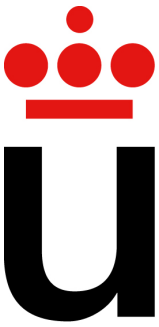

Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

GRADOS EN INGENIERÍA DE COMPUTADORES

Aplicación de Gestión de Recogida y Entrega de Paquetes: Diseño, Usabilidad y Evaluación

CURSO 2023-2024

Autor/a: **Arrab Nehme Roumani**

Tutelado por: Joaquín Arias Herrero

Agradecimientos

Ante todo, quiero agradecer especialmente a Hakima y Rami, quienes han sido un pilar fundamental y han realizado un esfuerzo significativo para que pudiera culminar exitosamente mi trabajo.

Asimismo, me gustaría expresar mi profundo agradecimiento a mi tutor, Joaquín, por su constante apoyo y orientación a lo largo del desarrollo de este trabajo. Sin su apoyo y conocimientos este trabajo no hubiese salido adelante.

Finalmente, pero no menos importante, quiero expresar mi gratitud a mi familia y amigos, quienes me han brindado la fuerza y la energía necesaria, a lo largo de este trayecto.

Resumen

Una empresa de logística VERTEX enfrenta desafíos en la gestión manual de paquetes y empleados debido a su crecimiento. Buscan una solución asequible y eficiente para mejorar la asignación de tareas y mantener actualizado el estado de los paquetes.

Se desarrolla una aplicación para optimizar la gestión de recogida y entrega de paquetes, así como para coordinar eficientemente a los conductores en el reparto de tareas, facilitando la coordinación en la asignación de paquetes tanto para la recogida como para la entrega.

La solución consiste en:

- El desarrollo de una aplicación móvil que permita gestionar estas operaciones de manera ágil y eficaz.
- La aplicación proporciona datos actualizados en tiempo real.
- Para garantizar la eficacia y eficiencia de la aplicación, se llevan a cabo pruebas de usabilidad que evalúan su facilidad de uso en situaciones reales.

Además, se contempla la posibilidad de realizar mejoras futuras en la aplicación, tanto para optimizar las funcionalidades existentes como para añadir nuevas características que puedan surgir en el futuro.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Una empresa de logística, inicialmente conformada por dos socios y situada en Beirut, conocida como VERTEX, se dedica a la recogida y entrega de paquetes. Con el tiempo, su expansión y el aumento en la demanda de rutas han llevado a la necesidad de ampliar su plantilla de empleados, que originalmente constaba de cuatro personas: dos oficinistas y dos conductores. Sin embargo, la creciente demanda de nuevas rutas ha llevado a los dos conductores a su capacidad máxima de trabajo.

VERTEX presta servicios a empresas que necesitan enviar paquetes a otras compañías, operando bajo el lema (una entrega a tiempo, un cliente satisfecho). La empresa opera bajo un modelo de trabajo a demanda, donde los oficinistas reciben pedidos, asignan números de referencia, establecen rutas y los conductores recogen y entregan los paquetes según la ruta asignada. Una vez finalizada la ruta, los conductores informan a los oficinistas, quienes archivan la ruta y envían la factura correspondiente a la empresa cliente.

Para satisfacer la creciente demanda de sus servicios la empresa de paquetería VERTEX necesita adaptarse y crecer. Conscientes de los desafíos que enfrentan debido al aumento en la demanda de nuevas rutas, los socios buscan soluciones innovadoras para optimizar su operación y mantener la calidad del servicio al cliente. El enfoque en la implementación de una aplicación móvil para gestionar las rutas refleja su compromiso con la eficiencia y la mejora continua. En resumen, la motivación es la búsqueda de soluciones innovadoras y eficientes para abordar los desafíos operativos y de crecimiento de la empresa.

La utilización de aplicaciones móviles para resolver problemas de rutas es una tendencia creciente en la industria logística. Estas aplicaciones ofrecen una amplia gama de funcionalidades diseñadas para optimizar la gestión de flotas, mejorar la eficiencia operativa y proporcionar una mejor experiencia tanto para los conductores como para los clientes.

Algunas de las características comunes de estas aplicaciones incluyen:

- Seguimiento en tiempo real: Permite a los gestores de flotas monitorear la ubicación y el estado de los vehículos en tiempo real, lo que facilita la planificación de rutas y la

asignación de tareas.

- Planificación de rutas optimizada: Ronny (2016) propone la utilización de algoritmos avanzados, en aplicaciones para que puedan calcular las rutas más eficientes para minimizar los tiempos de viaje y reducir los costos operativos.
- Comunicación instantánea: Facilitan la comunicación bidireccional entre los conductores y el equipo de gestión, lo que permite una coordinación más efectiva y una respuesta rápida ante cualquier problema que surja durante el viaje.
- Gestión de entregas y recogidas: Permiten el seguimiento de los pedidos desde el momento de la recogida hasta la entrega final, lo que garantiza una mayor transparencia y satisfacción del cliente.
- Análisis de datos y reporting: Ofrecen herramientas analíticas que permiten a las empresas de transporte recopilar y analizar datos sobre el rendimiento de la flota, los tiempos de entrega, el consumo de combustible, entre otros, para tomar decisiones informadas y mejorar continuamente sus operaciones.

En resumen, las aplicaciones móviles están transformando la forma en que las empresas de transporte gestionan sus operaciones, permitiendo una mayor eficiencia, visibilidad y control sobre sus flotas, y mejorando la experiencia tanto para los conductores como para los clientes.

Una vez desarrollada (o seleccionada) una aplicación concreta es necesario evaluar su idoneidad para la empresa mediante pruebas de usabilidad y/o evaluación de la accesibilidad.

La evaluación de la usabilidad se ha convertido en una tarea fundamental en el ciclo de vida del software porque asegura que el producto sea eficiente, efectivo y que cumpla las expectativas de los usuarios. Identifica y soluciona problemas de usabilidad, mejora la experiencia del usuario, y contribuye al éxito comercial del software al aumentar la aceptación del mercado y reducir los costos de soporte.

En el contexto de las pruebas de usabilidad, diversos autores han sugerido enfoques para abordar este desafío en aplicaciones web y móviles, que se pueden clasificar en dos grandes grupos: (i) los que se enfocan en la evaluación una vez desarrollada la aplicación y (ii) los que integran la evaluación en el desarrollo de la misma:

(i) Propuestas que se enfocan en la evaluación:

- Metodologías y problemas en las pruebas de usabilidad de aplicaciones móviles: Zhang y Adipat (2005) proponen dos métodos específicos para evaluar la usabilidad de aplicaciones móviles:
  - Pruebas de laboratorio: Pruebas virtuales realizadas en máquinas de escritorio.
  - Pruebas de campo: Simulación de un entorno de producción real para mejorar la experiencia del usuario y minimizar errores provocados por emuladores o entornos de prueba.
- Pruebas en Entornos Controlados: Zhang y Adipat (2005) estas pruebas definen características poblacionales, especificaciones de dispositivos y tareas específicas para los usuarios. Se acompañan de cuestionarios virtuales con preguntas cerradas en una escala del

uno al cinco.

- Evaluación en Dos Fases: Alarcón-Aldana, Díaz y Callejas-Cuervo (2014) sugieren una evaluación en dos fases:
  - Primera Fase: Preguntas cerradas evaluadas de cero a cinco para medir la percepción del usuario.
  - Segunda Fase: Evaluación de criterios específicos como comunicación, estética, operatividad, facilidad de uso, aprendizaje, comprensión, entrenamiento y documentación.

(ii) Integración de las pruebas de usabilidad en el Ciclo de Vida del Software:

- Evaluación a lo Largo del Desarrollo: En Fernández, Insfran y Abrahão (2010), se propone realizar evaluaciones de usabilidad mediante informes en cada etapa del desarrollo (diseño, transformación de modelos e implementación). Cada informe debe revisar las etapas anteriores y documentar cualquier inconformidad.
- Integración del Usuario en Cada Fase: Algunos autores sugieren involucrar a los usuarios en cada fase del proyecto para corregir errores de manera temprana y mejorar la satisfacción y calidad del producto (Alarcón et al. 2007). Sin embargo, este enfoque puede estar limitado por la metodología de desarrollo utilizada.

Por otro lado, en la evaluación de la accesibilidad hay que centrarse en garantizar que una aplicación pueda ser usada por cualquier persona, independientemente de su condición física. Para su implementación debemos tener en cuenta diversos aspectos:

- Pautas de Evaluación: La evaluación puede considerar pautas estructurales y de programación. Algunas pautas pueden evaluarse mediante revisión de código, mientras que otras requieren una revisión detallada de los contenidos de la aplicación para asegurar calidad y coherencia.
- Herramientas de Evaluación Semiautomática: Herramientas como MATE (Mobile Accessibility Testing), Accessibility Checker , Accessibility Tools Frameworks y XValid se enfocan en aspectos clave de las WCAG 2.0 y MWBP, y son populares por ser rápidas y de libre acceso.

Posteriormente debemos considerar las ventajas/desventajas de realizar la evaluación de manera manual o automática.

- Revisión Manual Inicial vs. revisión Automática: Erickson et al. (2013) enumeran las limitaciones de las herramientas automáticas para cumplir con todos los estándares (WCAG, MWBP), es necesario realizar una revisión manual inicial.
- Validación con Usuarios: La accesibilidad debe ser evaluada por usuarios potenciales, siguiendo los principios de percepción, operación, comprensión y robustez definidos por la W3C y adaptados por la norma NTC5854.
- Evaluación Independiente: Generalmente, la accesibilidad y la usabilidad se evalúan por separado. Para obtener una visión completa, es necesario aplicar varios métodos de evaluación simultáneamente.

Criterio	Herramientas Evaluadas
Percepción	MATE, Accessibility Checker
Operación	Accessibility Tools Frameworks, XValid
Comprensión	MATE, XValid
Robustez	Accessibility Checker, Accessibility Tools Frameworks

Tabla 1.1: Criterios evaluados por herramientas automatizadas según MWBP.

- Resultados Incompletos: Las herramientas semiautomáticas, aunque útiles, no abordan completamente la calidad y coherencia de la información presentada, lo que puede llevar a evaluaciones incompletas. En la Tabla 1.1 muestra la evaluación de los criterios (Percepción, Operación, Comprensión y Robustez) por herramientas automatizadas según (Mobile Web Best Practices)

Debido a las limitaciones inherentes de las pruebas automáticas para evaluar completamente la accesibilidad en aplicaciones web y móviles, se recomienda comenzar con una evaluación manual, seguida de una evaluación automática. Esto asegura que la accesibilidad y la usabilidad se valoren adecuadamente a través de métodos complementarios y adaptados al contexto específico de uso de la aplicación.

En nuestro caso, vamos a combinar varias metodologías, ya que creemos que esta combinación resultará en un análisis más claro y eficiente. Además, consideramos que el trabajo de (Andrés Paniagua L 2020) se alinea estrechamente con nuestra idea de evaluar las pruebas de usabilidad.

1.1. Objetivos y metodología

Este Trabajo de Fin de Grado (TFG) tiene dos objetivos principales: uno relacionado con el desarrollo de la aplicación y otro centrado en las pruebas de usabilidad.

1. Desarrollo de la Aplicación: El primer objetivo consiste en el desarrollo de la aplicación para la empresa VERTEX:
   - Desarrollar los requisitos funcionales y de diseño establecidos con el cliente.
   - Implementar las características planificadas con el cliente y desarrollar y garantizar la integridad del código.
2. Pruebas de Usabilidad: El segundo objetivo se centra en evaluar la usabilidad de la aplicación desarrollada. Esto implica:
   - Diseñar y ejecutar pruebas de usabilidad con usuarios reales para identificar posibles problemas de interfaz, dificultades de navegación o cualquier otro aspecto que pueda afectar la experiencia del usuario.
   - Recopilar datos cualitativos y cuantitativos que permitan identificar las mejoras de la usabilidad que la aplicación requiere.

- Análisis de los resultados de las pruebas de usabilidad.
- Identificación de áreas de mejora en la aplicación.
- Recomendaciones para las decisiones de diseño y desarrollo basadas en los resultados obtenidos.

La metodología empleada para la implementación de la aplicación se describe en el Capítulo 5, y consiste básicamente en realizar revisiones semanales con el cliente, presentando entregas funcionales para asegurar ajustes ágiles y tener una retroalimentación continua.

La metodología empleada para las pruebas de usabilidad se describen en el Capítulo 1, y como ya hemos mencionado consiste en combinar varias metodologías, ya que creemos que esta combinación resultará en un análisis más claro y eficiente. Este artículo Andrés Paniagua L (2020) describe una metodología de seis fases para realizar una prueba completa de usabilidad. En nuestro caso, usamos cinco fases, omitiendo la accesibilidad.

Capítulo 2

Fase previa al diseño

En este capítulo se describe la problemática principal en la gestión de empleados y paquetes de la empresa, así como los desafíos económicos debido al crecimiento proyectado de la plantilla. Para resolver estos problemas, se propone una aplicación móvil que permitirá gestionar y asignar paquetes en tiempo real. Se evalúan los sistemas operativos, destacando Android por su familiaridad entre empleados y la experiencia previa del desarrollador, y se compara el uso de Kotlin frente a Java, decantándose por el primero debido a sus ventajas en legibilidad y seguridad. Para el almacenamiento de datos en tiempo real, se selecciona Firebase por su facilidad de implementación. Finalmente, se elige el patrón de diseño MVC para la arquitectura de la aplicación, facilitando la separación de responsabilidades y el mantenimiento del código.

2.1. Requisitos previos (Propios de la empresa)

El problema principal se encuentra en la gestión de los empleados y de los paquetes, ya que un administrativo tiene que tener en todo momento actualizados los estados de los paquetes (pendiente, en camino, entregado), y además los administrativos tienen que asignar y comunicar a los conductores el paquete correcto. La otra cuestión se trataba de la economía de la empresa, ya que era una realidad que estaban creciendo, sin embargo, su proyección les limitaba el presupuesto para poder invertirlo en labores que no fuesen entrega y recogida de paquetes.

El crecimiento de la empresa en cuanto a sus empleados era el siguiente:

- Conductores: llegar a 5 en un año y a 20 en 5 años.
- Oficinistas: llegar a 5 en 5 años.
- Administradores: 2.

En la figura 2.1 se muestra la previsión del crecimiento de los empleados de la empresa en 5 años, son un total de 27 empleados.

La solución es una aplicación móvil, que disponga de las funcionalidades de gestionar y asignar

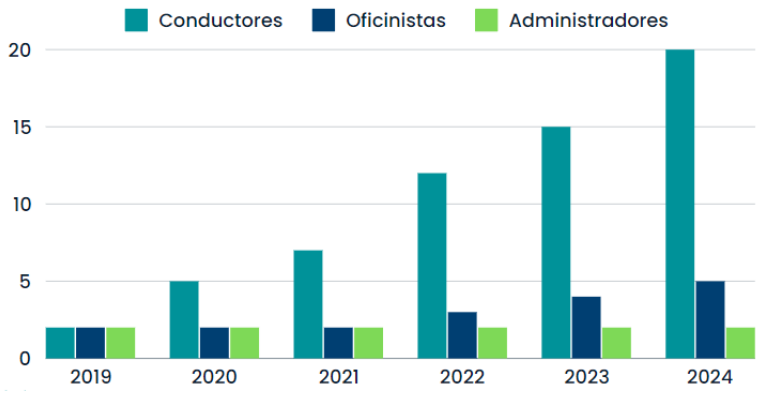

Figura 2.1: Crecimiento Vertex.

los paquetes en tiempo real. Una toma de requisitos se define por (cerrar un presupuesto, un *timing* y las tareas que se van a necesitar), y que lo firme el cliente (como un contrato): al estar aprobado por el cliente al inicio, cualquier cambio en él (añadir otra característica, por ejemplo), se presupuestará por separado. Sin embargo, el único propósito de esta toma de requisitos era cerrar una primera lista de las tareas y el *timing*, y dejar abierto que el cliente pueda requerirnos un cambio o varios.

Durante las primeras reuniones con el cliente, nos centramos en definir las funcionalidades clave de la aplicación. Identificamos los requisitos principales y creamos una maqueta de las pantallas.

2.2. Sistema operativo (Android)

Según la información de la que disponemos, los empleados están más familiarizados con Android y usan dispositivos Android, pero este dato no tiene que ser tan relevante ya que a futuro la plantilla va a crecer y esto podría implicar que los nuevos empleados puedan usar otras plataformas. Por lo que había que tener en cuenta tres factores importantes:

- La viabilidad de llevar a cabo un desarrollo en (Android, IOS, ambas): Como el desarrollador iba a ser solo una persona, se requería realizar un análisis exhaustivo de la capacidad para ofrecer un desarrollo eficiente y rápido. Es importante señalar que la experiencia que tenía el desarrollador era con Android, y que unos meses antes había trabajado con Kotlin en un proyecto con otra empresa. En contra posición había que aprender un nuevo lenguaje para escribir código en IOS.
- La empresa pueda proporcionar a los empleados los dispositivos: ¿Cuál tendrían mejor relación calidad-precio? Los dispositivos Android son más económicos y pueden garantizar el mismo servicio que los de Apple.
- Las limitaciones de cada plataforma en caso que solo se elija una: Android se ejecuta en una amplia variedad de dispositivos con diferentes versiones de Android y especificaciones de hardware. Esto puede dificultar la compatibilidad y la optimización de la

aplicación en todos los dispositivos, así como puede ser más desafiante optimizar el rendimiento de la aplicación. IOS por su parte es un sistema operativo cerrado, controlado por Apple. Esto significa que hay restricciones más estrictas sobre qué se puede hacer en una aplicación y cómo se puede distribuir. Cada aplicación iOS debe pasar por un proceso de revisión por parte de Apple antes de ser publicada en la App Store. Esto puede llevar tiempo y puede haber restricciones adicionales sobre el contenido y la funcionalidad de la aplicación. A diferencia de Android, iOS se ejecuta en dispositivos específicos fabricados por Apple. Esto puede limitar las opciones de hardware y funcionalidades específicas que se pueden aprovechar en la aplicación. El desarrollo en iOS generalmente implica un costo más alto debido a la necesidad de utilizar hardware y software específicos de Apple, como Mac y Xcode, así como el pago de una tarifa anual para acceder al programa de desarrolladores de Apple.

Finalmente elegimos Android, ya que este dispone de todas las herramientas necesarias para el desarrollo de la aplicación y que cumple con los aspectos que acabamos de mencionar.

2.3. Lenguaje (Kotlin)

Para decidir el IDE y el lenguaje de programación más adecuado para Android, existen dos opciones relevantes: Java y Kotlin.

- Java: Es el lenguaje de programación tradicional de Android y ha sido ampliamente utilizado durante muchos años. Tiene una amplia base de conocimientos y una gran cantidad de recursos y bibliotecas disponibles. Es un lenguaje orientado a objetos y relativamente fácil de aprender para aquellos que ya tienen experiencia en programación. Java tiene una gran comunidad de desarrolladores, lo que facilita encontrar soporte y soluciones a problemas comunes. Sin embargo, en comparación con Kotlin, Java puede ser más largo y requerir más código para lograr ciertas funcionalidades.
- Kotlin: Es el lenguaje de programación oficialmente respaldado por Google para el desarrollo de aplicaciones en Android. Ofrece una sintaxis más concisa y expresiva que Java, lo que facilita la escritura de código más limpio y legible. Es compatible con Java, lo que significa que se puede utilizar bibliotecas y código existentes de Java en proyectos de Kotlin. Este tiene un enfoque en la seguridad y la reducción de errores comunes de programación. La comunidad de desarrolladores de Kotlin está creciendo rápidamente, y hay una amplia gama de recursos y bibliotecas disponibles.

En general, Kotlin se ha vuelto cada vez más popular en la comunidad de desarrollo de Android debido a sus ventajas en términos de legibilidad, concisión y seguridad.

2.4. BBDD (Firebase)

El entorno que estamos construyendo precisa almacenar y cargar datos actualizados en tiempo real. Se puede utilizar una base de datos en tiempo real o una API de comunicación en tiempo real.

- Firebase Realtime Database: Es una plataforma desarrollada por Google que proporciona una base de datos en tiempo real. Puede almacenar y sincronizar datos en tiempo real entre los dispositivos y el backend de tu aplicación. Utiliza el modelo de eventos y notificaciones para mantener los datos actualizados en tiempo real. Puede realizar cambios en la base de datos y los datos se propagarán automáticamente a todos los dispositivos conectados.
- WebSockets: Los WebSockets son una tecnología de comunicación bidireccional en tiempo real que permite la transferencia de datos entre un cliente y un servidor de forma eficiente. Se puede utilizar WebSockets para establecer una conexión persistente entre la aplicación y el servidor, lo que permite enviar y recibir datos en tiempo real. Esto es útil cuando necesitas una comunicación en tiempo real más personalizada y controlada.

