguaje orientado a objetos, independiente de la plataforma hardware donde se desarrolla, y que utiliza una sintaxis similar a la de C++ pero reducida. Es un lenguaje con una curva de aprendizaje baja (se puede decir que es fácil de aprender) y que dispone de una gran funcionalidad de base (incrementada por la gran cantidad de código de terceros existente). Java, como lenguaje de programación, ofrece un código robusto, que ofrece un manejo automático de la memoria, lo que reduce el número de errores.

La comunidad de programadores Java existente es muy extensa, en torno a los 9 millones en todo el mundo, y muy activa, lo que genera una gran cantidad de recursos actualizados.

Lenguaje Gherkin

Gherkin es un DSL o Lenguaje Específico de Dominio (Domain-Specific Languaje), es decir, un lenguaje que está creado para resolver un problema.

Tiene una estructura generada por varios elementos, como vemos en la siguiente imagen.

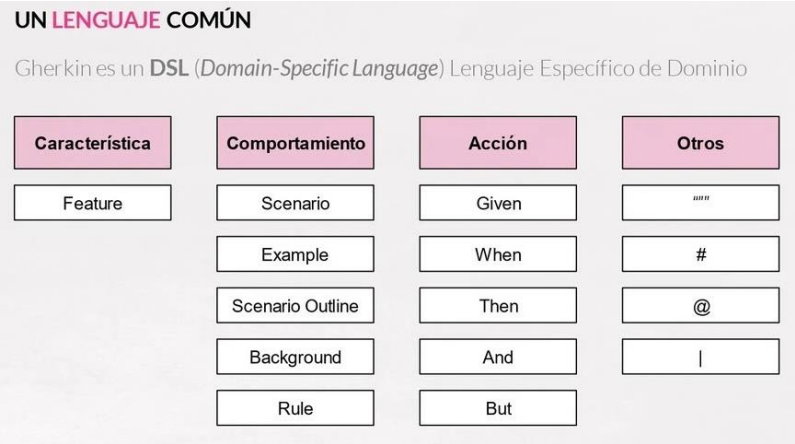

Ilustración 9. Características lenguaje Gherkin

Estos elementos nos ayudan a que todos esos comportamientos vayan poco a poco bajando de nivel, hasta llegar a un lenguaje que entiendan fácilmente los desarrolladores.

Los elementos más utilizados habitualmente son Feature, Scenario, Example, Scenario Outline, Given, When, Then y And. El resto son un poco más desconocidos en general o bien que se han publicado recientemente. Ejemplo de un escenario escrito en gherkin:

- **Feature:** Realizar un curso online
- **Scenario:** Seleccionar un curso comenzado
- **Given** un usuario con un curso comenzado
- **When** selecciona un curso comenzado
- **Then** arranca el curso desde donde lo dejó

Metodología Scrum

La metodología Scrum es un proceso para llevar a cabo un conjunto de tareas de forma regular con el objetivo principal de trabajar de manera colaborativa, es decir, para fomentar el trabajo en equipo.

Con este método de trabajo lo que se pretende es alcanzar el mejor resultado de un proyecto determinado. Las prácticas que se aplican con la metodología Scrum se retroalimentan unas con otras y la integración de estas tiene su origen en un estudio de cómo hay que coordinar a los equipos para ser potencialmente competitivos.

En Scrum se van realizando entregas regulares y parciales del trabajo final, de manera prioritaria y en función del beneficio que aportan dichas entregas a los receptores del proyecto. Por este motivo, es una metodología especialmente indicada para proyectos complejos, con requisitos cambiantes y en los que la innovación y la flexibilidad son protagonistas.

Node.js

Node.js, es un entorno en tiempo de ejecución multiplataforma para la capa del servidor (en el lado del servidor) basado en JavaScript.

Node.js es un entorno controlado por eventos diseñado para crear aplicaciones escalables, permitiéndote establecer y gestionar múltiples conexiones al mismo tiempo. Gracias a esta característica, no tienes que preocuparte con el bloqueo de procesos, pues no hay bloqueos.

Patrón de diseño Screenplay

El patrón de screenplay tiene un enfoque de desarrollo encaminado por comportamiento Behaviour Driven Development (BDD). Esta no es una herramienta para testing sino una estrategia de desarrollo que se enfoca en prevenir defectos en

lugar de encontrarlos en un ambiente controlado, tal y como se explica en el artículo BDD para la automatización de pruebas

Para automatizar procesos, también se utiliza la herramienta Cucumber para implementar metodologías como BDD, las cuales permiten ejecutar descripciones funcionales escritas en texto plano como pruebas de software automatizadas.

Para la implementación en los temas de la automatización, debemos tener en cuenta que se deben manejar ciertos conceptos que abarquen los siguientes elementos: Runners, User Interface, Features, StepDefinitions, task, Interactions, Questions y Reports.

Runners: es el ejecutor de los features, tags, glue (donde se encuentra el step definition), URL del driver y la conexión con excel del drive.

User Interface: las UI son el mapeo de la interfaz, donde capturaremos todos los elementos con los cuales podríamos llegar a interactuar durante la automatización. Además, se le puede añadir la URL donde se iniciará la prueba.

Features: los features son las historias de usuario que se llevarán a cabo en las pruebas y proveerá los métodos que utilizaremos más adelante para los StepDefinitions.

StepDefinitions: los Step Definitions son la traducción de los features a código. Los métodos que se utilizaran son los features (historias de usuario), por lo tanto, iremos a la clase “RunnerTags”, le daremos clic derecho sobre ésta, y en la opción “Run as” escogemos “JUnit Test”.

Task: son las interacciones que se llevarán a cabo para cumplir con las historias de usuarios planteadas. Las tasks se pueden caracterizar porque no se habla en términos de clic, set data o select. Son verbos más amplios como ingresar a un formulario, cerrar sesión y buscar.
Interactions: indicar acciones como dar clic, select, enviar datos, scroll, entre otras cosas.
Questions: son lo assert a llevar a cabo para asegurar el cumplimiento de ciertos parámetros.

Programación orientada a objetos

La Programación Orientada a Objetos (POO) es un paradigma de programación, es decir, un modelo o un estilo de programación que nos da unas guías sobre cómo trabajar con él. Se basa en el concepto de clases y objetos. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.

Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma. Esto nos ayuda muchísimo en sistemas grandes, ya que, en vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.

Pruebas automatizadas

Definir las pruebas automatizadas es muy fácil. Por ejemplo, el proceso de ejecutar varias pruebas una y otra vez sin ejecutarlas manualmente se conoce como pruebas automatizadas. Lo único que lo hace diferente de las pruebas manuales es que las pruebas automatizadas utilizan una herramienta de automatización como LambdaTest para ejecutar los scripts de prueba.

Existen varios tipos de pruebas automatizadas como, por ejemplo:

- Automatización de Pruebas Unitarias
- Pruebas Automatizadas de un API
- Automatización de pruebas de Interface Gráfica

Pruebas de software

Las pruebas de software (Software Testing) comprenden el conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación, por medio de pruebas sobre el comportamiento de este.

Los sistemas informáticos, programas y aplicaciones han crecido a niveles inimaginables en complejidad e interoperabilidad, con lo cual también se han incrementado las posibilidades de defectos (bugs), a simple vista insignificantes, pero que pudieran adquirir proporciones catastróficas.

Además, factores como el uso de software de terceros desde aplicaciones móviles han añadido niveles adicionales de complejidad y por ende incrementado los posibles puntos de fallas.

Frente a esto, el reto de los profesionales de Software Testing es modernizar sus metodologías, tecnologías y herramientas que les permitan automatizar tareas, ejecutar ciclos de pruebas más rápidos y así reducir al mínimo las posibilidades de errores en el Software.

Rest-Assured

Es un Framework escrito en Java y diseñado para simplificar las pruebas sobre servicios basados en REST.

Ofrece un DSL descriptivo (lenguajes específicos de dominio), que muestra una unión a un punto de conexión HTTP y da los resultados esperados. Este framework soporta las operaciones POST, GET, PUT, DELETE, OPTIONS, PATCH y HEAD y contiene herramientas para invocarlas y verificarlas.

Selenium

Selenium es un conjunto de utilidades que facilita la labor de obtener juegos de pruebas para aplicaciones web. Para ello nos permite grabar, editar y depurar casos de prueba, que podrán ser ejecutados de forma automática e iterativa posteriormente.

Además de ser una herramienta para registrar acciones, permite editarlas manualmente o crearlas desde cero. Las acciones se basan en el uso de diferentes API's en diferentes lenguajes (PHP, Ruby, JAVA, Javascript, etc.). Entre sus principales características podemos nombrar:

- Facilidad de registro y ejecución de los test.
- Referencia a objetos DOM en base al ID, nombre o a través de XPath.
- Autocompletado para todos los comandos.
- Las acciones pueden ser ejecutadas paso a paso.
- Herramientas de depuración y puntos de ruptura (breakpoints).
- Las pruebas pueden ser almacenados en diferentes formatos.

El potencial de esta herramienta puede ser utilizado para la grabación de las pruebas funcionales durante la Generación de pruebas de regresión. Con este servicio se consigue obtener una batería de pruebas automatizadas que podrán ser utilizadas cuando sea necesario repetir las pruebas.

Serenity BDD.

Es una librería de código abierto que ayuda a escribir pruebas de aceptación automatizadas de mayor calidad y más rápido, ya que facilita hacer BDD centralizando los test de una aplicación.

Sus principales características son:

1. Escribir test flexibles y fáciles de mantener
2. Generar informes ilustrativos sobre las pruebas, sirve como documentación viva
3. Ajustar las pruebas automatizadas a las necesidades del proyecto
4. Ver cuanto de la aplicación se está probando

Para el desarrollo del proyecto se utilizó el marco de trabajo o framework basado en esta herramienta en conjunto con Cucumber BDD. La combinación de estas dos librerías permite utilizar los escenarios descritos en un lenguaje a nivel de negocio en conjunto con los informes, por lo que es muy sencillo comprender qué se está probando.

7. DESARROLLO DE ACTIVIDADES:

Consiste en hacer una descripción detalla y en forma cronológica de todas las actividades realizadas durante el periodo de práctica. Ya sea en la realización del proyecto (en la cual debe tener relación con los objetivos) o en el desarrollo de las actividades solicitadas en sus funciones de trabajo y complementarias.

Debe mostrar evidencias relacionadas con el desarrollo de la actividad (si son muchas se agrega con anexo, con la debida autorización de la empresa).

Para el desarrollo de este proyecto se requirió trazar una línea crítica de tareas que lleven a completar el objetivo principal del proyecto, cada una de estas tareas fue manejada a través de la herramienta de seguimiento Jira Software. En este caso cada una de las tareas requeridas para la implementación de la automatización en los proyectos fue detallada y se realizó un seguimiento sobre el avance de estas.

1. **Investigación sobre tecnologías actuales para la automatización de pruebas de software.**

   La primera actividad consistió en la realización de una investigación sobre las nuevas tecnologías utilizadas para la automatización de pruebas. Se realizaron consultas en internet, partiendo de documentos antiguos que se tenían en la empresa para la implementación de la automatización con Appium y Selenium.

   Las primeras consultas se realizaron para escoger la tecnología de trabajo para la automatización sobre móviles y web, en este caso ya se contaba con Appium y Selenium como opciones principales, sin embargo, se encontraron otras herramientas que podrían ser de utilidad, pero debido a las limitaciones

o el costo que tienen fueron descartadas, como el caso de Calabash, Roboto, Katalon Studio o Espresso.

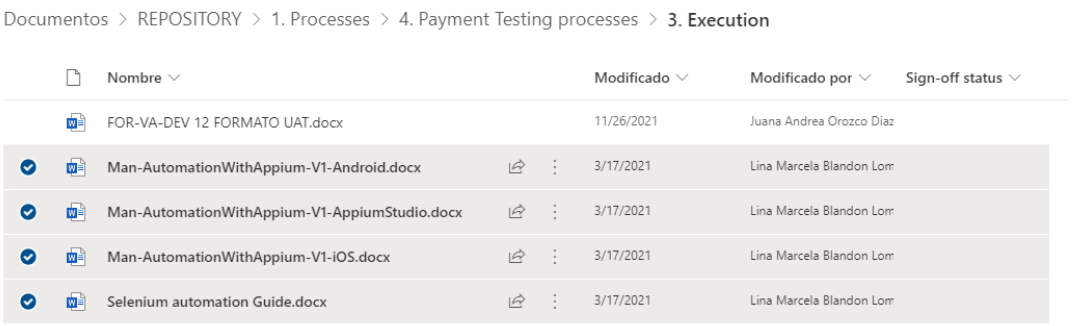

Ilustración 10. Formatos antiguos de automatización

Se decidió continuar con Appium y Selenium utilizando Java como lenguaje de programación, asimismo se decidió utilizar el IDE de desarrollo de Eclipse, posteriormente cambiaría a Intellij IDEA debido a la facilidad en el uso de ciertos plugins.

