Acta de Correcciones al Proyecto de Grado
Ingeniería de Sistemas y Computación

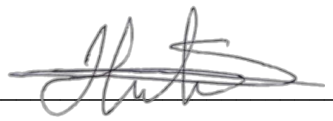
Fecha: 17 de febrero de 2023

Autores: Nicolás Sebastián Delgado Ríos, Karen Camila Paladines Muñoz

Nombre del Proyecto de Grado: Prototipo funcional de una aplicación móvil como complemento al proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion

Director: Ing. Héctor David Delgado Ríos

Como indica el artículo 2.27 de las Directrices de Trabajo de Grado, he verificado que los estudiantes indicados arriba han implementado todas las correcciones que los Jurados del Proyecto de Grado definieron que se efectuaran, como consta en el Acta de Calificación correspondiente.



Ing. Héctor David Delgado Ríos
Director del Proyecto de Grado

Nota de Aceptación

Aprobado por el Comité de Trabajo de Grado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana para optar el título de Ingeniero de Sistemas y Computación.

Dr. Hernán Camilo Rocha Niño
Decano de la Facultad de Ingeniería

Dr. Gerardo Mauricio Sarria Montemiranda
Director Carrera Ingeniería de Sistemas y Computación

Ing. Héctor David Delgado Ríos
Director Trabajo de Grado

Dr. Gerardo Mauricio Sarria Montemiranda
Codirector Trabajo de Grado

Mg. Juan Pablo García Cifuentes
Jurado 1

Dra. Luisa Fernanda Rincón Pérez
Jurado 2

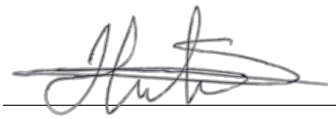
Santiago de Cali, 16 de febrero de 2023.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria Montemiranda.
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Por medio de la presente me permito informarle que los estudiantes de Ingeniería de Sistemas y Computación: Nicolás Sebastián Delgado Ríos (código: 8943848) y Karen Camila Paladines Muñoz (código: 8944079) trabajaron bajo mi dirección en el proyecto de grado titulado "Prototipo funcional de una aplicación móvil como complemento al proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion", el cual se encuentra corregido.

Atentamente,



Ing. Héctor David Delgado Ríos

PIYION INC
EIN: 37-2064343
5350 NW 84th AVE, Doral 33166,
Miami, Florida
+1 (754) 317 0892

Santiago de Cali, enero 22 del 2023

Señores
Pontificia Universidad Javeriana Cali
Dr. Gerardo Mauricio Sarria Montemiranda
Director Carrera de Ingeniería en Sistemas y Computación
Cali

Cordial saludo,

Por medio de la presente me permito informarles que los estudiantes de Ingeniería en Sistemas y Computación: **Camila Paladines** y **Nicolás Delgado** cumplieron con todos los compromisos adquiridos con nuestra empresa para la realización de su Trabajo de Grado titulado *"Prototipo funcional de una aplicación móvil como complemento al proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion"*.

Atentamente,

Lina Vanessa Espinosa Arias.

piyion
01-22-2023

Lina Vanessa Espinosa Arias
COO. & Co-founder, Piyion INC

Santiago de Cali, 16 de febrero de 2023.

Señores
Pontificia Universidad Javeriana Cali.
Dr. Gerardo Mauricio Sarria Montemiranda.
Director Carrera de Ingeniería de Sistemas y Computación.
Cali.

Cordial Saludo.

Nos permitimos presentar a su consideración el trabajo de grado titulado "Prototipo funcional de una aplicación móvil como complemento al proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion" con el fin de cumplir con los requisitos exigidos por la Universidad para optar al título de Ingeniero de Sistemas y Computación.

Al firmar aquí, damos fe que entendemos y conocemos las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,

Nicolás Sebastián Delgado Ríos
Código: 8943848

Karen Camila Paladines Muñoz
Código: 8944079

Pontificia Universidad Javeriana Cali
Facultad de Ingeniería
Ingeniería de Sistemas y Computación
Trabajo de Grado

Prototipo funcional de una aplicación móvil como complemento al proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion

Nicolás Sebastián Delgado Ríos
Karen Camila Paladines Muñoz

Director: Ing. Héctor David Delgado Ríos
Co-director: Dr. Gerardo Mauricio Sarria Montemiranda

16 de febrero de 2023

Resumen

El uso de dispositivos móviles ha incrementado notablemente en los últimos años, puesto que su portabilidad se vuelve útil en distintos escenarios, permitiendo a las personas realizar actividades sin la necesidad de contar con un dispositivo de mayor tamaño y capacidad. Además, al ser de menor costo, es más asequible que los computadores personales, y el día de hoy un gran porcentaje de la población cuenta con un teléfono inteligente.

La realidad de los usuarios de Piyion, una plataforma que permite unificar todos los canales de comunicación y administrar los asesores de una empresa, no es diferente; debido a este creciente uso de dispositivos móviles, y la necesidad de los usuarios de Piyion de contar con una aplicación móvil en la que puedan extender el funcionamiento de la aplicación web para facilitar algunas tareas que no son necesarias de completar en un computador personal, en este trabajo de grado se desarrolló una aplicación móvil como complemento a la aplicación web del sistema, con el fin de apoyar las actividades que llevan a cabo los usuarios dentro de Piyion.

Palabras Clave: Chat, Comunicación, Piyion, Atención al cliente, Aplicación móvil, Ingeniería de Software, Desarrollo móvil.

Abstract

The use of mobile devices has increased significantly in recent years, since their portability has become useful in different scenarios, allowing people to perform activities without the need for a larger and more capacious device. In addition, being lower cost, it is more affordable than personal computers, and today a large percentage of the population has a smartphone.

The reality of Piyion users, a platform that allows unifying all communication channels and managing the advisors of a company, is no different; due to this growing use of mobile devices, and the need of Piyion users to have a mobile application in which they can extend the operation of the web application to facilitate some tasks that are not necessary to complete on a personal computer, in this degree work a mobile application was developed as a complement to the web application of the system, in order to support the activities carried out by users within Piyion.

Keywords: Chat, Communication, Piyion, Customer Service, Mobile Application, Software Engineering, Mobile Development.

Por todos aquellos que están y los que no. Por aquellos que aún viven en nuestros recuerdos y nos motivan a seguir adelante: "ser un buen recuerdo en la vida de alguien es una forma de quedarse para siempre"

Índice general

Índice de figuras

Introducción

Muchas empresas requieren de un sistema de atención al cliente que les permita llegar a las personas interesadas en sus productos o servicios, con el fin de proporcionarles la información adecuada para que estos logren adquirirlos. En la actualidad existen diferentes medios por los cuales se puede llegar a ofrecer dicha atención, como el chat en línea de la página web de la empresa, sus redes sociales, correo electrónico, llamadas telefónicas, entre otros. Sin embargo, debido a esta gran diversidad de canales de comunicación, la manera en que se brinda la atención puede presentar diferentes inconvenientes, como la poca organización de las conversaciones, las limitaciones relacionadas al personal encargado de atender las solicitudes por problemas con el manejo de las cuentas de las redes sociales, etc.

Para dar solución a las dificultades anteriormente mencionadas, se crea la aplicación Piyion [4], que proporciona diferentes servicios para la gestión de los chats que pueden venir desde la página web de la empresa y algunas de las principales redes sociales, como Instagram, Facebook, Telegram y WhatsApp. Además, permite el manejo de estas conversaciones por varias personas (llamados agentes), que son quienes brindan la atención a los clientes. También, en esta aplicación es posible guardar la información de contacto de las personas atendidas (si estos la proporcionan), haciendo que si estas recurren de nuevo al chat, sea posible brindar una mejor atención y se permita hacer un seguimiento de manera efectiva.

Por otro lado, con el creciente uso de dispositivos móviles y el papel fundamental que han jugado en el día a día de las personas, la necesidad de contar con una aplicación móvil que complemente el uso de la aplicación web de Piyion ha incrementado notablemente, debido a que es posible aprovechar varias de las ventajas que brindan estos dispositivos para el desarrollo de las actividades fundamentales para la atención al cliente, como la pronta respuesta por parte de los agentes, que conlleva a una buena relación entre la empresa y las personas interesadas en sus productos.

Teniendo en cuenta lo anterior, en este documento se presenta el proceso de desarrollo que se llevó a cabo para implementar una aplicación móvil que cuenta con las principales funcionalidades que tiene la aplicación web de Piyion, apoyando en la ejecución de tareas que los usuarios pueden completar desde un dispositivo móvil.

Capítulo 1

Descripción del Problema

1.1. Planteamiento del Problema

La atención al cliente es un componente esencial para muchas empresas, puesto que mediante esta es posible una comunicación directa con las personas potencialmente interesadas en adquirir sus productos o servicios, o en su defecto, clientes que se deben mantener y a los que hay que brindarles soporte y garantías de calidad. Es posible brindar esta atención mediante diferentes canales de comunicación, como los chats en línea dispuestos en la página web de las empresas, correo electrónico, redes sociales, entre otros.

Como posible solución a esta problemática se creó la aplicación web Piyion, cuyo objetivo es unificar las conversaciones de atención a los clientes que se pueden comunicar desde diferentes medios como la página web y algunas redes sociales, pudiendo gestionarlos desde un solo lugar. Además, también es posible que dichas conversaciones sean atendidas por diferentes personas dentro de la empresa, con el fin de optimizar la respuesta a los clientes y eliminar los obstáculos que ocasiona la diversidad de canales disponibles.

Piyion brinda una amplia gama de servicios, entre los que destacan por supuesto, el manejo de las conversaciones que vienen desde diferentes canales, así como también la gestión de los agentes, que son los encargados de atender dichas conversaciones. Adicionalmente, en esta herramienta es posible guardar los datos de contacto de las personas que fueron atendidas, permitiendo un seguimiento a estos de manera efectiva.

Hasta el momento, todas estas opciones están disponibles en la aplicación web. Sin embargo, teniendo en cuenta la gran utilidad que representan los dispositivos móviles hoy en día y la necesidad de los usuarios de Piyion de facilitar el uso de la plataforma sin las limitaciones que un computador de escritorio representa, se plantea la posibilidad de extender las funcionalidades actuales a un dispositivo donde se puedan aprovechar al máximo sus virtudes. En este sentido, cabe preguntarse: ¿de qué manera facilitar el proceso de respuesta a conversaciones y la gestión de agentes dentro de Piyion?

Para dar solución a este problema se plantea la implementación de una aplicación móvil que complemente a la aplicación web, permitiéndole a los usuarios atender a los chats de manera oportuna y cómoda desde sus dispositivos móviles. En este caso, no es estrictamente necesario contar con un computador personal para la ejecución de sus funciones. Esto sería de gran utilidad, ya que tanto agentes como administradores tienen la oportunidad de aprovechar los beneficios de los

dispositivos móviles en la realización de sus tareas, como la respuesta a los chats en casi cualquier momento y el seguimiento a la calidad de la atención por parte de los miembros del equipo.

1.1.1. Formulación

¿Cómo diseñar e implementar una aplicación móvil que complemente el proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion?

1.1.2. Sistematización

- ¿Cuáles son los requerimientos funcionales y no funcionales de la aplicación móvil?
- ¿Cómo será el diseño de la aplicación y de los componentes a desarrollar?
- ¿Cómo se va a implementar la aplicación?
- ¿Cómo verificar el correcto funcionamiento de la aplicación?

1.2. Objetivos

1.2.1. Objetivo General

Implementar una aplicación móvil que complemente el proceso de respuesta a conversaciones y gestión de agentes dentro de Piyion.

1.2.2. Objetivos Específicos

- Establecer los requerimientos funcionales y no funcionales de la aplicación.
- Diseñar la aplicación y los componentes a desarrollar.
- Desarrollar un prototipo funcional que cumpla con los requerimientos definidos para la aplicación.
- Verificar el correcto funcionamiento de la aplicación.

Capítulo 2

Desarrollo del Proyecto

2.1. Marco de Referencia

2.1.1. Áreas Temáticas

Tomando como base la clasificación de la ACM Computing Classification System [5], las áreas temáticas que se trabajaron en el proyecto de grado son las que se mencionan a continuación.

- Software and its engineering → Software creation and management → Designing software.
- Software and its engineering → Software organization and properties → Software system structures → Software architectures.
- Software and its engineering → Software creation and management → Software verification and validation → Software prototyping.
- Software and its engineering → Software notations and tools → Software configuration management and version control systems.
- CCS → Human-centered computing → Human computer interaction (HCI) → Interaction devices → Touch screens.
- CCS → Applied computing → Computers in other domains → Personal computers and PC applications.

2.1.2. Marco Teórico

En este proyecto de grado se implementó una aplicación móvil que complementa los servicios ofrecidos por la aplicación web de Piyion, con el fin de facilitar algunas tareas que son posibles de realizar en un dispositivo móvil. Por esto, a continuación se presentan los conceptos más importantes usados durante su desarrollo.

2.1.2.1. Piyion

Piyion es una empresa creada recientemente, la cual está en constante crecimiento y evolución. Consiste en una propuesta acerca de una plataforma de omnicanalidad para las empresas, de tal forma que se pueda gestionar de mejor manera los chats de sus canales digitales como Instagram, Facebook, WhatsApp, Telegram y página web. Adicionalmente, Piyion facilita la gestión de los

asesores de una empresa y le permite a los administradores saber si están dando o no una buena atención a los clientes. Se pueden identificar los siguientes servicios:

- **Gestión de conversaciones**

  Consiste en el manejo de las diferentes solicitudes de atención al cliente. Estas pueden tener como origen cinco canales disponibles: Instagram, Facebook, WhatsApp, Telegram y el chat en línea de la página web de la empresa. En este sentido, lo que se pone a disposición de los usuarios de Piyion es un espacio de chats donde tienen la posibilidad de atender a los clientes usando mensajes de texto y algunos tipos de contenido multimedia, como imágenes, documentos PDF, audios, entre otros. Además, se permite diferenciar entre las conversaciones que ya han sido asignadas a un asesor y las que aún están pendientes para ser atendidas.

- **Gestión del equipo**

  Es posible administrar la información relacionada a los miembros del equipo que brindan la atención al cliente. Esto incluye, por supuesto, la funcionalidad de invitar a una persona (vía correo electrónico) para que haga parte del equipo. Por otro lado, se permite ver y modificar algunos datos como el rol de un usuario dentro del sistema (agente o administrador), la máxima cantidad de clientes que puede atender, etc. Cabe resaltar que estas opciones sólo están disponibles para miembros del equipo con el rol de administrador.

- **Supervisión del equipo**

  Permite a los usuarios administradores poder ver en tiempo real el contenido de las conversaciones que están llevando a cabo los agentes. Este es un espacio para que se pueda verificar la calidad de la atención que favorece notablemente a la empresa. Debido a su naturaleza, posee una estructura similar a la que tiene la gestión de conversaciones, es decir, una lista de chats y una zona de mensajes, pero esta vez sin la posibilidad de interactuar en la conversación (es decir, en modo lectura).

- **Gestión de contactos**

  Los contactos son todas aquellas personas que reciben atención por parte de los asesores (cualquiera que sea su rol), y que, debido a su interés en los productos o servicios que estos ofrecen, brindan información de contacto para mantener la comunicación y agilizar los procesos que se lleven a cabo en futuras ocasiones. Esta información corresponde a sus nombres, apellidos, número de celular, correo electrónico, empresa de la que hace parte (si es el caso) y los canales desde los cuales se ha comunicado; sin embargo, sólo es obligatorio ingresar los nombres, apellidos y número de celular para guardar un contacto. Una vez guardado, es posible visualizar estos datos ya sea en la sección de “Contactos” o en la pestaña “Datos de usuario” que aparece en la conversación con dicho cliente (en la sección de “Chats”).

Aunque Piyion se encuentra en constante crecimiento, los servicios anteriormente mencionados hacen parte del núcleo del sistema, por lo que fueron tomados como referencia para la construcción de la aplicación móvil.

2.1.2.2. Canales de comunicación

Los canales de comunicación son un medio por el cual se transmite un mensaje que es enviado por un emisor hacia un receptor objetivo [6]. Estos canales son herramientas muy utilizadas por las empresas para establecer relaciones o comunicarse con potenciales clientes. Existen diferentes canales de comunicación y la elección de estos puede resultar crucial para la relación con el cliente. Algunos de los canales más populares son:

- **Llamadas**

  Este medio permite una comunicación más directa al dar la posibilidad de ofrecer una atención inmediata al usuario/consumidor por parte de la empresa, ya que se trata de un medio voz a voz instantáneo [6].

- **E-mail**

  Este es considerado por las empresas como el medio más importante, al ser una herramienta muy simple y eficaz [7]. En el mundo de los e-mails existen una serie de proveedores, servicios, herramientas adicionales que permiten una mayor medición en cuanto a las estadísticas de lectura, apertura o aceptación de los correos electrónicos.

- **Chat online**

  Dentro de estos se consideran los chats embebidos en páginas web, chatbots que remiten a un asesor, entre otros medios. Son una excelente opción de comunicación con el cliente. En este tipo de medios digitales existe una gran diferencia respecto a las llamadas. Y es que las llamadas permiten que un solo asesor de la empresa se comunique con solamente un cliente a la vez, mientras que con los medios digitales se permite una interacción de uno a muchos, es decir, un solo asesor atiende a muchos clientes.

- **Redes sociales**

  En los últimos años han tenido un crecimiento exponencial, tanto en la cantidad de usuarios que la utilizan como en el número de empresas que confían en estas para potencializar o dar a conocer su negocio. Tanto así que alrededor de 90 millones de empresas pequeñas usan Facebook [8]. Estas mismas empresas también pautan con Meta para darse a conocer, ya que lograr un crecimiento orgánico es cada vez más difícil, al tener tanto volumen dentro de las plataformas.

- **SMS**

  Cada vez más personas, en especial los jóvenes, usan frecuentemente el teléfono celular, esto debido al crecimiento de las redes sociales [9]. Todo el mundo quiere mantenerse conectado a las redes sociales, hacer esto mediante un celular o teléfono inteligente es mucho más cómodo. Es por esto que las empresas muchas veces optan por hacerse notar o enviar campañas promocionales a través de mensajes de texto o SMS, para así aprovechar esa oportunidad, en la que se presenta el uso frecuente de dispositivos móviles [6].

2.1.2.3. Chat multiagente

Un chat multiagente se refiere a un chat de atención al cliente que permite que varios agentes brinden dicha atención por un mismo canal. Un ejemplo sencillo es el chat multiagente de WhatsApp que proporciona Callbell [10], el cual ofrece la posibilidad de que varias personas puedan realizar su atención a los diferentes clientes con un mismo número de celular. En este sentido, se soluciona el problema de que la bandeja de entrada sólo esté disponible para una persona.

2.1.2.4. Aplicación móvil

También conocida como "app", se define como *un programa informático desarrollado específicamente para su uso con un dispositivo móvil, como un teléfono inteligente o una tableta* [11]. Estas se diferencian de las aplicaciones web o de escritorio al estar destinadas para dispositivos que cuentan con una menor capacidad de procesamiento en comparación con un computador de escritorio, por lo que es necesario que estén optimizadas para éstos [12].

2.1.2.5. Desarrollo de software

El desarrollo de software se puede definir como *un conjunto de actividades informáticas dedicadas al proceso de creación, diseño, despliegue y soporte de software* [13]. En este sentido, es podría decir que se trata de una serie de pasos definidos que llevan a la obtención de un producto final que es un programa de software. Se han creado diversos modelos que definen el proceso de desarrollo de software, algunos de ellos son el modelo en cascada, iterativo, evolutivo, en espiral, etc.

En este proyecto se usó metodologías ágiles, que tienen un enfoque iterativo que ayuda a los equipos a ofrecer valor a sus clientes de forma rápida y con menos dolores de cabeza [14]. Se centra principalmente en la entrega de incrementos pequeños pero lo suficientemente consumibles por los usuarios. En este sentido, se evalúan constantemente los requisitos, planes y resultados, permitiéndole a los equipos responder adecuadamente ante los cambios. De manera gráfica, se puede comprender mejor el funcionamiento de la metodología en la Figura 2.1.

Se eligió este enfoque de desarrollo debido a sus ventajas de flexibilidad y visualización rápida de resultados.

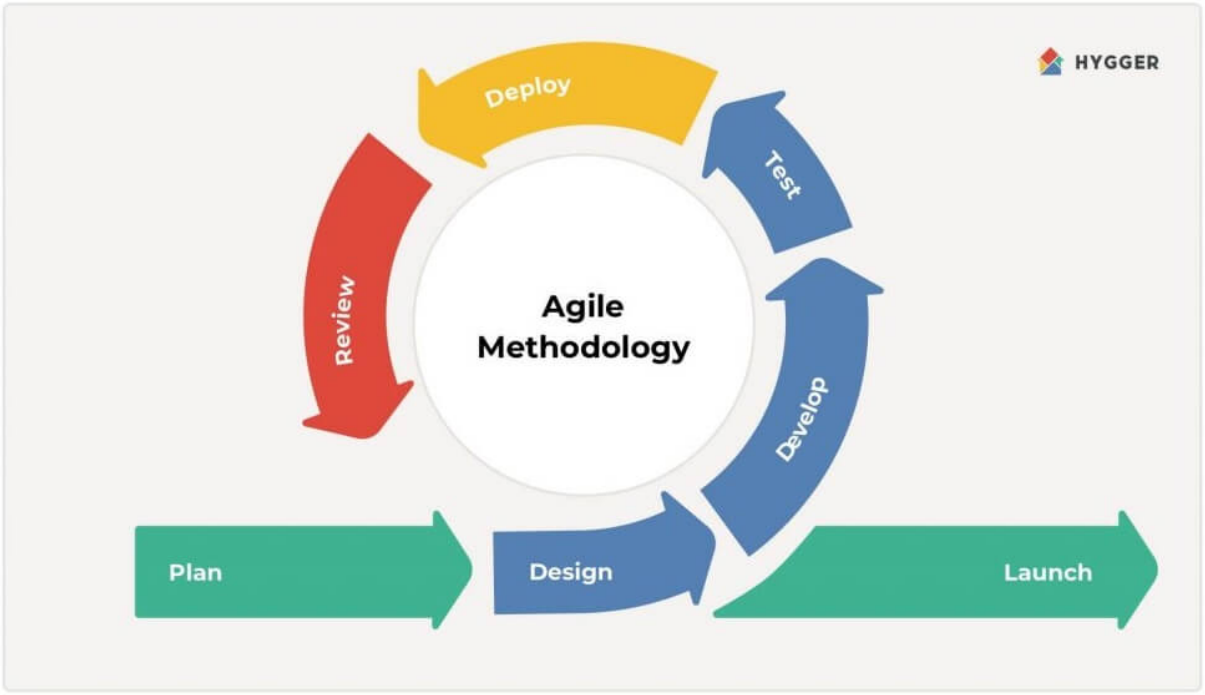


Figura 2.1: Metodologías ágiles [1]

2.1.2.6. Desarrollo móvil

El desarrollo móvil es la *creación de software destinado a funcionar en dispositivos móviles y optimizado para aprovechar las características y el hardware únicos de esos productos* [15]. De esta forma, se trata de realizar un proceso de desarrollo que permita la construcción de software en celulares y tablets.

El uso de dispositivos móviles se ha ido incrementando a lo largo de los últimos años, y es por esto que el desarrollo de aplicaciones móviles es un medio de creación de software cada vez más popular. A pesar de tener sus bases en el desarrollo de software tradicional, este busca utilizar de la mejor manera las particularidades en términos de hardware que tienen estos dispositivos, para lograr crear herramientas nuevas que no era posible realizar mediante los dispositivos de escritorio.

El proceso de desarrollo móvil se fundamenta en el proceso de desarrollo de software en general. Se destacan las siguientes etapas [16]:

1. **Inicio**. Se comienza con una idea que se va perfeccionando hasta llegar a convertirse en una base sólida para una aplicación.

2. **Diseño**. Consiste en definir tanto la experiencia de usuario de la aplicación (diseño general, funcionamiento, etc.) como el diseño de la interfaz de usuario (de más alta fidelidad).

3. **Desarrollo**. Consiste en la construcción real de la aplicación, en la que se trabaja sobre los diseños elaborados en la etapa anterior.

4. **Estabilización**. Sucede cuando el desarrollo está lo suficientemente avanzado como para que se realice el control de calidad donde se prueba la aplicación y se corrijan los errores. Además, existe la posibilidad de que la aplicación entre en una fase beta limitada en la que los usuarios tienen la oportunidad de utilizarla y aportar sus comentarios y sugerencias de cambio.

5. **Despliegue**. Ocurre cuando la aplicación ya está estabilizada y lista para sacarla al mercado. Existen diversas opciones de distribución dependiendo de la plataforma, como Apple App Store (para iOS) y Google Play Store (para Android).

Cabe resaltar que estas etapas se pueden usar en cualquier metodología de ciclo de vida del desarrollo de software, como Cascada, Espiral, Agile, etc. Sin embargo, a pesar de que el desarrollo de aplicaciones móviles no es muy diferente del desarrollo de software tradicional en términos de proceso o arquitectura, se deben tener en cuenta algunas consideraciones especiales [16]:

- **Multitarea**

  En los dispositivos móviles sólo se tiene una sola aplicación en primer plano a la vez. Además, si hay más de una aplicación abierta y realizando tareas, esto puede agotar la batería rápidamente.

- **Tamaño de la pantalla**

  Teniendo en cuenta que hay dos tipos de dispositivos móviles (celulares y tablets), los controles de la interfaz de usuario se han de diseñar específicamente para ser eficaces en pantallas más pequeñas.

- **Recursos limitados**

  Aunque los dispositivos móviles han ido mejorando su potencia a lo largo de los últimos años, no dejan de tener capacidades limitadas en comparación con los computadores personales. Por ejemplo, es muy común que en los dispositivos móviles se consuma rápidamente toda la memoria disponible con sólo cargar algunas imágenes de alta calidad.

2.1.2.7. Prototipo funcional

Un prototipo funcional se define como *una muestra o modelo de un producto que se construye para probar un concepto o proceso, o para que sirva de apoyo visual y se pueda reproducir, mejorar y aprender de él* [17]. En el caso del desarrollo de software, un prototipo tiene el objetivo de mostrar el funcionamiento de un sistema sin la necesidad de ser desplegado o llevado a los usuarios para su uso comercial [18]. En el prototipo funcional, es posible validar los requerimientos planteados inicialmente, lo que suele ahorrar tiempo y dinero.

2.1.2.8. Arquitectura serverless

La arquitectura "serverless", también conocida como FaaS (Functions as a Service) o "Funciones como Servicio"; es un modelo en el cual se pueden crear y ejecutar servicios sin tener que administrar ni preocuparse por la infraestructura, ya que un tercero, como Google o AWS (Amazon Web Services), se encargará de eso. Con este modelo, no se tiene que aprovisionar, escalar ni mantener servidores para ejecutar aplicaciones, bases de datos y sistemas de almacenamiento [19]; ya que el proveedor es quien se encarga de esto. Este modelo reduce costos, ya que no se deben mantener servidores por horas, sino que cuando un servicio es solicitado, el proveedor le brinda los recursos necesarios de manera dinámica hasta que termine su ejecución, que usualmente son milisegundos; permitiendo al usuario que pague solo por los recursos utilizados [19].

Mediante el uso de una arquitectura sin servidor, los desarrolladores se pueden enfocar en el producto principal en lugar de preocuparse por la administración y el funcionamiento de los servidores, o los tiempos de ejecución, tanto en la nube como en las instalaciones. Gracias a esta reducción de gastos, los desarrolladores pueden emplear tiempo y energía en desarrollar productos increíbles que sean de confianza y se puedan escalar. [19]

2.1.2.9. Cloud Computing

Este término se puede definir como el uso de una red de servidores remotos conectados a internet para almacenar, administrar y procesar datos, servidores, bases de datos, redes y software. En lugar de depender de un servicio físico instalado, se tiene acceso a una estructura donde el software y el hardware están virtualmente integrados [20]; también se le conoce como "la nube".

2.1.2.10. Pruebas de software

La etapa de pruebas de software se define como el *proceso de evaluación y verificación de que un producto o aplicación de software hace lo que se supone que debe hacer* [21]. Las pruebas son un componente esencial si se quiere garantizar una alta calidad en el software, ya que permiten detectar errores en entornos controlados, evitando riesgos con personas o infraestructura. Existen distintos tipos y niveles de pruebas de software, y cada una de ellas tiene diversos objetivos y estrategias. Algunas de las más destacadas son:

- **Niveles de prueba** [22]
  - **Pruebas de componente**. También conocidas como "pruebas unitarias", se utilizan para probar el funcionamiento de cada elemento del software de manera individual.
  - **Pruebas de integración**. Permiten verificar que la integración entre dos o más componentes del sistema se haga de manera adecuada. Además, se incluyen en esta categoría las pruebas que se hacen para verificar que el sistema interactúe correctamente con sistemas externos.

  - **Pruebas de sistema**. Tienen como objetivo probar el software de forma global para verificar que cumplan con los requisitos planteados.
  - **Pruebas de aceptación**. Son las que realiza el cliente para verificar que el software desarrollado es lo que esperaba y responde adecuadamente a sus necesidades.

- **Tipos de prueba** [23]

  - **Pruebas funcionales**. Verifican los requerimientos funcionales especificados al inicio del proyecto.
  - **Pruebas no funcionales**. Verifican los requerimientos no funcionales, tales como el desempeño, la usabilidad, etc.

2.1.2.11. Usabilidad

La usabilidad es un *atributo cualitativo definido comúnmente como la facilidad de uso, ya sea de una página web, una aplicación informática o cualquier otro sistema que interactúe con un usuario* [24]. También se puede entender como una medida de la calidad de la experiencia que tiene un usuario al interactuar con un producto o sistema. La usabilidad se define mediante 5 componentes de calidad [25]:

- **Aprendibilidad**. Se refiera a la facilidad con la que el usuario aprende sobre el funcionamiento y comportamiento del sistema, teniendo en cuenta que es la primera vez que interactúa con él.
- **Eficiencia**. Define el nivel de productividad que alcanza el usuario una vez que haya aprendido a usar el sistema, es decir, la rapidez con la que puede desarrollar las tareas teniendo conocimiento previo de su comportamiento.
- **Memorabilidad**. Mide la facilidad con que el usuario memoriza el funcionamiento del sistema, haciendo que la curva de aprendizaje entre la primera y la segunda vez que lo usó sea mínima.
- **Errores**. Consiste en conocer la capacidad del sistema para ofrecer una tasa baja de errores, haciendo que se cometan pocos errores durante el uso del sistema o ayudando a los usuarios a recuperarse fácilmente en caso de que cometan algún error.
- **Satisfacción**. Es la impresión subjetiva que tiene el usuario con respecto al sistema.

2.1.3. Trabajos Relacionados

Las empresas tradicionales, a nivel global, se están quedando atrás frente a otras que optaron por ofrecer sus productos, servicios o atención al cliente de manera virtual, o por las redes sociales, incrementando sus ventas [26]. Algunas pequeñas empresas se ayudan de contestadores automáticos al inicio de una conversación o con el uso de respuestas rápidas preestablecidas por el usuario. Sin

embargo, esta opción no está presente en todas las redes sociales, es por esto que algunas empresas (la mayoría con un gran flujo de conversaciones), optan por usar robots (chatbots) que contesten las preguntas que se le hagan, cuando estén en su repertorio de conocimiento. Algunos son capaces de responder usando inteligencia artificial [27], como es el caso del banco HSBC, el cual desarrolló un chatbot para responder de manera dinámica a preguntas de sus clientes sobre sus políticas [28].

Sin embargo, los chatbots a veces se quedan cortos a la hora de responder, ya que mantener una conversación coherente con un humano es algo complejo, como lo planteaba Alan Turing [29]. Por esta razón, muchas empresas deciden asignar las conversaciones a unos agentes o personas, si ven que lo solicitado por el usuario o potencial cliente no lo puede responder el chatbot. Algunas empresas, como Google, Uber o Adobe [30], manejan herramientas como B2Chat [31] o MessageBird [30], las cuales permiten crear campañas de marketing, además de juntar múltiples canales de comunicación en uno solo. Este es el caso de empresas como Domino's Pizza, quienes han incrementado sus ventas en un 40 % [32]; o Houm, quienes pudieron atraer un 64 % más de clientes potenciales y reducir el tiempo de respuesta a sus clientes [33] gracias a herramientas como Piyion o MessageBird.