Después de analizar detenidamente nuestras necesidades y las características de las tecnologías disponibles, hemos optado por Firebase como solución para almacenar y cargar datos actualizados en tiempo real. En comparación, aunque los WebSockets ofrecen comunicación bidireccional en tiempo real, consideramos que Firebase proporciona una solución más sencilla y fácil de implementar para nuestras necesidades específicas.

2.5. Patrón de diseño (MVC)

La elección del patrón de diseño más adecuado para una aplicación móvil dependerá de varios factores, como los requisitos específicos del proyecto, la arquitectura general de la aplicación y las preferencias personales. Modelo-Vista-Controlador (MVC): Este patrón separa la aplicación en tres componentes principales: el Modelo (encargado de los datos y la lógica de negocio), la Vista (encargada de la interfaz de usuario) y el Controlador (mediador entre el Modelo y la Vista). MVC proporciona una clara separación de responsabilidades y facilita la modificación y el mantenimiento del código.

Capítulo 3

Diseño gráfico, funcionalidades y flujos de navegación

Antes de comenzar con el desarrollo, fue crucial establecer el esqueleto de la aplicación, definiendo las funcionalidades básicas y el flujo de navegación. Para ello, inicialmente recopilamos los requisitos del cliente y creamos una maqueta de las pantallas necesarias. A continuación, presentamos estas maquetas al cliente para su aprobación.

3.1. Requisitos funcionales

Una vez confirmado el esquema de la aplicación, pudimos iniciar el desarrollo, enfocándonos en tres requisitos funcionales principales:

- Pantalla de inicio de sesión: Esta pantalla es crucial para permitir el acceso seguro a la aplicación. Implementamos medidas de autenticación para garantizar que solo los usuarios autorizados puedan ingresar, utilizando métodos como la verificación por correo electrónico.
- Interfaz de selección: Se desarrolló una pantalla intuitiva y fácil de usar que permite al usuario elegir entre dos opciones principales: recoger o entregar paquetes. Esta interfaz está diseñada para ser clara y eficiente, reduciendo el tiempo necesario para que los usuarios realicen su selección y mejorando así la experiencia general del usuario.
- Creación de tareas: Se implementaron varias pantallas que facilitan la creación de nuevas tareas de recogida o entrega. Estas pantallas incluyen formularios detallados donde los usuarios pueden ingresar información específica sobre los paquetes, como las direcciones de origen y destino, las fechas y horas de recogida o entrega.

Estos requisitos funcionales son fundamentales para la operatividad de la aplicación y fueron diseñados con el objetivo de proporcionar una experiencia de usuario fluida y segura.

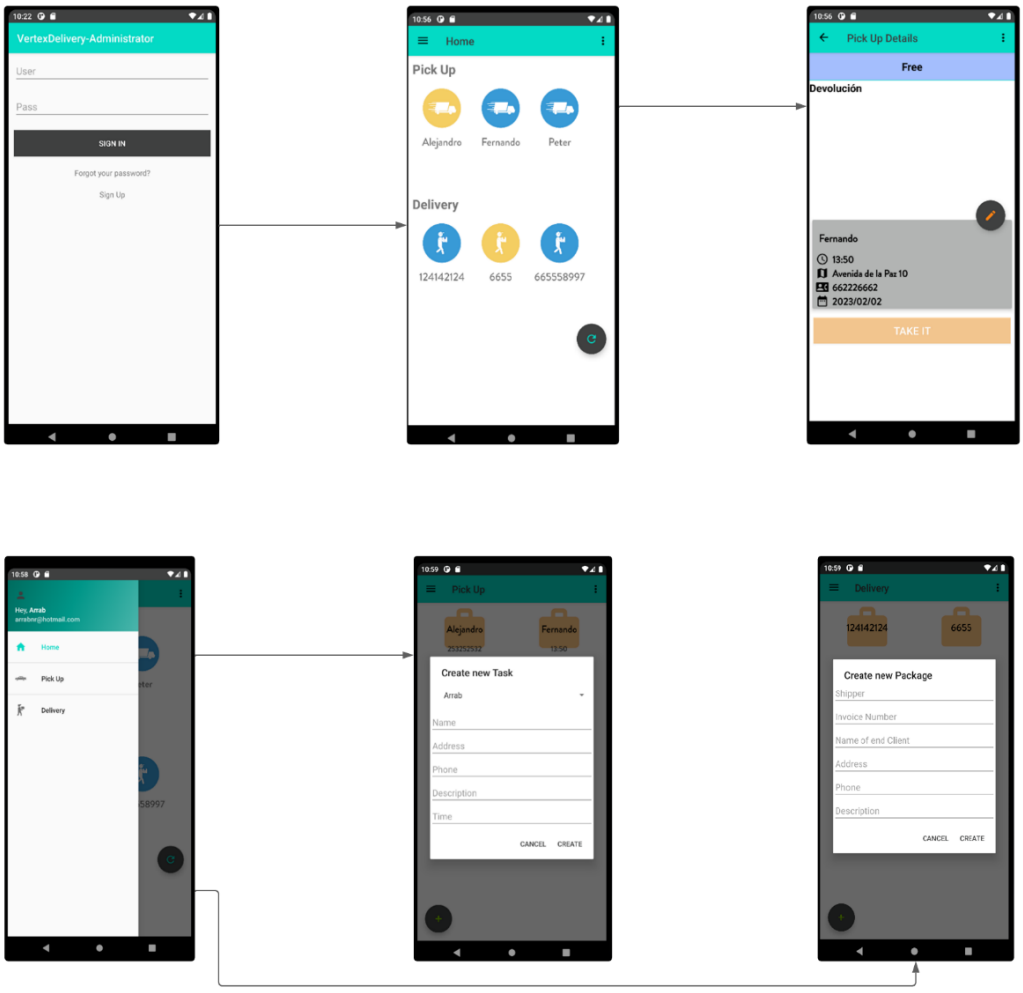

Figura 3.1: Maqueta de la versión final de la aplicación.

3.2. Descripción de las funcionalidades

En la figura 3.1 se muestran las funcionalidades de la versión final de la aplicación. A continuación, se describen estas funcionalidades:

1. Inicio de sesión: Esta es la primera pantalla que aparece al abrir la aplicación. Contiene un campo usuario (email) y un campo contraseña. También incluye un TextBottom en los casos de *Forgot Passowrd* y *Sign Up*.
2. Registrarse: Esta pantalla aparece cuando pulsamos el botón de la pantalla anterior *Sign Up*, y en la cual aparece un formulario para rellenar y enviar con los siguientes datos(nombre, apellido, correo electrónico y contraseña). Una vez enviados los datos, tiene que volver a la pantalla de Inicio de sesión, para poder acceder a la aplicación.
3. Recordar contraseña: En la pantalla *Forgot Password*, hay que introducir el correo electrónico con el que se realizó el registro y pulsar el botón enviar, en este momento aparece un mensaje emergente *Check your mail inbox* confirmando el envío. Una vez recibido el

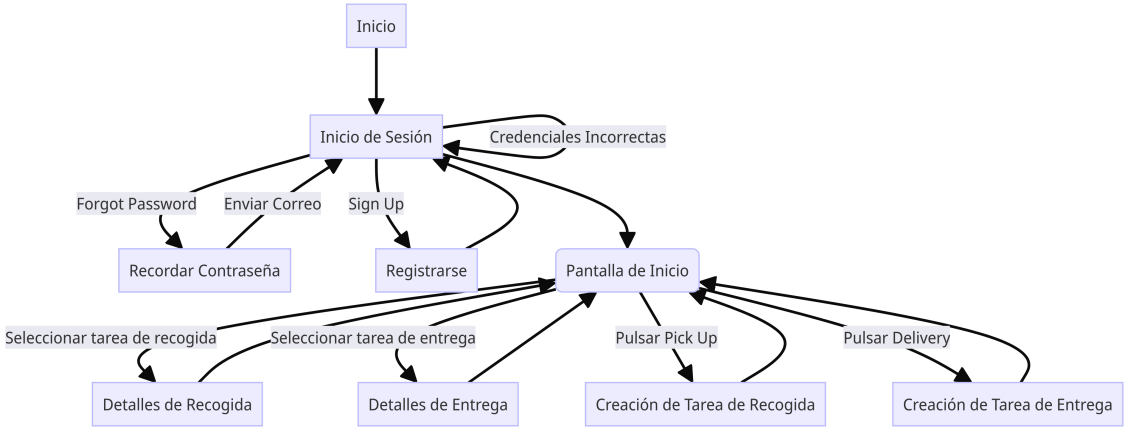

Figura 3.2: Diagrama del flujo de navegación.

correo que contiene el enlace para actualizar la contraseña, aparece una pantalla emergente para introducir la nueva contraseña. Una vez restablecidos los datos de acceso, se podrá volver a la página de inicio de sesión para acceder a la aplicación.

4. Home: La pantalla principal de la aplicación se divide en tres partes. La parte superior contiene los menús y la parte central se divide en recogida y entrega. Todos los paquetes *Tareas* se dividen en tres colores: Azul (Pendiente), Amarillo (En camino), y Verde (Entregado o Finalizado).
5. Detalles Recogida: Esta pantalla aparece cuando seleccionamos una de las tareas Recogida en la pantalla Home. En ella se desglosan los datos del paquete. El estado del paquete (Free, In Progress, Done), comentario, hora de recogida, dirección, número de contacto y fecha de creación de la tarea. Luego en la parte inferior está el botón de cambio de estado (recoger paquete, finalizado).
6. Detalles Entrega: Similar a la pantalla de detalles de recogida, esta muestra los datos de la entrega. Incluye el estado del paquete (Free, In Progress, Done), comentario, número de factura, nombre del cliente, dirección, número de contacto y fecha de creación de la tarea. Luego en la parte inferior está el botón de cambio de estado (recoger paquete, finalizado).
7. Creación de tarea de Recogida: Para llegar a esta pantalla hay que seleccionar en la pantalla *Home* el correspondiente botón *Pick Up*. En el área de contenido de la pantalla se hayan las tareas existentes sin terminar, y en la parte inferior hay un botón +, para abrir una ventana emergente que contiene el formulario de creación de recogida.
8. Creación de tarea de Entrega: Similar a la creación de tarea de recogida, se accede a esta pantalla al seleccionar el botón *Delivery* en la pantalla *Home*. En la vista principal de la pantalla se hayan las tareas existentes sin terminar, y en la parte inferior hay un botón +, para abrir una ventana emergente que contiene el formulario de creación de entrega.

3.3. Diseño del flujo de navegación

En la figura 3.2 se muestra el flujo de navegación de las pantallas. A continuación, se describen paso a paso:

1. Inicio: Ejecutar la aplicación.
2. Inicio de Sesión:
   - Decisión: ¿El usuario introdujo credenciales correctas?
     - Sí → [5. Pantalla Principal (Home)]
     - No → Mostrar mensaje de error y permanecer en la Pantalla de Inicio de Sesión.
   - Acción: Pulsar *Forgot Password* → [3. Recordar Contraseña]
   - Acción: Pulsar *Sign Up* → [2. Registrarse]
3. Registrarse:
   - Acción: Completar y enviar formulario.
   - Acción: Redirigir a la Pantalla de Inicio de Sesión → [2. Inicio de Sesión]
4. Recordar Contraseña:
   - Acción: Introducir correo y pulsar Enviar.
   - Acción: Mostrar mensaje emergente (Check your mail inbox).
   - Acción: Redirigir a la Pantalla de Inicio de Sesión → [2. Inicio de Sesión]
5. Pantalla Principal (Home):
   - Acción: Seleccionar una tarea de recogida → [6. Detalles de Recogida]
   - Acción: Seleccionar una tarea de entrega → [7. Detalles de Entrega]
   - Acción: Pulsar botón (Pick Up) → [8. Creación de Tarea de Recogida]
   - Acción: Pulsar botón (Delivery) → [9. Creación de Tarea de Entrega]
6. Detalles de Recogida:
   - Acción: Cambiar estado del paquete.
   - Acción: Redirigir a la Pantalla Principal (Home) → [5. Home]
7. Detalles de Entrega:
   - Acción: Cambiar estado del paquete.
   - Acción: Redirigir a la Pantalla Principal (Home) → [5. Home]
8. Creación de Tarea de Recogida:
   - Acción: Mostrar tareas existentes sin terminar.
   - Acción: Pulsar + para abrir formulario de creación de recogida.
   - Acción: Completar y enviar formulario.
   - Acción: Redirigir a la Pantalla Principal (Home) → [5. Home]
9. Creación de Tarea de Entrega:
   - Acción: Mostrar tareas existentes sin terminar.

- Acción: Pulsar + para abrir formulario de creación de entrega.
- Acción: Completar y enviar formulario.
- Acción: Redirigir a la Pantalla Principal (Home) → [5. Home]

3.4. Diseño gráfico de las pantallas

El objetivo de este apartado es proporcionar una guía visual y estructural del diseño de la interfaz de usuario de la aplicación. A través del maquetado de pantallas, se busca definir claramente cómo se organizan y presentan los distintos elementos visuales y funcionales de la aplicación, facilitando así el desarrollo y la implementación coherente del proyecto.

El diseño gráfico de las pantallas es crucial para asegurar una experiencia de usuario fluida y consistente. La figura 3.1 muestra la versión final de la aplicación, hecha en Canva (2024).

El estilo visual de la aplicación se basa en un enfoque minimalista (prioriza la simplicidad y la eliminación de elementos innecesarios), utilizando una paleta de colores suaves, los colores principales incluyen tonos de azul, amarillo y verde, que se utilizan para representar los diferentes estados de los paquetes (pendiente, en camino, entregado/finalizado):

- Azul: Representa los paquetes pendientes y se utiliza para botones e iconos de acciones principales.
- Amarillo: Indica los paquetes en camino, proporcionando un contraste visual claro.
- Verde: Señala los paquetes entregados o finalizados, destacando el éxito y la finalización de tareas.

Se ha seleccionado una tipografía clara y legible para todos los textos de la aplicación, asegurando que la información sea fácil de leer en diferentes dispositivos y tamaños de pantalla.

Se emplean iconos sencillos y universalmente reconocibles para acciones comunes, como iniciar sesión, registrarse, enviar formularios, y cambiar estados de tareas. Los iconos ayudan a mejorar la comprensión y la velocidad de uso de la aplicación.

Capítulo 4

Implementación de la aplicación

En este capítulo, se describe la implementación de la aplicación, comenzando con el desarrollo basado en un *template* visto en YouTube. Este enfoque proporcionó una base sólida, confirmando la idoneidad de las tecnologías elegidas y ofreciendo una guía clara que ahorró tiempo y recursos. La estructura del proyecto se detalla desde la instalación del IDE (Android Studio), la configuración inicial, el uso de control de versiones con GitLab, y la integración de Firebase. Se describe la organización del proyecto utilizando el patrón Modelo-Vista-Adaptador (MVA o MVC), la creación de las actividades y recursos en Android Studio, y la implementación específica de la pantalla DetailDelivery, que permite editar y visualizar detalles de entrega. La configuración de la interfaz de usuario y el modelo de datos en Firebase, junto con el uso de MutableLiveData para la actualización reactiva de la UI, son elementos clave. Además, se destacan la creación del adaptador para gestionar listas en RecyclerView y la conexión entre la vista y el modelo a través del adaptador, finalizando con la implementación del fragmento que maneja la interfaz y la lógica de negocio.

4.1. Desarrollo a partir de un plantilla

Para optimizar nuestro proceso de desarrollo y asegurarnos de que estábamos en el camino correcto, decidimos utilizar el *template* publicado en YouTube `https://www.youtube.com/watch?v=qSl6I_4I31k&list=PLaoF-xhnnrRUF02btNC2npGFnQ312TvdU&index=2&ab_channel=EDMTDev`. Este se enfoca en el desarrollo de una aplicación móvil utilizando Kotlin y Firebase. En este tutorial, se cubren los aspectos fundamentales para construir una aplicación de comida a domicilio, incluyendo la configuración del proyecto en Android Studio, la creación de interfaces de usuario, y la integración con Firebase para manejar la autenticación, la base de datos en tiempo real, y el almacenamiento.

El tutorial confirmó que el entorno de desarrollo elegido, Kotlin para la programación y Firebase para la gestión de la base de datos, eran los adecuados para nuestro proyecto. Ver cómo estas tecnologías se implementaban en un ejemplo práctico nos dio la confianza necesaria para

seguir adelante con nuestra elección inicial.

El tutorial proporcionaba una visión clara del resultado final, permitiéndonos imaginar cómo podría ser nuestra aplicación una vez completada. Este aspecto nos facilitó la creación de un plano mental y visual de la aplicación. Esto también nos permitió ahorrar una cantidad significativa de tiempo y recursos. En lugar de empezar desde cero y resolver problemas técnicos por nuestra cuenta, pudimos seguir una guía probada que nos llevaba paso a paso a través del proceso de desarrollo. Esto no solo aceleró nuestro progreso, sino que también redujo el margen de error.

Aunque el tutorial ofrecía una guía detallada, también proporcionó la flexibilidad necesaria para adaptar el proyecto a nuestras necesidades específicas. Pudimos modificar y personalizar el código y las funcionalidades para alinearlos con los requisitos particulares de nuestra aplicación. Esta adaptabilidad fue crucial para asegurar que el producto final cumpliera con nuestras expectativas y necesidades específicas.

El uso de una aplicación preexistente como referencia y el apoyo del tutorial detallado fueron decisiones estratégicas que nos permitieron optimizar el desarrollo de nuestra aplicación. Estos recursos facilitaron la planificación, validación y ejecución de nuestro proyecto, asegurando que nos mantuviéramos dentro de los límites de tiempo y recursos disponibles.

4.2. Estructura del proyecto

En esta sección, describiremos la configuración inicial del proyecto, desde la instalación del entorno de desarrollo hasta la organización de la estructura del código.

- Instalación del IDE y Configuración Inicial:

  El primer paso consistió en instalar el IDE, en este caso, Android Studio, que era la única herramienta necesaria para el desarrollo local. Las demás herramientas se utilizarían a través de plataformas web.
- Control de Versiones y Elección de Plataforma:

  El segundo paso implicó la creación de un proyecto de control de versiones. El control de versiones es una práctica esencial en el desarrollo de software, facilitando la gestión de cambios en el código. Optamos por trabajar con GitLab debido a su Integración Continua (CI/CD) y sus completas herramientas de gestión de proyectos.
- Configuración del Proyecto:

  Una vez configurado el entorno, creamos el proyecto y seleccionamos Kotlin como lenguaje de programación. Android Studio ya incluye una versión de Gradle, un sistema de construcción y gestión de dependencias que automatiza la compilación y el empaquetado del código fuente.
- Integración de Firebase y Asociación con Git:

  El siguiente paso fue integrar el componente de Firebase en nuestro proyecto para comenzar con el desarrollo del código. Actualizamos las dependencias en el archi-

vo `build.gradle` para incluir las funcionalidades de Firebase. Finalmente, asociamos nuestro proyecto con Git para mantener un historial de cambios. Sin embargo, nos demoramos en abordar este punto y lo realizamos más tarde, lo cual comentaremos en el capítulo 5.

A continuación, mostramos el código referente a las dependencias que había que importar en el proyecto:

```
dependencies {
    implementation 'com.google.firebase:firebase-database:19.3.1'
    implementation 'com.google.firebase:firebase-auth:19.3.1'
```

- Organización del Proyecto en MVA o MVC:

  Iniciamos la organización del proyecto utilizando el patrón Modelo-Vista-Adaptador (MVA o MVC). Creamos las carpetas principales del proyecto:

  - Modelo *Model*: Representa los datos y la lógica de negocio de la aplicación.
  - Vista *View*: Es la capa de presentación o interfaz de usuario.
  - Adaptador *Adapter*: Actúa como un intermediario entre el Modelo y la Vista.
  - *Activity*: Creamos la clase *Activity* de cada una de las funciones de las pantallas *Login, Register, ForgotPassword, Home*. Esta clase va a contener las operaciones y llamadas necesarias para completar una acción, como podría ser la de cargar una pantalla para que se pueda visualizar, como por ejemplo *Home*.
  - *Resources*: La Figura 4.1 muestra la carpeta *res*. Esta carpeta desempeña un papel crucial en el desarrollo de aplicaciones Android al gestionar todos los recursos gráficos y de diseño necesarios para la aplicación. Dentro de esta carpeta, creamos los *Layouts* en formato .xml para definir el diseño y la estructura de las interfaces de usuario.

    Los *Layouts* en formato XML se utilizan para pintar el diseño de las pantallas de la aplicación. Estos archivos XML describen la disposición de los elementos de la interfaz de usuario, como botones, texto, imágenes, y otros *widgets*.