Posteriormente se realizaron consultas para la implementación de automatización de pruebas en APIS, para este caso no contábamos con ninguna guía y partimos de un curso en Udemy que contemplaba las pruebas de principio a fin utilizando el framework de Cucumber y Rest-Assured como principal librería para el manejo de las peticiones.

2. Capacitación sobre nuevas tecnologías encontradas

Para lograr implementar las tecnologías que se escogieron, fue necesario realizar varios cursos, la empresa nos da un usuario de Udemy for Business, por lo que fue de gran ayuda al momento de investigar sobre cómo implementar las pruebas de manera organizada. Se seleccionaron 3 cursos para avanzar:

- **Appium-Mobile Testing (Android/iOS) from Scratch + Frameworks**. Este curso fue utilizado para aprender como automatizar pruebas de interfaz gráfica en dispositivos Android y iOS.

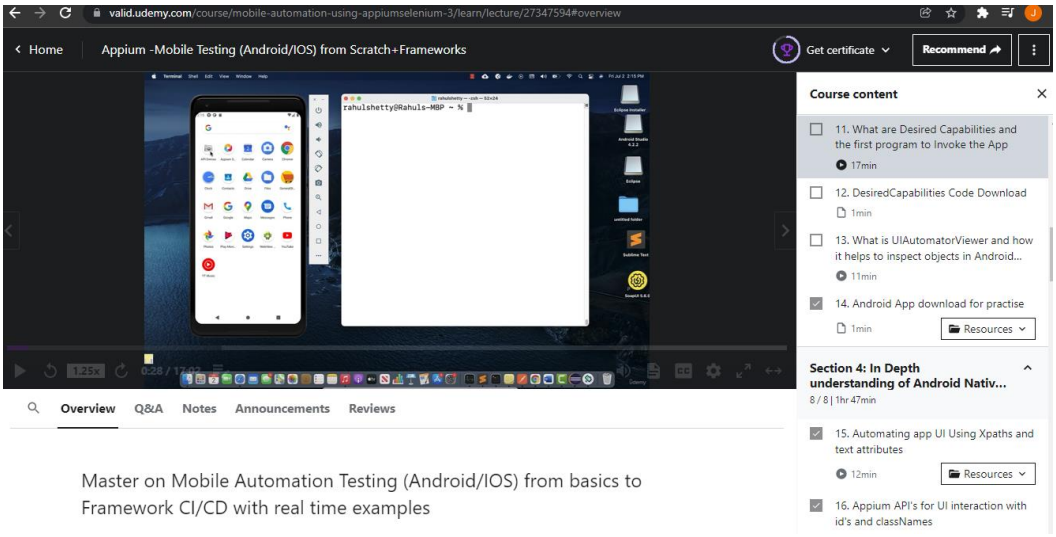

Ilustración 11. Curso automatización con Appium

- **Rest API Testing (Automation) from Scratch-Rest Assured Java.** Este curso comprendía todo el manejo de pruebas automatizadas para APIS

- **Selenium WebDriver with Java – Basics to Advanced + Frameworks.** Este curso comprende el uso de Selenium para la automatización de aplicaciones Web.

Al finalizar cada uno de estos cursos se fueron construyendo a la par las bases para la implementación de las pruebas automatizadas los distintos proyectos, según su necesidad.

3. Elección de tecnología para la implementación de la automatización de pruebas

Una vez completados los cursos se decidió optar por utilizar como lenguaje de programación Java. Debido a que es el lenguaje afín al utilizado en la compañía para la mayoría de sus procesos, y fue el lenguaje utilizado en la

materia de programación orientada a objetos, por lo que su uso sería el más conveniente.

En este caso optamos por uso de Appium como librería para automatizar aplicativos móviles, ya que permite trabajar con aplicaciones nativas (Android y iOS) así como aplicativos híbridos, los cuales también son frecuentemente utilizados en la empresa.

Inicialmente se realizará la automatización en aplicativos Android, debido a la falta de equipos Mac para los automatizadores, en este momento contamos con dispositivos Windows, lo que impide el trabajo con iOS.

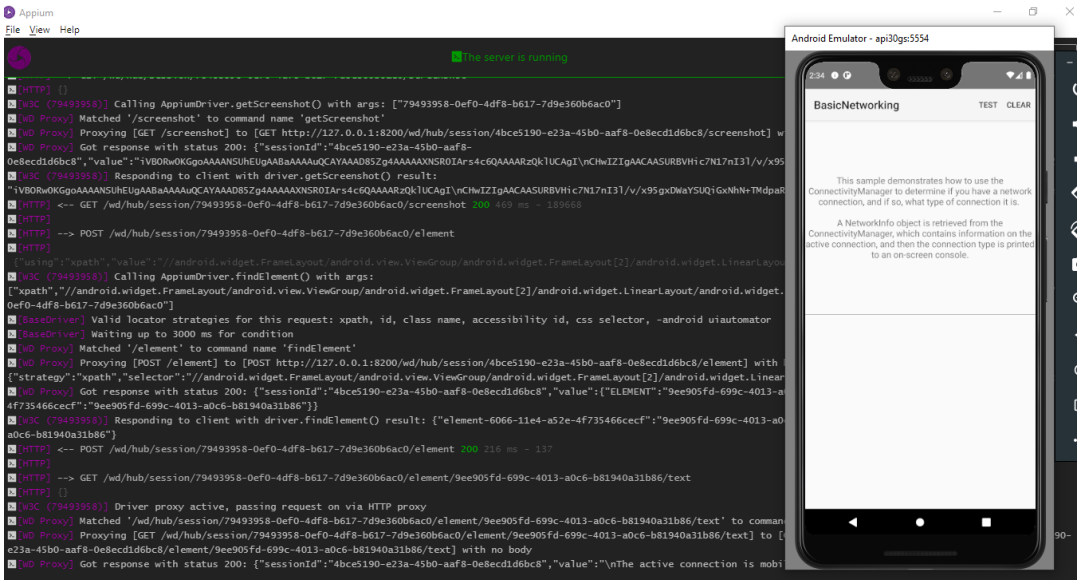

Ilustración 12. Ejecución de pruebas en Appium

Adicionalmente se utilizará Selenium como librería para las pruebas automatizadas en WEB. Selenium es la herramienta más extendida para esta función, por lo que es una de las mejores alternativas encontradas, adicionalmente es compatible con muchos lenguajes de programación y originalmente fue creado en Java. Esto facilita su integración y encontrar muchos plugins y complementos compatibles.

Para el caso de las pruebas automatizadas en APIs se optó por Rest Assured, pues brinda muchas facilidades para el manejo que se espera dar.

En todos los casos, las tecnologías seleccionadas se implementarán bajo las librerías de Serenity BDD. Serenity dispuso un gran nuevo de librerías basadas en las anteriormente mencionadas, pero facilitando el uso de estas a través de ciertos métodos propios, es decir, se utilizarán las librerías de Serenity BDD, que están basadas en Selenium, Appium y Rest Assured.

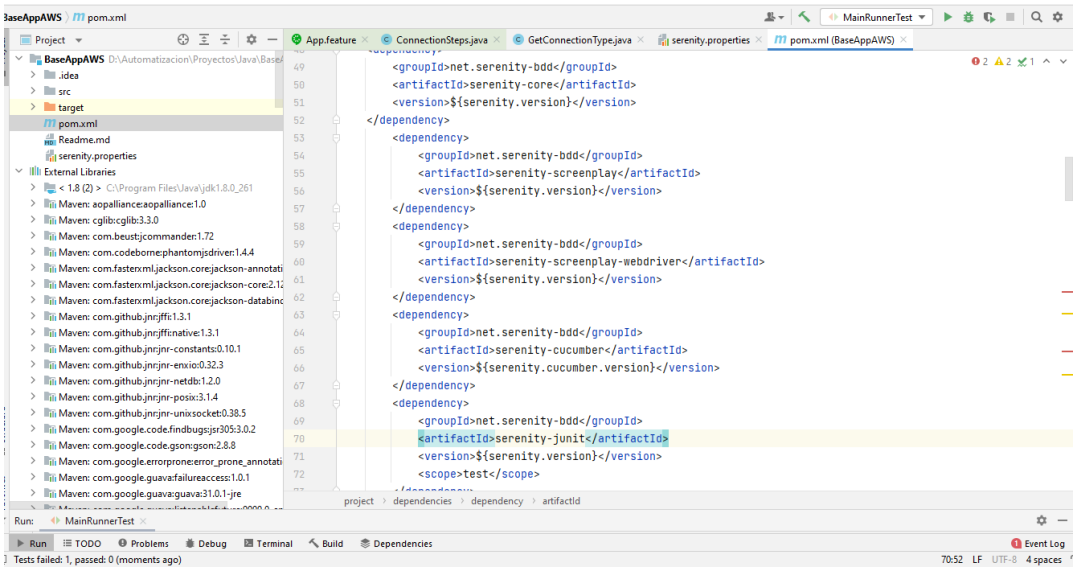

Ilustración 13. Uso de librerías de Serenity BDD en los proyectos

La razón por la que se escogió esta librería fue porque encontramos dentro de la investigación que con Serenity se pueden generar reportes bastante ilustrativos de las pruebas, facilite la captura de pantalla durante cada caso ejecutado, también facilita la organización del proyecto, ya que el código es mucho más limpio y diciente, sobretodo para personas no tan técnicas. En este caso, el código es fácilmente comprensible a través de los métodos y llamados que Serenity pone a disposición.

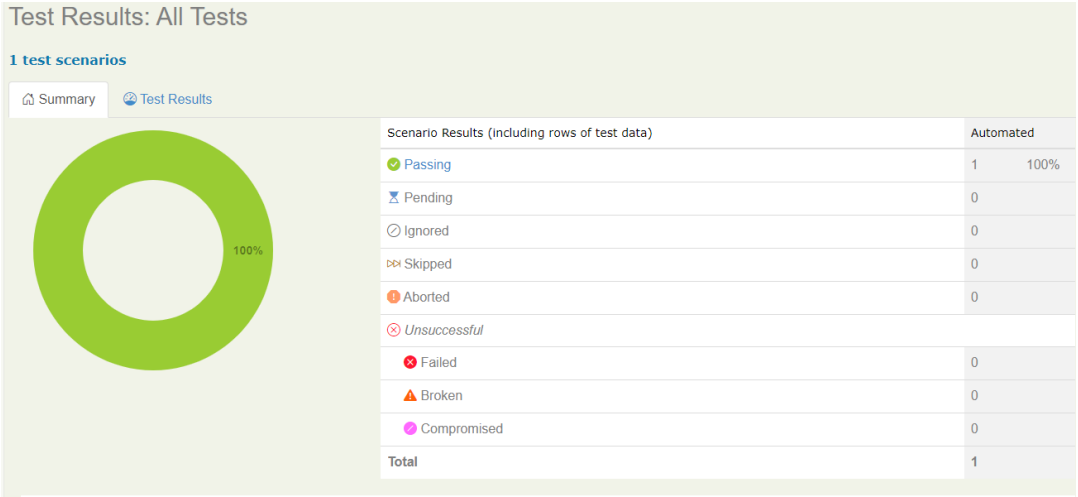

Ilustración 14. Reportes en Serenity

Para poder utilizar Serenity BDD, fue necesario realizar un curso corto sobre el uso de esta librería llamado Curso de QA Automation con Serenity – ¡De novato a experto!

Para el framework de pruebas que se debe utilizar, se escogió Junit como principal motor de ejecución de las pruebas, debido a que Serenity solo es compatible con esta herramienta y con Jbehave de la cual no se tenía suficiente conocimiento. Adicionalmente los cursos realizados mencionaban y enseñaban sobre el uso de Junit y TestNG. Este último fue descartado por la misma razón de la compatibilidad.