Incrementar la cantidad de atenciones a clientes va a potenciar el número de ventas. A su vez, la facilidad y rapidez de los agentes para responder mensajes, aumentará la cantidad de personas atendidas. Es por esto que empresas como Callbell se preocupan porque sus usuarios tengan la manera de contestar los mensajes de su plataforma desde sus dispositivos móviles, dejándoles la opción "al alcance del bolsillo" [34]. Y es que, evidentemente, un teléfono inteligente se usa, en promedio, poco menos de 4 horas al día [35]. Hay empresas que, debido a la pandemia del SARS-CoV-2, se vieron forzadas a implementar el teletrabajo. Esta modalidad de trabajo le permite a los empleados tener una mayor flexibilidad. Trabajar de manera remota, poder recibir notificaciones de todo y con la posibilidad de responder chats desde el celular resulta increíble para alguien que haga parte del equipo de asesores de una empresa que use este tipo de plataformas para la atención al cliente [36] [37].

Es por esto que Piyion ha estado desarrollando una aplicación web [4], en donde le permiten a empresas comunicarse con sus clientes a través de diferentes canales digitales. Dentro de esta plataforma se permite la gestión y supervisión de agentes, la configuración de canales digitales, la visualización de estadísticas relacionadas a la atención de la empresa o de los agentes de la empresa, entre otras cosas. Piyion web sabe que sus clientes, en su mayoría, usan la modalidad de teletrabajo. Esta modalidad de trabajo le permite a los empleados tener una mayor flexibilidad en cuanto a horarios y ubicación [38]. Además, las aplicaciones móviles para la atención al cliente, son una herramienta clave para los agentes que trabajan desde casa. Al tener la aplicación al alcance del bolsillo, los agentes pueden responder rápidamente a los mensajes de los clientes, ya sea desde su casa, mientras se trasladan de un lugar a otro, etc. Esto les permite trabajar de manera más eficiente y tener una mejor gestión del tiempo, lo que mejora su calidad de vida y disminuye el estrés asociado con el trabajo [39]. Además, la posibilidad de responder chats desde el celular les da a los agentes la libertad de trabajar desde cualquier lugar, lo que les permite estar más conectados con los clientes y, en última instancia, mejorar la calidad de la atención al cliente. Por estas razones,

Piyion requiere a una aplicación móvil propia, para poder brindarle su producto a sus clientes, al alcance del bolsillo.

2.2. Metodología de desarrollo

La metodología que se usó para el desarrollo del proyecto fue Scrum. Se eligió debido a la sencilla forma en que se planean y organizan las actividades que se van a realizar, además de fomentar el trabajo en equipo y ser capaz de adaptarse a las necesidades del proyecto a lo largo del tiempo. Otra de las razones por las cuales se eligió Scrum es la flexibilidad que tiene con respecto a los cambios que se deban realizar en algún momento del proyecto, ya que permite que el proceso de desarrollo se haga enfocado individualmente en las funcionalidades del sistema y no de manera conjunta. Se compone de tres partes esenciales [40] que se explican a continuación.

2.2.1. Equipo

Se refiere al grupo de todas las personas que hacen parte del proyecto. Este es auto-organizado y multi-funcional. Debido a que se trabajó en conjunto con la empresa Piyion, los roles del equipo se establecieron de la siguiente manera:

- **Product Owner**

  Es el responsable de maximizar el valor del producto siendo el representante del cliente en el equipo. En este caso fue un miembro de Piyion, que colaboró con la revisión de la aplicación en términos de producto.

- **Scrum Master**

  Es el responsable de promover y apoyar Scrum para la realización de las actividades, siendo un guía experto en la metodología que verifica que esta se aplique y sea útil para todos los involucrados. En este caso fue representado por un miembro de Piyion.

- **Development Team**

  Son los profesionales que realizan el trabajo de entregar un incremento del producto cada cierto tiempo. En este caso fue integrado por ambos miembros del equipo del trabajo de grado y algunos miembros de Piyion que colaboraron en algunos aspectos como el diseño UI de la aplicación y temas relacionados al desarrollo backend.

2.2.2. Eventos

Son definidos con el fin de crear cierto tipo de regularidad y organización, además de minimizar las reuniones innecesarias. Debido a las limitaciones de tiempo para el desarrollo de la aplicación sólo se tomaron algunos de ellos:

- **Sprint**

  Es la base de Scrum, que es donde se crea un incremento del producto. Su duración puede ser hasta de máximo un mes. En este caso la duración fue de dos semanas, aproximadamente.

- **Sprint Planning**

  Es una reunión para definir qué se hará durante el sprint. En este caso, se definían las actividades de la siguiente iteración dependiendo de su prioridad y los alcances para el trabajo de grado.

- **Daily Stand-up**

  Es una reunión diaria donde cada miembro del equipo de desarrollo comenta sus avances respecto del día anterior, si ha tenido dificultades, y lo que pretende hacer después. En este caso se llevó a cabo aproximadamente dos veces a la semana por medio de mensajes de texto o reuniones.

- **Sprint Review**

  Es una reunión donde se muestra el incremento desarrollado durante el sprint. En este caso se realizaba en la misma reunión del Sprint Planning para cerrar el sprint anterior y planear el siguiente.

2.2.3. Artefactos

Representan trabajo o valor, que son útiles para que todo el equipo de Scrum pueda conocer los verdaderos avances del proyecto. Para el manejo de estos artefactos y en general de toda la planeación con Scrum se usó la herramienta Shortcut. Estos son:

- **Product Backlog**

  Es una lista de todo lo que se conoce que es necesario en el producto. Es administrado por el Product Owner pero es visible para cualquier miembro del equipo. Debido a que se trabajó en conjunto con la empresa Piyion, la organización de las actividades relacionadas con el desarrollo se acordaban entre ambas partes (la empresa y el equipo de trabajo de grado).

  Teniendo en cuenta lo anterior, se decidió crear una “épica” (grupo de historias de usuario) llamada “Piyion - App” para manejar todo lo relacionado al desarrollo de la aplicación mòvil. En la Figura 2.2 se muestran algunas de las historias de usuario asociadas a dicha épica.

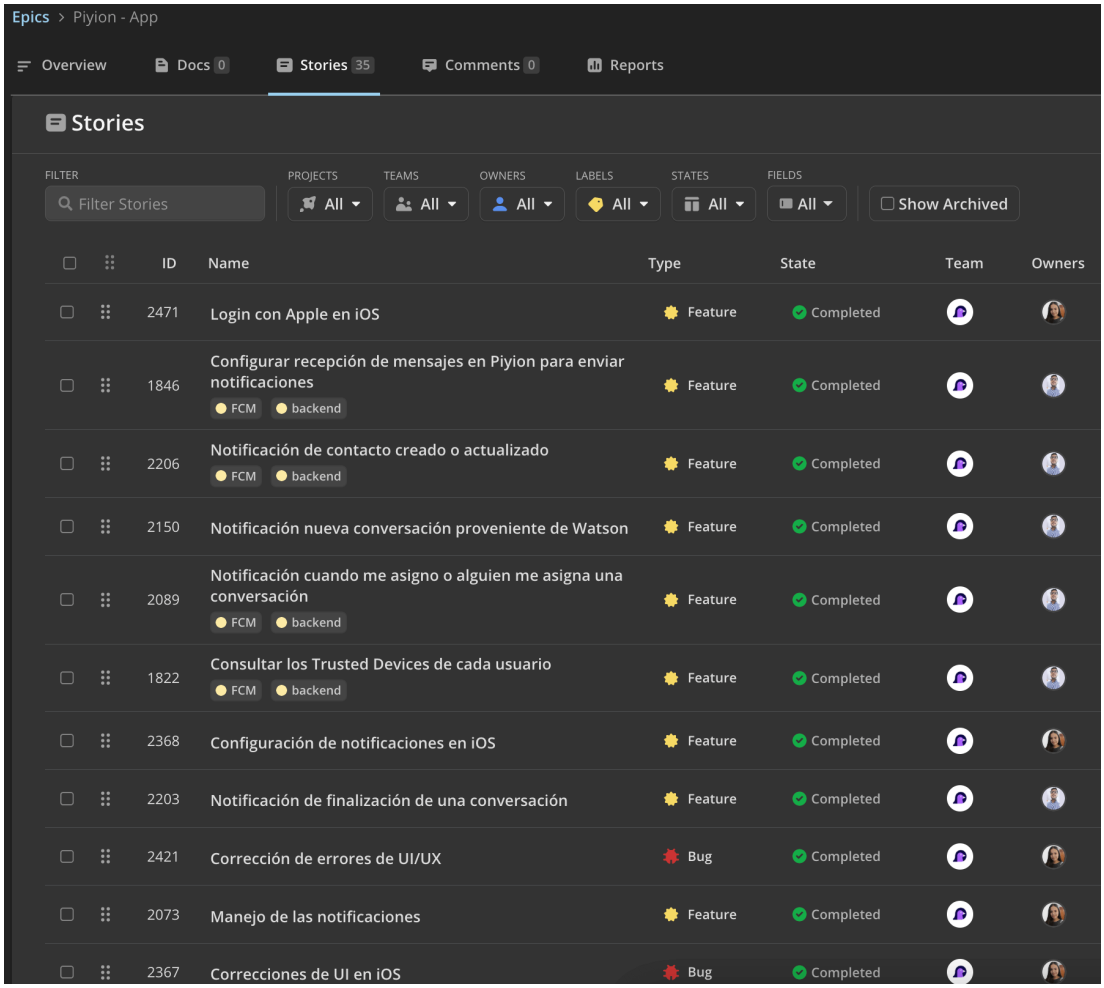


Figura 2.2: Algunas historias de usuario de la épica “Piyion - App”

En la Figura 2.3 se muestra un ejemplo de una historia de usuario. En este caso, se trata sobre la historia de usuario que consiste en implementar la interfaz gráfica donde es posible iniciar sesión en el sistema. Además, se observa que posee un título junto con una descripción apropiada, el estado de la tarea como “Completed”, la épica anteriormente mencionada, el sprint en la que se desarrolla, entre otros aspectos.

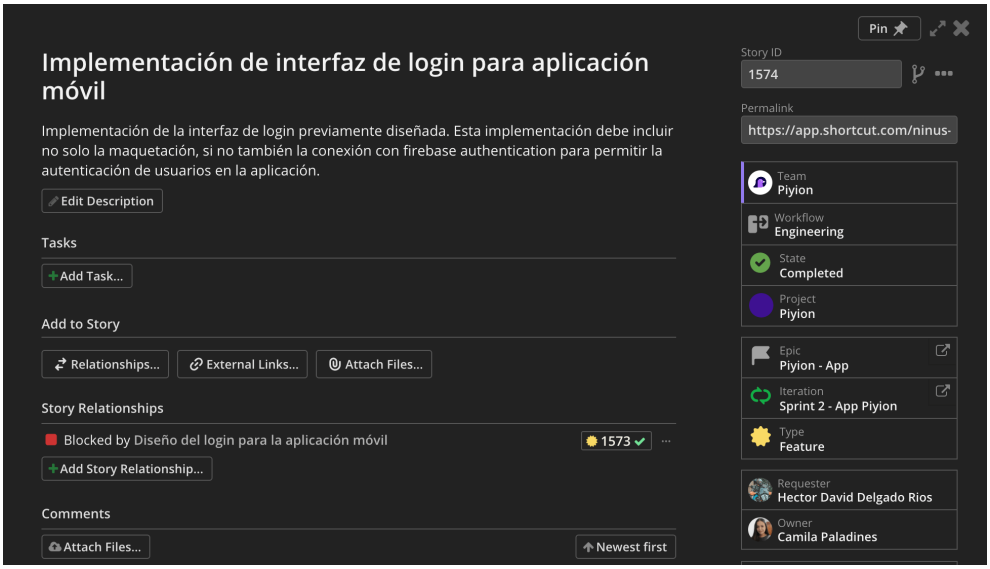


Figura 2.3: Ejemplo de historia de usuario

- **Sprint Backlog**

  Es un subconjunto del Product Backlog que contiene los elementos que se pretenden trabajar durante un sprint en específico. En la Figura 2.4 se muestra un ejemplo, en este caso del Sprint 1. En él se puede ver que las primeras actividades se relacionaron con el diseño UI y la inicialización del proyecto.

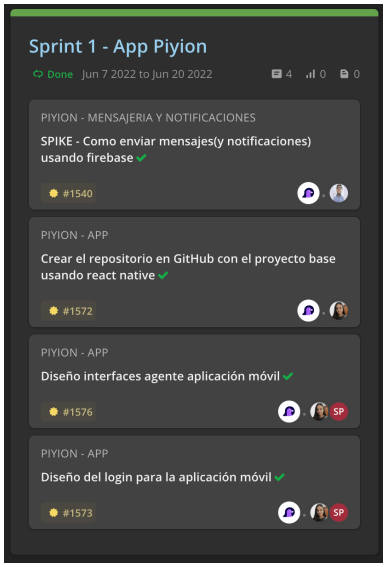


Figura 2.4: Ejemplo de sprint

Capítulo 3

Requisitos

La etapa de obtención de requisitos en un proyecto de software es de suma importancia debido a que es el momento donde se define qué es lo que se va a desarrollar. Los requerimientos permiten un entendimiento acordado entre las partes interesadas (como los usuarios, clientes, operadores, proveedores, etc) sobre necesidades que pueden ser resueltas mediante un software [41]. Además, estos proporcionan una referencia para verificar los posteriores diseños y soluciones. Se pueden destacar dos tipos de requisitos [42]:

- **Requisitos funcionales**

  Describen las funciones o tareas que debe realizar el sistema. En este sentido, son los que definen el comportamiento esperado de la solución que se va a desarrollar, tomando en cuenta los distintos escenarios en los que se pueda estar envuelto.

  Para su definición se puede usar la estructura COMO-QUIERO-PARA con el fin de establecer las necesidades del usuario desde su perspectiva. En ella se destaca:

  - COMO, que indica quién es el actor que requiere dicha funcionalidad.
  - QUIERO, que establece cuál es la funcionalidad esperada.
  - PARA, que explica las razones por las cuales se pide el desarrollo de la funcionalidad.

- **Requisitos no funcionales**

  Son también llamados "requisitos de calidad" y definen las características del software que se busca, como su nombre lo indica, con una alta calidad en el producto final. Algunos de ellos son, la mantenibilidad, la reutilización, la seguridad, la usabilidad, entre otros.

3.1. Requisitos funcionales

En primer lugar se construyó un diagrama de casos de uso para comprender mejor las necesidades generales del sistema (ver Figura 3.1). En él se puede observar la distinción entre los dos roles que puede tener un usuario en Piyion: agente y administrador. El usuario agente puede llevar a cabo varios casos de uso básicos como chatear, asignarse conversaciones a sí mismo, gestionar contactos de conversaciones, etc. Por otro lado, el usuario administrador, además de poder realizar las mismas acciones que el usuario agente, tiene opciones como supervisar el equipo, cambiar la información de la compañía, entre otros.

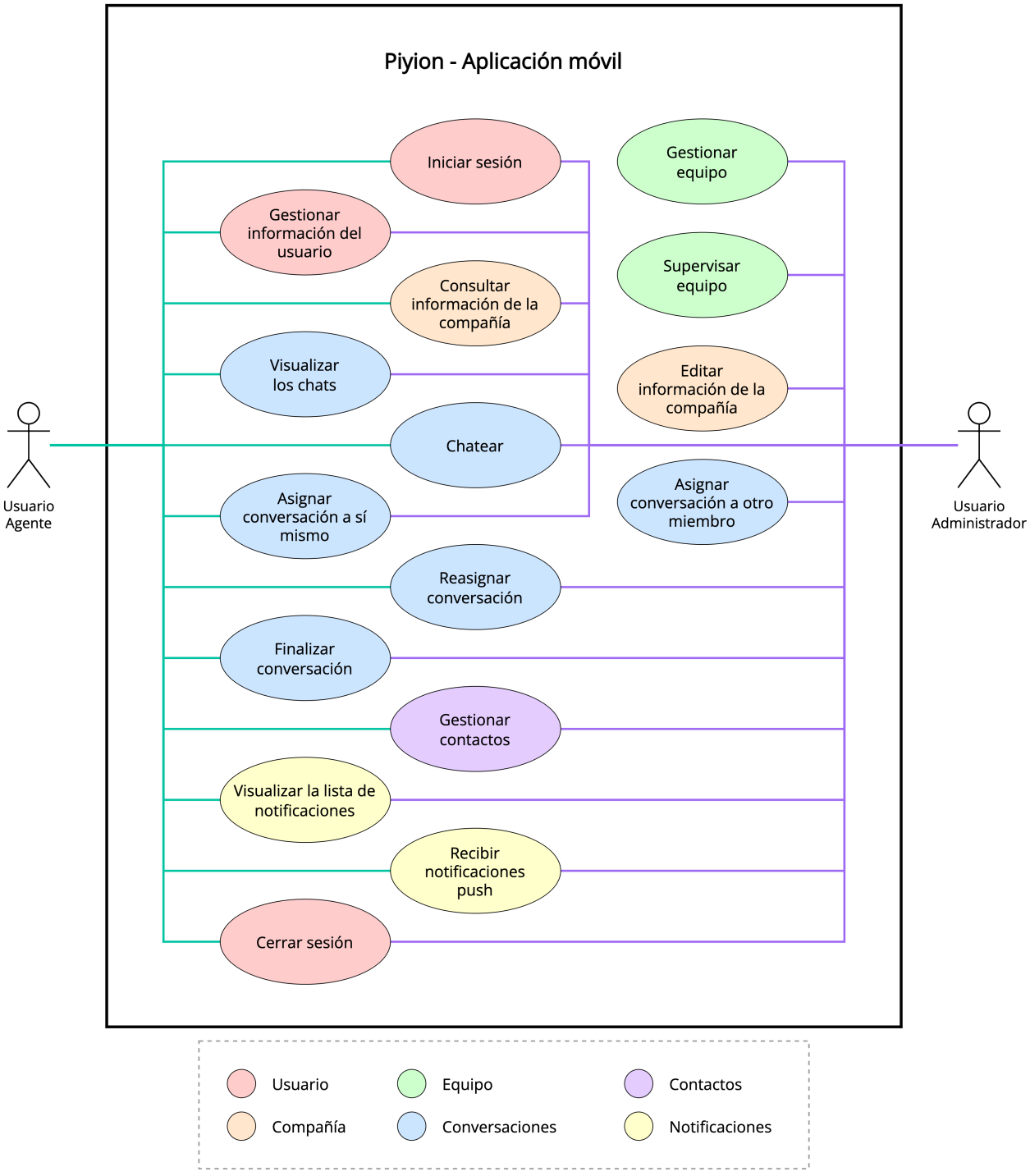


Figura 3.1: Diagrama de casos de uso del sistema

Debido a que ya se contaba con una versión web del sistema, la especificación de los requisitos para la aplicación móvil se basó en las funcionalidades presentes en la aplicación web, en este caso, adaptadas a un dispositivo móvil. Cabe resaltar, además, que debido a los límites de tiempo definidos para el desarrollo de este proyecto, sólo se tuvo en cuenta los aspectos básicos y suficientes para considerar una primera versión de la aplicación.

Los requisitos funcionales del sistema se dividen en distintas categorías. En la Figura 3.2 se muestran los requisitos correspondientes a la gestión de usuarios, como el manejo de la sesión y la modificación de sus datos personales.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
USU-01	Inicio de sesión con correo electrónico y contraseña	**COMO** usuario **QUIERO** poder iniciar sesión con mi correo electrónico y contraseña. **PARA** ingresar a la aplicación.	Alta	Ninguna	* El usuario puede ingresar a la aplicación usando su correo electrónico y contraseña. * Se muestran errores de correo no registrado o contraseña incorrecta en los casos correspondientes.
USU-02	Inicio de sesión con Google	**COMO** usuario **QUIERO** poder iniciar sesión con mi cuenta de Google **PARA** ingresar a la aplicación.	Alta	Ninguna	* El usuario puede ingresar a la aplicación usando su cuenta de Google. * Se muestra un error en el caso de que no se logró iniciar sesión.
USU-03	Cambio de disponibilidad	**COMO** usuario **QUIERO** poder ver y cambiar mi disponibilidad **PARA** mantener actualizado mi estado en el sistema.	Media	USU-01 USU-02	* El usuario puede ver claramente si está disponible o no para brindar atención. * El usuario puede cambiar fácilmente su disponibilidad.
USU-04	Visualización de la información del usuario	**COMO** usuario **QUIERO** poder ver mis datos personales como mi nombre y número de celular **PARA** verificar la correctitud de dicha información.	Media	USU-01 USU-02	* El usuario puede ver su información personal, como su nombre, apellido y número de celular.
USU-05	Edición de la información del usuario	**COMO** usuario **QUIERO** poder editar mis datos personales como mi nombre y número de celular **PARA** mantener actualizada mi información.	Media	USU-04	* El usuario puede editar su información personal, como su nombre, apellido y número de celular.
USU-06	Salida del sistema	**COMO** usuario **QUIERO** poder cerrar sesión **PARA** salir del sistema.	Media	USU-01 USU-02	* El usuario puede salir del sistema y ser redirigido a la pantalla de inicio de sesión.

Figura 3.2: Requisitos sobre la gestión de usuarios

En la Figura 3.3 se muestran los requisitos que corresponden a la gestión de los datos de la compañía.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
COM-01	Visualización de la información de la compañía	**COMO** usuario **QUIERO** poder ver los datos de la compañía **PARA** verificar la correctitud de dicha información.	Media	USU-01 USU-02	* El usuario puede ver la información de la compañía, como su nombre, país, página web, etc.
COM-02	Edición de la información de la compañía	**COMO** usuario con el rol de administrador **QUIERO** poder editar los datos de la compañía **PARA** mantener actualizada su información.	Media	COM-01	* El usuario puede editar la información de la compañía, como su nombre, país, página web, etc.

Figura 3.3: Requisitos sobre la gestión de la compañía

En la Figura 3.4 se pueden observar los requisitos para el manejo del equipo dentro de la compañía.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
EQU-01	Visualización de una lista con los miembros del equipo	**COMO** usuario con rol de administrador **QUIERO** poder ver la lista de usuarios que son miembros de la compañía, tanto los agentes como los administradores. **PARA** conocer a las personas que brindarán la atención al cliente.	Media	USU-01 USU-02	* El usuario puede ver la lista de los agentes, mostrando principalmente su nombre. * Se cuenta con un buscador donde se puede buscar a un usuario por sus nombres o apellidos.
EQU-02	Visualización de los datos de un miembro del equipo en específico	**COMO** usuario con rol de administrador **QUIERO** poder ver la información asociada a un miembro del equipo en específico, como su correo electrónico, rol, calificación, entre otros **PARA** conocer los datos de dicho usuario.	Media	EQU-01	* El usuario puede ver la información personal de un miembro del equipo, como su correo electrónico, rol, número de conversaciones asignadas, calificación, etc.
EQU-03	Supervisión del equipo	**COMO** usuario con rol de administrador **QUIERO** poder supervisar a los miembros del equipo para observar las conversaciones en las que están atendiendo **PARA** mantenerme al tanto de la atención que brindan dichos agentes.	Alta	EQU-01	* El usuario puede ver una lista de los miembros del equipo y puede presionar sobre uno de ellos para ver la lista de chats que tiene asignados. * El usuario puede entrar a ver la conversación en tiempo real que está teniendo el usuario supervisado.

Figura 3.4: Requisitos sobre la gestión del equipo

En la Figura 3.5 se muestran los requisitos relacionados a las conversaciones y chats, que son el componente más importante del sistema.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
CHA-01	Visualización de los chats asignados	**COMO** usuario **QUIERO** poder ver la lista de chats que tengo asignados **PARA** tener presente las conversaciones que están siendo atendidas.	Alta	USU-01 USU-02	* El usuario puede ver la lista de chats asignados con una estructura similar a la que tiene What'sApp, en la que se muestra el último mensaje (enviado o recibido), el tiempo que ha pasado desde el último mensaje, etc.
CHA-02	Visualización de los chats no asignados	**COMO** usuario **QUIERO** poder ver la lista de chatd no asignados **PARA** tener presente las conversaciones que requieren atención.	Alta	USU-01 USU-02	* El usuario puede ver la lista de chats que aún no han sido asignados a un asesor, mostrando principalmente el canal de donde vienen y el tiempo que ha pasado desde el último mensaje.
CHA-03	Visualización de una conversación	**COMO** usuario **QUIERO** poder ver los mensajes de una conversación **PARA** conocer el estado de la atención.	Alta	CHA-01	* El usuario puede presionar en una de las tarjetas de chat y abrir la conversación para ver los mensajes. * La interfaz debe ser similar a la de una conversación de What'sApp, mostrando los mensajes enviados o recibidos, diferenciando entre los tipos de mensaje (texto, imagen, PDF, etc), además de la hora y fecha de envío.
CHA-04	Envío de mensajes	**COMO** usuario **QUIERO** poder enviar un mensaje **PARA** brindar atención al cliente.	Alta	CHA-03	* El usuario puede enviar un mensaje de texto, de imagen o de documento PDF. * El usuario puede ver que se actualiza la lista de mensajes con el nuevo mensaje enviado.
CHA-05	Asignación de una conversación a sí mismo	**COMO** usuario **QUIERO** poder asignar una conversación a mí mismo **PARA** atender la solicitud del cliente.	Alta	CHA-02	* El usuario puede asignarse una conversación que aún esté pendiente, y abrir el chat en cuanto ejecute dicha acción.
CHA-06	Asignación de una conversación a otro miembro del equipo	**COMO** usuario con rol de administrador **QUIERO** poder asignar una conversación que no ha sido atendida a un miembro del equipo disponible **PARA** agilizar el manejo de las conversaciones y los miembros del equipo	Alta	CHA-02	* El usuario puede asignar una conversación pendiente a uno de los miembros del equipo que se encuentren disponibles.
CHA-07	Reasignación de una conversación	**COMO** usuario **QUIERO** poder reasignar una conversación a otro miembro del equipo **PARA** que la atención sea flexible en caso de requerir dichos cambios.	Alta	CHA-01	* El usuario puede reasignar una conversación a otro miembro del equipo que se encuentre disponible. * La acción de reasignar consiste en cambiar el agente asignado para una conversación específica.
CHA-08	Finalización de una conversación	**COMO** usuario **QUIERO** poder finalizar una conversación cuando lo considere necesario **PARA** dar por terminada la atención.	Alta	CHA-03	* El usuario puede finalizar una conversación en específico cuando lo considere necesario, lo que hace que la conversación ya no se muestre para el usuario y se inicie un proceso de calificación por parte del cliente atendido.

Figura 3.5: Requisitos sobre la gestión de las conversaciones

En la Figura 3.6 se observan los requisitos para el manejo de contactos.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
CON-01	Visualización de la información de contacto de un chat	**COMO** usuario **QUIERO** poder ver la información relacionada a un cliente en una conversación **PARA** conocer los datos asociados a él.	Alta	CHA-03	* El usuario puede ver la información de contacto del cliente que se encuentra recibiendo la atención, como su nombre, correo electrónico, número de celular, canales por los cuales se ha comunicado, entre otros.
CON-02	Creación de un contacto	**COMO** usuario **QUIERO** poder crear un contacto asociado a un cliente en una conversación **PARA** hacer seguimiento efectivo y mejorar la atención.	Alta	CHA-03	* El usuario puede guardar por primera vez la información de un cliente, como su nombre, número de celular, correo electrónico, entre otros. * El usuario puede editar la información de contacto de un cliente previamente guardado.
CON-03	Edición de la información de un contacto	**COMO** usuario **QUIERO** poder editar la información de contacto de un cliente **PARA** hacer seguimiento efectivo y mejorar la atención.	Alta	CON-02	* El usuario puede guardar por primera vez la información de un cliente, como su nombre, número de celular, correo electrónico, entre otros. * El usuario puede editar la información de contacto de un cliente previamente guardado.

Figura 3.6: Requisitos sobre la gestión de los contactos

En la Figura 3.7 se presentan los requisitos que corresponden a las notificaciones, tanto las que se muestran dentro de la app como las notificaciones push.

ID	Título	Descripción	Importancia	Dependencias	Criterios de aceptación
NOT-01	Visualización del panel de notificaciones	**COMO** usuario **QUIERO** poder ver todas las notificaciones **PARA** conocer las actualizaciones de la información	Media	USU-01 USU-02	* El usuario puede ver la lista de notificaciones de la aplicación, donde informen si hay conversaciones nuevas sin atender, nuevos mensajes en conversaciones asignadas, conversaciones reasignadas, entre otros.
NOT-02	Notificaciones push de nuevos mensajes	**COMO** usuario **QUIERO** poder recibir notificaciones push cuando hay nuevos mensajes **PARA** conocer los nuevos mensajes	Media	USU-01 USU-02	* El usuario puede recibir notificaciones push, donde se informe sobre mensajes nuevos.
NOT-03	Notificaciones push de nuevas conversaciones	**COMO** usuario **QUIERO** poder recibir notificaciones push cuando hay nuevas conversaciones para ser atendidas **PARA** saber si hay nuevos clientes pendientes de atención	Media	USU-01 USU-02	* El usuario puede recibir notificaciones push, donde se informe sobre conversaciones nuevas.
NOT-04	Notificaciones push de conversaciones asignadas a mí	**COMO** usuario **QUIERO** poder recibir notificaciones push cuando me han asignado una conversación **PARA** saber si tengo que atender una conversación	Media	USU-01 USU-02	* El usuario puede recibir notificaciones push, donde se informe si le han asignado una conversación.
NOT-05	Notificaciones push de conversaciones reasignadas a mí	**COMO** usuario **QUIERO** poder recibir notificaciones push cuando me han reasignado una conversación **PARA** saber si tengo que atender una conversación	Media	USU-01 USU-02	* El usuario puede recibir notificaciones push, donde se informe si le han reasignado una conversación.

Figura 3.7: Requisitos sobre las notificaciones

3.2. Requisitos no funcionales

Los requisitos no funcionales definidos para la aplicación móvil se presentan en la Figura 3.8. Cabe resaltar que ante las limitaciones de tiempo se dará prioridad a los requisitos de alta importancia, dejando los demás como trabajo futuro.

ID	Título	Descripción	Importancia	Dependencias
RNF-01	Rendimiento	La aplicación debe tener un tiempo de respuesta promedio máximo de 5 segundos al ejecutar una tarea.	Media	Ninguna
RNF-02	Escalabilidad	La aplicacion debe ser capaz de soportar 100 conversaciones simultaneas para una compañía.	Media	Ninguna
RNF-03	Usabilidad	La aplicación debe tener una tasa de satisfacción del usuario del 80% o superior en las encuestas de usabilidad.	Alta	Ninguna
RNF-04	Mantenibilidad	La aplicación se debe desarrollar usando las prácticas recomendadas de las tecnologías usadas para garantizar la legibilidad del código y la prevención de los errores.	Alta	Ninguna
RNF-05	Disponibilidad	La aplicación debe estar disponible el 97% del tiempo en un periodo de una semana.	Media	Ninguna

Figura 3.8: Requisitos no funcionales del sistema

Capítulo 4

Diseño

Ya habiendo definido los requisitos del software en cuestión, el paso siguiente es realizar un bosquejo o diseño de lo que se va a implementar antes de llevarlo al código. En este caso, se tienen en cuenta dos partes del diseño:

- **Diseño UI/UX**

  Consiste en la planeación de las interfaces gráficas del software a desarrollar y el flujo que estas van a tener dentro de la lógica del sistema [43]. En este sentido, se tienen en cuenta los elementos visuales y su comportamiento teniendo en cuenta los requisitos definidos, tanto en el aspecto funcional como en los que se refieren a la calidad (la usabilidad, por ejemplo).

  El diseño UI/UX de una aplicación en cualquier plataforma es un paso fundamental para que su construcción se haga de manera correcta, puesto que brinda un primer acercamiento de lo que se va a implementar y permite tener retroalimentación del cliente antes de empezar a desarrollar las primeras pantallas.

  Los diseños de la aplicación en este proyecto fueron realizados por Santiago Páez (un miembro de Piyion) y fueron aprobados por el Product Owner para su implementación.

- **Arquitectura**

  La arquitectura de software representa las decisiones de diseño relacionadas con la estructura y el comportamiento global del sistema [44]. Este se define de acuerdo con la lógica de negocio, la información que se va a manejar, etc. Además, debe tenerse en cuenta los requisitos establecidos al inicio del proyecto.

  Uno de los objetivos fundamentales de definir una arquitectura es establecer las partes que conforman el sistema y la manera que se relacionan entre sí, teniendo en cuenta los requerimientos y las herramientas disponibles para la construcción de un software que responda a dichas necesidades. De manera general, se busca que la arquitectura sea escalable y que tenga en cuenta la seguridad de la información.

4.1. Diseño UI/UX

En la Figura 4.1 se muestra la pantalla de inicio de sesión, donde se puede observar las diferentes opciones disponibles, como el ingreso mediante correo electrónico y contraseña, o el ingreso usando una cuenta de Google. Además se encuentran los botones que permiten al usuario registrarse o recuperar la contraseña en caso de haberla olvidado.

