

**DESARROLLO DE FUNCIONALIDADES PARA UNA APLICACIÓN ANDROID
NATIVA DE NAVEGACIÓN Y GESTIÓN DE ENCARGOS PARA FLOTAS DE
TRANSPORTE ESTUDIANTIL Y FLOTAS DE RESIDUOS DE DIFERENTES
TIPOS EN CANADÁ Y ESTADOS UNIDOS**



KEVIN FELIPE MENESES PALTA

CORPORACIÓN UNIVERSITARIA AUTÓNOMA DEL CAUCA

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS INFORMÁTICOS

PASANTÍA

2021

**DESARROLLO DE FUNCIONALIDADES PARA UNA APLICACIÓN ANDROID
NATIVA DE NAVEGACIÓN Y GESTIÓN DE ENCARGOS PARA FLOTAS DE
TRANSPORTE ESTUDIANTIL Y FLOTAS DE RESIDUOS DE DIFERENTES
TIPOS EN CANADÁ Y ESTADOS UNIDOS**



KEVIN FELIPE MENESES PALTA

Trabajo de Grado para optar al título de Ingeniero de Sistemas Informáticos

Director Académico:

Ing. Yuli Garcés

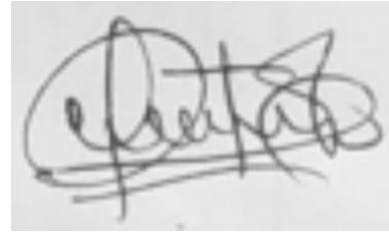
CORPORACIÓN UNIVERSITARIA AUTÓNOMA DEL CAUCA

FACULTAD DE INGENIERÍA

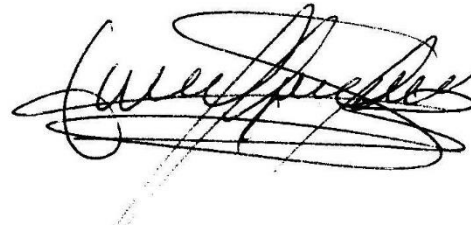
INGENIERÍA DE SISTEMAS INFORMÁTICOS

PASANTÍA

2021

A handwritten signature in black ink, appearing to be 'Yuli Sidney Garcés Bolaños Esp', written in a cursive style.

Directora. Ing. Yuli Sidney Garcés Bolaños Esp

A handwritten signature in black ink, appearing to be 'Sandra Patricia Castillo Mag', written in a cursive style.

Jurado. Ing. Sandra Patricia Castillo Mag

A handwritten signature in black ink, appearing to be 'PABLO EDUARDO CAICEDO R', written in a cursive style.

Jurado. Ing. Pablo Eduardo Caicedo PhD

CONTENIDO

RESUMEN	1
INTRODUCCIÓN	3
CAPÍTULO I: PROBLEMÁTICA	5
1.1 Planteamiento del problema.....	5
1.2 Justificación.....	6
1.3 Objetivos	7
1.3.1 Objetivo general.....	7
1.3.2 Objetivos específicos.....	7
CAPÍTULO II: MARCO TEÓRICO.....	8
2.1 MARCO REFERENCIAL.....	8
2.1.1 Antecedentes.....	8
2.2 MARCO CONCEPTUAL	11
2.2.1 Metodología de desarrollo	11
2.2.2 Herramientas de integración y entrega continua	12
2.2.3 Versionamiento con Git y Git Flow.....	13
2.2.4 Principios SOLID	15
2.2.5 Arquitectura limpia.....	16
2.2.6 Patrón de repositorio	17
2.2.7 Componentes de arquitectura de Google	18
2.2.8 Prácticas recomendadas de Google.....	20
CAPÍTULO III: METODOLOGÍA.....	23
CAPÍTULO IV: DEFINICIÓN DE LOS REQUERIMIENTOS.....	25
4.1 Características del negocio	25

4.1.1	Características asociadas a la navegación.....	25
4.1.2	Características asociadas a la gestión de encargos.....	26
4.2	Requerimientos funcionales.....	26
4.3	Requerimientos no funcionales.....	28
CAPÍTULO V: DESARROLLO.....		30
5.1	Resumen del trabajo realizado.....	30
5.2	Investigación de frameworks, librerías y herramientas.....	31
5.2.1	Peticiones HTTP.....	31
5.2.2	Manejo de base de datos.....	34
5.2.3	Procesos asíncronos y eventos.....	36
5.2.4	Inyección de dependencias.....	38
5.2.5	Descarga y manejo de imágenes.....	40
5.2.6	Análisis y manejo de información JSON.....	41
5.2.7	Manejo de RecyclerView y adaptadores.....	42
5.2.8	Pruebas unitarias y de instrumentación.....	44
5.2.9	Otras herramientas y librerías experimentales.....	47
5.3	Prueba de concepto de la investigación.....	50
5.3.1	Evidencias y funcionamiento.....	52
5.3.2	Conclusiones de la prueba:.....	54
5.4	Elementos propuestos para el desarrollo.....	56
5.4.1	Arquitectura general.....	56
5.4.2	Patrones de arquitectura.....	58
5.4.3	Patrones de comportamiento.....	58
5.4.4	Librerías seleccionadas.....	58

5.5	Reportes de calidad del proyecto	59
5.5.1	Cobertura del código	59
5.5.2	Métricas de inspección del código	60
5.5.3	Implementación	60
5.6	Barra de búsqueda personalizada.....	66
5.6.1	Diseño XML	67
5.6.2	Comportamiento de la barra	68
5.7	Creación de rutas	71
5.7.1	Evento del botón de creación de rutas de la barra	73
5.7.2	Guardar ruta en la base de datos	73
5.8	Escenarios sin conexión de la aplicación	77
5.8.1	Manejo de conexión a Internet.....	78
5.8.2	Manejo de conexión al servidor	80
5.9	Lector de eventos del brazo mecánico de parada.....	85
5.10	Manejo de eventos de GPS, RFID y brazo mecánico de parada	88
5.11	Pantalla de navegación para cada negocio	91
CAPÍTULO VI: RESULTADOS.....		94
6.1	Caso de estudio	94
6.2	Resultados de las pruebas:.....	102
CAPÍTULO VII: CONCLUSIONES Y RECOMENDACIONES		127
REFERENCIAS.....		129

LISTA DE TABLAS

Tabla 1. Requerimientos funcionales	26
Tabla 2. Requerimientos no funcionales	28

LISTA DE FIGURAS

Figura 1. Diagrama de Git Flow.....	15
Figura 2. Diagrama de arquitectura.....	17
Figura 3. Diagrama del patrón de repositorio	18
Figura 4. Diagrama de pirámide de pruebas	21
Figura 5. Resumen del trabajo realizado para SafeFleet Mobile.....	30
Figura 6. Ticket de Jira PPM-4	31
Figura 7. Ticket de Jira PPM-36.....	50
Figura 8. Pantalla de inicio de la aplicación The CatSelector.....	52
Figura 9. Pantalla de bienvenida de la aplicación The CatSelector.....	53
Figura 10. Lista de datos de la aplicación The CatSelector.....	54
Figura 11. Diagrama de arquitectura de la aplicación	57
Figura 12. Tickets de Jira VMN-527 y VMN-1264	59
Figura 13. Reporte de calidad del proyecto.....	64
Figura 14. Pull requests de los tickets VMN-527 y VMN-1264.....	65
Figura 15. Tickets de Jira VMN-461, VMN-876 y VMN-961	67
Figura 16. Diagrama de barra de búsqueda personalizada.....	69
Figura 17. Pull request del ticket VMN-876	70
Figura 18. Tickets de Jira VMN-960, VMN-1226, VMN-1227 y VMN-1231	72
Figura 19. Diagrama del caso de uso crear ruta	74
Figura 20. Pull requests de los tickets VMN-1226 y VMN-1227	76
Figura 21. Tickets de Jira VMN-1128 y VMN-1265	78
Figura 22. Diagrama del administrador de red	79
Figura 23. Pull request del ticket VMN-1265	80
Figura 24. Ticket de Jira VMN-1178.....	80

Figura 25. Diagrama del administrador de estado del servidor	81
Figura 26. Diagrama del observador del estado del servidor	82
Figura 27. Pull requests del ticket VMN-1178	84
Figura 28. Tickets VMN-1438 y VMN-1449.....	85
Figura 29. Diagrama del lector de brazo mecánico de parada.....	86
Figura 30. Pull request del ticket VMN-1438	87
Figura 31. Ticket VMN-1327.....	88
Figura 32. Diagrama de clases del flujo del servicio de manejo de eventos	89
Figura 33. Pull request del ticket VMN-1438	90
Figura 34. Ticket VMN-1347.....	91
Figura 35. Diagrama de clases de la pantalla de navegación para los diferentes negocios.....	92
Figura 36. Pull request del ticket VMN-1347	93
Figura 37. Ruta trazada del recorrido.....	94
Figura 38. Segmento 1: punto de partida y parada 2	96
Figura 39. Segmento 2: parada 3.....	96
Figura 40. Segmento 3: parada 4.....	97
Figura 41. Segmento 4: parada 5.....	97
Figura 42. Segmento 5: parada 6.....	98
Figura 43. Segmento 6: parada 7.....	98
Figura 44. Segmento 7: parada 8.....	99
Figura 45. Segmento 8: parada 9.....	99
Figura 46. Segmento 9: parada 10.....	100
Figura 47. Segmento 10: parada 11	100
Figura 48. Segmento 11: destino final.....	101
Figura 49. Seguimiento de la ruta realizada	101
Figura 50. Prueba 1 de RF-01	102
Figura 51. Prueba 2 de RF-01: confirmación.....	103
Figura 52. Prueba 2 de RF-01: navegación.....	103
Figura 53. Prueba 3 de RF-01	104

Figura 54. Prueba 1 de RF-02.....	105
Figura 55. Prueba 2 de RF-02: base de datos de la aplicación.....	106
Figura 56. Prueba 2 de RF-02: lista de rutas.....	106
Figura 57. Prueba 1 de RF-03.....	107
Figura 58. Prueba 2 de RF-03.....	108
Figura 59. Prueba 3 de RF-03.....	109
Figura 60. Prueba 4 de RF-03: base de datos de la aplicación.....	110
Figura 61. Prueba 1 de RF-05.....	111
Figura 62. Prueba 2 de RF-05.....	112
Figura 63. Prueba 3 de RF-05.....	113
Figura 64. Prueba 1 de RF-07: respuesta exitosa al envío del evento.....	114
Figura 65. Prueba 2 de RF-07: respuesta exitosa al envío del evento.....	115
Figura 66. Prueba 3 de RF-07: respuesta exitosa al envío del evento.....	116
Figura 67. Prueba 4 de RF-07: respuesta exitosa al envío del evento.....	117
Figura 68. Prueba 5 de RF-07: respuesta exitosa al envío del evento.....	118
Figura 69. Prueba 6 de RF-07: base de datos de la aplicación.....	118
Figura 70. Prueba 7 de RF-07: base de datos de la aplicación.....	119
Figura 71. Prueba 1 de RF-08.....	120
Figura 72. Prueba 2 de RF-08.....	121
Figura 73. Prueba 3 de RF-08.....	122
Figura 74. Prueba 4 de RF-08.....	123
Figura 75. Prueba 5 de RF-08.....	124
Figura 76. Prueba 6 de RF-08.....	125
Figura 77. Prueba 7 de RF-08.....	126

RESUMEN

Uno de los temas más importantes respecto al desarrollo y uso de nuevas tecnologías es su integración con los diferentes sectores de la sociedad, incluyendo el transporte como uno de los principales para el funcionamiento urbano. La visión del cliente para el que se desarrolla el producto software es: flotas y transporte seguro, en este caso en el contexto escolar y ambiental de los países de Canadá y Estados Unidos, donde existen condiciones de riesgo para los pasajeros (estudiantes menores de edad) y donde también se busca contribuir a la correcta gestión de diferentes tipos de desechos. El producto que se desarrolla consiste en una aplicación Android nativa de navegación para buses escolares y vehículos de desechos en un dispositivo especial creado por la misma compañía. El aporte que se hizo en esta pasantía es un módulo de seguimiento, envío de eventos y consulta de rutas, junto con las investigaciones, tecnologías secundarias y conceptos que conlleva este tipo de desarrollo para asegurar la integración y calidad del producto. Además, se evidencia el correcto funcionamiento del módulo desarrollado con un caso de estudio en el contexto de la ciudad de Medellín.

PALABRAS CLAVE: transporte, transporte escolar, vehículos de desechos, flotas seguras, seguridad, desarrollo de software, Android, navegación.

ABSTRACT

One of the most relevant topics about the development and use of new technologies is its integration with the different sectors of society, including transport as one of the principals for urban functioning. The vision of the client for whom the software product is developed is safe fleets and transport, in this case in the school and environmental context of Canada and the United States, where risk conditions for the passengers (underage students) exist, and where also it seeks to contribute to the correct management of different kinds of waste. The product that is developed consists of an Android native application for scholar and waste vehicles navigation on a special device created by the same company. The contribution made in this internship is a tracking, event sending, and route query module along with the investigations, secondary technologies, and concepts that this type of development entails ensuring the integration and quality of the product. In addition, the correct functioning of the developed module with a study case in the context of Medellín city is evidenced.

KEYWORDS: transport, scholar transport, waste vehicles, safe fleet, safety, software development, Android, navigation.

INTRODUCCIÓN

El siguiente informe presenta el trabajo de pasantía realizado durante varios meses en la empresa llamada Productora de Software Limitada (PSL), en su sede ubicada en Medellín, Antioquia. La cual fue recientemente adquirida por Perficient, Inc, y ahora es conocida como Perficient Latin America.

En palabras de su página oficial, “PSL es una empresa de desarrollo de software ágil, especializada en ofrecer servicios de TI desde sus centros de desarrollo en Latinoamérica”. Con más de 30 años de experiencia ha sido reconocida “con prestigiosos premios internacionales de Ingeniería de Software, tales como el IEEE/SEI Watts Humphrey Award (ganado antes por la NASA e IBM) y el premio a la Excelencia del Software Europeo”.

PSL se compone por “dos grandes líneas de negocio servicios de outsourcing de TI (desarrollo de software, aplicaciones web, aplicaciones móviles, aplicaciones de misión crítica, software factory, outsourcing de recursos, BPO) y el desarrollo de productos pre-programados de software (sistemas de gestión empresarial ERP, sistemas de gestión de garantías, plataformas web-banking, entre otras)”.

Durante su trayectoria la compañía “ha desplegado cientos de proyectos exitosos con clientes que abarcan desde las más grandes compañías del continente (Deloitte, FedEx, Bridgestone, Ecopetrol, el Canal de Panamá, Bancolombia, Grupo Aval) hasta exitosas y pujantes empresas PYME y startups de base tecnológica” [1].

Ante la necesidad creciente de software para sistemas operativos móviles, aumenta la demanda de desarrolladores de este tipo de tecnologías y surgen nuevos proyectos para los diferentes clientes de la empresa. En este contexto, deciden adquirir nuevos talentos para suplir su necesidad, momento en el que consideran mi solicitud para aplicar al cargo de desarrollador Android, iniciando un proceso de contratación para ser aceptado posteriormente y entrar a ser parte de uno de sus proyectos.

La motivación principal para el desarrollo de este proyecto de pasantía es obtener el título de Ingeniero de Sistemas Informáticos, además de enfocar el ejercicio de la carrera profesional hacia el desarrollo de aplicaciones móviles, específicamente para el sistema operativo Android, y tener la posibilidad de aportar nuevos conocimientos a la empresa.

El proyecto en el que se basa este trabajo se llama SafeFleet Mobile, de la empresa SafeFleet Holdings, LLC, ubicada en Estados Unidos y Canadá, cliente de los servicios de outsourcing de PSL, que incluye las aplicaciones:

- Navigator: para la gestión de vehículos, estudiantes y rutas de transporte escolar.
- WasteApp: para la gestión de vehículos y encargos basados en diferentes tipos de desechos.
- Base Library: repositorio de uso general compartido entre los diferentes proyectos Android de SafeFleet.

Las cuales trabajan sobre un dispositivo llamado VT7 (Vehicle Tablet 7”) el cual es una tableta de 7 pulgadas con un sistema operativo Android que cuenta con una estación de acoplamiento que se conecta a diferentes dispositivos del vehículo.

Entre los temas que se presentan en este informe se encuentran: librerías, componentes y patrones de arquitectura de Google, arquitectura CLEAN para desarrollo de aplicaciones modularizadas en Android, principios SOLID para programación orientada a objetos, técnicas de versionamiento usando Git, herramientas de reporte de calidad del código y herramientas de integración y entrega continua. También se explica el desarrollo de un módulo de seguimiento, envío de eventos, consulta y creación de rutas para el proyecto asignado, implementando varios de los temas mencionados anteriormente. Sin embargo, con el propósito de proteger los intereses y la confidencialidad del producto del cliente, la explicación de su desarrollo se limita a una representación abstracta y evidencia gráfica de su funcionamiento.

CAPÍTULO I: PROBLEMÁTICA

1.1 Planteamiento del problema

En Estados Unidos y Canadá, como en muchos otros países del mundo, existen empresas que se encargan de prestar servicios de transporte de diferentes tipos, ya sea de personas, de carga de materiales, de limpieza o de desechos, entre otros. Algunas de estas empresas llegan a hacerse lo suficientemente grandes para operar a nivel nacional, por lo que hacer seguimiento a los vehículos y sus encargos de manera tradicional no resulta ser tan eficiente como debería.

En el Norte de América existen en la actualidad flotas de transporte estudiantil compuestas en un 70% de los típicos buses escolares amarillos, aproximadamente 500 mil vehículos en Estados Unidos, y otro 30% de otros vehículos como carros particulares, vanes y camionetas [2]. Entre 2017 y 2018 se reportaron un poco más de 23 millones de estudiantes transportados diariamente por servicios de transporte público escolar y un poco menos de 2 millones por servicios de transporte privados en Estados Unidos [3].

Relacionado a esto, existen casos en el mundo en los que debido a las fuertes estaciones climáticas niños y jóvenes han muerto congelados en paradas de buses, camino a su escuela o de regreso a casa después de perder el autobús escolar, como es el caso de un joven de 15 años que murió de hipotermia en Devon, Inglaterra [4], o al ser dejados dentro de un autobús por descuido del conductor, lo que casi le cuesta la vida a un niño de 5 años en Delaware, en Estados Unidos [5].

En consecuencia, uno de los objetivos del plan departamental de transporte de 2019-2020 de Canadá es alcanzar un sistema seguro y protegido mejorando la monitorización, coordinación y el desarrollo de herramientas digitales, además de buscar reducir las emisiones dañinas para el medio ambiente [6], por lo que las ventas de vehículos eléctricos aumentarán en los próximos años, incrementando el tamaño total de las flotas, lo que supone una mayor carga en las tareas relacionadas.

Asimismo, el transporte de desechos sólidos y cargas ocupan un papel importante en Norte América, pues la recolección es un paso crítico en el manejo de este tipo de tareas, así como su clasificación por clases de residuos y el uso de vehículos adecuados para su transporte. Según estimaciones realizadas por The World Bank, en un periodo de tiempo del 2016 al 2050 aumentará en América del Norte la generación de desechos de 289 a 396 millones de toneladas al año [7], lo que le da aún más prioridad a la necesidad de fortalecer y generar herramientas que garanticen la eficiencia de todo el proceso.

1.2 Justificación

La motivación para llevar a cabo esta pasantía es la posibilidad de participar en un proyecto internacional, aprender e implementar las tecnologías, metodologías y prácticas actuales en el desarrollo de software, además de apoyar la construcción de herramientas tecnológicas del futuro que contribuyen a la mejora de todo tipo sistemas, en este caso, el manejo de flotas que ofrecen diferentes servicios de transporte.

En Norte América es común el manejo de buses escolares para el transporte de estudiantes a sus escuelas, por lo que existen padres preocupados por la seguridad de sus hijos, interesados en saber si han sido recogidos y si han llegado a salvo a su destino. Asimismo, detrás de las flotas de buses escolares existen empresas cuyo objetivo es mantener satisfechos a sus clientes asegurándose de llevar un control y gestión adecuada de sus vehículos y de esta forma garantizar a los padres la seguridad de sus hijos. Casos como los mencionados en la problemática de este proyecto, provocados por la falta de monitoreo del sistema de transporte, son los que le dan importancia a este desarrollo.

También, con la situación mundial del cambio climático, es importante hacer un buen manejo de los desechos [8]. Utilizar el vehículo adecuado dependiendo del tipo de desecho que se vaya a transportar es un paso fundamental en este proceso, y un sistema que integre sus elementos y los organice de manera eficiente para su

gestión, haciendo uso de las herramientas tecnológicas actuales [9], es una solución idónea para las empresas que se encargan de ello y para el medio ambiente.

Este proyecto de pasantía es posible llevarlo a cabo gracias a la utilización de herramientas tecnológicas basadas en el uso de dispositivos Android como tablets que funcionan con otros tipos de hardware, lectores de tarjetas RFID [10], entre otros, los cuales hacen parte del vehículo que necesita ser monitoreado y permiten enviar información GPS y llevar un registro de los estudiantes y de los encargos que se hacen en la labor de recolección de desechos.

En este orden, se espera poner en práctica los conocimientos adquiridos durante la carrera de Ingeniería de Sistemas Informáticos y utilizar sus bases en la investigación para el aprendizaje de nuevas tecnologías y herramientas y de esta forma implementarlas y dejarlas evidenciadas como insumo de este proyecto.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar un módulo de seguimiento, envío de eventos y consulta de rutas para una aplicación Android nativa de navegación y gestión de encargos para flotas de transporte estudiantil y flotas de residuos de diferentes tipos en Canadá y Estados Unidos, en la empresa PSL.

1.3.2 Objetivos específicos

- Identificar las características asociadas a la navegación y gestión de encargos en Canadá y Estados Unidos.
- Implementar los componentes de arquitectura pertinentes en el desarrollo de aplicaciones Android para la solución propuesta.
- Validar las funcionalidades implementadas a través de un caso de prueba de navegación y gestión de encargos en la ciudad de Medellín.

CAPÍTULO II: MARCO TEÓRICO

2.1 MARCO REFERENCIAL

2.1.1 Antecedentes

Gracias a la invención de los dispositivos inteligentes y su posterior desarrollo en la primera década del nuevo milenio, los sistemas operativos han sido transformados para responder a las necesidades de sus avances en hardware [11]. Es un hecho que hoy en día las aplicaciones móviles tienen gran importancia en la vida de las personas [12]. Poco a poco se han adaptado a las necesidades del mundo como un medio utilizado para la innovación, desde las aplicaciones más comunes como recordatorios, reproductores de música, calendarios, etcétera...[13] hasta la adaptación de plataformas web como Facebook, Twitter y YouTube, sin mencionar las nuevas aplicaciones de streaming como Spotify, Twitch o Netflix.

Es importante resaltar que, con el avance de la tecnología y la integración de múltiples elementos de hardware como GPS, cámara, sensores y wifi, estos dispositivos móviles se han convertido en los nuevos ordenadores de la actualidad, más compactos de lo que se habría pensado hace 30 o 40 años y bien integrados algunos modelos en red y servicios en la nube [14]. Debido a esto hoy se utilizan como parte de sistemas de información con diferentes tipos de necesidades y procesos, desde educativos e informativos, hasta empresariales e industriales y muchos otros [15].

Además, existen múltiples aplicaciones relacionadas con vehículos, como servicios de transporte personalizados, mapas online con informe de tráfico y localización, servicios de domicilio, transporte compartido, entre otras. Como parte de la investigación realizada para este proyecto, a continuación, se presentan algunas de las aplicaciones relacionadas con el desarrollo de la aplicación que se pretende apoyar en el transcurso de la pasantía:

Google Maps Api: Google Maps es una solución de navegación completa, su funcionamiento está basado en el uso de mapas con imágenes satelitales de todo el mundo, ofreciendo imágenes detalladas, localización, enrutamiento, navegación e información de los lugares que se desea visitar. También ofrece servicios Api para su utilización en proyectos como páginas web o aplicaciones móviles [16].

- Desventajas: su uso para producción es costoso y algunos de sus mapas pueden no contener calles privadas a las cuales se necesita acceso, como campus escolares.

Waze: esta aplicación es bastante similar a Google Maps, en cuanto a la ubicación, mapas y navegación, pero con la diferencia en que está basado en los aportes de su comunidad de usuarios, los cuales comparten información en tiempo real de accidentes, tráfico, puntos de control, entre otros. También es propiedad de Google.

- Desventajas: su SDK no permite incluirla en una aplicación de navegación, solamente permite proporcionar datos [17].

Skole: es una aplicación diseñada para manejar el negocio del transporte escolar. Cuenta con flavors o versiones para transportadores y para padres, en las que se maneja el estado de los pasajeros y se envía notificaciones a los padres cuando su hijo es recogido y cuando se baja en la escuela. También ayuda a gestionar recordatorios por el cobro del servicio de transporte. Tiene un costo mensual de uso para los usuarios [18].

- Desventajas: no cuenta con navegación ni localización por lo cual no se puede saber dónde se encuentra el vehículo en tiempo real ni las paradas en las que se debe recoger el estudiante.

Rube: otra aplicación para gestionar servicios de transporte escolar. Al igual que la anterior cuenta con versiones para conductores y para padres en las que se muestran los estudiantes, pero da la posibilidad de acceder a algunos datos de los padres, crear paradas y visualizarlas en un mapa junto con la ubicación en tiempo real [19].

- Desventajas: no cuenta con servicio de enrutamiento e instrucciones en tiempo real. Además, la versión para padres tiene varios comentarios negativos sobre mal funcionamiento por parte de sus usuarios.

Yo-Waste: su aplicación móvil tiene dos versiones, una para personas que buscan contratar un servicio de manejo de desechos y otra para empresas que prestan servicios de transporte y manejo de desechos, en la cuales sus usuarios se pueden registrar y en el caso de los clientes escoger la empresa con la que desean contratar servicios [20].

- Desventajas: no cuenta con servicio de enrutamiento ni instrucciones en tiempo real, además está hecha para ser abierta en cuanto a competencia, lo que no es ideal en este negocio.

Por otra parte, es importante destacar las tecnologías usadas en el desarrollo de software y la programación de aplicaciones móviles, así como algunas arquitecturas implementadas en diferentes tipos de proyectos:

Flutter: es un SDK de Google enfocado en el diseño y experiencia de usuario, el cual permite crear aplicaciones de diferentes tipos, entre ellas aplicaciones móviles multiplataforma, es decir que genera un archivo para dispositivos Android y otro para IOS. Entre sus funcionalidades incluye la posibilidad de integrarse a otras plataformas de desarrollo o trabajar por sí sola [21].

- Desventajas: tiene más de 5000 asuntos sin resolver en su repositorio oficial en GitHub [22]. Está enfocado más en el diseño de interfaces que en la construcción de aplicaciones robustas.

React Native: es un framework declarativo de interfaces de usuario, creado por Facebook, que permite desarrollar aplicaciones multiplataforma, bastante parecido al framework original creado para desarrollo web, facilitando a los desarrolladores familiarizados con él crear aplicaciones móviles [23][24].

- Desventajas: su alcance puede estar limitado cuando se trata de escribir librerías para dispositivos hardware. No es tan robusto como las aplicaciones nativas ni tiene el mismo rendimiento [25].

2.2 MARCO CONCEPTUAL

2.2.1 Metodología de desarrollo

2.2.1.1 Programación extrema (XP)

Es la más específica de las metodologías ágiles ya que se enfoca en prescribir los procesos de ingeniería. Prácticas como desarrollo basado en pruebas (TDD), pruebas del lado del cliente, integración continua, iteraciones cortas, entregas pequeñas, programación en parejas, planeación, refactorización, estándares de código, entre otras. Todo lo cual se traduce eventualmente en un código de alta calidad. También puede resultar menos flexible que otras metodologías [26].

2.2.1.2 SCRUM

Es un marco de trabajo de metodologías ágiles que aporta flexibilidad al proceso de desarrollo de software, definiendo requerimientos considerados como pequeñas partes del desarrollo el cual se lleva a cabo de manera iterativa e incremental. Fue diseñado para incrementar la productividad del desarrollo y puede ser utilizado en proyectos pequeños y grandes. El proceso de este marco de trabajo consiste en Sprints, los cuales se llevan a cabo en un periodo de dos a cuatro semanas. Cada Sprint es una entrega que define una cantidad historias de usuario como requerimientos funcionales del cliente y otro tipo de tareas como mejoras, arreglo de errores, investigaciones, entre otras. Además, se llevan a cabo prácticas como reuniones diarias de 15 minutos para definir el estado de las tareas, refinamiento de las historias de usuario, planeación del Sprint, retrospectivas y limpieza [27].

2.2.2 Herramientas de integración y entrega continua

Un tema importante para el desarrollo de este proyecto fue la automatización de las tareas relacionadas con la integración, el despliegue y la entrega continua del proyecto de software. A continuación, se presenta en qué consiste cada una de ellas y cuáles son sus beneficios [28]:

- **Integración continua:** su objetivo es verificar el correcto funcionamiento del proyecto antes de integrar nuevos cambios a la rama principal. Para ello, realiza tareas automatizadas como construir y ejecutar las pruebas del proyecto en un entorno aislado, como una imagen de Docker en la nube. Algunas de las ventajas que ofrece es la temprana detección de errores, una menor cantidad de bugs generados y menos tiempo de pruebas por parte del equipo de aseguramiento de calidad.
- **Entrega continua:** es una extensión de la integración continua, con la diferencia de que se agrega encima de ella una tarea de automatización para el proceso de lanzamiento de versiones. Reduce el tiempo que gasta el equipo en la preparación de un lanzamiento y permite hacerlo de manera más frecuente.
- **Despliegue continuo:** el despliegue va un paso más allá de la entrega continua y consiste en entregar el producto al cliente, en el caso de las aplicaciones móviles subir automáticamente las nuevas versiones a las plataformas Play Store o App Store. Permite dar una mejor experiencia al cliente al entregar nuevas funcionalidades con más frecuencia, en lugar de hacer entregas cada semestre o trimestre.

Existen múltiples herramientas que permiten automatizar las tareas mencionadas como: Circle CI, Travis CI, Jenkins, Bamboo, TeamCity, Gitlab, entre otras. Las razones para implementar una u otra de estas herramientas dependen de las necesidades del proyecto ya que algunas ofrecen integración con herramientas que pueden necesitar o que el cliente utiliza para otros proyectos. En el caso de este

proyecto se utilizó **Jenkins** porque el cliente cuenta con un servidor para sus tareas de automatización con esta herramienta.

¿Qué es Jenkins?: Jenkins es un servidor de automatización de código abierto autónomo que se puede utilizar para automatizar todo tipo de tareas relacionadas con la creación, prueba y entrega o implementación de software. Puede instalarse a través de paquetes de sistema nativos, Docker o incluso ejecutarse de forma independiente en cualquier máquina que tenga instalado Java Runtime Environment (JRE) [29].

2.2.3 Versionamiento con Git y Git Flow

El desarrollo de software en equipo es una tarea que implica la escritura de código simultánea por parte de sus integrantes con el objetivo de entregar múltiples funcionalidades cada sprint bajo un marco de trabajo SCRUM. Existe la posibilidad de que durante el desarrollo los programadores hagan modificaciones en partes coincidentes del código y surjan conflictos en el momento de integrar el trabajo realizado por todos. Usualmente existen tareas que son desarrolladas primero que otras e integradas en el código fuente del proyecto, por lo que es necesario que el resto del equipo actualice su código sin perder el trabajo que han adelantado. Por esta razón existen proveedores de servicios en Internet basados en Git [30] que ofrecen herramientas para almacenar el código en sus servidores, comúnmente llamados repositorios, además de la posibilidad de tener diferentes versiones llamadas ramas y puntos de modificación llamados *commits*.

La técnica para el versionamiento de código llamada Git Flow [31] es utilizada transversalmente en los proyectos realizados por la empresa. También es una práctica común en la integración continua de software que consiste en el manejo de ramas dependiendo del objetivo, existen 5 tipos de ramas:

- **Feature:** cada rama creada para una funcionalidad se llama *feature* y la convención utilizada en el proyecto incluye el alcance o funcionalidad

afectada, las siglas del proyecto junto con el ticket desarrollado y el título de la tarea.

- **Develop:** es la rama en la que se van integrando las nuevas funcionalidades desarrolladas por el equipo. Cuando un desarrollador termina una funcionalidad crea un *pull request* de su rama apuntando a *develop* y una vez aprobado por los miembros del equipo se integran los cambios realizados.
- **Release:** una vez completadas todas las tareas del alcance de una versión, se crea una rama de release, cuyo objetivo es preparar el código para un lanzamiento. Esta rama se utiliza para pruebas, regresiones, etc. y no se pueden integrar nuevas funcionalidades sobre ella, únicamente se pueden arreglar errores, agregar documentación o tareas relacionadas con el lanzamiento.
- **Master:** es la rama que contiene el historial de lanzamientos oficiales del proyecto. Cuando se han completado las tareas de lanzamiento y se ha verificado el código correctamente, se integra esta rama con la rama del release específico.
- **Hotfix:** cuando se encuentra un bug en una versión lanzada se crea este tipo de rama para arreglar el error y se hacen dos *pull request*, uno apuntando a master, para arreglar la versión y otro a develop para actualizar el código de desarrollo.

El flujo y las ramas de Git Flow se pueden visualizar en el siguiente diagrama:

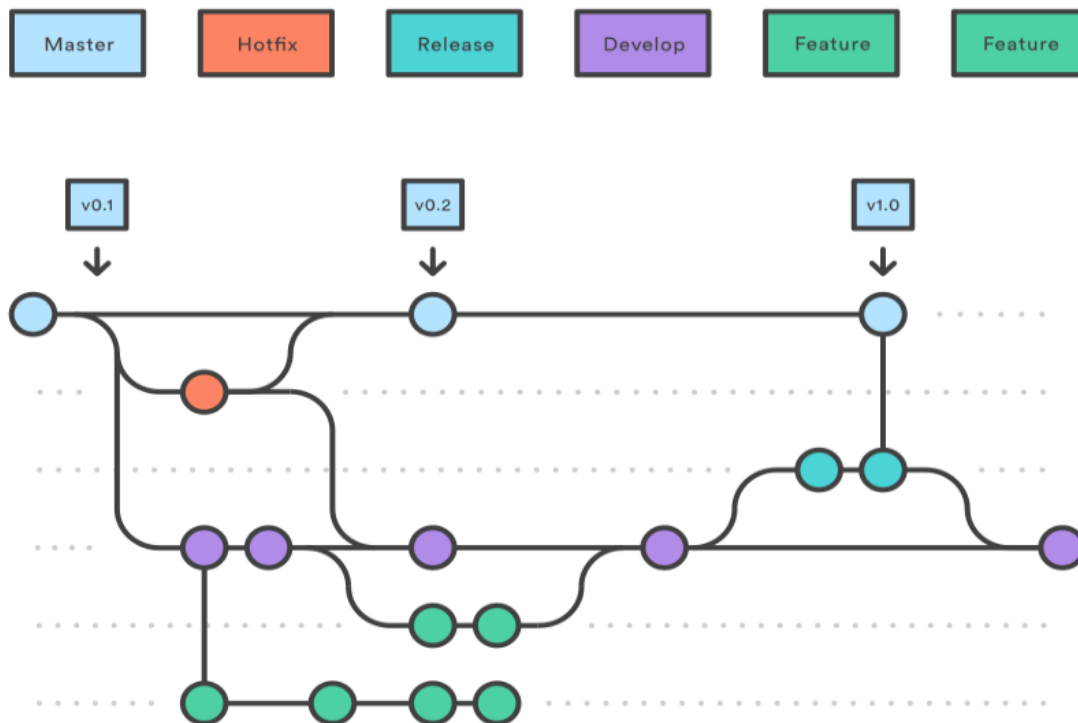


Figura 1. Diagrama de Git Flow. Fuente: Atlassian [31]

2.2.4 Principios SOLID

Una de las guías requeridas e implementadas en el desarrollo de este proyecto son los principios SOLID un acrónimo que indica cinco principios para la adecuada programación orientada a objetos [32]:

- Responsabilidad única (single responsibility): cada módulo o clase debería tener responsabilidad sobre una única funcionalidad del software.
- Abierto/cerrado (open/closed): las clases, módulos y funciones deberían estar abiertos a la extensión y cerrados a la modificación.
- Sustitución de Liskov (Liskov substitution): los objetos de un programa deberían ser reemplazables con instancias de sus subtipos sin alterar su correcto funcionamiento.
- Segregación de interfaces (interface segregation): ninguna clase debería depender de métodos que no utiliza. Es mejor tener interfaces específicas para cada funcionalidad.

- Inversión de dependencias (dependency inversion): módulos de alto nivel no deberían depender de módulos de bajo nivel ambos deberían depender de abstracciones. Las abstracciones no deberían depender de implementaciones, las implementaciones deberían depender de abstracciones.

2.2.5 Arquitectura limpia

La arquitectura de un proyecto de desarrollo de software es una parte fundamental para garantizar la mantenibilidad del producto, una de las métricas de calidad definidas como estándar en la norma *ISO 9126* [33]. La mantenibilidad cuenta con cuatro características:

- Estabilidad: fortaleza con la que el software responde ante situaciones inesperadas.
- Facilidad de análisis: rapidez del software para diagnosticar y mostrar resultados de fallos o circunstancias excepcionales.
- Facilidad de cambio: versatilidad con la que el software puede ser actualizado o modificado.
- Facilidad de pruebas: simplicidad con la que se pueden verificar las diferentes funcionalidades del software.

Robert C. Martin conocido como Uncle Bob (el tío Bob) es un programador desde 1970, cofundador de *cleancoders.com* y fundador de Uncle Bob Consulting LLC que ofrece consultorías de software, entrenamiento y servicios de habilidades de desarrollo a grandes compañías internacionales. También es autor de múltiples artículos y libros sobre programación, como *Clean Architecture* [32], el libro en el que está basada la arquitectura implementada para este proyecto.

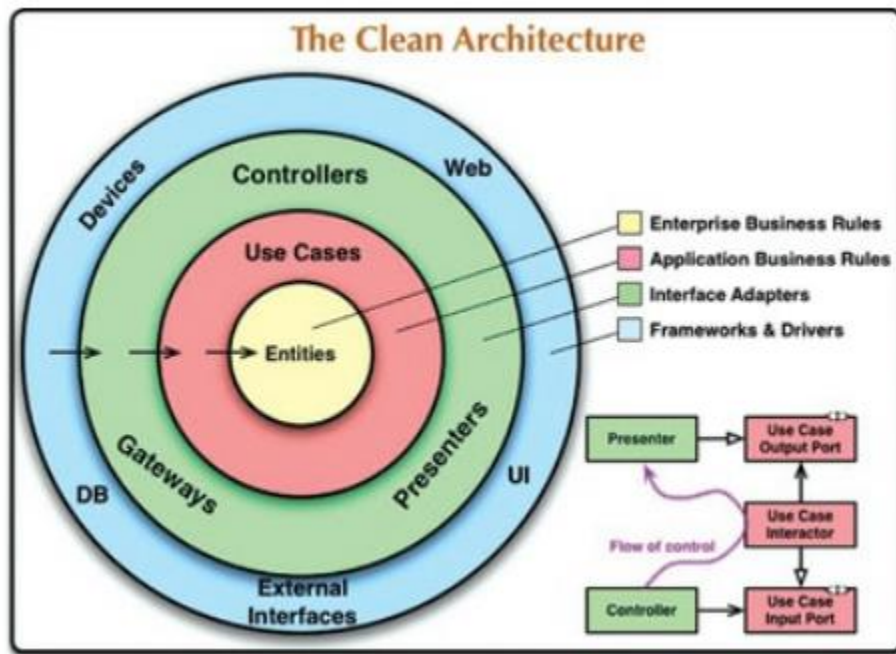


Figura 2. Diagrama de arquitectura. Fuente Clean Architecture [32]

El propósito de la arquitectura limpia como muchas otras es separar los asuntos del software en diferentes capas. La característica que la hace particular es la regla de dependencia, en la que ninguna de las capas internas puede saber nada de las capas externas. Nada que haya sido declarado en una capa externa puede ser mencionado en una capa interna, incluyendo funciones, clases, variables, etc. Esto hace más fácil el cambio de frameworks, librerías y fuentes de datos sin modificar la presentación o la lógica de negocio, ya que la aplicación no depende directamente de ellos, lo cual se traduce en desacoplamiento entre capas.