4. Investigación patrones de diseño de software

Posterior a la elección y la capacitación sobre las tecnologías que se pueden utilizar, se profundizó en el diseño del proyecto. Para este caso fue necesario recurrir a una capacitación corta sobre patrones de diseño y principios de buenas prácticas de desarrollo como SOLID, para comprender la relevancia y la necesidad de trabajar bajo un patrón organizado y que permitiera el fácil mantenimiento del código.

Se tenían dos opciones para realizar el trabajo, el primero de ellos es el patrón de diseño Page Object Model, que se preocupa de la separación de las pruebas o tests, de la lógica y del modelado de los objetos. Este fue el primer patrón que aprendimos a utilizar y se realizó un pequeño piloto de un proyecto para la muestra. La desventaja de este patrón de diseño es que el equipo no era muy experto en pruebas automatizadas por lo que se dificultaba un poco comprender como se implementan y como se debe organizar de la mejor forma para evitar retrabajo.

Este problema se pudo solventar al encontrar el patrón de Screenplay, que fue el que se selección finalmente, ya que a pesar de requerir más investigación y un conocimiento más avanzado en lenguajes de programación orientado a objetos, al final brinda la ventaja de que es más sencillo de organizar, y mucho más sencillo de leer para cualquier persona que no tenga un manejo tan técnico del proyecto, por ejemplo, es posible que los equipos de pruebas manuales puedan leer fragmentos del código y entender a grandes rasgos que es lo que se realiza en el o que se espera encontrar.

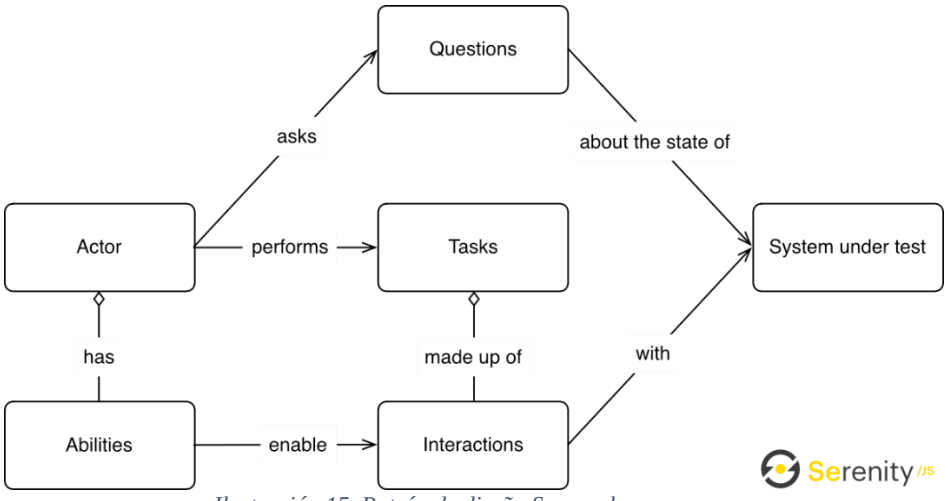

Ilustración 15. Patrón de diseño Screenplay

Con el patrón de POM era un poco más complejo y requería involucrar más el manejo directo en el código para algunas tareas, con Screenplay las configuraciones se pueden llegar a parametrizar de manera más cómoda.

5. Elección de patrones de diseño

Para seleccionar el patrón de diseño, se tuvo en cuenta la investigación principal enfocada en este aspecto y algunos de los puntos que se tuvieron en cuenta fueron:

Mantenimiento. Los proyectos más antiguos de la empresa presentan una situación de mantenimiento complejo, esto debido a que los desarrollos de algunas funcionalidades no fueron analizados desde el punto de vista de arquitectura y diseño de software, es por esto que prestamos especial atención a que el proyecto de pruebas automatizado no tenga estos inconvenientes también, por lo que uno de los factores más relevantes para nosotros fue la mantenibilidad del proyecto. A largo plazo se espera que realizar ajustes y cambios en los scripts no sea demasiado costoso y que no requiera de un conocimiento muy especializado.

Organización. Es indispensable que, para llegar a tener una buena mantenibilidad dentro del proyecto, este se encuentre bien organizado y partiendo de una estructura definida por un patrón de diseño esto es posible. La paquetería debería ser diciente, y debe contener siempre los archivos necesarios para utilizar adecuadamente las clases y métodos dentro del proyecto.

Curva de aprendizaje. El ideal de la empresa era poder empezar a automatizar tan pronto como fuera posible, pero esto requiere tiempo de

aprendizaje, es por esto que se escogió un patrón de diseño que fuera fácil de comprender, y fácil de implementar a corto plazo.

Necesidad del negocio. Para el caso de las pruebas automatizadas vimos la necesidad de contar con un enfoque orientado al negocio, puesto que muchas de las pruebas que se realizan manualmente son socializada y compartidas con el cliente. En este sentido es necesario contar con las herramientas que nos permitan hacer estas entregas de manera casi transparente y que el cliente conozca con muy poco detalle técnico que es lo que se está contemplando en las pruebas automatizadas.

Escalabilidad. Los proyectos más grandes de la empresa van expandiéndose horizontal y verticalmente, es por esto que las pruebas automatizadas deben poder crecer en conjunto con el proyecto, y bajo un patrón de diseño bien estructurado es mucho más sencillo conseguir este objetivo.

Buenas prácticas de desarrollo. Es importante que se manejen buenas prácticas de desarrollo en todos los proyectos de la compañía, ya que una de las principales ofertas en el mercado es la calidad del trabajo entregado. Como área de calidad debemos velar por qué nuestros desarrollos estén acordes con las buenas prácticas que se han diseñado y que nos garantizan una ejecución del código más eficiente y menos propensa a sufrir errores por mal uso de las herramientas.

Documentación y soporte. Dentro de los factores para determinar el patrón de diseño, se revisó la documentación y al soporte en general de la comunidad para poder resolver los inconvenientes y errores que se puedan presentar.

Bajo estas necesidades encontramos que tanto el patrón de diseño POM como el patrón Screenplay cumplen con brindar un buen soporte, una curva de aprendizaje relativamente sencilla (mayor en Screenplay), pero en cuanto al manejo de los reportes, el mantenimiento, la necesidad del negocio y la promoción de las buenas prácticas de desarrollo, decidimos optar por Screenplay que ofrece estas características sin problema, en cuanto al patrón POM, es menos utilizado en conjunto con las buenas prácticas de desarrollo.

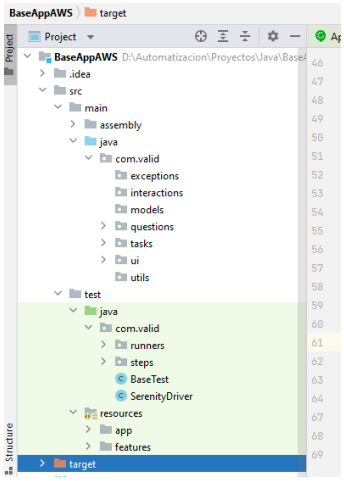

Ilustración 16. Estructura principal Screenplay

La estructura contempla distintos paquetes y clases para funcionar:

Features: Contiene los escenarios planteados con la herramienta Cucumber en lenguaje Gherkin, que recibe el nombre de “Feature”, estos son casos de prueba escritos en un lenguaje menos técnico.

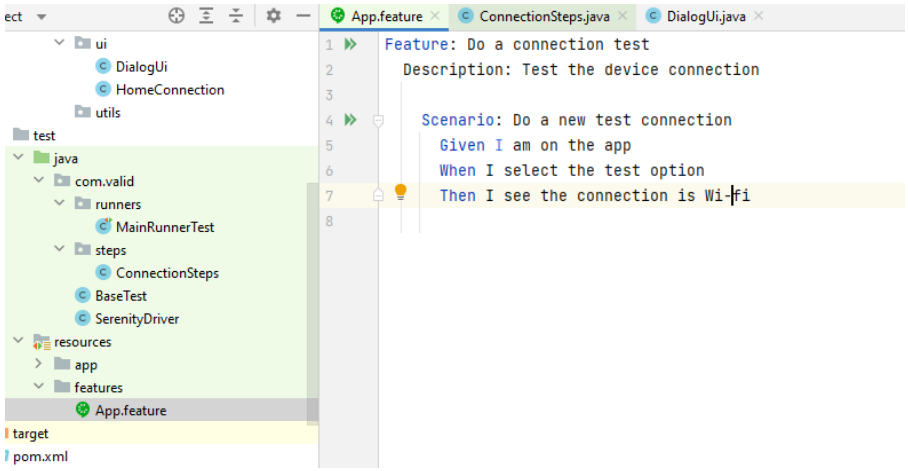

Ilustración 17. Ejemplo de un feature

Steps: Este paquete contiene las clases de prueba donde se describen los pasos necesarios para llevar a cabo la automatización de los escenarios descritos en los features.

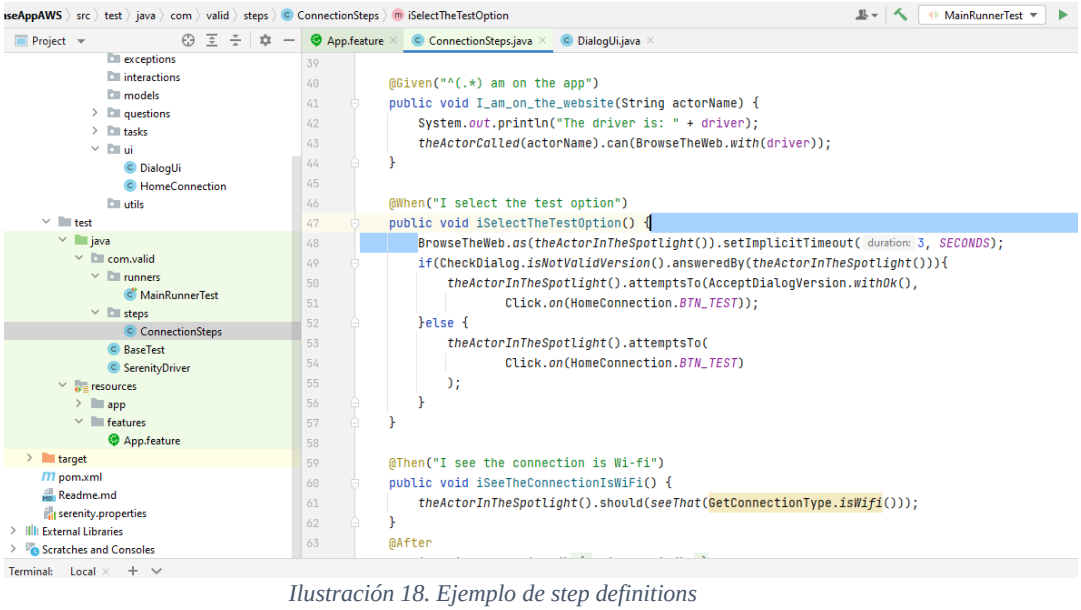

Ilustración 18. Ejemplo de step definitions

Runners: contiene las clases que enlazan los features con los steps para poder ejecutar los pasos dados en lenguaje de negocio con los pasos en código.

Exceptions: el objetivo de este paquete es contener las clases encargadas de mapear errores conocidos y detallarlos de manera más natural en los reportes.

Interactions: contiene clases diseñadas para realizar interacciones comunes con la interfaz del usuario, por ejemplo, hacer scroll en la vista.

Models: son clases que modelan objetos utilizados durante los flujos y escenarios. Por ejemplo, modelo de una nota o el modelo de una conexión.