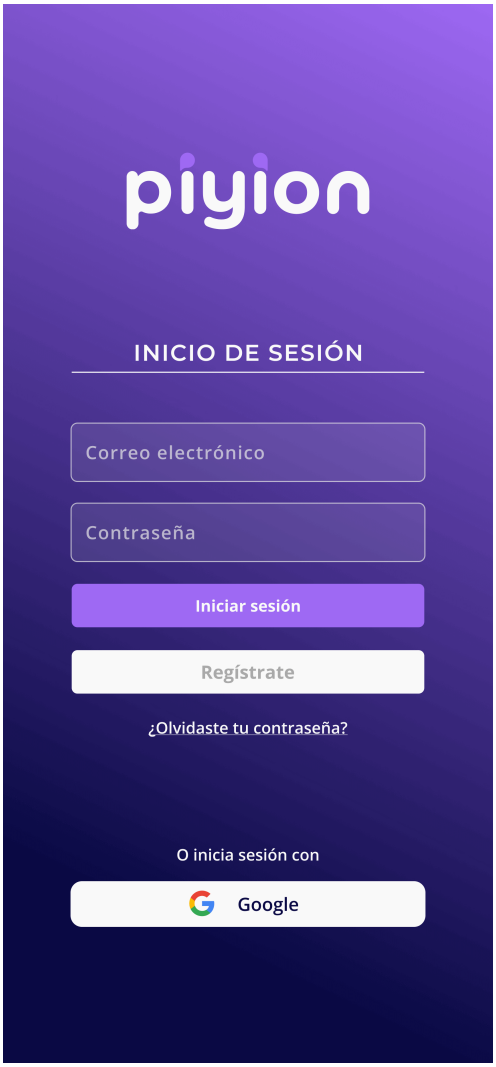


Figura 4.1: Diseño de la pantalla de inicio de sesión

En la Figura 4.2 se observan las pantallas de la sección de “Equipo”. La primera es una lista de todos las personas que brindan la atención al cliente, mostrando su disponibilidad (el círculo verde o rojo) nombre completo y rol dentro del sistema. Al seleccionar alguno de los items de la lista se navega a la segunda pantalla, indicando algunos datos adicionales del usuario seleccionado, como su correo electrónico, número máximo de conversaciones que se le pueden asignar, número de conversaciones que tiene asignadas actualmente, calificación, entre otros.

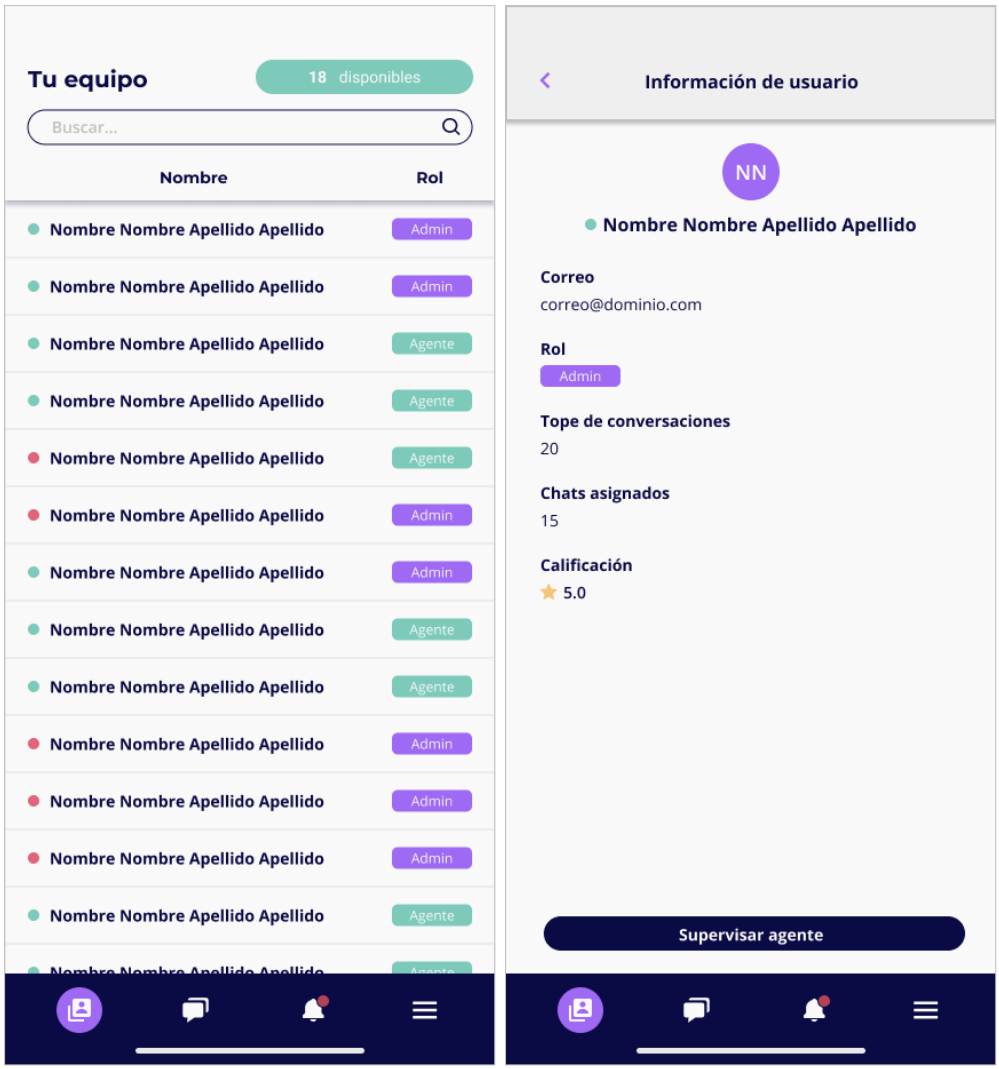


Figura 4.2: Diseño de las pantallas de la gestión del equipo

En la Figura 4.3 se muestran las pantallas de la sección de “Chats”. En la primera se encuentra una lista de las conversaciones que están asignadas al usuario actual y en la segunda las que aún están pendientes de ser asignadas. En ambos casos es posible destacar el canal de donde es requerida la atención y el tiempo transcurrido desde el envío del último mensaje. Además, al seleccionar alguno de los chats de la lista (primera pantalla) se navega a la tercera pantalla, en donde se encuentran los mensajes.

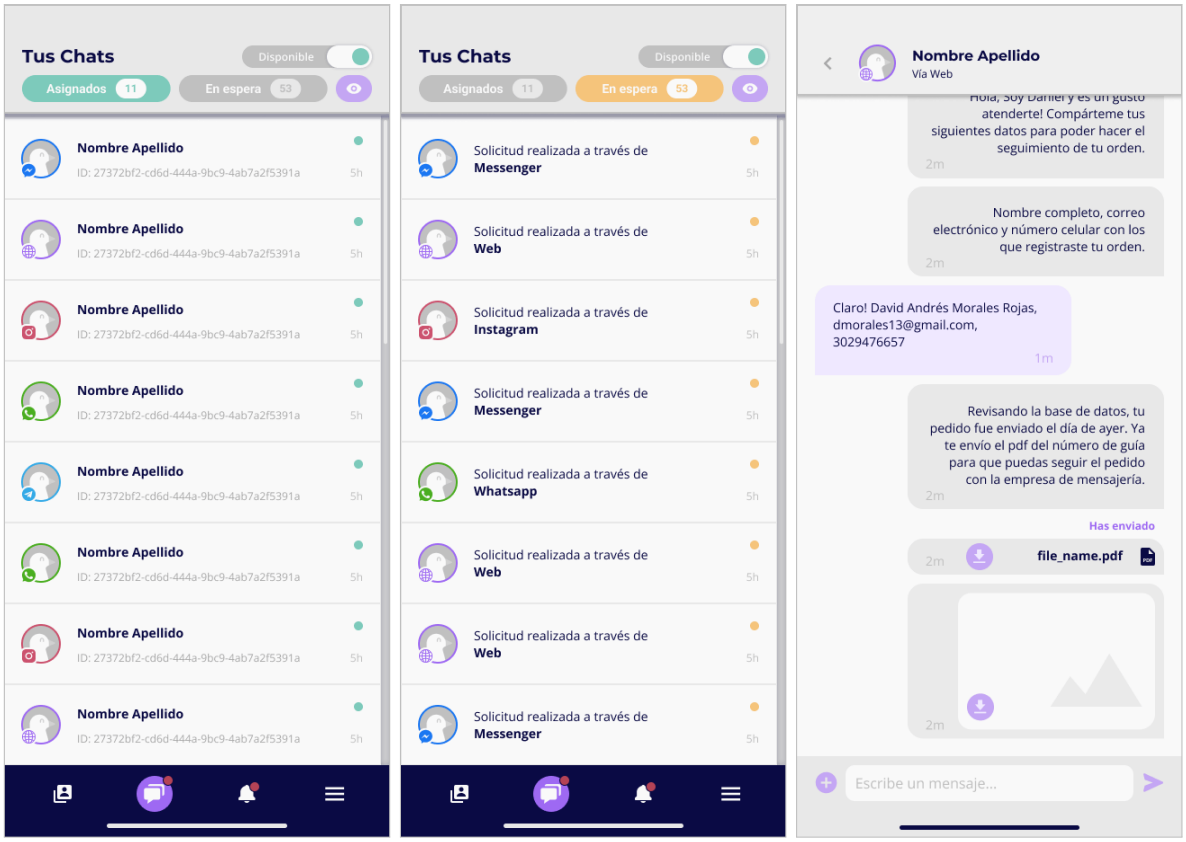


Figura 4.3: Diseño de las pantallas de listas de chats y conversación

En la Figura 4.4 se muestran la vistas de la supervisión del equipo. En la primera pantalla se encuentra una lista de todos los miembros del equipo, donde al presionar en cualquiera de ellos se navega a la segunda pantalla, que permite ver todas las conversaciones asignadas a dicho usuario. Si se selecciona alguno de los chats de la lista se va a la tercera pantalla, que muestra la conversación, esta vez sin tener la posibilidad de enviar un mensaje.

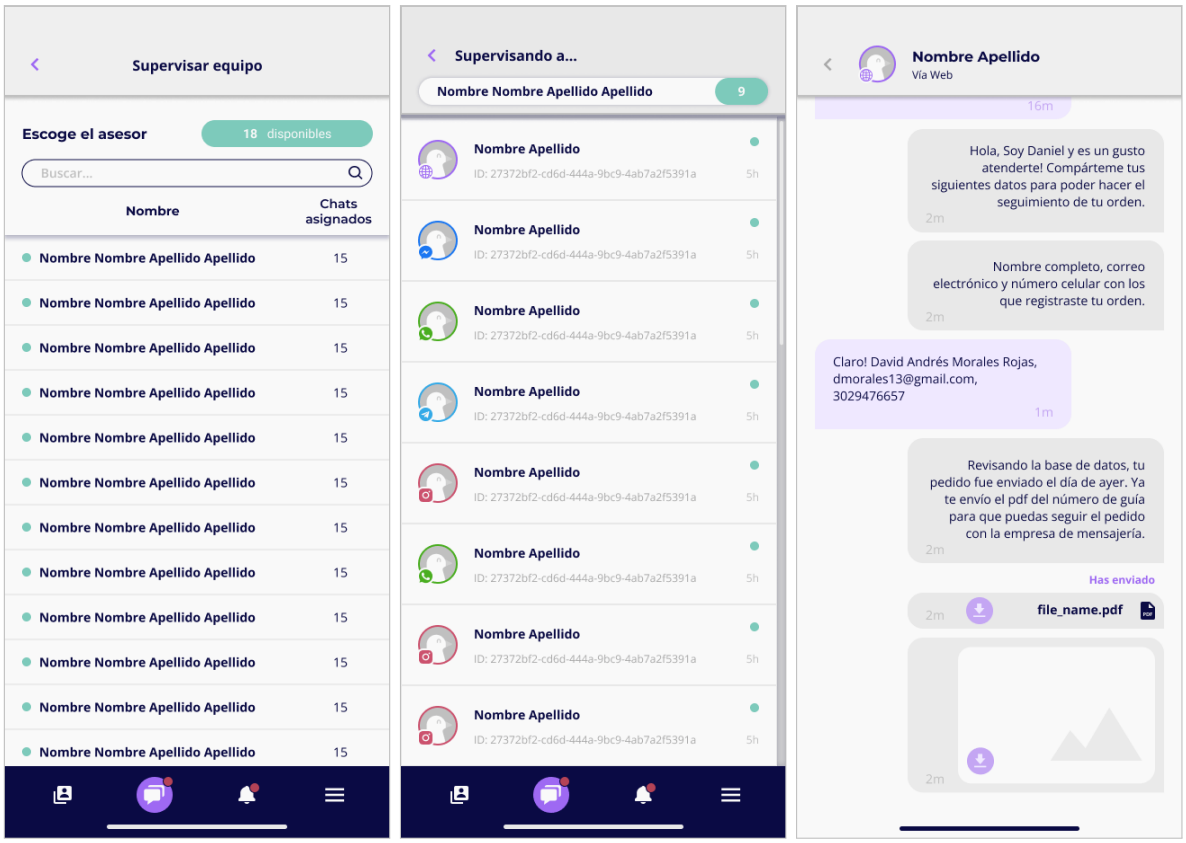


Figura 4.4: Diseño de las pantallas de supervisión

En la Figura 4.5 se muestran las interfaces que permiten visualizar y guardar respectivamente los datos de la persona a la que se le está brindando la atención. Entre estos datos se encuentra el nombre, correo electrónico, número de celular, etc. Además, desde la primera pantalla es posible realizar más acciones relacionadas con el cliente, como reasignarlo a otro miembro del equipo o finalizar la solicitud de atención.

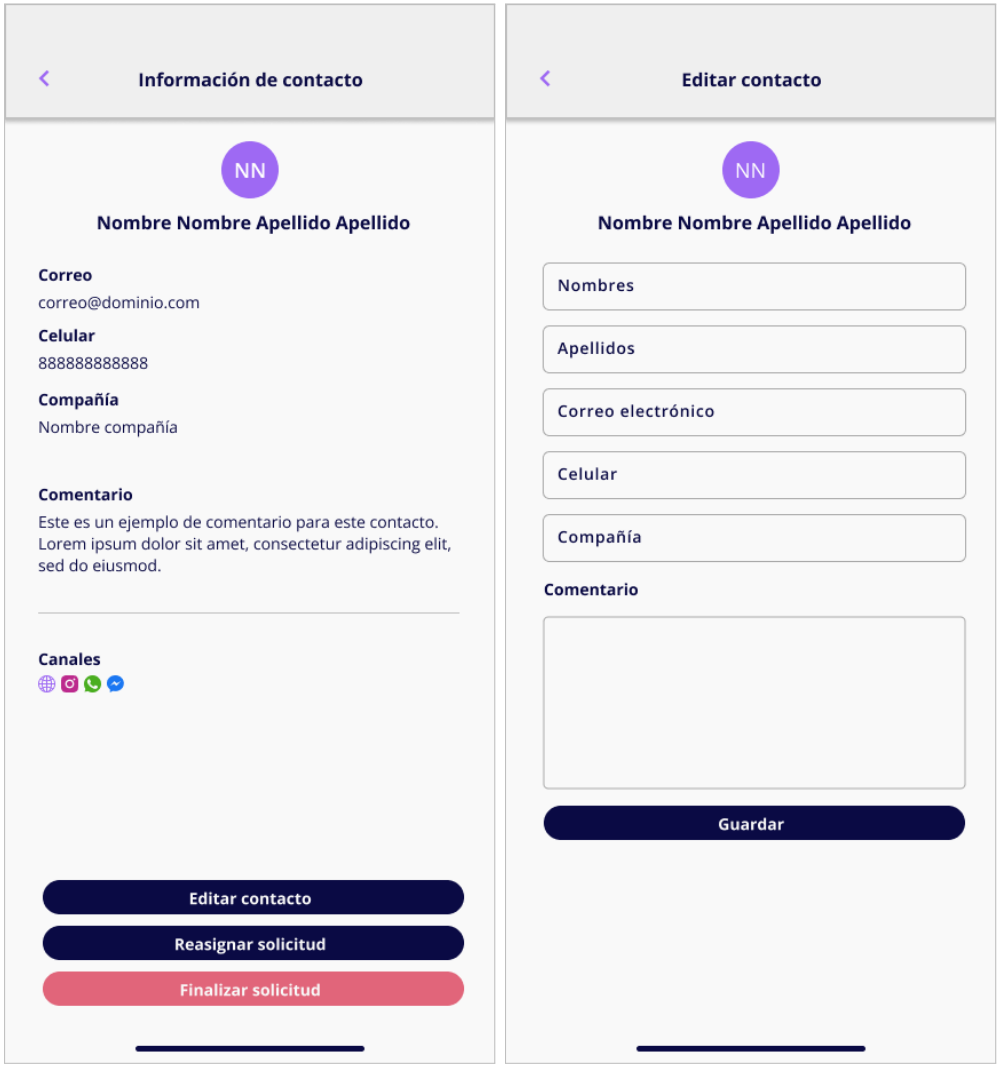


Figura 4.5: Diseño de las pantallas de información de contacto

En la Figura 4.6 se muestran las pantallas de asignar y reasignar una conversación. Cuando el usuario tiene el rol de administrador tiene la posibilidad de asignarse la conversación así mismo o asignarlo a otro miembro del equipo (primera y segunda pantalla). Por otro lado, la reasignación la puede realizar cualquier usuario sin importar su rol (tercera pantalla).

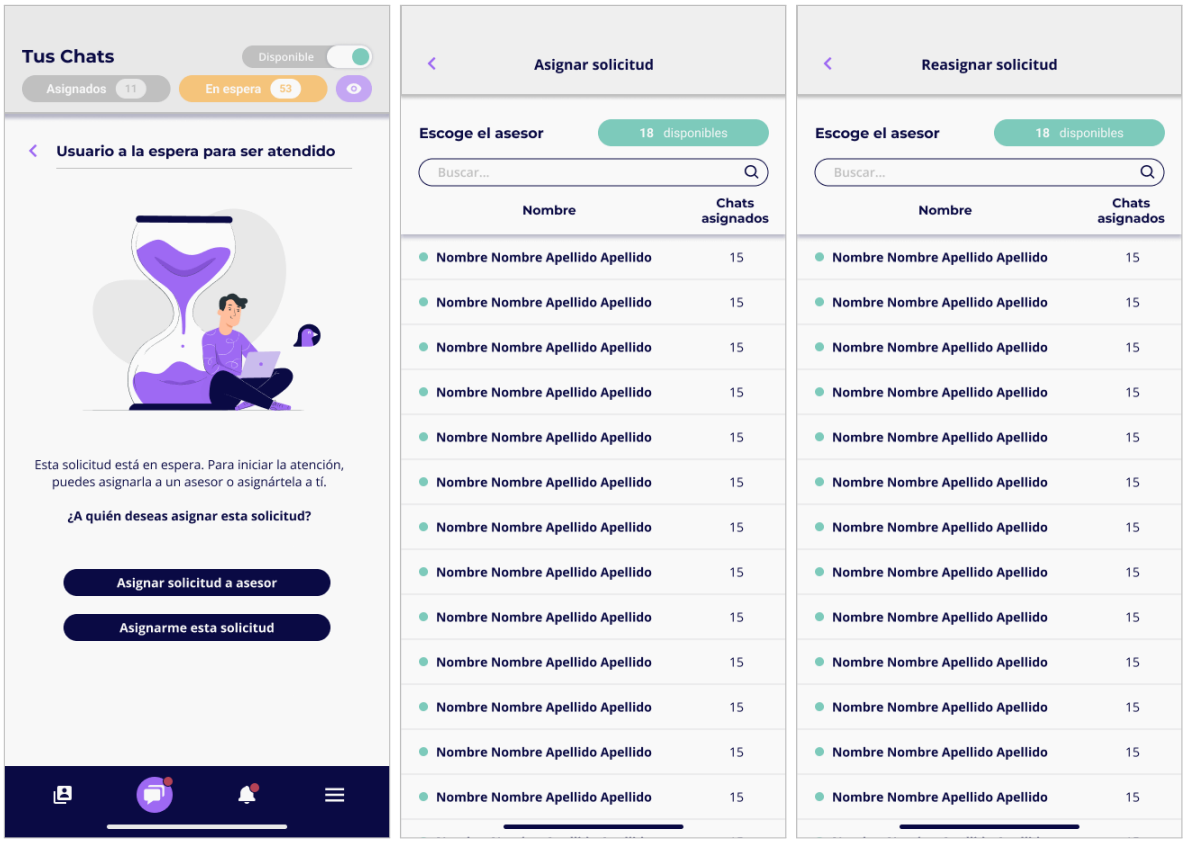


Figura 4.6: Diseño de las pantallas de asignar y reasignar

En la Figura 4.7 se muestra la pantalla del panel de notificaciones, donde es posible observar cada notificación con un título, una descripción y hace cuánto tiempo fue creada, además de distinguirse si fue no o leía por el usuario.

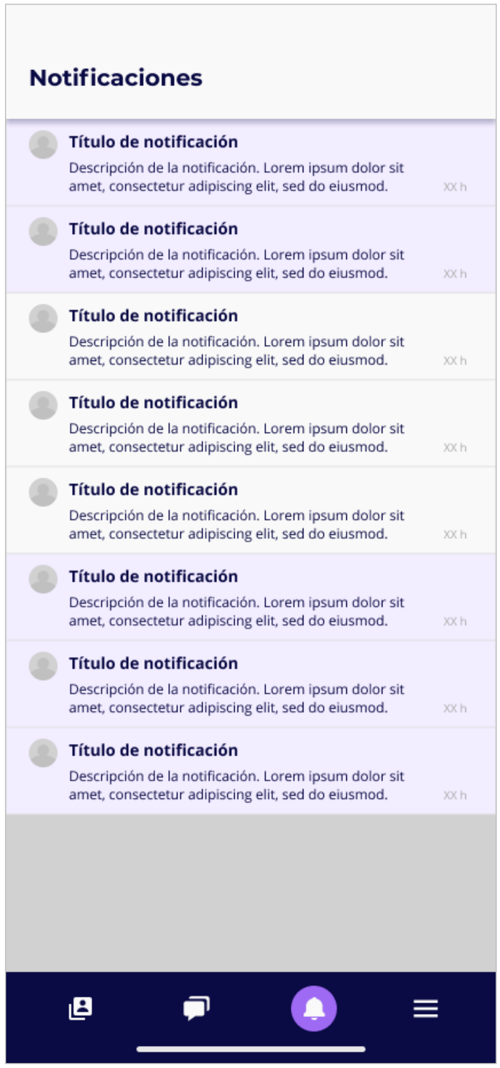


Figura 4.7: Diseño de la pantalla de notificaciones

En la Figura 4.8 se observan las pantallas relacionadas con la configuración del usuario y la compañía. La primera es la que permite ir a las otras dos, además de disponer el botón de "Cerrar sesión". En la segunda pantalla se pueden editar los datos del usuario, como su nombre, apellido y número de celular. La tercera pantalla permite editar la información de la compañía, como su nombre, ID oficial, país, y página web. Además, es posible ir a la configuración de los horarios de atención desde esta pantalla.

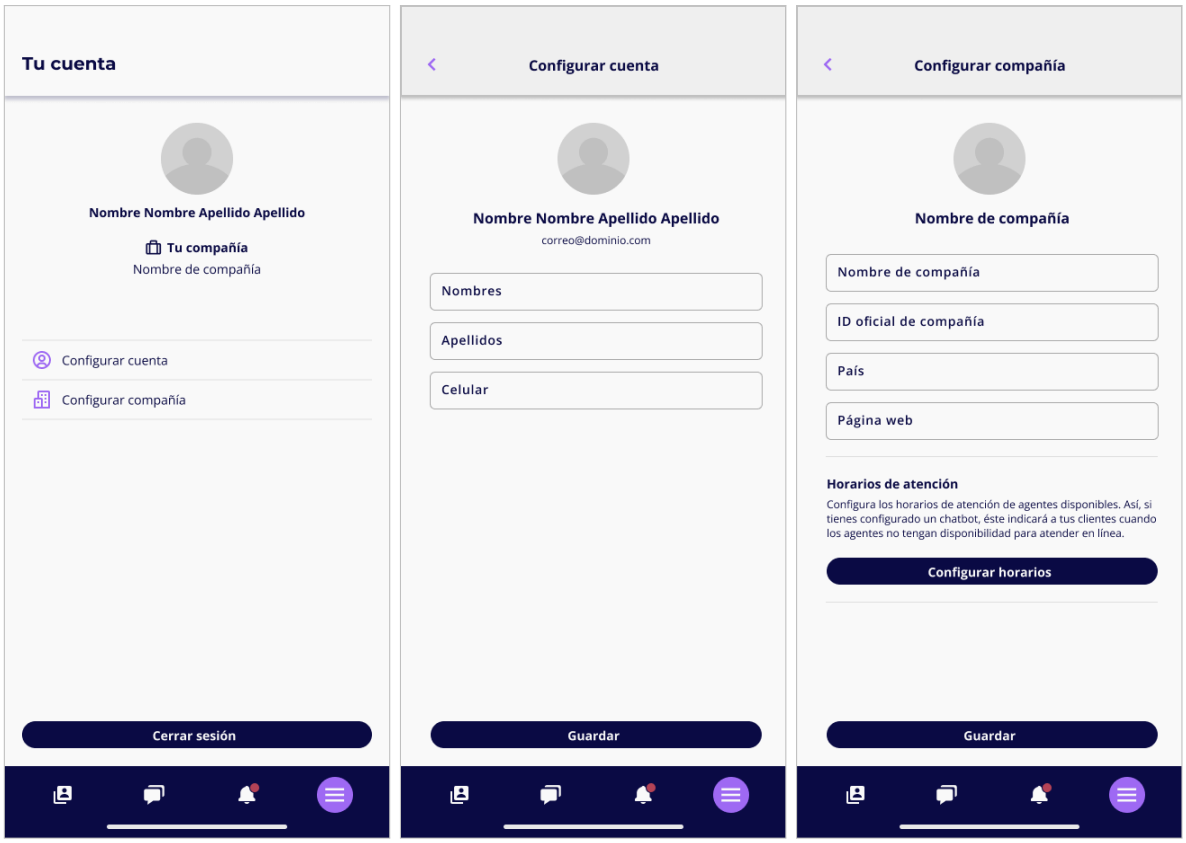


Figura 4.8: Diseño de las pantallas de configuraciones

Los resultados finales de las pantallas implementadas se pueden observar en el Anexo #1.

4.2. Arquitectura

En la Figura 4.9 se presenta la arquitectura general de toda la aplicación, es decir, la manera en que el backend y el frontend trabajan juntos para el funcionamiento del sistema.

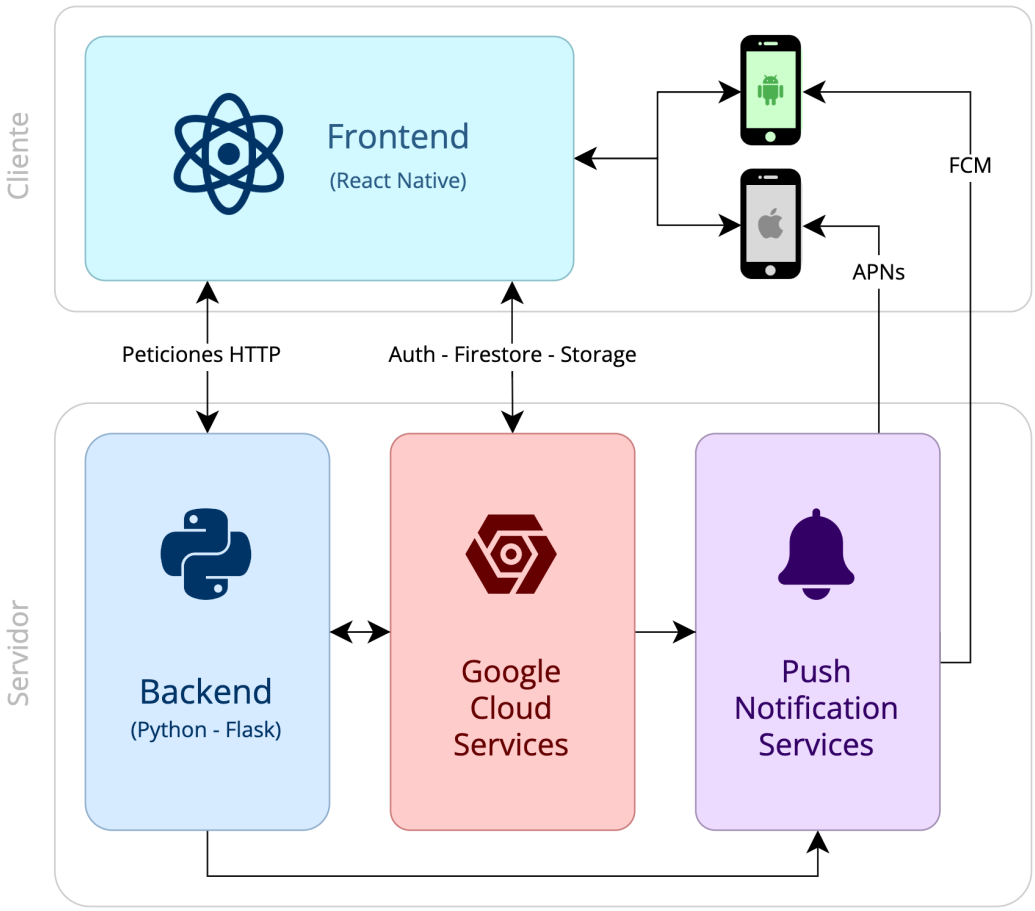


Figura 4.9: Arquitectura general

Se puede observar que la comunicación directa entre el frontend y el backend se da mediante peticiones HTTP. Por otro lado, el frontend usa algunos servicios de Google Cloud como Auth, Firestore y Storage, que también están relacionados con el backend. Este, por su parte, se comunica con algunos servicios de notificaciones push.

Debido a que es una app de mensajería, los servicios de notificaciones sirven para mantener actualizados los datos de la aplicación, especialmente al llegar mensajes y solicitudes de atención por parte de los clientes. Para Android se usa Firebase Cloud Messaging (FCM) y para iOS se usa Apple Push Notification service (APNs). Todos estos servicios se explican con más detalle en la sección 5.1.

4.2.1. Arquitectura del frontend

En la Figura 4.10 se muestra la arquitectura que se implementó en el frontend.

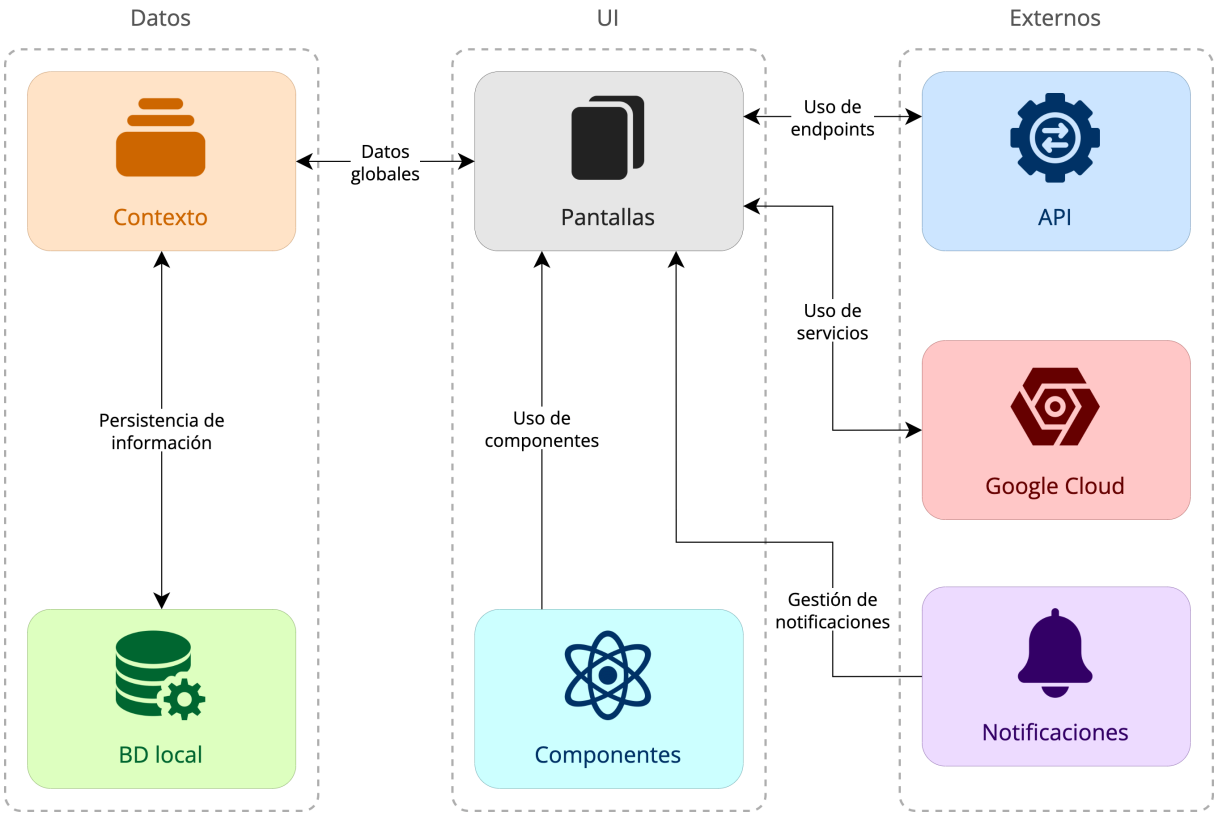


Figura 4.10: Arquitectura del frontend

Esta arquitectura se compone de tres capas:

- **Datos**

  Se encarga de administrar la información dentro de la aplicación y la provee a los componentes que la necesiten. Posee dos partes:

  - **Contexto**

    Un "contexto" es una herramienta de React que permite el flujo de variables a través de varios componentes de cualquier nivel de jerarquía dentro de un componente padre llamado "provider". En un contexto se suelen definir las variables y sus modificadores para ponerlos a disposición de todos sus componentes hijos.