    Además de los *Layouts*, la carpeta res almacena todos los recursos gráficos necesarios para la aplicación, tales como imágenes, colores, cadenas de texto y estilos. La organización de estos recursos se realiza en subcarpetas específicas dentro de res, tales como:

    - *Drawable*: Contiene imágenes y gráficos de la aplicación. Aquí se pueden almacenar archivos en formatos como .png, .jpg, .gif.
    - *Layout*: Contiene los archivos XML de los Layouts, donde se definen las interfaces de usuario.
    - *Values*: Contiene archivos XML para definir recursos como cadenas (strings.xml), colores (colors.xml), dimensiones (dimens.xml), y estilos (styles.xml).
    - *Mipmap*: Almacena los iconos de la aplicación en diferentes resoluciones para adaptarse a diferentes tamaños de pantalla y densidades de píxeles.

    En la carpeta de *layout* tenemos varios tipos de archivos .xml:

    - *Activity*: Este archivo define el diseño de la pantalla de una actividad. En An-

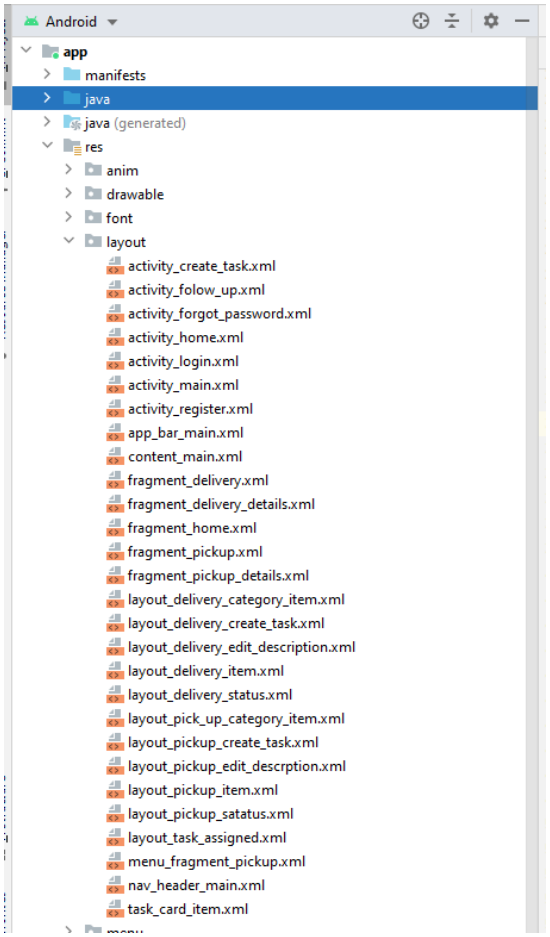

Figura 4.1: Carpeta *Resources layouts*.

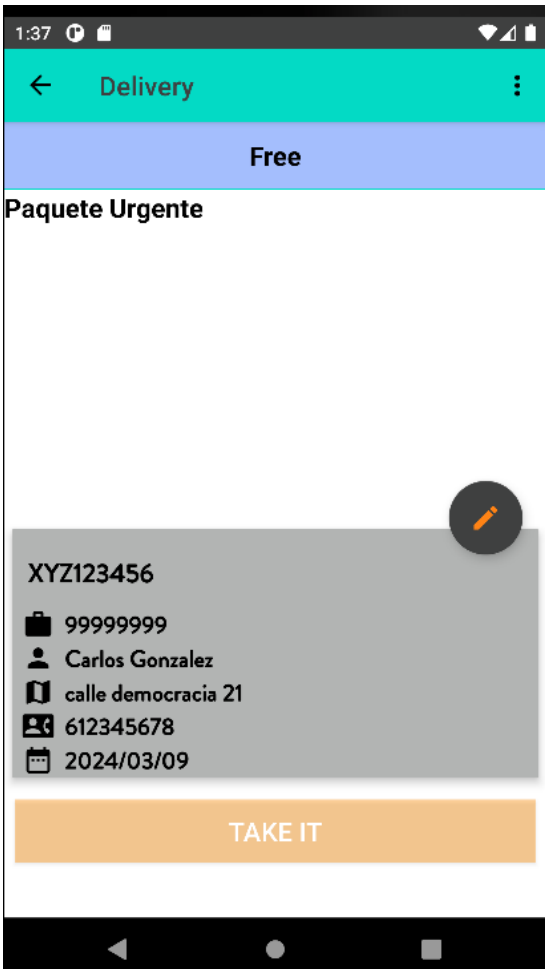

Figura 4.2: Captura de la pantalla Detalle de Entrega (Detail Delivery).

droid, una actividad representa una sola pantalla con una interfaz de usuario, similar a una página en una aplicación web.

- *Fragment*: Este archivo define el diseño de un fragmento. Los fragmentos son componentes modulares de una actividad que permiten crear interfaces más flexibles y reutilizables. Un fragmento puede ser pensado como una sub-pantalla dentro de una actividad.
- *Layout*: Este archivo define el diseño de una interfaz específica, probablemente parte de una funcionalidad más grande. Este archivo lo estamos utilizando para formularios o paneles donde los usuarios pueden ingresar y enviar información.

4.3. Descripción de la implementación de (DetailDelivery)

A continuación, vamos a detallar el desarrollo del código necesario para habilitar la funcionalidad de la pantalla figura 4.2. Esta muestra la pantalla *Detail Delivery*, en la que el usuario puede editar la descripción del paquete, visualizar los datos correspondientes a ese paquete y

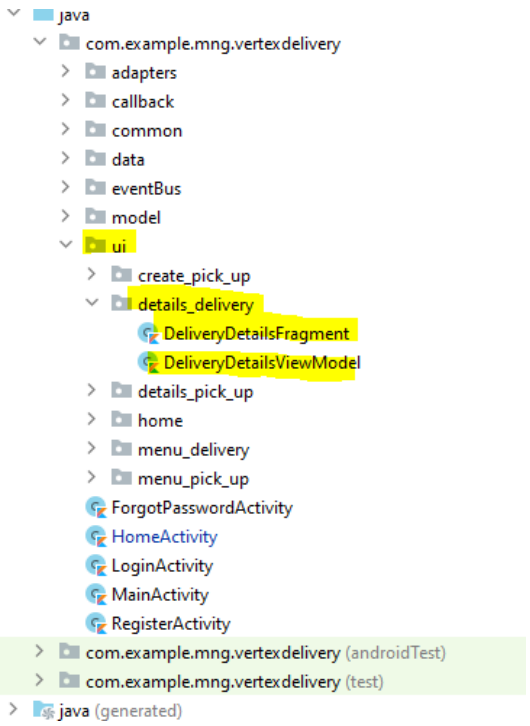

Figura 4.3: Estructura de archivos de DetailDelivery.

cambiar el estado del paquete. Y donde la función DetailDelivery se encarga de realizar las modificaciones y mostrar los datos.

A continuación, para desarrollar las funcionalidades relacionadas con DetailDelivery, tenemos que crear la estructura de archivos como se muestra en la figura 4.3 donde la carpeta Details_Delivery contiene dos clases, DeliverDetailsFragment y DeliveryDetailsViewModel. Estas dos clases van a manejar la interfaz y su lógica, para que muestre los datos y el usuario pueda interactuar con ellos.

La interfaz se implementa usando el *layout* fragment_delivery_details. La Figura 4.4 muestra la interfaz visual del DeliveryDetails donde el *layout* utiliza CoordinatorLayout como contenedor principal, lo que permite coordinar las interacciones de sus hijos, especialmente útiles cuando se combinan elementos como AppBarLayout y NestedScrollView.

Dentro del AppBarLayout, se incluye un CollapsingToolbarLayout que proporciona un efecto de colapso al abrir la pantalla de detalles. Los LinearLayout y TextView dentro del CollapsingToolbarLayout muestran información esencial sobre el estado de la entrega y su descripción.

Y finalmente, el FloatingActionButton permite al usuario editar la descripción de la entrega. Detalles del (nombre de la entrega, número de factura, cliente final, dirección, contacto y fecha), está organizado dentro de CardView para una presentación visualmente agradable. Finalmente, un botón grande en la parte inferior permite al usuario tomar una acción específica, como confirmar la recepción de la entrega

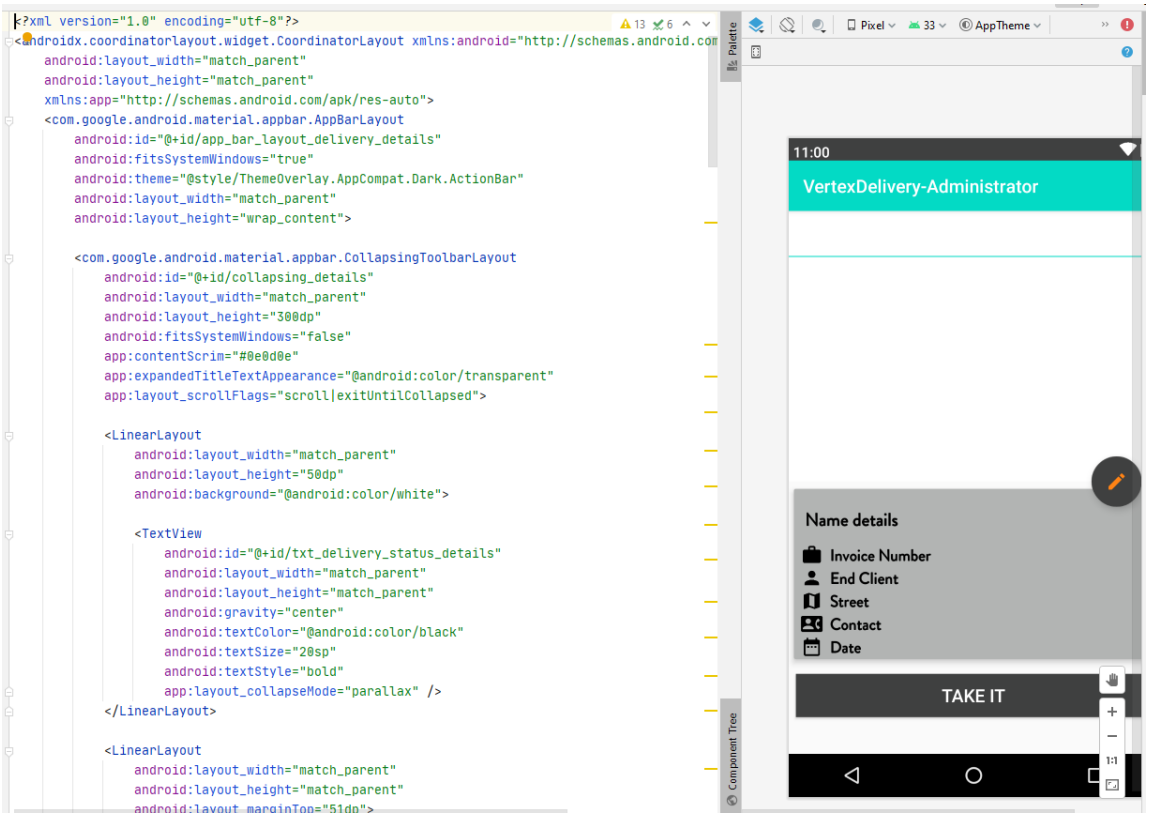

Figura 4.4: Detalle de entrega (Detail Delivery Layout Fragment).

Una vez definida la interfaz tenemos que crear el Modelo de la aplicación, i.e., la clase que va a contener el código cuya función es la de modelar los datos para que Firebase pueda gestionarlos. Para ello tenemos que definir la estructura de los datos en la base de datos y crear un prototipo de los datos que se van a necesitar. Estos son los datos del modelo de datos: package_id, invoiceNumber, shipper, phone, address, endClientName, status, image, description, date, dateOperation. [1]

Por falta de espacio, únicamente mostramos el código de la clase DeliveryDetailsViewModel que se ocupa de refrescar los datos de manera reactiva (cuando le llega un dato se ejecuta).[2]

```
1         class DeliveryDetailsViewModel : ViewModel() {
2         private var mutableLiveData: MutableLiveData<DeliveryModel>? = null
3         private var mutableStatusLiveData: MutableLiveData<DeliveryModel>? = null
4         init {
5             mutableStatusLiveData = MutableLiveData()
6         }
7   //llamadas para interactuar con la view y sus datos
8         fun getMutableLiveData(): MutableLiveData<DeliveryModel> {
9
10        }
11  //llamadas para interactuar con la view y el estado
```

[1] Para conveniencia del lector, el código completo de la clase DeliveryModel está disponible en el apéndice A.1.
[2] La clase DeliveryDetailsViewModel está disponible en el apéndice A.2.

```
12          fun getMutableStatusLiveData(): MutableLiveData<DeliveryModel> {
13
14          }
15  //establecer el estado en la view
16          fun setStatusModel(statusModel: DeliveryModel) {
17
18          }
19  }
```

donde en la linea 2 se crea el objeto MutableLiveData. El objeto MutableLiveData es a su vez una clase de la arquitectura de componentes de Android que permite almacenar y observar cambios en datos de manera reactiva. Los componentes de la UI observan estos datos, con el método getMutableLiveData (en la línea 8), y se actualizan automáticamente cuando cambian, con el método setStatusModel (en la línea 16).

Posteriormente, para poder conectar la vista con el modelo es necesario crear un adaptador o controlador. La vista transmite la interacción del usuario al adaptador, que funciona como intermediario entre la vista y el modelo.

Como por ejemplo la clase DeliveryAdapter. Esta clase se utiliza para gestionar y vincular una lista de objetos DeliveryModel a una vista de RecyclerView, la cual se usa para mostrar eficientemente listas grandes de datos, como por ejemplo los datos que se cargan en la pantalla de las entregas (ver Figura 4.2).

```
1
2       class DeliveryAdapter (internal var context: Context,
3       internal var deliveryCategoryModels: List<DeliveryModel>):
4       RecyclerView.Adapter<DeliveryAdapter.MyViewHolder>(){
5       // Clase ViewHolder interna que mantiene y maneja los
6       // elementos de la vista
7           inner class MyViewHolder(itemView:View)
8           :RecyclerView.ViewHolder(itemView),
9           View.OnClickListener{
10      // Declaración de vistas del layout
11              var delivery_invoice_nummber: TextView?= null
12              var delivery_image: CircleImageView?= null
13      // Listener para manejar los clics en los elementos del RecyclerView
14              internal var listener:IRecycleItemClickListener? = null
15      // Método para establecer el listener
16          fun setListener(listener:IRecycleItemClickListener){
17              this.listener = listener
18          }
19      // Inicialización de vistas y configuración del click listener
20          init {
21              delivery_invoice_nummber = itemView.findViewById
22              (R.id.txt_category_delivery_invoice_number) as TextView
23              delivery_image = itemView.findViewById
24              (R.id.img_category_delivery_image) as CircleImageView
25              itemView.setOnClickListener(this)
```

```
26            }
27        }
```

la implementación de esta clase muestra que el constructor toma un contexto de la aplicación y una lista de datos de entrega deliveryCategoryModels. La clase MyViewHolder es una clase interna que extiende RecyclerView.ViewHolder e implementa View.OnClickListener. Esta clase contiene las vistas individuales de cada ítem en el RecyclerView. El init inicializa las vistas (TextView para el número de factura y CircleImageView para la imagen de la entrega) y configura un OnClickListener para el ítem. El método setListener permite configurar un interceptor (listener) de clics (IRecycleItemClickListener).[3]

Por último, la clase *Fragment* representa un componente de la interfaz de usuario (UI) que define un fragmento de la interfaz de usuario y su comportamiento. Es una parte modular y reutilizable de la interfaz de usuario que puede contener una porción de la interfaz de usuario y puede combinarse con otros Fragments en una sola Activity para crear una interfaz de usuario más compleja y flexible. Como la clase DeliveryDetailsFragment que muestra los detalles de una tarea de entrega y permite al usuario realizar acciones como cambiar el estado de la tarea y editar la descripción.[4] A continuación, se muestran los aspectos más relevantes de su implementación:

1. Declaración de Variables: deliveryDetailsViewModel es una variable que interactúa con la lógica del negocio para obtener y actualizar datos. TextViews y Buttons son variables para mostrar y gestionar la información del paquete en la interfaz de usuario.
2. Método onCreateView: se inicializa deliveryDetailsViewModel del ViewModel para obtener y observar datos. Usamos el LayoutInflater que es una clase de Android que se utiliza para convertir archivos de layout XML en objetos View en tiempo de ejecución.
3. Métodos para Actualizar Firebase: submitToFirebase guarda el estado del paquete en Firebase.Y el submitToFirebaseDescription guarda la descripción del paquete en Firebase.
4. Mostrar Información del Paquete: el displayInfo muestra los detalles del paquete en las vistas correspondientes.

4.3.1. Actualización de BBDD en la nube

En este apartado vamos a ver como se actualizan los datos en Firebase, y ver como registra los estados actuales de los paquetes. *Ejemplo 4.5*

- *Address*: la dirección de entrega.
- *Date*: fecha de la creación de la tarea.
- *DateOperation*: fecha y hora exacta en el momento que se cambia el estado al paquete.
- *Description*: comentario del detalle del paquete.
- *EndClientName*: nombre del cliente.

[3] El código completo de la clase DeliveryAdapter está disponible en el apéndice A.3.
[4] El código completo de la clase DeliveryDetailsFragment está disponible en el apéndice A.4.

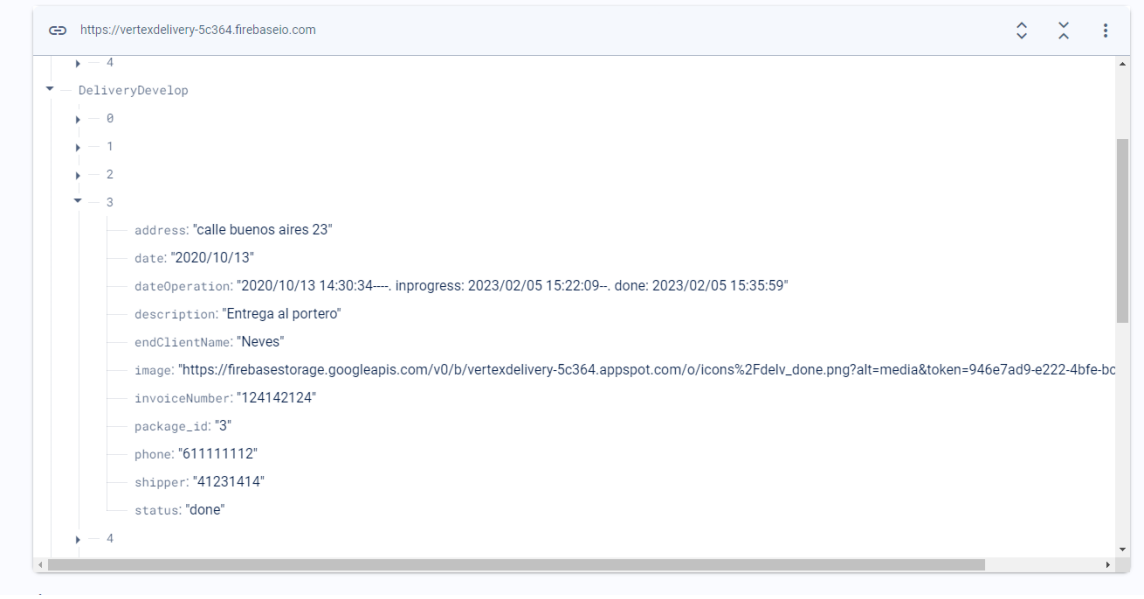

Figura 4.5: Datos en Firebase.

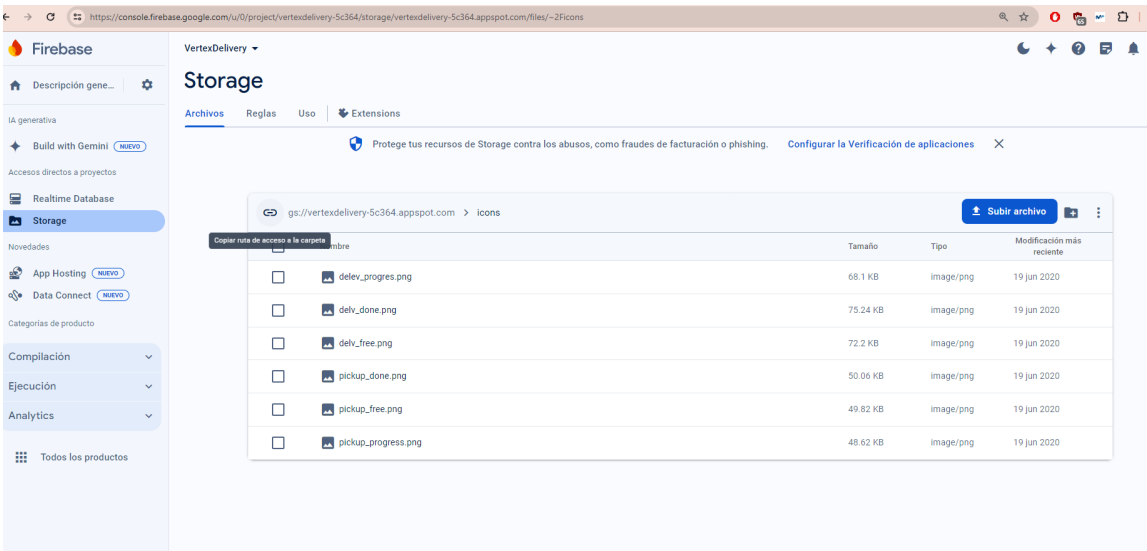

Figura 4.6: Carpeta de imágenes en Firebase.