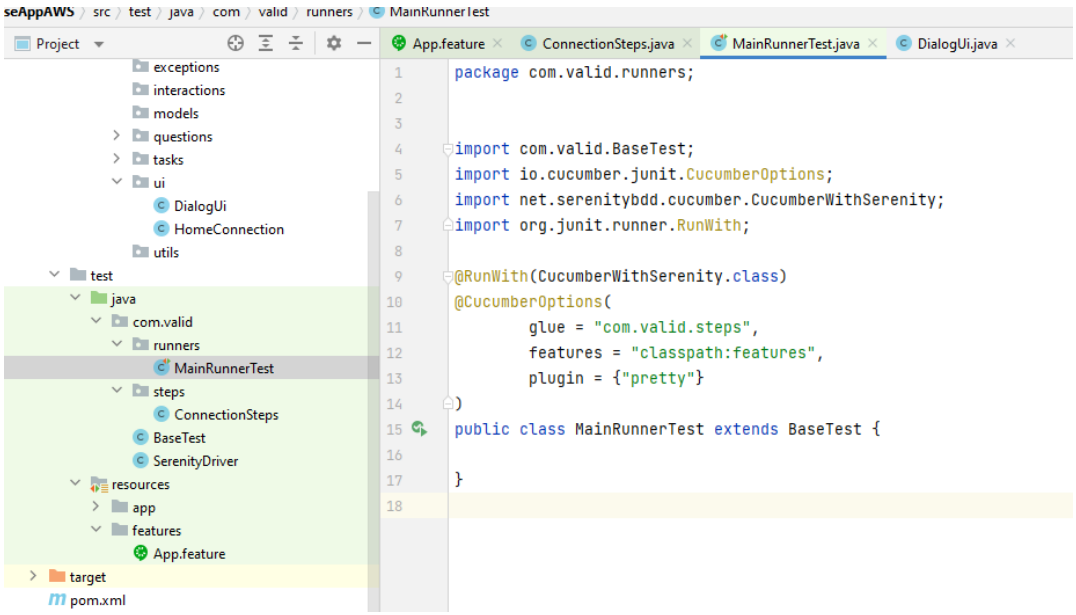

Ilustración 19. Ejemplo de la clase Runner

Tasks: este paquete contiene las clases creadas para ejecutar pasos repetitivos reutilizables en los escenarios.

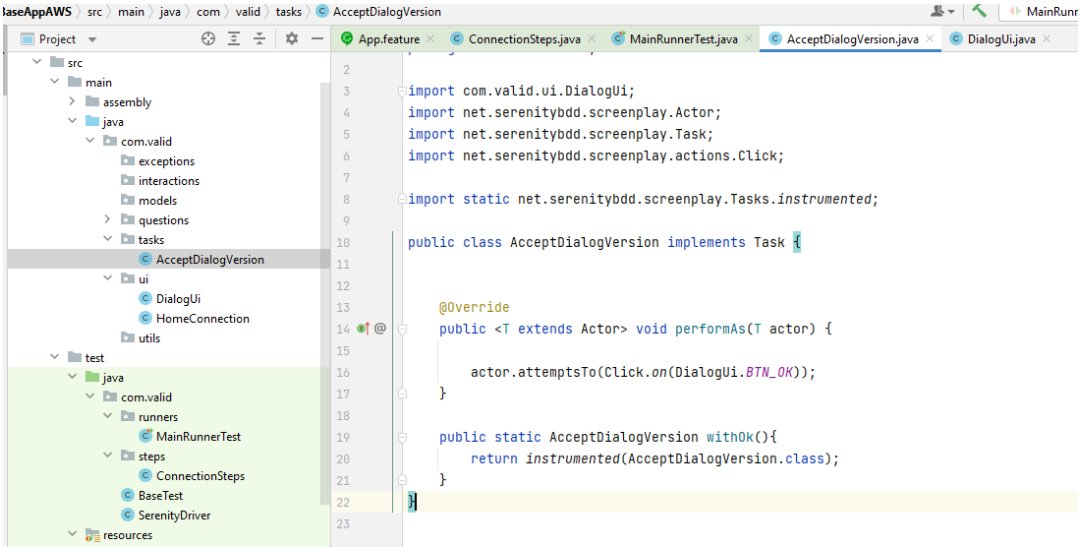

Ilustración 20. Ejemplo de una clase Task

Questions: Dentro de este paquete se agregan clases que representan preguntas por el estado de los elementos de prueba.

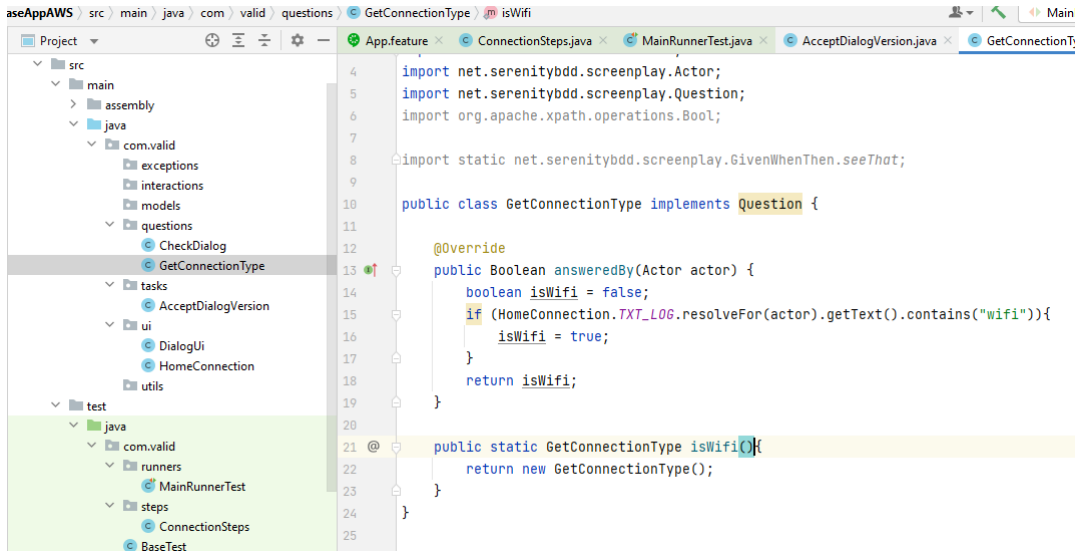

Ilustración 21. Ejemplo de una Question

UI: Dentro del paquete UI se alojan las clases encargadas de modelar los elementos fijos de la cada vista, facilitando la ubicación y el llamado de estos.

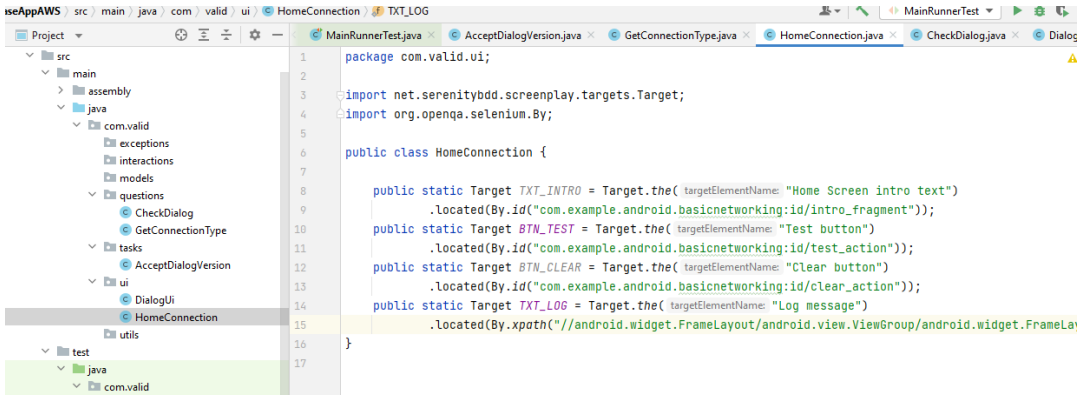

Ilustración 22. Ejemplo de una clase UI

Utils: se agrupan las clases utilizadas con funciones variadas que son de utilidad general en el proyecto.

6. Creación de proyecto base para automatización en móviles

Una vez escogida la tecnología que se utilizaría para las pruebas en móviles, se comenzó con la creación de un proyecto base. La idea de esta base es permitirles a los miembros del equipo de automatización contar con un proyecto con la estructura, las librerías y características necesarias para iniciar con el desarrollo particular del proyecto. El proyecto base se creó a partir del patrón de diseño Screenplay, con esto logramos dejar un esquema de paquetes bien organizado y documentado.

Para hacer uso del proyecto base se debía dejar alojado en el repositorio de código utilizado por la empresa, en este caso Bitbucket. Aquí se creó un proyecto de automatización con ayuda del administrador del sitio, dentro de este proyecto fueron creados 3 repositorios: Base App Automation, Base Web Automation y Base API Automation.

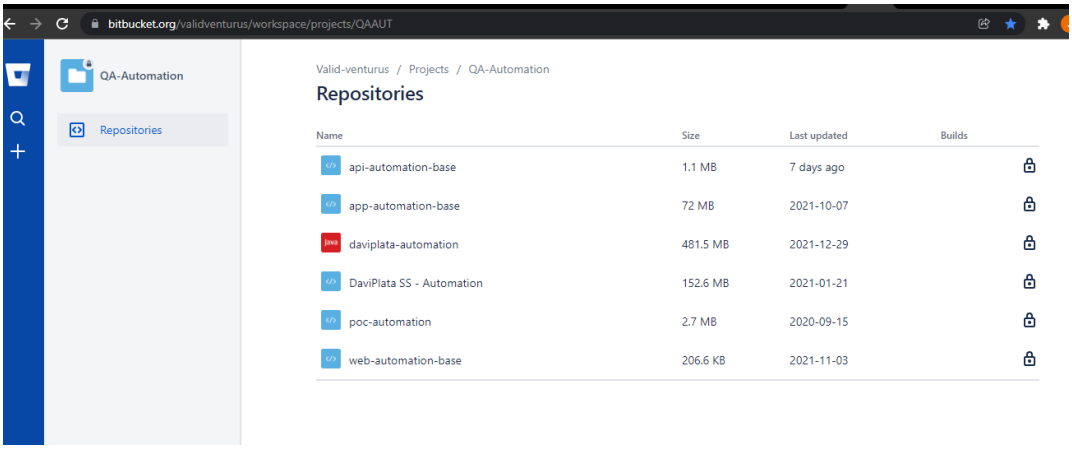

Ilustración 23. Repositorios base alojados en Bitbucket

Dentro del proyecto Base App Automation se subió el proyecto base que se planteó en principio. Este contiene un proyecto basado en Maven, el cual ya trae el listado de todas las librerías básicas necesarias para ejecutar el proyecto dentro del archivo POM.xml.

De esta forma es fácil comprender la utilidad de cada una de las clases en el proyecto, así como la estructura que debe seguir una prueba, el lugar en donde deben alojarse y como ejecutar las pruebas.

Para realizar la ejecución de las pruebas automatizadas con Appium en este caso, es necesario contar con un dispositivo físico, o un emulador, tener instalado Node.js, pues actúa como servidor para levantar la interfaz de trabajo de Appium. Luego de esto se procede a configurar el dispositivo dentro de las propiedades del proyecto. Una vez completado esto se ejecuta la prueba y la prueba base se puede observar en el equipo.

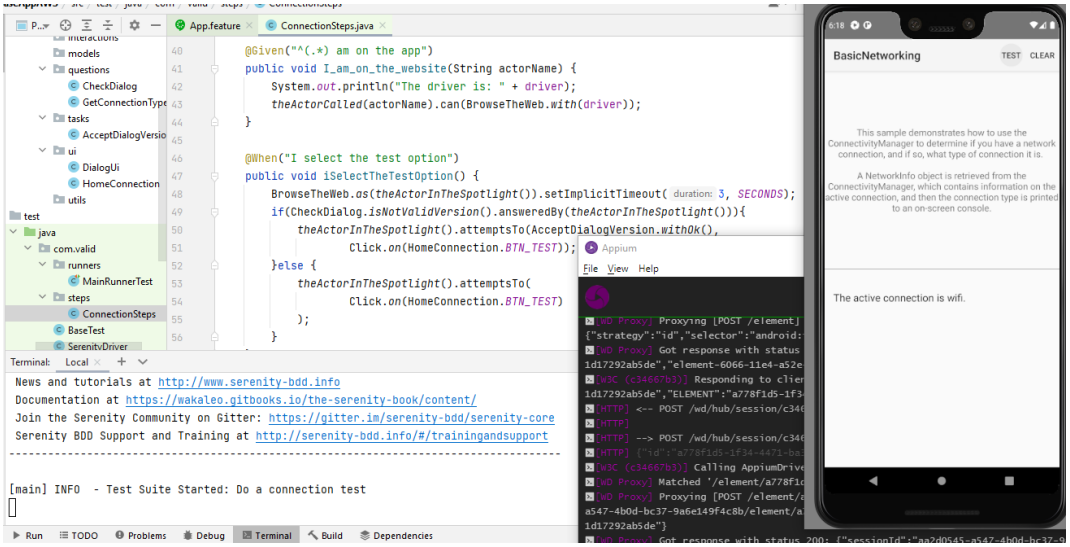

Ilustración 24. Proyecto base Mobile en ejecución

A partir de esta ejecución es más fácil poder iniciar con las pruebas particulares que requiere cada proyecto.