- **BD local**

  Dado que cuando se cierra la aplicación se pierde el valor de todas las variables definidas dentro del programa, se vio la necesidad de definir una herramienta para persistir algunos datos de manera local en el almacenamiento del dispositivo.

  Cabe destacar que, como se observa en la Figura 4.10, el contexto es el que se encarga de persistir la información de manera local, por lo que este procedimiento es transparente para las pantallas y componentes que usen dicho contexto.

- **UI**

  Es la capa cuya función principal es mostrar la interfaz gráfica y manejar los eventos que ocurren cuando el usuario interactúa con la aplicación. Se compone de dos partes:

  - **Pantallas**

    Son componentes de React que muestran una pantalla de la aplicación. En el proyecto se definen cinco principales grupos de pantallas: uno para el Login y los otros cuatro para las secciones dentro del sistema (*Team*, *Chats*, *Notifications* y *Account*). Un ejemplo de pantalla es la de la conversación (ver Figura 4.6), en donde se tiene una cabecera, una lista de mensajes y un espacio para enviar un mensaje.

  - **Componentes**

    Son componentes atómicos de React que hacen parte de las pantallas. Un ejemplo de ellos son los botones, alertas, campos de texto, etc.

- **Externos**

  Es la capa donde se define la forma en que se interactúa con entes externos como la API, las notificaciones y los servicios de Google Cloud. Se compone de tres partes:

  - **API**

    Es el puente que permite la conexión entre los datos de las peticiones a la API y los datos internos de la aplicación. Cuando se requiere una comunicación con el backend, se llama a sus funciones para que estas traduzcan la información según las reglas de los endpoints y hagan los respectivos llamados a la API.

  - **Notificaciones**

    Se establece la manera en que se gestionan los diferentes tipos de notificaciones que llegan a la aplicación. Cada uno de ellos implica un proceso especial en el que se actualizan los datos locales para garantizar el acceso a la información más reciente.

- **Google Cloud**

  Se define con el objetivo de usar algunos de los servicios de Google Cloud como: Authentication (para inicio de sesión), Storage (para almacenar archivos de mensajes) y Firestore (para obtener información desde la base de datos).

4.2.2. Arquitectura del backend

En la Figura 4.11 se muestra la arquitectura utilizada en el backend.

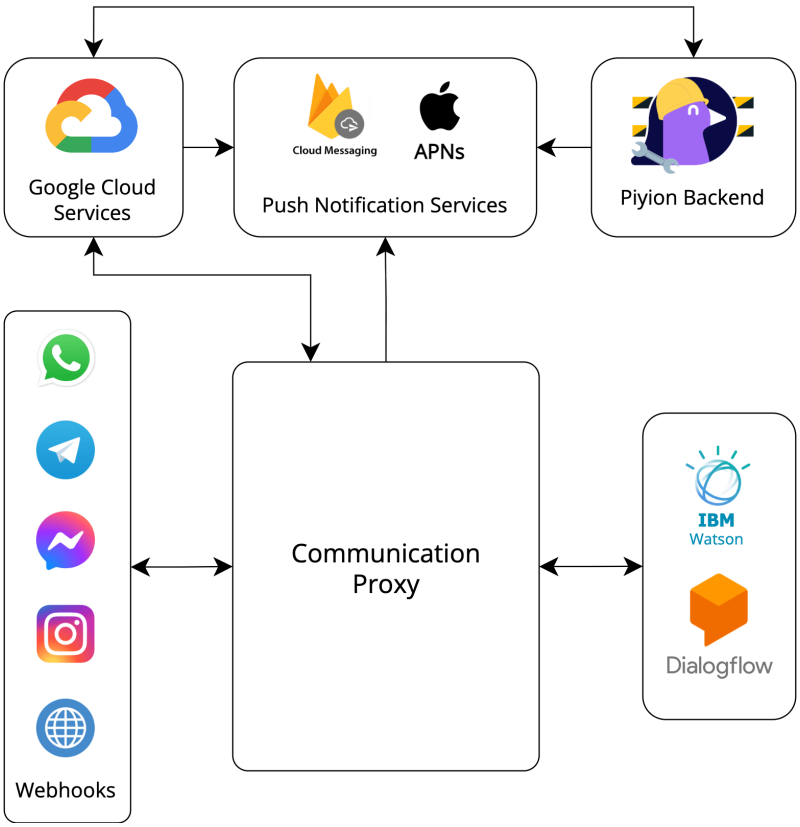


Figura 4.11: Arquitectura del backend

Se pueden apreciar varios componentes, como lo son:

- **IBM Watson & Dialogflow**

  IBM Watson es un sistema basado en inteligencia artificial, capaz de responder a preguntas en lenguaje natural; usualmente es usado por las empresas para crear chatbots para responder a las consultas de sus clientes [45].

Dialogflow es una herramienta similar a Watson, pero desarrollada por Google, capaz de comprender el lenguaje natural. Se utiliza para diseñar e implementar interfaces de usuario conversacional [46].

- **Google Cloud Services**

  Para este proyecto fueron usados algunos servicios Cloud de Google, como lo son Storage, Authentication y Cloud Messaging (descritos en la arquitectura del frontend). Además, se utilizó el servicio de Cloud Firestore, la cual es una base de datos no relacional, para el almacenamiento de toda los datos que sean generados desde y hacia el cliente.

  Además, se usaron servicios como Cloud Functions para desplegar las funciones, Cloud Scheduler para la ejecución de tareas de manera periódica, Secret Manager para el almacenamiento de información sensible de manera segura, Pub/Sub para el encolamiento y procesamiento de las notificaciones, entre otros.

  Todos estos servicios se explican con más detalle en la sección 5.1.5.

- **Push Notification Services**

  Se utilizó Firebase Cloud Messaging (FCM) para enviar notificaciones push a dispositivos Android y web. Esta herramienta también puede ser usada para dispositivos iOS; sin embargo, se presentaron algunos problemas con su uso; así que se decidió consumir APIs propias de Apple para enviar notificaciones push a dispositivos iOS. Estos servicios se explican más adelante en la sección 5.1.

- **Piyion Backend**

  El backend de Piyion consta de una serie de funciones (FaaS) las cuales exponen servicios web/APIs que permiten la manipulación de los datos de las diferentes empresas, conversaciones, agentes, administradores, estadísticas de compañías, contactos de una compañía, notificaciones de un usuario, entre otras cosas. Este backend se comunica con otros actores para poder hacer que Piyion funcione.

- **Webhooks**

  Los Webhooks son herramientas que permiten la comunicación entre diferentes aplicaciones en tiempo real, a veces se les conoce como API inversa [47]. En el caso de Piyion, permite la comunicación entre el Communication Proxy y algunos de los diferentes canales de comunicación que puede tener una empresa (WhatsApp, Instagram, Messenger, entre otros).

- **Communication Proxy**

  Es un elemento creado para comunicar a la plataforma Piyion con todos los posibles canales de comunicación que puede tener una empresa. Actualmente se permiten comunicaciones vía WhatsApp, Telegram, Facebook/Messenger, Instagram y chatbot web. Este "Communication Proxy" tiene la capacidad de comunicar los diferentes canales con otros elementos como lo son IBM Watson o Dialogflow, sin necesidad de pasar por la plataforma Piyion, esto debido a que algunos clientes tienen configurados chatbots y/u otras plataformas para el servicio al cliente, por lo que, además, se les podría vender el servicio de integración de los diferentes canales que tienen.

Capítulo 5

Implementación

En la etapa de implementación es donde se construye el software requerido, teniendo en cuenta el diseño elaborado previamente [48]. Cabe destacar que es en este momento que se empieza a escribir el código y llevar los modelos a la realidad, por lo que es imprescindible elegir las tecnologías adecuadas y aplicar las mejores prácticas de programación para garantizar un producto de calidad.

5.1. Tecnologías y herramientas

Para la implementación de la aplicación móvil se requirió de diferentes lenguajes, frameworks y herramientas que permitieron que su construcción se hiciera de forma eficiente y organizada. En el caso del frontend se usó React Native como framework principal junto con TypeScript y Sass como complementos. Para el backend se usó Flask (un framework de Python) junto con varios servicios de Google Cloud Platform (GCP), como Cloud Functions, Cloud Messaging, Cloud Pub/Sub, Cloud Firestore, Secret Manager, Cloud Storage, Cloud Scheduler, entre otros. Adicionalmente, se utilizó Apple Push Notification service (APNs) para el envío de notificaciones a dispositivos iOS.

Como utilidades generales se usó Git como sistema de control de versiones de la mano con GitHub para el manejo colaborativo del código.

5.1.1. React Native

React Native es un framework que permite la construcción de aplicaciones nativas para Android y iOS escribiendo código JavaScript y renderizando código nativo (Java para Android y Objective-C para iOS) [49]. Es de código abierto y mantenido por Facebook y la comunidad de desarrolladores. Se encuentra presente en varias aplicaciones como Walmart, Instagram, Wix, Facebook, entre muchas otras [50].

React Native posee el núcleo de React JS, es decir, su construcción se basa en componentes, lo que da la posibilidad de tener un código mantenible y reutilizable. En el Anexo #2 se muestra un ejemplo de cómo se aprovechó la reutilización de código gracias a los componentes de React.

En la Figura 5.1 se puede observar su arquitectura general, donde la aplicación se escribe en JavaScript y mediante un "puente" es posible la comunicación con los componentes nativos, no sólo traduciendo la lógica desde JavaScript a Java u Objective-C sino también en sentido inverso, escuchando los eventos de hardware que se ejecutan cuando el usuario interactúa con la aplicación.

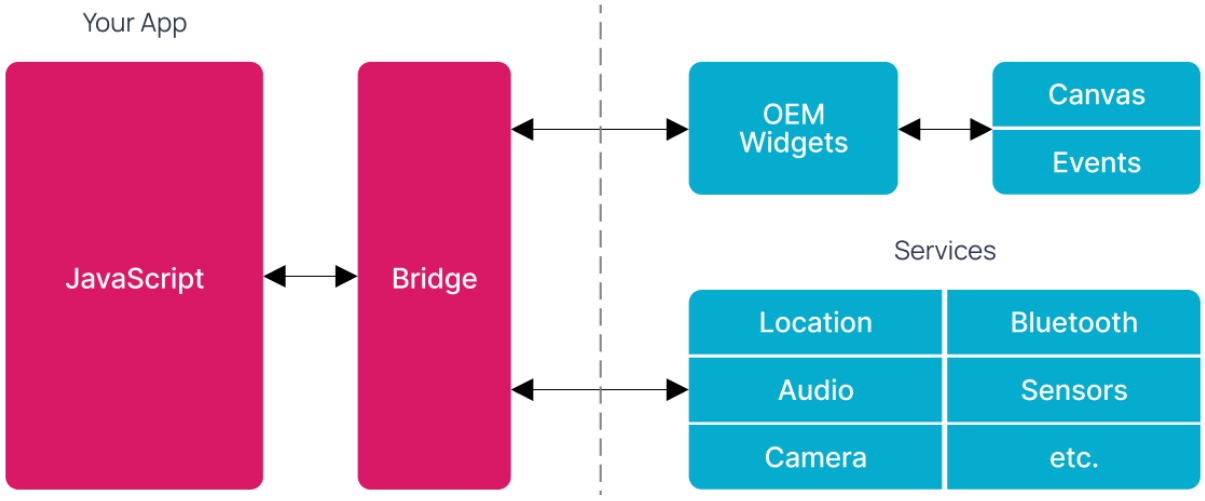


Figura 5.1: Arquitectura de React Native [2]

Las razones por las que se eligió usar esta tecnología para el desarrollo de la aplicación fueron las siguientes [51]:

- Debido a que está basado en componentes se facilita su mantenimiento y la reutilización de código.
- Permite adicionar "plugins" de terceros cuando por defecto no puede satisfacer las necesidades requeridas, instalando paquetes desde NPM.
- Facilita el proceso de desarrollo al poseer "live reload", que consiste en actualizar los cambios a medida que se van modificando los archivos.
- Es una tecnología en crecimiento y posee una gran comunidad de apoyo.

Algunas de estas ventajas pueden ser encontradas en otras herramientas como Flutter [52] o Cordova [53]; sin embargo, la elección de React Native se dio principalmente porque la aplicación web de Piyion está construida con React JS, por lo que el equipo de desarrollo está familiarizado con este enfoque. En este sentido, fue posible construir una aplicación con buenas prácticas aprendidas con anterioridad, y por supuesto, acelerando el proceso de desarrollo al "saltarse" la etapa de aprender un nuevo framework (incluso un nuevo lenguaje de programación).

5.1.2. TypeScript

TypeScript es un lenguaje de programación de código abierto [54] que es mantenido por Microsoft y la comunidad. Está construido a un nivel superior de JavaScript, lo que quiere decir que TypeScript provee al lenguaje de varias características adicionales que permite escribir código con

menos errores y más sólido [55], como se observa en la Figura 5.2. En este proyecto se usó TypeScript con la ayuda de un "template" que provee React Native que se configuró al inicializar la aplicación.

Las razones por las que se eligió usar esta tecnología para el desarrollo de la aplicación fueron las siguientes [56]:

- Posee análisis de código estático, que permite identificar varios errores en el momento que se está escribiendo y no sólo al ejecutarlo.
- Permite crear tipos personalizados que ayudan a que el código sea más organizado y legible.
- Es ideal para el desarrollo de aplicaciones medianas a grandes, ya que JavaScript puede presentar más dificultades a la hora de construir software a gran escala.

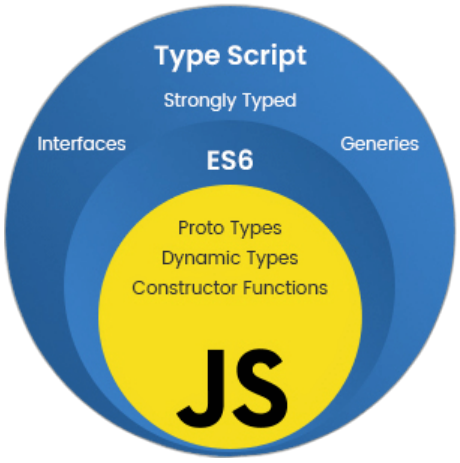


Figura 5.2: Características de TypeScript [3]

En el Anexo #3 se muestra un ejemplo de cómo se utilizó el análisis de código estático para encontrar errores en el código a medida que se iba escribiendo.

5.1.3. SASS

SASS (Syntactically Awesome Style Sheets) es un preprocesador de CSS [57] que provee una forma avanzada de escribir estilos, utilizando algunas características de los lenguajes de programación como las funciones, condicionales, ciclos y varias estructuras de datos (listas, mapas, etc).

Este lenguaje, a diferencia de CSS, utiliza sangría o indentación para definir los estilos; sin embargo, en este proyecto se usó SCSS (Sassy Cascading Style Sheets), que tiene todas las características de SASS pero con una sintaxis similar a la de CSS, lo que mejora su legibilidad [58].

Por lo general, en el desarrollo web los estilos se construyen a partir de código CSS; sin embargo, en React Native se deben aplicar con objetos de JavaScript. Por ello, en esta ocasión se usó una herramienta que transforma archivos `.scss` a archivos `.js` con los estilos compilados.

Las razones por las que se eligió usar esta tecnología para el desarrollo de la aplicación fueron las siguientes [59]:

- Permite una organización modular de los proyectos, lo que lo hace indispensable para proyectos grandes.
- Proporciona estructuras avanzadas propias de los lenguajes de programación como variables, listas, funciones y estructuras de control.

5.1.4. Flask

Flask es un framework escrito en Python, desarrollado para simplificar y hacer más fácil la creación de aplicaciones web, compatible con el patrón MVC (Modelo Vista Controlador) [60]. Flask a veces es denominado como un framework "micro", pero no para referirse a un proyecto pequeño o que solo sirva para hacer cosas mínimas, sino que con solo instalar Flask ya se dispone de las herramientas suficientes para la creación de una aplicación web funcional, lo que permite que se ahorre mucho espacio y no se instalen cosas innecesarias [61]. Además, en cualquier momento es posible instalar muy fácilmente algún plugin o extensión por si se requiere de alguna nueva funcionalidad. Algunas de las ventajas de utilizar Flask son [61]:

- Al ser un micro-framework, no se necesita nada más para crear un aplicación web básica de manera rápida. Si se pretende escalar es posible instalar extensiones.
- Posee un buen manejo de rutas, las cuales están a cargo del controlador, quien recibe todas las peticiones que hacen los clientes y determina a qué ruta está intentando acceder, para así ejecutar el código correspondiente.
- Cuenta con un depurador y soporte integrado para pruebas unitarias.
- Soporta de manera nativa el uso de cookies.

5.1.5. Google Cloud Platform

Google Cloud Platform (GCP) es una plataforma que ofrece al rededor de 100 servicios informáticos o productos, los cuales permiten trabajar con mayor eficacia o flexibilidad a equipos de desarrolladores, profesionales IT o empresas [62]. Estos productos o servicios se ejecutan remotamente en servidores de Google (Cloud Computing) y solo se cobra por el uso que se le de a la infraestructura que se utilice. Se escogió la suit de herramientas que brinda Google Cloud (y no otras como AWS o Azure) debido a que estas ya estaban siendo utilizadas en el desarrollo de la aplicación web de Piyion, por lo que era necesario continuar usándolas para las demás aplicaciones del sistema. Las herramientas de GCP que fueron usadas en este proyecto se describen a continuación.

5.1.5.1. Cloud Functions

Es un servicio que permite ejecutar porciones de código en la nube, dentro de entornos de ejecución que ellos mismos proveen, en donde se paga por el uso y tiempo que le tome a las funciones ejecutarse, un servicio denominado Functions as a Service (FaaS). Este servicio tiene la capacidad de aumentar, dependiendo de la demanda y la cantidad de recursos destinados a una misma función [63].

5.1.5.2. Cloud Messaging

Firebase Cloud Messaging (FCM) fue concebido como solución de mensajería multiplataforma que permite enviar mensajes sin algún costo de hasta 4000 bytes. Con FCM se puede notificar a una aplicación cliente que hay nuevos datos disponibles para sincronizar, lo que representa una gran utilidad para los casos de uso de mensajería instantánea [64]. En el proyecto fue utilizado para notificar a los dispositivos Android sobre nuevos eventos, como lo son la recepción de mensajes o la creación de nuevas solicitudes de atención. Más adelante se profundizará en los diferentes eventos y cómo se manejan tanto desde el backend como desde el frontend.

5.1.5.3. Cloud Pub/Sub

Este es un servicio que permite crear sistemas de productores y consumidores de eventos, los cuales son llamados “Publishers” y “Subscribers”. Los “Publishers” envían eventos al servicio de Pub/Sub de manera asíncrona, sin importar cómo o cuándo deben ser procesados [65]. De allí, Pub/Sub se encarga de entregar esos eventos a todos los “Subscribers” que reaccionen a ellos para que los puedan procesar [66]. Este servicio, dentro del proyecto, fue utilizado para el encolamiento o procesamiento de notificaciones.

5.1.5.4. Cloud Firestore

Es una base de datos de documentos NoSQL sin servidor que permite almacenar, sincronizar y consultar muy fácilmente datos en aplicaciones web [67]. Esta base de datos se escala muy fácilmente para ajustarse a cualquier demanda, sin necesidad de mantenimiento. Fue utilizado para el almacenamiento de todos los datos de Piyion.

5.1.5.5. Cloud Storage

Cloud Storage es un servicio para alojar objetos en Google Cloud, entendiéndose objeto como dato inmutable, un archivo en cualquier formato [68]. Para el proyecto se utilizó para el almacenamiento de objetos como imágenes, audios o documentos PDF que pueden ser enviados en una conversación en cualquiera de los diferentes canales digitales de una empresa.

5.1.5.6. Secret Manager

Secret Manager es un sistema de almacenamiento seguro que permite almacenar claves de APIs, contraseñas, certificados y otros datos que pueden llegar a ser sensibles [69]. Dentro de Piyion es utilizado para almacenar claves de APIs como las de los otros servicios de Google que son usados, o por ejemplo, para tokens que son necesarios para el envío o recepción de mensajes de los diferentes canales digitales de una empresa. Esta es información que en manos equivocadas puede ser usada para cosas diferentes a los objetivos de Piyion, por lo que es necesario el uso de este servicio.

5.1.5.7. Cloud Scheduler

Cloud Scheduler es un programador de trabajos periódicos, es decir, permite ejecutar algún trabajo cada cierta cantidad de tiempo [70]. Esto es útil ya que algunas empresas puede que requieran finalizar una conversación después de cierto tiempo en el cual no se ha respondido un chat. Es muy probable que requieran de esto debido al volumen de conversaciones que reciben en un día.

5.1.6. Apple Push Notification service

Apple Push Notification Service (APNs) es un servicio de Apple para enviar notificaciones push a dispositivos creados por ellos [71]. Este servicio debe ser usado desde una app que esté instalada en el dispositivo, que esté avalada por Apple. Dentro del proyecto fue usado para enviar notificaciones a dispositivos con sistema operativo iOS.

5.1.7. Git y GitHub

Git es un sistema de control de versiones distribuido, creado para el seguimiento de cambios en distintos tipos de proyectos de software. GitHub es una plataforma web que brinda herramientas para el control de versiones utilizando Git, además de permitir la colaboración y otras funciones relacionadas con el trabajo en equipo.

Para el desarrollo de este proyecto se utilizó GitHub debido a su facilidad de uso para el manejo de versiones y la colaboración.

Adicionalmente, se hizo uso de un servicio especial de GitHub llamado Github Actions, que permite implementar acciones que se ejecutan bajo ciertos eventos, como un “Pull Request”. Este servicio es utilizado principalmente para la automatización de pruebas unitarias y despliegue del backend [72].

5.2. Estructura del proyecto

5.2.1. Frontend

Para crear el proyecto de React Native se usó el comando de inicialización que la herramienta provee, configurando todo por defecto y agregando desde el inicio el uso de TypeScript. Debido a esto, se crearon varios archivos y carpetas base para una aplicación inicial. A medida que se iba avanzando en el desarrollo de la app se crearon algunas carpetas para la mejor organización de archivos. En la Figura 5.3 se muestra la distribución de carpetas de la raíz del proyecto.

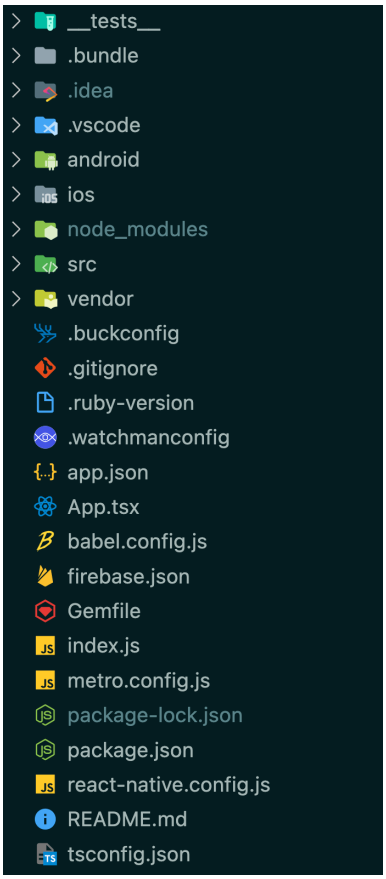


Figura 5.3: Estructura de archivos del frontend

La mayoría de estos archivos y carpetas son de configuración o se generan automáticamente al realizar cambios en el código de React Native. Las carpetas y archivos que interesan para la comprensión del proyecto son los siguientes:

- **/__tests__**: es donde se definen las pruebas que se realizan a la aplicación. En este caso se encuentran las pruebas de integración con la API, aunque también se pueden aplicar pruebas unitarias y de sistema.

- **/android**: es donde se encuentra el proyecto con el código nativo usando Java para la aplicación que corre en dispositivos Android. Sólo se modifica para la instalación de algunos plugins que requieren manipulación específica de componentes nativos.
- **/ios**: es donde se encuentra el proyecto con el código nativo usando Objective C para la aplicación que corre en dispositivos iOS. Sólo se modifica para la instalación de algunos plugins que requieren manipulación específica de componentes nativos.
- **/src**: es la carpeta más importante del proyecto debido a que es donde se encuentra casi todo el código fuente que le da la forma a la aplicación. Su estructura interna será explicada a detalle más adelante.
- **App.tsx**: es un archivo donde se define el componente "App" (padre de todos los componentes) y es el que se renderiza para construir la aplicación.
- **index.js**: es un archivo donde se registra tanto el componente "App" para ser renderizado, como el gestor de notificaciones en *background* (que llegan cuando la aplicación no está en primer plano).

La carpeta **/src** es la que contiene todo el código escrito bajo React Native. Su estructura se muestra en la Figura 5.4.

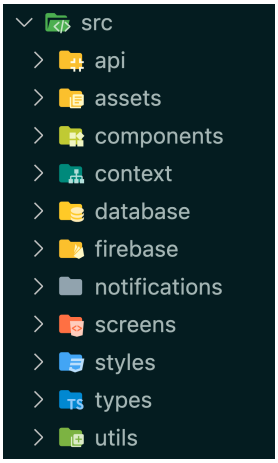


Figura 5.4: Estructura de archivos de la carpeta /src

Se destacan las siguientes carpetas:

- **/api**: es donde se definen todos los procedimientos de llamados a la API para el manejo de la información remota. Dentro de ella se encuentran especificados los endpoints y funciones que hacen las peticiones y transforman los datos a la estructura que se maneja localmente.

- **/assets**: guarda todos los activos generales del proyecto, como las imágenes y las fuentes.
- **/components**: es donde se encuentran los componentes UI generales, como los botones, inputs, alertas, etc.
- **/context**: es donde se definen los contextos para el manejo de la información global. Un contexto es un componente de React que permite el flujo de datos en toda la aplicación sin importar desde qué componente se consulte. Se definieron dos contextos:
  - DataContext, donde se encuentra toda la información dependiendo de cada usuario, como las conversaciones, los contactos, los miembros del equipo, etc.
  - NavigationContext, donde se definen las variables y funciones que se usan para la navegación dentro de la aplicación, como el nombre de la pantalla que se está mostrando, textos de las alertas, etc.
- **/database**: es donde se define la manera en que se va a persistir la información del usuario que ha ingresado al sistema. Aquí se inicializan las colecciones y se establecen los respectivos métodos para modificarlas. Se tienen las siguientes colecciones:
  - TeamStore, donde se guarda la información de los usuarios que son miembros de la compañía. Cada documento contiene datos como el nombre, correo, disponibilidad, número de conversaciones asignadas, etc.
  - ContactsStore, en la que se guarda la información de los contactos de la compañía. Cada documento tiene datos como el nombre, número de celular, comentario relacionado, etc.
  - ConversationsStore, donde se almacena toda la información relacionada a las conversaciones. Cada documento contiene datos como el canal, agente asignado, mensajes, etc.
  - NotificationsStore, en la que se guarda la información de las notificaciones del sistema. Cada documento tiene datos como el título, descripción, fecha, estado, etc.

  Los datos básicos del usuario (nombre, correo, etc) y los datos de la compañía (nombre, página web, etc) se almacenan directamente en el storage local de React Native (llamado "Async Storage"), debido a que es información que no puede ser listada.
- **/firebase**: es donde se definen las funciones y procedimientos relacionados con los servicios de Firebase. En este caso se usaron los siguientes módulos:
  - Auth, para la autenticación con correo electrónico y mediante una cuenta de Google.
  - Storage, para cargar y descargar archivos de mensajes.
  - Firestore, para obtener cierta información desde la base de datos.
- **/notifications**: es donde se define la forma en que se gestionan las notificaciones que llegan a la aplicación.

- **/screens**: es donde se encuentran definidas las pantallas de la aplicación organizadas por cada sección (*Login*, *Team*, *Chats*, *Notifications* y *Account*).
- **/styles**: contiene los estilos generales que se aplican a todos los componentes UI, como los colores y otras variables de SCSS.
- **/types**: es donde se especifican los tipos de datos propios que se usan en la aplicación. Estos son *User*, *Company*, *Contact*, *Conversation*, *Message* y *Notificacion*, principalmente.
- **/utils**: contiene varias funciones que no son propias de un componente sino que se pueden usar en diferentes lugares. Entre ellas destacan un método para ordenar un array mediante una fecha, un método para estandarizar un texto cambiándolo a minúsculas y transformando todas las tildes y caracteres especiales, entre otros.

5.2.2. Backend

El backend implementa una arquitectura serverless, que es ejecutada dentro de GCP. Google internamente utiliza Flask como framework para las Cloud Functions. En principio, Flask es muy laxo en la estructuración de los proyectos, pero debido a definiciones de diseño y arquitectura dentro del backend, la distribución de directorios se configuró como se muestra en la Figura 5.5.

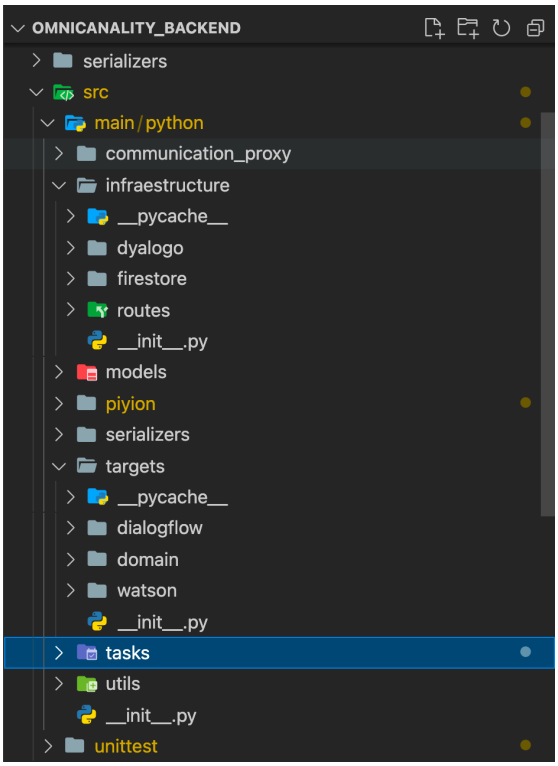


Figura 5.5: Estructura de archivos del backend

Se describen las siguientes carpetas:

- **/communication_proxy**: en este directorio se encuentran todos los componentes y la lógica que permite la distribución de las comunicaciones entre canales (que son los diferentes servicios de mensajería desde donde un usuario escribe) y los destinatarios (que podrían ser el buzón de Piyion para una compañía u otros servicios como Watson o Dyalogflow). Para más información, vea la sección 4.2.2.
- **/infraestructure**: define componentes clave de infraestructura de la aplicación, como lo son:
  - **/firestore**: en donde se implementan los controladores que permiten la comunicación con la base de datos Firestore.
  - **/routes**: en donde se definen los controladores que permiten la comunicación externa con el backend de Piyion.
- **/models**: aquí se encuentran las clases que contienen la estructura de los modelos de datos usados en la aplicación.
- **/piyion**: contiene la lógica de negocio propia de la aplicación.
- **/serializers**: se definen los serializadores y validaciones sobre entradas y salidas de las APIs y otros servicios.
- **/targets**: define la lógica que permite comunicar la aplicación con los diferentes destinatarios externos como podrían ser Watson o Dyalogflow.
- **/tasks**: contiene servicios periódicos que se ejecutan con el fin de realizar tareas automáticas, como el procesamiento de estadísticas y la expiración de conversaciones.
- **/utils**: contiene utilidades de propósito general usadas en diferentes lugares de la aplicación.
- **/unittest**: contiene las pruebas unitarias del proyecto. Permiten garantizar la calidad y consistencia de la aplicación.