- *Image*: imagen del estado actual del paquete. Esta guardado en la carpeta storage/icons en Firebase figura 4.6, accedemos a la imagen mediante de la URL. Esto lo hemos implementado de esta forma para optimizar el código y no consumir recursos innecesariamente.
- *InvoiceNumber*: número de la factura.
- *Package_id*: introducimos el id secuencial de la tarea, para comprobaciones a la hora de modificaciones.
- *Phone*: número de contacto.
- *Shipper*: remitente.
- *Status*: estado acutal.

```
1     //Guardar en FireBase los datos del estado del paquete
2        private fun submitToFirebase(it: DeliveryModel?) {
3            waitingDialog!!.show()
4            val data = FirebaseDatabase.getInstance().getReference(Common.DELIVERY_REF)
5            val data2 = data.child(Common.deliverySelected!!.package_id!!)
6            data2.child("status").setValue(it!!.status)
7            data2.child("dateOperation").setValue(it!!.dateOperation)
8            data2.child("image").setValue(it!!.image)
9            waitingDialog!!.dismiss()
10       }
```

Este código se utiliza para guardar datos en Firebase. La función submitToFirebase crea una variable local (data) para abrir una instancia de Firebase y especificar la tabla a la que se desea acceder. A continuación, se define otra variable (data2) para identificar el registro que se va a almacenar, utilizando el package_Id proporcionado por la clase que contiene esta función. Este package_Id, junto con otros datos necesarios, se obtiene del objeto MutableLiveData.

```
1     override fun onDataChange(p0: DataSnapshot) {
2        if (p0.exists()) {
3            Common.deliverySelected = p0.getValue(DeliveryModel::class.java)
4            FirebaseDatabase.getInstance()
5            .getReference(Common.DELIVERY_REF)
6            .child(event.deliveryCategoryModel.package_id!!)
7            .addListenerForSingleValueEvent(object : ValueEventListener {
8             override fun onDataChange(p0: DataSnapshot) {
9                if (p0.exists()) {
10                   Common.deliverySelected =
11                   p0.getValue(DeliveryModel::class.java)
12                   findNavController(R.id.nav_host_fragment)
13                   .navigate(R.id.nav_delivery_details)
```

A través de la función OnDataChange (que se encuentra en la clase HomeActivity) se extraen los datos, lo hacemos mediante el MutableLiveData, ya que lo que necesitamos es tenerlo en todo momento actualizado. Este MutableLiveData es alimentado a su vez por una variable global que va a contener una instancia de los datos actuales de la BBDD.

Capítulo 5

Ciclo de desarrollo

En este capítulo se detallan las etapas y procesos involucrados en el ciclo de desarrollo de este proyecto. Incluye la definición del alcance del trabajo, la planificación, los diagramas de las pantallas previos a la aplicación final y la gestión del control de versiones usado.

5.1. Alcance del encargo y seguimiento (semanal)

Como se menciona en el capitulo 1, el objetivo principal de este proyecto es mejorar la gestión y optimización de la entrega y recogida de paquetes. Se busca optimizar los procesos para manejar un mayor volumen con mayor precisión. En esta sección explicamos cómo lograrlo.

Hemos adoptado una metodología de trabajo que incluye revisiones semanales con el cliente. Esta frecuencia permite realizar ajustes de manera ágil y económica, manteniendo el proyecto alineado con las expectativas y necesidades del cliente. Cada semana, se presentan entregas funcionales y visualizables que el cliente puede evaluar y probar, asegurando una retroalimentación inmediata y continua.

Inicialmente, las primeras semanas se dedican a la configuración del entorno de desarrollo y a la implementación de la primera funcionalidad: el sistema de Login y Registro. Tras esta fase, el ciclo de trabajo semanal asegura que cada sábado se revisen los avances y se presente una nueva funcionalidad. Las reuniones se realizan vía *Skype*, lo que facilita la comunicación y permite mostrar los cambios directamente al cliente.

El desarrollo sigue una secuencia lógica, comenzando con el inicio de sesión y registro, seguido por la página principal *Home*, la creación y gestión de tareas de entrega y recogida, la visualización de los detalles de los paquetes, y finalmente, la funcionalidad de cierre de sesión. Esta estructura garantiza una construcción gradual y cohesiva del sistema.

5.1.1. Cronograma general y plan de trabajo

En este apartado se presenta un cronograma detallado que incluye las fechas y los plazos para cada fase del proyecto. También se describe el plan de trabajo, que desglosa las tareas específicas. Este plan garantiza que todas las actividades se realicen de manera organizada y dentro de los tiempos establecidos, facilitando el seguimiento y la gestión efectiva del progreso del proyecto.

Primeras Semanas

Configuración del entorno y desarrollo de la primera funcionalidad: inicio de sesión y registro.

Fase de Entregas Semanales

- Formato de Entregas: De sábado a sábado, con revisión semanal. Aunque no siempre se cumplía, en ese caso se aplazaba al siguiente sábado.
- Herramienta de Comunicación: Reuniones vía *Skype*, con compartición de pantalla para mostrar los avances.

Secuencia de Desarrollo

1. inicio de sesión y registro: Desarrollar y presentar el sistema de autenticación de usuarios.
2. Página Principal (Home): Crear y presentar la interfaz principal del sistema.
3. Creación de Tareas de Entrega y Recogida: Desarrollar y presentar la funcionalidad para crear y gestionar tareas.
4. Detalles de los Paquetes: Implementar y presentar la visualización y administración de los detalles de los paquetes.
5. Cierre de sesión: Desarrollar y presentar la funcionalidad de cierre de sesión. Sin embargo, esta función no se llegó a desarrollar ya que justo en el momento de comenzar esta funcionalidad, se dio por cerrado el proyecto.

5.2. Diagramas preliminares del flujo de navegación

Tras discutir los requisitos funcionales en las primeras reuniones con el cliente en detalle y explorar diferentes enfoques para abordarlos, logramos llegar a un diseño que satisfacía las necesidades tanto del cliente como del equipo de desarrollo. La figura 5.1 muestra el diagrama de navegación entre las pantallas y, dentro de cada pantalla, las funcionalidades que estas contienen. Esta figura es el primer diseño oficial de la aplicación completa.

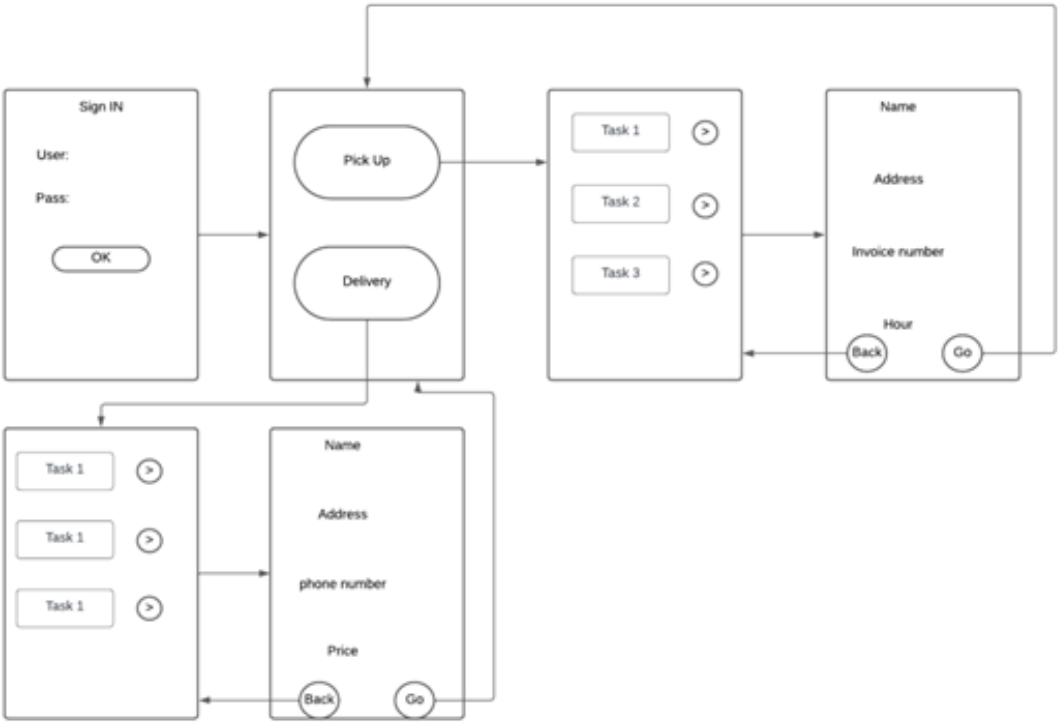

Figura 5.1: Diagrama de flujo de pantallas 1.0.

A medida que avanzamos en la implementación de las primeras características, como el inicio de sesión, realizamos una revisión inicial con el cliente. Durante esta sesión, el cliente expresó interés en modificar la funcionalidad de entrega, específicamente en la capacidad de ajustar el precio de la entrega. Por ello, ajustamos el diseño de las pantallas y añadimos una pantalla adicional para la modificación del precio. La figura 5.2 muestra la modificación solicitada por el cliente, que consiste en añadir la pantalla para la modificación del precio de la entrega.

Después de esta modificación, el cliente determinó que el precio no sería un factor crucial para la aplicación, ya que solo necesitaba funcionalidades relacionadas con el proceso de entrega de paquetes, como la recogida y la entrega. Por lo tanto, eliminamos la consideración del precio de las funcionalidades.

Posteriormente, el cliente nos contactó para informarnos que había replanteado el diseño de la pantalla de inicio *Home* y solicitó una nueva propuesta. Además, expresó la necesidad de una pantalla de detalles que mostrara toda la información relevante de cada paquete. Asimismo, nos pidió que modificáramos los datos requeridos al crear paquetes de recogida y entrega.

En respuesta, rediseñamos la pantalla de inicio dividiéndola en dos secciones: la parte superior para los paquetes pendientes de recogida y la parte inferior para los paquetes pendientes de entrega. También creamos una nueva pantalla de detalles, accesible desde la pantalla de inicio al hacer clic en uno de los paquetes listados. Para distinguir el estado de los paquetes, utilizamos un sistema de colores: azul para los paquetes pendientes, amarillo para los paquetes en tránsito y verde para los paquetes entregados. Además, añadimos un campo de descripción para anotar detalles como (paquete urgente). Finalmente, logramos la versión actual o final de la aplicación,

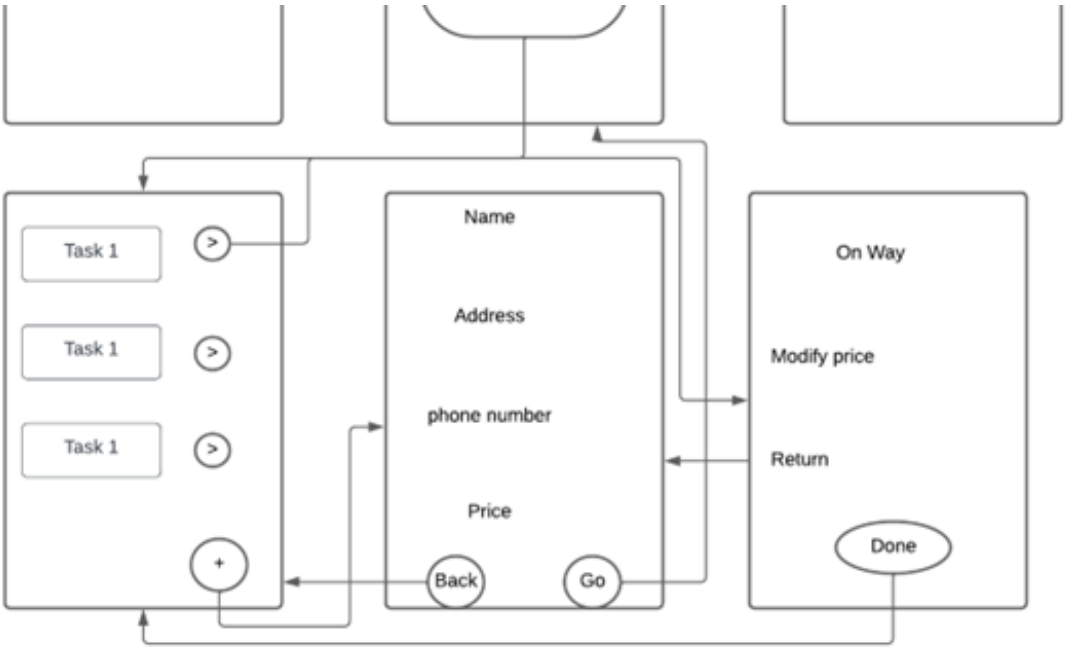

Figura 5.2: Pantalla de modificación.

que incorpora todas estas mejoras y cumple con los nuevos requisitos del cliente. Puede verse en el capítulo 3 en la figura 3.1

5.3. GitLab

Para la gestión de versiones del proyecto hemos usado GitLab. El proyecto está disponible en https://gitlab.com/Nemesis_HZB/vertexdelivery/-/tree/Checkout1.0.[1] Elegimos trabajar con GitLab por varias razones clave que benefician el desarrollo de nuestro proyecto. GitLab ofrece una solución de Integración Continua y Despliegue Continuo (CI/CD) integrada directamente en la plataforma, lo que permite automatizar el proceso de pruebas y despliegue del código. Esto asegura que cada cambio en el código sea verificado y desplegado de manera eficiente, reduciendo el riesgo de errores y mejorando la calidad del software. Aunque podríamos haber elegido GitHub, optamos por GitLab debido a nuestra experiencia previa con la plataforma y su facilidad de uso.

Sin embargo, al inicio del proyecto nos enfrentamos al desafío de abordar múltiples aspectos simultáneamente y con premura, lo que nos llevó a olvidar incluir la configuración de Git dentro de nuestro proyecto. Esto ocasionó demoras en la implementación y configuración de Git. Por lo que el historial de Git comienza posteriormente al inicio del desarrollo. Este fue un error importante, ya que desarrollar código sin tener un control de versiones configurado y asociado al proyecto es arriesgado. Si hubiéramos encontrado alguna incidencia durante este período inicial, podríamos haber perdido parte del código o incluso todo el trabajo realizado hasta ese

[1] En caso de que el lector no tenga cuenta en GitLab he clonado el proyecto en GitHub (https://github.com/Arrab/VERTEX/tree/Checkout1.0).

momento. El primer *push* al repositorio de GitLab, se realizó cuando el proyecto ya estaba aproximadamente al 50 % de su desarrollo.

Capítulo 6

Validación con pruebas de usabilidad

En este capítulo, se llevarán a cabo el diseño y la ejecución de las pruebas de usabilidad sobre la aplicación VERTEX. Durante la fase de desarrollo se había pensado en hacer unas pruebas de usabilidad para garantizar que la aplicación móvil sea fácil de usar y cumpla con las expectativas de los usuarios, sin embargo, se pospusieron estas pruebas por falta de tiempo.

Las pruebas de usabilidad son un proceso importante para garantizar que una aplicación móvil sea fácil de usar y cumpla con las expectativas de los usuarios. Estas pruebas pueden ayudar a identificar problemas de diseño y funcionalidad, así como obtener información valiosa sobre la experiencia del usuario.

Este TFG continúa la estela del desarrollo hecho previamente de esta aplicación, y se van a poner en práctica estas pruebas de usabilidad las cuales se habían pospuesto y que ahora son determinantes, ya que como veremos, estas pruebas van a poner en entredicho algunas de las funcionalidades ya implementadas.

Para ello, hemos decidido basarnos en un artículo de investigación del Instituto Tecnológico Metropolitano de Medellín, Colombia, Andrés Paniagua L (2020), ya que este artículo reúne y usa varias disciplinas relevantes como son las de Nielsen (1994) y también (W3C) (2008) (The World Wide Web Consortium) que desarrolla estándares y directrices para ayudar a todos a construir una web basada en los principios de accesibilidad, internacionalización, privacidad y seguridad.

Este artículo (Andrés Paniagua L 2020) describe una metodología de seis fases para realizar una prueba completa de usabilidad, como se muestra en la figura 6.1. En nuestro caso, utilizaremos cinco fases, omitiendo la accesibilidad. Aunque la accesibilidad es una parte crucial de las pruebas de usabilidad, no la incluiremos en nuestro estudio debido a la necesidad de realizar modificaciones significativas en el diseño de la aplicación. Es importante mencionar que actualmente la empresa no tiene previsto contratar a empleados con necesidades especiales. Sin embargo, esto podría cambiar en el futuro, momento en el cual revisaremos y evaluaremos la accesibilidad.

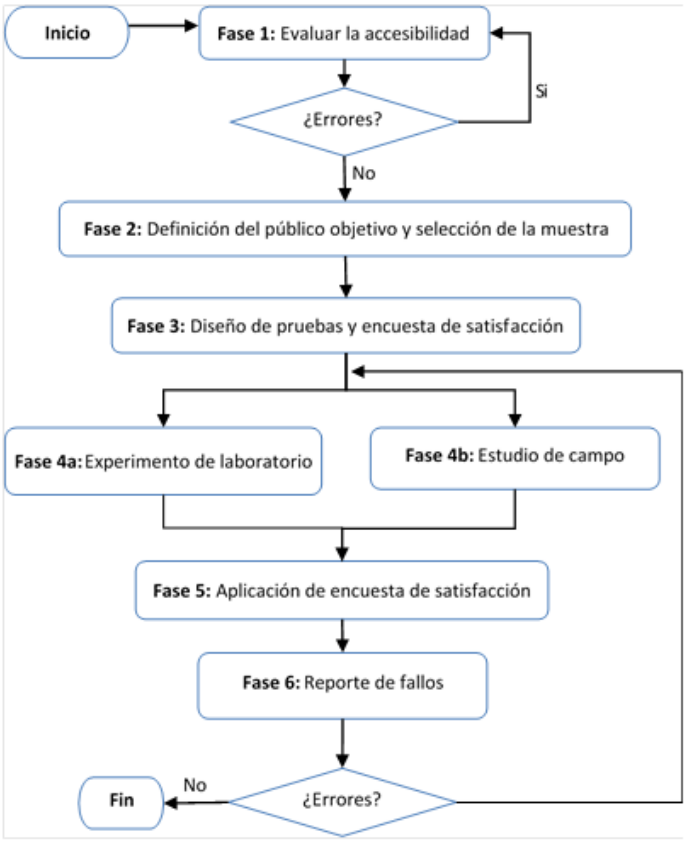

Figura 6.1: Metodología de las prueba de usabilidad.

6.1. Público objetivo

Primero, analizamos el público objetivo de la aplicación, ya sean conductores, administrativos o administradores, y seleccionamos una muestra representativa. También indicamos el tipo de pruebas y la cantidad mínima de personas necesarias. Habrá dos tipos de pruebas: una de laboratorio, utilizando un PC y un emulador de móvil, y una de campo, instalando una APK (ejecutable) en el móvil de los participantes.

La muestra del público objetivo debe representar alrededor del 30 % de los usuarios esperados, como menciona Andrés Paniagua L (2020). Esto se explica en el capítulo 2. Aunque Nielsen (1994) popularizó la idea de que con solo cinco usuarios se pueden identificar alrededor del 85 % de los problemas de usabilidad en un diseño, nuestra experiencia y la investigación realizada nos llevan a intentar alcanzar una muestra con mayor número de participantes.

La siguiente pregunta que se plantea es: ¿Qué conocimientos-experiencia en aplicaciones móviles deberían tener estos usuarios?, sabiendo que los empleados de esta empresa eran jóvenes menores de 25 años, y que tienen un conocimiento básico sobre app móviles, entonces la muestra debía de contener la máxima similitud con el público objetivo, eso quiere decir que debemos tener unas expectativas a la baja con respecto a los conocimientos de los usuarios. Los usuarios son conductores y oficinistas, en el caso de los conductores el perfil más común que la empresa

Categoría	Conocimientos	Participantes
Pruebas de laboratorio	Básicos (2 personas)	Grupo 1: Fran y Abel
	Avanzados (4 personas)	Grupo 2: Hakima, Abdul, Alice y Aron
Pruebas de campo	Básicos (6 personas)	Grupo 3: Marcos, Laso, Mohamed, Miguel, Rubén, Elisa
	Avanzados (2 personas)	Grupo 4: Javier y Juan Manuel

Tabla 6.1: Distribución de participantes en las pruebas de usabilidad

tiene previsto contratar son perfiles de personas jóvenes sin estudios o cursándolos, mientras que los oficinistas tienes que tener un nivel básico de estudios para poder desempeñar las tareas administrativas. Con estas premisas el parámetro para la muestra tiene que ser un perfil con conocimientos básicos en app móviles.