7. Creación de proyecto base para automatización en web

Al igual que la tarea para la creación del proyecto base para mobile, fue necesario crear un proyecto base del cual partiría toda el diseño y la creación de los proyectos para web. El proceso fue muy similar, primero al haber escogido Selenium como tecnología principal, se utilizaron las librerías de Serenity para dejar un esquema idéntico al proyecto base Mobile. La única diferencia en este caso consiste en la utilización del WebDriver de Selenium en lugar de Appium.

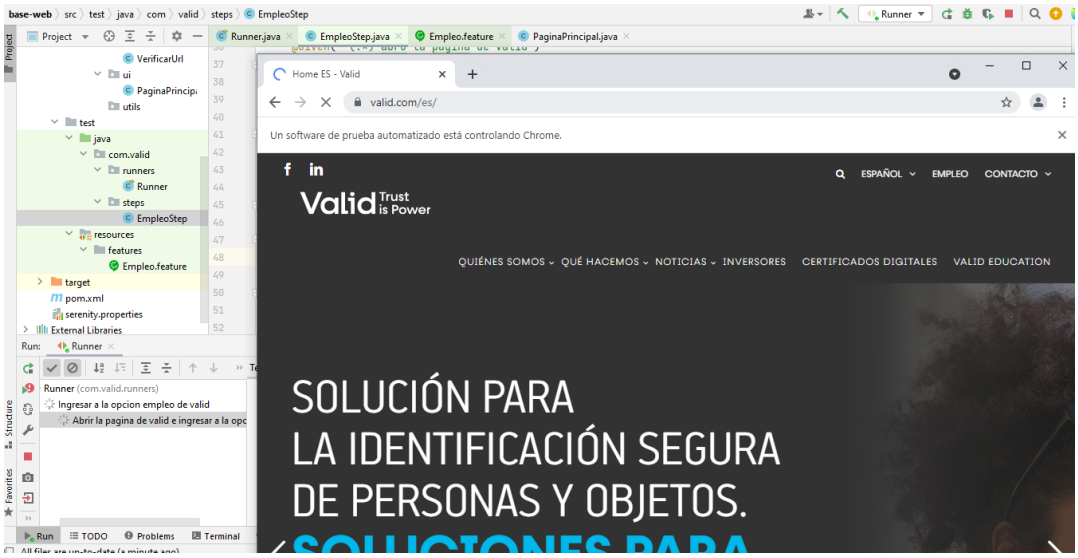

Ilustración 25. Ejecución de proyecto base Web

El proyecto base para web se alojó en Bitbucket, y está disponible para que las personas del equipo puedan descargarlo y a partir de el puedan iniciar con la codificación para su propio proyecto.

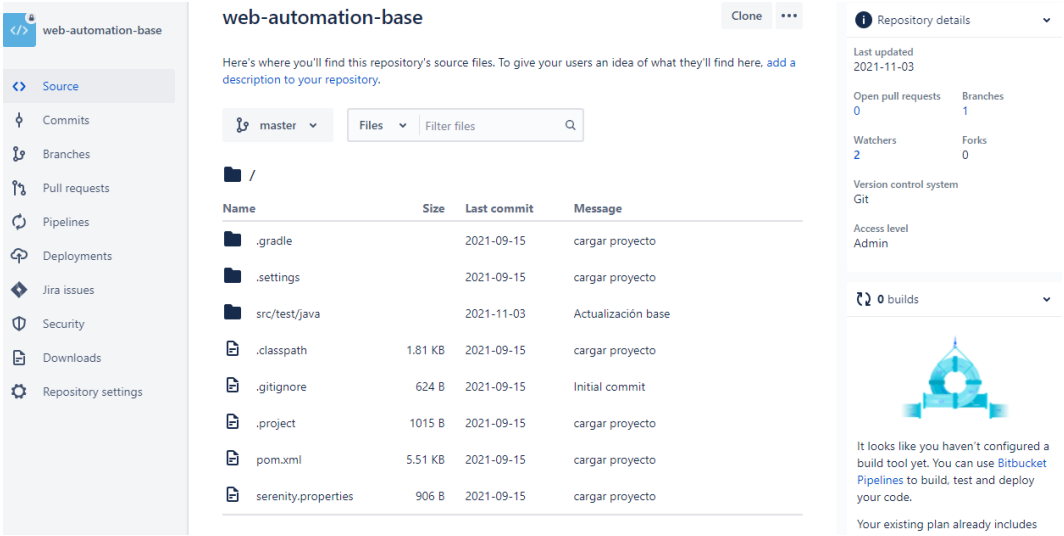

Ilustración 26. Proyecto base web alojado en Bitbucket

8. Creación de proyecto base para automatización de API's

Para el proyecto base de APIs se cuenta con una estructura basada puramente en el uso de Cucumber con Rest Assured y JUnit. En este caso, no hemos aplicado aún el patrón de diseño de Screenplay y no se están utilizando las librerías de Serenity, debido a que no se ha actualizado la base y el equipo no ha recibido la capacitación necesaria para su uso. Esto es un tema que está pendiente de ajustar y se espera concretar el próximo año.

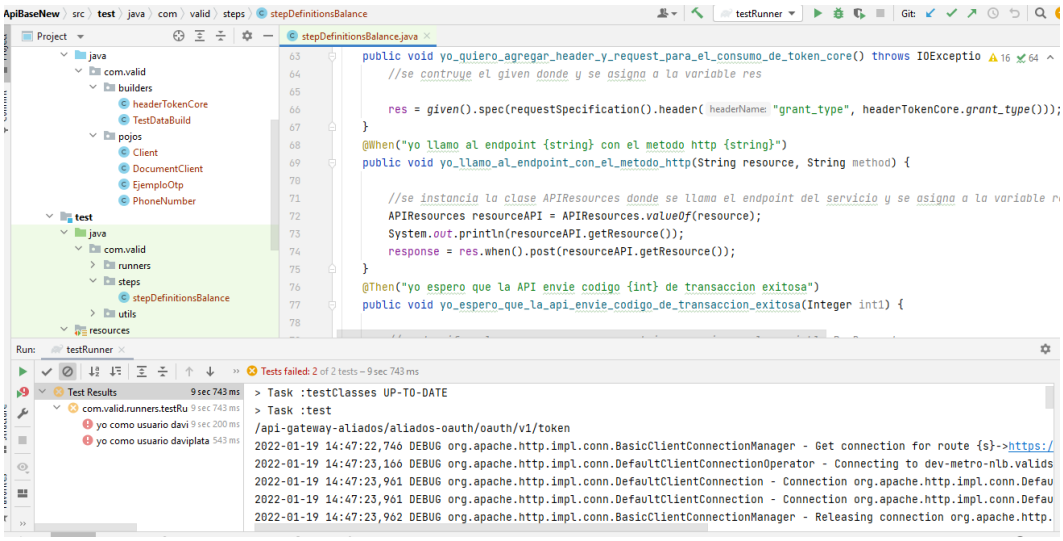

Ilustración 27. Proyecto base Apis en ejecución

En el caso del proyecto base de APIs respeta la organización y la separación de las responsabilidades de las clases, se procuró mantener buenas prácticas de desarrollo y un esquema sencillo de comprender basado en la distinción de los modelos de las peticiones y respuestas de los tests propiamente dichos.

Algunos paquetes que se diferencian de los proyectos de móviles y web son: ***Builders:*** Son clases encargadas de construir un request a partir de un modelo pojo dado, y que posteriormente será utilizado durante el consumo del servicio.

Pojos: Contiene las clases que modelan los request y response correspondientes a servicios particulares. A partir de estos modelos es posible construir un request complejo o mapear una respuesta en formato JSON.

9. Creación de diagrama de proceso de automatización.

Una vez que se contó con la base para iniciar el trabajo en los distintos proyectos, también fue necesaria la creación del flujo completo del proceso de automatización, ya que es necesario para la organización interna de la compañía y nos permite mostrar el trabajo que se debe realizar tanto al equipo de automatización como a los distintos participantes de cada proyecto en el que se espere implementar la automatización de pruebas funcionales.

Para la creación del diagrama de proceso se utilizó la herramienta Miro, ya que brinda la posibilidad de crear diagramas UML y muchos más en línea de manera ágil.

Este diagrama fue compartido y socializado con todos los miembros del equipo de automatización.

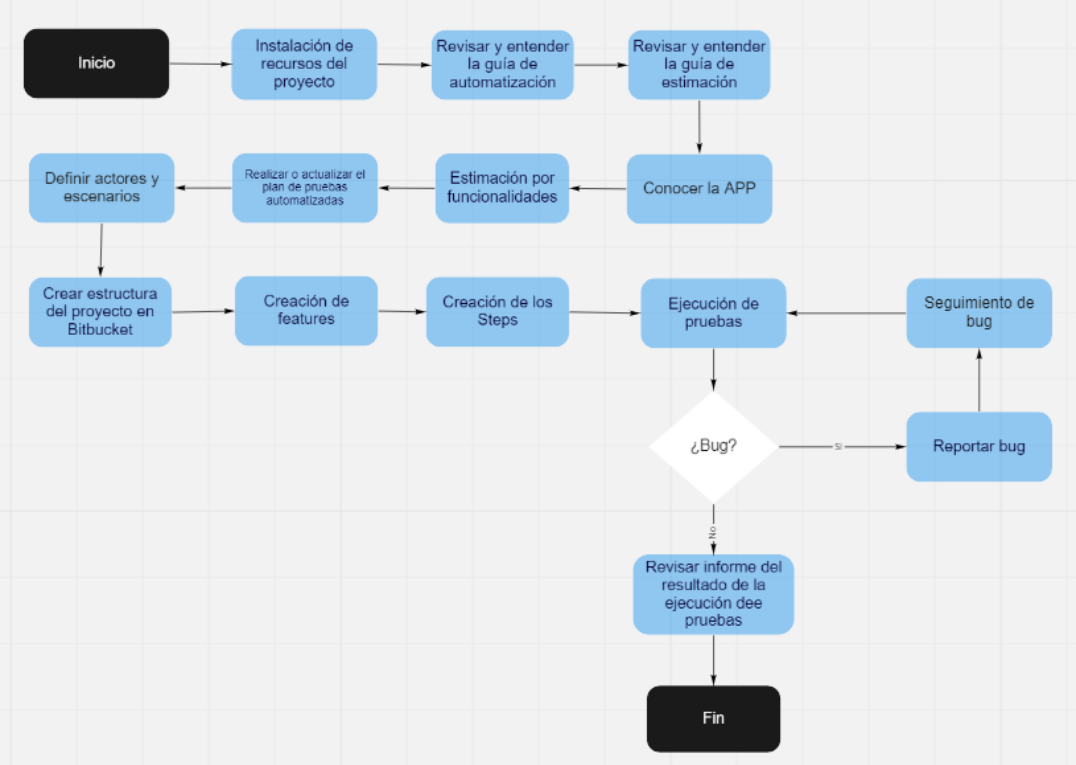

Ilustración 28. Diagrama de proceso de automatización

10. Creación de lineamientos de desarrollo para la automatización de pruebas

A partir de las definiciones que se realizaron se creó un documento en el cual se condensan todos lo lineamientos que se requieren para iniciar con las pruebas automatizadas. Estos lineamientos son la base teórica bajo la cual cada automatizador antiguo o nuevo debe basarse.

Dentro del documento se describen con claridad cuáles herramientas serán utilizadas y el paso a paso que se debe seguir para la instalación de cada una de ellas. Igualmente se describe cual es el patrón de diseño que se debe utilizar, donde se encuentran los repositorios base, recomendaciones para la redacción de los escenarios de prueba, recomendaciones generales para la

escritura de los métodos y clases, y el manejo adecuado que se debe hacer de las ramas y el versionamiento en Bitbucket.