5.3. Modelo de datos

El modelo de datos se definió siguiendo el principio KISS, el cual indica que las cosas funcionan y se entienden mejor si son simples y directas: *"**K**eep **I**t **S**imple and **S**traightforward"* [73]. Para entender el modelo de datos en cuestión, primero se debe recordar que Firestore es una base de datos no relacional de documentos, los cuales se agrupan y forman colecciones.

En la Figura 5.6 se explica la manera en la que se presenta una compañía dentro de la base de datos, la cual puede contar con subcolecciones para los contactos, chats, estadísticas diarias y notificaciones. Además, se muestra cómo cada chat tiene asociada una subcolección de mensajes. Cabe resaltar que las notificaciones, dependiendo de su tipo, son para unos u otros miembros de una compañía.

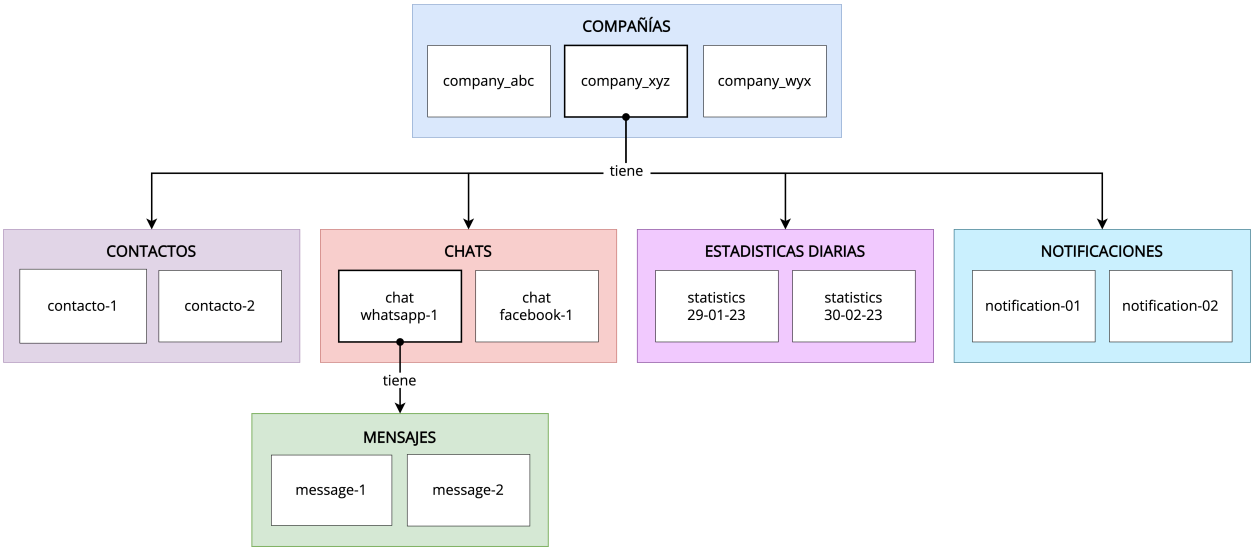


Figura 5.6: Modelo de datos de compañías

Por otro lado, en la Figura 5.7 se muestra cómo cada usuario tiene asociada una subcolección de calificaciones, que son las que obtiene después de atender una solicitud. Además, se presenta la subcolección de llaves de notificaciones, donde se almacena tanto el *id* como la fecha de creación de la notificación. En esta subcolección se almacenan las notificaciones que son propias de cada usuario; es decir, debido a que no todas las notificaciones son para todos los usuarios de una compañía, se define cuáles le corresponden a cada uno. Esta decisión de diseño se tomó porque resultaría muy ineficiente y se replicaría muchas veces la misma información si se almacenaba todo el contenido de una notificación en una subcolección de los usuarios, ya que en cada usuario al que le tendría que llegar la notificación se debería escribir todo el contenido de la misma. El proceso de notificaciones se explicará en la sección 5.4.1.

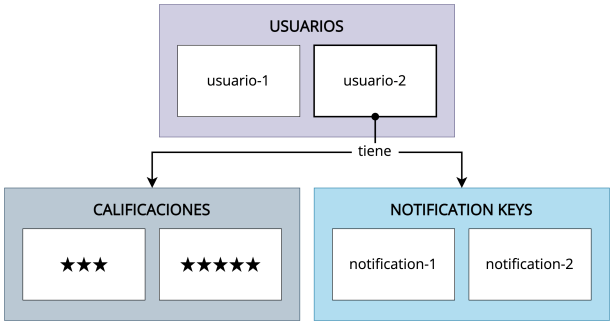


Figura 5.7: Modelo de datos de usuarios

En la Figura 5.8 se muestra cómo interactúan los diferentes modelos dentro de la base de datos. En ella se observa cómo un usuario pertenece a una compañía. Este tipo de relación es excluyente, es decir, que un usuario sólo puede ser parte de una compañía. También, se muestra cómo un usuario puede tener invitaciones de ingreso a compañías, cada invitación tuvo que haberla enviado un usuario administrador que ya perteneciera a dicha compañía. Asimismo, se puede observar cómo un usuario puede tener varios dispositivos registrados. A estos dispositivos es a los que llegarán las notificaciones, y su registro se hace al momento de iniciar sesión.

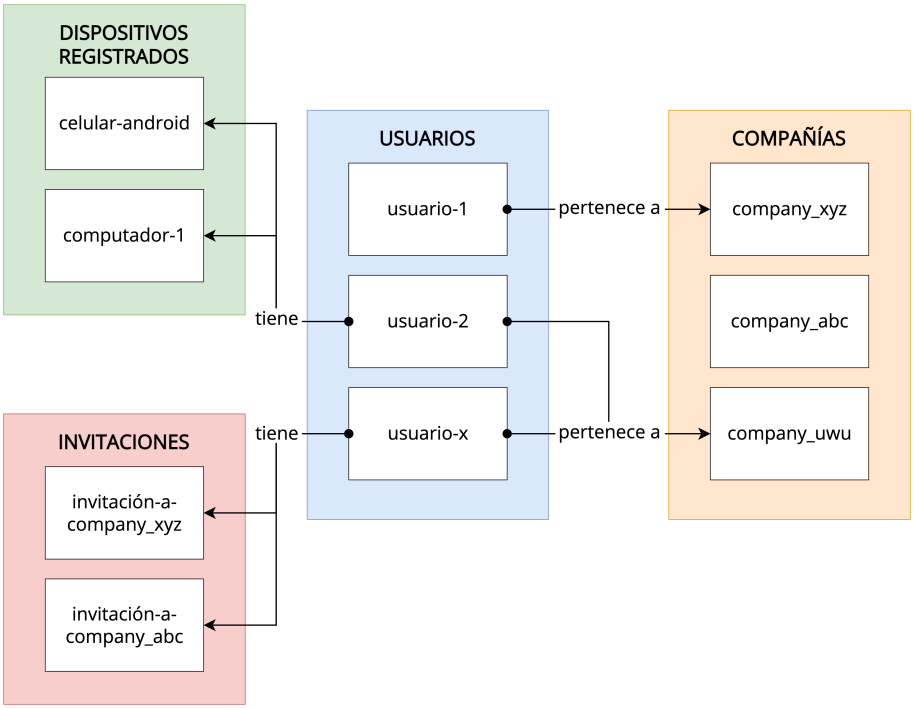


Figura 5.8: Modelo de datos completo

5.4. Notificaciones

Para poder tener actualizada la aplicación móvil con lo que pudiera pasar en la web u otros dispositivos, fue necesario implementar un sistema que permitiera notificar a los diferentes dispositivos de un usuario sobre un evento o cambio que hubo, para que se pudiera reaccionar a esa notificación dependiendo de su tipo. Las notificaciones se pueden clasificar en:

- **Nuevo mensaje**. Cuando llega un nuevo mensaje de una conversación.
- **Nueva conversación**. Cuando se crea una solicitud de atención que debe ser gestionada por los agentes. Inicialmente se encuentra en el estado de "en espera".
- **Invitación a compañía**. Cuando un usuario que no tiene una compañía asociada es invitado por un usuario administrador a unirse a una compañía.
- **Conversación asignada**. Cuando una conversación en estado de "en espera" es asignada a un agente para ser atendida.
- **Conversación reasignada**. Cuando una conversación cambia de agente encargado de atender la solicitud.
- **Conversación finalizada**. Cuando un agente da por terminada una solicitud de atención o cuando se finaliza si no se ha respondido después de cierto tiempo. Este tiempo puede ser configurado por un administrador de la compañía desde la aplicación web.
- **Contacto creado**. Cuando un agente crea un contacto relacionado a una conversación.
- **Contacto actualizado**. Cuando un agente modifica la información de un contacto existente.

Algunas de ellas solamente entregan la información y no se le informa al usuario; sin embargo, las que son de tipo "nuevo mensaje" (por ejemplo) muestran una notificación push en caso de que la aplicación esté cerrada, similar a WhatsApp.

5.4.1. Proceso de notificaciones

El proceso de notificaciones fue realizado con las herramientas de Google Cloud Pub/Sub, FCM y APNs. Para dar con la arquitectura actual fue necesario pasar por un problema, el cual se describe a detalle en la sección 5.5.4. El evento de enviar notificaciones pueden ser activado tanto desde las APIs como desde la consola de FCM [74] (ver Figura 5.9).

Si este evento es activado desde el backend entonces suceden dos cosas: (1) se escribe la información de la notificación en base de datos, la cual muta dependiendo del tipo de notificación (ya que la información de cada evento cambia); y, (2) se envía un mensaje al componente Pub/Sub Notifications Service de la Figura 5.9, el cual está encargado de publicar dicho mensaje en el tema

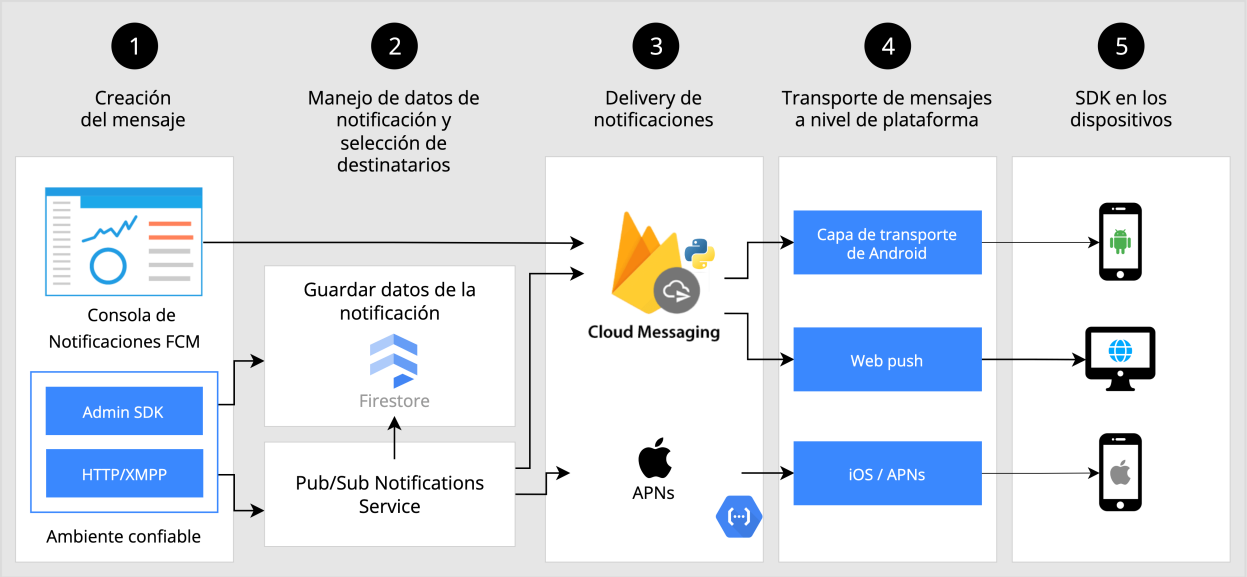


Figura 5.9: Proceso de notificaciones

de las notificaciones. En el mensaje se indica el identificador (*id*) de la compañía a la que pertenecen los usuarios a los que va dirigida la notificación y el identificador (*id*) de la notificación.

Estando allí, el *Notifications Subscriber* (ver Figura 5.10), toma en orden los mensajes que fueron publicados y los procesa de la siguiente manera: obtiene la notificación que fue almacenada anteriormente por el servicio web, calcula los destinatarios de la notificación y le envía esta información a los diferentes servicios que enviarán la notificación a los diferentes dispositivos. Para Web y Android el encargado será FCM, mientras que para iOS lo será APNs.

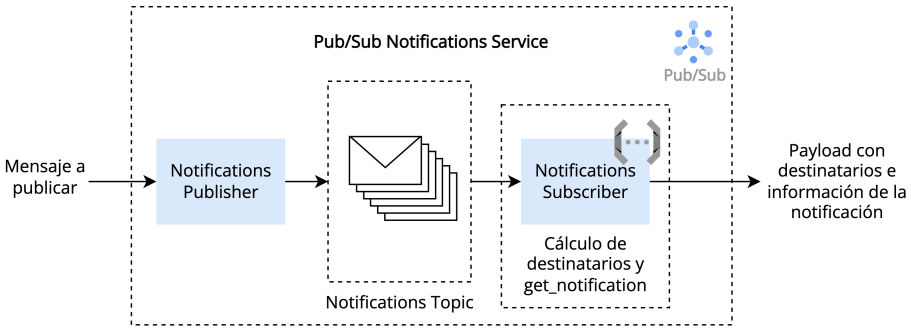


Figura 5.10: Diagrama del Pub/Sub para el servicio de notificaciones

5.4.2. Gestión de notificaciones desde la app

Teniendo en cuenta que las notificaciones sirven para mantener actualizados los datos en la aplicación, su comportamiento se puede entender como una instrucción que dice qué es lo que hay que cambiar. Un aspecto a tener en cuenta es que se tienen dos escenarios en los que puede llegar una notificación: *foreground* y *background*. Una aplicación se ejecuta en *foreground* cuando se encuentra en primer plano y en *background* cuando está en segundo plano.

Cuando llega una notificación en *foreground* los datos se actualizan tanto en la base de datos local (la que permite la persistencia de información) como en las variables del programa. Si la notificación llega en *background* sólo se actualiza la base de datos local, debido a que no es posible acceder a las variables del programa en este escenario. Sin embargo, se cuenta con un evento que indica que la app está de nuevo en *foreground*, y cuando esto sucede, se carga la información desde la base de datos local hacia las variables del sistema. Por otro lado, cada notificación se maneja de forma diferente dependiendo de su tipo:

- **Nuevo mensaje**. Se actualiza la conversación que recibe el mensaje y se le informa al usuario que hay nuevos mensajes. Esta notificación sólo la puede ver el usuario que tiene asignada la conversación, sin embargo, la tarea de actualizar la conversación se ejecuta para dicho usuario y todos los administradores de la compañía (para la funcionalidad de supervisión).
- **Nueva conversación**. Se agrega esta nueva conversación a la colección correspondiente y se le informa al usuario que hay nuevas solicitudes que atender. Esta notificación es visible para todos los miembros de la compañía.
- **Invitación a compañía**. Se le informa al usuario que ha sido invitado a unirse como asesor en una compañía, mostrándole el nombre de la persona que lo invitó y el nombre de dicha compañía.
- **Conversación asignada**. Se actualiza la conversación para relacionarla con el agente que la va a atender. Además, si dicho agente corresponde al usuario de la aplicación, se le informa que se le ha asignado una conversación para que sea atendida.
- **Conversación reasignada**. Se actualiza la conversación para cambiar el agente encargado de atenderla. De manera similar a la notificación de "conversación asignada", si el nuevo agente corresponde al usuario de la aplicación, se le informa sobre dicha reasignación.
- **Conversación finalizada**. En este caso, sólo se elimina la conversación de la colección correspondiente. No es necesario informar al usuario de dicho evento.
- **Contacto creado**. Se agrega el contacto a la colección correspondiente y se actualiza dicha información para relacionarla con la conversación desde la cual se creó. Este tipo de evento no se le informa al usuario.
- **Contacto actualizado**. Se actualiza la información de dicho contacto en la colección correspondiente. No es necesario informar al usuario sobre este evento.

5.4.3. Mejora en el consumo de recursos de la aplicación web

El proceso de notificaciones que fue añadido como parte del desarrollo de este trabajo permitió una importante mejora en el consumo de recursos de base de datos y servicios del backend por parte de la aplicación web.

A medida que crecía el uso de la aplicación web, se encontró que algunas funcionalidades como el almacenamiento, lectura y actualización de contactos estaban consumiendo gran cantidad de recursos, dado que los clientes registraban gran cantidad de estos diariamente y hacían uso de ellos.

Por medio de las notificaciones creadas para las aplicaciones móviles, se implementó en la aplicación web un aislamiento del consumo de recursos de base de datos, creando una base de datos local que se inicializa durante el inicio de sesión y se actualiza por medio de los eventos reportados por las notificaciones. Dicha mejora permitió, junto con otras mejoras, una disminución del consumo de recursos de más del 90 %. Se puede observar en la Figura 5.11 el alto consumo en los días de semana que venía en crecimiento, hasta las últimas dos barras, que muestran un comportamiento mucho más favorable.

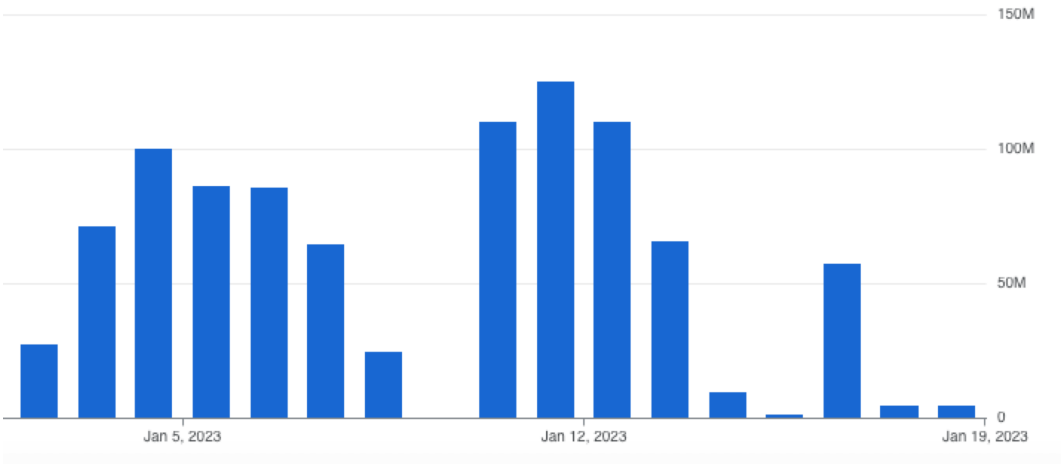


Figura 5.11: Consumo base de datos del 02/01/2023 al 19/01/2023

5.5. Problemas en el desarrollo

Durante la construcción de la aplicación se presentaron distintos inconvenientes que requerían de soluciones efectivas para continuar con el desarrollo. A continuación se describen los principales problemas y la manera en que se resolvieron.

5.5.1. Expo vs React Native CLI

Expo es una plataforma open source para desarrollar aplicaciones nativas para Android, iOS y web usando JavaScript y React [75]. La principal ventaja de utilizar Expo es que permite correr las aplicaciones de React Native de una forma sencilla, sin demasiadas configuraciones y teniendo pocos conocimientos sobre el desarrollo móvil. Por su parte, React Native CLI es la forma clásica de construir y ejecutar las aplicaciones, que contrario a Expo, requiere de una configuración del ambiente de desarrollo que suele ser larga y propensa a encontrarse con varios errores.

Al principio la aplicación se construyó usando Expo; sin embargo, en una etapa temprana del proyecto se cambió por React Native CLI, esto debido a que Expo, al ser más reciente y más amigable con los desarrolladores principiantes, tenía ciertas limitaciones en el uso de algunas características. Aunque la configuración del ambiente de desarrollo fue más compleja, se pudo solucionar los problemas iniciales y se continuó con este enfoque.

5.5.2. FCM en iOS

Al configurar FCM en la aplicación se obtuvo los resultados esperados en Android, mientras que en iOS no funcionó. Aunque se buscó el motivo por el cual no se comportaba según lo esperado, no se encontró una solución para ello.

Teniendo en cuenta lo anterior, se optó por usar directamente APNs, el servicio que permite enviar notificaciones a dispositivos Apple. En resumen, se usó FCM para enviar notificaciones a dispositivos Android y web, mientras que para dispositivos iOS fue APNs.

5.5.3. Notificaciones en background en iOS

Habiendo implementado las notificaciones en iOS, se encontró que en el caso de que la aplicación estuviese en primer plano (es decir, mostrándose en la pantalla) y cuando estuviese abierta pero no en primer plano, las notificaciones llegaban como se esperaba. Sin embargo, al probar las notificaciones cuando la aplicación estaba completamente cerrada, no era posible procesarlas dentro de la lógica de la aplicación.

Por otro lado, se encontró que en cuanto se abría de nuevo la aplicación, solamente llegaba la última notificación recibida. Investigando en la documentación oficial de Apple se pudo comprender que en el dispositivo se almacenaba sólo una notificación “pendiente” mientras que la aplicación estuviera en background, que luego sería entregada a la aplicación cuando esta se ejecutara de nuevo en foreground.

La solución que se implementó consistió en almacenar en la base de datos la información de las notificaciones. Con esto fue posible que cuando la aplicación se ejecutara de nuevo, tomara la fecha de la última notificación procesada y realizara una petición al backend para obtener, a partir de esa fecha, las notificaciones faltantes en ese dispositivo.

5.5.4. Desbordamiento de recursos en APIs por utilidad de notificaciones

Durante la implementación de las notificaciones en el backend, se creó una utilidad que permitía encolar las notificaciones en el servicio de FCM cuando ocurría un evento que requería notificación. Muchos de estos eventos debían ser enviados a múltiples dispositivos, ya que en principio, un usuario puede tener varios dispositivos asociados. Adicional a ello, cada notificación debe ser enviada a uno o varios usuarios dependiendo de su tipo, lo cual implica cálculos adicionales.

En un principio, esto no representaba un problema; sin embargo, una vez esta lógica ingresó a producción y hubo estrés dentro de la aplicación web, se vio un detrimento en el rendimiento de las APIs. Por ello, y basado en el principio de *single responsability* [76], el proceso de cálculo y envío de notificaciones fue trasladado a un servicio activado por medio del patrón pub/sub. De ahí en adelante, el backend encola un mensaje que permite saber a este servicio qué se debe notificar. Para más detalles sobre esta implementación, vea la sección 5.4.1.

5.5.5. Soporte en el desarrollo móvil con React Native

Uno de los mayores retos en el caso del frontend consistió en el hecho de que aunque el desarrollo móvil ha ido creciendo, el soporte encontrado en línea (y otros medios) sigue siendo mucho menor en comparación con otras áreas como el desarrollo web. Esto combinado con la naturaleza del desarrollo nativo en que se debe tener en cuenta el comportamiento del sistema operativo y todo lo que esto conlleva, hace que sea más común encontrarse con problemas que no siempre se solucionan de forma rápida y sencilla. Sin embargo, a medida que se avanzaba en el proyecto, se iba haciendo más habitual encontrarse con estos problemas y buscar soluciones creativas para poder lograr implementar lo que se requería.

5.5.6. Asimilación de nuevos conceptos y su implementación

En el contexto del desarrollo de software y, en particular, en la computación en la nube, la asimilación de nuevos conceptos y tecnologías puede ser un desafío significativo. La complejidad inherente a estos conceptos y la necesidad de adaptarse constantemente a los cambios en el entorno de la nube requieren un compromiso continuo por parte de los desarrolladores. Además, la implementación práctica de estos conceptos puede ser un desafío adicional, ya que cada proyecto es único y presenta sus propios requisitos y limitaciones. El desafío personal que esto conlleva puede ser significativo para los desarrolladores, quienes deben dedicar tiempo y esfuerzo para mantenerse

actualizados y adquirir nuevas habilidades. Además, es común encontrarse con situaciones donde es necesario experimentar y cometer errores para poder aprender y mejorar.

5.6. Estrategias para la mantenibilidad

Con la finalidad de que el código sea mantenible y poder hacer que esté bien estructurado, legible y con facilidad para ser modificado, se plantearon varias estrategias. En el caso del frontend fueron las siguientes:

- **Formateo del código.** Se usó Prettier, una herramienta que sirve para darle un formato especial al código siguiendo algunos estándares sobre legibilidad. Este se aplicó en el proyecto mediante una extensión de Visual Studio Code [77].

- **Análisis estático del código.** Se usó TypeScript, que como se explica en la sección 5.1.2 posee algunas características para identificar errores en el momento que se escribe el código y no al ejecutar el programa.

  Utilizando un paquete de NPM llamado "TypeScript Coverage Report" [78], se ejecutó un análisis de todo el código que mide el uso de las prácticas de TypeScript. El resultado se muestra en la Figura 5.12.

TypeScript coverage report

Summary

Percent	Threshold	Total	Covered	Uncovered
100%	100%	11489	11489	0

Files

Filename	Percent	Total	Covered	Uncovered
src/context/DataContext/DataContext.ts	100.00%	96	96	0
src/context/NavigationContext/NavigationContext.ts	100.00%	56	56	0
src/api/endpoints.ts	100.00%	25	25	0
src/api/services/notificationsServices.ts	100.00%	30	30	0
src/utils/date.ts	100.00%	153	153	0
src/api/actions/notificationsActions.ts	100.00%	50	50	0
src/firebase/messaging.ts	100.00%	23	23	0
src/database/setup.ts	100.00%	31	31	0
src/utils/text.ts	100.00%	90	90	0
src/database/notifications.ts	100.00%	247	247	0
src/database/conversations.ts	100.00%	111	111	0

Figura 5.12: Análsis del uso de TypeScript

En el caso del backend se implementaron las siguientes estrategias:

- **Formateo del código siguiendo *PEP8*.** Dado que el código del backend está hecho en Python, se vio necesario un estilo de código, por lo cual se utilizó un *formatter* llamado *Black* [79], que sigue los estándares de PEP8. Esta es una guía creada por la comunidad y es ampliamente aceptada, es considerada como la guía estándar para escribir código legible y mantenible en Python [80].

- **Análisis estático del código.** Con el uso de la herramienta Flake8 para Python se logró hacer un análisis de código estático antes de hacer un *commit*, ejecutando de manera manual o, al hacer un *commit* mediante un Action de GitHub de manera automática. Esta herramienta basa en:

  - **PyFlakes**: es un verificador de código estático que busca errores de sintaxis y de semántica en el código [81].
  - **Pycodestyle**: es un verificador de estilo de código que busca violaciones de las recomendaciones de estilo de código descritas en PEP8 [82].
  - **Ned Batchelder's McCabe script**: es un script escrito por Ned Batchelder que mide la complejidad ciclomática del código. La complejidad ciclomática se refiere a la cantidad de caminos únicos que puede tomar el control de un programa. Un valor alto de complejidad ciclomática puede indicar una posible falta de claridad y mantenibilidad en el código [83].

  En la Figura 5.13 se puede evidenciar cómo la herramienta Flake8 es capaz de detectar las importaciones no usadas y dar aviso de esto. El código del backend está libre de errores de sintaxis o semántica y *warnings* de cualquier tipo, ya que todo se limpia (de errores y warnings) y prueba antes de desplegar a producción.

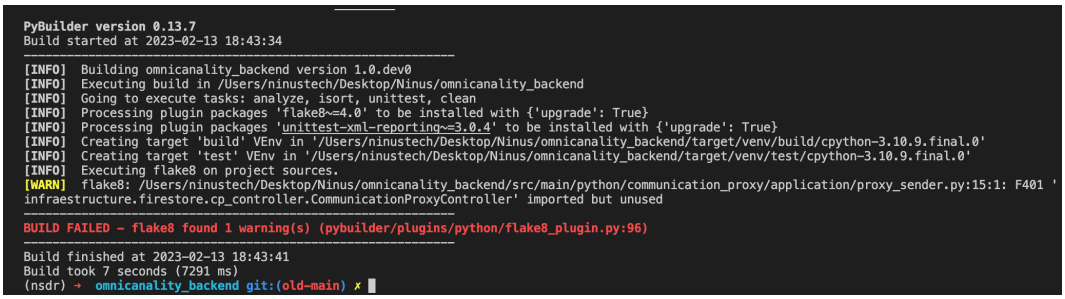


Figura 5.13: Error que lanza Flake8 cuando detecta importaciones no usadas

Capítulo 6

Pruebas

Las pruebas aplicadas a un software permiten verificar que este cumpla con ciertas medidas de calidad previamente identificadas. Estas medidas de calidad están directamente asociadas a los requisitos definidos al inicio del proyecto, por lo que en esta etapa se contrastan los resultados con las especificaciones del software que fueron dadas [84].

6.1. Pruebas funcionales

En el frontend se realizaron pruebas funcionales en los niveles de sistema e integración. En el backend se llevaron a cabo pruebas de componente (o unitarias).

6.1.1. Pruebas de sistema

Para realizar las pruebas de las principales funcionalidades de la aplicación se utilizaron tres diferentes estrategias de prueba:

- **Tablas de decisión**. Consiste en una representación tabular donde se definen los valores de entrada, casos, reglas y condiciones de prueba. Es una herramienta muy eficaz que se utiliza tanto para la gestión de requisitos como para las pruebas de software complejas [85].
- **Particiones de equivalencia**. Se basa en dividir las variables de entrada en particiones equivalentes que pueden utilizarse para derivar casos de prueba, lo que hace que se reduzca el tiempo necesario para la ejecución de las pruebas debido a que se eligen representantes por cada escenario identificado [86].
- **Transición de estado**. Ayudan a analizar el comportamiento de la aplicación para diferentes condiciones de entrada. En este sentido, se identifican las funcionalidades en donde los valores de entrada provocan cambios de estado o cambios en la salida [87].

En la Figura 6.1 se muestran las estrategias utilizadas para cada una de las funcionalidades (columna 3), además del número de casos de prueba que se ejecutaron (columna 4) y cuáles de ellos resultaron exitosos (columna 5). En el Anexo #5 se presentan algunos ejemplos de las técnicas utilizadas en algunas funcionalidades.

ID	Nombre	Estrategia	CP ejecutados	CP exitosos
F01	Iniciar sesión con correo y contraseña	Tablas de decisión	3	3
F02	Iniciar sesión con Google	Tablas de decisión	2	2
F03	Consultar la lista de los miembros del equipo	No aplica	1	1
F04	Consultar la información de un miembro del equipo	No aplica	1	1
F05	Consultar la lista de chats asignados	No aplica	1	1
F06	Consultar la lista de chats sin asignar	No aplica	1	1
F07	Supervisar el equipo	Transición de estado	1	1
F08	Cambiar la disponibilidad	Transición de estado	2	2
F09	Consultar los mensajes de un chat	No aplica	1	1
F10	Enviar un mensaje	Tablas de decisión	3	3
F11	Consultar la información del contacto de un chat	No aplica	1	1
F12	Guardar un contacto	Particiones de equivalencia	8	7
F13	Reasignar la conversación	Transición de estado	1	1
F14	Finalizar la conversación	Transición de estado	1	1
F15	Asignar una conversación	Transición de estado	2	2
F16	Consultar las notificaciones	No aplica	1	1
F17	Editar la información del usuario	Particiones de equivalencia	4	4
F18	Editar la información de la compañía	Particiones de equivalencia	4	4
F19	Cerrar sesión	Transición de estado	1	1
		TOTAL	**39**	**38**

Figura 6.1: Estrategias de prueba usadas para cada funcionalidad

Tal como se observa en la Figura 6.1 la funcionalidad F12 (“Guardar un contacto”) obtuvo un caso de prueba fallido. Este corresponde a la validación que se hace al ingresar un correo en el que se debe verificar que tenga la estructura propia de un correo electrónico, por ejemplo “nombre@dominio.com”. Dado que esta validación no se estaba realizando, era posible guardar un correo como “nombre.apellido”, por esto, se corrigió en el código lo que se muestra en la Figura 6.2 en las líneas 54 a 56.

```
36      const saveContact = () => {
37          const newFirstName = firstName.trim();
38          const newLastName = lastName.trim();
39          const newEmail = email.trim();
40          const newPhone = phone.trim();
41          const newCompany = company.trim();
42          const newComment = comment.trim();
43
44          // Verifica la correctitud de los datos
45          if (newFirstName === '') {
46              openAlert('Valor inválido', 'Por favor, ingrese un nombre');
47              return;
48          } else if (newLastName === '') {
49              openAlert('Valor inválido', 'Por favor, ingrese un apellido');
50              return;
51          } else if (newPhone.length < 12) {
52              openAlert('Valor inválido', 'Por favor, ingrese un número de celular válido');
53              return;
54          } else if (email !== '' && !/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email)) {
55              openAlert('Valor inválido', 'Por favor, ingrese un correo electrónico válido');
56              return;
57          } else {
```

Figura 6.2: Corrección para la funcionalidad F12

6.1.2. Pruebas de integración

Este proceso consistió en verificar la conexión entre el frontend y el backend, no sólo validando que el backend respondiera adecuadamente ante las peticiones, sino que también el frontend estuviera listo para procesar las respuestas obtenidas o que realizara correctamente los llamados a las APIs.