2.2.6 Patrón de repositorio

El patrón de repositorio consiste en una capa encargada de separar y manejar los datos de la aplicación del resto de componentes. El repositorio es un mediador entre las diferentes fuentes de datos como modelos de persistencia, servicios web y bases de datos locales permitiendo utilizar múltiples *backends*. Una de sus utilidades es obtener datos de fuentes remotas como servicios web y guardarlos en la base de datos local de la aplicación para ofrecer una mejor

experiencia de usuario al tener menor tiempo de respuesta y beneficiando a aquellos que tienen conexión intermitente [34]. A continuación, se muestra el gráfico del patrón de repositorio:

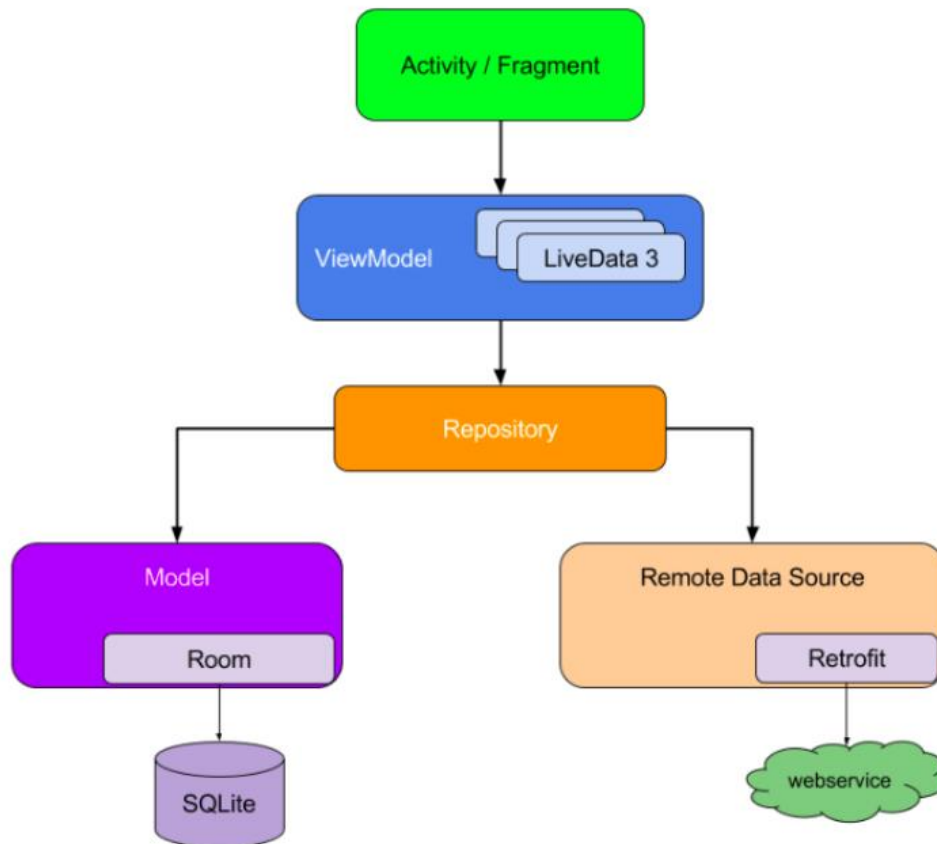


Figura 3. Diagrama del patrón de repositorio. Fuente Android Developers [34]

2.2.7 Componentes de arquitectura de Google

Según Google: “los componentes de la arquitectura de Android son una colección de bibliotecas que te ayudan a diseñar apps sólidas que puedan someterse a prueba y admitan mantenimiento. Comienza con clases para administrar el ciclo de vida de los componentes de la IU y manejar la persistencia de los datos” [35]. Algunos de los componentes implementados en este proyecto son:

2.2.7.1 Patrón de arquitectura MVVM

El patrón de arquitectura que se utilizó en este proyecto es MVVM: Model, View, ViewModel. El cual destaca el uso de dos clases que hacen parte de los componentes de arquitectura de Google:

- **ViewModel:** es una clase mediadora entre los modelos de datos obtenidos desde el repositorio y la interfaz de usuario o vista sin conocer nada de ella. Separa la responsabilidad de la lógica de datos de las vistas y su alcance comparte el ciclo de vida de las actividades. Cuando la actividad se destruye el ViewModel también. Generalmente contiene propiedades de tipo LiveData que se observan desde las actividades [36].
- **LiveData:** es una clase contenedora de datos observable optimizada para seguir los ciclos de vida de las actividades y fragmentos lo que permite la programación reactiva de las vistas. Conoce los ciclos de vida de las actividades por lo que únicamente notifica a observadores que se encuentren en un ciclo activo. También genera persistencia en los datos cuando se reinicia el ciclo de vida, como cuando se gira la pantalla y se recrea la actividad [37].

2.2.7.2 Biblioteca de persistencias Room

Room es una librería que ofrece una capa de abstracción para SQLite y permite acceder a la base de datos de manera sencilla para realizar transacciones, guardar y consultar datos con esquemas relacionales. Utilizando anotaciones e interfaces esta librería se encarga de generar implementaciones para las operaciones que se definan en la misma [38].

2.2.7.3 Corrutinas de Kotlin

Kotlin es el nuevo lenguaje de programación recomendado por Google para el desarrollo de aplicaciones Android y cuenta con corrutinas que proporcionan una API que permite escribir código asíncrono. Permiten definir el alcance y el momento

en que se ejecutan gracias a los *CoroutineScope*, el hilo en el que se desean ejecutar gracias a los *Dispatchers* y además se encuentran bien integradas con el resto de las componentes de arquitectura [39].

2.2.8 Prácticas recomendadas de Google

2.2.8.1 Inyección de dependencias

La inyección de dependencias es una práctica ampliamente usada en la programación que facilita la reutilización, refactorización e implementación de pruebas del código. El concepto detrás de la inyección de dependencias es lo que se hace comúnmente en los métodos y funciones a los cuales se les provee los parámetros que necesitan para ejecutar una acción, así mismo las clases deben obtener las instancias que necesiten a través de sus constructores en lugar de crearlas por sí mismas [40]. Tener un módulo encargado de crear y proveer las dependencias que cada clase necesita en combinación al uso de interfaces para cada implementación permite cumplir el principio de inversión de dependencias de SOLID.

En el momento de la realización de este proyecto la librería recomendada para la inyección de dependencias en Android es Dagger [41], la cual cuenta con módulos y componentes que permiten definir el alcance de las instancias que se proveen para cada clase dentro de aplicación en cada uno de los módulos definidos en la arquitectura, como se puede ver en el gráfico del tercer apartado: *módulos de la arquitectura*.

2.2.8.2 Implementación de pruebas

La implementación de pruebas para aplicaciones Android nativas [42] está basado en la pirámide de pruebas, la cual establece 3 niveles que determinan el tipo y cantidad de pruebas que se deben realizar, como se puede visualizar en el siguiente gráfico:

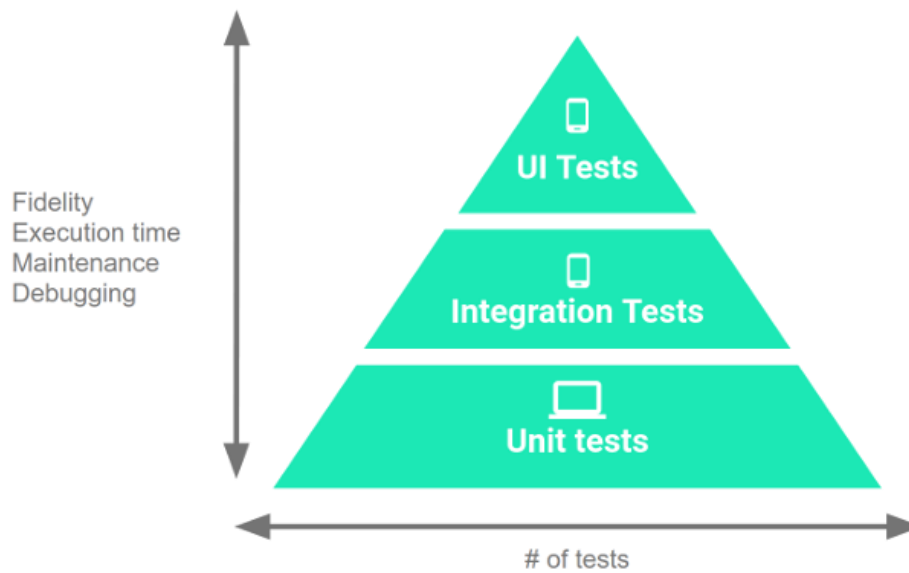


Figura 4. Diagrama de pirámide de pruebas. Fuente Android Developers [42]

Pruebas unitarias: las pruebas unitarias, ubicadas en la parte más baja de la pirámide de pruebas, son las más cuantiosas y son las encargadas de verificar la funcionalidad de los métodos de cada clase las cuales no requieren ningún tipo de simulación ni de interfaz de usuario, pero si requieren librerías para crear instancias ficticias de clases requeridas por éstos [43]. El framework de pruebas recomendado y utilizado en este caso es Junit 4 el cual nos permite crear clases y métodos de prueba junto con varias herramientas de verificación y aserciones para validar sus resultados. Para las dependencias ficticias Google recomienda el uso de Mockito el cual es común en proyectos que utilizan Java, sin embargo, para este proyecto se decidió utilizar MockK [44] debido a que es una librería desarrollada completamente para Kotlin que contiene funciones específicas para los tipos de clases que tiene Kotlin que no tiene Java.

Pruebas de interfaz de usuario: las pruebas de interfaz de usuario, ubicadas en la punta de la pirámide de pruebas, son las menos cuantiosas y se encargan de verificar el correcto funcionamiento de las interacciones de los usuarios con la interfaz de la aplicación. Por lo general este tipo de pruebas se realiza de manera manual por una persona encargada de aseguramiento de calidad (QA), sin embargo, se recomienda implementar pruebas automatizadas que hagan el proceso

más rápido y eficaz [45]. Existen herramientas para realizar pruebas automatizadas de diferentes tipos de aplicaciones: web, escritorio o móviles. Pero existe una librería nativa de Android llamada Espresso [46], recomendada por Google, que incluye el código de pruebas junto con el código de la aplicación y ofrece mejor integración con sus elementos y recursos.

Pruebas de integración: las pruebas de integración, en el medio de la pirámide de pruebas, se implementan de diferentes formas para verificar la integración de los componentes de la aplicación como pueden ser: APIs externas, bases de datos, flujo entre módulos o entre interfaces de usuario, etc. Para esta tarea se pueden implementar pruebas unitarias o pruebas automatizadas con Espresso.

CAPÍTULO III: METODOLOGÍA

La metodología bajo la que se desarrolló este proyecto es XP dentro del marco de desarrollo SCRUM [26][27].

Fase I: Identificación de las características y componentes relacionados con la navegación y gestión de encargos en Canadá y Estados Unidos. En esta fase se realizó una preparación inicial con el objetivo de conocer todos los aspectos que hacen parte del proyecto y las características asociadas a la funcionalidad de la aplicación.

1. Conocer los detalles del negocio de transporte escolar y transporte de desechos de diferentes tipos.
2. Conocer el estado actual de la aplicación: arquitectura, componentes, tecnologías y funcionalidades implementadas hasta el momento.
3. Analizar los requisitos funcionales establecidos para el desarrollo de la aplicación.

Fase II: Investigación y prueba de librerías y componentes de arquitectura. En esta fase se investigaron las librerías y componentes de arquitectura pertinentes para el desarrollo del proyecto.

1. Investigar las ventajas y características que ofrece la utilización del lenguaje de programación Kotlin para el desarrollo de aplicaciones Android.
2. Probar los diferentes componentes de arquitectura de Google para el desarrollo de aplicaciones Android.
3. Indagar las librerías más utilizadas y estandarizadas en el manejo de operaciones sincrónicas y asincrónicas para la gestión de hilos en Android.
4. Conocer el patrón MVVM y la arquitectura CLEAN implementadas en el desarrollo de aplicaciones Android.
5. Aprender las estrategias de versionamiento, ramificación y tecnologías de integración y entrega continua implementadas en la empresa.

Fase III: Desarrollo y descripción de las tareas de programación e implementación de tecnologías, librerías y arquitecturas. En esta fase se describe detalladamente la implementación de las funcionalidades, librerías y arquitecturas utilizadas en el desarrollo de la aplicación.

1. Explicar las arquitecturas utilizadas para el desarrollo de la aplicación.
2. Describir los componentes de arquitectura seleccionados.
3. Explicar las funcionalidades asignadas desarrolladas en cada sprint.

Fase IV: Validación de las funcionalidades implementadas en el desarrollo a través de un caso de estudio realizado en la ciudad de Medellín. En esta fase se determinó y se realizó las pruebas para validar el correcto comportamiento de las funcionalidades desarrolladas en el transcurso de la pasantía.

1. Determinar una ruta en la ciudad de Medellín que sea idónea para el caso de estudio.
2. Diseñar las pruebas específicas para cada funcionalidad desarrollada.
3. Realizar el caso de estudio y verificar el funcionamiento de cada funcionalidad.
4. Documentar el proceso llevado en el caso de estudio

CAPÍTULO IV: DEFINICIÓN DE LOS REQUERIMIENTOS

4.1 Características del negocio

Según la información adquirida y analizada del negocio de transporte escolar y transporte de diferentes tipos de desechos, basándose en las necesidades del cliente (SafeFleet) para el que se desarrolló este proyecto, se identificaron las siguientes características:

4.1.1 Características asociadas a la navegación

Consulta de rutas: el conductor en los dos negocios puede tener asignadas múltiples rutas cada día, por lo cual debe ser capaz de consultar las rutas que le corresponden y acceder a ellas cuando lo necesite, ya sea antes o en el momento que desee iniciar el viaje.

Trazado de rutas: una forma de facilitar el trabajo del conductor, en el caso del transporte escolar, es trazar una ruta óptima para recoger a los estudiantes y llevarlos a su respectiva institución educativa. Por otra parte, en el caso del transporte de desechos, el trazado de ruta es opcional ya que el conductor puede decidir si quiere ver la ruta trazada en el mapa o no.

Seguimiento de vehículos: es necesario recoger y enviar datos de GPS de manera periódica con el objetivo de utilizarlos en otras plataformas de la empresa para el conocimiento de sus clientes, como padres de familia que necesitan conocer el momento en que sus hijos deben estar en la parada de bus o agentes de la empresa que llevan a cabo el proceso de control de los vehículos de desechos.

Instrucciones de navegación: otra forma de facilitar el trabajo del conductor que aplica para los dos negocios, son las instrucciones de navegación, las cuales consisten en ayudas audiovisuales, que indican a donde debe dirigirse el vehículo a continuación. De nuevo estas instrucciones son opcionales para el transporte de desechos.

Estado del vehículo: además es necesario para el control de los vehículos de transporte escolar conocer cuando el vehículo se detiene en una parada para recoger estudiantes, esto se conoce a través del estado del brazo mecánico de parada que se despliega en las paradas de bus y se repliega antes de que el bus se ponga de nuevo en marcha.

4.1.2 Características asociadas a la gestión de encargos

Puntos de parada y destino: es necesario para los dos negocios que el conductor conozca las respectivas paradas y puntos de encargo, así como sus destinos finales, por lo que se deben mostrar en el mapa con su debida numeración o diferenciación como marcador, es decir, si es una parada o un destino.

Estado de los encargos: también es importante para SafeFleet y sus clientes conocer el estado de sus encargos. En el caso del transporte escolar, se debe conocer cuando un estudiante es recogido en una parada y cuando se baja del bus escolar, y en el caso del transporte de desechos, se debe conocer cuando el vehículo recoge el contenido de un contenedor y cuando se descarga en un depósito. Esto se lleva a cabo con la ayuda de dispositivos RFID (Radio-Frequency Identification).

4.2 Requerimientos funcionales

Tabla 1. Requerimientos funcionales

Código	Nombre del requerimiento	Descripción
RF-01	Barra de búsqueda personalizada	Implementar un control no nativo que funciona como una barra de búsqueda desplegable con sus respectivas animaciones, iconos y comportamientos, además de un botón opcional

		para guardar rutas que no aparezcan en la búsqueda.
RF-02	Guardar rutas	Obtener la ruta inexistente de la barra de búsqueda y guardarla en la base de datos de la aplicación.
RF-03	Observar el estado de conexión a Internet	Implementar una clase que permita observar si el dispositivo se encuentra conectado a Internet para las diferentes funcionalidades de las aplicaciones.
RF-04	Observar el estado de conexión al servidor	Implementar una clase que permita observar el estado de los servidores frecuentemente con un intervalo de 10 segundos entre cada actualización de estado.
RF-05	Notificar el estado de conexión al servidor	Mostrar un icono parpadeante en todas las pantallas de la aplicación que indique si alguno de los servidores no se encuentra disponible ya sea por falta de conexión a internet o por un error interno (50X)
RF-06	Observar el estado del brazo mecánico de parada del bus escolar	Implementar una clase que permita enviar comandos de consola para: <ul style="list-style-type: none"> • Configurar los estados de los GPIOs de la estación de acoplamiento del dispositivo con el bus. • Leer los cambios en el voltaje para conocer si el brazo mecánico se encuentra desplegado o recogido y exponerlos en una propiedad observable.

RF-07	Servicio de manejo y envío de eventos	Implementar un servicio que permita manejar los eventos de GPS, RFID y brazo mecánico de parada teniendo en cuenta: <ul style="list-style-type: none"> • Si hay conexión a Internet enviarlos al servidor procesador de datos del dispositivo. • Si no hay conexión a Internet guardarlos en la base de datos de la aplicación. • Enviar los eventos guardados en la base de datos de la aplicación al servidor cuando haya conexión a Internet.
RF-08	Pantalla de navegación de cada negocio	Refactorizar la lógica de la pantalla de navegación para tener un mejor manejo de cada negocio y mantener el código compartido entre ambos.

Fuente propia

4.3 Requerimientos no funcionales

Tabla 2. Requerimientos no funcionales

Código	Nombre del requerimiento	Descripción
RNF-01	Versionamiento de código con Git	Manejar estrategias de versionamiento de código con Git para los diferentes repositorios del proyecto en Bitbucket.
RNF-02	Automatización de tareas de integración y entrega continua	Utilizar la herramienta Jenkins para automatizar tareas de verificación de funcionamiento y calidad del proyecto.
RNF-03	Kotlin como lenguaje de programación	Utilizar Kotlin (recomendado por Google) como lenguaje de programación para el proyecto.

RNF-04	Buenas prácticas de programación	Implementar buenas prácticas y principios de programación orientada a objetos para garantizar la calidad del código.
RNF-05	Arquitectura y patrón de arquitectura	Investigar, seleccionar e implementar una arquitectura y patrones arquitectura adecuados para el proyecto y para garantizar su calidad.
RNF-06	Librerías recomendadas y soportadas	Investigar, seleccionar y utilizar librerías recomendadas por Google y por la comunidad que sean de calidad y que reciban soporte continuo.
RNF-07	Pruebas unitarias	Implementar pruebas unitarias de cada funcionalidad para garantizar su correcto funcionamiento y comportamientos esperados.
RNF-08	Reportes de cobertura y calidad	Implementar reportes de cobertura de clases y funciones por pruebas unitarias y verificación de calidad de código.

Fuente propia

CAPÍTULO V: DESARROLLO

5.1 Resumen del trabajo realizado

A continuación, se presenta un diagrama guía que resume el trabajo realizado en esta pasantía y que será explicado durante el transcurso de este capítulo:

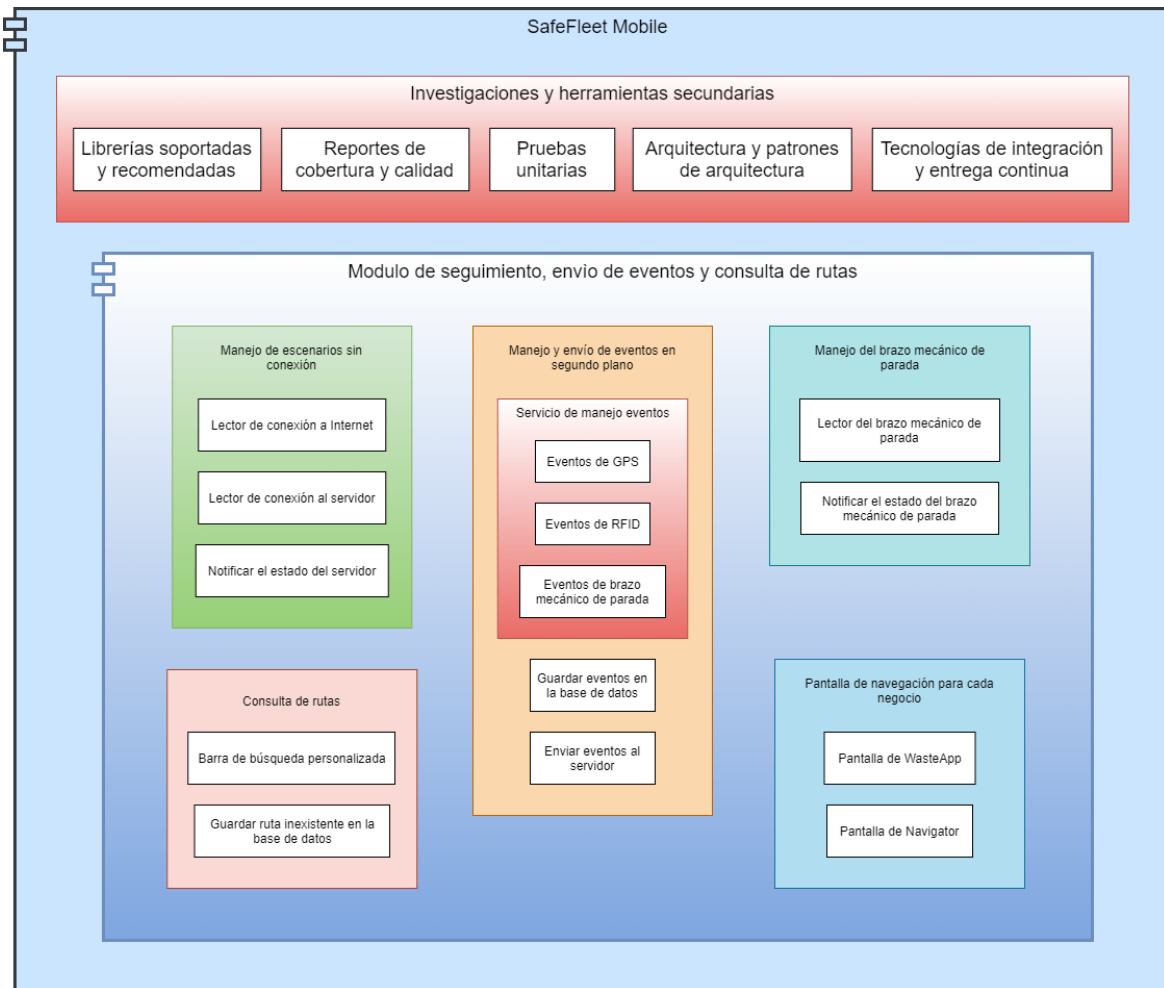


Figura 5. Resumen del trabajo realizado para SafeFleet Mobile. Fuente propia

5.2 Investigación de frameworks, librerías y herramientas

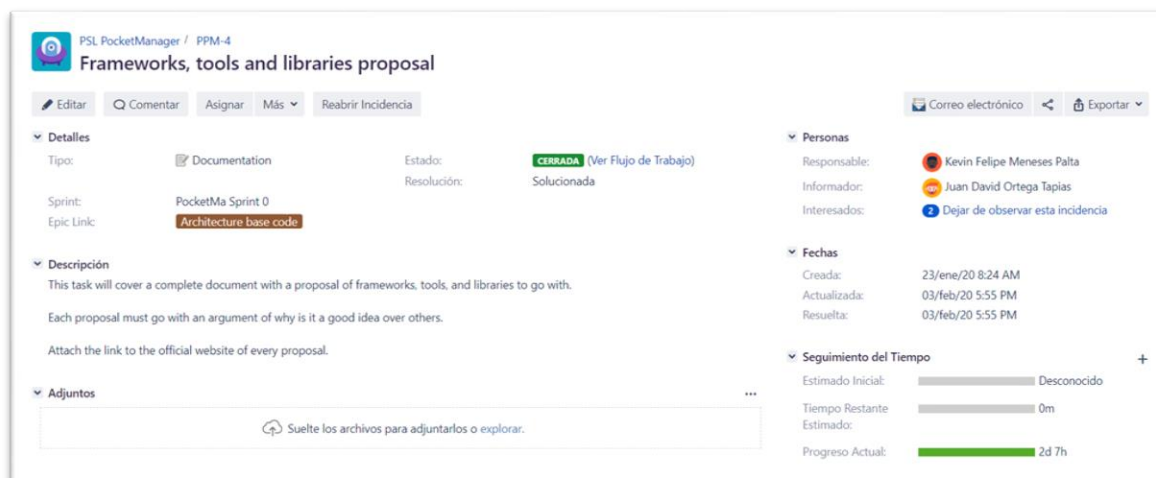


Figura 6. Ticket de Jira PPM-4. Fuente: Jira de PSL

Descripción del trabajo: esta tarea consistió en la investigación de diferentes librerías útiles para el desarrollo de aplicaciones Android y pertinentes para el proyecto. Además, con los resultados se realizó una prueba de concepto, junto con las arquitecturas recomendadas por Google, para evaluar su uso de manera más profunda y presentar los resultados obtenidos en un informe.

Descripción de las librerías: se procuró investigar las librerías más populares usadas por los desarrolladores de aplicaciones Android y las recomendadas como componentes de arquitectura por Google, las cuales se clasificaron en las siguientes categorías: peticiones HTTP, manejo de base de datos, procesos asíncronos y eventos, inyección de dependencias, descarga y manejo de imágenes, análisis y manejo de información JSON, manejo de *RecyclerView* y adaptadores, pruebas unitarias y de instrumentación y otras herramientas y funcionalidades.

5.2.1 Peticiones HTTP

Las librerías seleccionadas en esta categoría se encargan de manejar las peticiones a diferentes tipos de API's web como SOAP o REST, entre otras. Algunas manejan menor o mayor integración con otras librerías como, por ejemplo: análisis

y conversión de información JSON y manejo de operaciones asíncronas. A continuación, se presenta cada una de ellas:

Asynchronous HTTP Client: es una librería que proporciona un cliente Http asíncrono basado en devolución de llamada para Android construido sobre las bibliotecas HttpClient de Apache [47].

Características:

- Puede realizar solicitudes fuera del hilo principal
- Capacidad de devolver llamadas en el hilo de la clase que creó la lógica de devolución
- Reconocimiento automático del contexto en que se creó (servicio, subproceso)
- Usado en producción por Instagram y Pinterest

Ventajas:

- Contiene funciones que manejan el estado de las peticiones
- Capacidad de guardar cookies
- Reintentos automáticos de solicitudes inteligentes optimizadas para conexiones móviles irregulares

Desventajas: no se asegura su soporte en el tiempo debido a que es manejada por terceros.

ION: es una librería de carga de imágenes y redes asincrónicas de Android [48].

Características:

- Maneja integración y soporte para corrutinas en Kotlin
- Permite realizar descargas asíncronas tanto de imágenes como de información en formato JSON

- Está integrado con la librería Future de Java y todas sus operaciones devuelven un objeto de tipo *Future* para realizar *callbacks* cuando termina la petición y además se pueden cancelar
- Gestiona la invocación nuevamente en el hilo principal

Ventajas:

- Maneja múltiples peticiones
- Soporta corrutinas
- Cuenta con funciones que manejan el estado de las peticiones

Desventajas: no se asegura su soporte en el tiempo debido a que es manejada por terceros

Retrofit: es una librería que proporciona un cliente HTTP con seguridad de tipos para Android y Java [49].

Características:

- Convierte su API HTTP en una interfaz
- Genera una implementación de esa interfaz
- Permite solicitudes HTTP síncronas o asíncronas
- Anotaciones para describir la solicitud HTTP
- Permite el uso de conversores para la información recibida
- Soporte para el uso de módulo RxJava

Ventajas:

- Soporte en el tiempo al ser Square una empresa de desarrollo
- Recomendada por Google
- Usa anotaciones

Desventajas:

- No tiene manejo de múltiples peticiones
- No tiene manejo de caché

Volley: es una biblioteca HTTP que facilita y agiliza el uso de redes para aplicaciones de Android [50].

Características:

- Programación automática de solicitudes de red
- Varias conexiones de red simultáneas
- Compatibilidad con la priorización de solicitudes
- Permite cancelar una única solicitud o grupos de solicitudes establecidos
- Incluye descarga y carga de imágenes

Ventajas:

- Usa caché
- Tiene reglas de reintento

Desventajas: necesita código adicional para integrar las librerías de conversión de información tipo JSON

5.2.2 Manejo de base de datos

Las librerías seleccionadas en esta categoría sirven para manejar las bases de datos internas de una aplicación Android, incluyendo consultas y transacciones en las mismas. Algunas pueden soportar integración con otras librerías encargadas del manejo de operaciones asíncronas y las clases que utilizan.

Realm: es una base de datos móvil que se ejecuta directamente en teléfonos, tabletas o dispositivos portátiles [51].

Características:

- Es la primera base de datos móvil creada desde cero
- Es simple, ya que los datos se exponen directamente como objetos
- Admite seguridad de hilos, relaciones y cifrado fáciles
- Es más rápido que SQLite en operaciones comunes

Ventajas:

- No necesita ORM, no usa consultas SQL
- Maneja transacciones síncronas y asíncronas

Desventajas:

- No puede retornar objetos de tipo LiveData directamente
- Aumenta el tamaño de APK, usa más ram y tiempo de construcción
- No devuelve modelos POJO hay que hacerlo manualmente
- No puede hacer JOIN de múltiples tablas
- Es menos confiable ya que puede generar problemas si el desarrollador olvida cerrar la base de datos.

Room: es una librería proporciona una capa de abstracción sobre SQLite que permite acceder a la base de datos sin problemas y, al mismo tiempo, aprovechar toda la potencia de SQLite [52].

Características:

- Permite crear una memoria caché de los datos en un dispositivo que ejecute tu aplicación
- Maneja una interfaz para el acceso a la base de datos y genera su implementación
- Utiliza anotaciones para definir sus componentes

Ventajas:

- Utiliza SQLite que es la base de datos por defecto de Android
- Soporta corrutinas y puede devolver clases como, por ejemplo: Publisher, Flowable, Observable y LiveData
- Recomendado por Google
- Capacidad para retornar modelos POJO
- Facilidad para ser probado en pruebas unitarias

Desventajas:

- No maneja hilos por sí misma
- Requiere un nivel de configuración
- Utiliza consultas SQL

5.2.3 Procesos asíncronos y eventos

Las librerías seleccionadas en esta categoría se encargan de manejar los diferentes hilos en los que se ejecutan operaciones como peticiones a servidores, consultas a bases de datos y otros procesos que tengan alta carga computacional, con el fin de evitar bloquear el hilo principal de las aplicaciones y mejorar la experiencia de usuario.

Corrutinas de Kotlin: es una rica biblioteca para corrutinas desarrollada por JetBrains [53][54].

Características:

- Administran tareas prolongadas que podrían bloquear el hilo principal.
- Ejecutan de manera segura operaciones de red o disco desde el hilo principal.
- Crea hilos livianos, que no son realmente hilos, pero pueden existir varios por hilo real.

Ventajas:

- Hecho para el lenguaje de programación Kotlin
- Permite gestionar diferentes hilos
- Facilidad de aprendizaje e implementación
- Integración con múltiples librerías parte de los componentes de arquitectura de Google

Desventajas: puede bloquear el hilo principal si no se maneja adecuadamente.

EventBus: es una biblioteca de código abierto para Android y Java que utiliza el patrón editor/suscriptor para un acoplamiento flexible [55].

Características:

- Permite la comunicación central para desacoplar clases con pocas líneas de código
- Se basa en anotaciones

Ventajas:

- Desacopla los remitentes y receptores de eventos
- Evita dependencias complejas y propensas a errores y problemas del ciclo de vida
- Fácil de aprender

Desventajas: permite la creación directa e indirecta de eventos anidados lo que aumenta la complejidad y dificulta la depuración

RxJava: es una biblioteca para componer programas asíncronos y basados en eventos mediante el uso de secuencias observables [56].

Características:

- Maneja tiempo virtual y planificadores para concurrencia parametrizada
- Utiliza clases base para las diferentes concurrencias de la información manejada (Flowable, Observable, Completable, Single y Maybe)

Ventajas:

- Ejecución y manejo de operaciones síncronas o asíncronas
- Suscripción a cambios en la información dentro de sus clases base

Desventajas:

- Nivel de complejidad alto-medio

- No está integrada para manejar los ciclos de vida de las aplicaciones Android por si misma

5.2.4 Inyección de dependencias

Las librerías seleccionadas en esta categoría se encargan de crear y proveer a cada clase con las dependencias necesarias para facilitar el desacoplamiento entre clases y módulos y la testeabilidad de las mismas.

AndroidAnnotations: es un framework de código abierto que acelera el desarrollo de Android. Se encarga de las tuberías y le permite concentrarse en lo que es realmente importante. Simplificando el código, lo que facilita su mantenimiento [57].

Características:

- Permite inyectar vistas, extras, servicios del sistema, recursos, entre otros.
- Manejo de hilos simplificado
- Maneja enlace de eventos y *listeners*
- Cuenta con cliente de API REST
- Genera subclases en tiempo de compilación

Ventajas:

- Nivel bajo de complejidad
- Maneja hilos
- Maneja eventos

Desventajas:

- No se asegura su soporte en el tiempo ya que es manejada por terceros
- Menos potente que otras librerías componentes de arquitectura de Google

Dagger Android: es un framework que se ejecuta en tiempo de compilación para la inyección de dependencias. No utiliza reflexión ni generación de código de bytes en tiempo de ejecución, realiza todo su análisis en tiempo de compilación y genera código fuente simple de Java [58].

Características:

- Es un inyector rápido de dependencias
- Utiliza anotaciones
- Realiza todo su análisis en tiempo de compilación

Ventajas:

- Robusto
- Parte de los componentes de arquitectura de Google
- Libera de escribir código repetitivo tedioso y propenso a errores
- Permite la creación de flujos (alcances de cada instancia)
- Es mejor para proyectos grandes

Desventajas: alto nivel de complejidad

Spork: es una biblioteca de procesamiento de anotaciones en tiempo de ejecución rápido [59].

Características:

- Es una biblioteca de procesamiento de anotaciones e inyección de dependencias en tiempo de ejecución
- Se enfoca en la inyección de vistas

Ventajas: bajo nivel de complejidad

Desventajas:

- No se asegura su soporte en el tiempo al ser una librería de terceros
- Existe Syntetic para Kotlin para la instanciación de vistas

KOIN: es un framework de inyección de dependencia ligera para desarrolladores de Kotlin. Escrito en Kotlin puro usando solo resolución funcional: sin proxy, sin generación de código, sin reflexión [60][61].

Características:

- Se puede inyectar en Android
- Soporta inyección de clases de tipo ViewModel
- Se puede usar en pruebas

Ventajas:

- Bajo nivel de complejidad
- Es mejor para proyectos pequeños
- Escrito en Kotlin

Desventajas:

- Es un localizador de servicios
- No es tan eficiente en proyectos grandes

5.2.5 Descarga y manejo de imágenes

Las librerías seleccionadas en esta categoría se encargan de recibir imágenes a través de direcciones URL para después mostrarlas en las vistas que las necesiten dentro de la aplicación.

Glide: es un framework de carga de imágenes y administración de medios de código abierto rápido y eficiente para Android que incluye la decodificación de medios, el almacenamiento en caché de disco y memoria y la agrupación de recursos en una interfaz simple y fácil de usar [62].

Características:

- Permite cargar imágenes en vistas
- Guarda la imagen original y una copia del tamaño de la vista para rapidez de carga.

Ventajas:

- Diseñada para funcionar con el ciclo de vida de las aplicaciones Android
- Menor consumo de memoria al cargar imágenes

- Es más rápido cargando imágenes guardadas previamente en caché
- Admite el uso de GIF
- Permite usar miniaturas

Desventajas: tres veces más pesada que otras librerías

Picasso: una potente biblioteca de descarga y almacenamiento en caché de imágenes para Android [63].

Características:

- Permite cargar imágenes en vistas
- Guarda la imagen original y la ajusta tamaño de la vista

Ventajas: es más rápido cargando la imagen en la vista por primera vez

Desventajas: puede causar errores de memoria

5.2.6 Análisis y manejo de información JSON

Las librerías seleccionadas en esta categoría se encargan de analizar y convertir los datos tipo JSON a modelos POJOS dentro de la aplicación y también en sentido contrario.

GSON: es una biblioteca de Java que se puede utilizar para convertir objetos Java en su representación JSON. También se puede utilizar para convertir una cadena JSON en un objeto Java equivalente [64].

Características:

- Convierte objetos Java en su representación JSON y viceversa
- Proporciona métodos simples para la conversión
- Puede trabajar con objetos Java arbitrarios, incluidos objetos preexistentes de los que no tiene código fuente

Ventajas:

- No necesita de anotaciones en las clases (POJO).

- Tiene varios adaptadores incorporados.

Desventajas: está deprecado según los desarrolladores que le dan soporte

Moshi: es una biblioteca JSON moderna para Android y Java. Facilita el análisis de JSON en objetos Java [65].

Características:

- Facilita la conversión de JSON en objetos Java/Kotlin
- Permite crear adaptadores personalizados
- Tiene menos adaptadores incorporado.

Ventajas:

- Entiende los tipos anulables de Kotlin y los valores de parámetros predeterminados
- Contiene anotadores como `@HexColor` que permite múltiples representaciones JSON para un solo tipo de Java
- Evita la decodificación UTF-8 innecesaria

Desventajas:

- Menos configurable
- Pide el mismo nombre de los campos o anotaciones que lo especifiquen

5.2.7 Manejo de RecyclerView y adaptadores

Las librerías seleccionadas en esta categoría se encargan de facilitar el manejo de vistas de tipo *RecyclerView* y/o de sus adaptadores además de agregar funcionalidades útiles que mejoran la experiencia de usuario.

Epoxy: es una biblioteca de Android para construir pantallas complejas en un *RecyclerView* [66]. Desarrollada por Airbnb.

Características:

- Los modelos se generan automáticamente a partir de vistas personalizadas

- Cada elemento de la lista está representado por un modelo
- Es similar a cómo React aborda las interfaces de usuario en JavaScript

Ventajas:

- Tiempos de diseño más rápidos
- Menos uso de memoria
- Una transición más fluida

Desventajas: menos configurable.

FastAdapter: es una librería que simplifica la creación de adaptadores para RecyclerView [67]. Desarrollada por Mike Penz.

Características:

- Su módulo central está hecho 100% en Kotlin
- Maneja Click / Long-click listeners
- Maneja elementos de selección múltiple, arrastrar y soltar, encabezados
- Elementos expandibles, desplazamiento sin fin, paginación
- Vista y modelo de elemento dividido

Ventajas: simplifica la creación de adaptadores para List y RecyclerView.

Desventajas: menos configurable.

UltimateRecyclerView: es un RecyclerView con capacidad para actualizar, cargar más, deslizar para descartar, arrastrar y soltar, animaciones, encabezado fijo, mostrar u ocultar barra de herramientas y FAB cuando se desplaza y muchas otras funciones [68]. Desarrollada por Marshal Chen.

Características:

- Desliza para actualizar o para descartar
- Muchos tipos de animaciones
- Arrastra y suelta elementos
- Desplazamiento infinito

- Mostrar u ocultar la barra de herramientas y el botón flotante al desplazarse
- Estilos coloridos de "Swipe to refresh"

Ventajas: hace más dinámico el diseño.

Desventajas: puede requerir cambios en el adaptador.

5.2.8 Pruebas unitarias y de instrumentación

Las librerías seleccionadas en esta categoría se encargan de facilitar el desarrollo de pruebas unitarias y de instrumentación para las diferentes funcionalidades de una aplicación Android.

Barista: es una librería usada sobre Espresso para facilitar el desarrollo de pruebas y reducir el código repetitivo [69]. Desarrollada por Adevinta Spain.

Características:

- Provee una API simple y entendible
- Provee métodos para ejecutar acciones y aserciones sobre diferentes tipos de vistas de manera rápida

Ventajas:

- Sus métodos son intuitivos y fáciles de utilizar
- Reduce el código repetitivo

Desventajas: no funciona con vistas personalizadas con callbacks personalizados.

Espresso: es un framework utilizado para realizar pruebas de interfaz de usuario en Android [70]. Desarrollado por Google.