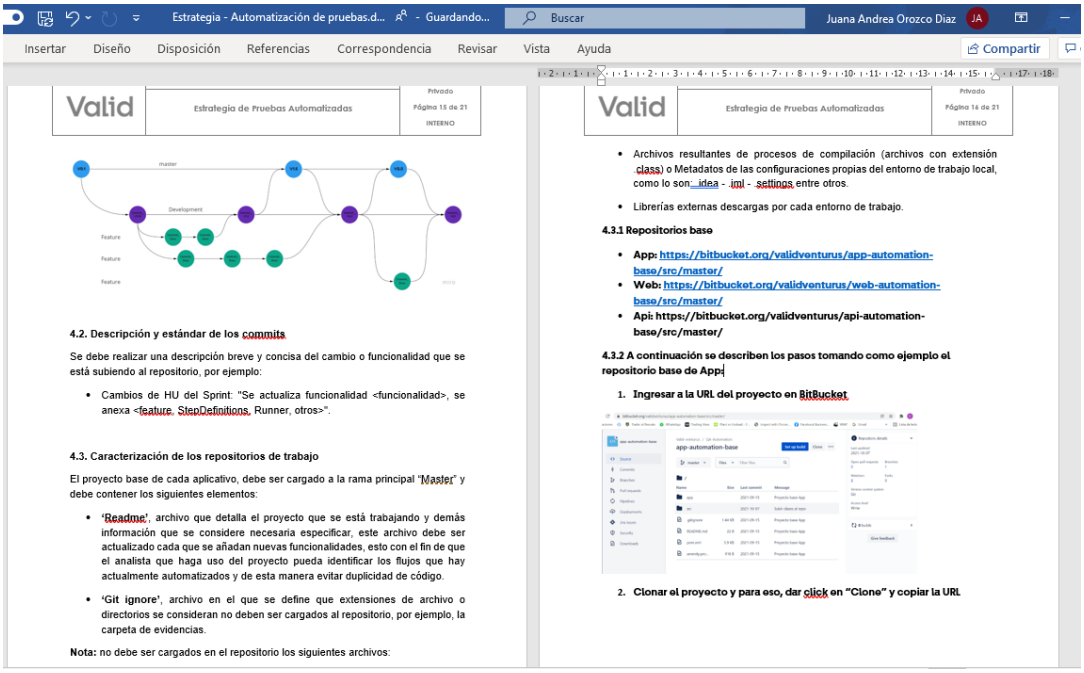

Ilustración 29. Documento de Estrategia de Automatización

11. Creación de formatos para la documentación de pruebas automatizadas

Otra de las actividades que se hizo necesaria fue la de crear algunos formatos que serían necesarios para los automatizadores, el primero de ellos es un formato de Estimación de pruebas automatizadas. Este documento da algunas pautas para lograr una estimación en tiempo de cuanto esfuerzo se requiere para algunas tareas y permite facilitar la comprensión de que tipos de pruebas se pueden o no automatizar. El resultado será un documento en el que se basen las tareas del automatizador durante los sprints o fases del proyecto al que se asigne. Estos documentos se alojan en el repositorio documental de la empresa en SharePoint.

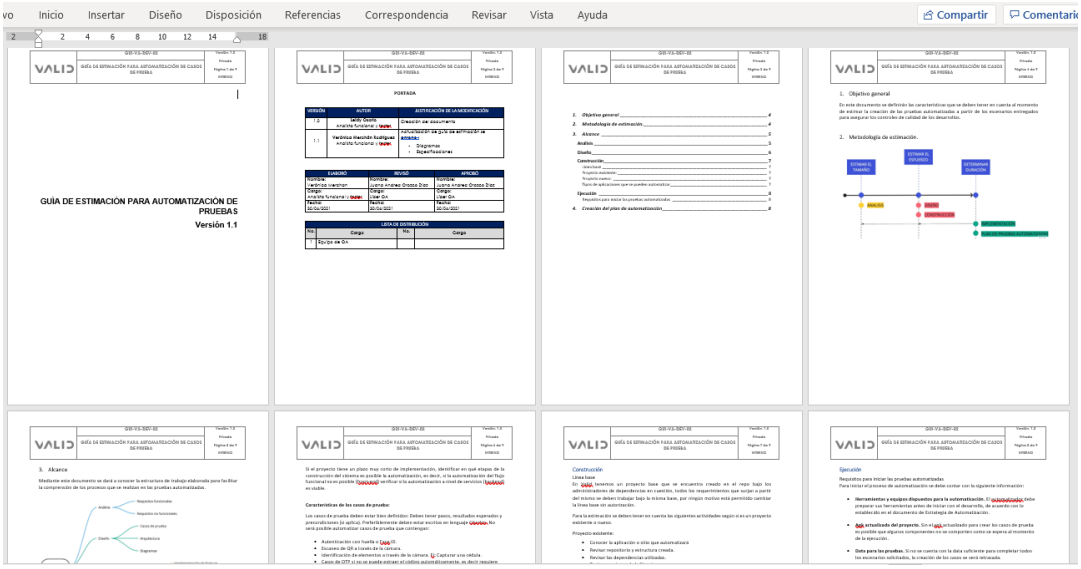

Ilustración 30. Documento Guía de estimación para automatización

12. Implementación de pruebas funcionales automatizadas en proyectos críticos

Proyecto DaviPlata (Mobile)

Al adquirir conocimiento sobre la automatización de pruebas se comenzó a trabajar en algunos proyectos piloto para lograr esta implementación, uno de los primeros proyectos escogidos fue DaviPlata. Este es el proyecto más grande que tiene la empresa actualmente y es uno de los que mayor tiempo de pruebas consume. La automatización para este proyecto inició siguiendo una línea básica de pruebas críticas, se crearon distintos escenarios que pudieran abarcar las funcionalidades básicas de la aplicación, otorgando un resultado exitoso o de error en el caso de encontrar algún problema.

Este proyecto inició con un patrón de diseño basado en el modelo POM, y posteriormente realizamos una migración a Screenplay tratando de seguir algunos de los lineamientos planteados.

El resultado fue un proyecto mucho más fácil de comprender para cada una de las personas que ha trabajado en el, así como el poder mantener los tests automatizados ya creados. En estos momentos el proyecto se encuentra en desarrollos y se siguen agregando escenarios.

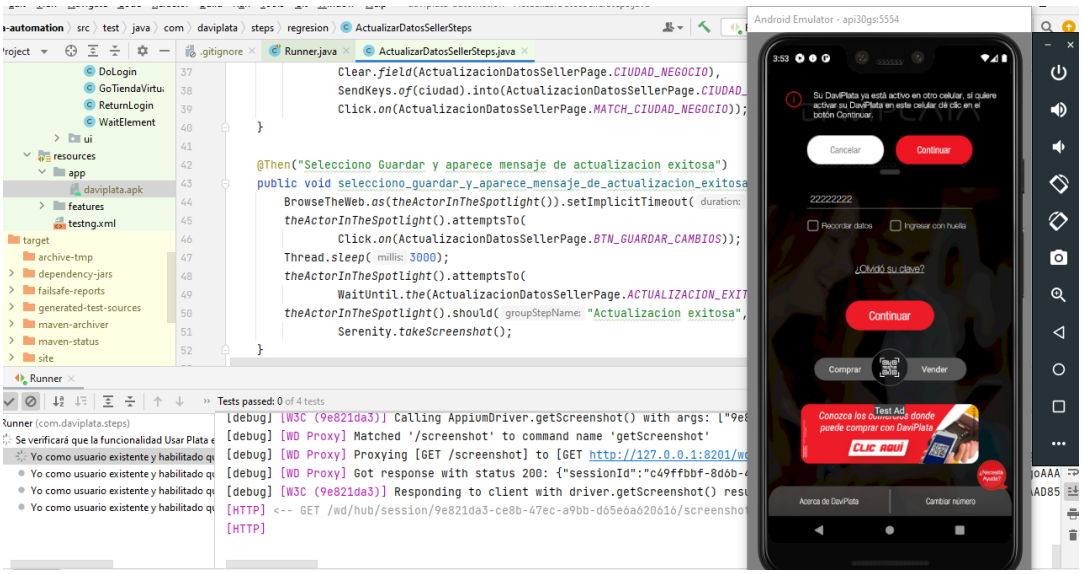

Ilustración 31. Ejecución de pruebas automatizadas DaviPlata

Se creó un repositorio en Bitbucket donde se aloja el proyecto automatizado, las pruebas se pueden ejecutar en modo local en las máquinas de las personas que puedan descargar este repositorio y se generan reportes al finalizar la ejecución de cada uno de los tests.

Este proyecto en particular sigue una metodología aún bastante tradicional, por lo que el seguimiento diario de las tareas se realiza desde el área de calidad, y no tanto desde el modelo de trabajo de Scrum.

Proyecto Super App Cívica (Mobile)

El proyecto de la Super App de Cívica partió de una base similar a DaviPlata, pero con tecnologías renovadas y una arquitectura mucho más estable. Esto hizo que se pudiera desarrollar un proyecto muy grande en un tiempo relativamente corto, por lo mismo fue necesario comenzar a trabajar en las pruebas funcionales automatizadas.

El proceso fue muy similar al que se siguió en DaviPlata, pero a diferencia del piloto que se había realizado, en este caso se comenzó a trabajar directamente siguiendo el patrón de diseño de Screenplay, sin pasar por POM. Este fue el primer proyecto en ser creado con el nuevo patrón de diseño seleccionado.

Se trazó una ruta crítica cubriendo las pruebas más importantes necesarias para la ejecución de las pruebas de aceptación del usuario. Dentro del proyecto de Super App Cívica se sigue una metología ágil, más exactamente Scrum, es por esto por lo que se realizan entregas periódicas de los desarrollos estabilizados, esto permite que el automatizador también trabaje en la dinámica de esta metodología. Se realiza una ruta crítica de las funcionalidades que se están desarrollando.

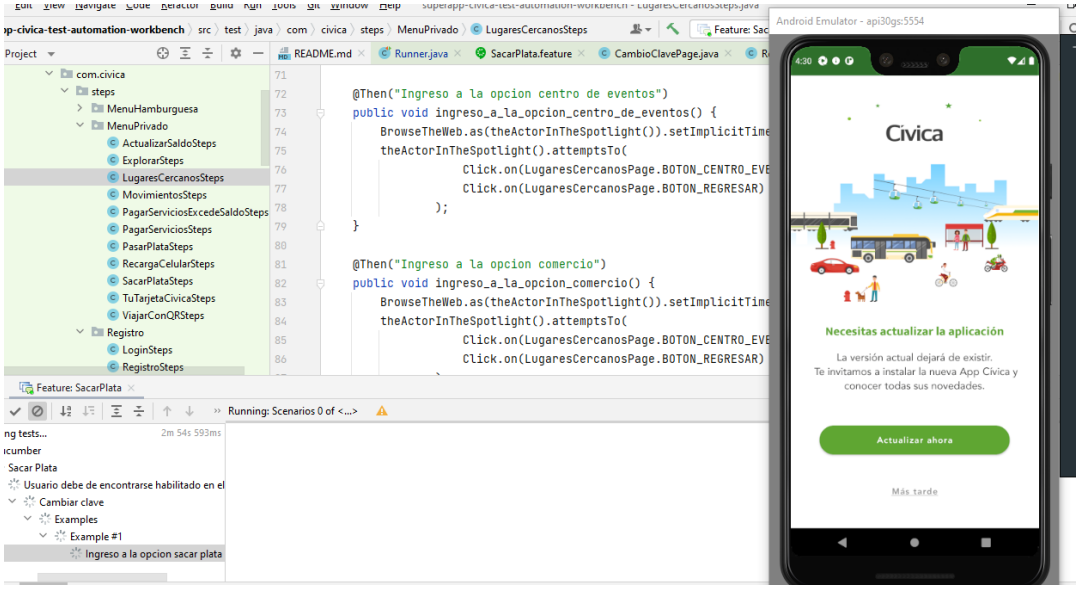

Ilustración 32. Ejecución del proyecto Super App Cívica

Proyecto DaviPlata Core Digital (API)

Para el caso del Core Digital se estimó un proyecto muy grande a nivel de servicios el cual está orientado a disponibilizar los consumos para otras aplicaciones. En esta ocasión se partió de la base que se tiene actualmente para el desarrollo de la automatización de servicios, enfocado en las pruebas funcionales.