En la Figura 4.10 del Capítulo 4 dedicado a la fase de "Diseño" del proyecto, se explicó que había una parte del frontend llamado "API" que estaba encargado de comunicarse directamente con el backend y gestionar la entrada y salida de información para que no solamente se hicieran los llamados a los endpoints con los datos correctos, sino que también se pudieran transformar los datos obtenidos desde el backend a la estructura que se manejaba dentro de la lógica de la aplicación. Las pruebas se dividieron en tres grupos:

- Sobre *usuarios*, incluyendo la obtención de los datos de un usuario en particular, la actualización de dicha información, la obtención de los datos de todos los miembros de una compañía, y el guardado del token de un dispositivo asociado a un usuario.
- Sobre *compañías*, que comprende tanto la obtención de los datos de una compañía como la actualización de los mismos.
- Sobre *contactos*, que involucra la obtención de todos los contactos de una compañía, la creación de un contacto, así como su actualización.

Para ejecutar estas pruebas se usó Jest, que es un framework para realizar pruebas en JavaScript. Es capaz de trabajar con proyectos que usan Babel, TypeScript, Node, React, Angular, Vue, entre otros [88]. Una de las principales características de Jest es que permite construir pruebas unitarias para el frontend de una manera más sencilla de lo que solía ser, ya que anteriormente se requería de una configuración extensa y lenta para llevar a cabo este tipo de pruebas [89].

En la Figura 6.3 se muestra el código para probar las funciones asociadas a los usuarios. Se puede observar que se definen las siguientes pruebas:

- "Obtiene los datos del usuario", que verifica si los datos obtenidos son los mismos de un usuario establecido.
- "Actualiza los datos del usuario", que prueba si al llamar la función de actualizar, estos datos realmente han cambiado.
- "Guarda el dispositivo del usuario", que se asegura de que la respuesta del endpoint es exitosa.
- "Obtiene los datos de los miembros de una compañía", que verifica si los datos obtenidos de los usuarios de una compañía dada son los que se espera.

```typescript
// Pruebas de los endpoints de usuarios
import {getUser, updateUser, saveUserDevice, getTeam} from './users';
import {userID, companyID, user, team} from './test-data';

describe('Pruebas de usuarios', () => {
    test('Obtiene los datos del usuario', async () => {
        const expectedUser: User = user;
        const obtainedUser: User = await getUser(userID);
        expect(obtainedUser).toEqual(expectedUser);
    });

    test('Actualiza los datos del usuario', async () => {
        const oldUser: User = user;
        const newUser: User = {...oldUser, lastName: 'Paladines 2'};

        await updateUser(newUser);

        const expectedUser: User = newUser;
        const obtainedUser: User = await getUser(userID);

        expect(obtainedUser).toEqual(expectedUser);
    });

    test('Guarda el dispositivo del usuario', async () => {
        const deviceID: string = 'token_de_prueba';
        const savedSuccessfully: boolean = await saveUserDevice(userID, deviceID, 'android');
        expect(savedSuccessfully).toBe(true);
    });

    test('Obtiene los datos de los miembros de una compañía', async () => {
        const expectedTeam = new Set(team);
        const obtainedTeam = new Set(await getTeam(companyID));
        expect(obtainedTeam).toEqual(expectedTeam);
    });
});
```

Figura 6.3: Código para probar las funciones sobre usuarios

En la Figura 6.4 se muestran los resultados que se obtuvieron al ejecutar las pruebas sobre usuarios.

```
> piyion@0.0.1 test
> jest "__tests__/api/usersTests.ts"

 PASS  __tests__/api/usersTests.ts
  Pruebas de usuarios
    ✓ Obtiene los datos del usuario (481 ms)
    ✓ Actualiza los datos del usuario (644 ms)
    ✓ Guarda el dispositivo del usuario (414 ms)
    ✓ Obtiene los datos de los miembros de una compañía (954 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        4.48 s, estimated 9 s
Ran all test suites matching /__tests__\/api\/usersTests.ts/i.
```

Figura 6.4: Resultados de las pruebas sobre usuarios

En la Figura 6.5 se muestran los resultados que se obtuvieron al ejecutar las pruebas sobre compañías.

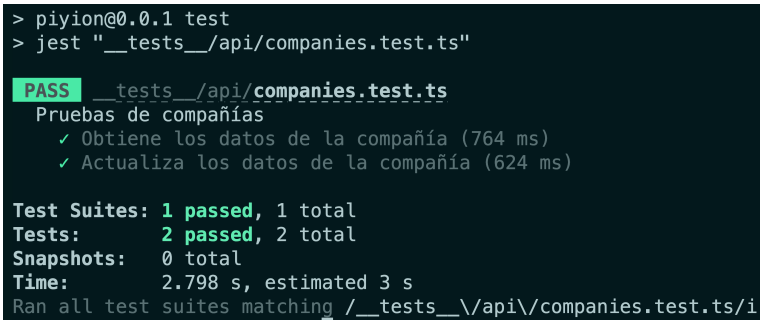


Figura 6.5: Resultados de las pruebas sobre compañías

En la Figura 6.6 se muestran los resultados que se obtuvieron al ejecutar las pruebas sobre contactos.

Figura 6.6: Resultados de las pruebas sobre contactos

6.1.3. Pruebas unitarias

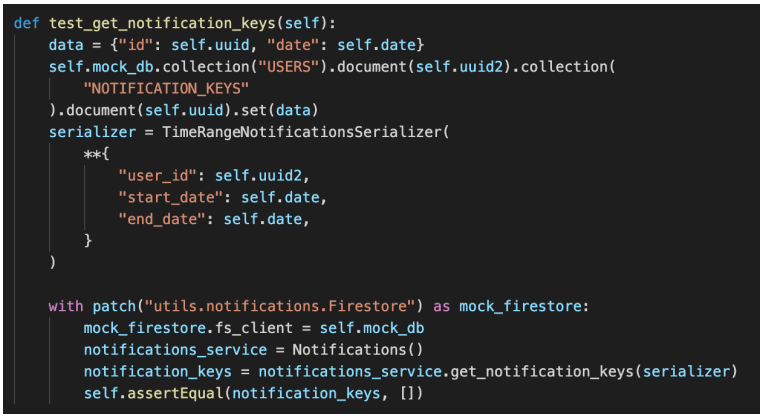
Este proceso consistió en verificar que dados unos casos de prueba, se recorriera la mayor cantidad de líneas de código posibles (cobertura de sentencia). Debido a que el backend se construyó tanto para la aplicación web como para la aplicación móvil, además del crecimiento que presenta Piyion como empresa, no se pudo probar todo el código del backend con pruebas unitarias. Sin embargo, todas las funcionalidades se prueban (mediante pruebas manuales) antes de ir a producción.

Con ayuda de las pruebas unitarias se logró una gran cobertura del código usando la herramienta Coverage [90], la cual permite medir la efectividad de las pruebas unitarias en función del número de líneas que fueron ejecutadas sobre el total de las mismas. En la Figura 6.7 se puede observar el código fuente del método *get_notification_keys* del módulo de notificaciones.

```python
def get_notification_keys(self, payload: TimeRangeNotificationsSerializer):
    # get notification keys from a given time range

    base = (
        self.db.collection("USERS")
        .document(payload.user_id)
        .collection("NOTIFICATION_KEYS")
    )

    if payload.start_date:
        base = base.where("date", ">", payload.start_date)

    if payload.end_date:
        base = base.where("date", "<", payload.end_date)

    notification_keys = [doc.to_dict().get("id") for doc in base.stream()]
Nicolás Delgado, 2 weeks ago • Nicolasdelgado/sc 2388/servicio web que permita c…
    return notification_keys
```

Figura 6.7: Código fuente del método *get_notification_keys* del módulo de notificaciones

Luego, se procedió a realizar la prueba con el código presentado en la Figura 6.8, en donde se puede detallar que en la subcolección *NOTIFICATION_KEYS*, perteneciente a cada usuario, se almacena un identificador (de la notificación) y una fecha.



```python
def test_get_notification_keys(self):
    data = {"id": self.uuid, "date": self.date}
    self.mock_db.collection("USERS").document(self.uuid2).collection(
        "NOTIFICATION_KEYS"
    ).document(self.uuid).set(data)
    serializer = TimeRangeNotificationsSerializer(
        **{
            "user_id": self.uuid2,
            "start_date": self.date,
            "end_date": self.date,
        }
    )

    with patch("utils.notifications.Firestore") as mock_firestore:
        mock_firestore.fs_client = self.mock_db
        notifications_service = Notifications()
        notification_keys = notifications_service.get_notification_keys(serializer)
        self.assertEqual(notification_keys, [])
```

Figura 6.8: Prueba unitaria del método *get_notification_keys*

Este método es esencial para solucionar el problema presentado en la sección 5.5.3, ya que permite filtrar los identificadores de las notificaciones de cada usuario dependiendo de la fecha en que se crearon. El método recibe un objeto, el cual tiene el *id* del usuario, y como campos opcionales, la fecha de inicio y la fecha final con las cuales se desea filtrar. Si no se proporcionan estas fechas, entonces retornaría todos los *id*'s de las notificaciones que se han creado.

La prueba presentada en la Figura 6.8 almacena en una base de datos simulada el *id* de la notificación y la fecha de su creación (*self.date*). Posteriormente, se crea un objeto con un *id* provisional para el usuario y fechas de inicio y fin (ambos *self.date*). Luego se le indica que use

la base de datos simulada como la base de datos del método a probar, se inicializa el módulo de notificaciones y se pasa el objeto como argumento del método. Al final se verifica que lo retornado por el método sea una lista vacía, ya que ningún elemento en la base de datos simulada se encuentra dentro del rango de fechas especificado de acuerdo a la lógica detallada en el código de la Figura 6.7. En el Anexo #6 se muestran los resultados de la cobertura de rama.

Todas las funcionalidades nuevas que fueron implementadas para Piyion fueron probadas de una u otra manera. Gracias a la herramienta Coverage, se logró una **cobertura de sentencia** del 81 % a través de pruebas unitarias. Cada una de las pruebas valida el correcto funcionamiento de la función a probar. En la Figura 6.9 se pueden apreciar todos los archivos que fueron probados, junto con el porcentaje individual de cobertura de cada uno de ellos.

```
Name                                                                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------------------------------------
src/main/python/communication_proxy/application/communication_proxy.py                43     16    63%   42–51, 57–7
src/main/python/communication_proxy/application/proxy_receiver.py                    86     50    42%   37–75, 81–1
src/main/python/communication_proxy/application/proxy_sender.py                     111     92    17%   34–35, 43–7
src/main/python/communication_proxy/domain/proxy_controller_repository.py             8      0   100%
src/main/python/infraestructure/dyalogo/dyalogo_adapter.py                           44     33    25%   32–39, 46–6
src/main/python/infraestructure/firestore/cp_controller.py                          115     73    37%   51, 60, 62–
src/main/python/infraestructure/firestore/firestore_connection.py                    11      1    91%   33
src/main/python/infraestructure/routes/agent_routes.py                               12      0   100%
src/main/python/infraestructure/routes/company_routes.py                            239     59    75%   165–185, 20
src/main/python/infraestructure/routes/contact_routes.py                             64      6    91%   42–44, 52,
src/main/python/infraestructure/routes/conversation_routes.py                        78      4    95%   129, 144–14
src/main/python/infraestructure/routes/dyalogo_routes.py                             31      0   100%
src/main/python/infraestructure/routes/fcm_routes.py                                 77      4    95%   40, 59, 78,
src/main/python/infraestructure/routes/gupshup_webhook.py                            33      1    97%   54
src/main/python/infraestructure/routes/instagram_webhook.py                          33      2    94%   36, 68
src/main/python/infraestructure/routes/message_routes.py                             34      0   100%
src/main/python/infraestructure/routes/messenger_webhook.py                          31      1    97%   37
src/main/python/infraestructure/routes/rates_routes.py                               25      0   100%
src/main/python/infraestructure/routes/telegram_webhook.py                           33      4    88%   51–54
src/main/python/infraestructure/routes/user_routes.py                               123      0   100%
src/main/python/models/company.py                                                    34      0   100%
src/main/python/models/contact.py                                                    19      0   100%
src/main/python/models/conversation.py                                               58      0   100%
src/main/python/models/device.py                                                     13      0   100%
src/main/python/models/message.py                                                    31      3    90%   53–55
src/main/python/models/rate.py                                                       11      3    73%   28–30
src/main/python/models/user.py                                                       30      0   100%
src/main/python/piyion/application/piyion.py                                        274     46    83%   246–258, 26
src/main/python/serializers/company_serializers.py                                   47      0   100%
src/main/python/serializers/contact_serializer.py                                    18      0   100%
src/main/python/serializers/conversation_serializers.py                              11      0   100%
src/main/python/serializers/dyalogo_serializers.py                                    9      0   100%
src/main/python/serializers/fcm_serializers.py                                       83      0   100%
src/main/python/serializers/global_serializers.py                                     6      0   100%
src/main/python/serializers/gupshup_serializers.py                                   24      0   100%
src/main/python/serializers/mails_serializers.py                                      9      0   100%
src/main/python/serializers/messenger_serializers.py                                 37      0   100%
src/main/python/serializers/telegram_serializers.py                                  42      0   100%
src/main/python/serializers/user_serializers.py                                      29      0   100%
src/main/python/targets/dialogflow/dialogflow.py                                     29     18    38%   34–49, 54–6
src/main/python/targets/domain/assistant_repository.py                                3      0   100%
src/main/python/targets/domain/target_repository.py                                   9      0   100%
src/main/python/targets/watson/watson.py                                             23      0   100%
src/main/python/utils/class_utils.py                                                  7      2    71%   21–22
src/main/python/utils/collections_utils.py                                            7      7     0%   13–25
src/main/python/utils/decorators.py                                                  10      0   100%
src/main/python/utils/email_templates/company_request.py                              7      0   100%
src/main/python/utils/exceptions.py                                                  16      0   100%
src/main/python/utils/firebase_utils.py                                              13      2    85%   34–35
src/main/python/utils/get_file_path_telegram.py                                      16      2    88%   33–34
src/main/python/utils/notifications.py                                              100      8    92%   76, 82, 164
src/main/python/utils/publisher.py                                                   12      6    50%   24–25, 28–3
src/main/python/utils/user_validations.py                                             5      0   100%
-----------------------------------------------------------------------------------------------------------------
TOTAL                                                                               2273    443    81%


[INFO]  Removing target directory /Users/ninustech/Desktop/Ninus/omnicanality_backend/target
------------------------------------------------------------
BUILD SUCCESSFUL
------------------------------------------------------------
```

Figura 6.9: Resultado final de pruebas unitarias sobre el backend

6.2. Pruebas no funcionales

Teniendo en cuenta los requisitos no funcionales definidos al inicio del proyecto y los límites de tiempo establecidos para el desarrollo de la aplicación, se decidió darle prioridad a la usabilidad y a realizar las pruebas correspondientes para su validación. Sin embargo, en los anexos #7 y #8 se muestran los detalles sobre las pruebas de rendimiento y escalabilidad, respectivamente.

6.2.1. Proceso

Las pruebas de usabilidad permiten verificar que la relación entre el producto y los usuarios es satisfactoria, ya que finalmente, son ellos los que aprovechan sus utilidades para lo que necesiten llevar a cabo. En este sentido, tomando como referencia los principales componentes de calidad definidas en la sección 2.1.2.11, se plantearon algunas preguntas que pudieran medir la calidad de la aplicación en términos de la relación con el usuario:

- ¿Consideras que es fácil aprender a usar la aplicación? (*aprendibilidad*)
- ¿Consideras que se necesitan pocos pasos para realizar una tarea? (*eficiencia*)
- ¿Puedes recordar fácilmente cómo usar la aplicación? (*memorabilidad*)
- ¿Cometiste pocos errores al usar la aplicación? (*errores*)
- ¿Te gustó la aplicación? (*satisfacción*)
- ¿Tienes alguna sugerencia?

Las cinco primeras preguntas se evaluaron en una escala de 1 a 5 donde 1 significa “Totalmente en desacuerdo”, 2 “En desacuerdo”, 3 “Neutral”, 4 “De acuerdo”, y 5 “Totalmente de acuerdo”. La última es una pregunta abierta con la que se busca retroalimentación especial por parte del usuario que no puede ser medida con las otras preguntas, y es opcional. Adicionalmente, se solicitó la edad y ocupación de los encuestados para analizar los resultados teniendo en cuenta las características demográficas.

6.2.2. Población

Para llevar a cabo las pruebas de usabilidad se buscó que la población fuera parte de los usuarios finales de la aplicación. Por esto, con la ayuda de la empresa Piyion se logró contactar a algunos de ellos para que realizaran dicha evaluación. Adicionalmente, se pidió participar en la prueba a otras personas que actualmente no son usuarios de Piyion, pero que pudieran brindar una perspectiva general con respecto a la aplicación. En total, fueron 24 personas las que respondieron la encuesta.

En la Figura 6.10 se muestra la distribución de edad de las personas que realizaron la prueba. En ella se observa que la mayoría de los encuestados está en el rango de edad de 20 a 29 años.

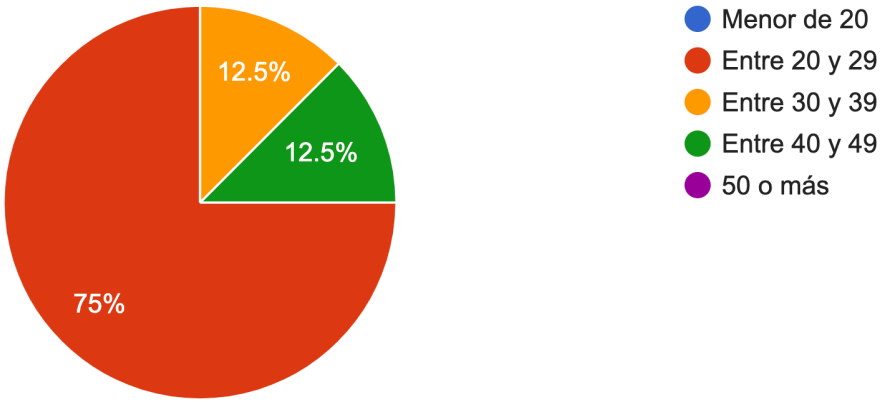


Figura 6.10: Edad de los encuestados

Por otro lado, en la Figura 6.11 se muestra la distribución de las ocupaciones de las personas encuestadas.

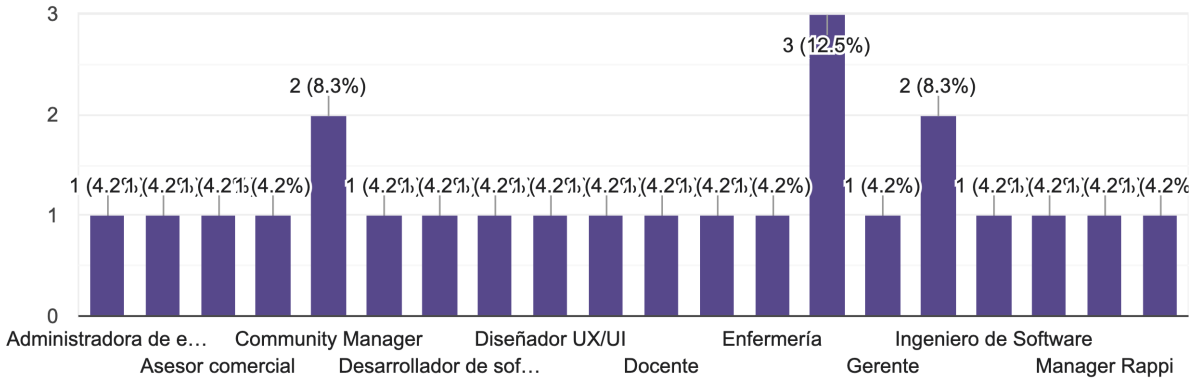


Figura 6.11: Ocupación de los encuestados

6.2.3. Resultados

Luego de haber realizado las pruebas a los usuarios, se obtuvo los resultados que se muestran a continuación.

- **Aprendibilidad**. Como se muestra en la Figura 6.12, el 75 % de los usuarios respondieron que estaban totalmente de acuerdo, y el 25 % que estaban de acuerdo. Esto indica que la facilidad para aprender a usar la aplicación es alta.

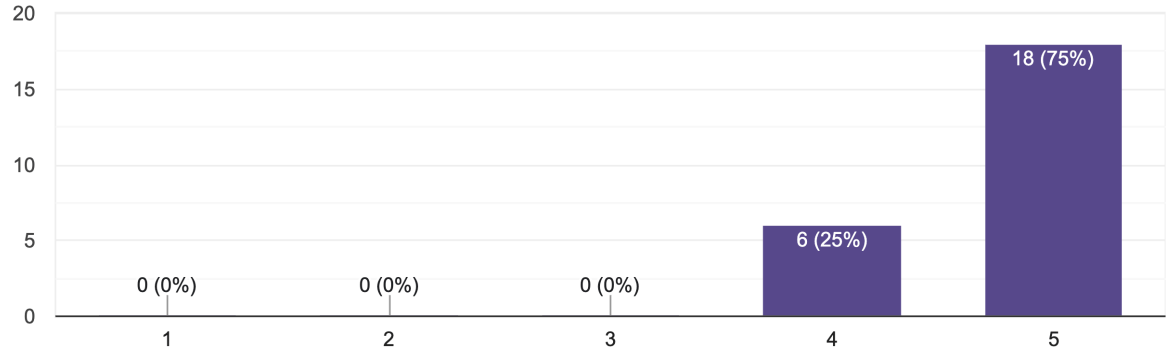


Figura 6.12: Resultados sobre aprendibilidad

- **Eficiencia**. Como se muestra en la Figura 6.13, el 45.8 % de los usuarios respondieron que estaban totalmente de acuerdo, el 45.8 % que estaban de acuerdo, y el 8.3 % de manera neutral. Esto quiere decir que en términos generales la aplicación es eficiente para realizar las tareas más comunes pero todavía hay aspectos por mejorar.

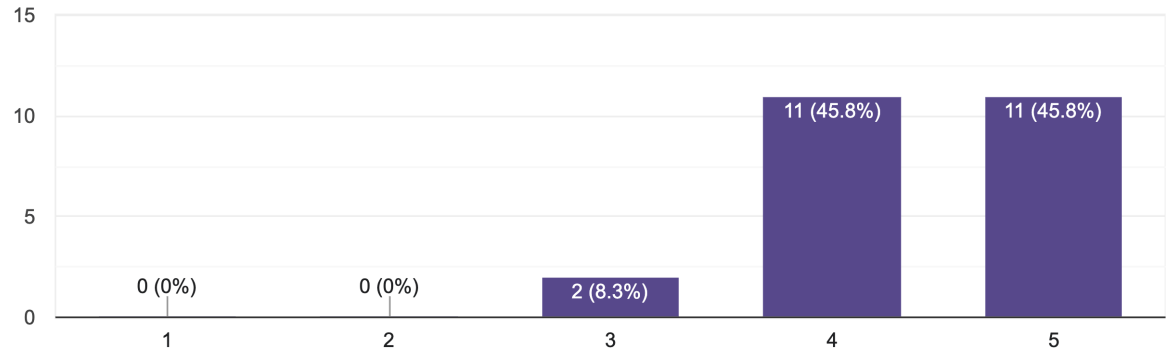


Figura 6.13: Resultados sobre eficiencia

- **Memorabilidad**. Como se muestra en la Figura 6.14, el 70.8 % de los usuarios respondieron que estaban totalmente de acuerdo, el 25 % que estaban de acuerdo, y el 4.2 % de manera neutral. Esto indica que la mayoría de los usuarios pueden recordar cómo usar la aplicación pero para algunos no es tan sencillo, lo que se tendrá en cuenta para mejorar la forma en que se presentan las interfaces y la información en general.

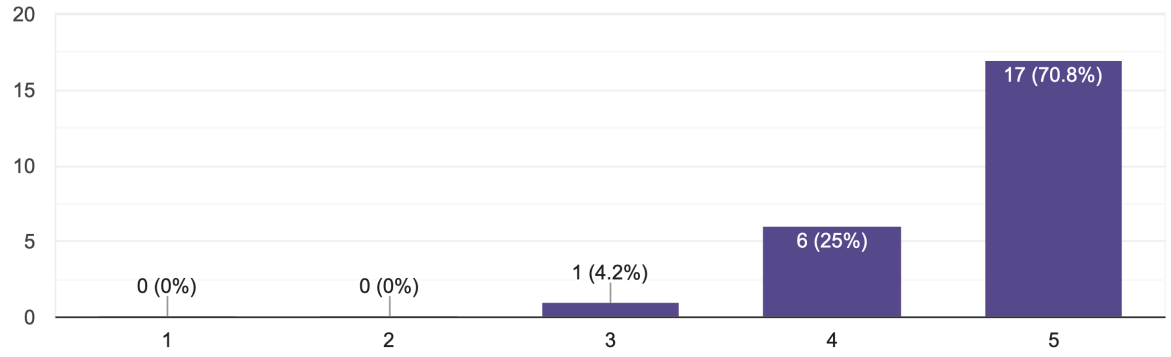


Figura 6.14: Resultados sobre memorabilidad

- **Errores**. Como se muestra en la Figura 6.15, el 33.3 % de los usuarios respondieron que estaban totalmente de acuerdo, el 58.3 % que estaban de acuerdo, y el 8.3 % de manera neutral. Esto quiere decir que gran parte de los usuarios no cometieron tantos errores al usar la aplicación, pero es necesario revisar este aspecto para poder mejorarlo.

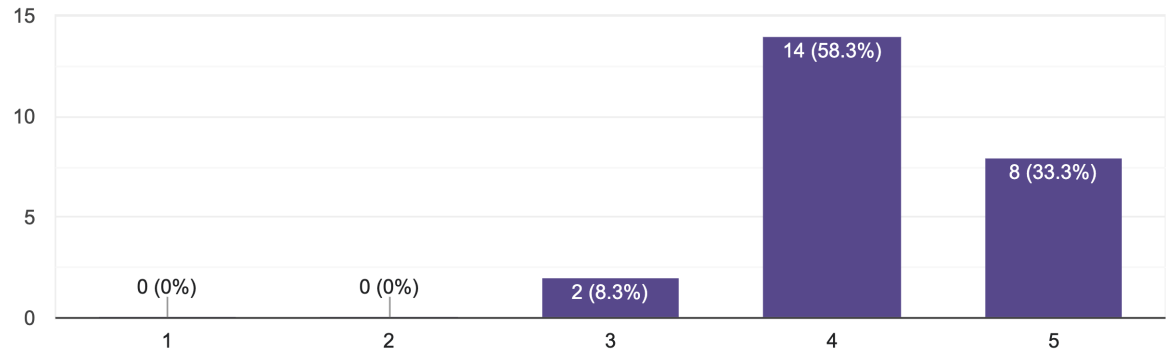


Figura 6.15: Resultados sobre errores

- **Satisfacción**. Como se muestra en la Figura 6.16, el 83.3 % de los usuarios respondieron que estaban totalmente de acuerdo, y el 16.7 % que estaban de acuerdo. Esto indica que en términos generales los usuarios se sienten satisfechos con la aplicación.

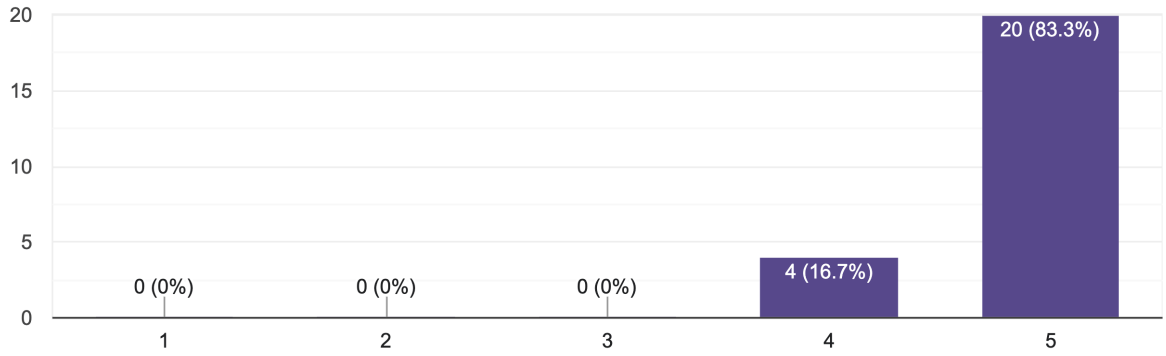


Figura 6.16: Resultados sobre satisfacción

Los resultados anteriores y las sugerencias de los usuarios ayudaron en gran medida a detectar algunos problemas en términos de usabilidad. Estos son aspectos que se tienen en cuenta para mejorar y ofrecer un servicio que sea agradable para ellos.

Capítulo 7

Trabajo Futuro

Aunque se cumplió con lo planteado inicialmente para el desarrollo del proyecto, hay algunos aspectos que se pueden llevar a cabo en un futuro para mejorar la calidad de la aplicación o para ampliar la gama de servicios que esta ofrece.

7.1. Frontend

7.1.1. Funcionalidades

Por restricciones de tiempo se descartó la implementación de algunas funcionalidades que no eran vitales para el funcionamiento de la aplicación pero que aportaban significativamente a su calidad. Algunas de ellas son:

- **Cambio de idioma**. Debido a que actualmente la aplicación móvil sólo está disponible en español, se pretende que se amplíen las posibilidades para poder usarla con otros idiomas, principalmente inglés.
- **Registro de usuario**. Hasta el momento sólo pueden usar la aplicación quienes ya tengan una cuenta creada en Piyion, sin embargo es necesario implementar esta funcionalidad para dispositivos móviles.
- **Recuperación de cuenta**. De manera similar al registro de usuario, esta es una funcionalidad que actualmente no es posible llevar a cabo en la aplicación móvil y que es necesario implementar.
- **Conteo de mensajes no leídos**. Esta es una funcionalidad que aún no se encuentra implementada ni en la aplicación web ni en la aplicación móvil, pero se pretende construir pronto ya que es un aspecto importante para el manejo de los chats.
- **Visualización de imagen completa**. Hasta el momento solo es posible mirar una imagen enviada o recibida como mensaje en el tamaño que se muestra en la conversación, pero se pretende implementar una manera de ver dicha imagen de forma ampliada similar a como se hace en WhatsApp.
- **Animaciones y transiciones**. Por restricciones de tiempo no se le dio prioridad a las animaciones que ocurren al cambiar entre pantallas o al presionar un botón, pero es algo que se pretende realizar en un futuro para hacer la aplicación más agradable ante el usuario.

- **Visualización de estadísticas**. Por cuestiones de tiempo no se pudo diseñar ni implementar las estadísticas de una compañía dentro de la aplicación móvil. Sin embargo, es algo que se pretende hacer en un futuro ya que añade mucho valor el que un administrador pueda ver las estadísticas de los agentes y conocer su rendimiento.

7.1.2. Pruebas

Teniendo en cuenta las limitaciones de tiempo se realizaron sólo pruebas funcionales de sistema y de integración. Sin embargo, se considera que es de gran importancia llevar a cabo otros tipos de pruebas para buscar garantizar un software de alta calidad.

- **Pruebas de componentes de interfaz**. Estas se usarían para verificar que cada componente UI funciona según lo esperado, como los botones, alertas, campos de texto, etc.
- **Pruebas de rendimiento**. Se pretende realizar en un futuro pruebas de carga y pruebas de estrés con el fin de garantizar la calidad del software en un aspecto tan importante como lo es el tiempo de respuesta para los usuarios.

7.2. Backend

7.2.1. Funcionalidades

Dada la naturaleza finita de este proyecto, no se implementaron algunas funcionalidades que podrían generar valor a la aplicación; sin embargo, a continuación se listan algunas de ellas que se pueden desarrollar en un futuro.