Por lo que se seleccionan dos tipos de usuarios: uno con conocimientos básicos en manejo de app móviles, y otro grupo con experiencia en testeo de app en general. El grupo primero va a ser el que determine la validez de las funcionalidades, mientras que el segundo grupo va a servir de control sobre el primer grupo, para poder comparar realmente la capacidad de análisis de ambos, así como los resultados. La selección se finalizó con 14 participantes, 8 de ellos con conocimientos básicos de app móviles, y las otras 6 con conocimientos informáticos y de testeo de apps.

La tabla 6.1 muestra las pruebas, sus características y los grupos que hemos determinado. Primero explicamos las dos categorías de pruebas:

Pruebas de Laboratorio:

Estas pruebas tienen las siguientes características:

- Control del entorno: En las pruebas de laboratorio, el entorno se controla cuidadosamente, lo que significa que se pueden minimizar las distracciones y variables externas que podrían afectar los resultados. Esto permite una observación más precisa del comportamiento del usuario.
- *Feedback* inmediato: Los evaluadores pueden proporcionar retroalimentación inmediata a los participantes y realizar un seguimiento en tiempo real de sus acciones y respuestas.
- Condiciones controladas: Se pueden crear situaciones específicas y controladas para evaluar aspectos concretos de la usabilidad. Esto es útil en la detección de problemas específicos de diseño.
- Entorno seguro: Los participantes se sienten más cómodos en un entorno de laboratorio, lo que puede fomentar la honestidad en sus respuestas.

Para la realización de estas pruebas hemos determinado una clasificación de los participantes

en función de sus conocimientos y tenemos dos grupos:

- Grupo 1: Fran y Abel.
- Grupo 2: Hakima, Abdul, Alice y Aron.

Pruebas de Campo:

- Contexto del mundo real: Las pruebas de campo se realizan en el entorno natural del usuario, lo que permite evaluar cómo el producto se utiliza en condiciones reales.
- *Feedback* auténtico: Los usuarios están más relajados y pueden proporcionar *feedback* más auténtico porque utilizan el producto en situaciones cotidianas.
- Observación en uso real: Los evaluadores observan el comportamiento de los usuarios en contextos reales, comprendiendo mejor cómo los usuarios se adaptan al producto.
- Detección de problemas reales: Las pruebas de campo a menudo revelan problemas que podrían pasar desapercibidos en un entorno de laboratorio. Esto es especialmente importante para productos que deben funcionar en situaciones variadas.

Para la realización de estas pruebas hemos determinado una clasificación de los participantes en función de sus conocimientos y tenemos dos grupos:

- Grupo 3: Marcos, Laso, Mohamed, Miguel, Rubén y Elisa.
- Grupo 4: Javier y Juan Manuel

Las pruebas de laboratorio son útiles para identificar problemas específicos de diseño y permiten un control riguroso, mientras que las pruebas de campo brindan una visión del uso del producto en situaciones reales. Ambos tipos de pruebas son complementarios y se utilizan en diferentes etapas del proceso de diseño para garantizar que el producto sea fácil de usar y satisfaga las necesidades de los usuarios en una variedad de contextos.

La cantidad de participantes requeridos para cada una de las pruebas varía en función de la regla de los 5 usuarios de Nielsen (2000), que se basa en la idea de que, a medida que se realizan pruebas con un número relativamente pequeño de participantes, los problemas de usabilidad más comunes se hacen evidentes. Esto se debe a que, con tan solo unos pocos usuarios, es probable que se repitan los patrones de comportamiento y se identifiquen problemas que afectan a una parte significativa de la audiencia.

6.2. Guía de usuario

Esta guía consiste en diseñar las pruebas que se han seleccionado en la fase anterior. Primero listar las tareas que puede hacer el usuario en la app. Establecer cuales de las tareas son más complejas o en las que pueden errar. Y por último establecer los tiempos de ejecución para cada una de las tareas.

Para poder seleccionar correctamente y de forma acotada la lista de las tareas, es necesario

Page	Name of Task	Diff.	Import.	Prior.	Complej.
Login	Registrar Usuario	2	2	Baja	Baja
	Forgot password	2	1	Baja	Baja
	Login App	1	4	Medio	Baja
Home	Pick Up: Editar descripción	3	4	Medio	Alta
	Delivery: Entregar Paquete	3	4	Medio	Alta
PickUp	seleccionar	1	2	Baja	Medio
	crear	4	5	Alta	Alta
	cambiar estado	2	4	Alta	Alta
Delivery	seleccionar	1	2	Baja	Medio
	crear	4	5	Alta	Alta
	cambiar estado	2	4	Alta	Alta

Tabla 6.2: Tabla de tareas, DIFFicultad, IMPORTancia, PRIORidad y COMPLEJidad

ponderar todas las tareas que se pueden realizar en la aplicación. Una vez hecha esa ponderación entonces evaluamos las tareas más importantes con las siguientes premisas:

- **Dificultad:** Asignar una puntuación de dificultad a cada tarea. En una escala del 1 al 5, donde 1 representa tareas fáciles y 5 representa tareas difíciles o complejas.
- **Importancia:** Evaluar la importancia de cada tarea para la experiencia del usuario. Una escala similar a la de la dificultad puede ser útil, donde 1 es menos importante y 5 es extremadamente importante.
- **Prioridad:** Asignar una prioridad a cada tarea basada en la urgencia de abordarla. Esto puede ayudar a determinar qué tareas deben abordarse primero. Utilizar una escala de alta, media o baja prioridad puede ser útil.
- **Complejidad técnica:** Evaluar la complejidad técnica asociada con cada tarea, especialmente si involucra cambios en el código, desarrollo de nuevas características o actualizaciones del sistema. Una escala numérica puede ayudar a clasificar la complejidad, donde valores más altos representan mayor complejidad.

La tabla 6.2 muestra la lista de todas las tareas que se puede realizar en la aplicación, y las premisas que acabamos de nombrar. Por ejemplo, la tarea de editar descripción en la pantalla *Home* (tarea 4), se clasifica como: (i) Tiene una dificultad intermedia (columna 3), ya que hay que tener un cierto conocimiento de la herramienta previamente para poder ejercer correctamente la función.(ii) Tiene una importancia alta (columna 4) ya que según avanza el paquete a su destino el transportista tiene que reflejar notas importantes en el flujo. (iii) Tiene una prioridad media (columna 5) ya que puede haber otras funcionalidades que se tengan que abordar antes que esta. (iv) Y tiene una complejidad alta (columna 6) por la implicación de varias estructuras del sistema a la hora de realizar un cambio en el código.

La tabla 6.3 muestra la ponderación de las tareas según su dificultad y los tiempos considerados

Descripción	Tarea	Dificultad	Tiempo (segundos)
Registrar Usuario	T1	2	30
Forgot password	T2	3	60
Login App	T3	1	10
Home Pick Up: Editar descripción	T4	1	20
Home Delivery: Entregar Paquete	T5	1	20
Pick Up: Crear New Task	T6	2	45
Delivery: Crear New Package	T7	2	45

Tabla 6.3: Ponderación de la lista de tareas con dificultad y tiempo estimado

válidos basados en un análisis previo. Cronometramos la duración de cada tarea para determinar estos tiempos. Posteriormente, redondeamos los tiempos y elaboramos la tabla correspondiente.

6.2.1. Ejecución de la guía de usuario

Los usuarios seleccionados van a ejecutar las pruebas, y nosotros vamos a llevar a cabo las anotaciones pertinentes para recoger la máxima información que podamos. Esta información luego la analizaremos y se harán estadísticas para encontrar problemas y posibles soluciones. Antes de nada, hay que diseñar una guía de usuario, con las funcionalidades que hemos escogido probar, para que el usuario vaya paso a paso probando, y facilitar la comprensión de nuestros objetivos, así como aumentar la retención de los usuarios. Elaboramos un documento con instrucciones claras y sencillas para que el usuario pueda realizar todas las pruebas de manera autónoma, sin necesidad de intervención por parte de los analistas. La figura 6.2 muestra el documento que se les entregó a los usuarios.

En primer lugar, hicimos las pruebas de laboratorio. Ambientamos una habitación sin ruidos ni distracciones, con el PC encendido y con el emulador de móvil listo y ejecutándose, los usuarios directamente se sientan y comienzan con la prueba. Mientras el usuario ejecutaba la prueba, se cronometraban y analizaban las dificultades que se encontraban. Les propusimos a los usuarios que interactuasen con nosotros lo menos posible. Estas pruebas la realizarán el Grupo 1 y 2, ver en la sección 6.1.

Lo primero que hay que tener en mente es que el emulador genera una interfaz que simula un móvil, por lo que los usuarios realmente tendrán que hacer las mismas acciones que realizarían con un móvil en la mano, pero con el ratón del PC clicando en la pantalla.

1. Los usuarios iniciaban la prueba abriendo la aplicación y se encontraban directamente con la pantalla de inicio de sesión. Como no disponían de cuentas registradas, debían dirigirse a la sección de *Registrarse* para crear una nueva cuenta. Una vez completado el registro con sus datos personales, eran redirigidos automáticamente a la pantalla de inicio de sesión para iniciar sesión con sus nuevas credenciales.
2. Antes de iniciar sesión, se les solicitaba probar la funcionalidad de *Forgot password*. Al

Es una aplicación de recogida y envío de paquetes. Esta guía va a tratar de enfocar las tareas para un conocimiento básico de la herramienta. Has de intentar realizar todas las tareas. Van puntuadas según dificultad, con lo que se te pedirá al final de la prueba una puntuación de la dificultad y el tiempo de cada una de las tareas. Es importante tomar nota de los aspectos importantes como dificultad y tiempo.
Después de haber instalado la aplicación comenzamos:

1. Abrir App y registrar el usuario.
2. Forgot password
3. Login
   Pantalla Home:
4. Recogida de paquete (pickUp). Acceder a una orden y Editar la descripción.
5. Entrega de paquete (Delivery). Cambiar estado de la orden. (Take it, In progress, Done)
   Pantalla PickUp:
6. Crear nueva Orden. Primero tiene que seleccionar su perfil. Luego Introducir los datos del paquete que debe ser recogido por uno de nuestros conductores. (Nombre, Dirección, número de teléfono, descripción, horario)
   Pantalla Delivery:
7. Crear nueva Orden. Introducir los datos del paquete que debe ser entregado por uno de nuestros repartidores. (Remitente, factura, Nombre del destinatario, dirección, número de teléfono, descripción)

Figura 6.2: Documento de la guía de usuario para las pruebas de usabilidad.

hacer clic en el botón de texto *Forgot password*, eran redirigidos a una página donde debían ingresar su dirección de correo electrónico y pulsar el botón Restablecer contraseña. Luego, recibían un correo electrónico con un enlace para cambiar su contraseña. Al hacer clic en el enlace, accedían a una página donde podían restablecer su contraseña.

3. Una vez logados, la pantalla inicial que se muestra es la de *Home*. En esta pantalla, los usuarios pueden visualizar las diferentes órdenes en curso en tiempo real, en caso de que las haya. Estas órdenes se categorizan mediante colores (azul, amarillo y verde), tal como se ha mencionado previamente. Para comenzar, el usuario deberá acceder a una orden de Pick Up y editar la descripción de la tarea correspondiente.
4. Luego, el siguiente paso es modificar el estado de una orden de Delivery. Para ello, el usuario debe acceder a la orden deseada y, una vez dentro, seleccionar la opción correspondiente pulsando el botón grande ubicado en la parte inferior, ya sea *Take it* o *Done*, según corresponda.
5. Luego, el usuario debe hacer clic en el botón *Pick Up* en el menú ubicado a la izquierda de la pantalla para acceder a esta sección. Una vez dentro, deberá crear una nueva orden de recogida. Lo primero que debe hacer es seleccionar el perfil con el que desea crear la orden (normalmente, su propio perfil). Luego, deberá ingresar los detalles del paquete que será recogido por los conductores y, finalmente, hacer clic en el botón *Create*.
6. La última prueba consiste en realizar el mismo procedimiento que en la prueba anterior, pero esta vez para la sección de Delivery. Debe hacer clic en el botón Delivery en el menú ubicado a la izquierda de la pantalla, y una vez dentro, crear una nueva orden de entrega.

Para ello, debe pulsar el botón (+) y completar los datos del paquete y su destinatario. Finalmente, haz clic en el botón *Create*.

Durante el desarrollo de la prueba, hemos registrado los tiempos empleados en cada tarea y las dificultades observadas por los participantes.

En la prueba de campo, continuamos siguiendo el mismo guion que utilizamos en la prueba de laboratorio, para mantener la coherencia en el proceso de evaluación. Sin embargo, la principal diferencia es que en esta ocasión los participantes utilizan sus propios móviles en lugar de los dispositivos proporcionados por nosotros. Estas pruebas las realizan el Grupo 3 y 4.

Después de completar la prueba, analizamos detalladamente tanto los datos que hemos recopilado como el *feedback* que nos han dado los participantes. Este análisis nos proporciona una visión completa y precisa de la experiencia del usuario, identificando patrones y áreas de mejora, y generando conclusiones significativas para el desarrollo y la optimización de la aplicación.

6.3. Encuesta de satisfacción

Esta encuesta es crucial, ya que en ella vamos a anotar y analizar cada uno de los comentarios y las opiniones de los usuarios que van a realizar la prueba. Es muy importante diseñar las preguntas adecuadas para que podamos tener un análisis coherente.

El objetivo de esta encuesta es evaluar la usabilidad general, la eficiencia y la satisfacción con el uso de la aplicación. Para esto, dividimos la encuesta en secciones como la navegación, la información, la complejidad y la satisfacción general. Combinamos preguntas abiertas y cerradas. Las preguntas cerradas proporcionan datos cuantificables, mientras que las preguntas abiertas ofrecen perspectivas cualitativas. Empleamos una escala de valoración combinada, una numérica (del 1 al 5, donde 1 es "Muy insatisfechoz 5 es "Muy satisfecho") y otra de Sí/No. La numérica describe las preguntas donde se proporcionan datos cualitativos, mientras que las de Sí/No se inclinan hacia las preguntas donde queremos datos cuantitativos.

Y por último dejaremos un bloque de sugerencias para obtener comentarios más detallados, en el que los participantes puedan sugerir mejoras o expresen cualquier problema específico que hayan encontrado.

Como hemos mencionado vamos a dividir la encuesta en 4 bloques:

- Identificación del entorno y de los botones: En este bloque, evaluaremos la capacidad del usuario para identificar con agilidad cada parte de la aplicación. Esto incluye las pantallas en las que se encuentra, las funciones de los elementos y los colores identificativos de ciertos estados. ¿Pudiste identificar con facilidad cada parte de la aplicación, incluyendo las pantallas en las que te encontrabas, las funciones de los elementos y los colores identificativos de los estados? (Si / No)
- Información de la prueba: En esta sección, pediremos evaluar la experiencia de la prueba

propuesta y la utilidad de la información proporcionada por la aplicación. Por favor, califica del 1 al 5 la claridad y utilidad de la información proporcionada para llevar a cabo la prueba. (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

- Dificultad: En este bloque, calificamos del 1 al 5 la complejidad para utilizar la aplicación y sus funcionalidades, como registrarse, crear una nueva tarea y editar la tarea. Por favor, califica del 1 al 5 qué tan difícil encontraste usar la aplicación y sus funciones. (1 - Muy difícil / 5 - Nada difícil).
- Sugerencias: Por último, damos la oportunidad de expresar cualquier sugerencia, idea o comentario sobre la aplicación y la experiencia al usarla. ¡Estamos deseando conocer tus pensamientos!.

Los participantes compartirán su percepción sobre las funcionalidades de la aplicación mediante la encuesta que hemos preparado, así como mediante las sugerencias que consideren que la aplicación debería incluir. Basándonos en las premisas establecidas, hemos diseñado la siguiente encuesta:

- Primer bloque, Identificación del entorno y los botones: Hemos elaborado una serie de preguntas que nos proporcionarán información cuantitativa. Por ejemplo, ¿Sabrías identificar en todo momento en qué pantalla te encuentras? (Si / No). Esta pregunta nos permitirá conocer cuántos usuarios responden afirmativa o negativamente.
- Segundo bloque, denominado Información de la prueba: Nos concentramos en recopilar información cualitativa con el objetivo de obtener una variedad de resultados que nos permitan realizar un análisis más profundo y detallado. En este bloque, al igual que en el siguiente, emplearemos una escala de ponderación del 1 al 5 para evaluar diferentes aspectos de la experiencia de usuario. Esta aproximación analítica nos ayudará a comprender en mayor profundidad las percepciones y opiniones de los participantes respecto a la prueba realizada. Por ejemplo, en la siguiente pregunta: ¿Consideras que los títulos de cada pestaña se relacionan adecuadamente con su cometido?, que será calificada en una escala del 1 al 5, buscamos entender cómo los participantes sintetizaron la información al leer y navegar por las distintas pestañas. Queremos determinar si consideraron que la información mostrada en las distintas navegaciones fue pertinente y adecuada para su cometido.
- Tercer bloque, Dificultad: como mencionamos previamente, vamos a recopilar información cualitativa. En este bloque, buscamos entender qué tan complicada o desafiante fue la experiencia del usuario al interactuar con las diferentes funcionalidades de la aplicación. Por ejemplo, en la siguiente pregunta: ¿Cómo calificarías la dificultad de crear una nueva tarea en la sección *Pick Up*?, estamos interesados en conocer el nivel de dificultad percibido por el usuario al realizar esta tarea, y cómo esto puede haber afectado la cantidad de esfuerzo y energía dedicada a completarla. La tabla 6.4 refleja las preguntas del bloque tres.
- Cuarto bloque, Sugerencias: este bloque desempeñará un papel fundamental en nuestro análisis, ya que proporcionará ideas valiosas sobre las áreas de mejora y las funcionalidades que pueden ser optimizadas en la aplicación. Las sugerencias recopiladas nos brindarán una visión detallada de las necesidades y expectativas de los usuarios, permitiéndonos identificar oportunidades de mejora tanto a nivel técnico como funcional.

Tarea	Cómo calificarías la dificultad de
P1	Navegar con el uso de la app en general.
P2	Registrar un nuevo usuario.
P3	Utilizar la función *Forgot password*.
P4	Iniciar sesión en la aplicación.
P5	Editar la descripción en la sección *Home Pick Up*.
P6	Entregar un paquete en la sección *Home Delivery*.
P7	Crear una nueva tarea en la sección *Pick Up*.
P8	Crear un nuevo paquete en la sección *Delivery*.

Tabla 6.4: Preguntas realizadas para el bloque 3 (Dificultad)

Tarea	Nombre de la Tarea
T1	Registrar Usuario
T2	Recordar contraseña
T3	Inicio de sesión
T4	Home Pick Up: Editar descripción
T5	Home Delivery: Entregar Paquete
T6	Pick Up: Crear Nueva Tarea de Recogida
T7	Delivery: Crear Nuevo Tarea de Entrega

Tabla 6.5: Lista de Tareas

Estas recomendaciones nos servirán como guía para priorizar los cambios y actualizaciones que se implementarán en futuras versiones de la aplicación.

Además, el análisis de las sugerencias nos ayudará a comprender mejor las preferencias de los usuarios y a alinear el desarrollo de la aplicación con sus necesidades reales. Esto nos permitirá ofrecer una experiencia de usuario más satisfactoria y mejorar continuamente la calidad y la relevancia de nuestra aplicación.

6.4. Análisis de los resultados de las pruebas

En esta sección, analizaremos los resultados de las pruebas mediante tablas comparativas. Estas tablas presentan las medias de todos los resultados obtenidos por los participantes, proporcionando una visión clara y estructurada de los datos recolectados.

Tarea	Tiempo estimado (segundos)	Media de tiempos (segundos)
T1	30	45
T2	60	67
T3	10	22
T4	20	21
T5	20	22
T6	45	60
T7	45	46

Tabla 6.6: Comparación de media de tiempos y estimación de tiempos para cada tarea

6.4.1. Descripción de los resultados

La tabla 6.5 enumera las tareas que han ejecutado los usuario a la hora de realizar las pruebas de usabilidad. Vamos a analizar los datos de tres modos, (i) análisis comparando los tiempos estimados y los tiempos de los participantes (ii) análisis desde el punto de vista del tipo de prueba (prueba de laboratorio o prueba de campo), (iii) análisis desde el punto de vista de los conocimientos, (iv) análisis de las preguntas de la encuesta de satisfacción y (v) análisis de la encuesta de satisfacción.

- **Análisis comparando los tiempos estimados y los tiempos de los participantes:**

  La tabla 6.6 muestra la comparativa de los tiempos con nuestra ponderación previa, observamos algunas desviaciones significativas en ciertas tareas. Por ejemplo, en la tarea 1, que implica registrar un usuario, encontramos una diferencia de 15 segundos. Esta discrepancia puede deberse a una subestimación de la complejidad de la tarea o a posibles problemas de usabilidad.