Características:

- Pruebas concisas, eficaces y confiables
- Capacidades de sincronización

- Cuenta con paquetes con utilidades para diferentes tipos de vistas y para la sincronización con trabajos en segundo plano

Ventajas:

- La API principal es pequeña, predecible y fácil de aprender
- Expone claramente las expectativas, las interacciones y las afirmaciones sin la distracción del contenido estándar
- Se ejecuta con una rapidez óptima. Permite dejar atrás esperas, sincronizaciones, tiempos inactivos y sondeos

Desventajas:

- El manejo de mensajes de errores es poco claro y no lleva a la solución de manera intuitiva
- Aparición de errores desconocidos
- Algunas veces las pruebas fallan por falta de memoria o sin razón aparente

Junit5: es un framework popular para la implementación de pruebas unitarias en Java [71]. Desarrollado por Junit Team

Características:

- Permite crear clases y métodos de tipo prueba a través de anotaciones
- Provee anotaciones para la implementación de pruebas descriptivas que se integran con la interfaz de Android Studio

Ventajas:

- Tiene baja complejidad y fácil aprendizaje
- Cuenta con gran cantidad y variedad de métodos para realizar aserciones sobre resultados

Desventajas: ninguna.

Mockk: es una librería para objetos ficticios de clases hecha para el lenguaje Kotlin [72]. Desarrollada por Mockk.io

Características:

- Provee clases para objetos ficticios de la mayoría de las clases que se necesitan en una prueba unitaria
- Permite crear impostores de clases propias del lenguaje Kotlin
- Provee clases que permiten controlar el retorno de una o varias funciones
- Provee funciones de verificación de llamado de métodos

Ventajas:

- Está creada propiamente para el lenguaje Kotlin
- Tiene funciones propias que soportan corrutinas
- Soporta diferentes métodos y tipos de retornos para casos específicos

Desventajas: ninguna.

Robolectric: es un framework que ofrece pruebas unitarias rápidas y confiables para Android [73]. Desarrollado por Robolectric.

Características: ejecuta pruebas dentro de un entorno limitado que permite configurar el entorno de Android con precisión.

Ventajas:

- Permite probar eventos en la UI
- Permite hacer pruebas no basadas en emulador
- No requiere frameworks para dobles de pruebas

Desventajas:

- Sus pruebas son más lentas que las hechas puramente en Java
- Su configuración es frágil y puede tomar tiempo arreglarlo
- Es más complejo a largo plazo

5.2.9 Otras herramientas y librerías experimentales

Las librerías seleccionadas en esta categoría ofrecen funcionalidades útiles o experimentales que pueden agregar valor al desarrollo de una aplicación Android.

Android KTX modules: es un conjunto de extensiones de Kotlin que se incluyen con Android Jetpack y facilita algunas funciones comunes de Android [74]. Desarrollado por Google.

Características:

- Extensiones para fragmentos
- Extensiones para paletas de colores
- Extensiones para colecciones
- Extensiones para ViewModel
- Extensiones para ReactiveStreams (LiveData, RxJava)
- Extensiones para WorkManager (para soportar corrutinas)

Ventajas:

- Aprovecha varias funciones del lenguaje Kotlin
- Provee funciones de extensión para manejar listas, conversión de tipos de variables y ciclos de diferentes tipos, entre otras, de manera fácil y rápida
- Contiene nuevas clases que facilitan algunas tareas en Android y soportan corrutinas

Compose: es un kit de herramientas moderno para crear IU nativas de Android [75]. Desarrollado por Google.

Características:

- Permite crear interfaces con Kotlin sin depender de recursos XML
- Permite manejar el comportamiento y estado de sus interfaces desde Kotlin
- Hace parte de la arquitectura JetPack

Ventajas:

- Mejora el control del comportamiento de la interfaz y su dinamismo
- Genera menos código y acelera el desarrollo
- Potente e intuitivo

Desventajas:

- Puede generar acoplamiento entre la vista y los controladores
- No cuenta con suficiente documentación en el momento
- Requiere de un tiempo considerable para el aprendizaje de sus diferentes clases y métodos

Chucker: es un Interceptor de la librería OkHttp que persiste en todos los eventos HTTP dentro de una aplicación [76]. Desarrollado por Chucker Team.

Características:

- Está desarrollado en Kotlin
- Permite inspeccionar solicitudes y respuestas HTTP y errores de la aplicación
- Se ejecuta junto con la aplicación como utilidad para hacer depuración
- Es un servicio que se corre en segundo plano

Ventajas:

- Es fácil de integrar, sólo requiere dos líneas de implementación en Gradle
- No deja rastros en el archivo de lanzamiento
- Soporta el modo multiventana

Desventajas: está creado sobre una librería llamada Chuck que ya no recibe soporte y no se puede asegurar que su equipo actual continúe brindando soporte en el tiempo.

LeakCanary: es una librería de detección de fugas de memoria para Android [77] desarrollada por Square.

Características:

- Se ejecuta junto con la aplicación como utilidad para hacer depuración
- Es un servicio que se corre en segundo plano

Ventajas:

- Es fácil de integrar, sólo requiere una línea de implementación en Gradle
- Ayuda a los desarrolladores a reducir los errores por quedarse sin capacidad de memoria
- No deja rastros en el archivo de lanzamiento

Desventajas: existe una utilidad de Android Studio llamada Profiler que cumple esta función.

Navigation Component: es una librería que facilita la configuración del flujo de navegación entre fragmentos [78] desarrollada por Google.

Características: está compuesto por los siguientes 3 elementos:

- NavGraph: un gráfico de navegación que es un recurso XML con toda la información relacionada con la navegación
- NavHost: un anfitrión de navegación que es un contenedor con los destinos del gráfico de navegación
- NavController: un controlador de navegación que es un objeto que administra la navegación de la aplicación dentro de un anfitrión de navegación

Ventajas:

- Permite implementar la navegación, desde simples clics de botones hasta patrones más complejos, como las barras de herramientas y los paneles laterales de navegación
- Garantiza una experiencia del usuario coherente y predecible, ya que se adhiere a un sistema establecido de conjunto de principios
- Hace parte de la arquitectura JetPack

Desventajas:

- Manejar una aplicación completamente a través de fragmentos tiene algunas implicaciones en el uso de herramientas de análisis
- Depende del framework de Android, lo que impide desacoplar la lógica de navegación la cual puede considerarse parte del controlador o ViewModel

5.3 Prueba de concepto de la investigación

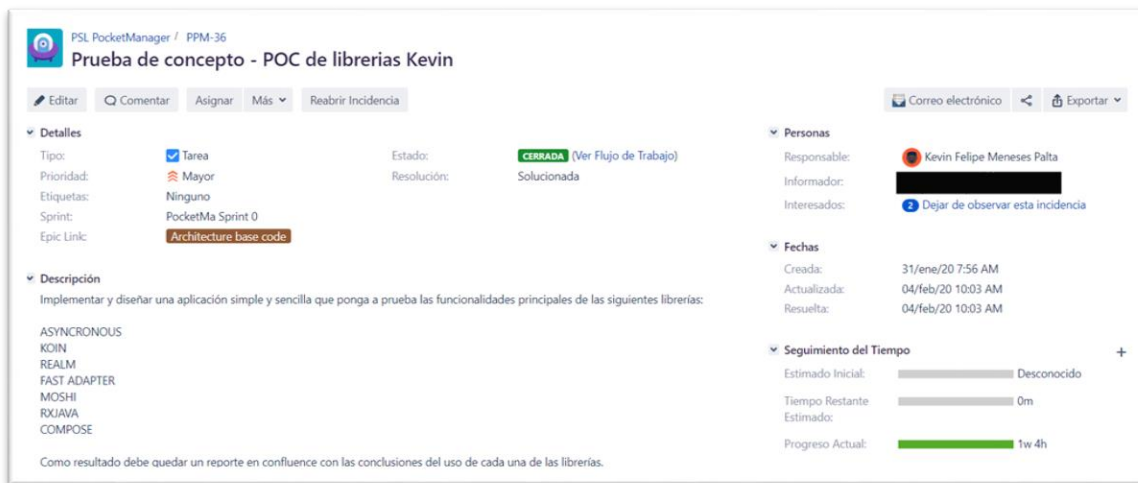


Figura 7. Ticket de Jira PPM-36. Fuente: Jira de PSL

Descripción de la prueba: se realizó una prueba con algunas de las librerías descritas anteriormente creando un proyecto corto para contrastar los resultados de la investigación con los resultados de su implementación y alcanzar conclusiones que resulten útiles para el resto de los desarrolladores Android de los diferentes proyectos actuales y futuros de la empresa.

- **Diseño de IU:** se utilizó la librería Compose para realizar el diseño de la interfaz de usuario para cuyo momento se encontraba en fase experimental por lo que fue necesario utilizar Android Studio Canary la versión 4.0 que incluye esta librería por defecto.
- **Peticiones HTTP:** con el fin de utilizar la librería Asynchronous Http Client se seleccionó un servicio API REST que consumiría la aplicación.

- **Manejo de hilos y eventos asíncronos:** se utilizó la librería RxJava y corrutinas de Kotlin para el manejo de los eventos asíncronos con el fin de compararlas.
- **Manejo de bases de datos:** se utilizó la librería Realm para la creación de la base de datos y el manejo de sus diferentes transacciones.
- **Inyección de dependencias:** se seleccionó la librería KOIN para manejar la inyección de dependencias de la aplicación.

La prueba de concepto desarrollada consiste en una aplicación que hace peticiones de tipo GET a un endpoint llamado <https://catfact.ninja/facts> y obtiene un objeto JSON con un texto aleatorio que contiene información sobre gatos. El usuario puede usar un botón para guardar la información como un objeto de una entidad en la base de datos y otro botón para visualizar la lista de objetos guardada en la base de datos.

5.3.1 Evidencias y funcionamiento

Pantalla de inicio: presenta una interfaz que solicita ingresar el nombre y la edad del usuario con el simple objetivo de mostrar un saludo. Contiene un botón para iniciar la actividad de bienvenida enviando los datos ingresados por el usuario.

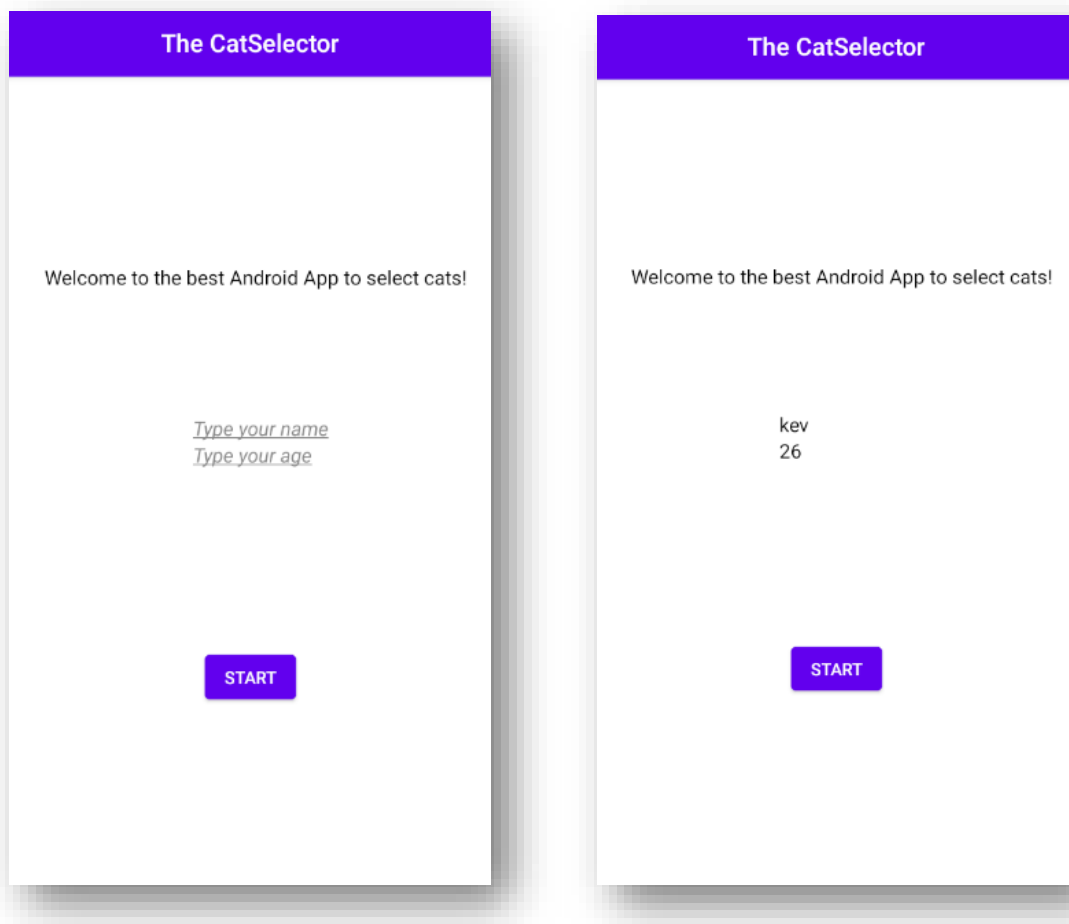


Figura 8. Pantalla de inicio de la aplicación The CatSelector. Fuente propia

Pantalla de bienvenida: muestra un mensaje de bienvenida al usuario con que incluye los datos solicitados en la pantalla de inicio. También realiza la petición al endpoint del servidor seleccionado y muestra el mensaje obtenido remotamente. Contiene un botón para guardar el dato obtenido en la base de datos otro botón que

muestra la lista de datos guardados. Muestra una alerta de tipo Toast que muestra si el resultado de la operación en base de datos fue exitosa o fallida.

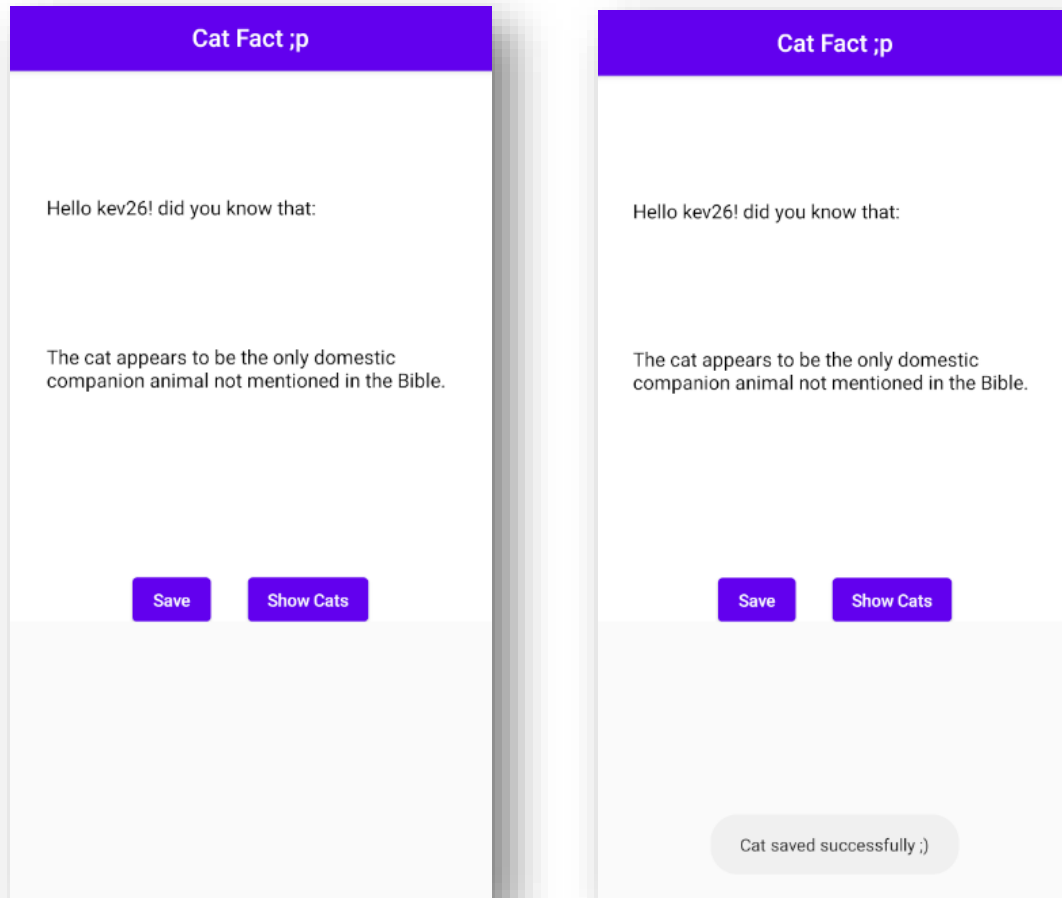


Figura 9. Pantalla de bienvenida de la aplicación The CatSelector. Fuente propia

Lista de información: obtiene todos los registros guardados en la base de datos y los muestra en una lista.

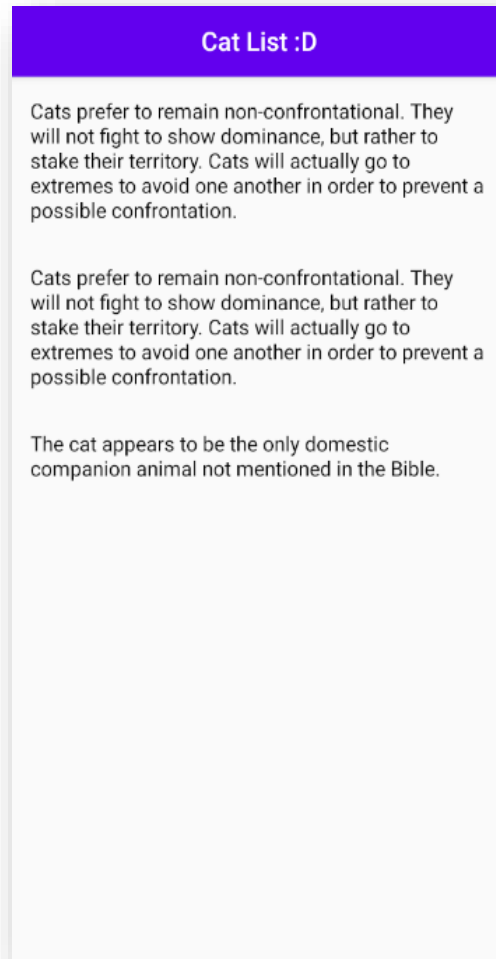


Figura 10. Lista de datos de la aplicación The CatSelector. Fuente propia

5.3.2 Conclusiones de la prueba:

JetPack Compose: es una librería prometedora en el desarrollo de interfaces de usuario para Android y proyectos multiplataforma de Kotlin, al utilizar únicamente Kotlin para su diseño y deshacerse de los recursos XML, sin embargo, se encuentra en versión Alpha, por lo que no debería ser usada en producción. Es una herramienta que vale la pena aprender para el momento en que sea lanzada a

producción, ya que apunta a ser el futuro del diseño de interfaces de usuario para aplicaciones desarrolladas con Kotlin.

Asynchronous Http Client: es una librería que tiene buen rendimiento y funcionalidades que en su momento fue utilizada en proyectos grandes, pero en la actualidad se encuentra deprecada pues ya no recibe soporte de sus desarrolladores. Existiendo soluciones más completas y mejor integradas como Volley o Retrofit que se han convertido en estándar para la creación de peticiones web.

RxJava versus Corrutinas de Kotlin: RxJava es una librería muy poderosa para el manejo de eventos asíncronos utilizada ampliamente en aplicaciones grandes y pequeñas desarrolladas en Java, sin embargo, tiene una curva de aprendizaje exponencial debido a la cantidad de funciones y operadores con los que cuenta, los cuales toman un tiempo significativo para entenderlos y aprenderlos a usar.

Por otra parte, Kotlin soporta nativamente el uso de funciones suspendidas junto con su librería de corrutinas las cuales son rápidas, fáciles de entender, y permiten manejar diferentes alcances y contextos en los cuales ejecutamos nuestro código de manera asíncrona en un hilo secundario para devolver sus resultados en el hilo principal. La facilidad, rapidez y robustez de las corrutinas muestran la ventaja, aunque ambas librerías tienen un excelente rendimiento.

Realm: su implementación es fácil de realizar, sus métodos son fáciles de entender, pero es más pesada que utilizar SQLite o la librería Room, agrega entre 2 y 4 megabytes a la aplicación. Además, tiene la limitación de no soportar su ejecución en múltiples hilos por lo que se debe tener cuidado al crear instancias de la base de datos y siempre cerrarlas. Otro contra es el manejo de relaciones entre tablas y consultas anidadas porque se realizan a través de métodos que se diferencian un poco de lo que se está acostumbrado a usar en consultas SQL. El elegir utilizar esta librería o no en un proyecto depende de sus necesidades.

KOIN: su implementación es realmente sencilla y rápida de entender, no requiere tantos archivos de configuración como Dagger, cuya curva de aprendizaje también es exponencial por la cantidad de archivos y de anotaciones con las que cuenta. Su única desventaja es que es un localizador de servicios. En proyectos pequeños, es una herramienta ideal, pero en proyectos más grandes con mayor cantidad de clases inyectadas su rendimiento disminuye respecto a otras librerías como Dagger que son más robustas.

5.4 Elementos propuestos para el desarrollo

Basados en la investigación realizada en los puntos anteriores, se decidió seguir las recomendaciones de Google y de la comunidad en la selección de librerías y patrones de arquitectura para cumplir con los requerimientos del proyecto. La arquitectura general del proyecto se escogió y se diseñó basados en el conocimiento aportado por Robert C. Martin en Clean Architecture [32].

5.4.1 Arquitectura general

Arquitectura CLEAN para proyectos Android modularizados.

Descripción de la arquitectura: la arquitectura implementada en esta aplicación Android nativa consta de cuatro módulos, cada uno con unas características y un propósito específico:

- **Aplicación:** se encarga de proveer las dependencias necesarias para la creación de instancias de cada clase a través de toda la aplicación, por lo que es capaz de ver todos los módulos internos y externos.
- **Presentación:** se encarga de las interfaces de usuario y las vistas de la aplicación. Contiene el framework de Android. Es capaz de ver el módulo de dominio para acceder a la lógica del negocio.
- **Dominio:** define la lógica de negocio de la aplicación como los casos de uso y los modelos de datos. Es visible para los módulos de presentación y de datos, pero no conoce nada de ellos.

- **Datos:** se encarga de proveer y manejar los datos que necesita la aplicación para su funcionamiento, ya sean provenientes de un servidor remoto o de la base de datos local. También se encarga de mapear los modelos de datos a modelos de negocio.

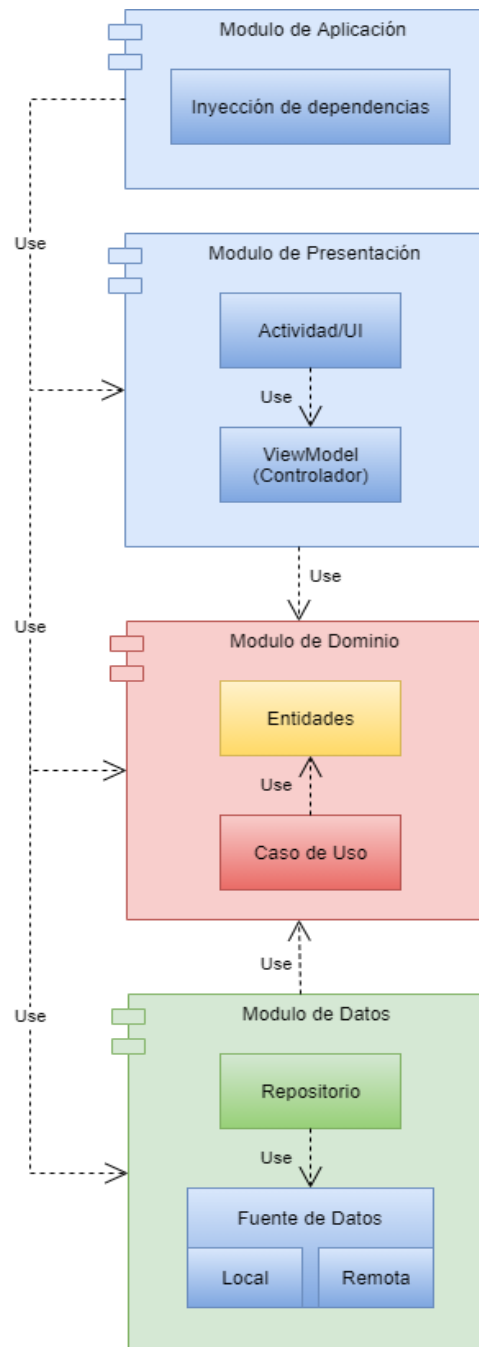


Figura 11. Diagrama de arquitectura de la aplicación. Fuente propia

5.4.2 Patrones de arquitectura

- Model – View – ViewModel (MVVM)
- Patrón de repositorio

5.4.3 Patrones de comportamiento

- Observador, usando LiveData
- Estado, usando interfaces e implementaciones o funciones de orden superior de Kotlin

5.4.4 Librerías seleccionadas

- Manejo de base de datos: Room de Google
- Peticiones HTTP: Retrofit de Square
- Manejo de operaciones asíncronas: Corrutinas de Kotlin
- Inyección de dependencias: Dagger Android de Google y Square
- Pruebas unitarias: JUnit5 y MockK
- Reportes de calidad: JaCoCo y SonarQube

5.5 Reportes de calidad del proyecto

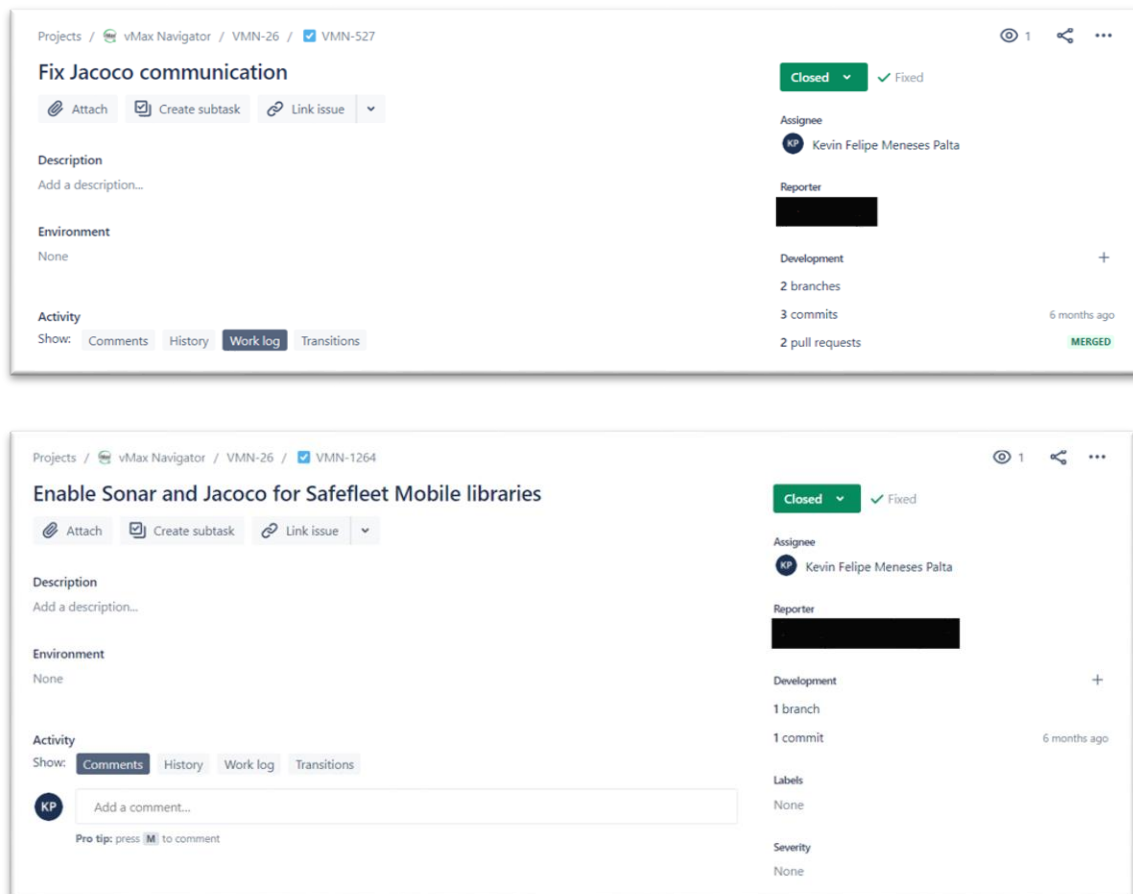


Figura 12. Tickets de Jira VMN-527 y VMN-1264. Fuente: Jira de SafeFleet

5.5.1 Cobertura del código

Según la documentación oficial en su sitio web, “JaCoCo es una librería gratuita para la cobertura de código en Java, que ha sido creada por el equipo de EclEmma basada en las lecciones aprendidas del uso e integración de bibliotecas existentes durante muchos años” [79].

Esta librería es capaz de calcular la cantidad y porcentaje de líneas, métodos y clases cubiertas por pruebas unitarias y generar un reporte en html o xml para su visualización en otras herramientas. Utilizando Gradle, que hace parte del ambiente de Android Studio para la configuración y administración de dependencias del proyecto, se pueden crear *tasks* o tareas para la generación de dichos reportes y

subirlos al servidor de sus clientes. Funciona perfectamente con el lenguaje de programación Kotlin ya que es completamente interoperable con Java.

5.5.2 Métricas de inspección del código

Según la documentación oficial en su sitio web, “SonarQube es una herramienta de revisión automática para detectar errores, vulnerabilidades y olores en su código. Puede integrarse con su flujo de trabajo existente para permitir la inspección continua del código en las ramas de su proyecto y pull requests” [80]. Además, permite la integración con otras librerías y herramientas como JaCoCo, Jenkins y GitHub.

SonarQube se configura como un servidor que puede ser utilizado localmente o en la nube y que permite la recepción de datos provenientes del proyecto como la cantidad de líneas de código, métodos, clases, cobertura de pruebas unitarias y mutantes para su posterior análisis y visualización. Esta herramienta permite mostrar los reportes xml subidos y generados por JaCoCo.

5.5.3 Implementación

Debido a la arquitectura modularizada del proyecto fue necesario hacer una investigación para conocer cómo generar los reportes de cobertura de cada módulo y subirlos al servidor de SonarQube del cliente.

Configuración de dependencias: es necesario agregar las dependencias en el archivo de configuración general de Gradle del proyecto de la siguiente forma:

```
dependencies {
    classpath "org.sonarsource.scanner.gradle:sonarqube-gradle-
plugin:$sonar_version"
    classpath "org.jacoco:org.jacoco.core:$jacoco_version"
}

plugins {
    id "org.sonarqube" version "2.7"
}
```

Configuración de SonarQube: esta configuración puede ser agregada directamente en el archivo de configuración general de Gradle del proyecto o en un archivo con extensión `.gradle` y ser incluido en la configuración general de la siguiente forma:

```
apply from: "directorio/sonarconfig.gradle"

archivo: sonarconfig.gradle:
sonarqube {
    properties {
        property "sonar.projectName", "proyecto"
        property "sonar.projectKey", "clave-proyecto"
        property "sonar.host.url", "URL del servidor de sonar"
        property "sonar.login", "código de login generado por SonarQube"
        property "sonar.sources", "directorio de los recursos del proyecto"
        property "sonar.binaries", "directorio de los archivos binarios del proyecto"
        property "sonar.java.binaries", "directorio de los archivos binarios java del proyecto"
        property "sonar.tests", "directorio de las pruebas unitarias del proyecto"
        property "sonar.java.test.binaries", "directorio de los archivos binarios de las pruebas unitarias del proyecto"
        property "sonar.exclusions", "exclusiones de archivos del reporte de sonar"
        property "sonar.java.coveragePlugin", "plugin de cobertura de pruebas, en este caso JaCoCo"
        property "sonar.jacoco.reportPath", "directorio de los reportes de JaCoCo"
        property "sonar.coverage.jacoco.xmlReportPaths", "directorio de los reportes de JaCoCo en XML"
        property "sonar.junit.reportsPath", "directorio de reportes de Junit"
        property "sonar.android.lint.report", "directorio de reportes de Lint"
        property "dc5.mutationAnalysis.pitest.sensor.reports.directory", "directorio de reportes de Pitest"
    }
}
```

Configuración de JaCoCo: esta configuración sirve para crear un *task* o tarea de Gradle que permite generar los reportes de JaCoCo para su posterior envío al servidor de SonarQube y debe estar presente en la configuración de Gradle de cada módulo para lo cual es necesario crear un archivo con la extensión `.gradle` e incluirlo en cada uno de ellos de la siguiente forma:

```

apply plugin: 'jacoco'
apply from: 'directorio/jacococonfig.gradle'

testOptions {
    unitTests.all {
        useJUnitPlatform() // función necesaria cuando se utiliza Junit5
        jacoco {
            destinationFile = file("directorio/testDebugUnitTest.exec")
            includeNoLocationClasses = true
            excludes = ['jdk.internal.*']
        }
    }
}

```

archivo jacococonfig.gradle:

```

task testDebugUnitTestCoverage(type: JacocoReport) {
    group = 'grupo de tareas en la que se ubica'
    description = "descripción de la tarea"

```

// esta sección sirve para ejecutar las pruebas unitarias de cada módulo antes de la generación del reporte

```

dependsOn ":presentation:testDebugUnitTest"
dependsOn ":domain:testDebugUnitTest"
dependsOn ":data:testDebugUnitTest"

```

```

classDirectories.from = files ([
    fileTree(
        dir: "directorio de clases de Kotlin",
        exclude: "archivos excluidos del reporte"
    ),
    fileTree(
        dir: "directorio de intermediarios de Java del módulo de presentación",
        exclude: "archivos excluidos del reporte"
    ),
    fileTree(
        dir: "directorio de intermediarios de Java del módulo de dominio",
        exclude: "archivos excluidos del reporte"
    ),
    fileTree(
        dir: "directorio de intermediarios de Java del módulo de datos",
        exclude: "archivos excluidos del reporte"
    )
])

```

```

def coverageSourceDirs = [
    "directorio de recursos del módulo de presentación",
    "directorio de recursos del módulo de dominio",
    "directorio de recursos del módulo de datos"
]

additionalSourceDirs.from = files(coverageSourceDirs)
sourceDirectories.from = files(coverageSourceDirs)
executionData.from = fileTree(dir: project.rootDir, include: [
    "directorio del archivo de JaCoCo del módulo de presentación",
    "directorio del archivo de JaCoCo del módulo de dominio",
    "directorio del archivo de JaCoCo del módulo de datos"
])

// configuración de la extensión de los reportes
reports {
    xml.enabled = true
    html.enabled = true
}
}

```

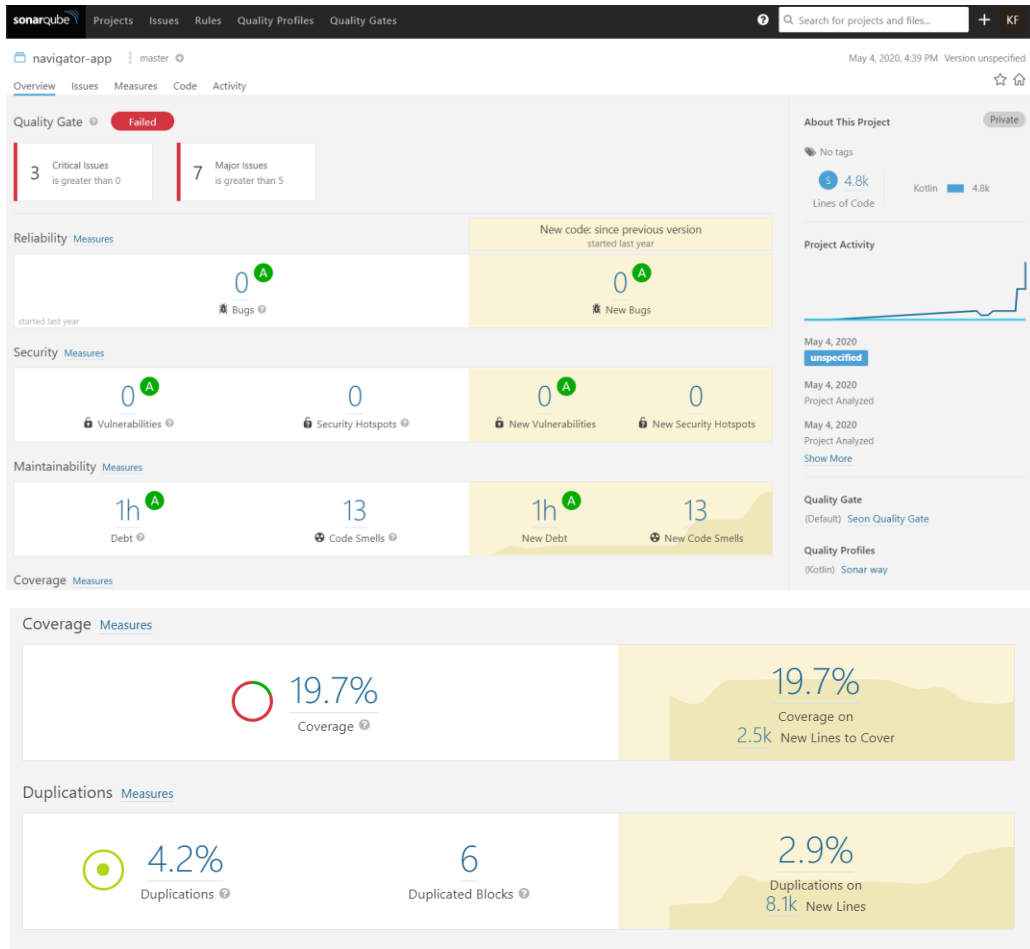
Ejecución y envío de los reportes: para ejecutar y enviar los reportes de JaCoCo a SonarQube es necesario ejecutar el *Task* creado anteriormente desde la pestaña de Gradle o desde el terminal de Android Studio con el comando:

```
“gradlew testDebugUnitTestCoverage”.
```

Finalmente, después de que las pruebas hayan pasado exitosamente y se hayan generado los reportes, para subirlos al servidor de SonarQube se debe ejecutar el comando:

```
“gradlew sonarqube”
```

Reportes en SonarQube: el resultado de los reportes se puede visualizar en el servidor de SonarQube del cliente como se evidencia en la Figura 12.



Project Overview

- Reliability
- Security
- Maintainability
- Coverage** Coverage

Coverage

- Overall: 19.7%
- Lines to Cover: 2,514
- Uncovered Lines: 1,948
- Line Coverage: 22.5%
- Conditions to Cover: 943
- Uncovered Conditions: 828
- Condition Coverage: 12.2%

Duplications

File Path	Coverage	Lines	Uncovered
presentation/src/main/java/com/navigator/presentation/mappers/WasteRouteMappers.kt	0.0%	12	-
presentation/src/main/java/com/navigator/presentation/ui/routes/waste/WasteRoutesViewEvent.kt	0.0%	1	-
presentation/src/main/java/com/navigator/presentation/ui/settings/waste/WasteSettingsViewModel.kt	0.0%	33	18
data/src/main/java/com/navigator/data/mappers/temporal/routes/WasteAssetsMappers.kt	0.8%	118	2
data/src/main/java/com/navigator/data/mappers/RoutesEntityMappers.kt	18.4%	77	3
data/src/main/java/com/navigator/data/datasource/remote/authentication/AuthenticationRemoteDataSourceImpl.kt	32.1%	8	11
domain/src/main/java/com/navigator/domain/usecase/ddp/DdpUseCaseImpl.kt	33.3%	2	-
data/src/main/java/com/navigator/data/datasource/remote/routerloader/RouterLoaderRemoteDataSourceImpl.kt	34.8%	49	39
data/src/main/java/com/navigator/data/datasource/local/routerloader/RouterLoaderLocalDataSourceImpl.kt	37.2%	32	71
presentation/src/main/java/com/navigator/presentation/ui/splash/SplashViewModel.kt	40.3%	24	16
presentation/src/main/java/com/navigator/presentation/ui/base/BaseViewModel.kt	42.9%	4	-
presentation/src/main/java/com/navigator/presentation/ui/routes/waste/WasteRouteListViewModel.kt	48.1%	10	18
data/src/main/java/com/navigator/data/repository/routerloader/RouterLoaderRepositoryImpl.kt	49.7%	91	81
presentation/src/main/java/com/navigator/presentation/ui/temporal/AppSelectorViewModel.kt	54.8%	9	5
presentation/src/main/java/com/navigator/presentation/ui/wastelogs/WasteLoginViewModel.kt	63.3%	9	13
data/src/main/java/com/navigator/data/repository/authentication/AuthenticationRepositoryImpl.kt	76.0%	2	4
domain/src/main/java/com/navigator/domain/usecase/routerloader/RouterLoaderUseCaseImpl.kt	85.7%	3	-
data/src/main/java/com/navigator/data/mappers/temporal/routes/WasteAssetStateMappers.kt	93.1%	2	-

Figura 13. Reporte de calidad del proyecto. Fuente: SonarQube de SafeFleet

SafeFleet PTLW / vMax Navigator / navigator-app / Pull requests

KMENESESP/VMN-527 FIX JACOCO CONFIG

#153 **MERGED** at ba792e0 `KMENESESP/VMN-527_FIX_J...` → `deve1op` Revert Approve 3

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta

Reviewers **SH**

Description **Evidence**

VMN-527 [Stop watching](#)

navigator-app | master

Overview Issues Measures Code Activity

Reliability Measures

0 **A** Bugs

started 4 days ago

New code since 1.0 started 8 hours ago

0 **A** New Bugs

Security Measures

0 **A** Vulnerabilities

0 Security Hotspots

0 **A** New Vulnerabilities

0 New Security Hotspots

Maintainability Measures

12min **A** Debt

2 Code Smells

0 **A** New Debt

0 New Code Smells

Coverage Measures

19.8% Coverage

64.3% Coverage on 18 New Lines to Cover

SafeFleet PTLW / vMax Navigator / sfm-base / Pull requests

KMENESESP/VMN-1264 ENABLE SONAR JACOCO

#89 **MERGED** at 4a8107d `KMENESESP/VMN-1264_ENAB...` → `deve1op` Revert Approve 2

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta

Reviewers **SH**, **SL**

Description **Description**

- Added the jacoco, sonar and exceptions files and configured all the modules.