El patrón de diseño en este caso no corresponde a Screenplay, ya que en estos momentos no se cuenta con el conocimiento necesario para trasladarlo a los proyectos con APIs, pero es una de las tareas pendientes en este sentido. Se utilizó un framework basado en Cucumber y JUnit para el diseño de los escenarios de prueba y la ejecución.

Inicialmente se trabajó en el desarrollo de una ruta crítica, abarcando todos los servicios disponibles y realizando validaciones sencillas sobre cada uno

de ellos, posteriormente se fueron adicionando casos de prueba para complementar las ejecuciones y entregar un proyecto mucho más robusto.

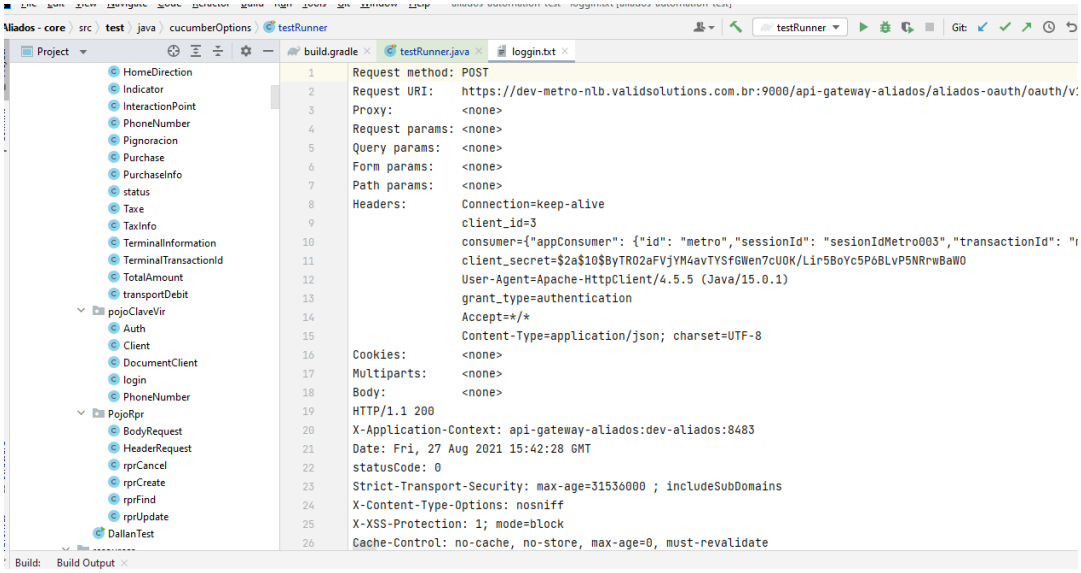

Ilustración 33. Ejecución de pruebas Core Digital

13. Investigación sobre ejecución de pruebas remotas en granjas de dispositivos

A raíz del inicio de la automatización surgieron algunas ideas sobre complementar las pruebas sobre distintos equipos móviles, ya que actualmente es necesario contar con un dispositivo físico para realizar la ejecución y esto limita el uso de los scripts de los proyectos mobile y dependen de que se cuente con la persona y el equipo disponibles.

En este caso se nos recomendó realizar una investigación a fondo sobre la ejecución de pruebas automatizadas en granjas de dispositivos. Estas granjas se encargan de disponibilizar equipos físicos de manera remota a través de una interfaz propia de cada compañía, la cual requiere un pago mensual para acceder al servicio.

Durante la investigación se evaluaron 3 granjas que son de las más utilizadas actualmente en el mercado:

Amazon Device Farm: La empresa cuenta con varios productos en la nube de Amazon, por lo que el servicio de Device Farm era una de las principales opciones para adquirir. En este momento se contaba con un proyecto de prueba básico que ejecutó el equipo de DevOps de la compañía, pero no estaba alineado con las nuevas características que se definieron para la automatización.

En este caso fue necesario realizar una nueva configuración para ejecutar las pruebas, la cual llevó bastante tiempo debido a la poca documentación que existe sobre la integración de Serenity y el patrón de diseño Screenplay con la granja de Amazon. Finalmente fue posible ejecutar un proyecto muy básico con las configuraciones necesarias.

Esta granja se basa en la integración de un plugin de Amazon dentro del proyecto tipo Maven, en el POM.xml, este plugin se encarga de realizar un empaquetamiento del proyecto separando las dependencias, los tests y otros componentes bajo una estructura reconocida por Amazon. Este proceso se ejecuta de manera manual en estos momentos y es necesario subir los archivos generados en conjunto con el aplicativo en formato APK (para Android) o IPA (IOS) a la plataforma de AWS, donde se escogen los dispositivos a utilizar y se genera un reporte de la ejecución con evidencias.

Amazon ofrece todos sus dispositivos limitados por ranuras que van en función del sistema operativo del equipo, esto significa que, al adquirir una ranura para Android, es posible utilizar todos los dispositivos Android disponibles (uno a la vez), pero para utilizar los equipos con iOS, es necesario adquirir otra ranura. Cada una de estas ranuras cuesta $250 usd.

Browserstack: En vista de que los primeros intentos con la granja de dispositivos de Amazon no fueron satisfactorios, fue necesario recurrir a otras alternativas para la integración de las pruebas. Una de ellas fue Browserstack, quienes cuentan con más años en el mercado ofreciendo una amplia variedad de dispositivos con distintos propósitos. La integración en este caso fue más sencilla que con AWS debido a que contaban con mucha más documentación con muchos frameworks soportados y con distintos proyectos de muestra que facilitaban el entendimiento y la configuración del servicio. A diferencia de Amazon no es necesario realizar un empaquetado del proyecto para ejecutarlo en los servidores, se realiza una configuración en el proyecto donde se ejecuta una conexión remota con los servicios de Browserstack, lo cual permite realizar la ejecución en el equipo local pero el dispositivo utilizado es el que se encuentra en la nube. Se genera un informe de la ejecución y las evidencias.

Para adquirir este servicio se nos sugirió contar con un plan de 1 ejecución en simultáneo, pero con posibilidad de utilizar cualquier dispositivo, Android o iOS por $200 usd.

SauceLabs: Este servicio es muy similar a Browserstack y funciona de la misma forma, no se realizó una configuración para probarlo ya que representa los mismos beneficios que ofrece Browserstack por un costo un poco más elevado y por un número de dispositivos ligeramente inferior. Se evaluó realizar las pruebas si no era posible crear la conexión en ninguna de las anteriores.

Al contar con varios servicios de Amazon integrados en los procesos de la compañía la elección finalmente fue contratar AWS Device Farm, puesto que se cuenta con un plan de integración continua y despliegue continúo

avanzado por parte del equipo de DevOps y sería mucho más sencillo de integrar en el flujo.

14. Creación de proyecto base para pruebas automatizadas remotas

Una vez seleccionada la granja que se utilizaría, se comenzó a trabajar en la estructura base para móviles integrando las configuraciones necesarias para realizar la ejecución de manera remota. Es por esto que se retomó la base alojada en Bitbucket inicialmente y se integraron algunos cambios necesarios:

Se añadió una carpeta main dentro del proyecto, la cual contiene un script que permite empaquetar las dependencias del proyecto y los tests dentro de un único archivo .zip.

Se agregó dentro del POM.xml un plugin propio de Amazon para separar y compilar la estructura requerida por ellos y alojarlos en una ruta específica.

Para realizar el empaquetamiento es necesario ejecutar el comando mvn package, con esto y los plugins integrados se guardan todos los archivos necesarios (clases y dependencias) dentro de un archivo .zip que posteriormente es requerido en la plataforma.

Para la ejecución se debe ingresar a la plataforma de AWS y en el servicio de Device Farm se genera un nuevo grupo de prueba. Aquí siguiendo los pasos se debe alojar la aplicación móvil y el archivo con los scripts de prueba generados anteriormente. Con esto se configura el dispositivo bajo el cual van a correr las pruebas y se ejecuta. Se debe esperar a que finalice la ejecución para comprobar el video de evidencia. Se puede observar un log en vivo donde se registran los eventos durante la ejecución.

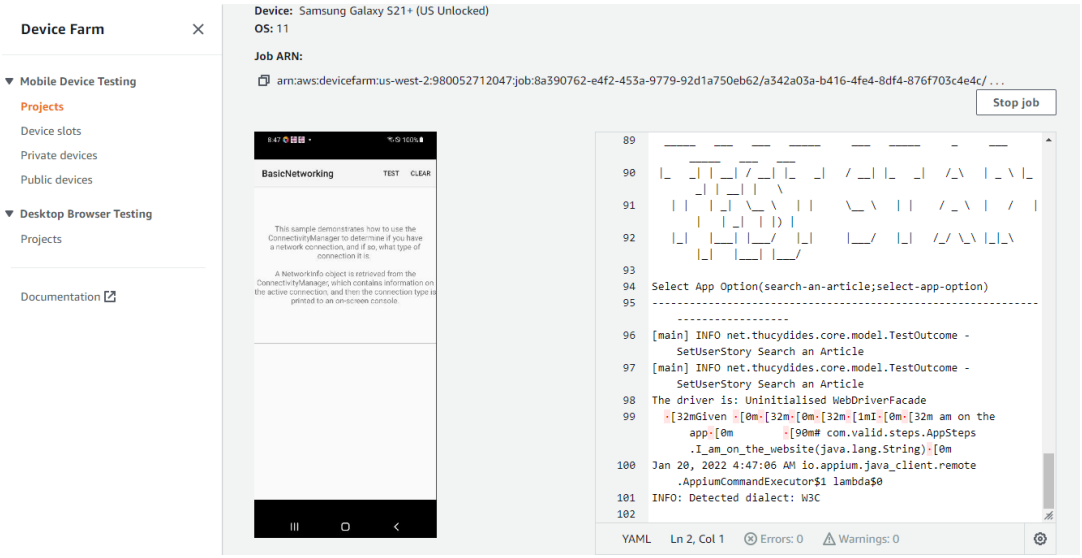

Ilustración 34. Ejecución de pruebas automatizadas en AWS Device Farm

15. Implementación de flujo de integración continua

Para la integración continua fue necesario establecer unas pautas del manejo de los repositorios y la compilación de los proyectos en la herramienta Jenkins. Este flujo de integración continua inicia con la creación de los repositorios en Bitbucket, desde aquí y haciendo uso de Git para alojar el código dentro del repositorio, es necesario que el equipo conozca sobre el flujo de versionamiento en la nube. Se diseñó un flujo básico para ejecutar los commits y el nombramiento de las versiones de los proyectos automatizados, así como el manejo de ramas y features. También se estableció la premisa de que cada commit debe ser realizado al finalizar el día dentro de los parámetros establecidos, no se deben quedar cambios por fuera del repositorio ni acumular demasiados ya que puede generar conflictos y hacer más lenta la integración del código.

Con el repositorio construido y en constante actualización, se realizó una configuración dentro de la herramienta Jenkins, la cual permite compilar de

manera automática el código fuente alojado en los repositorios. Se estableció la dirección y se indicaron los comandos para realizar la ejecución. Esta integración solo está creada para proyectos de APIS debido a que para los scripts en móviles y web es necesario todavía realizar ajustes más avanzados, debido a la necesidad de contar con una interfaz, bien sea el navegador o el dispositivo móvil.

16. Creación de presentación general para la compañía

Una vez finalizado todo el proceso de automatización y establecer los lineamientos fue necesario crear una presentación para la empresa en la cual se explique a alto nivel todo el proceso de automatización y de como quedará integrado dentro del ciclo de desarrollo de software de la empresa.