  En cuanto a la tarea 2, relacionada con recordar la contraseña, aunque notamos que los usuarios se mostraban más incómodos, nos sorprende la pequeña desviación con respecto a la estimación. Esto sugiere que posiblemente hayamos acertado en nuestra predicción.

  La tarea 3, que consiste en iniciar sesión, también muestra una desviación significativa del tiempo estimado. A pesar de ser un proceso aparentemente sencillo de ingresar correo, contraseña y hacer clic en *Sign In*, muchos participantes se confundieron e introdujeron su nombre en lugar del correo, ya que esta premisa no estaba claramente indicada en ningún lugar.

  Por último, en la tarea 6, que implica crear una nueva tarea de recogida, observamos una diferencia considerable con respecto a la estimación. Esta tarea implica completar un formulario con los datos correctos, lo que sugiere que la dificultad de esta tarea podría ser mayor de lo que inicialmente estimamos.

  Habría que señalar, que la tarea 7 fue significativamente más rápida debido a que es un proceso bastante similar al de la tarea 6, por lo que todos los participantes les resultaba más familiar hacer la prueba en esta tarea, y aunque dudaban en algún dato del formulario, en general avanzaron debidamente.

Tarea	Tiempos P.Laboratorio (segundos)	Tiempos P.Campo (segundos)
T1	45.1	46.5
T2	66.3	68.2
T3	19.3	24.6
T4	22.3	19.5
T5	22.5	22.3
T6	55.3	64.6
T7	45.6	47.2

Tabla 6.7: Comparación de la media de los tiempos de ejecución entre pruebas de laboratorio y pruebas de campo

- Análisis desde el punto de vista del tipo de prueba:

  La tabla 6.7 muestra la comparación de la media de tiempos en segundos entre las pruebas de laboratorio y las pruebas de campo, reflejados y anotados de cada uno de los participantes en cada una de las tareas.

  En general, los tiempos promedio para completar las tareas en las pruebas de laboratorio fueron más bajos que en las pruebas de campo. Esto sugiere que los participantes enfrentaron más dificultades o distracciones en entornos menos controlados como los de campo.

  En la tarea 2, también existe una diferencia y puede explicarse por la necesidad de acceder al correo electrónico y visualizar el mensaje que aparece en pantalla después de hacer clic en el botón reset password. Los usuarios en el entorno de campo no han detectado este mensaje debido a las limitaciones de espacio en la pantalla del móvil.

  Los resultados de la comparación en la tarea 3, presentan una diferencia de 5 segundos. Esto se debe a la falta de indicaciones en la etiqueta de usuario, que no especifica que se debe introducir el correo electrónico en lugar del nombre. Esta observación resalta la necesidad de proporcionar esta información, para las pruebas de campo.

  Los resultados de la tarea 6, muestran que los participantes en las pruebas de laboratorio fueron más rápidos para abordarla, mientras que en las pruebas de campo se observó que los usuarios tendían a distraerse con más facilidad al enfrentarse a una tarea de mayor complejidad.

  Algunas tareas como tarea 4 y tarea 5, mostraron tiempos de ejecución relativamente consistentes entre pruebas de laboratorio y campo, lo que podría sugerir que estas funciones son menos afectadas por el entorno de uso.

- Análisis desde el punto de vista de los conocimientos:

  La tabla 6.8 muestra la comparación de la media de tiempos entre participantes con conocimientos avanzados en software y conocimientos básicos en software.

  Es importante destacar notables diferencias de tiempo entre los participantes CA[1] CB[2] en

[1]CA: con conocimientos avanzados en probar software
[2]CB: conocimientos básicos en probar software

Tarea	Tiempos Conc.Avanzados (segundos)	Tiempo Conoc.Básicos (segunods)
T1	44.83	47.17
T2	42.83	54.17
T3	21.17	22.33
T4	17.33	19.5
T5	13.33	18
T6	67.67	68.17
T7	41	46.5

Tabla 6.8: Comparación de la media de tiempos entre participantes con conocimientos avanzados en software y conocimientos básicos en software

Respuesta	NO	SI
Votos	11	3

Tabla 6.9: Recuento de respuestas a la pregunta: ¿Sabrías identificar en todo momento en qué pantalla te encuentras? (Si / No).

la tarea 2 y tarea 6. Esta disparidad es especialmente evidente en la tarea 6. La dificultad radica en la introducción de datos necesarios para completar el formulario. Observamos que los CA, cuando se enfrentaban a obstáculos al introducir algún dato, ya sea por no entender su significado o por no saber dónde ubicarlo en esta tarea, optaban por ingresar un dato aleatorio para continuar con la prueba y evitar quedarse bloqueados durante mucho tiempo. En contraste, los CB dedicaban considerablemente más tiempo a interpretar y buscar un dato adecuado, tanto que a veces se rendían y buscaban la ayuda del supervisor, a pesar de nuestras indicaciones de evitar interactuar con nosotros en la medida de lo posible.

En la tarea 2 la mayoría de los participantes pasaron por alto el mensaje de *Check your mail Box*, los CA, cuando se dieron cuenta que al volver a la página de *login* sin que hubiese ningún cambio, fueron a chequear su correo. En cambio, los CB, solo hubo un participante que lo chequeó del grupo 1 (Abel).

Para asegurar que nuestras pruebas de usabilidad reflejen de manera más precisa el uso real de la aplicación, debemos darle mayor consideración a los CB, ya que son el público objetivo de nuestra aplicación. Como hemos mencionado anteriormente, los CA se utilizan como grupo de control en nuestro estudio. Y por tanto debemos tener muy en cuenta este impacto en la usabilidad de las tarea 2 y tarea 6.

- Análisis de las preguntas de la encuesta de satisfacción:
  - **Bloque 1:** Identificación del entorno y de los botones (Si/No).

    La tabla 6.9 identifica las respuestas de la encuesta de satisfacción a la pregunta ¿Sabrías identificar en todo momento en qué pantalla te encuentras? (Si / No). Parece que muchos usuarios tienen dificultades para identificar la pantalla en la que se encuentran en todo momento. Esto resalta la necesidad de una mayor claridad en los títulos de las pantallas de la aplicación.

Respuesta	NO	SI
Votos	10	4

Tabla 6.10: Recuento de respuestas a la pregunta: ¿Pudiste identificar la función de cada uno de los elementos de la aplicación? (Si / No).

Respuesta	NO	SI
Votos	3	11

Tabla 6.11: Recuento de respuestas a la pregunta: ¿Lograste identificar los colores de cada estado de los paquetes? (Si / No).

La tabla 6.10 muestra las respuestas a la pregunta ¿Pudiste identificar la función de cada uno de los elementos de la aplicación? (Si / No). Los participantes expresaron dificultades para identificar los elementos y funciones de la aplicación, lo que sugiere la conveniencia de incluir descripciones detalladas de los objetos y elementos.

Por último, la tabla 6.11 muestra las respuestas a la pregunta ¿Lograste identificar los colores de cada estado de los paquetes? (Si / No). La mayoría de los usuarios pudo identificar correctamente los colores que representan los distintos estados, lo cual constituye un punto a favor del desarrollo actual.

- **Bloque 2:** Información de la prueba. (1-5) donde 1 es nada de acuerdo y 5 totalmente de acuerdo.

  Según la tabla 6.12, ¿te resultó claro desempeñar las tareas propuestas?, se observa que a muchos participantes no les resultó satisfactoria la experiencia de las pruebas. La calificación cercana a 2, en lugar de 3, sugiere que los usuarios no quedaron satisfechos durante la evaluación.

  En la tabla 6.13, ¿encontraste los aspectos visuales de la aplicación fáciles de entender?, los usuarios continúan mostrando insatisfacción con el entorno visual de la aplicación. La mayoría ha calificado por debajo de 3, lo que sugiere que es necesario mejorar las visualizaciones de las pantallas.

  En la tabla 6.14, ¿consideras que los títulos de cada pestaña se relacionan adecuadamente con su cometido?, la pregunta tenía como objetivo determinar si los títulos de las pantallas y pestañas cumplían su función correctamente. También buscaba evaluar la sinceridad de los participantes, al incluir una pregunta similar realizada previamente. Aquellos usuarios que previamente habían respondido negativamente a la pregunta sobre la identificación de la función de los elementos de la aplicación, la mayoría calificaron esta pregunta con un 3 o menos, a excepción de dos personas. Los resultados de esta última pregunta, muestran un ligero aumento en

Ponderaciones	**1**	**2**	**3**	**4**	**5**
Votos por cada ponderación	0	8	6	0	0

Tabla 6.12: Recuento de respuestas a la pregunta: ¿Te resultó claro desempeñar las tareas propuestas? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

Ponderaciones	**1**	**2**	**3**	**4**	**5**
Votos por cada ponderación	1	7	6	0	0

Tabla 6.13: Recuento de respuestas a la pregunta: ¿Encontraste los aspectos visuales de la aplicación fáciles de entender? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

Ponderaciones	**1**	**2**	**3**	**4**	**5**
Votos por cada ponderación	0	1	7	6	0

Tabla 6.14: Recuento de respuestas a la pregunta: ¿Consideras que los títulos de cada pestaña se relacionan adecuadamente con su cometido? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

la calificación promedio con respecto a las preguntas anteriores, lo que sugiere que los títulos de las pantallas y pestañas podrían estar en buen camino. Sin embargo, esto no implica que se descarten posibles mejoras futuras.

- **Bloque 3:** Dificultad. (1-5) donde 1 es muy difícil y 5 nada difícil.

  En la tabla 6.15 en la que se muestran la media de las ponderaciones que los usuarios han asignado a las preguntas de la tabla 6.4, y tiene una calificación donde (1-Muy difícil y 5-Nada difícil). Por lo tanto, las puntuaciones que se encuentren por debajo del (2.5/5) hay que darles una importancia alta.

  Estas evaluaciones proporcionan una base sólida para identificar áreas de fortaleza y debilidad en la interfaz y la experiencia del usuario.

  - Pregunta 1: Navegar con el uso de la aplicación en general. Los participantes en promedio encontraron esta tarea moderadamente difícil (2.8/5). Esta puntuación está en el límite de lo que podríamos considerar una dificultad intermedia.
  - Pregunta 2: Registrar un nuevo usuario. En general, los usuarios percibieron esta tarea como manejable, con algunos participantes indicando una dificultad moderada (3.1/5).
  - Pregunta 3: Utilizar la función (Recordar contraseña). Esta tarea mostró dife-

Pregunta	**Puntuación Media de todos los participantes**
P1	2.86
P2	3.14
P3	2
P4	3.28
P5	2.78
P6	3.21
P7	2
P8	3.07

Tabla 6.15: Puntuación media de todos los participantes a las preguntas

rencias significativas entre los participantes. Al tener una puntuación de (2/5) eso quiere decir que los participantes la encontraron difícil de manejar y por lo tanto esto es un proceso a mejorar.

- Pregunta 4: Iniciar sesión en la aplicación. En general, esta tarea fue percibida como accesible (3.2/5).
- Pregunta 5: Editar la descripción en la sección (Home Pick Up). Esta tarea fue en promedio manejada con relativa dificultad, con la mayoría de los participantes reportando una dificultad moderada (2.7/5).
- Pregunta 6: Entregar un paquete en la sección (Home Delivery). La percepción en general los participantes la encontraron manejable (3.2/5).
- Pregunta 7: Crear una nueva tarea en la sección (Pick Up). Esta tarea fue en promedio considerada difícil, con una puntuación de (2/5), por lo que este proceso debe mejorarse.
- Pregunta 8: Crear un nuevo paquete en la sección (Delivery). En promedio, los participantes evaluaron esta tarea con poca dificultad (3/5).

Hay que destacar que la puntuación media general de todas las preguntas es de un (2.79/5) lo que denota una aplicación moderadamente difícil de manejar, y por tanto la satisfacción del uso de la aplicación disminuye proporcionalmente al aumento de la dificultad.

- **Análisis de las sugerencias de la encuesta de satisfacción:**

Los participantes ofrecieron los siguientes comentarios:

- En el box de user que ponga mail.
- Poder editar los datos del paquete.
- Indicar si hay error en la longitud de la pass.
- Que se pueda acceder a PickUp o Delivery directamente desde la pantalla Home y no darle al menú.
- Poder acceder al perfil.
- Poder editar o borrar las órdenes creadas.
- En delivery que los paquetes se llamen de otra forma, no con el número de factura.
- Mejorar la guía de usuarios para describir mejor las tareas, por ejemplo, la parte de crear nuevas órdenes.
- Control sobre los campos que se introducen, he podido seleccionar que el tiempo de entrega fuera 5:80.
- Los nombres de los deliveries se definen por el número de *invoice*, sería mejor tener un campo para indicar el nombre y evitar repetidos.
- En la creación de un delivery no tenía claro cómo los campos que rellenaba se reflejarían en la aplicación.
- Un botón que contenga la información de para qué sirve cada cosa podría ser muy ilustrativo para identificar los distintos elementos de la app.

Esta valiosa información proporciona una visión clara de los aspectos que más impactaron a los usuarios. Estos comentarios serán extremadamente útiles para guiar las mejoras

futuras de la aplicación.

Al analizar detenidamente cada comentario, se identifican áreas específicas que requieren ajustes o mejoras. Por ejemplo, el comentario sobre (En el box de user que ponga mail) resalta la dificultad que experimentaron los usuarios al ingresar su correo electrónico en el Login. Implementaremos cambios para facilitar este proceso.

Además, el comentario sobre (Los nombres de los deliveries se definen por el número de invoice) plantea una consideración interesante. Aunque algunos usuarios prefieren esta práctica, podríamos sugerir una solución alternativa para evitar confusiones al asignar nombres distintos a los paquetes.

Por último, el comentario sobre (Control sobre los campos que se introducen) subraya la necesidad de mejorar la funcionalidad de introducción de la hora, asegurando que se imponga una restricción válida para la hora máxima. Este ajuste será prioritario para garantizar la precisión de los datos ingresados.

Estos puntos destacados guiarán nuestras acciones para mejorar la aplicación, asegurando una experiencia más fluida y satisfactoria para todos los usuarios.

6.4.2. Identificación de Fortalezas

Una de las pruebas que se destacó por su buen desempeño fue la (tarea 4: editar descripción). En esta prueba, teníamos una estimación de 20 segundos para completarla satisfactoriamente, y nos sorprendió descubrir que los 4 grupos tuvieron un promedio de 18.4 segundos. En comparación, la (tarea 5: entregar paquete), que era similar y también estimábamos en 20 segundos, tuvo un promedio de 21.75 segundos para los 4 grupos. Esto sugiere que la función de editar descripción, es visualmente más eficiente que el botón de cambiar estados.

6.4.3. Identificación de Debilidades

Debilidades de la Aplicación

- (Tarea 2: Recordar Contraseña)

  Problema Identificado: La mayoría de los participantes pasó por alto el mensaje *Check your mail Box*.

  Dificultad Reportada: Los participantes CB[3] tardaron más en completar esta tarea y sólo uno verificó su correo. En contraste, los participantes CA[4] lograron identificar la necesidad de revisar el correo electrónico.

  Impacto: Esta confusión sugiere que la indicación para revisar el correo electrónico no es suficientemente clara o destacada.
- (Crear Nueva Tarea de Recogida)

[3]CB: conocimientos básicos en probar software
[4]CA: con conocimientos avanzados en probar software

Problema Identificado: La tarea implica completar un formulario con datos adecuados, lo cual resultó ser significativamente más difícil para los participantes CB.

Dificultad Reportada: Los CA tendieron a ingresar datos aleatorios para evitar bloqueos, mientras que los CB dedicaron mucho tiempo a interpretar y buscar datos adecuados, llegando a pedir ayuda al supervisor.

Impacto: La introducción de datos en esta tarea es confusa y difícil de entender para los usuarios con menos experiencia, lo que sugiere problemas de usabilidad en el formulario.

Sugerencias de los Usuarios

Indicar y controlar claramente los campos:

- En el campo de usuario, poner *correo electrónico*.
- Controlar los campos introducidos, por ejemplo, la hora de entrega.

Accesibilidad y navegación:

- Acceder directamente a *PickUp* o *Delivery* desde la pantalla de inicio.
- Incluir un botón con información sobre la función de cada elemento.

Funciones adicionales:

- Permitir editar los datos del paquete y las órdenes creadas.
- Acceder al perfil.
- Indicar errores en la longitud de la contraseña.

Claridad en los nombres:

- Renombrar los paquetes en *Delivery* para evitar confusiones con el número de factura.

Capítulo 7

Conclusiones y Mejoras futuras

Concluimos nuestro análisis evaluando el desarrollo de la aplicación y las pruebas de usabilidad realizadas. Recomendamos mejoras, priorizando estos cambios según su impacto en la experiencia del usuario, con el objetivo de proporcionar una base sólida para mejoras futuras.

7.1. Conclusión

El análisis de las pruebas de usabilidad destaca la importancia de mantener un enfoque centrado en el usuario en el futuro desarrollo de la aplicación. Abordar los problemas identificados mejorará la usabilidad y la satisfacción del usuario, lo que conducirá a una experiencia más positiva en general.

Basándonos en los datos recopilados, hemos identificado varios cambios y mejoras que deben implementarse en la aplicación.

En primer lugar, en el proceso de registro de usuarios, es crucial proporcionar un mensaje claro en caso de que el registro se realice con éxito o falle debido a datos incorrectos.

Para la función de *recordar contraseña*, es necesario mejorar la claridad del mensaje que indica dónde y cómo restablecer la contraseña para los usuarios que la hayan olvidado.

En cuanto al proceso de inicio de sesión, proponemos cambiar la etiqueta *User* por *Mail* para evitar confusiones entre los usuarios al momento de ingresar sus credenciales. Además, sería beneficioso permitir a los usuarios eliminar o editar las tareas creadas tanto en la función de *PickUp* como en la de *delivery*.

Implementar un control sobre la información ingresada en los campos de creación de tareas, con el fin de evitar la introducción de valores fuera de rango, como podría ser el caso de la hora. También se recomienda agregar información contextual en los campos de entrada para proporcionar pistas claras sobre su función y propósito.

Estas mejoras contribuirán a una experiencia de usuario más fluida y satisfactoria en la aplicación. Por último, consideramos necesario incorporar una nueva funcionalidad que permita a los usuarios cerrar sesión o salir de la aplicación de manera directa, sin depender de los botones de la interfaz del sistema del dispositivo móvil.

7.2. Priorización de Cambios

Priorizaremos los cambios en función de su impacto en las funcionalidades cruciales y la experiencia del usuario. Comenzaremos abordando las funcionalidades que consideramos bloqueantes, como la limitación del rango de la hora en la creación de tareas para evitar que los usuarios ingresen horas fuera del límite establecido. Luego, nos centraremos en el proceso de inicio de sesión, incluido el registro, la recuperación de contraseña y el inicio de sesión, para garantizar una experiencia inicial positiva para el usuario. Posteriormente, nos ocuparemos de mejorar la capacidad de editar o eliminar tareas, lo que brindará a los usuarios un mayor control sobre la aplicación. Luego, implementaremos una nueva funcionalidad que proporcione información contextual en los campos de entrada, mejorando la claridad y la usabilidad de la aplicación. Finalmente, añadiremos la función de cerrar sesión o salir de la aplicación para ofrecer a los usuarios una manera fácil y clara de finalizar su sesión cuando lo deseen.

7.3. Recomendaciones y mejoras futuras

- Consistencia en la interfaz: Mantener una interfaz intuitiva y consistente puede ayudar a reducir la percepción de dificultad en tareas similares como registrar usuarios o iniciar sesión.
- Claridad en las instrucciones: Mejorar la claridad de las instrucciones y los mensajes de retroalimentación, especialmente en funciones críticas como la recuperación de contraseña, puede ayudar a reducir errores y aumentar la satisfacción del usuario.
- Optimización por secciones: Considerar ajustes específicos en la interfaz de las secciones donde se observaron mayores dificultades, como mejorar la navegación en la sección *Home PickUp* o la claridad en la creación de tareas.

7.3.1. Objetivos cumplidos

En este Trabajo de Fin de Grado (TFG), se han logrado cumplir la mayoría de los objetivos establecidos, demostrando un avance significativo con las pruebas de usabilidad.

En cuanto al desarrollo de la aplicación, se ha implementado el proceso llevado a cabo para el desarrollo de las funcionalidades, enfatizando el cumplimiento de los requisitos funcionales y de diseño acordados. Aunque algunas funcionalidades adicionales aún deben ser desarrolladas

para optimizar completamente la aplicación y asegurar su adecuación para su uso final, los avances hasta la fecha muestran un sólido progreso hacia este objetivo.