- **Notificaciones para mejorar la experiencia de administración**. Para los administradores podría ser de gran utilidad contar con notificaciones adicionales que les permitieran saber cuándo los agentes no han respondido mensajes de forma eficaz (dados ciertos parámetros como tiempos de respuesta), o cuando se reciben conversaciones y no hay agentes disponibles.
- **Comunicación con nuevos canales de atención**. A mediano plazo sería de gran utilidad contar con nuevas integraciones de canales de atención, pues podría ser atractivo para posibles nuevos clientes. Un ejemplo de esto es el uso masivo de iMessage en el mercado americano.
- **Desarrollo de una herramienta propia para crear chatbots**. Actualmente se hace uso de diferentes herramientas para la creación de chatbots que los clientes pueden integrar, como lo son Dyalogflow y Watson; sin embargo, se espera poder implementar una herramienta propia, que permita no solo mejorar las funcionalidades que se ofrecen a los clientes en este aspecto, si no mejorar el lucro, pues estas herramientas se cobran por consumo y la mayor parte de dicho cobro queda en los proveedores externos.
- **Estadísticas en tiempo real**. Algo que ayudaría al seguimiento y supervisión de un agente serían las estadísticas en tiempo real: saber cuántas ventas potenciales tiene en proceso; la

calificación en tiempo real que le dan los clientes; entre otras métricas. Sin embargo, por limitaciones de tiempo, no se ha empezado la implementación de esta funcionalidad que daría mucho valor a los clientes.

7.2.2. Pruebas

Debido a las limitaciones de tiempo, los cambios de última hora que fue necesario realizar debido a que se desbordaban los recursos de las APIs por las notificaciones y otros aspectos, no se pudo lograr un coverage del 100 % con pruebas unitarias. Por tanto, sería conveniente finalizarlas y mantener una cobertura de sentencia más cercana al 100 %.

Conclusiones

En este trabajo de grado se implementó una aplicación móvil que apoya la realización de las tareas llevadas a cabo por los usuarios de Piyion. Para la construcción de esta aplicación se siguió los pasos del ciclo de desarrollo de software adaptados a las metodologías ágiles, que permitió obtener un producto que cumple con los requerimientos planteados al inicio del proyecto.

Se identificó la necesidad de los usuarios sobre una herramienta que fuera más accesible y que permitiera ejecutar las acciones que comúnmente se hacían desde la aplicación web del sistema. De esta manera, la aplicación funciona como un complemento necesario para la gestión de la atención al cliente y brinda a sus usuarios diversas posibilidades de mejorar su relación con dichos clientes.

En la etapa de obtención de requisitos se evidenció la importancia de una buena definición de las necesidades de los usuarios para que el software construido pueda responder adecuadamente ante ellas. Cabe resaltar que aunque ya se contaba con una plataforma web del sistema y ya se habían identificado los requerimientos para su construcción, fue necesario establecer los criterios de aceptación enfocados a los dispositivos móviles, debido a que siendo un entorno diferente se requiere hacer ciertas distinciones.

Al diseñar la aplicación se buscó definir una estructura del sistema acorde con los requerimientos definidos y usando las prácticas recomendadas para que todo funcionara de manera correcta y eficiente. Además, los diseños de interfaces respondieron adecuadamente a las necesidades establecidas y permitieron aterrizar las ideas sobre la estructura visual de la aplicación.

Es importante destacar que aunque ya se encontraba desarrollada una plataforma web, en el momento de implementar la aplicación se realizaron cambios relevantes para tener en cuenta la diversidad de dispositivos con los que los usuarios podían hacer uso del sistema. En este sentido, además de construir desde cero el frontend de la app, del lado del backend se agregaron varios procesos y se modificaron algunos otros con el fin de que el sistema pudiera responder ante las necesidades de los usuarios.

Las pruebas que se ejecutaron sobre la aplicación permitieron la identificación y corrección oportuna de errores. Además, con las pruebas de usabilidad fue posible conocer la percepción de los usuarios reales, lo que ayudó bastante para el mejoramiento de la relación producto-usuario.

Piyion se encuentra en constante crecimiento, por lo que esta primera versión de la aplicación se irá mejorando poco a poco al agregar nuevas funcionalidades o modificar algunas de las que ya existen, según las necesidades que se presenten en el mercado.

Bibliografía

[1] Hygger, "Agile Guide," 2021. Disponible en https://hygger.io/guides/agile.

[2] O. Goncharenko, "Flutter vs. React Native in 2022 - Detailed Framework Comparison," 2022. Disponible en https://brocoders.com/blog/flutter-vs-react-native.

[3] N. Raval, "TypeScript vs JavaScript: The Difference You Should Know," 2022. Disponible en https://radixweb.com/blog/typescript-vs-javascript.

[4] "Piyion," 2022. Disponible en https://piyion.com.

[5] ACM, "ACM Computing Classification System," *Computing Classification System*, 2012. Disponible en https://dl.acm.org/ccs.

[6] Hotmart, "Conoce los canales de comunicación más utilizados para enamorar a tus clientes," 2022. Disponible en https://hotmart.com/es/blog/canales-de-comunicacion.

[7] L. Cardozo, "¿Cómo elegir los tipos de canales de comunicación para la empresa?," 2020. Disponible en https://www.zenvia.com/es/blog/canales-de-comunicacion.

[8] C. Newberry, "33 estadísticas de Facebook que todo mercadólogo debe conocer en 2020," 2020. Disponible en https://blog.hootsuite.com/es/estadisticas-de-facebook.

[9] M. Pérez Wiesner, M. P. Fernández Martín, and F. López Muñoz, "El fenómeno de las redes sociales: evolución y perfil del usuario," 2014. Disponible en https://dialnet.unirioja.es/servlet/articulo?codigo=5126970.

[10] Callbell, "Multi-agent WhatsApp platforms," Disponible en https://www.callbell.eu/en/multi-agent-whatsapp-platforms.

[11] Outsystems, "What Is a Mobile Application?," 2022. Disponible en https://www.outsystems.com/glossary/what-is-mobile-application.

[12] L. Delia, P. Thomas, L. Corbalán, J. Fernandez Sosa, A. Cuitiño, G. Cáseres, and P. Pesado, *Development Approaches for Mobile Applications: Comparative Analysis of Features: Proceedings of the 2018 Computing Conference, Volume 2*, pp. 470–484. 01 2019.

[13] IBM, "What is software development?," 2021. Disponible en https://www.ibm.com/topics/software-development.

[14] Atlassian, "What is Agile?," *The Agile Coach*, 2021. Disponible en https://www.atlassian.com/agile.

[15] S. Peek, "What Is Mobile App Development?," 2021. Disponible en https://www.businessnewsdaily.com/5155-mobile-app-development.html.

[16] Microsoft, "Mobile Software Development Lifecycle," 2021. Disponible en https://docs.microsoft.com/en-us/xamarin/cross-platform/get-started/introduction-to-mobile-sdlc.

[17] 3D Prototype Designer, "What Is A Functional Prototype," *3D Printing & Rapid Prototyping*, 2015. Disponible en https://3d-printing-expert.com/what-is-a-functional-prototype.

[18] Ingsoftware, "Software prototyping," 2019. Disponible en https://www.ingsoftware.com/software-prototyping.

[19] A. W. Services, "Arquitecturas sin servidor," 2022. Disponible en https://aws.amazon.com/es/lambda/serverless-architectures-learn-more.

[20] G. Cloud, "¿Qué es cloud computing?," Disponible en https://cloud.google.com/learn/what-is-cloud-computing?hl=es.

[21] IBM, "What is software testing?," *IBM*, 2015. Disponible en https://www.ibm.com/topics/software-testing.

[22] L. Katara, "Levels of Testing in Software Testing," 2021. Disponible en https://qacraft.com/levels-of-testing-in-software-testing.

[23] T. Hamilton, "Functional Vs Non-Functional Testing – Difference Between Them," 2022. Disponible en https://www.guru99.com/functional-testing-vs-non-functional-testing.html.

[24] W. O. Sánchez *et al.*, "La usabilidad en Ingeniería de Software: definición y características," 2015. Disponible en http://www.redicces.org.sv/jspui/bitstream/10972/1937/1/2.LausabilidadenIngenieriadeSoftware-definicionycaracteristicas.pdf.

[25] J. Nielsen, "Usability 101: Introduction to Usability," 2012. Disponible en https://www.nngroup.com/articles/usability-101-introduction-to-usability.

[26] D. Aguilar, "Cómo las ventas por internet están salvando empresas en Colombia," 2020. Disponible en https://www.triario.co/blog/evolucion-ventas-por-internet-en-colombia.

[27] I. M. D. Andrade and C. Tumelero, "Increasing customer service efficiency through artificial intelligence chatbot," 2021. Disponible en https://www.emerald.com/insight/content/doi/10.1108/REGE-07-2021-0120/full/pdf?title=increasing-customer-service-efficiency-through-artificial-intelligence-chatbot.

[28] D. White, "HSBC deploys Dialogflow, easing call burden on policy experts," 2021. Disponible en https://cloud.google.com/blog/products/ai-machine-learning/hsbc-builds-an-internal-chatbot-to-answer-questions-on-policies.

[29] A. Gillis and B. George, "What is the Turing Test?," 2020. Disponible en https://www.techtarget.com/searchenterpriseai/definition/Turing-test.

[30] MessageBird, "MessageBird Website," 2022. Disponible en https://www.messagebird.com.

[31] B2Chat, "B2Chat Website," 2022. Disponible en https://www.b2chat.io.

[32] MessageBird, "Domino's Pizza increases incremental sales by 40 % with SMS campaigns," 2021. Disponible en https://www.messagebird.com/es/customer-stories/dominos.

[33] MessageBird, "Houm attracts 64 % more qualified leads and reduces response time," 2021. Disponible en https://www.messagebird.com/es/customer-stories/houm.

[34] Jean, "Testimonio de Jean, de ebike.es con Callbell," 2021. Disponible en https://anchor.fm/callbell/episodes/Jean-de-Ebike-es-e13docj.

[35] R. Rastreador.com, "Más de 7,6 millones de españoles se consideran adictos al móvil," 2019. Disponible en https://www.rastreator.com/sala-de-prensa/notas-de-prensa/2018-07-adiccion-movil-mas-de-siete-millones-adictos.aspx.

[36] A. Henao, "Testimonio de Anderson Henao, de Viajapues.co con Callbell," 2021. Disponible en https://anchor.fm/callbell/episodes/Anderson-de-Viajapues-e156cht.

[37] A. Gómez, "Testimonio de Andrés Gómez, de Fidalsa International Group con Callbell," 2021. Disponible en https://anchor.fm/callbell/episodes/Andres-de-Fidalsa-e1449kn.

[38] S. Lund, A. Madgavkar, J. Manyika, S. Smit, K. Ellingrud, M. Meaney, and O. Robinson, "The postpandemic economy: The future of work after COVID-19," 2021. Disponible en https://mck.co/3E9vIF8.

[39] L. Saborío Morales and L. F. Hidalgo Murillo, "Síndrome de Burnout," *Medicina Legal de Costa Rica*, vol. 32, pp. 119 – 124, 03 2015. Disponible en http://www.scielo.sa.cr/scielo.php?script=sci_arttext&pid=S1409-00152015000100014&nrm=iso.

[40] Scrum.org, "What is SCRUM?," *What is Scrum?*, 2021. Disponible en https://www.scrum.org/resources/what-is-scrum.

[41] IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering," *ISO/IEC/IEEE 29148:2018(E)*, p. 9, 2018.

[42] IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering," *ISO/IEC/IEEE 29148:2018(E)*, pp. 15–16, 2018.

[43] U. D. Institute, "What is UI design? A complete introductory guide," 2022. Disponible en https://www.uxdesigninstitute.com/blog/what-is-ui-design.

[44] C. M. University, "Software Architecture," 2022. Disponible en https://www.sei.cmu.edu/our-work/software-architecture.

[45] M. Shacklett, "IBM Watson: A cheat sheet," 2017. Disponible en https://www.techrepublic.com/article/ibm-watson-the-smart-persons-guide/.

[46] G. C. Documentation, "Dialogflow CX y ES," 2022. Disponible en https://cloud.google.com/dialogflow/docs.

[47] J. M. Alarcón, "Qué son los Webhooks, en qué se diferencian de una API REST y por qué deberías conocerlos," 2020. Disponible en https://bit.ly/3D7Md3I.

[48] C. Dyde, "What is software implementation?," 2021. Disponible en https://blogs.opentext.com/what-is-software-implementation.

[49] R. Native, "React Native: Learn once, write anywhere.," 2022. Disponible en https://reactnative.dev.

[50] M. Warcholinski, "React Native Apps – 10 Insanely Popular Examples," 2022. Disponible en https://brainhub.eu/library/react-native-apps.

[51] Ivozone, "7 Key Benefits of Choosing React Native for your Next Project," 2021. Disponible en https://invozone.com/blog/key-benefits-of-choosing-react-native-for-your-next-project.

[52] Fireart, "React Native vs Flutter: Which One is Better for 2023?," 2022. Disponible en https://bit.ly/40SCEQo.

[53] P. Barhate, "Cordova vs React Native: Choosing the Best Framework for Your App," 2022. Disponible en https://mobisoftinfotech.com/resources/blog/cordova-vs-react-native-comparison.

[54] TypeScript, "TypeScript is JavaScript with syntax for types," 2022. Disponible en https://www.typescriptlang.org.

[55] STX-Next, "What Is TypeScript? Pros and Cons of TypeScript vs. JavaScript," 2022. Disponible en https://www.stxnext.com/blog/typescript-pros-cons-javascript.

[56] GeeksForGeeks, "Advantages and Disadvantages of TypeScript over JavaScript," 2021. Disponible en https://www.geeksforgeeks.org/advantages-and-disadvantages-of-typescript-over-javascript.

[57] Sass, "SASS Basics," 2022. Disponible en https://sass-lang.com/guide.

[58] GeeksForGeeks, "What is the difference between SCSS and SASS?," 2022. Disponible en https://www.geeksforgeeks.org/what-is-the-difference-between-scss-and-sass.

[59] C. Mauri, "7 benefits of using SASS over conventional CSS," 2018. Disponible en https://www.mugo.ca/Blog/7-benefits-of-using-SASS-over-conventional-CSS.

[60] P. Team, "Flask Project," 2010. Disponible en https://flask.palletsprojects.com/en/2.2.x/.

[61] E. D. School, "¿Qué es Flask (Pyhon) y cuáles son sus principales ventajas?," 2021. Disponible en https://www.epitech-it.es/flask-python/.

[62] Acronis, "Google Cloud Platform: What it is, how to use it, and how it compares," 2019. Disponible en https://www.acronis.com/es-mx/blog/posts/google-cloud-platform/.

[63] Google, "Cloud Functions: Funciones en la nube para Firebase," 2021. Disponible en https://firebase.google.com/docs/functions.

[64] F. Documentation, "Firebase Cloud Messaging," Disponible en https://firebase.google.com/docs/cloud-messaging.

[65] Google, "¿Qué es Pub/Sub?," 2022. Disponible en https://cloud.google.com/pubsub/docs/overview?hl=es-419.

[66] Microsoft, "Publisher-Subscriber pattern," 2021. Disponible en https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber.

[67] Google, "Google Cloud Firestore," 2023. Disponible en https://cloud.google.com/firestore?hl=es.

[68] Google, "Cloud Storage: Almacenamiento de objetos para empresas de todos los tamaños," 2022. Disponible en https://cloud.google.com/storage#section-1.

[69] Google, "Secret Manager," 2022. Disponible en https://cloud.google.com/secret-manager.

[70] Google, "Cloud Scheduler," 2022. Disponible en https://cloud.google.com/scheduler?hl=es-419.

[71] A. Documentation, "Sending Notification Requests to APNs," Disponible en https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/sending_notification_requests_to_apns.

[72] B. Douglas, "How to build a CI/CD pipeline with GitHub Actions in four simple steps," 2022. Disponible en https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/.

[73] I. D. Foundation, "KISS (Keep it Simple, Stupid) - A Design Principle," 2020. Disponible en https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle.

[74] G. C. Services, "Firebase Console," 2022. Disponible en https://console.firebase.google.com/u/0/.

[75] E. Documentation, "What is Expo?," 2022. Disponible en https://docs.expo.dev/introduction/expo.

[76] C. Watson, "Cloud SOLID Part I: Cloud architecture and the single responsibility principle," 2017. Disponible en https://azure.microsoft.com/mediahandler/files/resourcefiles/e56cf87f-eb90-49bb-b40c-5c4247b7fa7c/Cloud-SOLID-The-single-responsibility-principle.pdf.

[77] Prettier, "Prettier - Code formatter," 2023. Disponible en https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode.

[78] A. Canesas, "TypeScript Coverage Report," 2022. Disponible en https://www.npmjs.com/package/typescript-coverage-report.

[79] Łukasz Langa, "`black` – the uncompromising python code formatter," 2018. Disponible en https://github.com/psf/black.

[80] P. Community, "`PEP 8` – style guide for python code," 2001. Disponible en https://www.python.org/dev/peps/pep-0008/.

[81] K. Samuel, "`PyFlakes` – passive checker of python programs," 2011. Disponible en https://github.com/PyCQA/pyflakes.

[82] T. P. Developers, "`pycodestyle` – simple python style checker in one python file," 2010. Disponible en https://github.com/PyCQA/pycodestyle.

[83] N. Batchelder, "`McCabe` – mccabe script for measuring code complexity," 1999. Disponible en https://github.com/PyCQA/mccabe.

[84] G. for Geeks, "Software Testing | Basics," 2022. Disponible en https://www.geeksforgeeks.org/software-testing-basics.

[85] R. Vats, "Decision Table Testing – Advantage and Scope," 2021. Disponible en https://www.upgrad.com/blog/decision-table-testing.

[86] T. Hamilton, "Boundary Value Analysis and Equivalence Partitioning Testing," 2022. Disponible en https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html.

[87] T. Hamilton, "State Transition Testing – Diagram Technique (Example)," 2022. Disponible en https://www.guru99.com/state-transition-testing.html.

[88] Jest, "Jest," 2022. Disponible en https://jestjs.io.

[89] N. Vaidya, "Jest Framework Tutorial: How to use it," 2020. Disponible en https://www.browserstack.com/guide/jest-framework-tutorial.

[90] N. Batchelder, "Coverage.py," 2021. Disponible en https://coverage.readthedocs.io/en/latest/.

Capítulo 8

Anexos

Anexo #1: Interfaces implementadas

La primera pantalla se muestra en la Figura 8.1, en donde es posible iniciar sesión mediante correo electrónico o con una cuenta de Google. La diferencia con respecto al diseño planteado inicialmente es que no se presentan las opciones para registrarse o para recuperar la cuenta al olvidar la contraseña, esto debido a que dichas funcionalidades no eran demasiado prioritarias y se dejaron propuestas para ser desarrolladas en un futuro cercano.

Figura 8.1: Pantalla implementada de inicio de sesión

En la Figura 8.2 se muestran la pantallas implementadas de la sección de “Equipo”. En la primera es posible observar la lista de los miembros del equipo identificando su nombre completo y su rol. Por otro lado, también se implementó un buscador en la parte superior que permite encontrar a un miembro por su nombre, apellido, o ambos. En la segunda imagen se observa la pantalla que muestra la información detallada de un miembro del equipo, como su correo electrónico, número de conversaciones que tiene asignadas, calificación, entre otros.

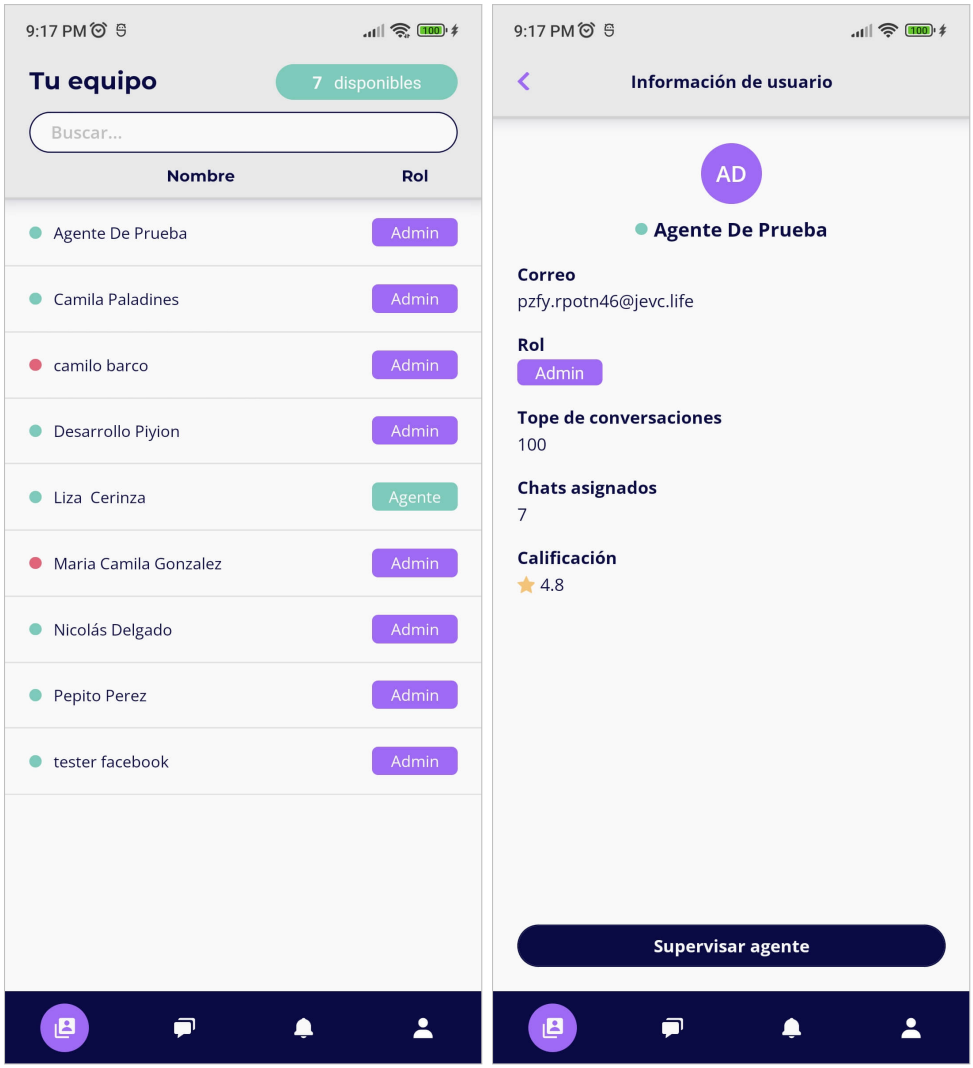


Figura 8.2: Pantallas implementadas de la sección de “Equipo”

En la Figura 8.3 se observan las pantallas de chats asignados y sin asignar, además de la visualización de una conversación en particular. En la primera se muestra una lista de chats que ya han sido asignados al usuario con una estructura similar a WhatsApp con el nombre del contacto asociado, el contenido del último mensaje, etc. En la segunda pantalla se muestra una lista de chats que aún no han sido asignados a un agente, esta vez sólo indicando el canal de donde proviene la solicitud de atención y hace cuánto tiempo se creó. En la tercera se puede observar una pantalla de conversación similar a WhatsApp, con el nombre del contacto en la cabecera, un espacio para enviar mensajes de texto y otros formatos (en la parte inferior), y por supuesto, los mensajes ordenados por tiempo de llegada o envío.

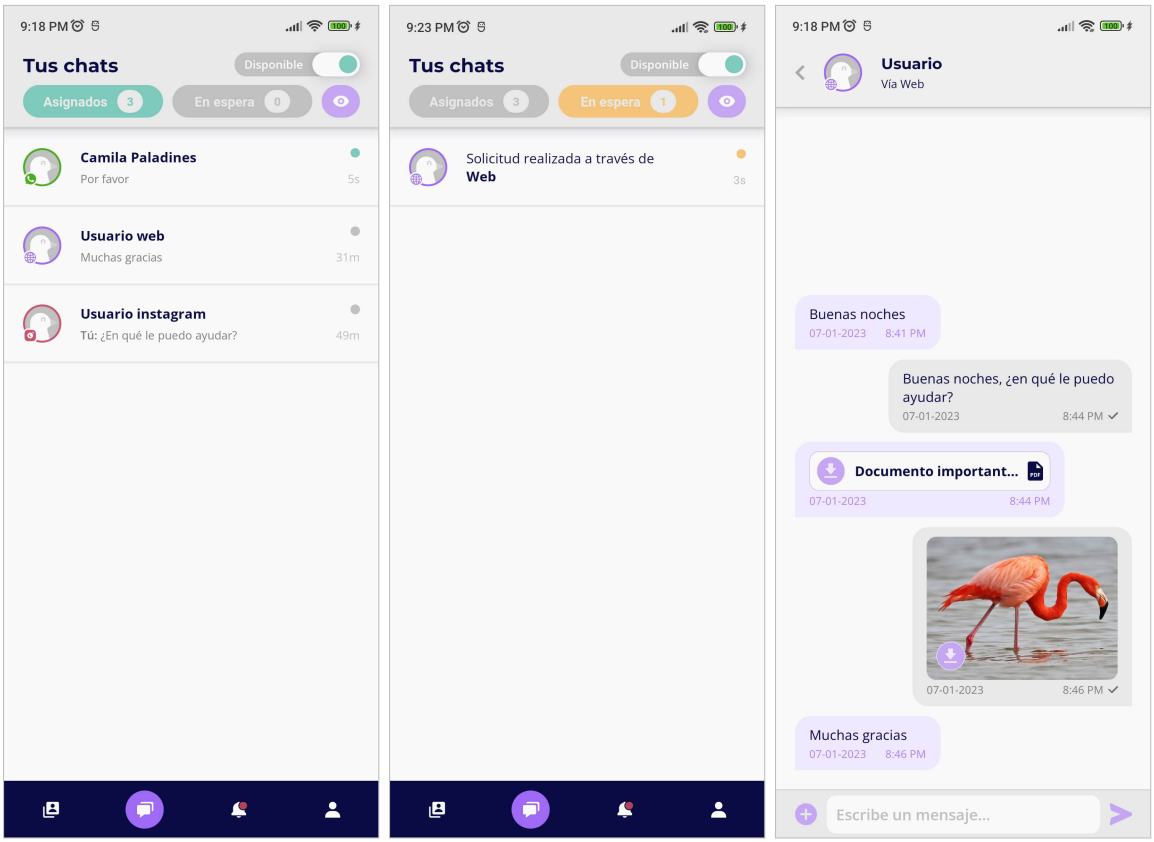


Figura 8.3: Pantallas implementadas sobre chats y conversación

En la Figura 8.4 se muestran las pantallas para la supervisión del equipo. La primera permite ver la lista de los miembros (disponibilidad, nombre completo y número de chats asignados), en la segunda se puede ver la lista de los chats asociados a un miembro, y en la tercera los mensajes de uno de esos chats, esta vez sin la posibilidad de enviar mensajes.

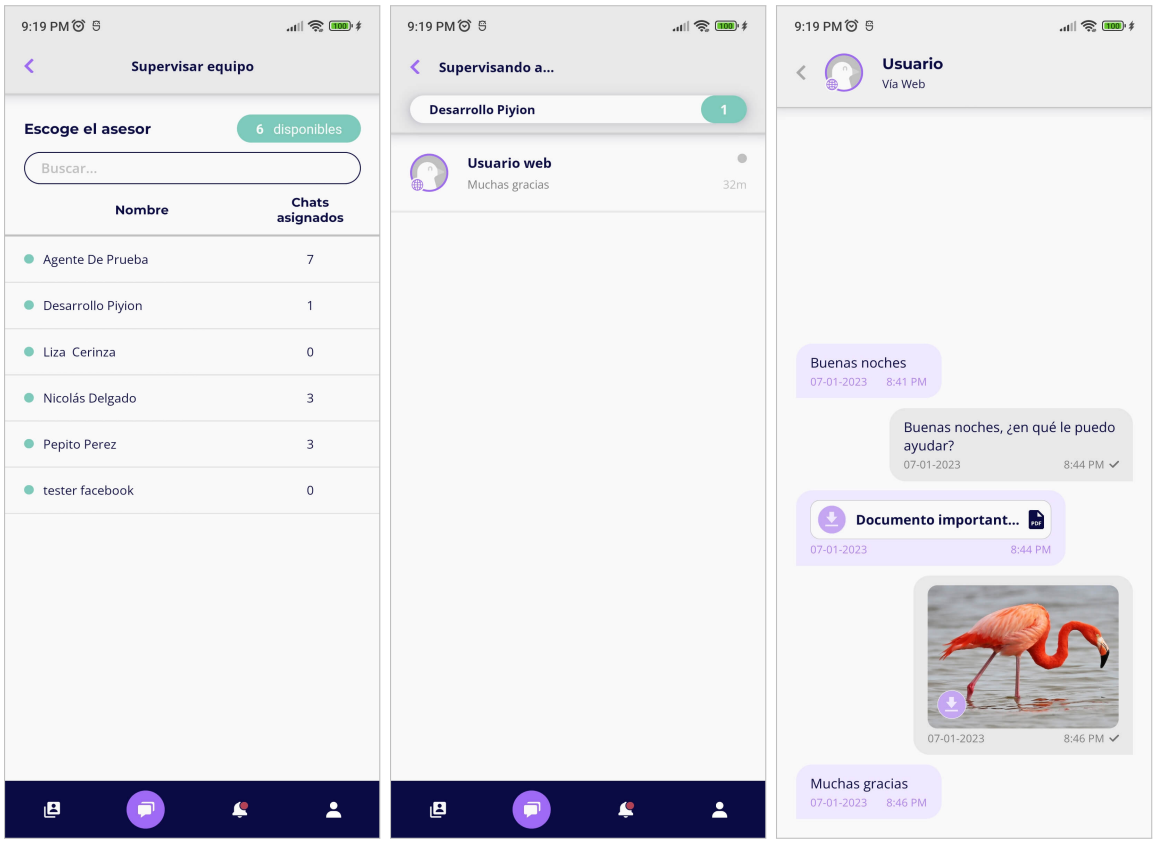


Figura 8.4: Pantallas implementadas sobre supervisión del equipo

En la Figura 8.5 se muestran las pantallas que permiten, en primer lugar, visualizar los datos de un contacto asociado a una conversación, como el nombre, correo electrónico, número de celular, entre otros. Además, también se facilitan aquí algunos botones útiles como el de reasignar la conversación o finalizarla si es el caso. Por otro lado, también se puede apreciar la pantalla que permite la edición de dicho contacto.

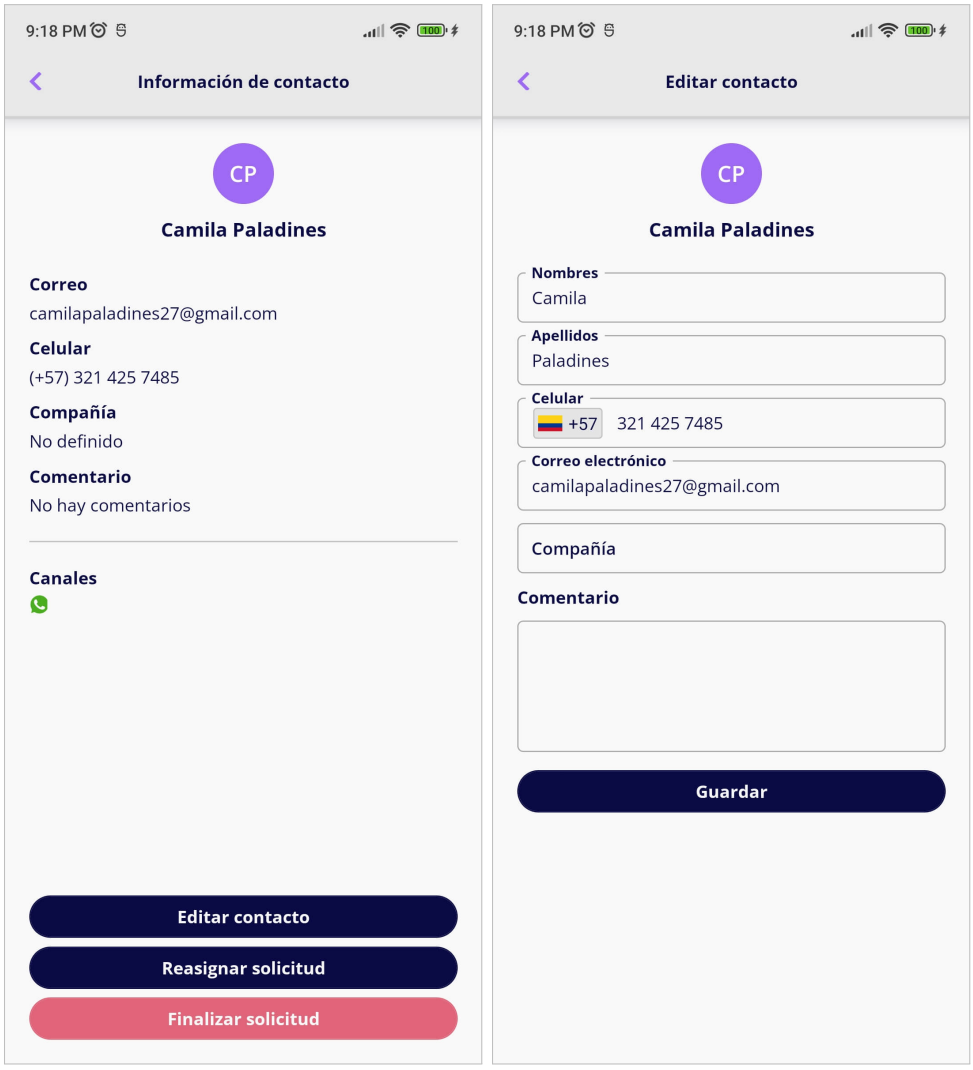


Figura 8.5: Pantallas implementadas sobre contactos

En la Figura 8.6 se muestran las pantallas correspondientes a las funcionalidades de asignar y reasignar conversaciones. En ellas es posible visualizar una lista de los agentes (miembros del equipo) disponibles que pueden tomar la conversación.