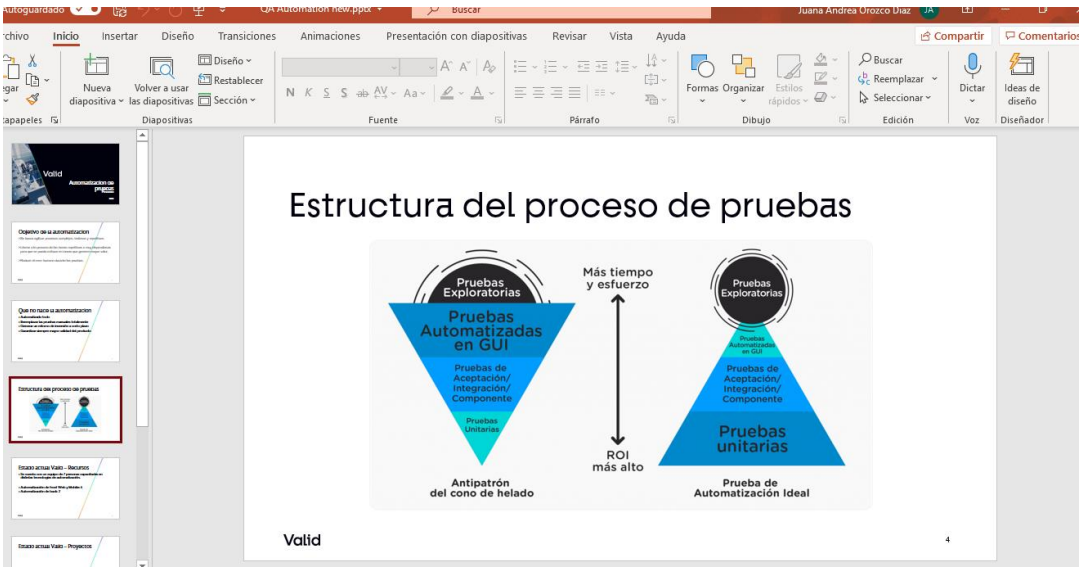

Ilustración 35. Presentación proceso de automatización

8. CRONOGRAMA:

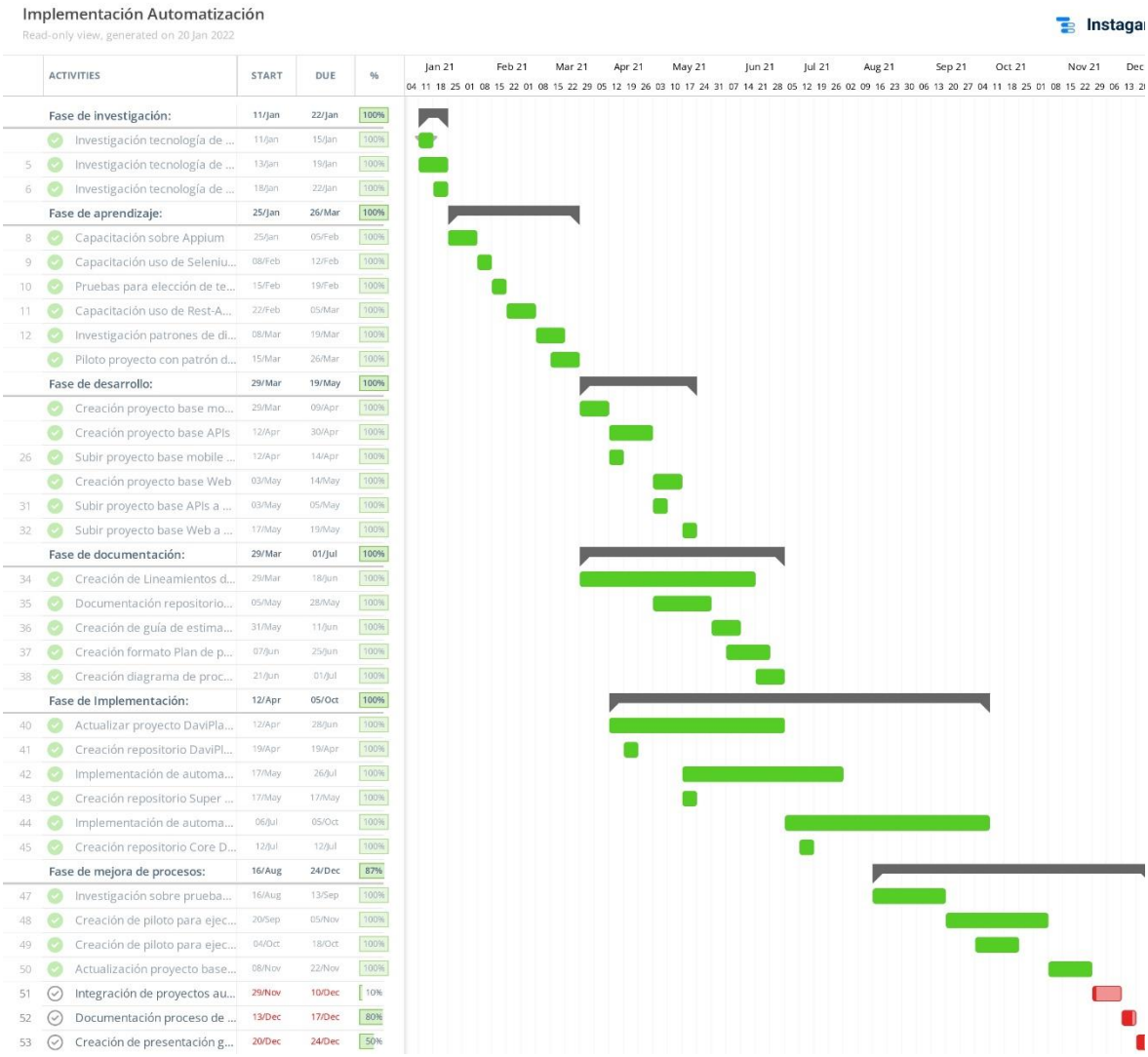

Ilustración 36. Cronograma del proyecto

9. CONCLUSIONES Y LÍNEAS FUTURAS

Luego de finalizar el proyecto con un resultado satisfactorio se deja una base para que más adelante se puedan seguir adelantado los procesos de automatización en la compañía. Se partió desde lo más básico que se tenía hasta llegar a un proceso muy avanzado que puede representar una gran ventaja para la empresa. Dentro de este proyecto se aprendió mucho sobre todo lo que envuelve la automatización de pruebas, desde qué es y cuando es conveniente utilizarla, los tipos de casos que pueden ser automatizados, los errores más frecuentes que se encuentran y sobre todo se aprendió que es un proceso que no retorna beneficios inmediatamente, sino que debe darse suficiente espacio para ver la reducción en los tiempos de prueba y a su vez ver la mejora del equipo de testing manual.

El proceso que se deja en la compañía abre la puerta a seguir investigando y avanzar aún más en la optimización de los recursos:

1. Se debe continuar con la automatización en los demás proyectos de la empresa, al existir ya las bases será más fácil replicar los resultados en los demás equipos.
2. Se debe continuar la optimización del proyecto de automatización base para API’s, ya que no está trabajando totalmente con el patrón de diseño Screenplay ni tampoco integra las librerías de Serenity.
3. Se debe integrar el flujo de pruebas remotas en las granjas de dispositivos en los proyectos actuales. Ya se cuenta con una base sólida y detallada de como utilizar la herramienta de AWS Device Farm, el paso siguiente es integrar los ajustes necesarios para que todos los proyectos puedan utilizar este servicio.

4. Se debe configurar la ejecución automatizada de los proyectos en Jenkins. Se realizó un piloto para dejar la base funcional, pero es necesario trasladar las configuraciones a cada uno de los proyectos que se ejecuten de aquí en adelante.
5. Se debe seguir trabajando en el documento de lineamientos de automatización, ya que las tecnologías seguirán en evolución, es por esto que se debe tener siempre presente con que herramientas se cuenta y cual es la mejor manera de utilizarlas.
6. Algunos proyectos cuentan con estructuras un poco menos organizadas que las que se plantearon al final de todo el proyecto, es por esto que deberían ser revisadas y reorganizarlas con el tiempo suficiente para que estén alineadas con las indicaciones entregadas.
7. Se debe continuar investigando sobre la automatización para nuevas tecnologías emergentes, tales como desarrollos de aplicaciones híbridas, que en ocasiones pueden generar conflictos con los scripts automatizados.
8. Se debe iniciar con la automatización de los casos de prueba en iOS, ya que de momento solo se cuenta con scripts para Android.

10. BIBLIOGRAFÍA

- Hernández, R. (2018, 9 noviembre). *BDD, Cucumber y Gherkin. Desarrollo dirigido por comportamiento*. Genbeta. https://www.genbeta.com/desarrollo/bdd-cucumber-y-gherkin-desarrollo-dirigido-por-comportamiento
- Atlassian. (s. f.). *Resumen de*. Bitbucket. Recuperado 20 de diciembre de 2021, de https://bitbucket.org/product/es/guides/getting-started/overview#a-brief-overview-of-bitbucket
- Briceño, G. (2020, 30 marzo). *Pruebas Automatizadas: tipos y conceptos erróneos*. Club de Tecnología. https://www.clubdetecnologia.net/blog/2020/pruebas-automatizadas-tipos-y-conceptos-erroneos/
- Canelo, M. M. (2021, 24 diciembre). *¿Qué es la Programación Orientada a Objetos?* Profile Software Services. https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/
- Hernández, R. (2018, 9 noviembre). *BDD, Cucumber y Gherkin. Desarrollo dirigido por comportamiento*. Genbeta. https://www.genbeta.com/desarrollo/bdd-cucumber-y-gherkin-desarrollo-dirigido-por-comportamiento

- *Introducción a pruebas automatizadas con Appium*. (2021, 7 abril). Tribalyte Technologies. https://tech.tribalyte.eu/blog-introduccion-a-pruebas-automatizadas-con-appium
- Mosquera, J. (s. f.). *Conceptos básicos para una ejecución en Screenplay*. Pragma.com. https://www.google.com/amp/s/www.pragma.com.co/academia/lecciones/conceptos-basicos-para-una-ejecucion-en-screenplay%3fhs_amp=true
- Ordoñez, M. A. E. (2021, 7 septiembre). *Qué es Gherkin*. OpenWebinars.net. https://www.google.com/amp/s/openwebinars.net/amp/blog/que-es-gherkin/
- *Pruebas de software*. (s. f.). La Oficina de Proyectos de Informática. http://www.pmoinformatica.com/p/pruebas-de-software.html?m=1
- *¿Qué es AWS Device Farm? - AWS Device Farm*. (s. f.). Aws Docs. Recuperado 20 de diciembre de 2021, de https://docs.aws.amazon.com/es_es/devicefarm/latest/developerguide/welcome.html
- *Resumen de*. (s. f.). Bitbucket. Recuperado 20 de diciembre de 2021, de https://bitbucket.org/product/es/guides/getting-started/overview#a-brief-overview-of-bitbucket

- Rodríguez, J. (s. f.). *Descubre cómo automatizar Service Tests con Rest-assured*. SDOS. https://www.sdos.es/blog/descubre-como-automatizar-service-tests-con-rest-assured
- SEAS, Estudios Superiores Abiertos. (2019, 7 agosto). *Conoce el lenguaje de programación Java | Blog SEAS*. Blog de SEAS. https://www.seas.es/blog/informatica/conoce-el-lenguaje-de-programacion-java/
- *Selenium y la automatización de las pruebas | Marco de Desarrollo de la Junta de Andalucía*. (s. f.). Junta de Andalucía. https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/381
- Simões, C. (2021, 27 julio). *¿Qué es Node.js y para qué sirve?* ITDO Desarrollo web y APPs Barcelona. https://www.itdo.com/blog/que-es-node-js-y-para-que-sirve/
- Soto, M. (2021, 12 diciembre). *Automatización de Pruebas Funcionales: Serenity BDD+Screen Play+Java*. Medium. https://medium.com/@marcela.soto/automatizaci%C3%B3n-de-pruebas-funcionales-serenity-bdd-screen-play-java-942be8217fca#:%7E:text=%C2%BFQu%C3%A9%20es%20Serenity%20BDD%3F,Sus%20principales%20caracter%C3%ADsticas%20son%3A&text=Ajustar%20las%20pruebas%20automatizadas%20a%20las%20necesidades%20del%20proyecto

- Unir, V. (2021, 1 diciembre). *Ingeniería de software: qué es, objetivos y funciones del ingeniero*. Universidad Virtual. | UNIR Colombia - Maestrías y Grados virtuales. https://colombia.unir.net/actualidad-unir/ingenieria-de-software-que-es-objetivos/