Jira

<https://safefleet.atlassian.net/browse/VMN-1264>

Comments (0)

KP What would you like to say?

Files changed (14)

+0	-3	M	app/build.gradle
+0	-51	D	app/jacoco.gradle
+0	-16	D	app/sonar.gradle
+15	-2	M	authentication/build.gradle
+0	-51	D	authentication/jacoco.gradle
+0	-16	D	authentication/sonar.gradle
+15	-0	M	avml/build.gradle
+3	-12	M	build.gradle
+15	-3	M	commons/build.gradle
+0	-8	D	...

Figura 14. Pull requests de los tickets VMN-527 y VMN-1264. Fuente: repositorio de SafeFleet en Bitbucket

5.6 Barra de búsqueda personalizada

Projects / vMax Navigator / VMN-404 / VMN-461

Create Search View

Attach Link issue

Description
Add a description...

Environment
None

Activity
Show: Comments History Work log Transitions

Pro tip: press **M** to comment

Closed ✓ Fixed

Assignee

Reporter

Development
2 branches
4 commits
4 pull requests
11 months ago
MERGED

Labels
None

Projects / vMax Navigator / VMN-860 / VMN-876

Adapt search existing functionality to support wastepapp flow

Attach Create subtask Link issue

Description
User Story
As a truck driver,
I want to see the message indicating not results found when I am searching for a route
so that I can be clear for that

Preconditions

- Master list must have been downloaded from Back Office.
- Driver must be logged on
- Routes associated with the truck must have been downloaded from Back Office.

To Do

- If the driver is searching for a not existing route, it should be possible to allow users to see a message if the required route name is not on the list.

Delivered ✓ Work Completed

Assignee

Reporter

Development
3 branches
14 commits
2 pull requests
6 months ago
MERGED

Labels
WAS

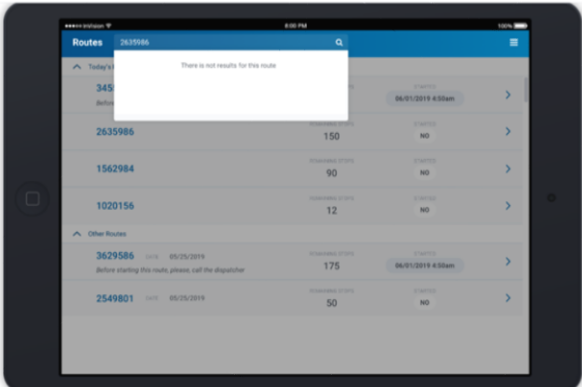
Severity
None

Projects / vMax Navigator / VMN-860 / VMN-876

Scenarios (Acceptance Criteria)

Routes not found

GIVEN the user is logged on AND the ROUTE list panel is shown
AND the user started typing a route name
AND the written route does not exist
THEN the message There is no results for this route is shown



Delivered ✓ Work Completed

Assignee

Reporter

Development
3 branches
14 commits
2 pull requests
6 months ago
MERGED

Labels
WAS

Severity
None

Customer(s)
None

Story Points
5

Time tracking

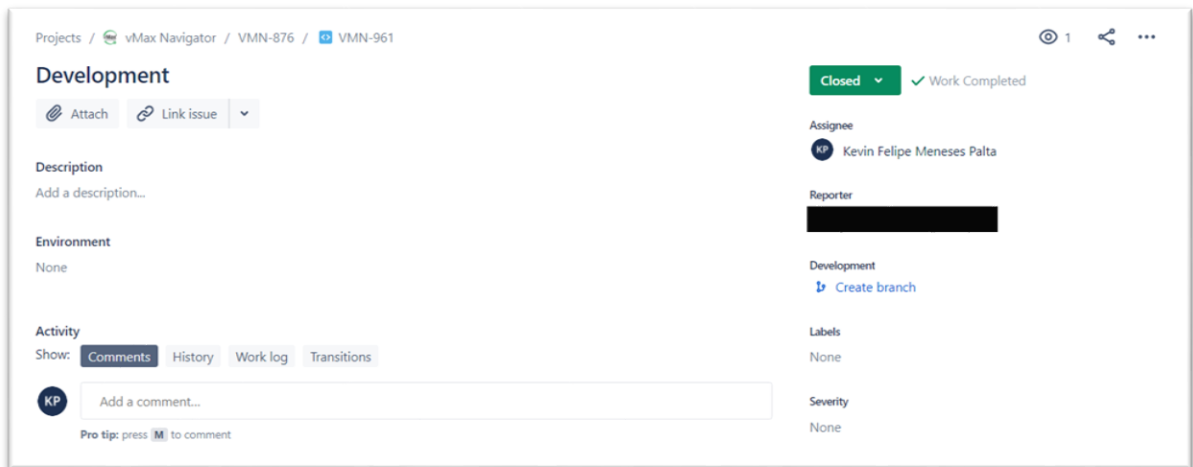


Figura 15. Tickets de Jira VMN-461, VMN-876 y VMN-961. Fuente: Jira de SafeFleet

Descripción: para el desarrollo de esta tarea fue necesario crear una vista personalizada que cumpliera con los criterios de aceptación del diseño propuesto para el proyecto, debido a que la implementación anterior no se ajustaba al original. Según la propuesta del equipo de diseño la barra de búsqueda debe ser desplegable y permitir seleccionar alguno de sus resultados para ejecutar una acción específica, en este caso mostrar una ruta.

5.6.1 Diseño XML

La implementación de esta vista personalizada incluye un layout xml con varias etiquetas para los diferentes elementos contenidos como se muestra en la jerarquía a continuación:

```
<merge> // etiqueta usada para remover las etiquetas de layout contenidas para
que hagan parte de un solo layout padre y mejorar el rendimiento de renderizado
<CardView>
  <RelativeLayout>
    <LinearLayout>
      <ImageView/> // Icono de flecha para cerrar la búsqueda
      <EditText/> // Entrada de texto para la búsqueda
      <ImageView/> // Icono para limpiar la entrada de texto
    </LinearLayout>
  <RelativeLayout>
    <View/> // Elemento divisor de la lista
```

```

<RecyclerView/> // Lista de elementos o rutas
<CardView>
  <LinearLayout>
    <TextView/> // Texto de confirmación de creación del elemento o ruta
    <Button/> // Botón para crear el elemento o ruta
  </LinearLayout>
</CardView>
</RelativeLayout>
<ImageView/> // Icono que confirma la búsqueda
</RelativeLayout>
</CardView>
</merge>

```

5.6.2 Comportamiento de la barra

La lógica que controla el comportamiento de la vista se implementó en una clase que extiende de la clase `FrameLayout`, encargada de inflar y manejar los elementos del layout creado y sus funcionalidades que son:

- Abrir o habilitar búsqueda
- Cerrar o deshabilitar búsqueda
- Despliegue animado de la lista
- Limpiar la búsqueda y la entrada de texto
- Configurar la cantidad de elementos visibles
- Ejecutar una acción con el elemento seleccionado
- Habilitar y deshabilitar la opción de crear un nuevo elemento

Esta clase implementa varias interfaces para el manejo de eventos:

- `View.OnClickListener`: para manejar el evento de click en las diferentes vistas de la clase.
 - Layout contenedor: abrir la barra de búsqueda.
 - Icono de flecha y equis: limpiar el texto y cerrar la barra de búsqueda.
 - Botón de crear: crear una nueva ruta.
- `Animation.AnimationListener`: para manejar los eventos de inicio y finalización de la animación en la lista de elementos.

- Esconder los elementos que adornan la barra de búsqueda cuando está cerrada.
- View.OnFocusChangeListener: para manejar los eventos cuando la barra de búsqueda se encuentra enfocada, abierta o cerrada.
 - Mostrar u ocultar el teclado
- TextView.OnEditorChangeListener: para manejar los eventos de cambio de texto en la entrada de texto de la barra de búsqueda.
 - Hacer una nueva petición al servidor

El diagrama de las clases relacionadas se puede ver a continuación en la figura 15.

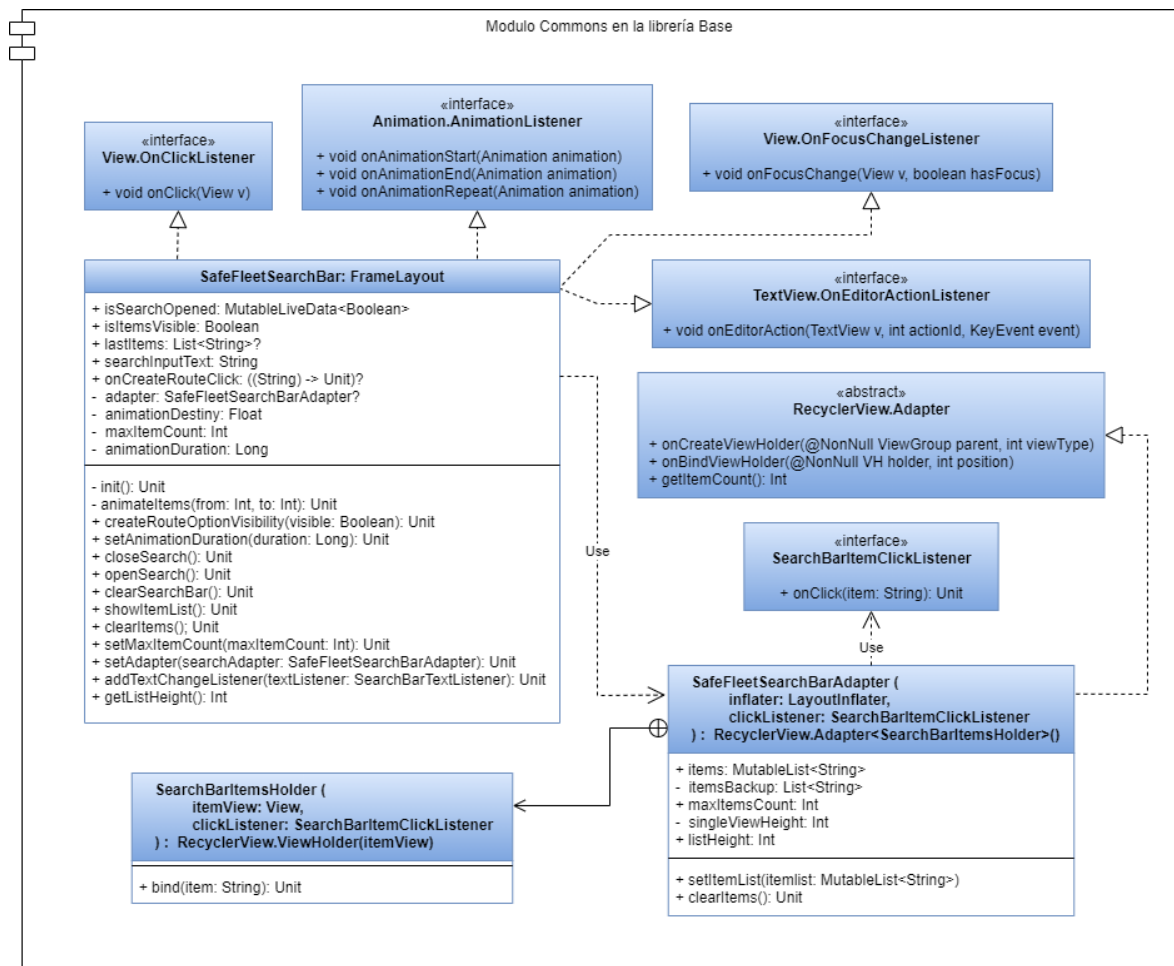


Figura 16. Diagrama de barra de búsqueda personalizada. Fuente propia

SafeFleet PTLW / vMax Navigator / navigator-app / Pull requests

KMENESESP/VMN-876 SEARCH BAR FULL IMPLEMENTATION

#127 **MERGED** at f6dfe15 `VMN-876-SEARCH_BAR_FULL--` → `develop` Revert Approve 3

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta VMN-876
 Reviewers Stop watching

Description

Routes ← a ×

Time	Route	Vehicle	Action
11:00 AM	AutomationRou...	jvallevehicle	Navigate
01:00 PM	AutomationRou...	jvallevehicle	Navigate
01:30 PM	AutomationRou...	jvallevehicle	Navigate
05:30 PM	AutomationRou...	jvallevehicle	Navigate

Routes ← 23325263 ×

No results found

Time	Route	Vehicle	Action
11:00 AM	AutomationRou...	jvallevehicle	Navigate
01:00 PM	AutomationRou...	jvallevehicle	Navigate
01:30 PM	AutomationRou...	jvallevehicle	Navigate
05:30 PM	AutomationRou...	jvallevehicle	Navigate

Figura 17. Pull request del ticket VMN-876. Fuente: repositorio de SafeFleet en Bitbucket

5.7 Creación de rutas

Projects / vMax Navigator / VMN-860 / VMN-960

Create route

Attach Create subtask Link issue

Description
User Story
 As a truck driver,
 I want to create a route when I don't find the searched route
 so that I can start my job

Preconditions

- Driver must be logged on
- Routes may be downloaded from the back end but not required.

To Do

1. Allow users to create routes locally if the required route name is not on the list

Scenarios (Acceptance Criteria)

Enter route manually - Create a new route

GIVEN the user has typed the new route name in the route list screen
 WHEN there is no route with the typed name available
 THEN show the "CREATE" button

Delivered ✓ Fixed

Assignee

Reporter

Development
[Create branch](#)

Labels
[WAS](#)

Severity
 None

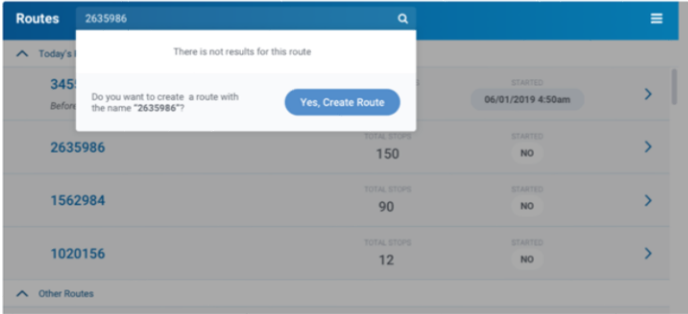
Customer(s)
 None

Story Points
 5

Time tracking
 5h logged

AND this created route appears in the route list panel
 AND if the user select this route, THEN navigate to the next WateApp screen (Assignment List panel) without any assignment

Attachment



Routes 2635986

There is not results for this route

Do you want to create a route with the name "2635986"? Yes, Create Route

ROUTE ID	TOTAL STOPS	STARTED
2635986	150	NO
1562984	90	NO
1020156	12	NO

Reporter

Development
[Create branch](#)

Labels
[WAS](#)

Severity
 None

Customer(s)
 None

Story Points
 5

Time tracking
 5h logged

Projects / vMax Navigator / VMN-960 / VMN-1226

Modify search bar functionality to support non-existent routes

Attach Link issue

Description

- Search on BD for the particular route: show messages (following UI requirements)
- Take into account this functionality is just for Wasteapp flow

Environment

None

Activity

Show: Comments History Work log Transitions

KP Add a comment...

Pro tip: press **M** to comment

Closed ✓ Work Completed

Assignee

Kevin Felipe Meneses Palta

Reporter

[Redacted]

Development

1 branch

2 commits 6 months ago

1 pull request **MERGED**

Labels

None

Projects / vMax Navigator / VMN-960 / VMN-1227

Create new route on BD - routes table

Attach Link issue

Description

- Create on route tables: ID, current date, Total stops = 0, vehicle
- Update UI to show the created route in the route list screen in the first place
- Update datatypes to accept null values

Environment

None

Activity

Show: Comments History Work log Transitions

KP Add a comment...

Pro tip: press **M** to comment

Closed ✓ Work Completed

Assignee

Kevin Felipe Meneses Palta

Reporter

[Redacted]

Development

1 branch

1 commit 6 months ago

1 pull request **MERGED**

Labels

None

Severity

None

Projects / vMax Navigator / VMN-960 / VMN-1231

Update UI (assignment List) to support showing screen without assignments

Attach Link issue

Description

Add a description...

Environment

None

Activity

Show: Comments History Work log Transitions

KP Add a comment...

Pro tip: press **M** to comment

Closed ✓ Work Completed

Assignee

Kevin Felipe Meneses Palta

Reporter

[Redacted]

Development

Create branch

Labels

None

Severity

None

Customer(s)

None

Figura 18. Tickets de Jira VMN-960, VMN-1226, VMN-1227 y VMN-1231. Fuente: Jira de SafeFleet

Descripción: para el desarrollo de esta funcionalidad fue necesario implementar una función dentro de la clase de la barra de búsqueda que permitiera mostrar y ocultar un botón de creación de ruta junto con el nombre de la ruta ingresada en la entrada de texto de la barra. Esta opción debe aparecer al final de la lista de la barra cuando el texto ingresado en la entrada de texto de la barra no arroje ningún resultado de búsqueda por parte del servidor. Además, debe implementar una forma de comunicar el evento del botón con la actividad para permitir ejecutar la acción deseada, que en este caso es guardar la ruta en la base de datos.

5.7.1 Evento del botón de creación de rutas de la barra

La forma común de comunicar eventos entre clases Java son interfaces, pero en Kotlin existen un tipo de funciones llamadas funciones de orden superior que pueden ser declaradas variables dentro de una clase e implementadas como funciones en otras clases. Solamente se debe invocarlas cuando ocurra un evento dentro de la clase. Un ejemplo de su implementación se muestra a continuación:

```
// en la clase de la vista personalizada
var onCreateRouteClick: ((String) -> Unit)? = null

createRouteButton.setOnClickListener {
    onCreateRouteClick?.invoke(searchInputText)
}

// en la clase de la actividad o fragmento
SafeFleetSearchBar.onCreateRouteClick = { route ->
    createRouteInDataBase(route)
}
```

5.7.2 Guardar ruta en la base de datos

Después de que el usuario seleccione la opción de crear ruta de la barra de búsqueda es necesario ir hasta la base de datos y hacer una operación de insertar en la tabla que guarda todas las rutas traídas desde el servidor. Para realizar esta operación es necesario seguir el flujo de la arquitectura manejada en el proyecto pasando desde: Actividad, ViewModel, Caso de Uso, Repositorio y Fuente de Datos

de los diferentes módulos de la aplicación devolviendo una respuesta del resultado de la operación para informar al usuario, como se puede visualizar en el diagrama de clases que representa el flujo de la operación realizada en la figura 18.

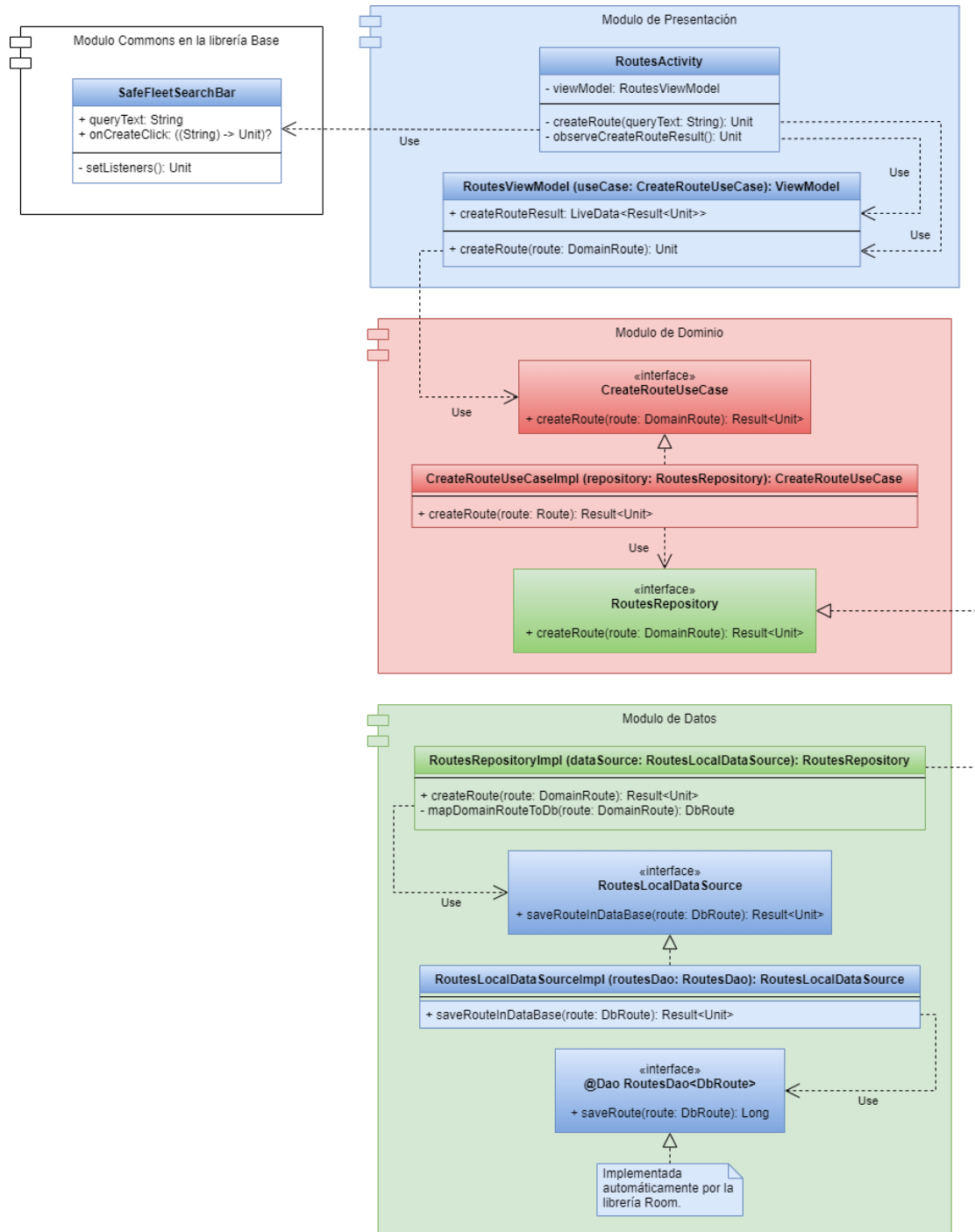


Figura 19. Diagrama del caso de uso crear ruta. Fuente propia

KMENESESP/VMN-1227 CREATE NEW ROUTE WASTEAPP

#175 **MERGED** at c449b41

KMENESESP/VMN-1227_CREA... → develop

Revert Approve 2

Overview **Commits** Activity

Author **KP** Kevin Felipe Meneses Palta

Reviewers **SL**

VMN-1227

Stop watching

Description **Evidence**

Routes			
← 2634587			
No results found			
Do you want to create a route with the name 2634587? <input checked="" type="button" value="Yes, Create Route"/>			
Today's Routes			
Route's not found			
Other Routes			
3455979	Date 11/03/2020	REMAINING STOPS 4	STARTED NO
3455981	Date 11/03/2020	REMAINING STOPS 3	STARTED NO
3455976	Date 13/03/2020	REMAINING STOPS 7	STARTED NO
<i>The incomplete route, continue it today</i>			
3455977	Date 13/03/2020	REMAINING STOPS 4	STARTED NO
<i>Before starting the route, please call dispatcher.</i>			

Routes			
Today's Routes			
Date 17/03/2020			
2634587		REMAINING STOPS 0	STARTED NO
Other Routes			
3455979	Date 11/03/2020	REMAINING STOPS 4	STARTED NO
3455981	Date 11/03/2020	REMAINING STOPS 3	STARTED NO
3455976	Date 13/03/2020	REMAINING STOPS 7	STARTED NO
<i>The incomplete route, continue it today</i>			
3455977	Date 13/03/2020	REMAINING STOPS 4	STARTED NO
<i>Before starting the route, please call dispatcher.</i>			
3455978	Date 13/03/2020	REMAINING STOPS	STARTED

SafeFleet PTLW / vMax Navigator / sfm-base / Pull requests

KMENESESP/VMN-1226 MODIFY SEARCH BAR TO ADD ROUTES

#90 **MERGED** at 639b406 `KMENESESP/VMN-1226_MODI...` → `develop` Revert Approve 2

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta VMN-1226
 Reviewers **SH** Stop watching

Description **Description**

- Added a cardview to show create route option in searchbar
- Added a function to enable or disable it.

Jira

<https://safefleet.atlassian.net/browse/VMN-1226>

Comments (0)

KP What would you like to say?

Files changed (8)

+20	-4	M	commons/src/main/java/com/safefleet/mobile/commons/widgets/searchbar/SafeFleetSearchBar.kt	1
+6	-0	A	commons/src/main/res/drawable/createroute_button.xml	
+43	-0	M	commons/src/main/res/layout/safefleet_searchbar.xml	
+5	-0	M	commons/src/main/res/values-sw400dp/dimens.xml	
+5	-0	M	commons/src/main/res/values-sw600dp/dimens.xml	
+5	-0	M	commons/src/main/res/values-sw720dp/dimens.xml	

Figura 20. Pull requests de los tickets VMN-1226 y VMN-1227. Fuente: repositorio de SafeFleet en Bitbucket

5.8 Escenarios sin conexión de la aplicación

Projects / vMax Navigator / VMN-411 / VMN-1128

Offline scenarios for Wasteapp main flow

Delivered ✓ Fixed

Attach Create subtask Link issue

Description

Feature

When there is no internet connection or connection with the back office, WasteApp should operate correctly. The same way it operates when both the internet connection and connection with back-office are available. Any information (data) which cannot be sent to the back office at any moment in time has to be kept locally until the connection is restored.

To do

Implement controls to allow the normal flow of events.

General Scenarios

Given the Wasteapp user turns on the device and starts the application
When there is no connection to the internet
Then on the application status panel, show the internet connection status "Offline" and the back-office connection status "Disconnected."

Given the Wasteapp user turns on the device and starts the application
When the internet connection is available
And there is no connection with the back office
Then on the application status panel, show the internet connection status "Online" and the back-office connection status "Disconnected."

Given the Wasteapp user turns on the device and starts the application
When both the internet connection and connection with the back office are available
Then on the application status panel, show the internet connection status "Online" and the back-office connection status "Connected."

Assignee: [Redacted]

Reporter: [Redacted]

Development +

2 branches

2 commits 6 months ago

1 pull request MERGED

Labels: WAS

Severity: None

Customer(s): None

Epic Link: TR - Offline Support

Projects / vMax Navigator / VMN-411 / VMN-1128

Driver login Scenarios

Delivered ✓ Fixed

Assignee: [Redacted]

Reporter: [Redacted]

Development +

2 branches

2 commits 6 months ago

1 pull request MERGED

Labels: WAS

Severity: None

Customer(s): None

Epic Link: TR - Offline Support

Driver login Scenarios

Given the Wasteapp user turns on the device and starts the application
When there is no list of drivers in the local application database
And there is no connection with the back office
Then the app shows the message: "Cannot retrieve the list of users from the remote server."

Given the Wasteapp user is on the login screen
When he wants to refresh the lists of drivers because his/her name is not in the list
And there is no internet connection
Then the app shows the message: "Cannot retrieve the list of users from the remote server."

Route select Scenario (VMN-960: Create route DELIVERED)

Given the WasteApp user is logged in and the route list panel is shown
When there are no routes in the local application database
Or the driver cannot find the route he/she needs in the list of available routes (Regardless of the reason. Connectivity with the back office may exist back routes due to technical issues on the back office side were not dispatched)
Then driver should be able to type the name of the route he/she needs to work on and create the route on locally. (as soon as assignments for the manually created route get dispatched by the back office, these assignments have to be added to the manually created route and shown to the driver)

Working on routes Scenario

Given the Wasteapp user is working with the application
When the back-office connectivity is not available
Then all messages (LON, RTS, RTD, VD, etc...) have to be stored locally and sent to the back office as soon as the connection is restored.

NOTE

All PDM messages have to be sent to the back office in order they were created.

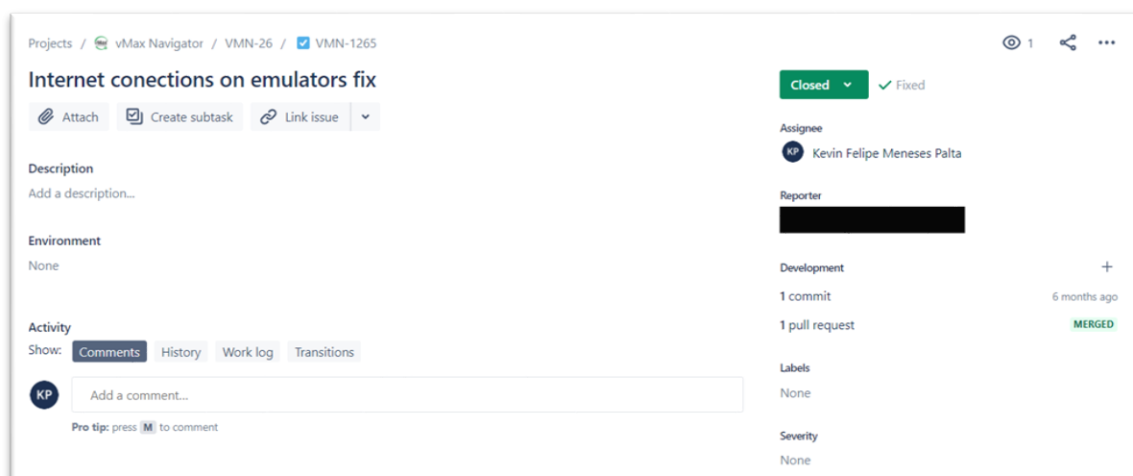


Figura 21. Tickets de Jira VMN-1128 y VMN-1265. Fuente: Jira de SafeFleet

5.8.1 Manejo de conexión a Internet

El manejo del estado de conexión a internet es una funcionalidad importante para la ejecución de las diferentes funciones de la aplicación. Por eso desde el principio del desarrollo de la aplicación se creó una clase que ha sido implementada de varias formas, pero ha tenido algunos inconvenientes en su funcionamiento. Una de las tareas asignadas implica el cambio en su implementación.

Para conocer si hay conexión a Internet a pesar de que el Wi-Fi o los datos del dispositivo estén activados se creó una clase extendiendo de una clase de tipo observable que contiene un valor booleano, cuya primera implementación realizaba ping a una dirección url confiable como Google, pero en ocasiones no se podía leer el valor esperado, así que en lugar de esto se utilizó la clase `URLConnection` con un tiempo de espera de 3 segundos para establecer una conexión con la dirección url deseada antes de arrojar una excepción de tiempo de espera, la cual es capturada para publicar un valor en la clase. Además, la clase utiliza un callback de red que permite conocer cuándo está disponible la conexión por Wi-Fi o por datos móviles en caso de que el usuario los desactive o el dispositivo se encuentre en modo avión. A continuación, se presenta el diagrama de la clase en la Figura 21.

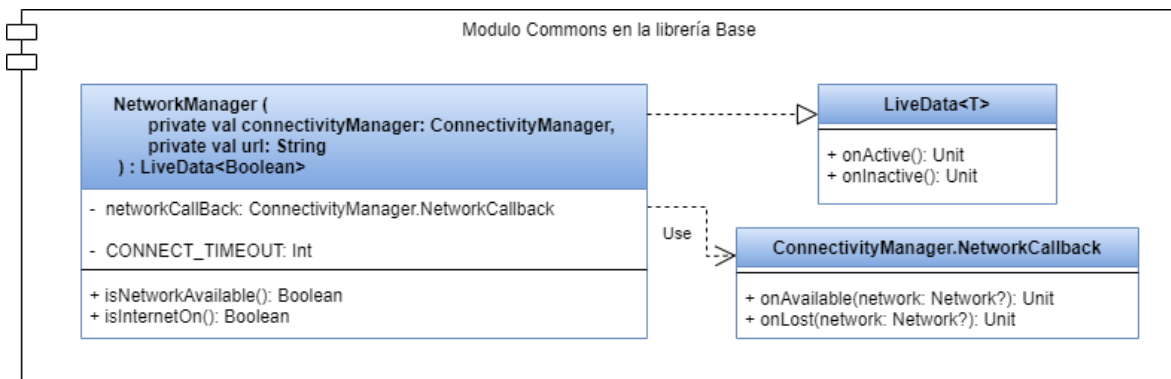


Figura 22. Diagrama del administrador de red. Fuente propia

La función `isNetworkAvailable()` permite conocer si existe una red de tipo Wi-Fi o datos móviles disponible devolviendo un valor booleano. La función `isInternetOn()` establece una conexión con la url proveída por el constructor de la clase para conocer si hay o no conexión a Internet también retornando un valor booleano. Las funciones `onActive()` y `onInactive()` parte de la clase `LiveData` se sobre escriben para registrar y anular el registro del callback de red. Finalmente, las funciones `onAvailable()` y `onLost()` del callback de red se sobre escriben para publicar un valor en la clase solo en caso de que sea distinto al existente.

El uso de esta clase se implementa en diferentes partes del proyecto antes de realizar peticiones al servidor, se observa constantemente para realizar envío de eventos de red a Firebase Analytics y Crashlytics, para el manejo los escenarios fuera de línea del proyecto y para el envío de todo tipo de eventos de la aplicación al servidor.

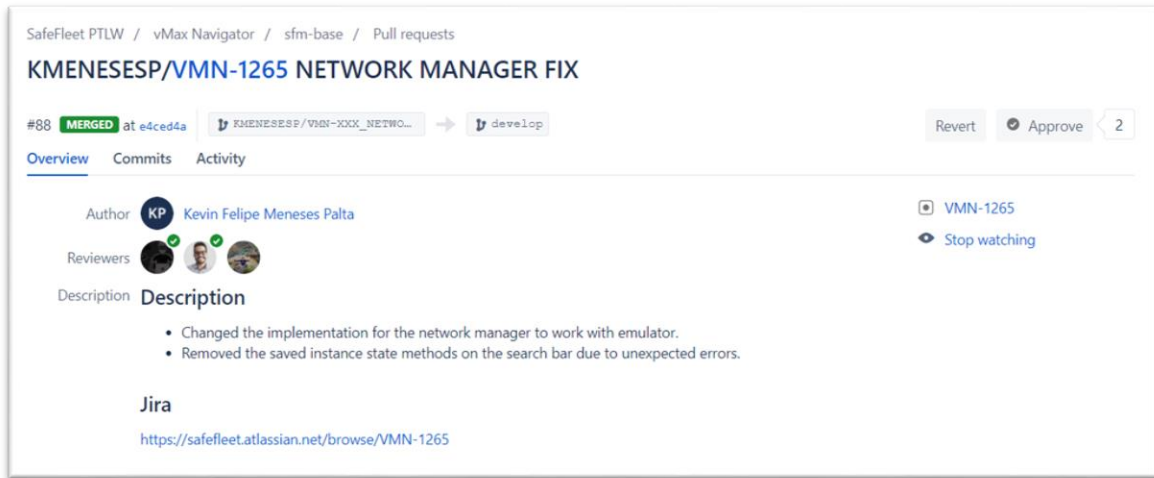


Figura 23. Pull request del ticket VMN-1265. Fuente: repositorio de SafeFleet en Bitbucket

5.8.2 Manejo de conexión al servidor

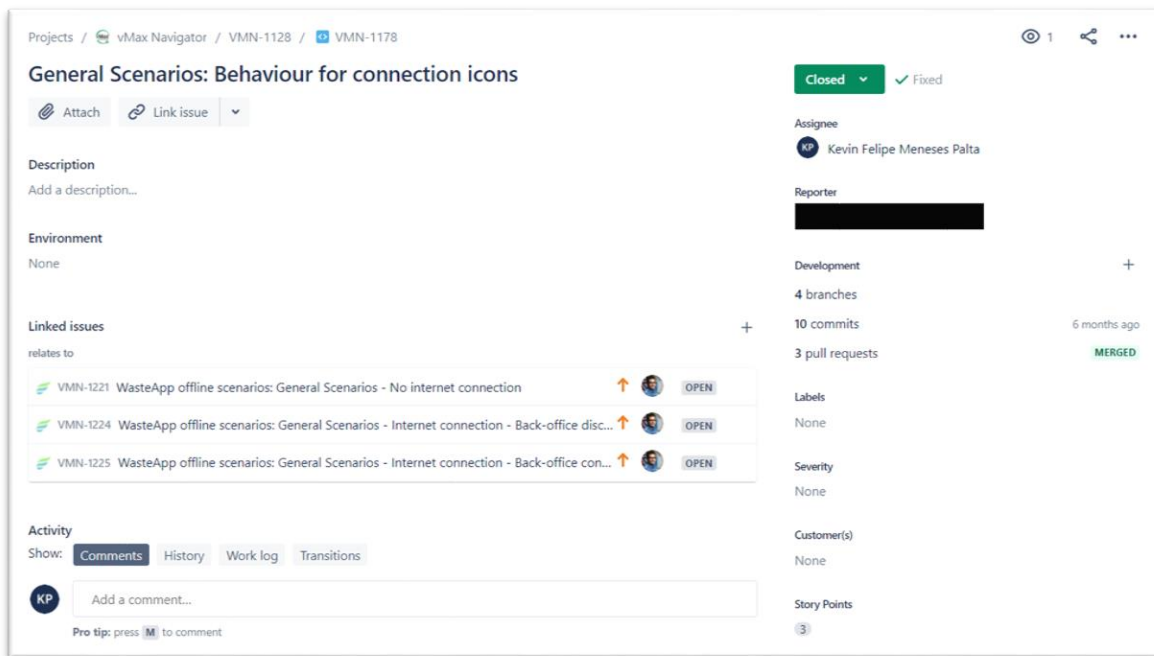


Figura 24. Ticket de Jira VMN-1178. Fuente: Jira de SafeFleet

Para los escenarios fuera de línea de la aplicación además de la necesidad de conocer el estado de la conexión a Internet hace falta conocer el estado del servidor. Con el fin de cumplir con los requerimientos de esta tarea, se implementó un

procedimiento recurrente que revisa el estado del servidor cada 10 segundos y publica una variable observable que la aplicación utiliza para mostrar u ocultar un icono de desconexión del servidor en la pantalla de rutas y de navegación de la aplicación.

La implementación de esta funcionalidad consta de dos clases y un objeto (clase estática en Kotlin). Su procedimiento al igual que el administrador de conexión a Internet utiliza la clase `URLConnection` para establecer una conexión con las urls proveídas al objeto, pero en este caso se lee el código de respuesta `Http` y se verifica si el código es de tipo `5xx` para publicar un valor booleano en el parámetro observable del objeto, siendo falsa en caso de que alguno de los endpoints del servidor tenga un error interno. A continuación, se presenta el diagrama de las clases en la Figura 24.