Por otro lado, las pruebas de usabilidad han sido completadas con éxito. Se diseñaron y ejecutaron pruebas con una muestra representativa de usuarios reales, permitiendo la identificación y análisis de diversos aspectos que afectan la experiencia del usuario. Los datos cualitativos y cuantitativos recopilados han proporcionado una base sólida para mejorar la usabilidad de la aplicación, destacando áreas específicas que requieren ajustes y mejoras.

En resumen, los resultados obtenidos de las pruebas de usabilidad guiarán las decisiones futuras de diseño y desarrollo, asegurando que la aplicación cumpla con las expectativas de los usuarios y sea eficaz en su uso final.

7.3.2. Como se hacen las mejoras

Una vez identificadas las debilidades de la aplicación, tanto en las funcionalidades inadecuadamente desarrolladas como en aquellas con problemas de usabilidad, es necesario abordar una serie de mejoras prioritarias. Estas mejoras se llevarán a cabo siguiendo la secuencia de prioridades establecida previamente.

Nuestra primera tarea será resolver la funcionalidad de introducción de la hora, ya que representa una incidencia bloqueante. Para lograrlo, identificaremos las clases y funciones que utilizan la hora y determinaremos cuál de ellas es responsable de recibir la entrada de hora. En la clase *MenuPickUpFragment* encontramos una variable de tipo String destinada a la introducción de la hora. Es necesario modificar tanto la declaración de esta variable como el proceso de análisis del valor introducido, de manera que solo se permitan valores horarios válidos, implementando un método o función que ofrezca esta solución. [1]

Agregar una función que verifique si la hora ingresada está dentro del rango válido antes de asignarla al modelo. Utilizando una función auxiliar isValidTime que devuelve true si la hora es válida y false si no lo es.

isValidTime(time: String): utiliza una expresión regular para verificar si la cadena time coincide con el formato de hora HH:mm válido (de 00:00 a 23:59). Antes de asignar txtTime_create.text.toString() a statusModel!!.time, se verifica si es una hora válida utilizando esta función. Si la hora no es válida, se muestra un mensaje de error con un Toast.

Y se reemplaza txtTime_create!!.text.toString() con horaIngresada donde se asigna la hora al statusModel.

```
1 val usuarioDriver = elegirUsuario(spinner_users)
2 if (!TextUtils.isEmpty(txtName_create.text.toString()) &&
3     !TextUtils.isEmpty(txtTime_create.text.toString()) &&
4     !TextUtils.isEmpty(txtAddress_create.text.toString()) &&
5     !TextUtils.isEmpty(txtPhone_create.text.toString()) &&
```

[1] Ver el código original de la clase MenuPickUpFragment en el apéndice A.5.

```
    !TextUtils.isEmpty(txtDescriptio_create.text.toString())) {
    val horaIngresada = txtTime_create.text.toString()
    if (isValidTime(horaIngresada)) {
        val statusModel: PickUpModel? = PickUpModel()
        statusModel!!.dateOperation = formatter.format(currentTime).toString()
        statusModel!!.date = formatter2.format(currentTime).toString()
        statusModel!!.status = Common.STATUS_FREE
        statusModel!!.user = usuarioDriver
        statusModel!!.address = txtAddress_create!!.text.toString()
        statusModel!!.description = txtDescriptio_create!!.text.toString()
        statusModel!!.image = Common.IMG_FREE
        statusModel!!.name = txtName_create!!.text.toString().toUpperCase()
        statusModel!!.phone = txtPhone_create!!.text.toString()
        statusModel!!.time = horaIngresada
        Common.pickupSelected = statusModel
        menuViewModel.setCreateModel(statusModel!!)
        menuViewModel.getMutableCreateLiveData().observe(viewLifecycleOwner, Observer {
            submitToFirebase(it)
        })
        menuViewModel.setCategoryList(statusModel)
        menuViewModel.getCategoryList().observe(viewLifecycleOwner, Observer {
            displayInfo(it)
        })
    } else {
        Toast.makeText(context, "Ingrese una hora válida (00:00 - 23:59)",
        Toast.LENGTH_LONG).show()
    }
} else {
    Toast.makeText(context, "¡NO se pudo crear! Asegúrese de ingresar todos los
    datos.", Toast.LENGTH_LONG).show()
}
private fun isValidTime(time: String): Boolean {
    val regex = """([01]?[0-9]|2[0-3]):[0-5][0-9]""".toRegex()
    return time.matches(regex)
}
```

Luego vamos a retomar el proceso de inicio de sesión, para garantizar una experiencia inicial positiva para el usuario.

Por ejemplo en la funcionalidad de *Forgot password*, el problema se puede solucionar directamente en *ForgotPasswordActivity*, en el cual es donde mostramos el mensaje de *Check your mail box* de esta forma:

```
task ->
if(task.isSuccessful){
    progressBar.visibility = View.VISIBLE
    Toast.makeText(this,"Check your mail box", Toast.LENGTH_LONG).show()
    startActivity(Intent(this, LoginActivity::class.java))
}
```

Para hacer que el mensaje del *Toast* se vea más grande y permanezca visible durante más tiempo sin crear un diseño personalizado, se pueden ajustar dos aspectos del *Toast* directamente en el código: Se puede cambiar el tamaño del texto del *Toast* mediante el método Toast.makeText() y el método setTextSize().

Y la duración del *Toast* se puedes ajustar cambiando el segundo parámetro del método makeText() a Toast.LENGTH_LONG $*$ 2 por ejemplo.

```
task ->
if(task.isSuccessful){
  progressBar.visibility = View.VISIBLE
  val toast = Toast.makeText(this, "Check your mail box", Toast.LENGTH_LONG)
  val toastTextView = toast.view.findViewById<TextView>(android.R.id.message)
  // Tamaño de texto ajustado a 20sp
  toastTextView.setTextSize(TypedValue.COMPLEX_UNIT_SP, 20.toFloat())
  // Duración del Toast extendida
  toast.duration = Toast.LENGTH_LONG * 2
  toast.show()
  startActivity(Intent(this, LoginActivity::class.java))
}
```

Bibliografía

Alarcón, H. F, Hurtado, A. M., Pardo, C., Collazos, C. y Pino, F. J. (dic. de 2007). **Integración de Técnicas de Usabilidad y Accesibilidad en el Proceso de Desarrollo de Software de las MiPyMEs**. En: *Rev. Av. en Sist. e Informática* 4(3), págs. 159-166.

Alarcón-Aldana, A. C., Díaz, E. L. y Callejas-Cuervo, M. (feb. de 2014). **Guía para la evaluación de la Usabilidad en los Entornos Virtuales de Aprendizaje (EVA)**. En: *Inf. tecnológica* 25(3), págs. 135-144.

Andrés Paniagua L Diana Bedoya R, C. M. (2020). **Una método para la evaluación de la accesibilidad y la usabilidad en aplicaciones móviles**. En: *Instituto Tecnológico Metropolitano*, págs. 225-265. URL: https://doi.org/10.22430/22565337.1553.

Canva (2024). **Amazingly Simple Graphic Design Software**. https://www.canva.com/. Accessed: 2024-06-02.

Erickson, W., Trerise, S., Lee, C., VanLooy, S., Knowlton, S. y Bruyère, S. (sep. de 2013). **The Accessibility and Usability of College Websites: Is your Website Presenting Barriers to Potential Students?** En: *Community Coll. J. Res. Pract.* 37(11), págs. 864-876.

Fernández, A., Insfran, E. y Abrahão, S. (2010). **Evaluación de Usabilidad para Aplicaciones Web**. En.

Nielsen, J. (1994). **Usability engineering**. Morgan Kaufmann.

Nielsen, J. (2000). **Why you only need to test with 5 users**.

Ronny, A. M. C. (2016). **Optimización de rutas para la gestión de pedidos y entregas aplicado a las empresas distribuidoras de productos**. En: *Quevedo: UTEQ*, págs. 225-265. URL: https://repositorio.uteq.edu.ec/handle/43000/1875.

(W3C), W. W. W. C. (2008). **Mobile Web Best Practices 1.0: Basic Guidelines**. Inf. téc. World Wide Web Consortium (W3C). URL: https://travesia.mcu.es/server/api/core/bitstreams/9d71c798-980b-40f7-9e36-2af78fbf0b36/content.

Zhang, D. y Adipat, B. (nov. de 2005). **Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications**. En: *Int. J. Hum. Comput. Interact.* 18(3), págs. 293-308.

Apéndice A

Apéndice

A.1. Código DeliveryModel

```
class DeliveryModel {
    var package_id:String?= null
    var invoiceNumber: String?= null
    var shipper: String? = null
    var phone: String? = null
    var address: String? = null
    var endClientName: String? = null
    var status: String? = null
    var image: String? = null
    var description: String? = null
    var date: String? = null
    var dateOperation: String? = null
    constructor()
    constructor(
        package_id: String?,
        invoiceNummber: String?,
        shipper: String?,
        phone: String?,
        address: String?,
        endClientName: String?,
        status: String?,
        image: String?,
        description: String?,
        date: String?,
        dateOperation: String?
    ){
        this.package_id = package_id
        this.invoiceNumber = invoiceNummber
        this.shipper = shipper
        this.phone = phone
```

```
        this.address = address
        this.endClientName = endClientName
        this.status = status
        this.image = image
        this.description = description
        this.date = date
        this.dateOperation = dateOperation
    }
```

A.2. Código DeliveryDetailsViewModel

```
        class DeliveryDetailsViewModel : ViewModel() {

        private var mutableLiveData: MutableLiveData<DeliveryModel>? = null
        private var mutableStatusLiveData: MutableLiveData<DeliveryModel>? = null

        init {
            mutableStatusLiveData = MutableLiveData()
        }
//llamadas para interactuar con la view y sus datos
        fun getMutableLiveData(): MutableLiveData<DeliveryModel> {
            if (mutableLiveData == null)
                mutableLiveData = MutableLiveData()
            mutableLiveData!!.value = Common.deliverySelected
            return mutableLiveData!!
        }
//llamadas para interactuar con la view y el estado
        fun getMutableStatusLiveData(): MutableLiveData<DeliveryModel> {
            if (mutableStatusLiveData == null)
                mutableStatusLiveData = MutableLiveData()
            mutableStatusLiveData!!.value = Common.deliverySelected
            return mutableStatusLiveData!!
        }
//establecer el estado en la view
        fun setStatusModel(statusModel: DeliveryModel) {
            if (mutableStatusLiveData != null)
                mutableStatusLiveData!!.value = (statusModel)
        }

}
```

A.3. Código DeliveryAdapter

```

class DeliveryAdapter (internal var context: Context, internal var deliveryCategoryModels:
RecyclerView.Adapter<DeliveryAdapter.MyViewHolder>(){
// Clase ViewHolder interna que mantiene y maneja los elementos de la vista
    inner class MyViewHolder(itemView:View):RecyclerView.ViewHolder(itemView),
    View.OnClickListener{
// Declaración de vistas del layout
        var delivery_invoice_nummber: TextView?= null
        var delivery_image: CircleImageView?= null
// Listener para manejar los clics en los elementos del RecyclerView
        internal var listener:IRecycleItemClickListener? = null
// Método para establecer el listener
        fun setListener(listener:IRecycleItemClickListener){
            this.listener = listener
        }
// Inicialización de vistas y configuración del click listener
        init {
            delivery_invoice_nummber = itemView.findViewById(R.id.txt_category_delivery_invoi
            delivery_image = itemView.findViewById(R.id.img_category_delivery_image) as Circl
            itemView.setOnClickListener(this)
        }
// Maneja el evento de clic en el elemento
        override fun onClick(v: View?) {
            listener!!.onItemClick(v!!,adapterPosition)
        }
    }
// Alimenta el layout de los elementos del RecyclerView y crea un ViewHolder
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
        return MyViewHolder(LayoutInflater.from(context).inflate(R.layout.layout_delivery_ca
    }
// Devuelve la cantidad de elementos en la lista
    override fun getItemCount(): Int {
        return deliveryCategoryModels.size
    }
// Vincula los datos del modelo a las vistas del ViewHolder
    override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
// Carga la imagen de entrega en la vista de imagen usando Glide
        Glide.with(context).load(deliveryCategoryModels.get(position).image).into(holder.del
// Establece el número de factura en la vista de texto
        holder.delivery_invoice_nummber!!.setText(deliveryCategoryModels[position].invoiceNu
// Establece el listener para manejar clics en el elemento
        holder.setListener(object :IRecycleItemClickListener{
            override fun onItemClick(view: View, pos: Int) {
                EventBus.getDefault().postSticky(DeliveryItemClick(deliveryCategoryModels[pos
            }

        })
    }
}
```

A.4. Código DeliveryDetailsFragment

```
class DeliveryDetailsFragment : Fragment() {
//Declaración de variables
    private lateinit var deliveryDetailsViewModel: DeliveryDetailsViewModel

    private var shipperName_details: TextView? = null
    private var invoiceNummber_details: TextView? = null
    private var end_client_name_details: TextView? = null
    private var description_details: TextView? = null
    private var address_details: TextView? = null
    private var phone_details: TextView? = null
    private var date_details: TextView? = null
    private var status_title: TextView? = null
    private var btn_done_details: Button? = null
    private var btn_edit_description: FloatingActionButton? = null

    private var waitingDialog: AlertDialog? = null
    val btnstatusModel:DeliveryModel?= Common.deliverySelected

    // Alimentar el diseño de la interfaz de usuario
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        deliveryDetailsViewModel = ViewModelProviders.of(this).
        get(DeliveryDetailsViewModel::class.java)
        val root = inflater.inflate(R.layout.fragment_delivery_details, container, false)
        initViews(root)

        deliveryDetailsViewModel.getMutableLiveData().
        observe(viewLifecycleOwner, Observer {
            displayInfo(it)
        })



        return root
    }
//Guardar en FireBase los datos del estado del paquete
    private fun submitToFirebase(it: DeliveryModel?) {
        waitingDialog!!.show()
        val data = FirebaseDatabase.getInstance().getReference(Common.DELIVERY_REF)
        val data2 = data.child(Common.deliverySelected!!.package_id!!)
        data2.child("status").setValue(it!!.status)
        data2.child("dateOperation").setValue(it!!.dateOperation)
        data2.child("image").setValue(it!!.image)
```

```
47          waitingDialog!!.dismiss()
48      }
49  //Guardar la descripcion en Firebase
50      private fun submitToFirebaseDescription(it: DeliveryModel?) {
51          waitingDialog!!.show()
52          val data = FirebaseDatabase.getInstance().getReference(Common.DELIVERY_REF)
53          val data2 = data.child(Common.deliverySelected!!.package_id!!)
54          data2.child("description").setValue(it!!.description)
55          waitingDialog!!.dismiss()
56      }
57  //Mostrar los datos relacionados con el paquete
58      private fun displayInfo(it: DeliveryModel?) {
59          shipperName_details!!.text = StringBuilder(it!!.shipper!!)
60          invoiceNummber_details!!.text = java.lang.StringBuilder(it!!.invoiceNumber!!)
61          end_client_name_details!!.text = StringBuilder(it!!.endClientName!!)
62          description_details!!.text = StringBuilder(it!!.description!!)
63          address_details!!.text = StringBuilder(it!!.address!!)
64          phone_details!!.text = StringBuilder(it!!.phone!!)
65          date_details!!.text = StringBuilder(it!!.date.toString())
66          wichStatus(it)
67          if (btnstatusModel!!.status == Common.STATUS_DONE)
68              btn_done_details!!.isClickable = false
69
70      }
71  //En que estado se encuentra actualmente
72      private fun wichStatus(it: DeliveryModel) {
73          if (it.status == Common.STATUS_FREE){
74              status_title!!.setText(Common.txt_FREE)
75              status_title!!.
76              setBackgroundColor(Common.COLOR_STATUS_FREE)
77          } else if (it.status == Common.STATUS_INPROGRESS){
78              status_title!!.setText(Common.txt_INPROGRESS)
79              status_title!!.
80              setBackgroundColor(Common.COLOR_STATUS_INPROGRESS)
81          } else if (it.status == Common.STATUS_DONE){
82              status_title!!.setText(Common.txt_DONE)
83              status_title!!.
84              setBackgroundColor(Common.COLOR_STATUS_DONE)
85          }
86      }
87  //Establecer los datos del estado mostrados dentro del botón
88      private fun setStatusBoton() {
89          if (btnstatusModel!!.status == Common.STATUS_FREE){
90              btn_done_details!!.setText(Common.txt_TAKE_IT)
91              btn_done_details!!.
92              setBackgroundColor(Common.COLOR_STATUS_INPROGRESS)
93              btn_done_details!!.isClickable = true
94          } else if (btnstatusModel!!.status == Common.STATUS_INPROGRESS){
95              btn_done_details!!.setText(Common.txt_DONE)
```

```
96              btn_done_details!!.
97              setBackgroundColor(Common.COLOR_STATUS_DONE)
98              btn_done_details!!.setTextColor(Color.BLACK)
99              btn_done_details!!.isClickable = true
100         }else if (btnstatusModel!!.status == Common.STATUS_DONE){
101             btn_done_details!!.setText(Common.txt_ITS_DONE)
102             btn_done_details!!.isClickable = false
103             btn_done_details!!.
104             setBackgroundColor(Common.COLOR_STATUS_FINISHED)
105             btn_done_details!!.setTextColor(Color.BLACK)
106         }
107         wichStatus(btnstatusModel!!)
108     }
109 //Estado clickado
110     private fun wichStatusClicked(statusModel:
111     DeliveryModel?, fecha: String) {
112         if (statusModel!!.status == Common.STATUS_FREE) {
113             statusModel!!.status = Common.STATUS_INPROGRESS
114             statusModel!!.dateOperation += "----. Common.STATUS_INPROGRESS :{fecha}"
115
116             statusModel!!.image = Common.
117             IMG_DELIV_INPROGRESS
118             btn_done_details!!.setText(Common.txt_DONE)
119             btn_done_details!!.setBackgroundColor(Common.
120             COLOR_STATUS_DONE)
121             btn_done_details!!.setTextColor(Color.BLACK)
122         } else if (statusModel!!.status == Common.
123         STATUS_INPROGRESS) {
124             statusModel!!.status = Common.STATUS_DONE
125             statusModel!!.dateOperation += "--. Common.STATUS_DONE :{fecha}"
126
127             statusModel!!.image = Common.IMG_DELIV_DONE
128             btn_done_details!!.setText(Common.txt_ITS_DONE)
129             btn_done_details!!.isClickable = false
130             btn_done_details!!.setBackgroundColor(Common.
131             COLOR_STATUS_FINISHED)
132             btn_done_details!!.setTextColor(Color.BLACK)
133         }
134         wichStatus(statusModel!!)
135         deliveryDetailsViewModel.setStatusModel(statusModel!!)
136     }
137
138 //suprimir los errores de advertencia relacionados
139 //con el uso de recursos como colores
140     @SuppressLint("ResourceAsColor")
141     //vista inicial de la pantalla, carga los datos en ese instante
142     private fun initViews(root: View?) {
143         waitingDialog =
144             SpotsDialog.Builder().setContext(requireContext()).
```

```
145             setCancelable(false).build()
146
147         btn_done_details = root!!.findViewById(R.id.
148         btn_status_details_delivery) as Button
149         shipperName_details = root!!.findViewById(R.id.
150         delivery_name_details) as TextView
151         invoiceNummber_details = root!!.findViewById(R.id.
152         invoice_number_details_delivery)
153         end_client_name_details =
154         root!!.findViewById(R.id.end_client_details_delivery) as TextView
155         description_details = root!!.findViewById(R.
156         id.txt_delivery_details_description) as TextView
157         address_details = root!!.findViewById(R.id.
158         address_details_delivery) as TextView
159         phone_details = root!!.findViewById(R.id.
160         phone_details_delivery) as TextView
161         date_details = root!!.findViewById(R.id.
162         date_details_delivery) as TextView
163         status_title = root!!.findViewById(R.id.
164         txt_delivery_status_details)
165         btn_edit_description =
166         root!!.findViewById(R.id.btn_delivery_edit_description) as FloatingActionButton
167
168         setStatusBoton()
169         wichStatus(btnstatusModel!!)
170
171         btn_done_details!!.setOnClickListener {
172             showDialogStatus()
173             deliveryDetailsViewModel.getMutableStatusLiveData().
174             observe(viewLifecycleOwner, Observer {
175                 submitToFirebase(it)
176             })
177         }
178         btn_edit_description!!.setOnClickListener {
179             editPickupDescription()
180             deliveryDetailsViewModel.getMutableStatusLiveData().
181             observe(viewLifecycleOwner, Observer {
182                 submitToFirebaseDescription(it)
183                 displayInfo(it)
184             })
185         }
186
187     }
188 //Modificar el comentario del detalle
189     private fun editPickupDescription() {
190         if (btnstatusModel!!.status != Common.STATUS_DONE) {
191             var builder = AlertDialog.Builder(requireContext())
192             builder.setTitle("Edit description")
193
```

```
194             val itemView =
195                 LayoutInflater.from(context).inflate(R.layout.
196                 layout_delivery_edit_description, null)
197             var txt_description: EditText =
198             itemView.findViewById(R.id.txt_delivery_edit_descrption)
199             txt_description.setText(btnstatusModel!!.
200             description.toString())
201             builder.setView(itemView)
202             builder.setNegativeButton("CANCEL") { dialogInterface,
203             i -> dialogInterface.dismiss() }
204             builder.setPositiveButton("OK") { dialogInterface,
205             i ->
206                 val statusModel: DeliveryModel? = Common.
207                 deliverySelected
208                 statusModel!!.description = txt_description.
209                 text.toString()
210                 deliveryDetailsViewModel.
211                 setStatusModel(statusModel!!)
212             }
213             val dialog = builder.create()
214             dialog.show()
215         }
216
217     }
218
219     //requiere al menos la API nivel 26 (Android 8.0, Oreo)
220     @RequiresApi(Build.VERSION_CODES.O)
221     //Establecer la fecha en un formato concreto
222     fun dateOfZone():String{
223         val fromTimeZone =ZoneId.of("Africa/Cairo")
224         //Zona horaria, la más cercana a Beirut
225         val today = LocalDateTime.now() //fecha actual
226         val currentTime = today.atZone(fromTimeZone)
227         val DATE_FORMAT = "yyyy/MM/dd HH:mm:ss"
228         val formatter =DateTimeFormatter.ofPattern(DATE_FORMAT)
229         return formatter.format(currentTime).toString()
230     }
231
232     //Cuadro "Cancel" o "Ok"
233     @SuppressLint("NewApi", "ResourceAsColor")
234     private fun showDialogStatus() {
235         var builder = AlertDialog.Builder(requireContext())
236         builder.setTitle("Set Status of Task Delivery")
237
238         val itemView = LayoutInflater.from(context).inflate(R.
239         layout.layout_delivery_status, null)
240         builder.setView(itemView)
241         builder.setNegativeButton("CANCEL") { dialogInterface, i ->
242         dialogInterface.dismiss() }
```

```
243         builder.setPositiveButton("OK") { dialogInterface, i ->
244             val statusModel:DeliveryModel?= Common.deliverySelected
245             val fecha = dateOfZone()
246             wichStatusClicked(statusModel,fecha)
247         }
248         val dialog = builder.create()
249         dialog.show()
250     }
251
252
253 }
```