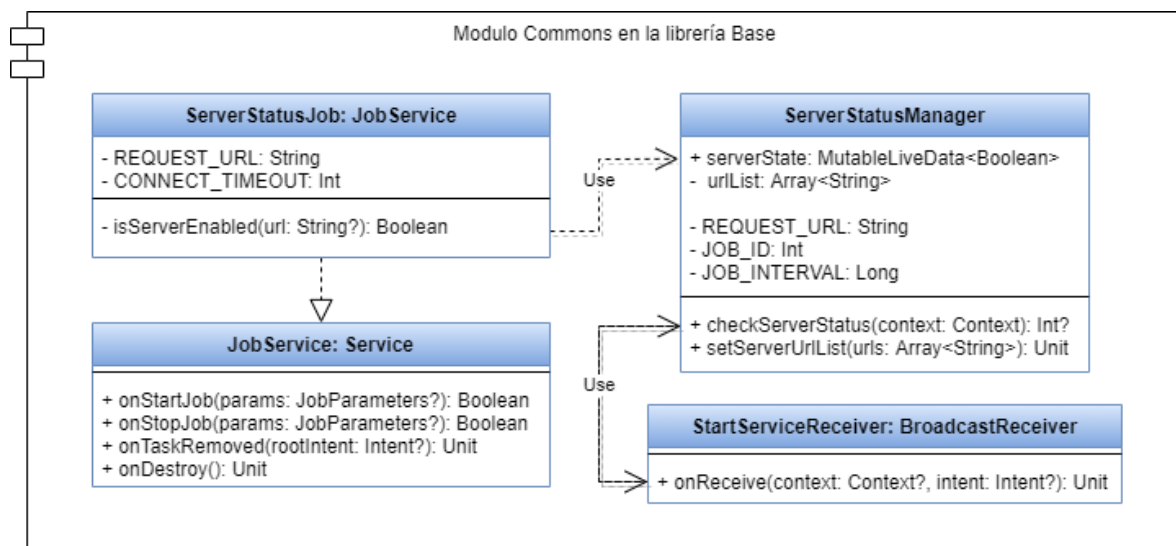


Figura 25. Diagrama del administrador de estado del servidor. Fuente propia

La clase `ServerStatusJob` que realiza la ejecución recurrente del procedimiento extiende de una clase llamada `JobService` la cual se registra mediante un método en el objeto `ServerStatusManager` llamado `checkServerStatus(context)` que además establece los parámetros de tiempos de ejecución y extras que recibe el servicio. El método `setServerUrlList(urls)` establece la lista de urls que deben ser verificadas por el servicio antes de publicar un valor en el parámetro observable del

ServerStatusManager. La última clase implementada, StartServiceReceiver extiende de BroadcastReceiver y se encarga de llamar nuevamente la función checkServerStatus() en el método onReceive() para garantizar la ejecución recurrente de la tarea.

La clase principal con la que interactúa la aplicación para empezar a monitorear el estado del servidor es ServerStatusManager la cual se inicia y se establece la lista de urls a ser verificadas una sola vez en la clase BaseApplication. Después se observa su parámetro serverState en cualquier actividad de la aplicación para mostrar un icono parpadeante que indica al usuario el estado actual del servidor utilizando un objeto creado con este propósito llamado ServerStatusObserver, como se muestra en la Figura 25.

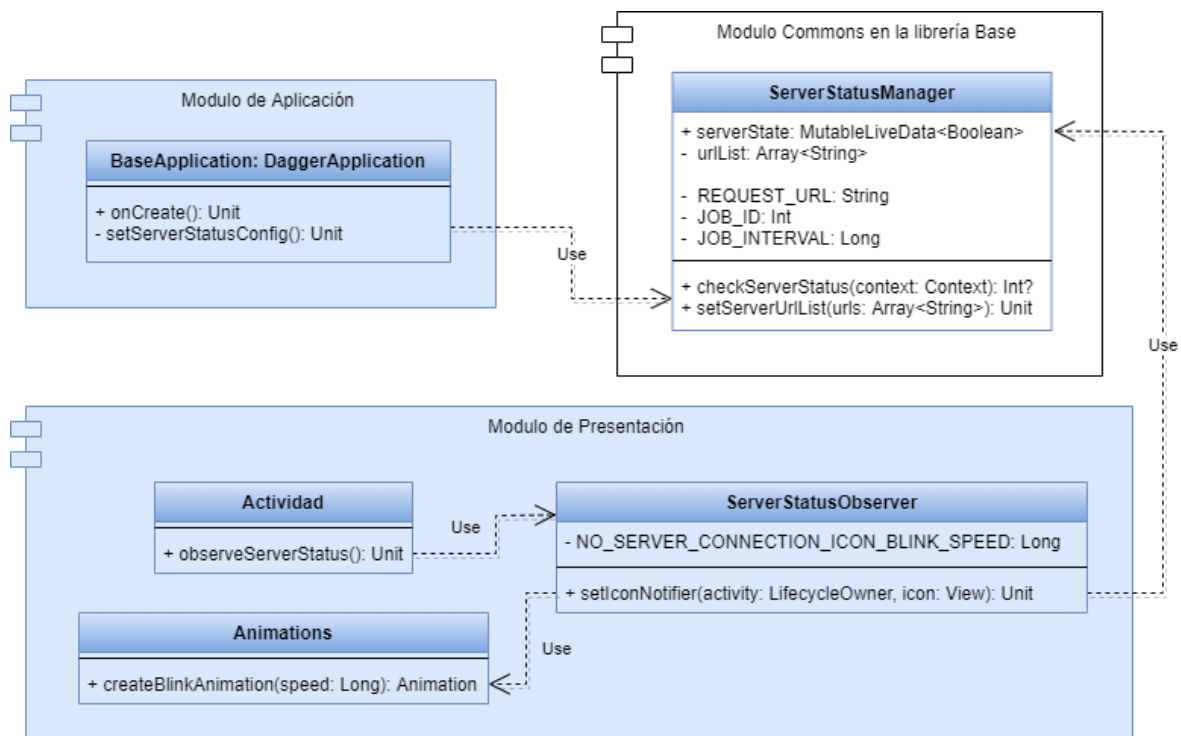


Figura 26. Diagrama del observador del estado del servidor. Fuente propia

KMENESESP/VMN-1178 BEHAVIOR FOR CONNECTION ICONS

#166 **MERGED** at 9d8558d `KMENESESP/VMN-1178_BEHA...` → `develop`

Revert Approve 1

[Overview](#) [Commits](#) [Activity](#)

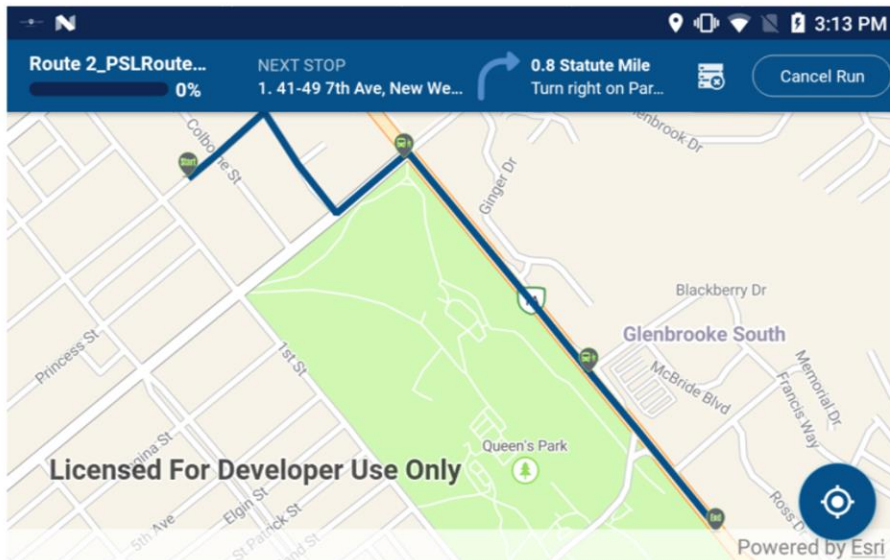
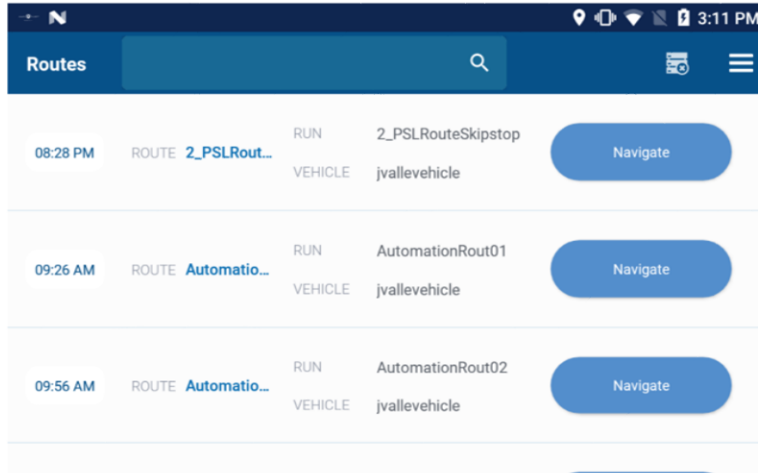
Author  Kevin Felipe Meneses Palta

VMN-1178

Reviewers 

Stop watching

Description **Evidence**



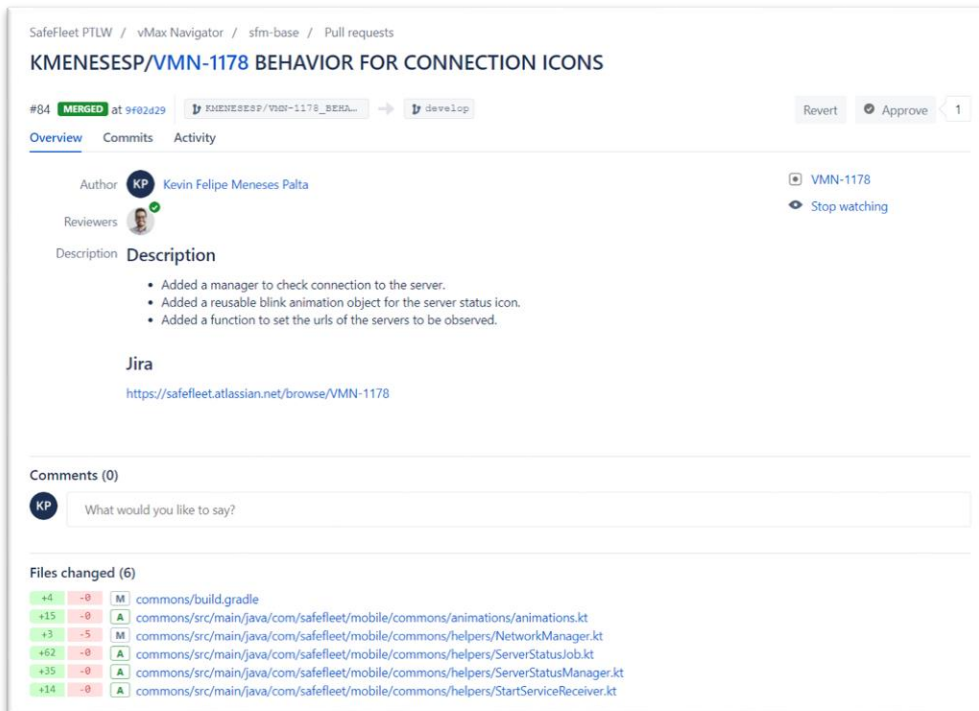
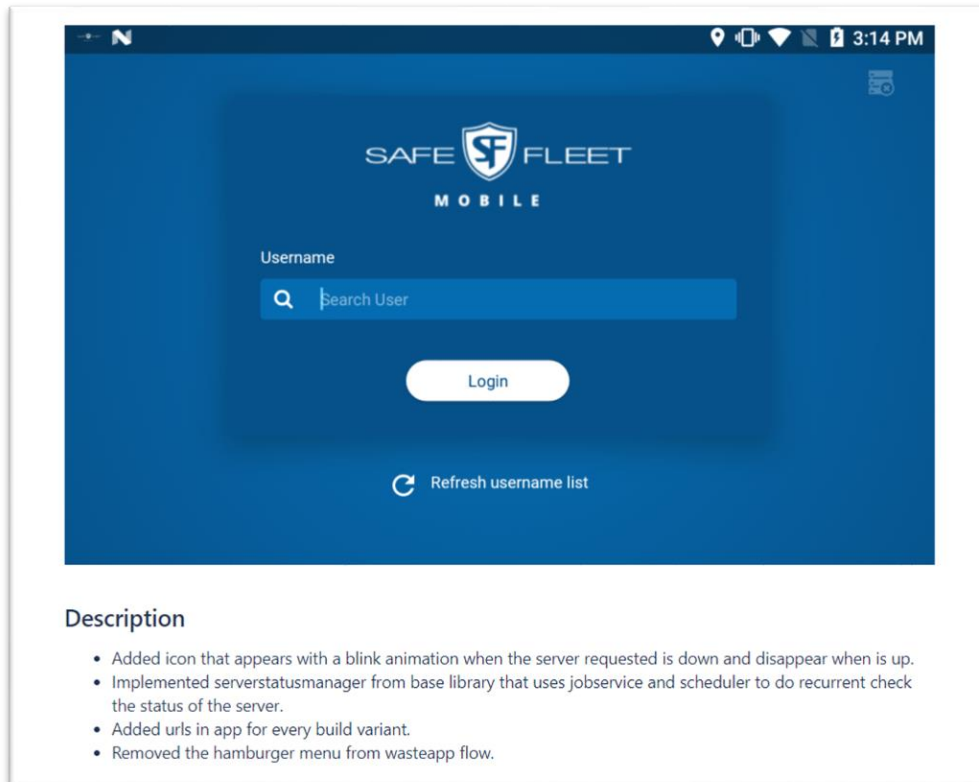


Figura 27. Pull requests del ticket VMN-1178. Fuente: Bitbucket de SafeFleet

5.9 Lector de eventos del brazo mecánico de parada

The image contains two screenshots of Jira tickets. The top screenshot is for ticket VMN-1438, titled "Stop arm". It shows a description: "Integration with stop arm sensor and send stop arm events to DDP." and a "TO DO" list with items like "STOP_ARM_DEPLOYED" and "STOP_ARM_RETRACTED". The bottom screenshot is for ticket VMN-1449, titled "Development". It shows a description field "Add a description..." and a comment section with a text input and a "Pro tip: press M to comment" message.

Figura 28. Tickets VMN-1438 y VMN-1449. Fuente: Jira de SafeFleet

Según los requerimientos de esta historia de usuario se debe interactuar con los puertos de propósito general de entrada/salida (GPIO) de la estación de acoplamiento de la tableta y leer el flujo de voltaje para conocer si el brazo mecánico de parada del autobús está desplegado o retraído. Para llevar a cabo esta función fue necesario enviar comandos a la tableta a través de una clase estática llamada ShellUtils, encargada de cumplir las funciones de shell o consola de comandos del

dispositivo retornando una clase llamada `CommandResult` que maneja la respuesta del resultado, además de un mensaje exitoso o un mensaje erróneo dependiendo del caso.

Asimismo, se creó una clase estática encargada de interactuar con ella, llamada `StopArmReader` con métodos que: establecen el estado y la dirección de los puertos GPIO: 2 de entrada y 2 de salida; e inicializan y detienen un método que lee constantemente el estado de los puertos de entrada y publica su estado en una variable observable que puede ser leída desde la aplicación. El diagrama de las clases se puede visualizar a continuación en la Figura 28.

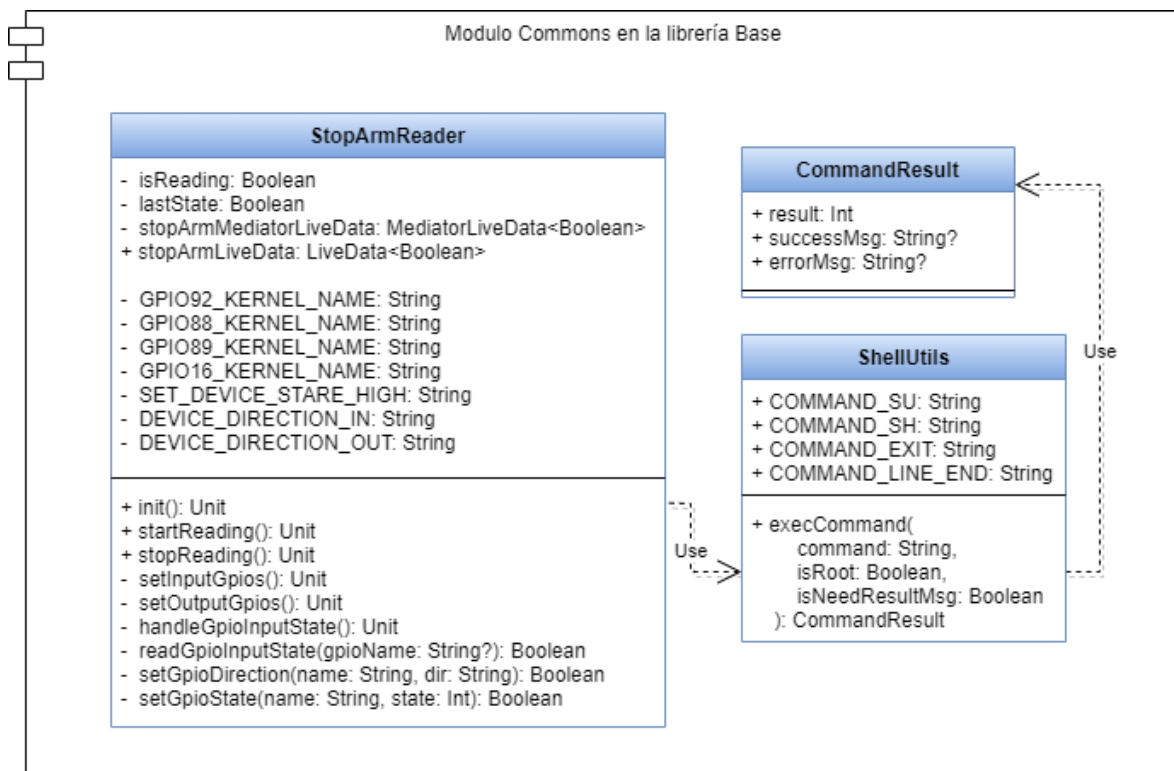


Figura 29. Diagrama del lector de brazo mecánico de parada. Fuente propia

Debido a que la señal del voltaje es intermitente, porque los puertos GPIO están en realidad conectados a las luces del brazo mecánico, el procedimiento implementado utiliza dos parámetros de tipo `CountDownTimer` con duración y tiempos de intervalo de lectura determinados para detectar cuando el brazo está desplegado o retraído para hacer un llamado a la función encargada de manejar los eventos del brazo

mecánico, además de iniciar la lectura del StopArmReader, observar el parámetro stopArmLiveData y ejecutar los CountdownTimer cuando el valor de lectura cambia.

SafeFleet PTLW / vMax Navigator / sfm-base / Pull requests

KMENESESP/VMN-1438 STOP ARM

#111 **MERGED** at f43bbe1 `KMENESESP/VMN-1438_STOP...` → `develop` Revert Approve 2

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta VMN-1438
Reviewers **SL** Stop watching

Description **Description**

- Modified gps data source implementation
- Added Shell Utils to send commands
- Added Stop Arm Reader to set gpio states and read input gpios

Jira
<https://safefleet.atlassian.net/browse/VMN-1438>

Comments (0)
KP What would you like to say?

Files changed (3)

+4	-9	M	avml/src/main/java/com/safefleet/mobile/avml/datasource/gps/GPSDataSourceImpl.kt
+208	-0	A	commons/src/main/java/com/safefleet/mobile/commons/helpers/ShellUtils.kt
+116	-0	A	commons/src/main/java/com/safefleet/mobile/commons/helpers/StopArmReader.kt

Figura 30. Pull request del ticket VMN-1438. Fuente: repositorio de SafeFleet en Bitbucket

5.10 Manejo de eventos de GPS, RFID y brazo mecánico de parada

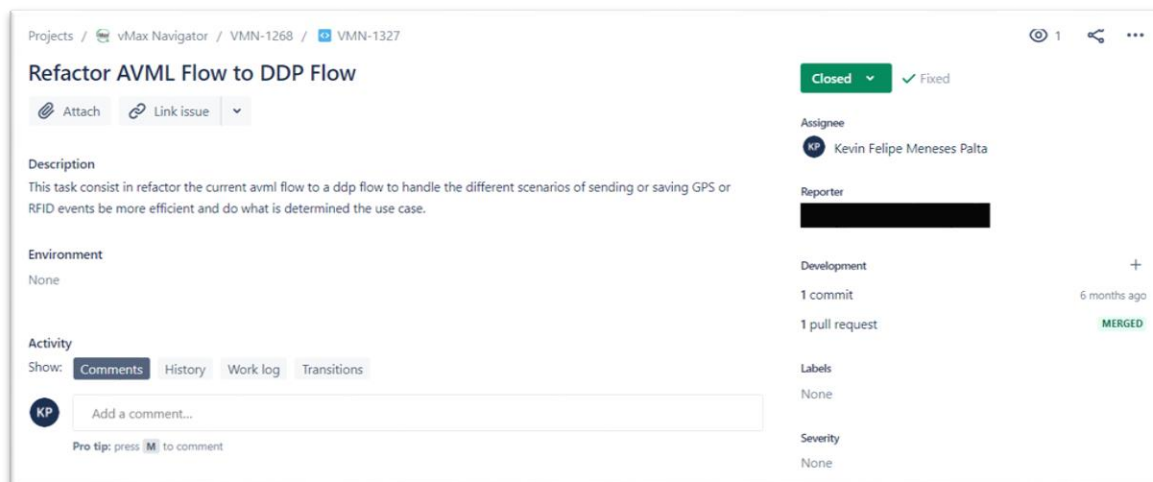


Figura 31. Ticket VMN-1327. Fuente: Jira de SafeFleet

El requerimiento de esta tarea fue realizar una mejor implementación del flujo encargado de manejar eventos de RFID y GPS, y agregar el manejo de eventos de brazo mecánico de parada. Para cumplir con este requerimiento la aplicación debe ser capaz de observar y manejar estos eventos continuamente, por lo que se creó un servicio que se ejecuta en segundo plano desde el momento en que se inicia la aplicación.

El servicio se encarga de crear una tarea recurrente en un hilo secundario que obtiene los datos de GPS (latitud y longitud) del dispositivo cada vez en un determinado intervalo de tiempo. En el caso de los eventos de RFID y brazo mecánico, el servicio se suscribe a los observables de ambas clases que se encargan de su lectura para obtener sus valores cada vez que un nuevo evento de cada tipo ocurre. Además, también se suscribe al observable de NetworkManager de manera indeterminada para solicitar al repositorio enviar los eventos pendientes cuando la conexión vuelve a estar disponible.

Los diferentes eventos pasan a través de los módulos hasta llegar al repositorio el cual se encarga de decidir qué hacer con ellos dependiendo de la conexión a Internet, ya sea guardarlos en la base de datos como pendientes para ser enviados

cuando haya conexión a Internet o enviarlos directamente al servidor. El diagrama de clases se puede visualizar en la Figura 31 a continuación.

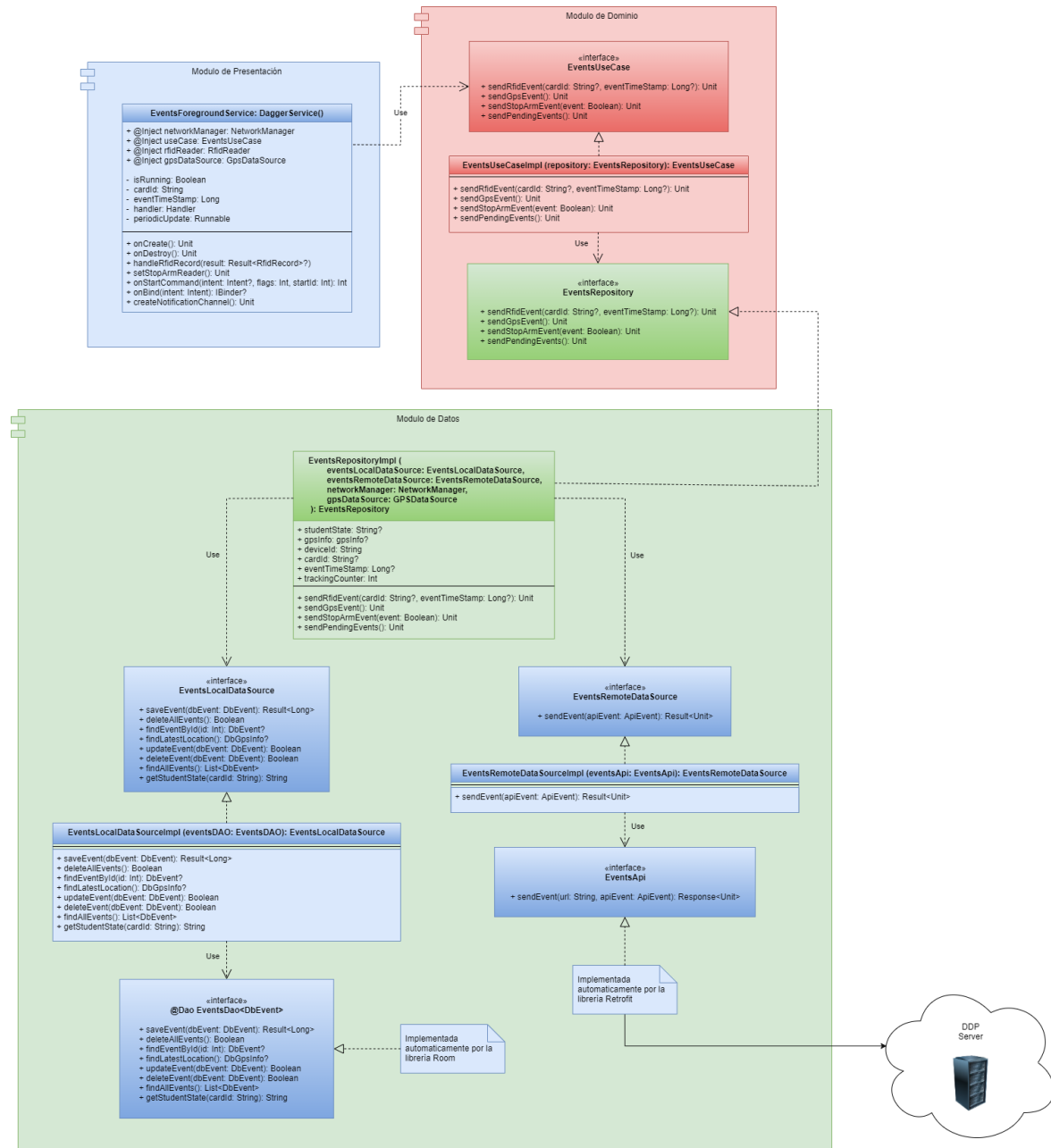


Figura 32. Diagrama de clases del flujo del servicio de manejo de eventos. Fuente propia

safeFleet PTLW / vMax Navigator / navigator-app / Pull requests

KMENESESP/VMN-1438 STOP ARM

#229 **MERGED** at 3309d79 `KMENESESP/VMN-1438_STOP...` → `develoP` Revert Approve 2

Overview Commits Activity

Author **KP** Kevin Felipe Meneses Palta VMN-1438
 Reviewers Stop watching

Description **Description**

- Made a refactor to foreground service and LogsErrorManager
- Injected LogsErrorManager
- Added folder to separe navigator and waste entities
- Changed the name of GpsForegroundService to AmvForegroundservice
- Added StopArmReader for stop arm events
- Added flow to send or save stop arm events depending on internet connection and send stop arm pending events when the connection comes back
- Added unit test functions for the stop arm events flow

Evidence

- When internet connection
 - 2020-04-24 10:54:23.437 17464-18334/com.safefleet.navigator.debug D/DdpRepositoryImpl: Stop arm event successfully sent to DDP
 - 2020-04-24 10:54:34.308 17464-19310/com.safefleet.navigator.debug D/DdpRepositoryImpl: Stop arm event successfully sent to DDP

- When there is no internet and then comes back
 - 2020-04-24 10:53:01.289 17464-18334/com.safefleet.navigator.debug D/DdpLocalDataSourceImpl: Stop arm event 1 saved successfully
 - 2020-04-24 10:53:14.796 17464-18334/com.safefleet.navigator.debug D/DdpLocalDataSourceImpl: Stop arm event 2 saved successfully
 - 2020-04-24 10:53:15.852 17464-18334/com.safefleet.navigator.debug D/DdpRepositoryImpl: Stop arm pending event successfully sent to DDP
 - 2020-04-24 10:53:16.060 17464-18334/com.safefleet.navigator.debug D/DdpRepositoryImpl: Stop arm pending event successfully sent to DDP

Figura 33. Pull request del ticket VMN-1438. Fuente: repositorio de SafeFleet en Bitbucket

5.11 Pantalla de navegación para cada negocio

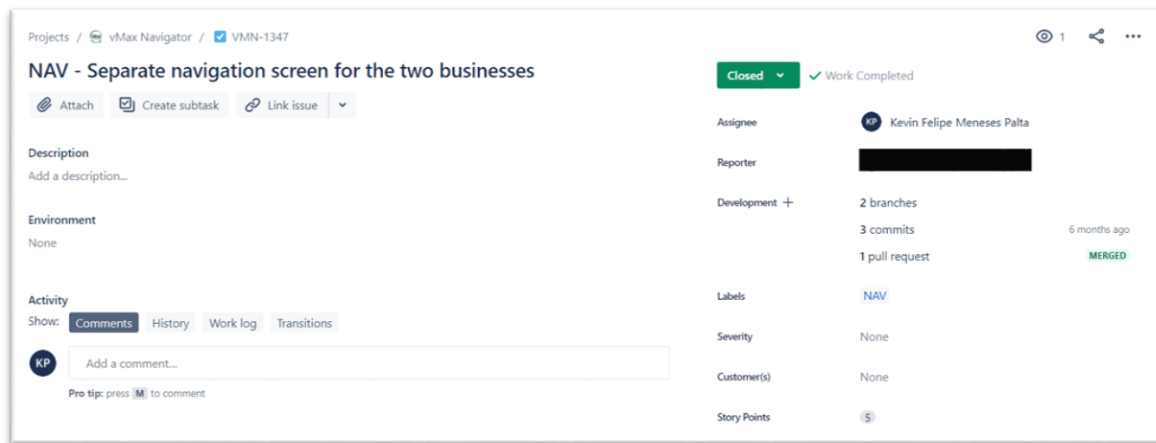


Figura 34. Ticket VMN-1347. Fuente: Jira de SafeFleet

Con el objetivo de reutilizar la mayor cantidad de código posible la pantalla de navegación del bus escolar y del camión de desechos comparten la misma interfaz de usuario y un código que se encarga de manejar sus elementos dependiendo del negocio. Sin embargo, la utilización de condicionales para habilitar y deshabilitar las vistas y determinar las funcionalidades de cada negocio no es la manera óptima de manejar su lógica, por lo cual se determinó realizar una refactorización de la pantalla de navegación tratando de manejar el código compartido de una forma óptima y separando sus características específicas sin necesidad de hacer validaciones todo el tiempo.

La actividad de navegación llamada `NavigationActivity` que contaba con un archivo de diseño llamado `activity_navigation` fue separada en dos clases con un diseño para cada una y se creó una nueva clase padre con las funcionalidades compartidas de la que la navegación de buses escolares y la navegación de camiones de desechos pudieran heredar como: la ubicación en tiempo real, el trazado de rutas, las instrucciones de navegación, entre otras. A continuación, se muestra el diagrama de clases de la pantalla de navegación:

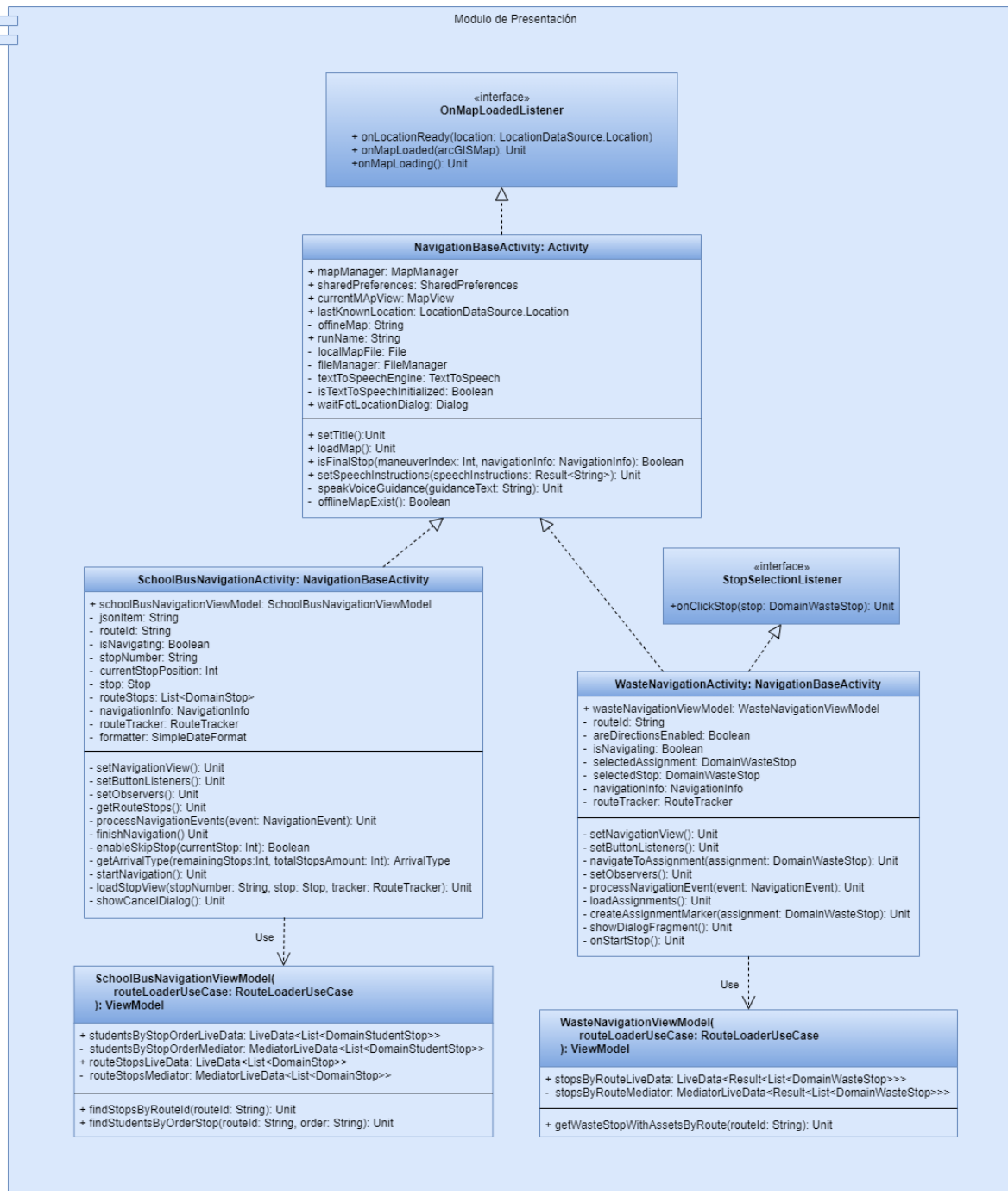






Figura 35. Diagrama de clases de la pantalla de navegación para los diferentes negocios. Fuente propia

SafeFleet PTLW / vMax Navigator / navigator-app / Pull requests

KMENESESP/VMN-1347 SEPARATE NAVIGATION SCREEN FOR THE 2 BUSINESSES

KP KMENESESP/VMN-1347_SEPAR... → develop **MERGED**   ... 

Created 2020-04-02 · Last updated 2020-04-03

 **Merged pull request**
Merged in KMENESESP/VMN-1347_SEPARATE_NAVIGATION_SCREEN_FOR_THE_2_BUSINESSES (pull request #201)
[9cd704b](#) · [Kevin Felipe Meneses Palta](#) · 2020-04-03

Description

- Separated the navigation activity for both navigator and wasteapp flow.
- Added ServerStatusObserver to handle icon notifier
- Fixed an issue when the internet is off the app crash due to exception at trying to reach message receiver url.
- Created an activity to extend and moved some files to new packages.
- Created a constants object to manage constants.

Jira

<https://safefleet.atlassian.net/browse/VMN-1347>

Figura 36. Pull request del ticket VMN-1347. Fuente: repositorio de SafeFleet en Bitbucket

CAPÍTULO VI: RESULTADOS

6.1 Caso de estudio

Con el objetivo de verificar las funcionalidades desarrolladas en la aplicación en el transcurso de la pasantía, se realizó un caso de estudio en el municipio de Copacabana a las afueras de la ciudad de Medellín, en donde un conductor haciendo uso de los dispositivos de la empresa: tablet VT7 y estación de acoplamiento, realizó una ruta con paradas, asignada a su dispositivo, en la que se cubren los requerimientos funcionales planteados en la primera parte de este documento.

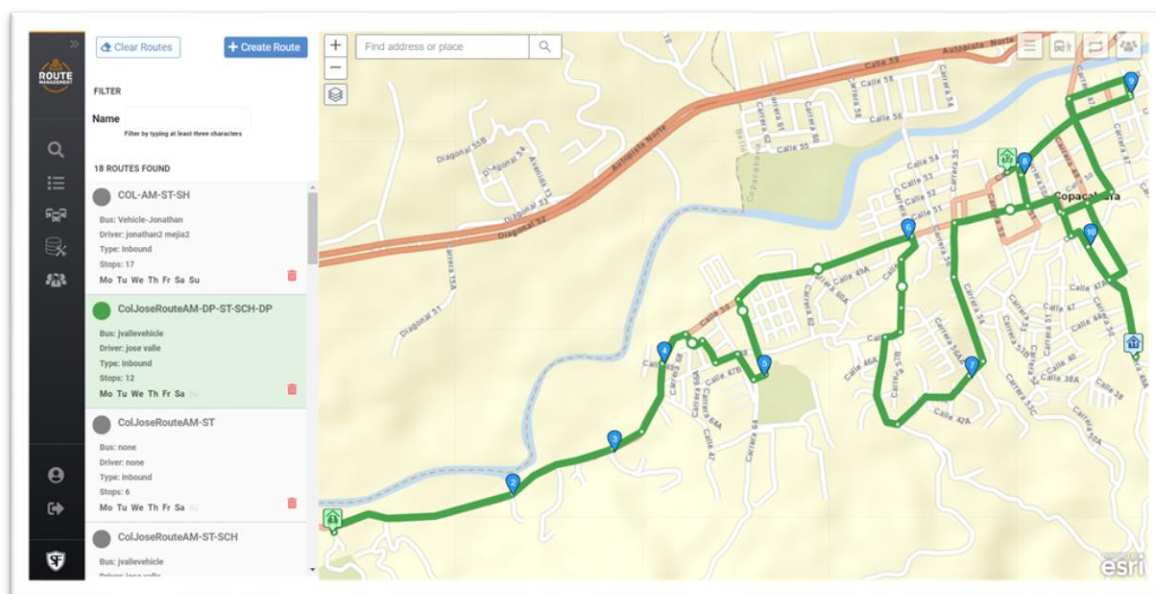


Figura 37. Ruta trazada del recorrido. Fuente: página de rutas de SafeFleet

La ruta asignada para el desarrollo del caso de estudio llamada “ColJoseRouteAM-DP-ST-SCH-DP” consta de un tramo de 7.39 kilómetros con 12 paradas incluyendo el punto de partida y el destino final las cuales son:

- **Punto de partida:** Doresky, Calle 46, Copacabana, Antioquia
- **Parada 2:** Las Canteras, Calle 46, Copacabana, Antioquia
- **Parada 3:** La Cocina de Teo, Calle 46, Copacabana, Antioquia

- **Parada 4:** El Remanso, Carrera 69, Copacabana, Antioquia
- **Parada 5:** El Pedregal, Calle 47A 63 1-49, Copacabana, Antioquia
- **Parada 6:** Asunción (Alta y Baja) Guadalajara, Calle 50 57 1-99, Copacabana, Antioquia
- **Parada 7:** Las Vegas (Alta y Baja), Calle 43 56 51-149, Copacabana, Antioquia
- **Parada 8:** Simón Bolívar, Carrera 51, Copacabana, Antioquia
- **Parada 9:** Pedrera Mirador Azul, Carrera 46 52 2-98, Copacabana, Antioquia
- **Parada 10:** Barrio Obrero, Carrera 49A 47A 74-144, Copacabana, Antioquia
- **Parada 11:** Cristo Rey Cuenca Verde, Carrera 49A 39 1-99, Copacabana, Antioquia
- **Destino final:** Simón Bolívar, Calle 52 49 2-98, Copacabana, Antioquia

A continuación, se presentan las figuras correspondientes a los segmentos de cada parada:

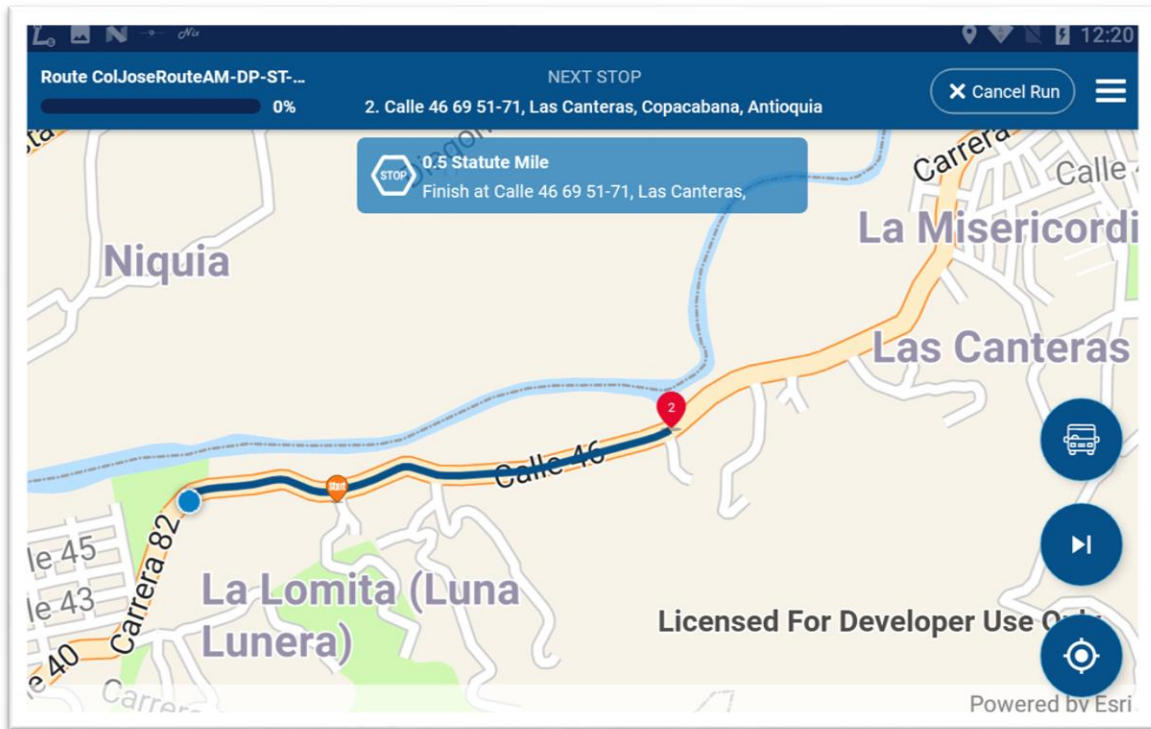


Figura 38. Segmento 1: punto de partida y parada 2. Fuente: aplicación Navigator

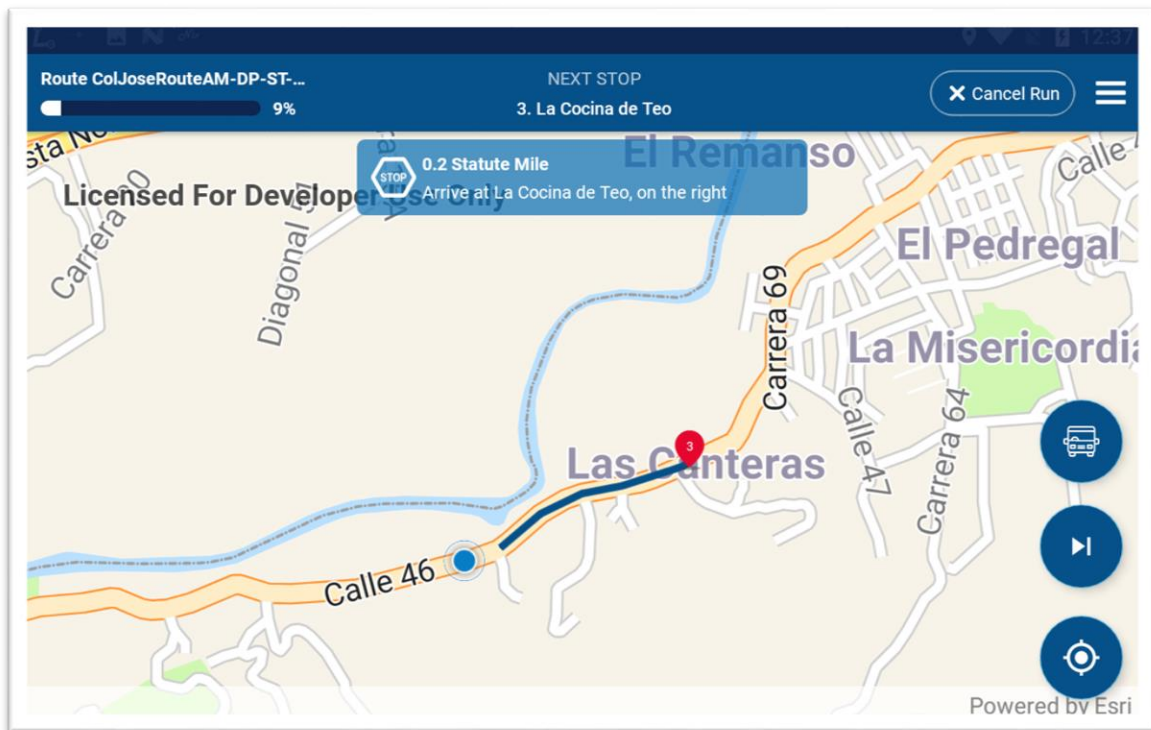


Figura 39. Segmento 2: parada 3. Fuente: aplicación Navigator

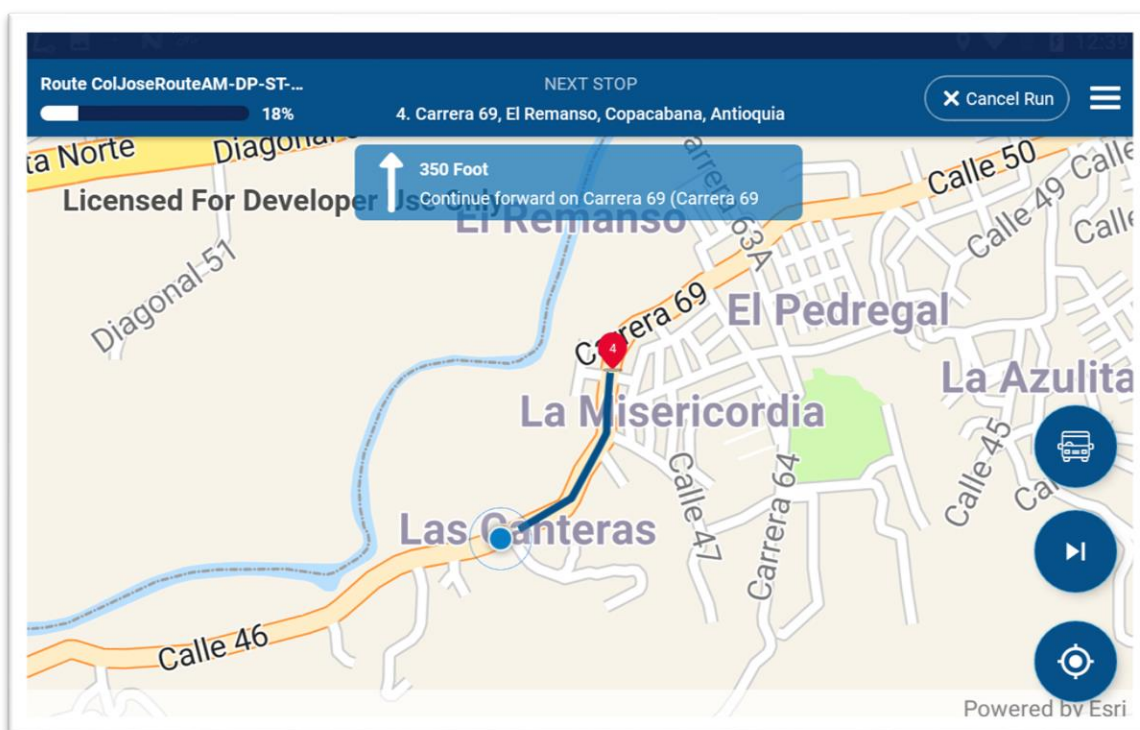


Figura 40. Segmento 3: parada 4. Fuente: aplicación Navigator

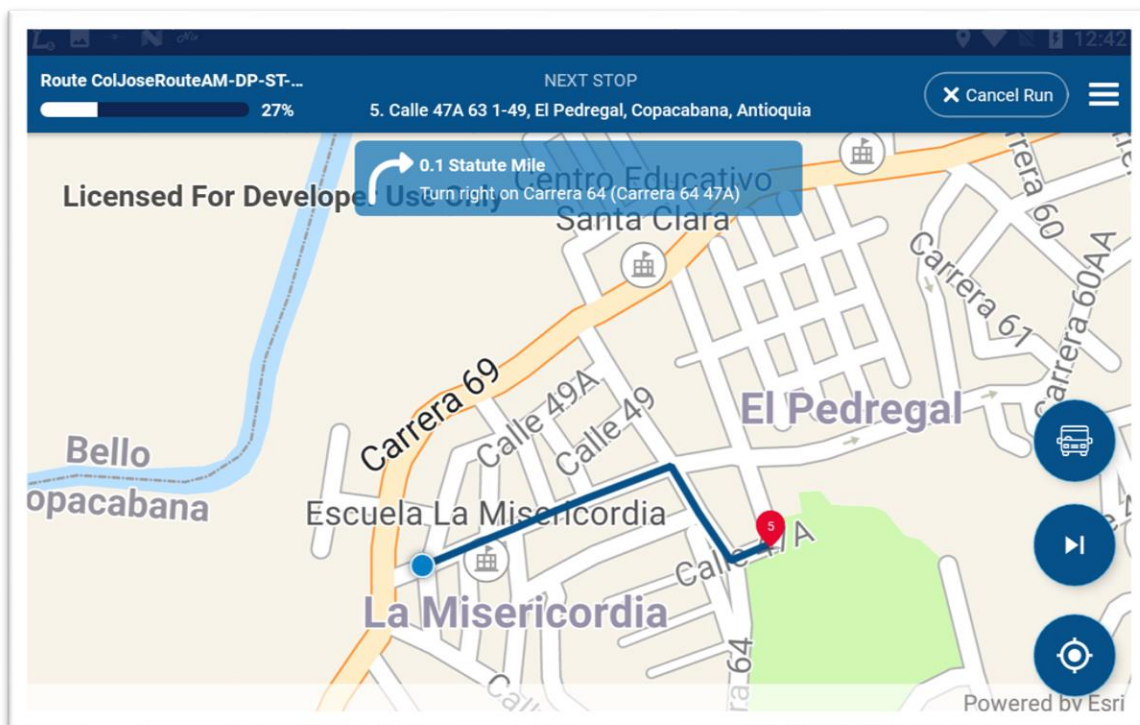


Figura 41. Segmento 4: parada 5. Fuente: aplicación Navigator

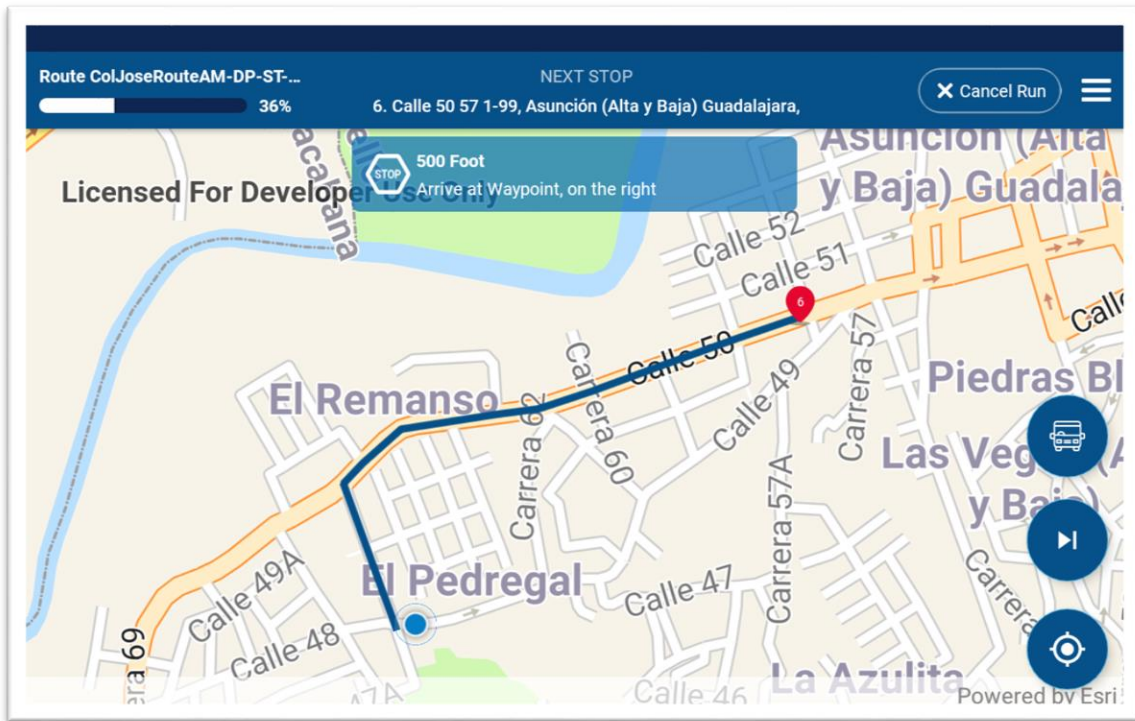


Figura 42. Segmento 5: parada 6. Fuente: aplicación Navigator

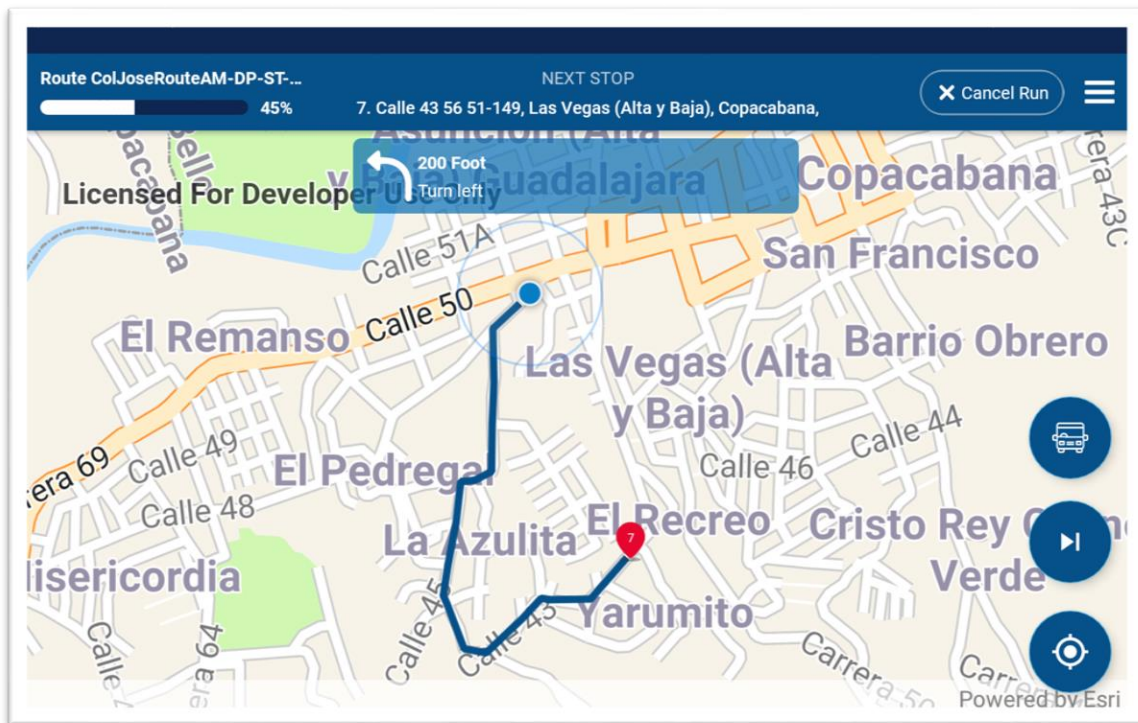


Figura 43. Segmento 6: parada 7. Fuente: aplicación Navigator

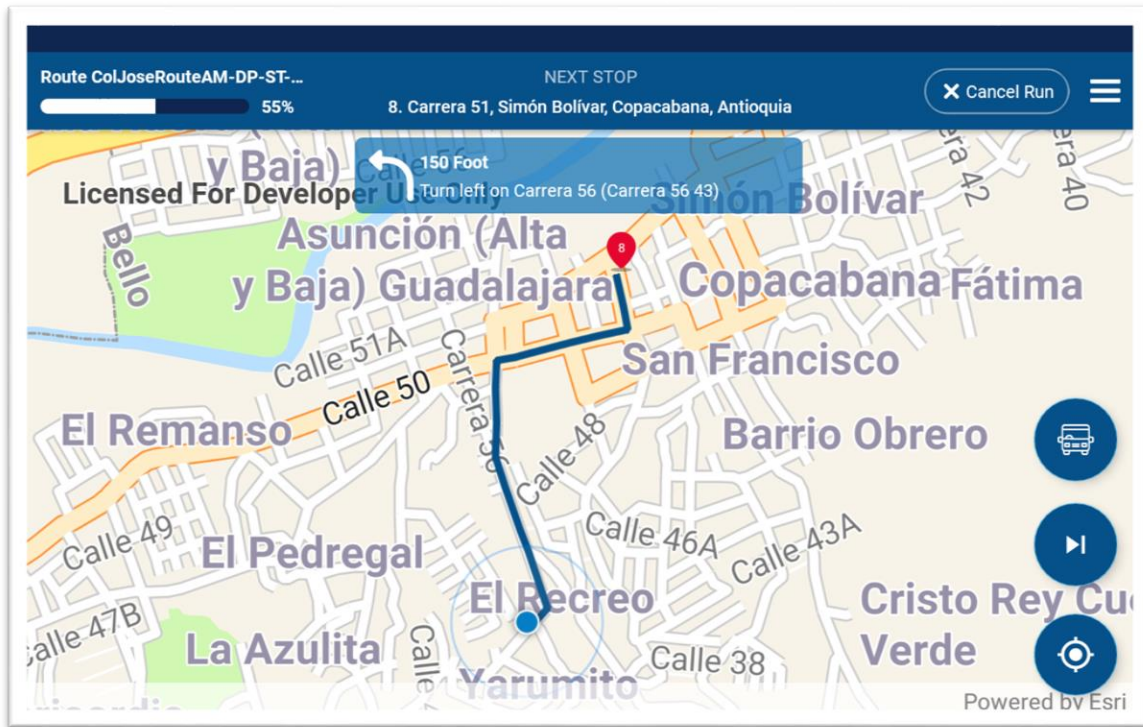


Figura 44. Segmento 7: parada 8. Fuente: aplicación Navigator

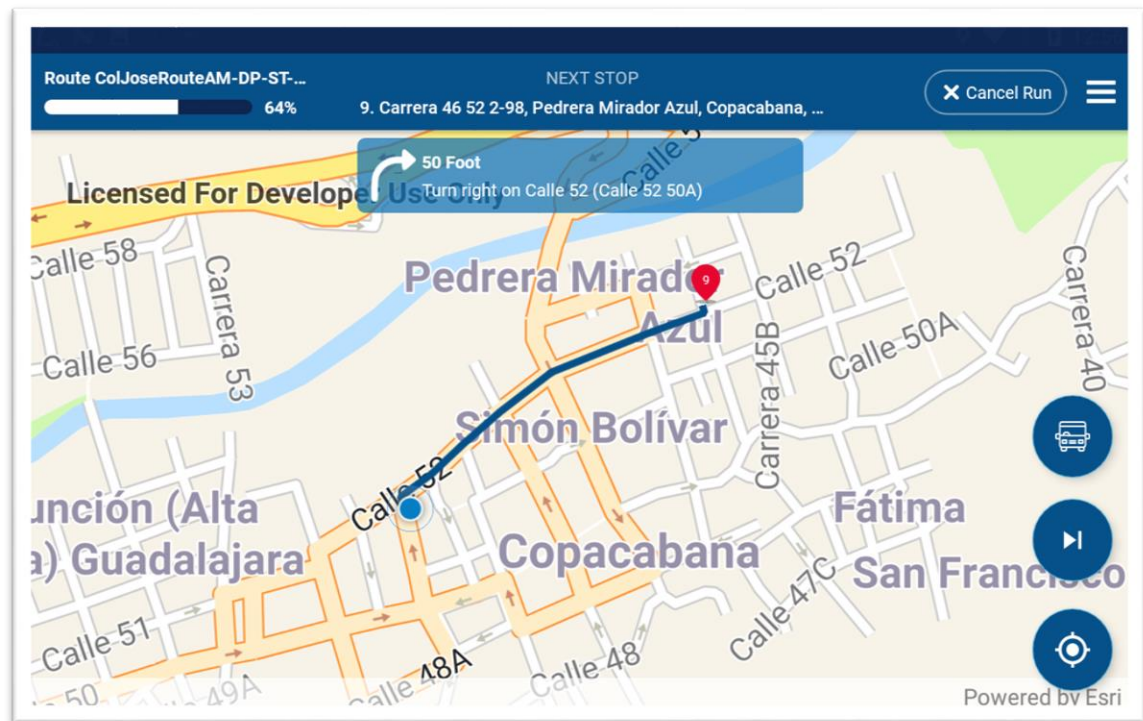


Figura 45. Segmento 8: parada 9. Fuente: aplicación Navigator

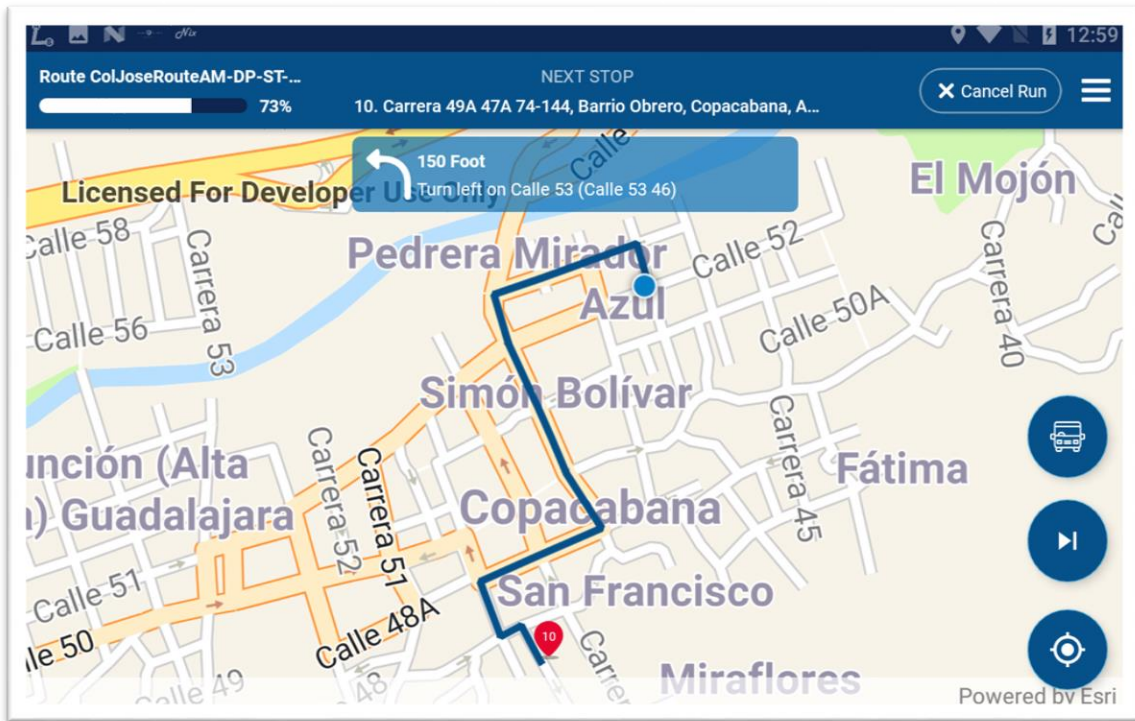


Figura 46. Segmento 9: parada 10. Fuente: aplicación Navigator

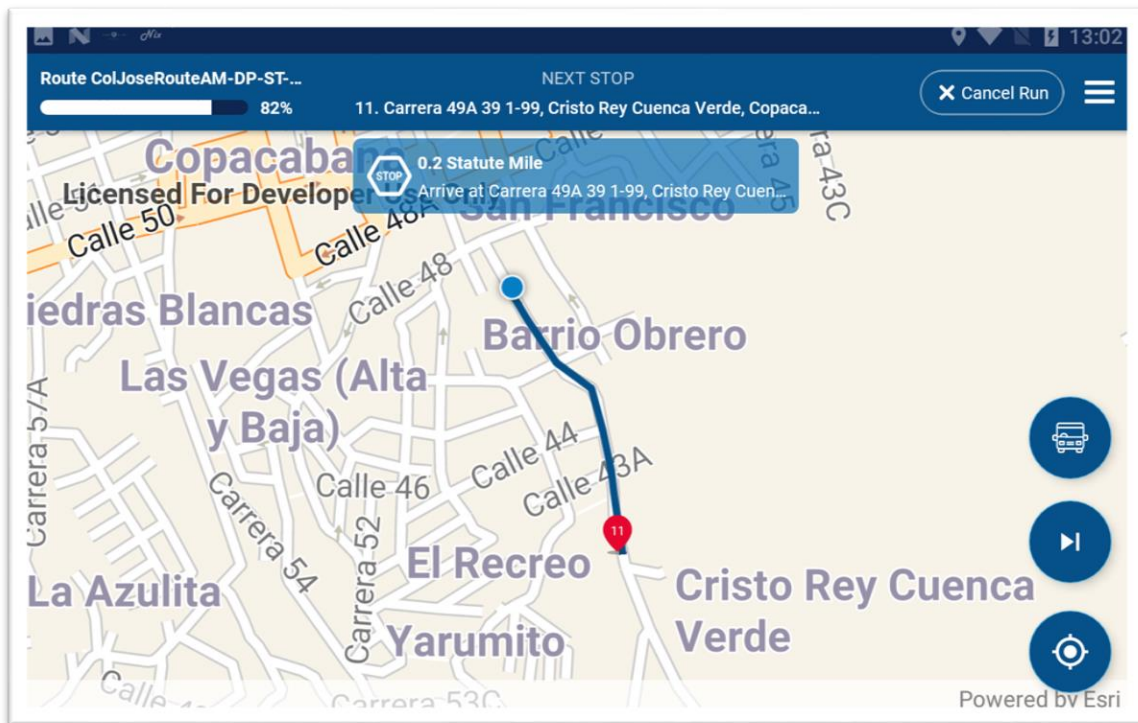


Figura 47. Segmento 10: parada 11. Fuente: aplicación Navigator

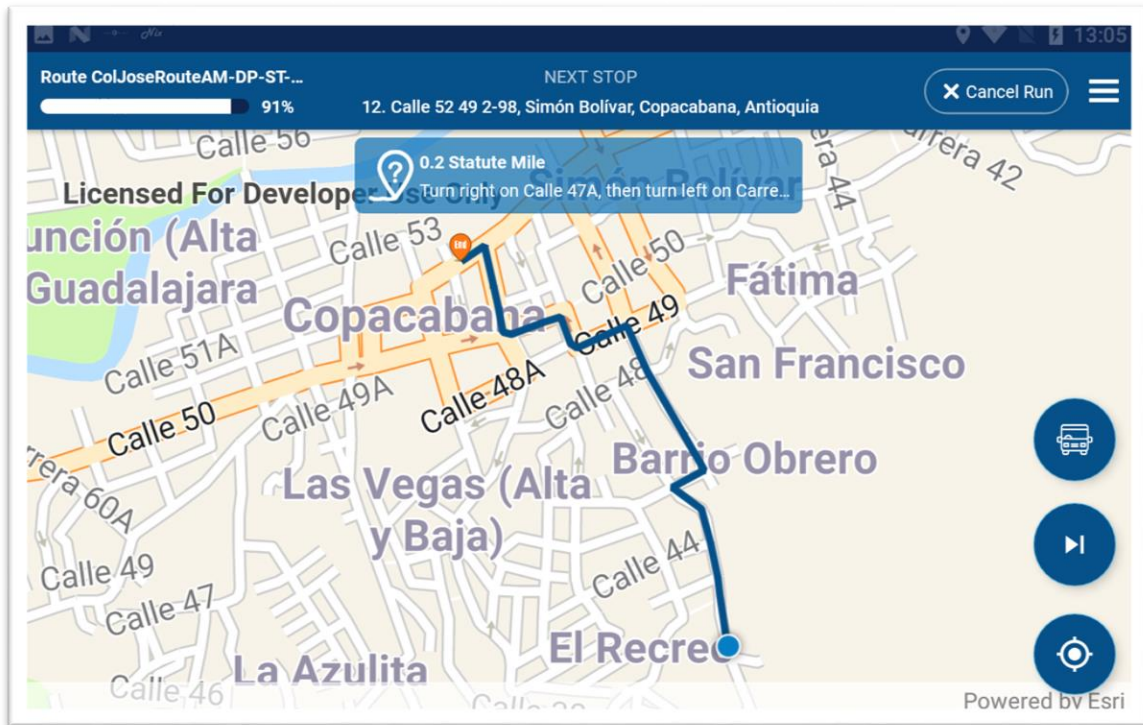


Figura 48. Segmento 11: destino final. Fuente: aplicación Navigator

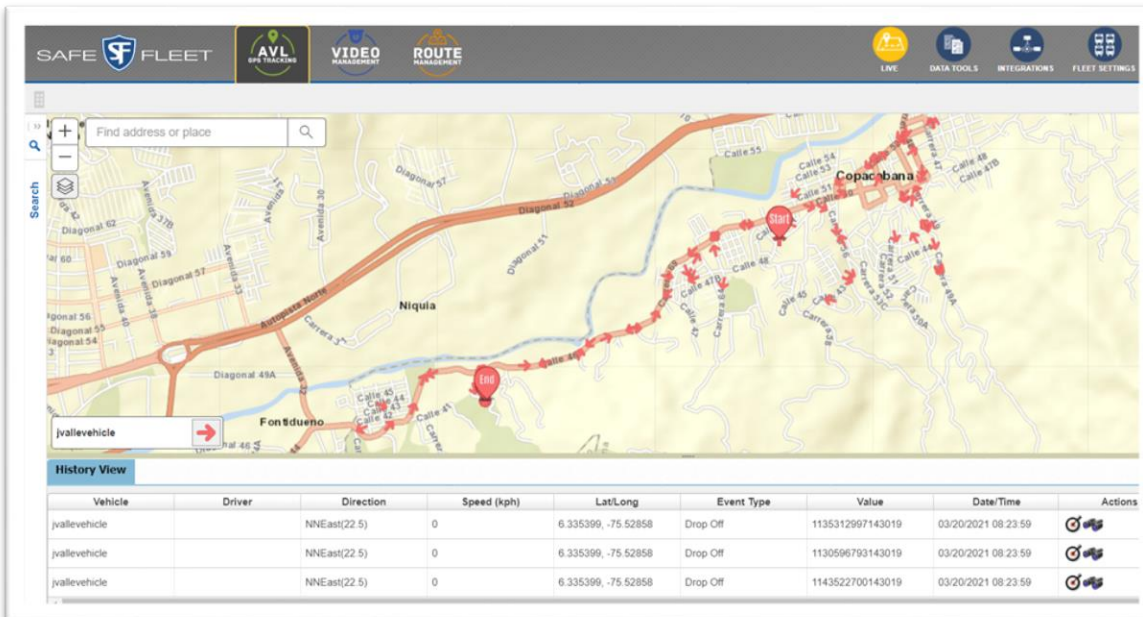


Figura 49. Seguimiento de la ruta realizada. Fuente: página de seguimiento de SafeFleet

6.2 Resultados de las pruebas:

RF-01: Barra de búsqueda personalizada. En ambas aplicaciones WasteApp y Navigator, en la lista de rutas, al seleccionar la barra de búsqueda el conductor fue capaz de:

1. Ingresar el nombre de la ruta asignada: “ColJoseRouteAM-DP-ST-SCH-DP” y obtuvo el resultado esperado inmediatamente:

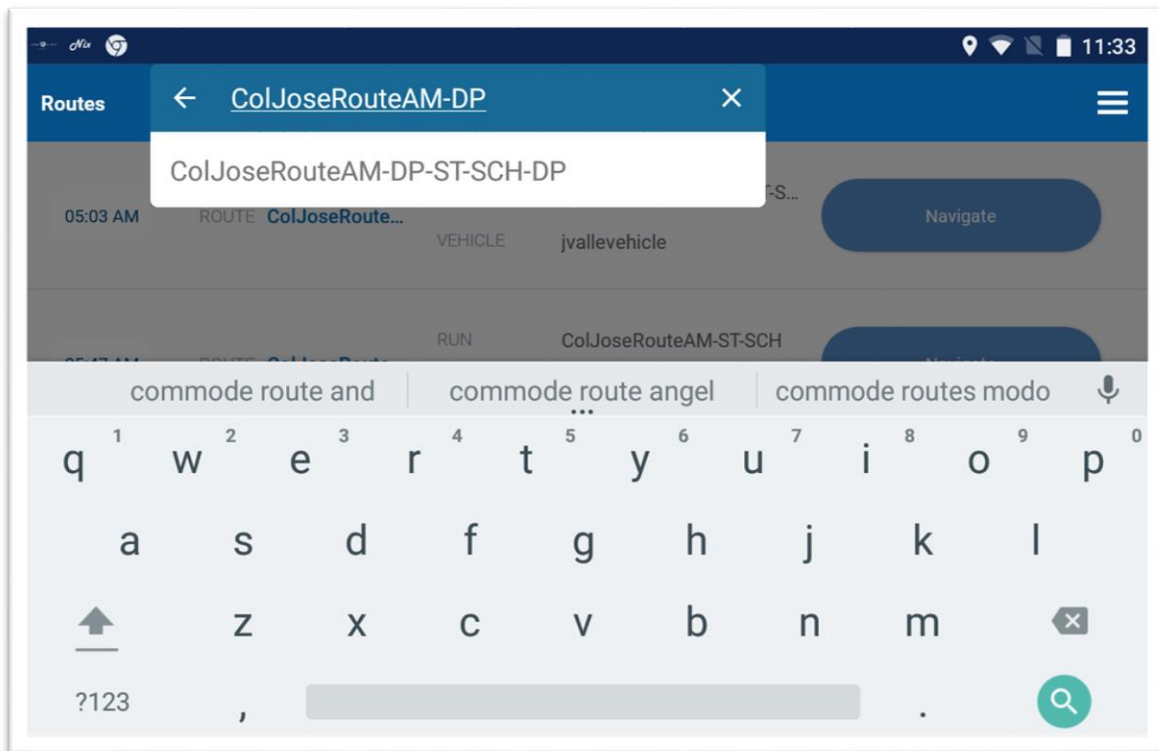


Figura 50. Prueba 1 de RF-01. Fuente: aplicaciones Navigator y WasteApp

2. Seleccionar la ruta de la lista de resultados y ver un diálogo de confirmación para acceder a la pantalla de navegación el mapa con la ruta trazada desde el inicio hasta la primera parada:

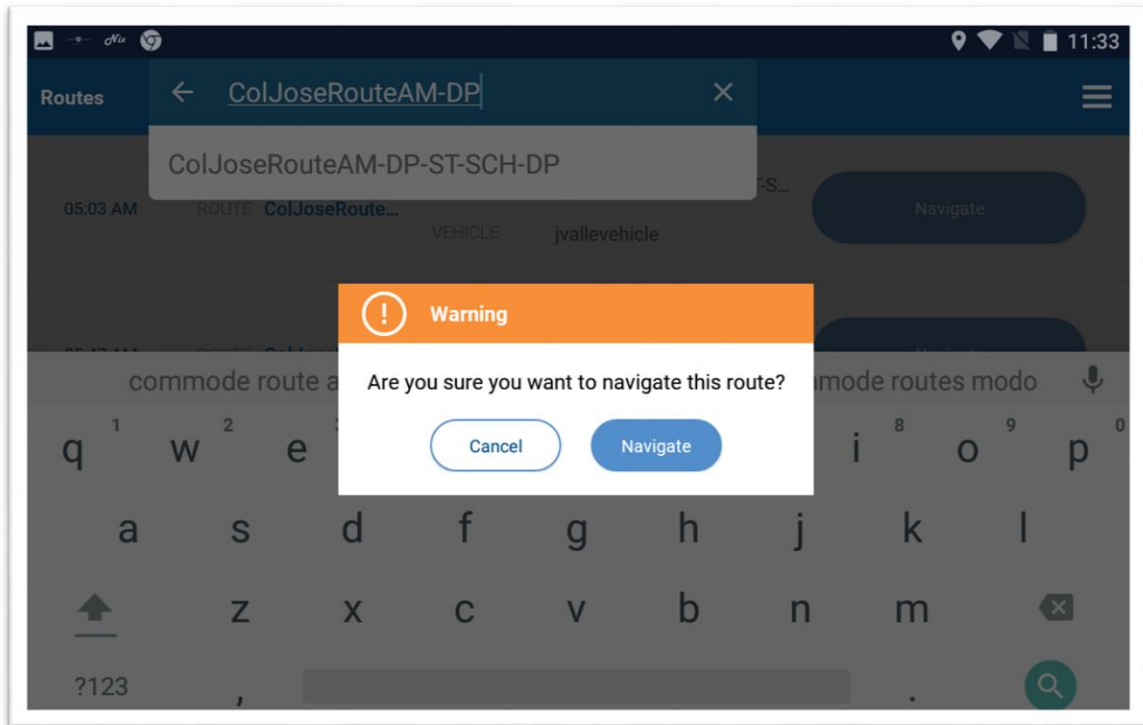


Figura 51. Prueba 2 de RF-01: confirmación. Fuente: aplicaciones Navigator y WasteApp

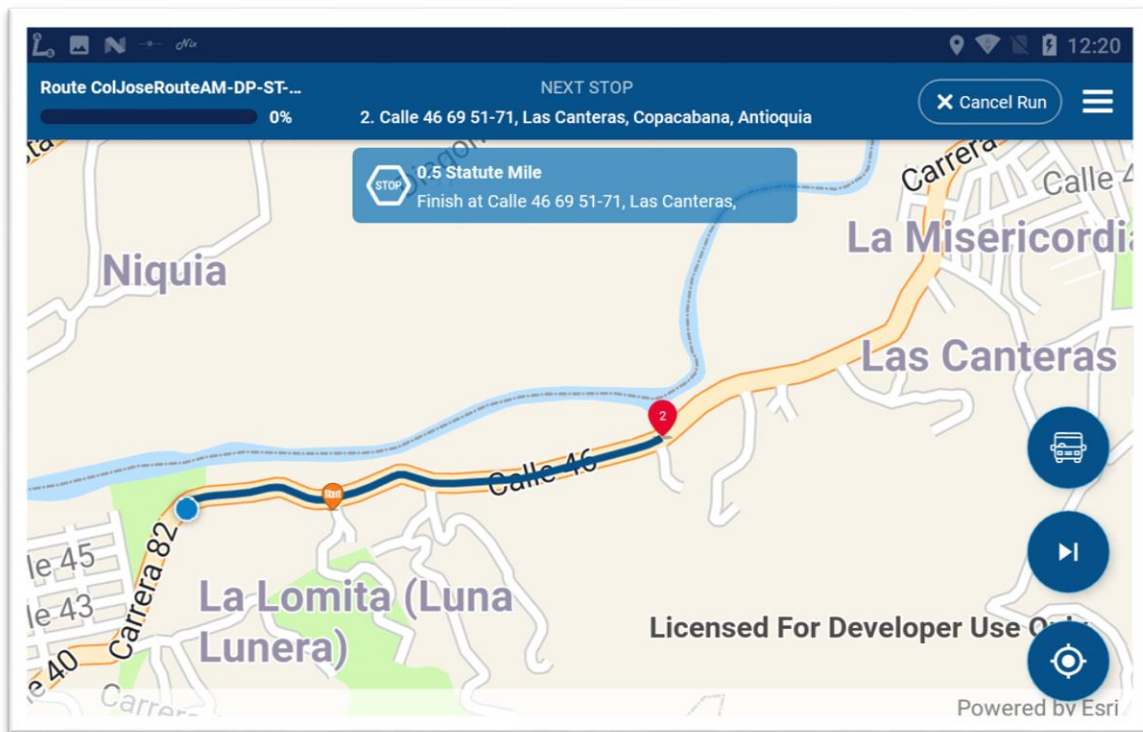


Figura 52. Prueba 2 de RF-01: navegación. Fuente: aplicaciones Navigator y WasteApp