A.5. Código MenuPickUpFragment

```
1  class MenuPickUpFragment : Fragment() {
2  val txtTime_create = itemView.findViewById<EditText>
3  (R.id.txtTime_pickup_create_task)
4  if (!TextUtils.isEmpty(txtName_create.text.toString())
5  && !TextUtils.isEmpty(txtTime_create.text.toString())
6      && !TextUtils.isEmpty(txtAddress_create.text.toString())
7      && !TextUtils.isEmpty(txtPhone_create.text.toString())
8      && !TextUtils.isEmpty(txtDescriptio_create.text.toString())) {
9      val statusModel: PickUpModel? = PickUpModel()
10     statusModel!!.dateOperation = formatter.format(currentTime).toString()
11     statusModel!!.date = formatter2.format(currentTime).toString()
12     statusModel!!.status = Common.STATUS_FREE
13     statusModel!!.user = usuarioDriver
14     statusModel!!.address = txtAddress_create!!.text.toString()
15     statusModel!!.description = txtDescriptio_create!!.text.toString()
16     statusModel!!.image = Common.IMG_FREE
17     statusModel!!.name = txtName_create!!.text.toString().toUpperCase()
18     statusModel!!.phone = txtPhone_create!!.text.toString()
19     statusModel!!.time = txtTime_create!!.text.toString()
20     Common.pickupSelected = statusModel
21     menuViewModel.setCreateModel(statusModel!!)
22     menuViewModel.getMutableCreateLiveData().observe(viewLifecycleOwner, Observer {
23         submitToFirebase(it)
24     })
```

A.6. Historial Git

Historial de Commits

1. Jun 21, 2020
   Commit: 164e058d
   Mensaje: First push
2. Jun 21, 2020
   Commit: 70993bc3
   Mensaje: Delivery función Home con click en iconos funcionando, Cuidado con los nombres...
3. Jun 21, 2020
   Commit: 49ba6fbb
   Mensaje: Cambiado el proceso de listar y cambiar los estados en los detalles
4. Jun 21, 2020
   Commit: ad0075a5
   Mensaje: Icono de empresa en app
5. Jun 24, 2020
   Commit: d94c91a2
   Mensaje: Solucion a algunos problemas de refresco. Creación de paquetes para el Delivery
6. Jun 26, 2020
   Commit: 665c40b0
   Mensaje: Mostrar Usuario OK
7. Jun 27, 2020
   Commit: 2b2c3a74
   Mensaje: MenuPick OK
8. Jun 27, 2020
   Commit: f593f989
   Mensaje: Delivery Create Empty OK
9. Jun 29, 2020
   Commit: caaae077
   Mensaje: Deployed first Version App To Client. Fixed Not enter all data on create Task, fixed Login
10. Jun 29, 2020
    Commit: a97159e6
    Mensaje: Change Db Ref
11. Jul 01, 2020
    Commit: bae8c88c
    Mensaje: Change Login to only admin person, and add Spinner to layout
12. Jul 13, 2020
    Commit: a54ef02a
    Mensaje: Add Users on Tasks PickUp Ok

A.7. Gráficas

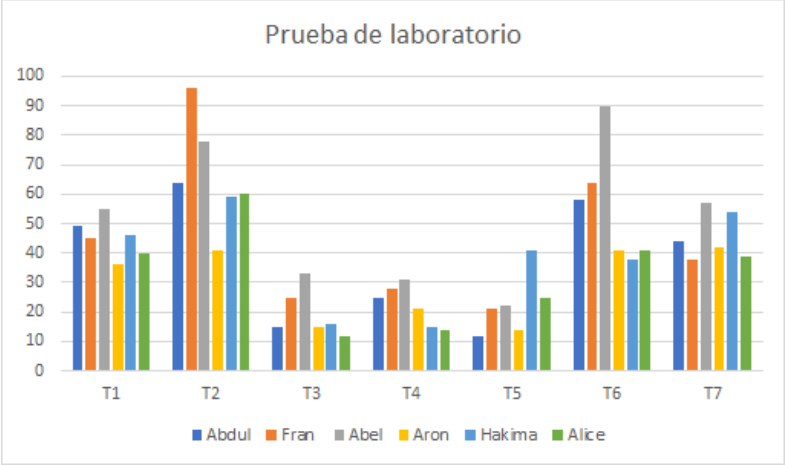

Figura A.1: Tiempos de Pruebas de Laboratorio.

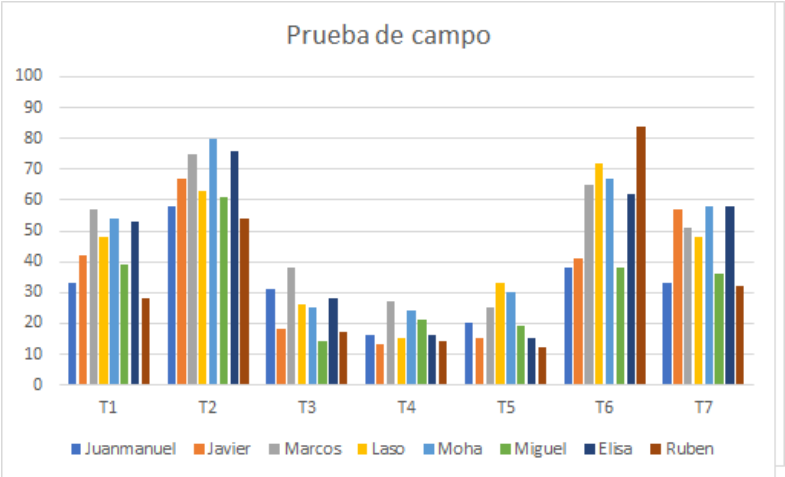

Figura A.2: Tiempos de Pruebas de Campo.

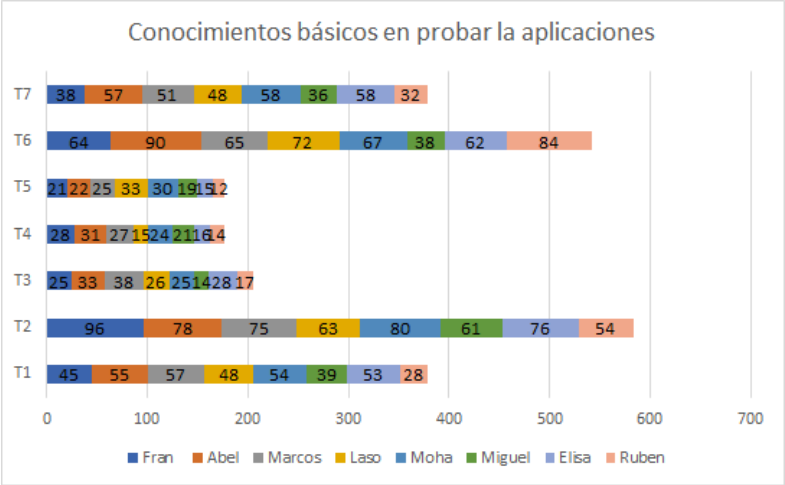

Figura A.3: Tiempos de usuarios con conocimientos básicos en probar la aplicaciones.

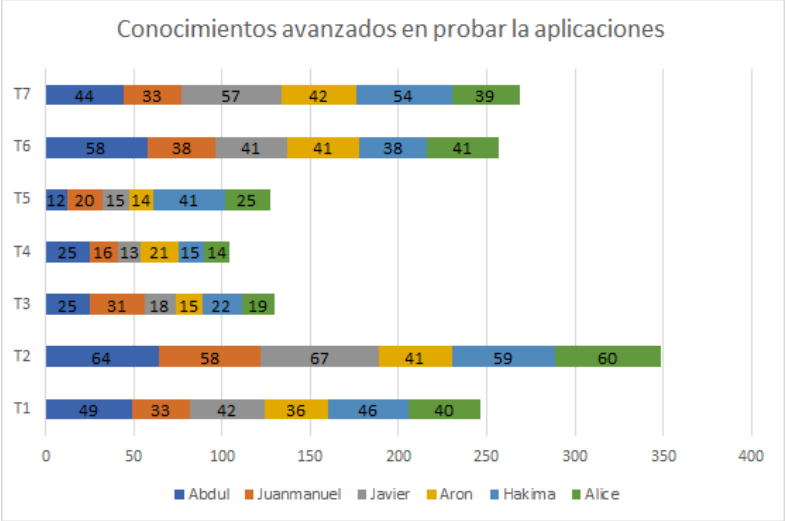

Figura A.4: Tiempos de usuarios con conocimientos avanzados en probar la aplicaciones.

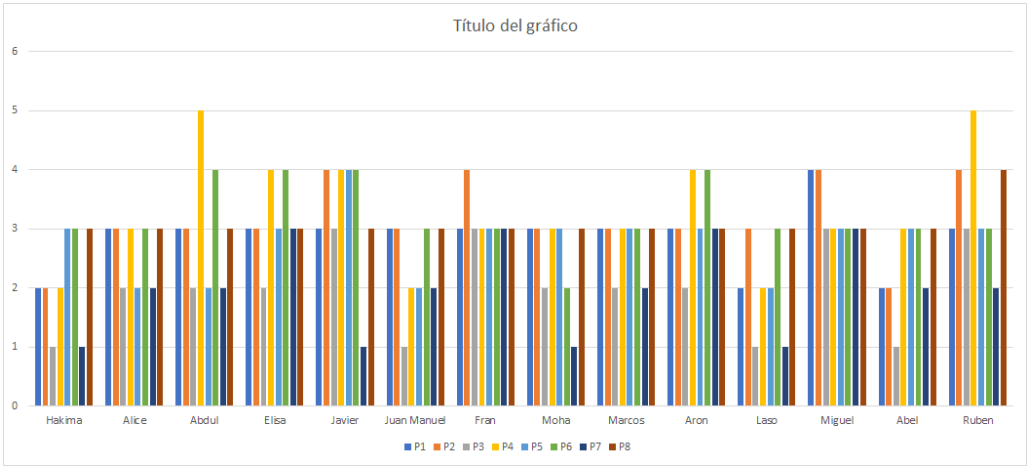

Figura A.5: Puntuación de la dificultad de realizar la prueba donde (1-Muy difícil y 5-Nada difícil).

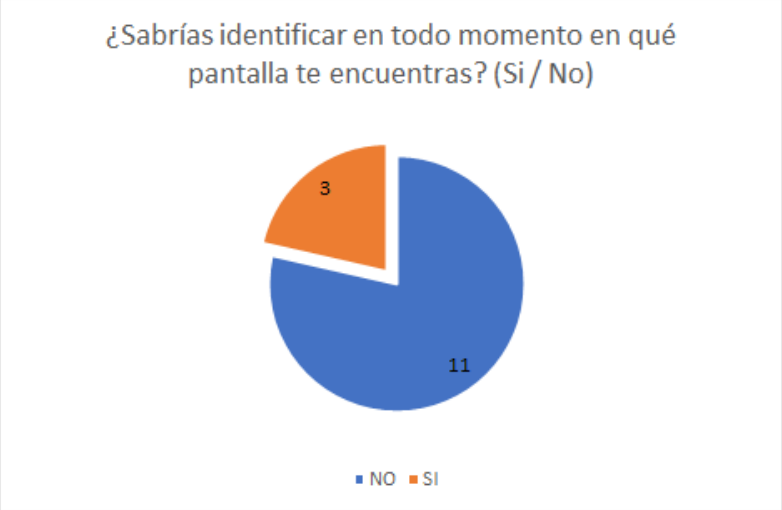

Figura A.6: ¿Sabrías identificar en todo momento en qué pantalla te encuentras? (Si / No).

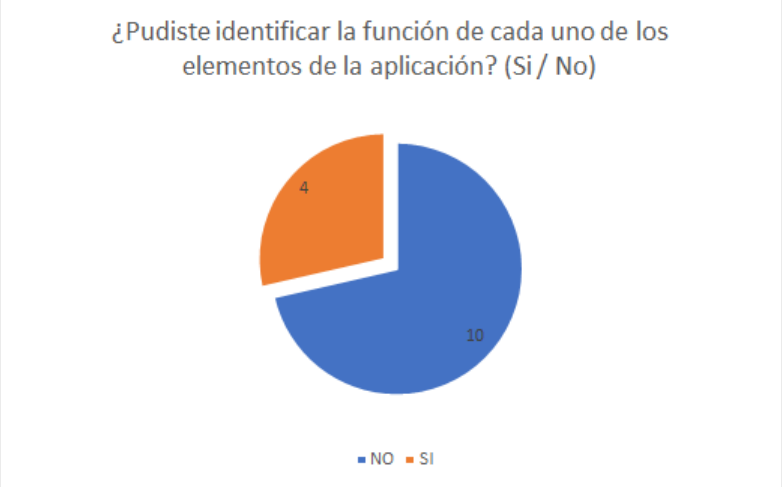

Figura A.7: ¿Pudiste identificar la función de cada uno de los elementos de la aplicación? (Si / No).

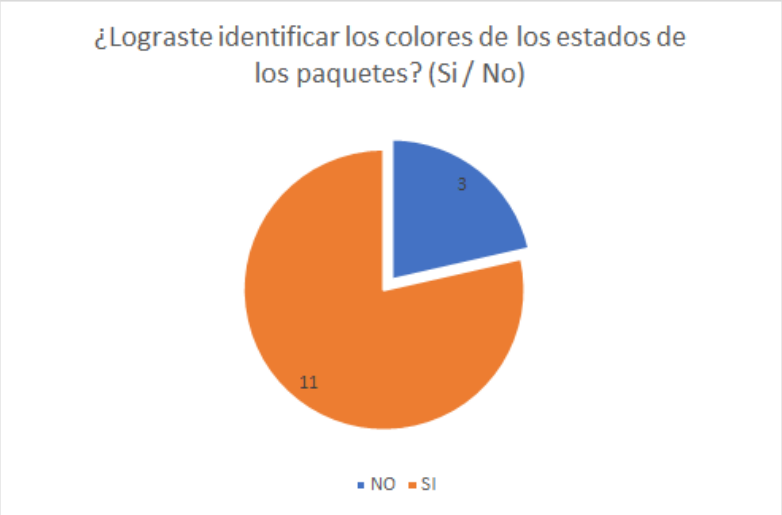

Figura A.8: ¿Lograste identificar los colores de los estados de los paquetes? (Si / No).

Figura A.9: ¿Te resultó claro desempeñar las tareas propuestas? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

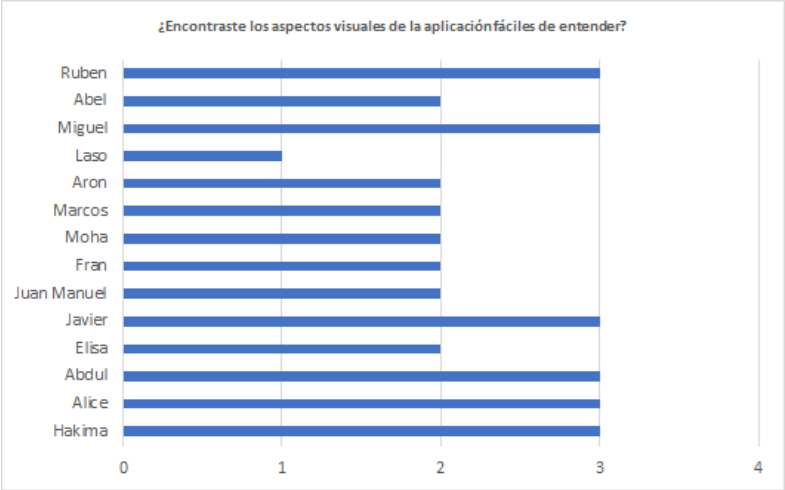

Figura A.10: ¿Encontraste los aspectos visuales de la aplicación fáciles de entender? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

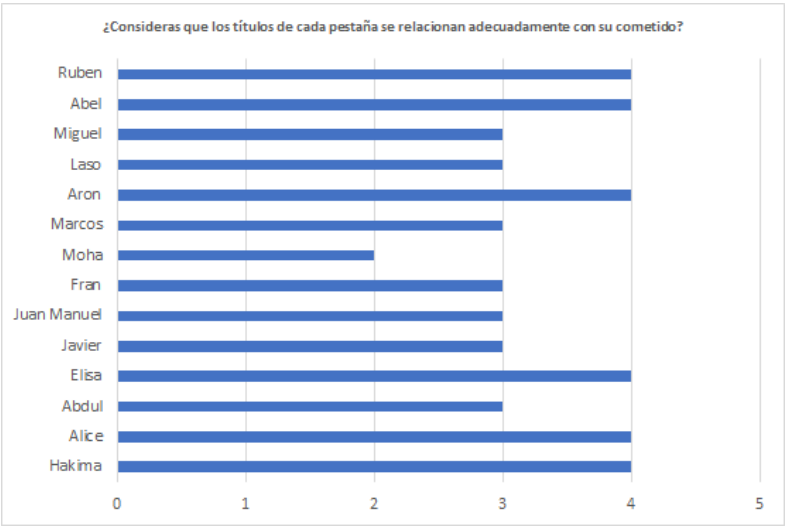

Figura A.11: ¿Consideras que los títulos de cada pestaña se relacionan adecuadamente con su cometido? (1 - Nada de acuerdo / 5 - Totalmente de acuerdo).

A.8. Tablas de resultados

	Grupo 1		Grupo 2			
	Fran	Abel	Abdul	Aron	Hakima	Alice
T1	45	55	49	36	46	40
T2	96	78	64	41	59	60
T3	25	33	15	15	16	12
T4	28	31	25	21	15	14
T5	21	22	12	14	41	25
T6	64	90	58	41	38	41
T7	38	57	44	42	54	39

Tabla A.1: Resultados de los tiempos de las prueba de laboratorio

	Grupo 3						Grupo 4	
	Marcos	Laso	Moha	Miguel	Elisa	Ruben	Juan Manuel	Javier
T1	57	48	54	39	53	28	33	42
T2	75	63	80	61	76	54	58	67
T3	38	26	25	14	28	17	31	18
T4	27	15	24	21	16	14	16	13
T5	25	33	30	19	15	12	20	15
T6	65	72	67	38	62	84	38	41
T7	51	48	58	36	58	32	33	57

Tabla A.2: Resultados de la prueba de campo

	Abdul	Juan Manuel	Javier	Aron	Hakima	Alice
T1	49	33	42	36	46	40
T2	64	58	67	71	86	60
T3	15	10	12	15	16	12
T4	25	16	13	21	15	14
T5	12	20	15	14	41	25
T6	58	38	41	41	38	41
T7	44	33	57	42	54	39

Tabla A.3: Resultados de tiempos de los participantes con conocimientos avanzados en probar aplicaciones móviles