3. Cuando ingresó una ruta no existente, ej: “medellinRoute”, el resultado de la búsqueda fue: “No results found”:

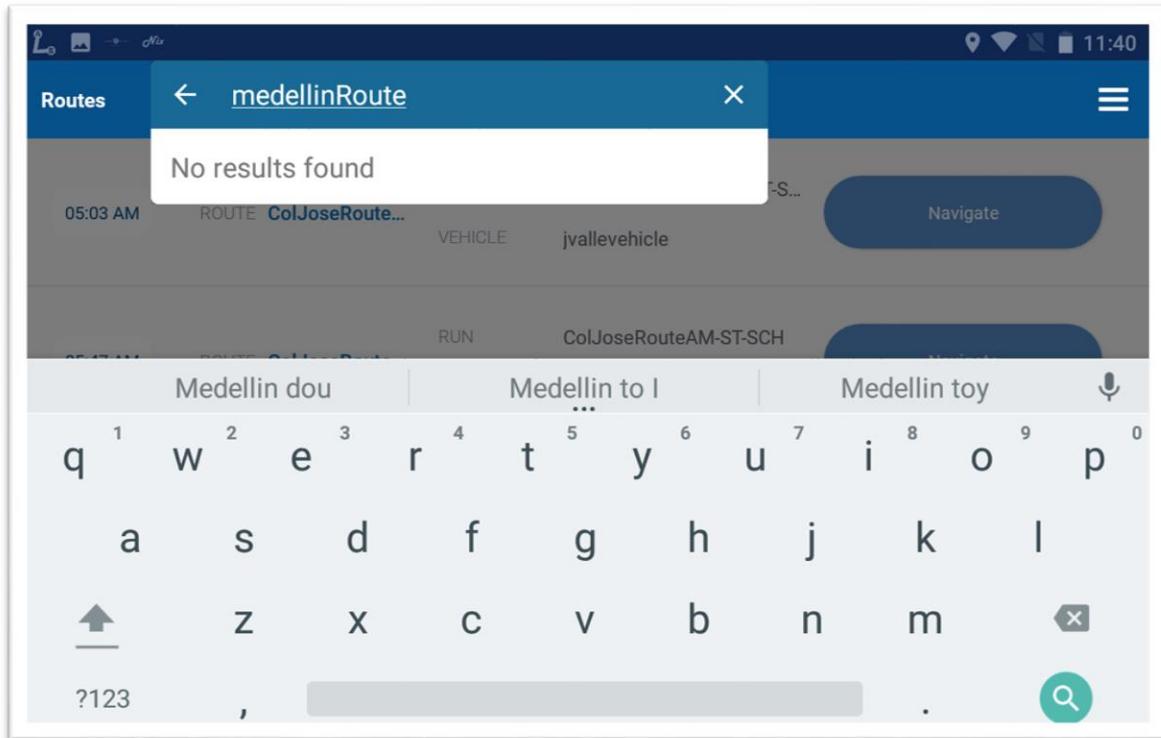


Figura 53. Prueba 3 de RF-01. Fuente: aplicaciones Navigator y WasteApp

RF-02: Guardar rutas. En la aplicación WasteApp (vehículos de desechos) en la lista de rutas, al ingresar el nombre de una ruta inexistente el conductor fue capaz de:

1. Ver el resultado “No results found” y una opción adicional con el texto: “Do you want to create a route with the name ... ?” con el nombre de la ruta ingresada y un botón con el texto “Yes, Create Route”:

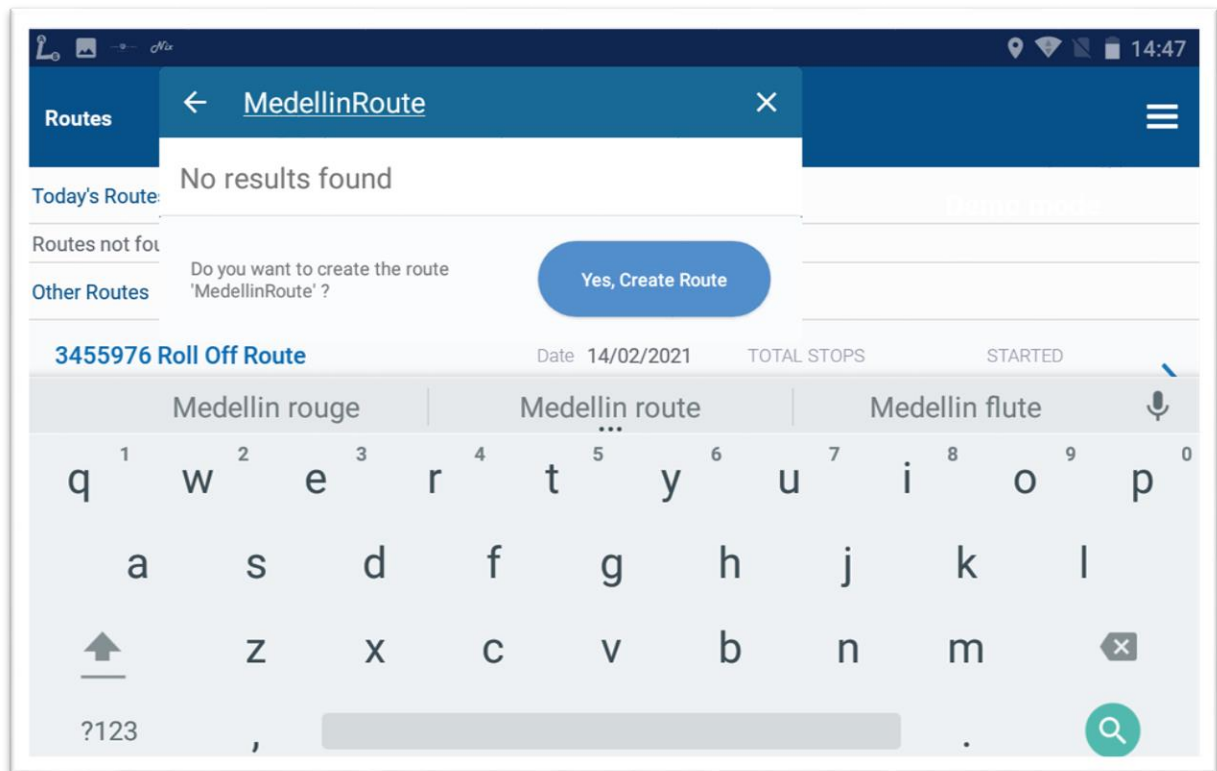


Figura 54. Prueba 1 de RF-02. Fuente: aplicación WasteApp

2. Seleccionar el botón con el texto “Yes, Create Route”, el cual guardó la ruta como último registro en la base de datos y la mostró como primer resultado en la lista de rutas:

autoid	autoid	id	name	driver	vehicle	version	routeLayerId	date	detailedNote	started	totalStops	remainingStops
1	1	3455976	3455976 Roll Off Route	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16132096...	Note 3455976 Roll Off Route	7	7	7
2	2	3455982	3455982 Roll Off Route	Samuel Johnson	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16133760...		3	3	3
3	3	3455977	3455977 Cart Maintenance r...	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16132896...	Before starting the route, please call dispatcher.	4	4	4
4	4	3455978	3455978 Commercial Route	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16132896...	Note 3455978 Commercial Route	7	7	7
5	5	3455979	3455979 Commercial Route	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16133760...		4	4	4
6	6	3455981	3455981 Commercial Route	Samuel Johnson	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16134624...		3	3	3
7	7	3455983	3455983 Residential Route	Richard Henry ...	TR-306	100	1ed2b0cd0f524e8b671dfc71e54...	16132096...	Note 3455983 Residential Route	7	7	7
8	8	3455984	3455984 Residential Route	Samuel Johnson	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16134624...		3	3	3
9	9	3455985	3455985 Residential Route	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16133760...		4	4	4
10	10	3455986	3455986 Cart Maintenance c...	Richard Henry ...	TR-306	100	3ed9b0cd0f524e8b671dfc71e57...	16133760...	Before starting the route, please call dispatcher.	4	4	4
11	11	3455987	3455987 Cart Maintenance c...	Richard Henry ...	TR-306	100	4f05b0cd0f72986c7aac70e59...	16134624...	Before starting the route, please call dispatcher.	2	2	2
12	12	MEDELLI...	MEDELLINROUTE		TR-306	0		16163563...		0	0	0

Figura 55. Prueba 2 de RF-02: base de datos de la aplicación. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

Route Name	Date	TOTAL STOPS	STARTED
MEDELLINROUTE		0	No
3455976 Roll Off Route	14/02/2021	7	No
<i>Note 3455976 Roll Off Route</i>			
3455977 Cart Maintenance route	14/02/2021	4	No
<i>Before starting the route, please call dispatcher.</i>			
3455978 Commercial Route	14/02/2021	7	No
<i>Note 3455978 Commercial Route</i>			

Figura 56. Prueba 2 de RF-02: lista de rutas. Fuente: aplicación WasteApp

RF-03: Observar el estado de conexión a Internet. En ambas aplicaciones WasteApp y Navigator cuando no hubo conexión a Internet, el conductor fue capaz de:

1. Ver un diálogo informativo sobre el estado de desconexión del dispositivo en la pantalla de inicio de la aplicación:

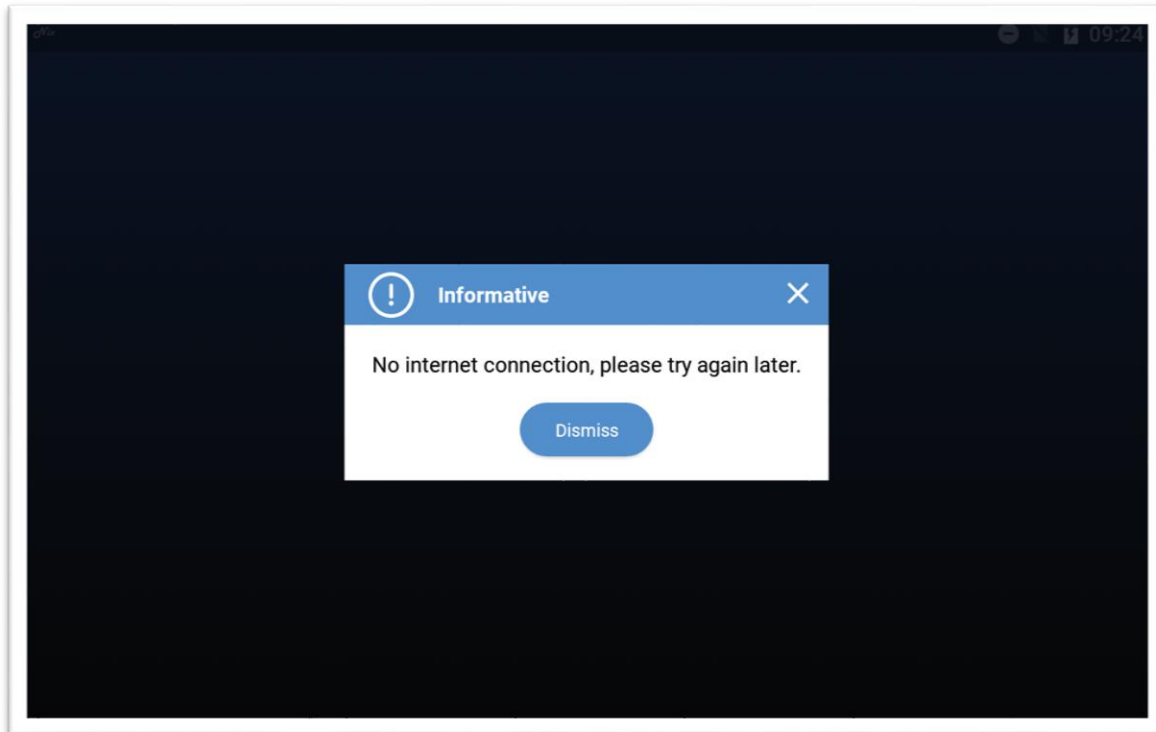


Figura 57. Prueba 1 de RF-03. Fuente: aplicación Navigator

2. Ver el texto: “No Internet connection” en el resultado de la barra de búsqueda, cuando ingresó a la aplicación y se perdió la conexión:

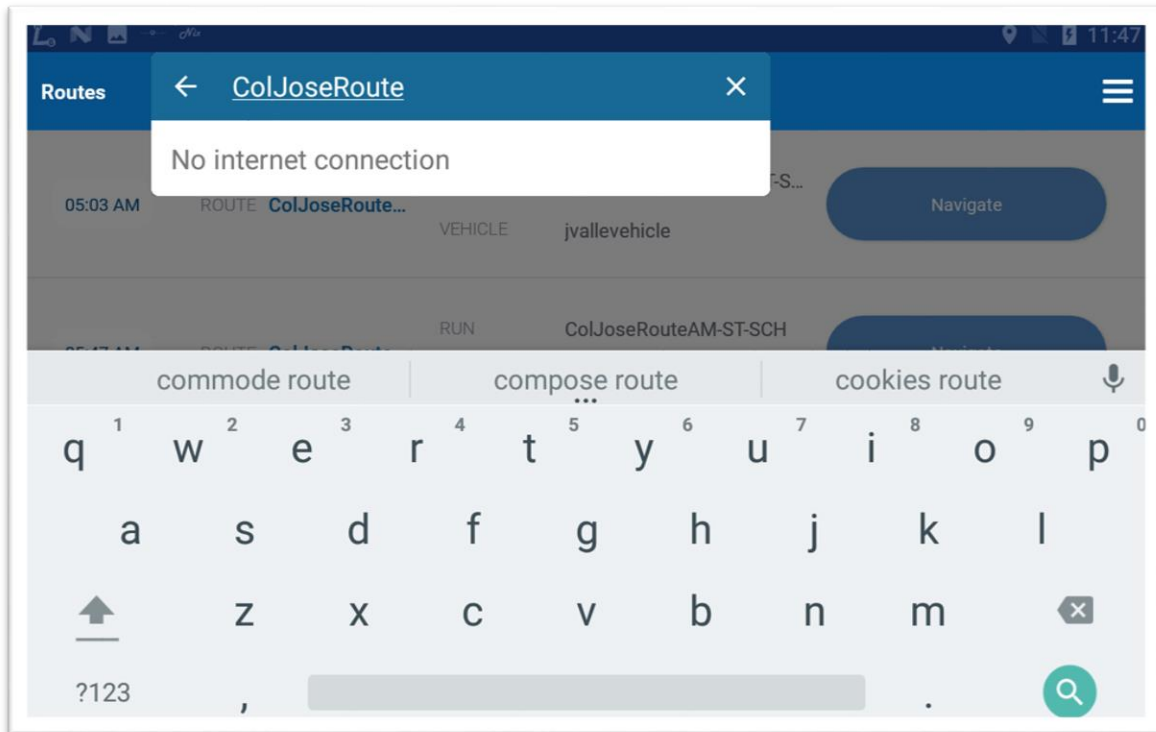


Figura 58. Prueba 2 de RF-03. Fuente: aplicaciones Navigator y WasteApp

3. Ver un diálogo indicando que no se pudo obtener la ruta al seleccionar navegar una ruta de la lista de rutas:

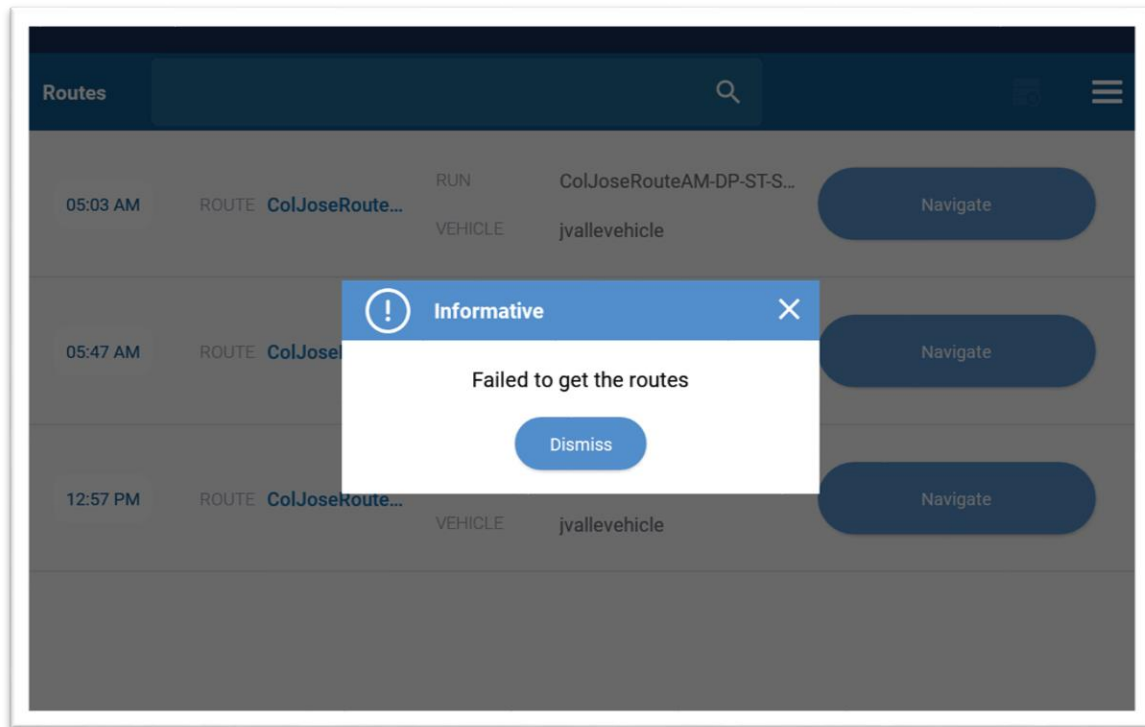


Figura 59. Prueba 3 de RF-03. Fuente: aplicaciones Navigator y WasteApp

La aplicación Navigator (buses escolares) fue capaz de:

4. Guardar los diferentes eventos de navegación como pendientes en la base de datos en lugar de enviarlos al servidor procesador de datos.

id	id	deviceId	latitude	longitude	spe...	time	direct...	origin	eventType	e...	sourceTy...	custom
52	52	30283252	6.336666...	-75.53145...	0	1616345507095	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
53	53	30283252	6.336666...	-75.53145...	0	1616345547120	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
54	54	30283252	6.336666...	-75.53145...	0	1616345567111	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
55	55	30283252	6.336666...	-75.53145...	0	1616345587045	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
56	56	30283252	6.336666...	-75.53145...	0	1616345627095	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
57	57	30283252	6.336666...	-75.53145...	0	1616345647089	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
58	58	30283252	6.336666...	-75.53145...	0	1616345667096	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...
59	59	30283252	6.336666...	-75.53145...	0	1616345707103	N	NAVIGATOR	GPS		SEON	[{"converterKey":"CustomGc...

Figura 60. Prueba 4 de RF-03: base de datos de la aplicación. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

RF-04: Observar el estado de conexión al servidor

En ambas aplicaciones WasteApp y Navigator cuando el servidor de autenticación y/o cuando el servidor de procesamiento de datos se encontró caído o no hubo conexión a Internet la aplicación fue capaz de conocer el estado del servidor y realizar una acción para notificar al conductor lo cual se evidencia en el siguiente requerimiento.

RF-05: Notificar el estado de conexión al servidor

En ambas aplicaciones WasteApp y Navigator, cuando el servidor de autenticación y/o cuando el servidor de procesamiento de datos se encontró caído o no hubo conexión a Internet el conductor fue capaz de:

1. Ver un icono parpadeante en la parte superior derecha en la pantalla de inicio de sesión de WasteApp:

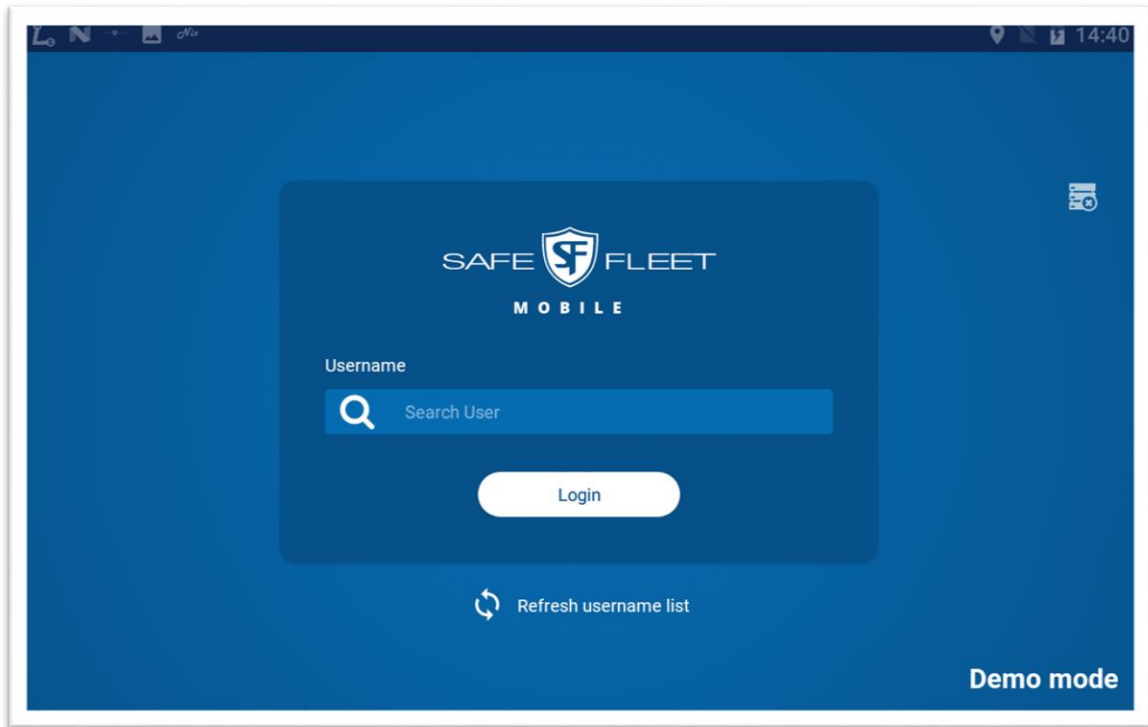


Figura 61. Prueba 1 de RF-05. Fuente: aplicación WasteApp

2. Ver un icono parpadeante en la parte superior derecha en la pantalla de la lista de rutas:

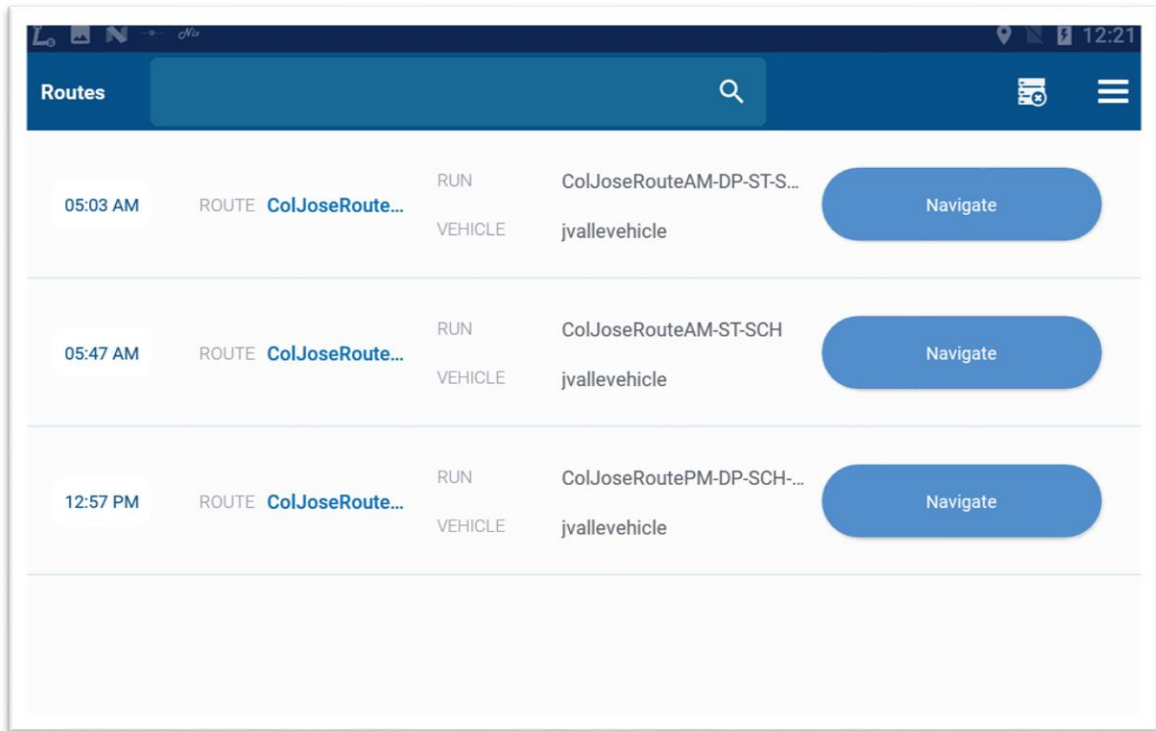


Figura 62. Prueba 2 de RF-05. Fuente: aplicaciones Navigator y WasteApp

3. Ver un icono parpadeante en la parte superior derecha en la pantalla de navegación:

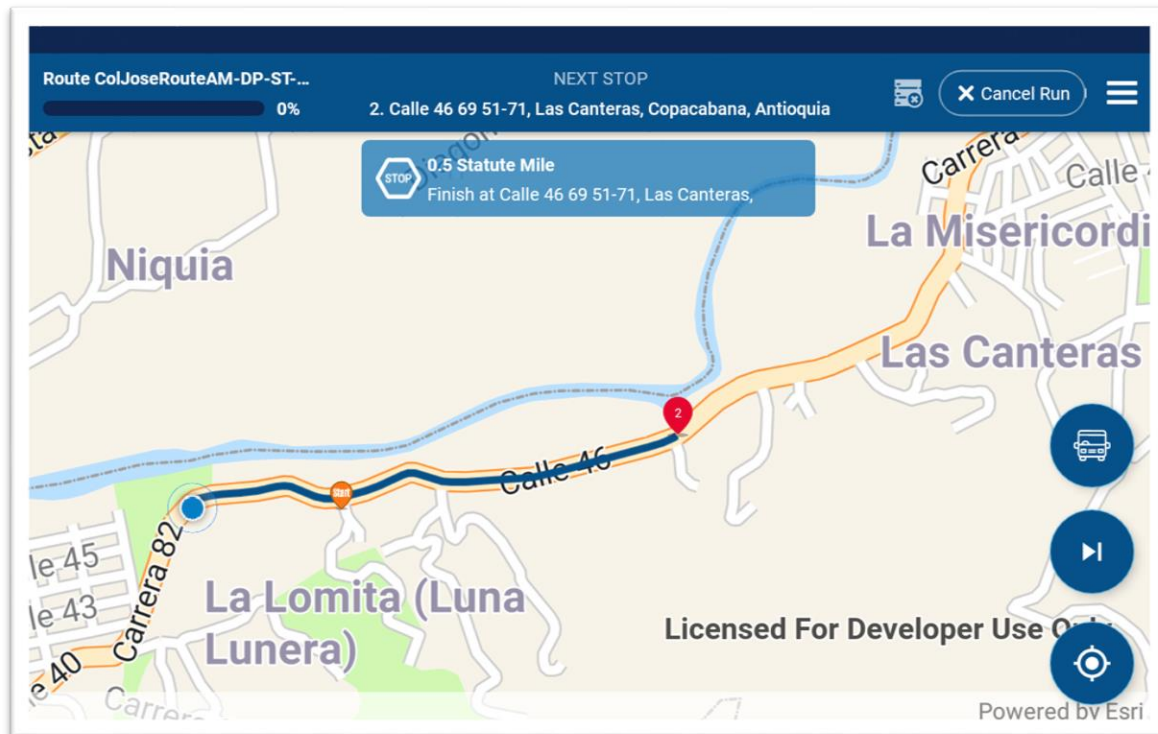


Figura 63. Prueba 3 de RF-05. Fuente: aplicaciones Navigator y WasteApp

RF-06: Observar el estado del brazo mecánico de parada del bus escolar

En la aplicación Navigator (buses escolares) la aplicación fue capaz de conocer:

1. Cuando se desplegó el brazo mecánico de parada para crear el evento que fue manejado por el servicio en segundo plano.
2. Cuando se replegó el brazo mecánico de parada después de estar desplegado para crear el evento que fue manejado por el servicio en segundo plano.

Lo cual se puede evidenciar en el siguiente requerimiento:

RF-07: Servicio de manejo y envío de eventos

En la aplicación Navigator (buses escolares) la aplicación fue capaz de:

1. Enviar eventos de brazo mecánico al servidor procesador de datos cada vez que se desplegó brazo mecánico de parada:

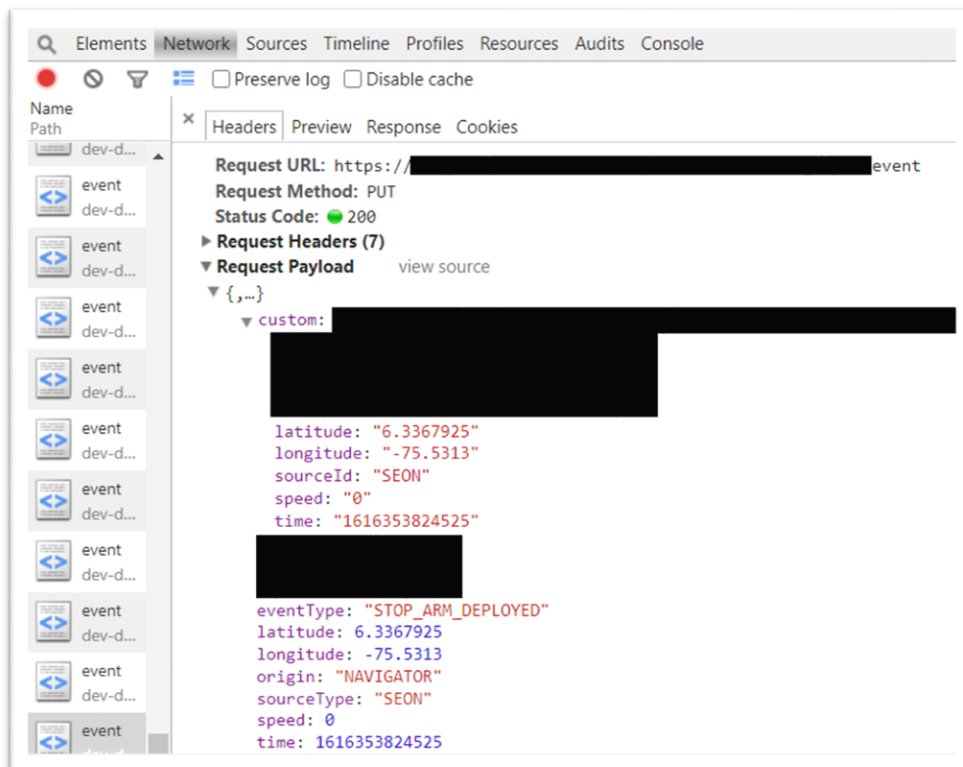


Figura 64. Prueba 1 de RF-07: respuesta exitosa al envío del evento. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

2. Enviar eventos de brazo mecánico al servidor procesador de datos cada vez que se replegó el brazo mecánico de parada:

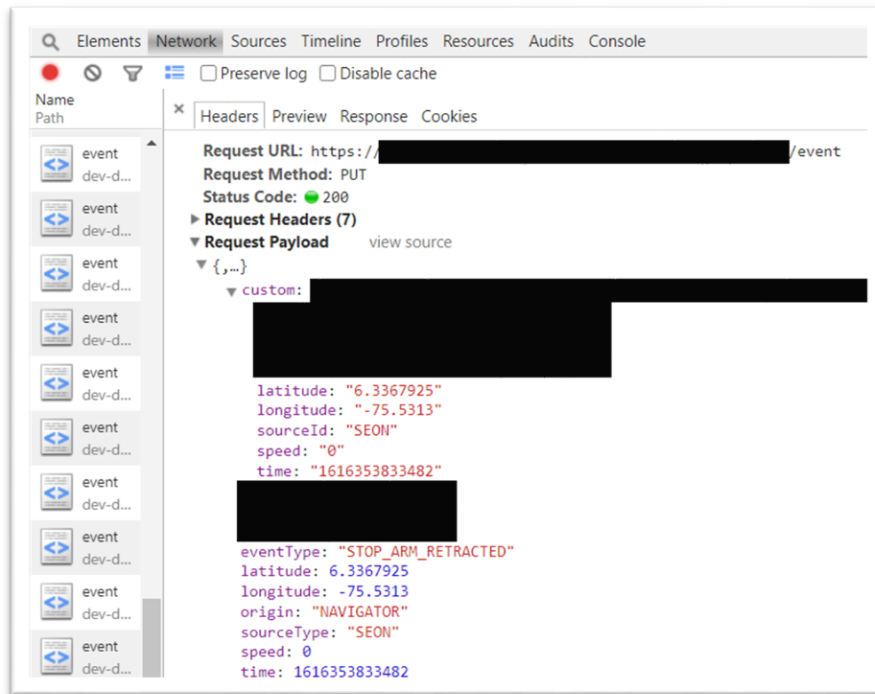


Figura 65. Prueba 2 de RF-07: respuesta exitosa al envío del evento. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

3. Enviar eventos de GPS al servidor procesador de datos a cada momento dentro de un intervalo de tiempo específico:

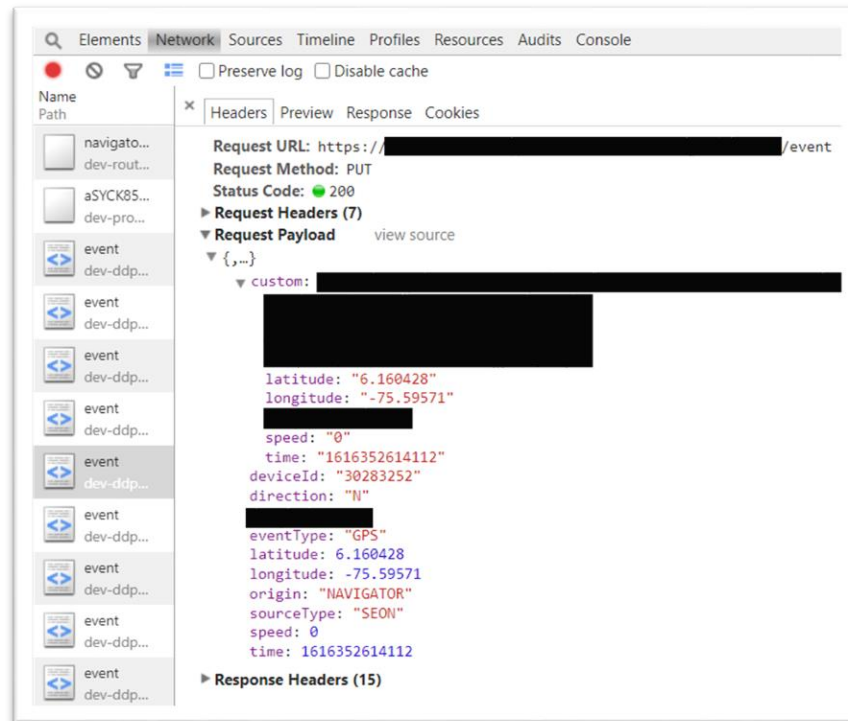


Figura 66. Prueba 3 de RF-07: respuesta exitosa al envío del evento. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

4. Enviar eventos de RFID cuando un estudiante ingresó al vehículo en una parada específica y pasó su tarjeta por el lector RFID:

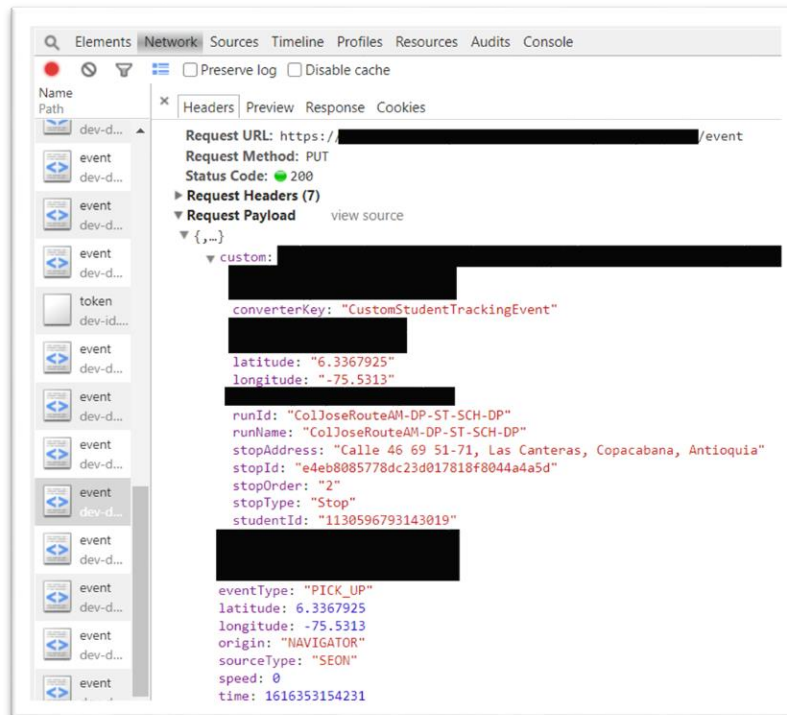


Figura 67. Prueba 4 de RF-07: respuesta exitosa al envío del evento. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

- Enviar eventos de RFID cuando un estudiante se bajó del vehículo en una parada específica y pasó su tarjeta por el lector RFID:

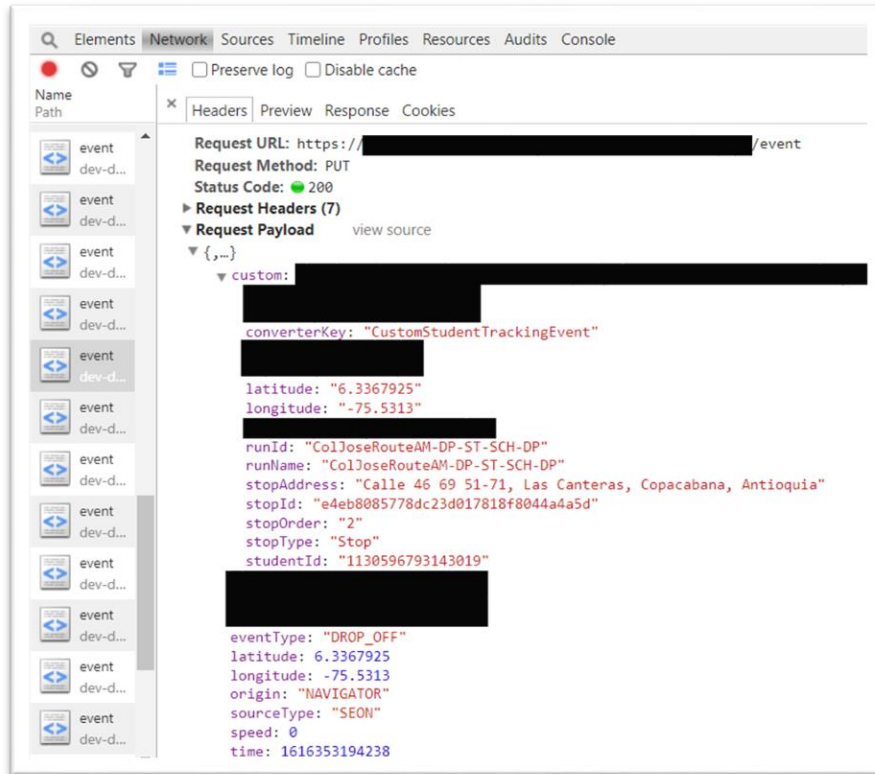


Figura 68. Prueba 5 de RF-07: respuesta exitosa al envío del evento. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

- Guardar los diferentes eventos de navegación en la base de datos cuando no hay conexión a Internet:

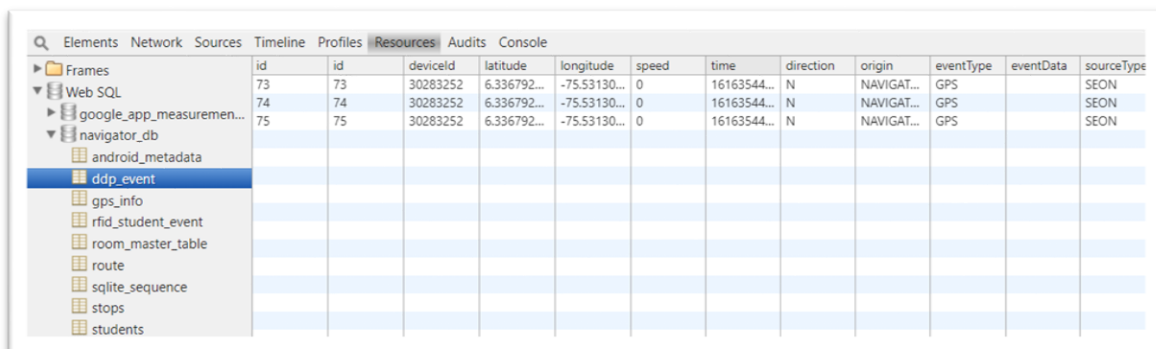


Figura 69. Prueba 6 de RF-07: base de datos de la aplicación. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

7. Enviar los eventos pendientes cuando hay conexión a Internet y una vez recibidos borrarlos de la base de datos:

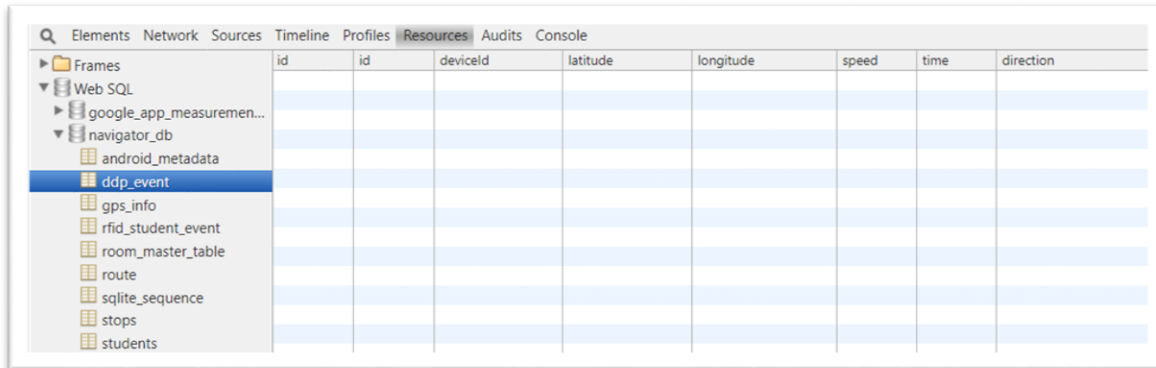


Figura 70. Prueba 7 de RF-07: base de datos de la aplicación. Fuente: herramienta Inspect de Google Chrome con el plugin Stetho para aplicaciones Android

RF-08: Pantalla de navegación de cada negocio

En la pantalla de navegación en Navigator (buses escolares) después de haber seleccionado una ruta, el conductor fue capaz de:

1. Ver la ruta trazada por segmentos en el mapa con sus respectivas paradas:

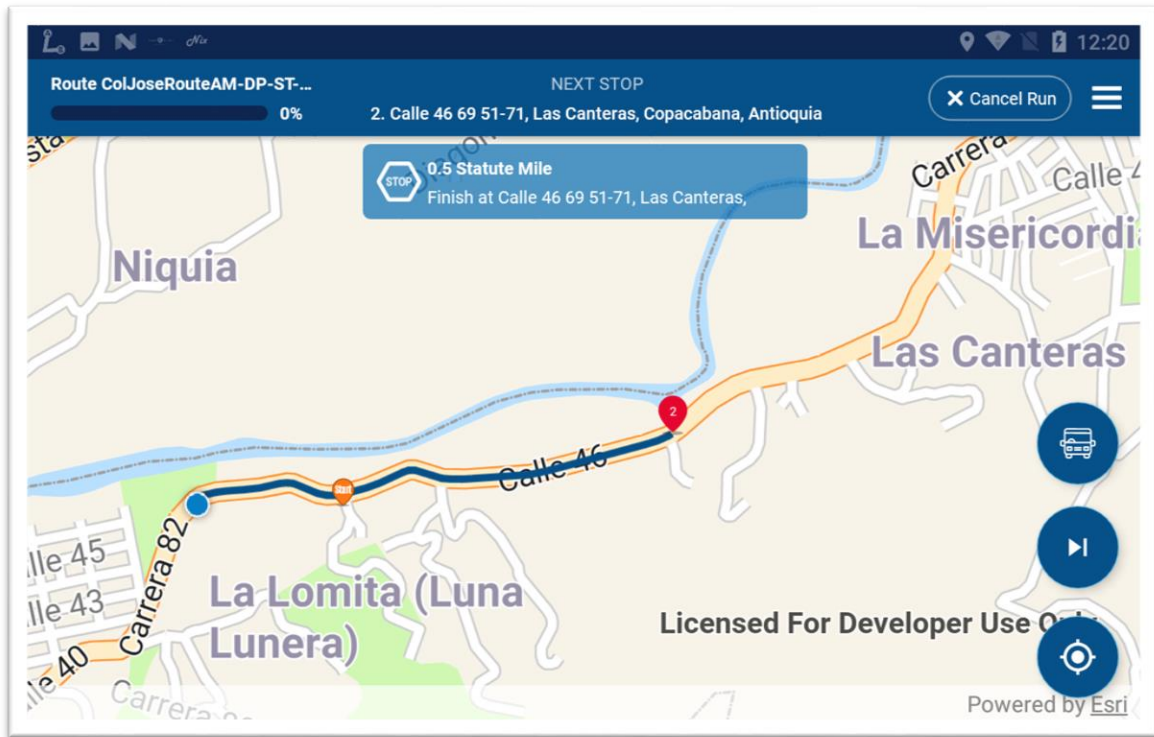


Figura 71. Prueba 1 de RF-08. Fuente: aplicación Navigator

2. Seleccionar cancelar ruta en la barra superior y ver un diálogo de confirmación para cancelar la ruta:

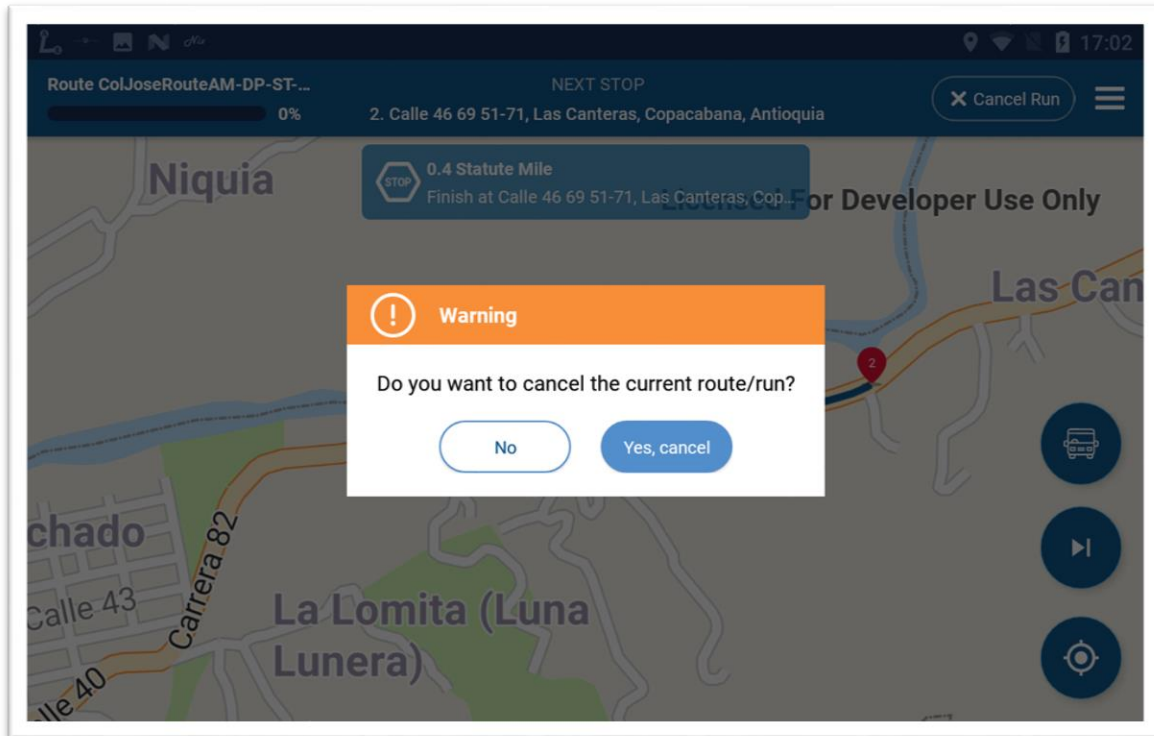


Figura 72. Prueba 2 de RF-08. Fuente: aplicación Navigator

3. Seleccionar el primer botón en la parte inferior derecha y ver la lista de estudiantes:

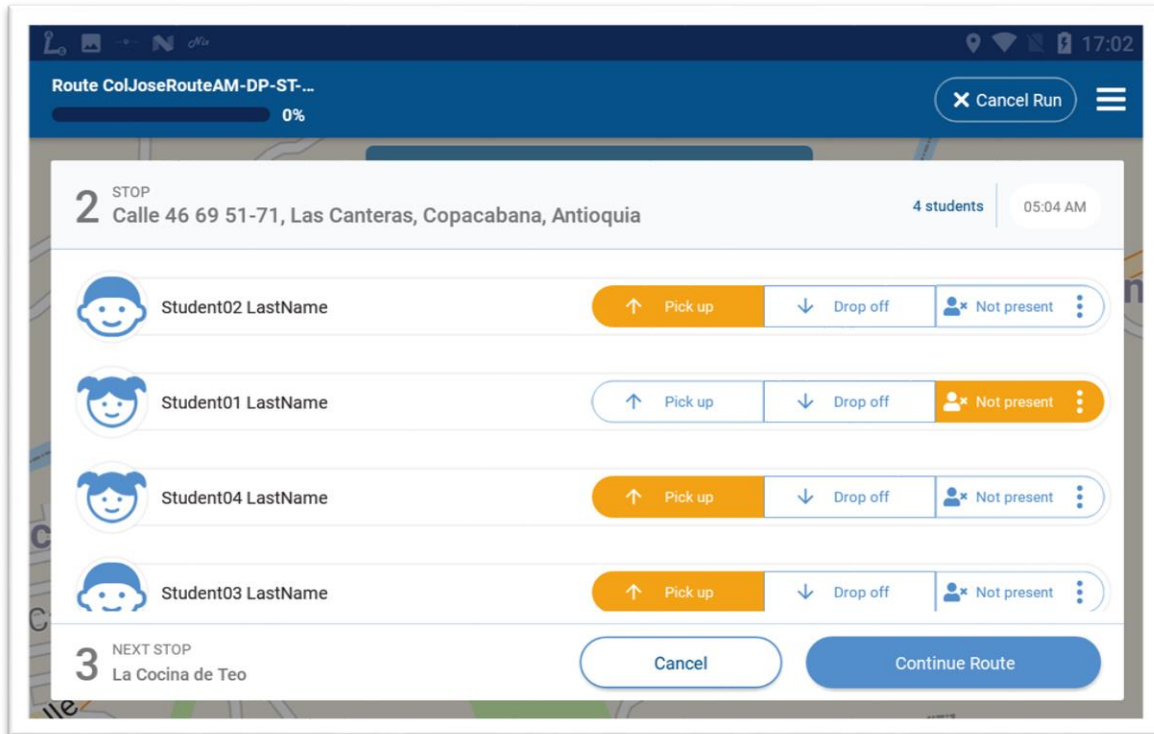


Figura 73. Prueba 3 de RF-08. Fuente: aplicación Navigator

4. Seleccionar el segundo botón en la parte inferior derecha y ver un diálogo de confirmación de saltar parada:

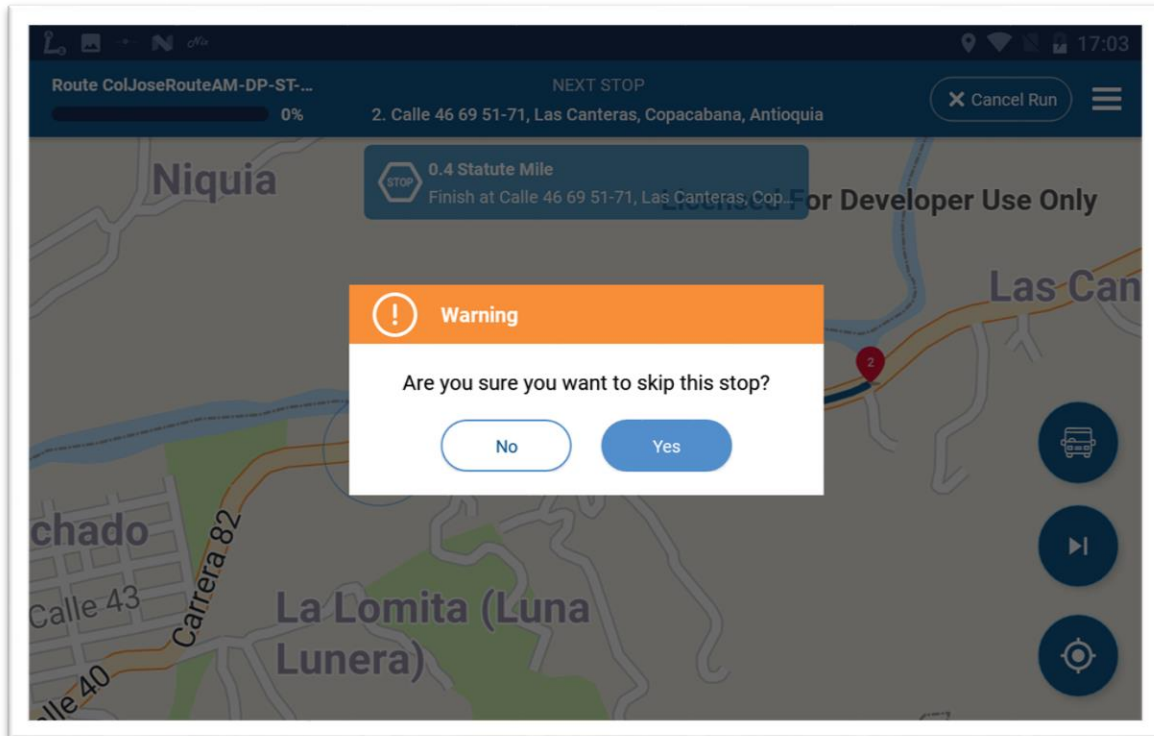


Figura 74. Prueba 4 de RF-08. Fuente: aplicación Navigator

En la pantalla de navegación en WasteApp (vehículos de desechos) después de haber seleccionado una ruta, el conductor fue capaz de:

5. Ver la primera asignación en el mapa:

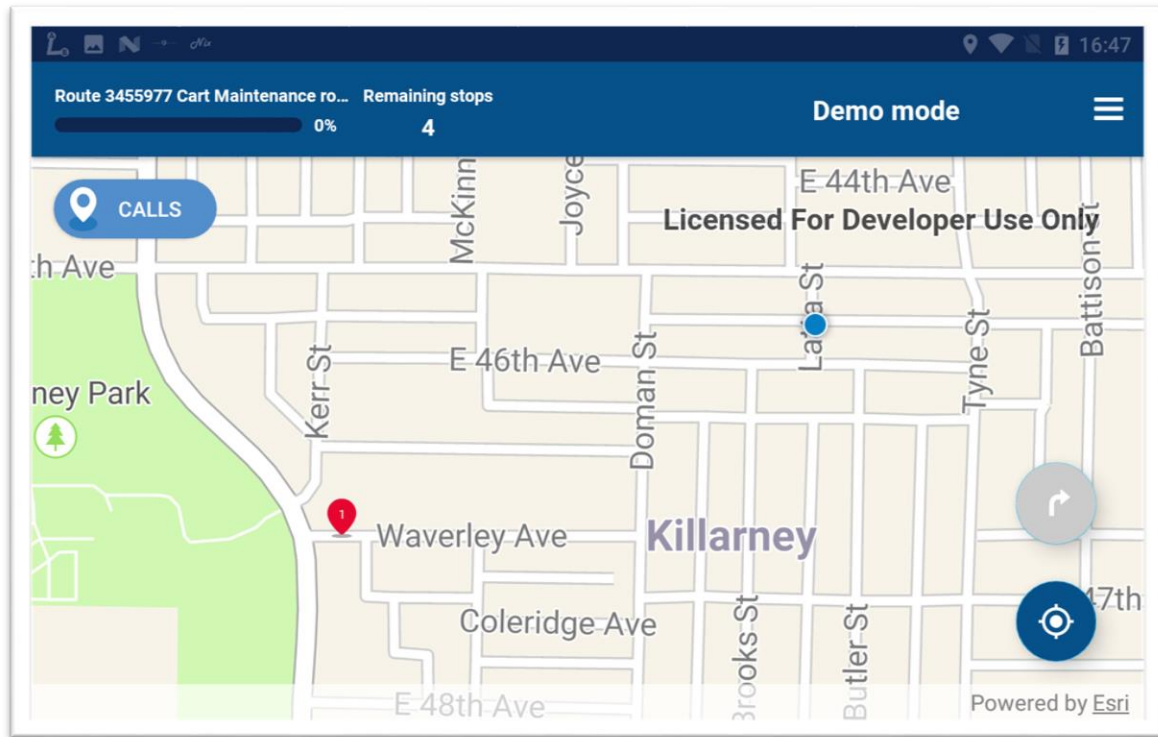


Figura 75. Prueba 5 de RF-08. Fuente: aplicación WasteApp

6. Seleccionar el primer botón en la parte inferior derecha para activar el trazado de ruta y sus respectivas indicaciones:

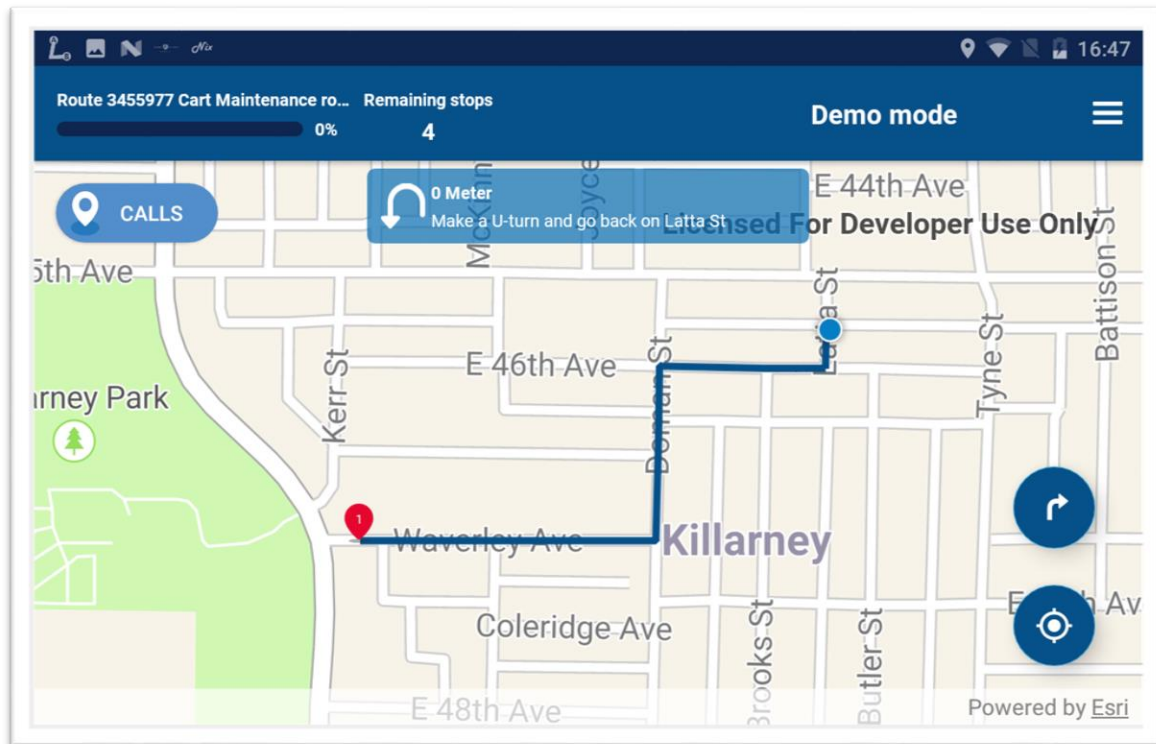


Figura 76. Prueba 6 de RF-08. Fuente: aplicación WasteApp

7. Seleccionar el botón en la parte superior izquierda “CALLS” y ver la lista de asignaciones de la ruta:

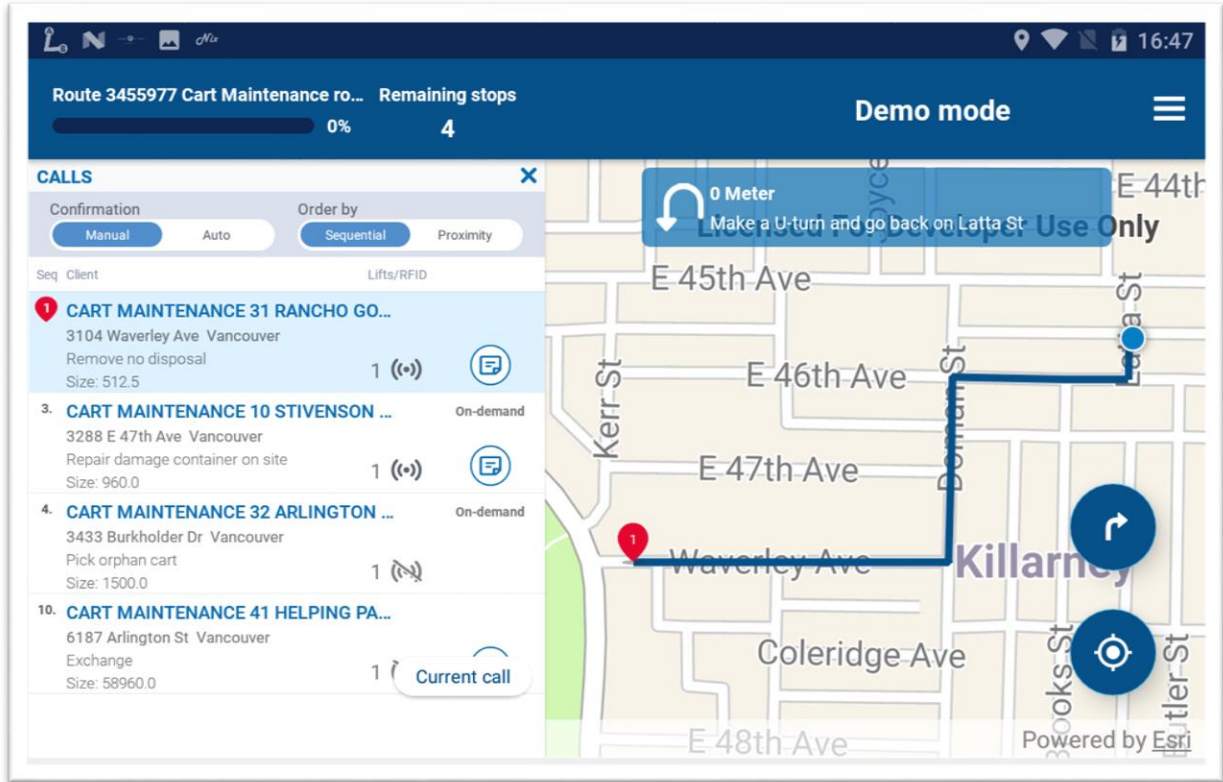


Figura 77. Prueba 7 de RF-08. Fuente: aplicación WasteApp

CAPÍTULO VII: CONCLUSIONES Y RECOMENDACIONES

En el desarrollo de este proyecto se aplicaron múltiples conocimientos que hacen parte de la profesión de Ingeniería de Sistemas Informáticos enseñados durante el proceso de formación de la carrera, como son: programación, manejo de bases de datos, calidad de software, procesos y herramientas de desarrollo de software, entre otros. Los cuales se pusieron en práctica en un proyecto real bajo un contexto empresarial ubicado en Canadá y Estados Unidos. Entre ellos se destacan: buenas prácticas, principios SOLID, pruebas unitarias, documentación y código limpio. Prácticas de desarrollo ágil y trabajo en equipo bajo el marco Scrum: programación en parejas, reuniones diarias, retrospectivas, refinamientos, limpiezas y demostraciones.

Durante cada sprint se asignaron las tareas planteadas en la definición de los requerimientos como historias de usuarios con descripciones sobre los detalles necesarios para su desarrollo. Cada una de ellas está directamente relacionada con el funcionamiento de navegación y la integración del producto con el sistema de información (servidor) que se encarga de procesar los diferentes tipos de datos provenientes de los dispositivos VT7 ubicados en los vehículos, permitiendo su seguimiento y gestión para la asignación de rutas, estudiantes y encargos.

Los diferentes componentes de la aplicación, incluyendo librerías, patrones y arquitectura seleccionados, demuestran su pertinencia al garantizar los principios SOLID de programación orientada a objetos, desacoplando la lógica entre clases y módulos, facilitando la creación de pruebas, ayudando a garantizar estándares y características de calidad como la mantenibilidad, la eficiencia y la seguridad, y haciendo de la aplicación un producto de alto valor para el cliente.

Las funcionalidades implementadas en el desarrollo de este proyecto de pasantía fueron probadas basadas en el detalle de las historias de usuario antes de ser entregadas junto con sus respectivas pruebas unitarias, revisadas por el equipo, aprobadas e integradas en la aplicación con el código de los diferentes repositorios

manejados. También fueron probadas, después de entregadas, por el equipo de aseguramiento de calidad para garantizar su correcto funcionamiento basados en casos de prueba diseñados por ellos mismos y por el cliente.

La información obtenida con el caso de estudio, permite evidenciar el correcto funcionamiento de la aplicación en un contexto de la vida real, con una ruta realizada por un conductor a las afueras de la ciudad de Medellín, la cual contenía los elementos necesarios probar los diferentes requerimientos funcionales planteados para el desarrollo del proyecto de pasantía: desde la interacción del conductor con la aplicación como la comunicación de la aplicación con el servidor procesador de eventos de SafeFleet y sus diferentes escenarios.

Cabe resaltar que, con la experiencia que se obtuvo en este proyecto, se logró visibilizar problemáticas en la logística de desarrollo como situaciones en las que tareas relacionadas, trabajadas por diferentes desarrolladores en un mismo momento y que involucran la modificación de una misma parte del código, traían consigo conflictos, consumiendo tiempo y esfuerzo en su resolución. Estas problemáticas pueden ser evitadas con una planeación previa de los desarrolladores, con diagramas generales de la aplicación y de sus componentes. Por lo que se debe dar relevancia a este tipo de tareas de desarrollo y considerar delegarlas a un rol del equipo de trabajo.

Asimismo, este proyecto permite considerar realizar pasantías como una opción relevante de trabajo de grado, ya que permite obtener experiencia en el ámbito laboral y empresarial donde, contrario a lo que se puede pensar, se lleva a cabo trabajo de investigación continuo, de igual o en mayor medida que en el ámbito académico, al acceder a problemáticas y necesidades de clientes que requieren un arduo estudio para la creación e implementación de nuevos conocimientos, con el objetivo de proponer y desarrollar productos que solucionen esas problemáticas y que suplan esas necesidades.

REFERENCIAS

- [1] PSL. (n.d.). ¿Quiénes Somos? - Nearshore & Offshore Software Development Outsourcing from Colombia! PSL is now part of Perficient - Highest Quality Agile Development. Global Delivery. ISO27001 Compliant. Let's Talk! Retrieved February 9, 2021, from <https://www.psl.com.co/psl/sobre-nuestra-compania.html>
- [2] Schlosser, N. (2020). Bus-to- Technician Ratio on the Rise. Maintenance Survey, March, 14–18. www.schoolbusfleet.com
- [3] School Bus Fleet. (2020). School Transportation: 2017-18 School Year. In Fact Book 2020 (Issue Pupil Transportation Statistics, pp. 42–43). www.schoolbusfleet.com
- [4] BBC News. (2018, October 25). Devon schoolboy froze to death after missing bus. BBC News. <https://www.bbc.com/news/uk-england-devon-45981980>
- [5] Bies, J. (2019, January 23). 5-year-old Delaware boy left on freezing school bus for several hours. Delaware Online. <https://www.delawareonline.com/story/news/2019/01/23/5-year-old-delaware-boy-left-freezing-school-bus-several-hours/2655843002/>
- [6] Transport Canada. (2019). Transport Canada Departmental Plan 2019-2020. <https://www.tc.gc.ca/eng/corporate-services/transport-canada-2019-2020-departmental-plan.html#toc3-1>
- [7] The World Bank. (2016). Trends in Solid Waste Management. Datatopics.Worldbank.Org. https://datatopics.worldbank.org/what-a-waste/trends_in_solid_waste_management.html
- [8] Hettiaratchi, J. P. A. (2007). New trends in waste management: North American perspective. International Conference on Sustainable Solid Waste Management, January 2007, 9–14.

- [9] Science, E., Barbara, S., & Williamson, O. E. (1999). Exposing Strategic Assets to Create New Acquisition in the Waste Management. *Industrial and Corporate Change*, 8(4), 635–671.
- [10] Fong, S. L., Bin Abu Bakar, A. A. A., Ahmed, F. Y. H., & Jamal, A. (2019). Smart transportation system using RFID. *ACM International Conference Proceeding Series, Part F147956*, 579–584. <https://doi.org/10.1145/3316615.3316719>
- [11] Alonso, A. B., Artime, I. F., Rodríguez, M. Á., Baniello, R. G., & Telecomunicación, E. P. S. I. G. I. De. (2010). Dispositivos móviles. Universidad de Oviedo.
- [12] Aguado, J.-M., Martínez, I. J., & Cañete-Sanz, L. (2015). Tendencias evolutivas del contenido digital en aplicaciones móviles. *El Profesional de La Información*, 24(6), 787. <https://doi.org/10.3145/epi.2015.nov>.
- [13] 10Lucas, G. (2015). Evolucion De Las Aplicaciones Para Moviles. 21. <https://empresarias.camara.es/estaticos/upload/0/007/7438.pdf>
- [14] Morales Morante, L. F., Mas Manchón, L., & Tous, R. (2016). Modelo en red de los contenidos mediáticos en la era de los dispositivos inteligentes. *Ibersid*, 10(2), 69–78.
- [15] Michael E., P., & James E., H. (2015). How smart, connected products are transforming companies. *Harvard Business Review*, 93(10), 96–114.
- [16] Google. (n.d.). Mapas personalizados | Google Maps Platform | Google Cloud. Retrieved April 7, 2020, from <https://cloud.google.com/maps-platform/maps?hl=es-419>
- [17] Google. (n.d.). Waze| Google Developers. Google. Retrieved April 7, 2020, from <https://developers.google.com/waze>
- [18] SKOLE. (n.d.). Retrieved April 5, 2020, from <http://www.skole.cl/>

- [19] Rube. (n.d.). Retrieved April 5, 2020, from <https://rubeapp.com/>
- [20] Yo-Waste App | Location based on demand garbage collection service app. (n.d.). Retrieved April 5, 2020, from <https://yowasteapp.com/>
- [21] Google. (2018). Flutter - Beautiful native apps in record time. Google. <https://flutter.dev/>
- [22] Google. (2019). GitHub - flutter/flutter: Flutter makes it easy and fast to build beautiful mobile apps. <https://github.com/flutter/flutter>
- [23] Facebook Open Source. (2018). React Native - A framework for building native apps using React. Facebook Inc. <https://reactnative.dev/>
- [24] Facebook. (2019). GitHub - Facebook/react-native: A framework for building native apps with React. <https://github.com/facebook/react-native>
- [25] Axelsson, O., & Carlström, F. (2016). Evaluation Targeting React Native in Comparison to Native Mobile Development. 62. <http://lup.lub.lu.se/luur/download?func=downloadFile&recordId=8886469&fileId=8886473>
- [26] Bibik, I. (2018). How to kill the scrum monster quick start to agile scrum methodology and the scrum master role. Palgrave Macmillan.
- [27] Hayat, F., Rehman, A. U., Arif, K. S., Wahab, K., & Abbas, M. (2019, July). The influence of agile methodology (Scrum) on software project management. In 2019 20th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD) (pp. 145-149). IEEE.
- [28] Pittet, S. (2018). Continuous Integration Vs Continuous Delivery Vs Continuous Deployment. 1–2. <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

- [29] Jenkins Developers. (2018). Jenkins User Documentation. 2018. <https://www.jenkins.io/doc/>
- [30] Git. (2020). Git. <https://git-scm.com/>
- [31] Atlassian. (2017). Gitflow Workflow | Atlassian Git Tutorial. Atlassian.Com. <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>
- [32] Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston, MA: Prentice Hall. ISBN: 978-0-13-449416-6
- [33] Raul Calderón, Xavier Martínez, & Escofet Carne. (2016). El Estándar ISO y su Aportación al Proceso de Calidad del Desarrollo de Software. 1–60. <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/53422/8/fcalderonmTFC0616memoria.pdf>
- [34] Android Developers. (2020). Android Kotlin Fundamentals: Repository. <https://developer.android.com/codelabs/kotlin-android-training-repository#5>
- [35] Android Developers. (2020). Componentes de la de arquitectura Android | Desarrolladores de Android. <https://developer.android.com/topic/libraries/architecture?hl=es>
- [36] Android Developers. (2020). Descripción general de ViewModel | Desarrolladores de Android. <https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es>
- [37] Android Developers. (2020). Descripción general de LiveData | Desarrolladores de Android. <https://developer.android.com/topic/libraries/architecture/livedata?hl=es>
- [38] Android Developers. (2020). Biblioteca de persistencias Room | Desarrolladores de Android. <https://developer.android.com/topic/libraries/architecture/room?hl=es>

- [39] Android Developers. (2020). Cómo usar las corrutinas de Kotlin con componentes de la arquitectura. <https://developer.android.com/topic/libraries/architecture/coroutines?hl=es>
- [40] Android Developers. (2020). Inyección de dependencias en Android | Desarrolladores de Android. <https://developer.android.com/training/dependency-injection?hl=es>
- [41] Android Developers. (2020). Conceptos básicos de Dagger | Desarrolladores de Android. <https://developer.android.com/training/dependency-injection/dagger-basics?hl=es>
- [42] Android Developers. (2020). Aspectos básicos de las pruebas | Desarrolladores de Android. <https://developer.android.com/training/testing/fundamentals?hl=es>
- [43] Android Developers. (2020). Cómo compilar pruebas de unidades locales | Desarrolladores de Android. <https://developer.android.com/training/testing/unit-testing/local-unit-tests?hl=es>
- [44] MockK. (2020). MockK | mocking library for Kotlin. <https://mockk.io/>
- [45] Android Developers. (2020). Cómo automatizar las pruebas de la interfaz de usuario. <https://developer.android.com/training/testing/ui-testing?hl=es>
- [46] Android Developers. (2020). Prueba la IU para una sola app | Desarrolladores de Android. <https://developer.android.com/training/testing/ui-testing/espresso-testing?hl=es>
- [47] Smith, J. (n.d.). Android Asynchronous Http Client. Retrieved November 23, 2020, from <https://loopj.com/android-async-http/>
- [48] Koush. (n.d.). ion: Android Asynchronous Networking and Image Loading. Retrieved November 23, 2020, from <https://github.com/koush/ion>

- [49] Square. (n.d.). Retrofit. Retrieved November 24, 2020, from <https://square.github.io/retrofit/>
- [50] Android Developers. (2020). Descripción general de Volley | Desarrolladores de Android. <https://developer.android.com/training/volley>
- [51] Realm. (n.d.). Realm for Android. Realm.io. Retrieved November 24, 2020, from <https://realm.io/blog/realm-for-android/>
- [52] Android Developers. (2020). Biblioteca de persistencias Room | Desarrolladores de Android. <https://developer.android.com/topic/libraries/architecture/room>
- [53] JetBrains. (n.d.). Coroutines - Kotlin Programming Language. Retrieved November 24, 2020, from <https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html>
- [54] Android Developers. (2020). Corrutinas de Kotlin en Android | Desarrolladores de Android. <https://developer.android.com/kotlin/coroutines>
- [55] Greenrobot. (2016). EventBus: Events for Android - Open Source by greenrobot. <https://greenrobot.org/eventbus/>
- [56] ReactiveX. (n.d.). ReactiveX. Retrieved November 24, 2020, from <http://reactivex.io/>
- [57] AndroidAnnotations. (n.d.). AndroidAnnotations. [Androidannotations.org](http://androidannotations.org). Retrieved November 24, 2020, from <http://androidannotations.org/>
- [58] Square and Google. (n.d.). Dagger & Android. Dagger.Dev. Retrieved November 24, 2020, from <https://dagger.dev/dev-guide/android.html>
- [59] ByteWelder. (n.d.). Spork. Retrieved November 24, 2020, from <https://spork.bythewelder.com/>
- [60] Insert Koin. (n.d.). KOIN: a smart Kotlin dependency injection framework. [Insert-Koin.io](https://insert-koin.io). Retrieved November 24, 2020, from <https://insert-koin.io/>

- [61] Prada, J. (2019). Laboratorio móvil de Belatrix: Dagger vs Koin. <https://www.belatrixsf.com/whitepapers/dagger-vs-koin-sp/>
- [62] Bump Technologies. (2020). Glide v4 : Fast and efficient image loading for Android. <https://bumptech.github.io/glide/>
- [63] Square. (n.d.). Picasso. Retrieved November 24, 2020, from <https://square.github.io/picasso/>
- [64] Google. (2015). GSON: A Java serialization/deserialization library to convert Java Objects into JSON and back. <https://github.com/google/gson>
- [65] Square. (2020). Moshi: A modern JSON library for Kotlin and Java. <https://github.com/square/moshi>
- [66] Hart, E. (2016). Epoxy: Airbnb's View Architecture on Android | by AirbnbEng | Airbnb Engineering & Data Science | Medium. Medium.com. <https://medium.com/airbnb-engineering/epoxy-airbnbs-view-architecture-on-android-c3e1af150394#.9hno4vdc0>
- [67] Penz, M. (n.d.). FastAdapter: The bullet proof, fast and easy to use adapter library, which minimizes developing time to a fraction... Retrieved November 24, 2020, from <https://github.com/mikepenz/FastAdapter>
- [68] Chen, M. (n.d.). UltimateRecyclerView. Retrieved November 24, 2020, from <http://blog.marshalchen.com/UltimateRecyclerView/>
- [69] AdevintaSpain. (n.d.). Barista: The one who serves a great Espresso. Retrieved November 24, 2020, from <https://github.com/AdevintaSpain/Barista>
- [70] Android Developers. (2020). Espresso | Desarrolladores de Android. <https://developer.android.com/training/testing/espresso>
- [71] The JUnit Team. (2019). JUnit 5. <https://junit.org/junit5/>
- [72] MockK. (2020). MockK | mocking library for Kotlin. <https://mockk.io/>

- [73] Robolectric. (n.d.). Robolectric. Robolectric.Org. Retrieved November 25, 2020, from <http://robolectric.org/>
- [74] Android Developers. (2020). Android KTX | Desarrolladores de Android. <https://developer.android.com/kotlin/ktx>
- [75] Android Developers. (2020). Jetpack Compose | Desarrolladores de Android. <https://developer.android.com/jetpack/compose>
- [76] Corti, N. (2019). Introducing Chucker. Chucker, a fork of Chuck, is a simple... | ProAndroidDev. Proandroiddev.Com. <https://proandroiddev.com/introducing-chucker-18f13a51b35d>
- [77] Square. (n.d.). LeakCanary. Retrieved November 25, 2020, from <https://square.github.io/leakcanary/>
- [78] Android Developers. (2020). Navigation | Desarrolladores de Android. <https://developer.android.com/guide/navigation>
- [79] EclEmma. (2020). Biblioteca de cobertura de código Java de JaCoCo. EclEmma.Org. <https://www.eclEmma.org/jacoco/>
- [80] SonarSource. (2020). Documentation | SonarQube Docs. <https://docs.sonarqube.org/latest/>