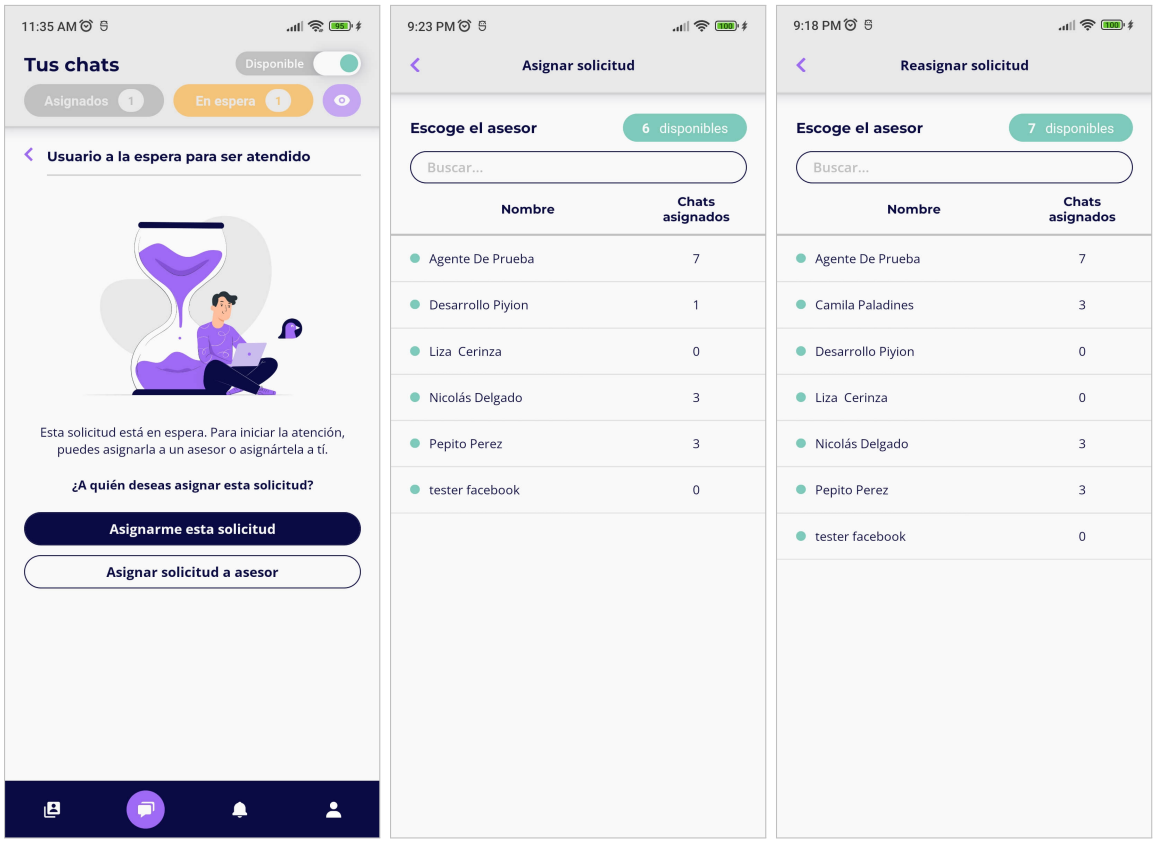


Figura 8.6: Pantallas implementadas sobre asignar y reasignar

En la Figura 8.7 se muestran las pantallas correspondientes a las notificaciones. En la primera se puede observar el panel de notificaciones como sección en la app, indicando un título, una descripción, hace cuánto tiempo ocurrió y un círculo que indica que aún no había sido leída. En la segunda y tercera pantalla se aprecia la manera en que aparecen las notificaciones push cuando la aplicación no se está corriendo en primer plano.

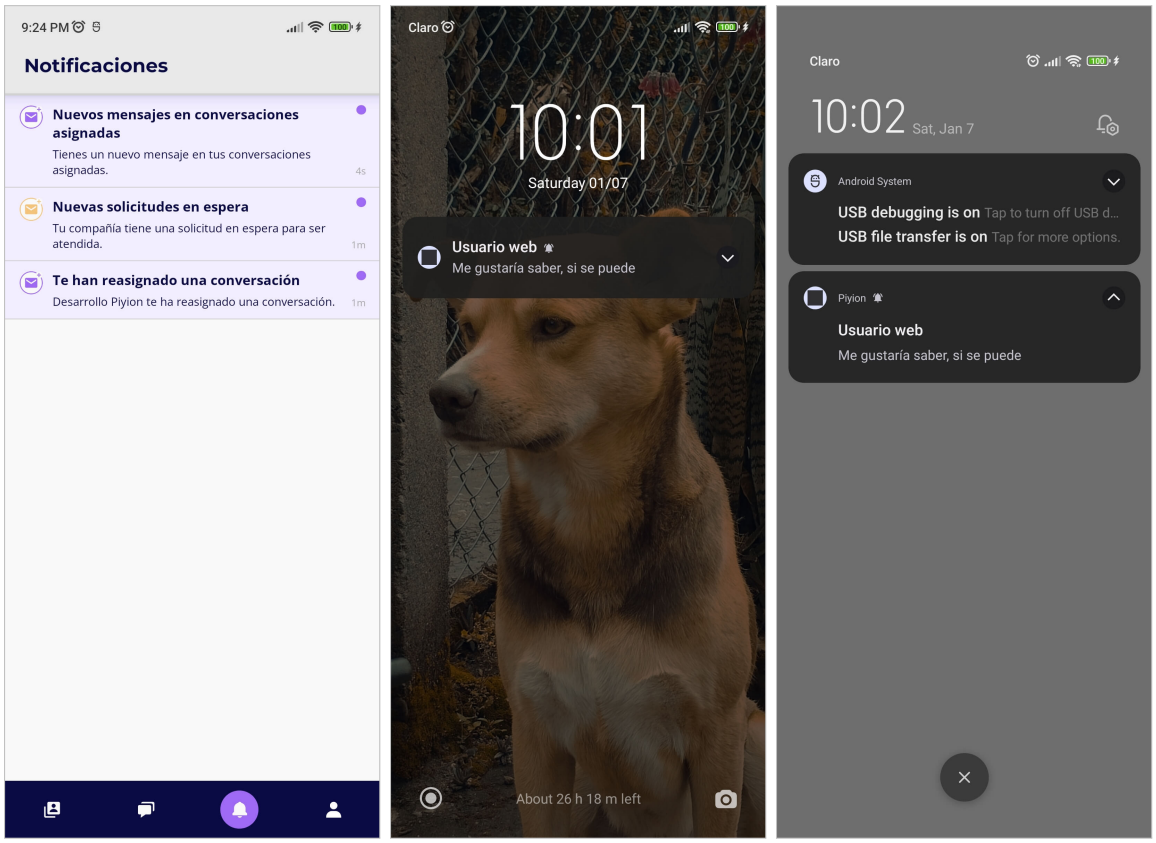


Figura 8.7: Pantallas implementadas sobre notificaciones

En la Figura 8.8 se muestran las pantallas correspondientes a las configuraciones del usuario y de la compañía. En ellas es posible modificar los datos tanto del usuario (nombres, apellidos y celular) como de la compañía (nombre, ID oficial, país y página web). Además, en la pantalla principal de la sección (primera imagen) se encuentra la opción de cerrar sesión y navegar hasta la pantalla de inicio de sesión.

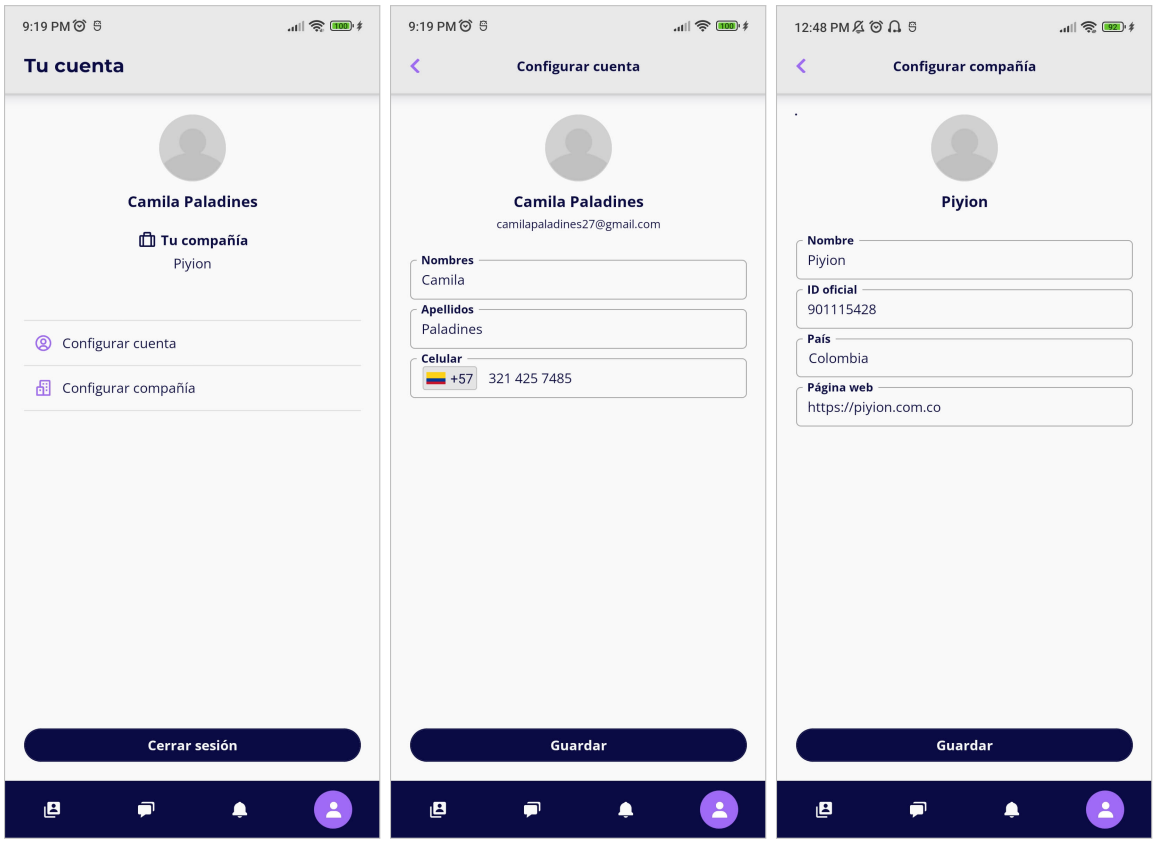


Figura 8.8: Pantallas implementadas sobre configuraciones

Anexo #2: Reutilización de código con componentes

Uno de los principios de React es el uso de componentes como piezas de código que producen un elemento visual. Estos pueden ser reutilizados las veces que sea necesario, e incluso cambiando ligeramente algunas de sus características. Por ejemplo, en la Figura 8.9 se muestra la definición del componente *Button*, que se puede personalizar asignándole un título (*title*) y un evento que se ejecuta al presionarlo (*onPress*). Además, se puede definir opcionalmente un tipo (*type* que puede ser "primary" o "secondary"), un color (*color* que puede ser "blue" o "red") y otros estilos (*style*).

```
interface ButtonProps {
    title: string;
    onPress: () => void;
    type?: 'primary' | 'secondary';
    color?: 'blue' | 'red';
    style?: any;
}

You, 2 months ago | 1 author (You)
export default function Button(props: ButtonProps) {
    // Estilos y multimedia
    const styles = require('./Button.scss');
    const OpenSansBold = {fontFamily: 'OpenSans-Bold'};
    const mainColor = props.color === 'blue' ? colors.azulOscuro : colors.estadoRojo;
    const backgroundColor = {backgroundColor: props.type === 'primary' ? mainColor : 'transparent'};
    const borderColor = {borderColor: mainColor};
    const color = {color: props.type === 'primary' ? colors.blanco1 : mainColor};

    return (
        <Pressable onPress={props.onPress} style={[styles.container, backgroundColor, borderColor, props.style]}>
            <Text style={[styles.text, color, OpenSansBold]}>{props.title}</Text>
        </Pressable>
    );
}

Button.defaultProps = {
    type: 'primary',
    color: 'blue',
    style: {},
};
```

Figura 8.9: Definición del componente "Button"

En la Figura 8.10 se muestra una de las formas de usar este componente, estableciendo las propiedades requeridas para su construcción.

```
<Button title='Guardar' onPress={save} style={styles.saveButton} />
<Button title='Cancelar' onPress={goBack} type='secondary' style={styles.cancelButton} />
```

Figura 8.10: Uso del componente "Button"

Anexo #3: Análisis de código estático con TypeScript

Una de las principales ventajas de usar TypeScript para este proyecto fue el análisis de código estático. Como se había mencionado anteriormente, este permite identificar una gran cantidad de posibles errores en el momento que se está escribiendo el código y no sólo a la hora de ejecutarlo. Un ejemplo sencillo de esto es la operación que se realiza cuando se asigna una solicitud de atención a un asesor. Además de relacionar la conversación con dicho asesor, es necesario actualizar su atributo *conversationsAmount*, incrementándolo en 1. Sin embargo, si se usa un tipo de dato diferente a *number*, que es el que posee, se muestra un error (ver Figura 8.11).

```
// Actualiza la cantidad de conversaciones del agente
let agent = await teamDB.getOne(data.agent_id);
agent.conversationsAmount += '1';
await teamDB.updateOne(agent);
```

Figura 8.11: Código con un error

Al posicionar el puntero del mouse sobre el error, aparece el mensaje que se muestra en la Figura 8.12. Este indica que no es posible asignar una cadena de texto a una variable de tipo *number*. Al "sumar" un número con una cadena de texto, da como resultado una cadena que concatena ambos valores, lo que podría generar errores en tiempo de ejecución; sin embargo, con TypeScript es posible detectar esto a tiempo.

```
(property) conversationsAmount: number
Type 'string' is not assignable to type 'number'. ts(2322)
View Problem (⌥F8)    No quick fixes available
agent.conversationsAmount += '1';
await teamDB.updateOne(agent);
```

Figura 8.12: Mensaje de error

En la Figura 8.13 se puede observar que al cambiar la cadena de texto por un número, el error desapareció.

```
// Actualiza la cantidad de conversaciones del agente
let agent = await teamDB.getOne(data.agent_id);
agent.conversationsAmount += 1;
await teamDB.updateOne(agent);
```

Figura 8.13: Código sin errores

Anexo #4: Manejo de errores en el frontend

El manejo de errores dentro de un programa de software ayuda no solo al usuario a informarse que algo no ocurrió como se esperaba, sino que también proporciona al desarrollador una mejor alternativa para encontrar fácilmente el origen del problema que se está presentando. En el flujo de la aplicación se pueden presentar generalmente dos tipos de errores que se explican a continuación.

Información inválida en campos de texto

En este caso sólo se muestra un mensaje de error al usuario si este no ha diligenciado un campo correctamente. Además, no se permite ejecutar una acción (de actualizar dicha información) hasta que los datos ingresados sean corregidos. Un ejemplo de esto se puede observar en la Figura 8.14, que muestra un error ya que el campo "ID oficial" no puede ser vacío.

Figura 8.14: Error de campo diligenciado incorrectamente

En la Figura 8.15 se muestra el código fuente que permite dicho manejo de errores. En este caso, se mostrarán los mensajes de error cada vez que se presione el botón de guardar y alguno de los campos no sea correcto. Sólo cuando se cumplan las condiciones se podrá guardar la información.

```
const save = () => {
    const newName = name.trim();
    const newOfficialID = officialID.trim();
    const newCountry = country.trim();
    const newWebPage = webPage.trim();

    // Verifica la correctitud de los datos
    if (newName === '') {
        openAlert('Valor inválido', 'Por favor, ingrese un nombre');
        return;
    } else if (newOfficialID === '') {
        openAlert('Valor inválido', 'Por favor, ingrese un número de identificación');
        return;
    } else if (newCountry === '') {
        openAlert('Valor inválido', 'Por favor, seleccione un país');
        return;
    } else if (newWebPage === '') {
        openAlert('Valor inválido', 'Por favor, ingrese una página web');
        return;
    } else {
        // Nueva informacion
        const newCompanyData = {
            ...company,
            name: newName,
            officialID: newOfficialID,
            country: newCountry,
            webPage: newWebPage,
        };

        // Actualiza los datos de la compañía
        setLoading(true);
        companiesRemote
            .updateCompany(newCompanyData)
            .then(async () => {
                await setCompany(newCompanyData);
                goBack();
                setLoading(false);
            })
            .catch((error) => {
                showError('Actualizar compañía', error, 'No se pudo actualizar la información de la compañía');
            });
    }
};
```

Figura 8.15: Manejo de errores en actualizar datos de la compañía

Errores relacionados a la API

Cuando ocurre un error en un llamado a la API se imprime en consola dicho error para realizar el debugging (en la etapa de desarrollo) y se muestra al usuario un mensaje personalizado según en dónde se generó el error.

En la Figura 8.16 se puede observar la función utilizada para mostrar posibles errores relacionados con las peticiones a la API. En este caso se muestra un error en consola para el desarrollador, además de abrir una ventana de alerta para mostrarle al usuario que ocurrió un error.

```
// Mensajes de error
const showError = (state: string, error: string, message: string, callback?: () => void) => {
    console.error(`${state} -> ${error}`);
    openAlert('Ocurrió un error', message);
    setLoading(false);

    if (callback) callback();
};
```

Figura 8.16: Función para mostrar errores al usuario

La función `showError` recibe cuatro parámetros:

- `state`: indica en qué estado o parte de la aplicación ocurre el error.
- `error`: es el error que lanza el sistema, recibido desde una petición al backend.
- `message`: es un mensaje informativo para el usuario indicándole que ocurrió un error en una operación.
- `callback`: es opcional y corresponde a una función que se ejecuta al mostrar el error (puede usarse para descartar cambios y regresar a un estado en específico).

En la Figura 8.17 se muestra el uso de la función de mostrar errores. En este caso se ejecuta cuando se hace un llamado a la API para actualizar la información de un contacto y la operación no se pudo realizar correctamente.

```
// Actualiza un contacto
setLoading(true);
contactsRemote
    .updateContact(newContact)
    .then(() => {
        const newConversation = {...currentConversation, contact: newContact};
        setCurrentConversation(newConversation);
        setLoading(false);
        goBack();
    })
    .catch((error) => {
        showError('Actualizar contacto', error, 'No se pudo actualizar el contacto');
    });
```

Figura 8.17: Usando la función para mostrar errores al usuario

Anexo #5: Ejemplos de pruebas funcionales de sistema

Para llevar a cabo las pruebas funcionales de sistema se utilizaron tres estrategias de prueba. A continuación se muestran ejemplos de cada una de ellas.

Tablas de decisión

En la Figura 8.18 se muestra el gráfico de causas y efectos de la funcionalidad sobre "Iniciar sesión con correo y contraseña". Los círculos de color azul corresponden a las causas y los de color violeta a los efectos. Las flechas acompañadas por una bandera indican que sucede lo contrario de lo que se menciona en la causa.

En este caso, se tienen dos causas asociadas a la correctitud de los dos campos que se necesitan para iniciar sesión. Si el correo se encuentra registrado y la contraseña es la correcta sucede un ingreso exitoso (líneas verdes). Sin embargo, si el correo es correcto pero la contraseña no, se produce un ingreso fallido (líneas rojas). Finalmente, si el correo no se encuentra registrado el ingreso es fallido (línea naranja), sin importar el valor de la contraseña.

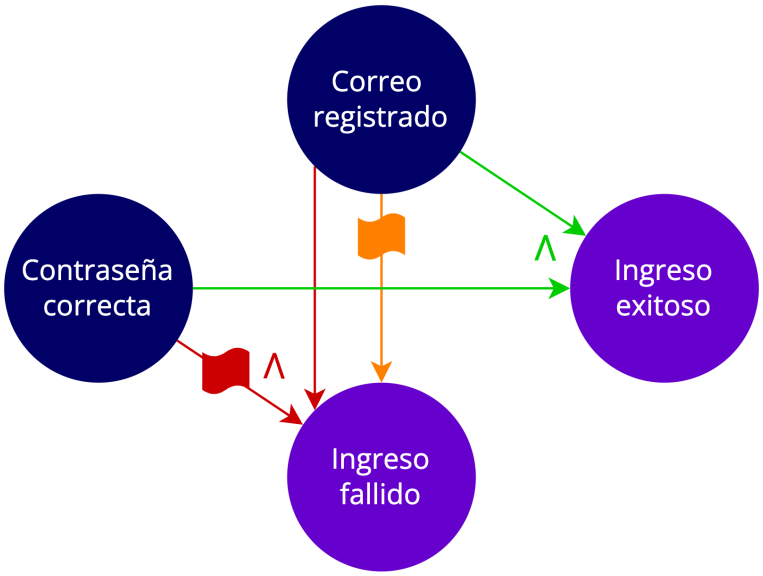


Figura 8.18: Gráfico de causas y efectos de la funcionalidad F01

Teniendo en cuenta lo anterior, se muestra en la Figura 8.19 la tabla de las causas y efectos, además de los tres casos de prueba definidos para cada escenario posible.

Causas y efectos		P1	P2	P3
Correo registrado	Causa 1	1	1	0
Contraseña correcta	Causa 2	1	0	-
Ingreso exitoso	Efecto 1	1		
Ingreso fallido	Efecto 2		1	1

Figura 8.19: Tabla de causas y efectos de la funcionalidad F01

Particiones de equivalencia

En la Figura 8.20 se muestra la tabla con las variables (columna 1), las clases de equivalencia o particiones (columna 2), si es válida o no válida (columna 3), el representante de dicha clase (columna 4) y los ocho casos de prueba al combinar las clases de equivalencia.

Variables	Clases de equivalencia	Tipo	Representante	P1	P2	P3	P4	P5	P6	P7	P8
Nombres	firstName != ""	V	"John"	x	x	x	x	x	x	x	
	firstName == ""	NV	""								x
Apellidos	lastName != ""	V	"Doe"	x	x	x	x	x	x		x
	lastName == ""	NV	""							x	
Correo electrónico	format(email) == "email"	V	"john.doe@domain.com"	x	x	x	x	x		x	x
	format(email) != "email"	NV	"iamjohn"						x		
Celular	phone != ""	V	"1234567890"	x	x	x	x		x	x	x
	phone == ""	NV	""					x			
Compañía	company != ""	V	"Ninus"	x	x			x	x	x	x
	company == ""	V	""			x	x				
Comentario	comment != ""	V	"Este es un comentario"	x		x		x	x	x	x
	comment == ""	V	""		x		x				

Figura 8.20: Particiones de equivalencia de la funcionalidad F12

En este caso, se tienen seis variables: “Nombres”, “Apellidos”, “Correo electrónico”, “Celular”, “Compañía” y “Comentario”, los cuales poseen cada uno sus clases de equivalencia o grupos de posibles valores. De cada clase de equivalencia se toma un representante para realizar las pruebas. Por otro lado, los casos de prueba se construyen a partir de la combinación de los valores válidos (Tipo “V”) y de conseguir una prueba por cada valor no válido (Tipo “NV”). Cabe resaltar que debido a que las variables “Compañía” y “Comentario” son campos opcionales para el usuario, sus dos clases de equivalencia son válidas en el sistema.

Al ejecutar las pruebas de esta funcionalidad se obtuvo sólo un caso fallido (P6), el cual consistía en verificar si el sistema rechazaba el valor del campo “Correo electrónico” que no coincidía con la estructura propia de un correo electrónico.

Transición de estado

En la Figura 8.21 se muestra el gráfico de estados de la funcionalidad sobre "Cambiar la disponibilidad". En esta prueba se tuvo en cuenta que cuando el usuario cambia su disponibilidad tiene la posibilidad de ver o no la lista de chats. En este sentido, cuando está disponible (rectángulo verde) puede ver los chats, pero cuando no está disponible (rectángulo amarillo) no puede hacerlo. Las flechas indican la acción de presionar el switch (ver Figura 4.3) que cambia el valor de la variable `available` entre sus dos valores booleanos.

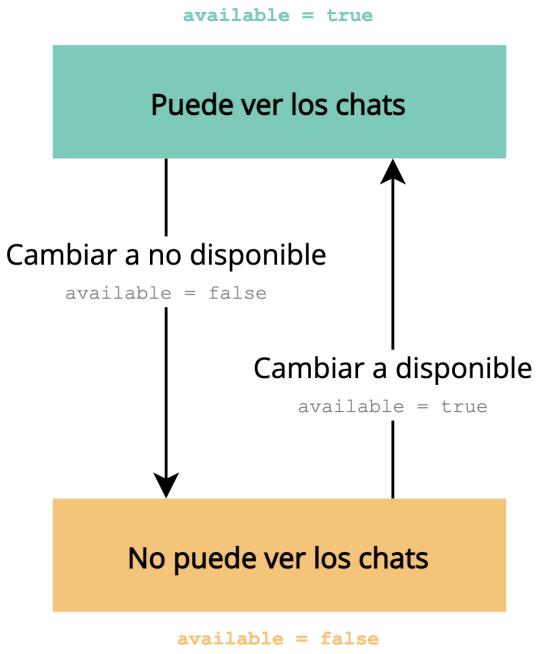


Figura 8.21: Gráfico de estados de la funcionalidad F08

En la Figura 8.22 se muestra la tabla con los dos casos de prueba realizados. El primero consiste en partir desde un estado donde se pueden ver los chats, realizar la acción de cambiar a "no disponible" para obtener como estado final que no puede ver los chats. El segundo consiste en partir de un estado donde no se ven los chats y al cambiar el switch a "disponible" es posible visualizarlos.

Caso	Estado inicial	Acción	Estado final
P1	Puede ver los chats	Cambiar a no disponible	No puede ver los chats
P2	No puede ver los chats	Cambiar a disponible	Puede ver los chats

Figura 8.22: Casos de la funcionalidad F08

Anexo #6: Cobertura de rama

En el backend también se hizo la medición del porcentaje de caminos que fueron cubiertos con las pruebas unitarias o, como comúnmente se le conoce, cobertura de rama. En la Figura 8.23 se muestran los resultados de esta medición.

Name	Stmts	Miss	Branch	BrPart	Cover
src/main/python/communication_proxy/application/communication_proxy.py	43	16	6	0	59%
src/main/python/communication_proxy/application/proxy_receiver.py	86	50	26	0	38%
src/main/python/communication_proxy/application/proxy_sender.py	111	92	36	0	13%
src/main/python/communication_proxy/domain/proxy_controller_repository.py	8	0	0	0	100%
src/main/python/infraestructure/dyalogo/dyalogo_adapter.py	44	33	6	0	22%
src/main/python/infraestructure/firestore/cp_controller.py	115	73	32	4	33%
src/main/python/infraestructure/firestore/firestore_connection.py	11	1	0	0	91%
src/main/python/infraestructure/routes/agent_routes.py	12	0	4	0	100%
src/main/python/infraestructure/routes/company_routes.py	240	59	100	5	74%
src/main/python/infraestructure/routes/contact_routes.py	64	6	32	4	88%
src/main/python/infraestructure/routes/conversation_routes.py	78	4	30	3	94%
src/main/python/infraestructure/routes/dyalogo_routes.py	31	0	14	2	96%
src/main/python/infraestructure/routes/fcm_routes.py	77	4	28	5	91%
src/main/python/infraestructure/routes/gupshup_webhook.py	33	1	10	1	95%
src/main/python/infraestructure/routes/instagram_webhook.py	33	2	12	2	91%
src/main/python/infraestructure/routes/message_routes.py	34	0	12	0	100%
src/main/python/infraestructure/routes/messenger_webhook.py	31	1	10	1	95%
src/main/python/infraestructure/routes/rates_routes.py	25	0	8	0	100%
src/main/python/infraestructure/routes/telegram_webhook.py	33	4	8	1	88%
src/main/python/infraestructure/routes/user_routes.py	123	0	42	0	100%
src/main/python/models/company.py	34	0	0	0	100%
src/main/python/models/contact.py	19	0	0	0	100%
src/main/python/models/conversation.py	58	0	0	0	100%
src/main/python/models/device.py	13	0	0	0	100%
src/main/python/models/message.py	31	3	0	0	90%
src/main/python/models/rate.py	11	3	0	0	73%
src/main/python/models/user.py	30	0	0	0	100%
src/main/python/piyion/application/piyion.py	274	46	16	0	79%
src/main/python/serializers/company_serializers.py	47	0	4	0	100%
src/main/python/serializers/contact_serializer.py	18	0	0	0	100%
src/main/python/serializers/conversation_serializers.py	11	0	0	0	100%
src/main/python/serializers/dyalogo_serializers.py	9	0	0	0	100%
src/main/python/serializers/fcm_serializers.py	83	0	0	0	100%
src/main/python/serializers/global_serializers.py	6	0	0	0	100%
src/main/python/serializers/gupshup_serializers.py	24	0	0	0	100%
src/main/python/serializers/mails_serializers.py	9	0	0	0	100%
src/main/python/serializers/messenger_serializers.py	37	0	6	0	100%
src/main/python/serializers/telegram_serializers.py	42	0	8	0	100%
src/main/python/serializers/user_serializers.py	29	0	0	0	100%
src/main/python/targets/dialogflow/dialogflow.py	29	18	0	0	38%
src/main/python/targets/domain/assistant_repository.py	3	0	0	0	100%
src/main/python/targets/domain/target_repository.py	9	0	0	0	100%
src/main/python/targets/watson/watson.py	23	0	0	0	100%
src/main/python/utils/class_utils.py	7	2	2	1	67%
src/main/python/utils/collections_utils.py	7	7	4	0	0%
src/main/python/utils/decorators.py	10	0	2	0	100%
src/main/python/utils/email_templates/company_request.py	7	0	0	0	100%
src/main/python/utils/exceptions.py	16	0	0	0	100%
src/main/python/utils/firebase_utils.py	13	2	0	0	85%
src/main/python/utils/get_file_path_telegram.py	16	2	0	0	88%
src/main/python/utils/notifications.py	100	8	36	6	88%
src/main/python/utils/publisher.py	12	6	0	0	50%
src/main/python/utils/user_validations.py	5	0	0	0	100%
TOTAL	2274	443	494	35	78%

Figura 8.23: Resultados de la cobertura de rama

Esta medida permite saber qué tan exhaustivas fueron las pruebas. El resultado obtenido indica que, con las pruebas realizadas, fueron cubiertos el 78 % de los caminos del código, un valor muy cercano al que se logró en la cobertura de sentencia. Esta medida también fue obtenida usando la herramienta Coverage [90], indicando que se requería la cobertura de rama, añadiendo el atributo `--branch`.

Anexo #7: Pruebas de rendimiento

En el backend se llevaron a cabo pruebas de rendimiento para ver el comportamiento de la arquitectura final. Esto, con el objetivo de ver que este enfoque no se "colgara" ante una alta demanda o un flujo considerable de datos, como pasaba con la arquitectura de notificaciones que se tenía anteriormente.

Para esto, se escribió un pequeño script en Python que encolara en el servicio de notificaciones tantos mensajes como se quisiera. Este script fue desplegado como cualquier otra API, para poder ser llamado múltiples veces y de manera concurrente. El script en cuestión se muestra en la Figura 8.24.

```python
@use_headers(allowed_methods=["POST"])
def send_notifications_stress_route(request, headers):
    request_method = request.method

    # Set CORS headers for the preflight request
    if request_method == "OPTIONS":
        return ("", 200, headers)
      You, a month ago • stress notifications endpoint definition
    if request_method == "POST":
        payload = request.get_json()
        device_type = payload["device_type"]
        device_id = payload["device_id"]
        num_notifications = payload["num_notifications"]
        data = {
            "conversation_id": payload["conversation_id"],
            "agent_id": payload["agent_id"],
            "type": "text",
            "notification_type": payload["notification_type"],
            "state": "assigned",
            "target": "piyion",
        }
        notification_service = Notifications()

        for x in range(num_notifications):
            data["notification_id"] = str(uuid4())
            data["not_number"] = x
            data["content"] = payload["content"] + " " + str(x)
            notification_service.send_single_notification(
                device_id=device_id, device_type=device_type, data=data
            )
    else:
        return abort(400)

    return ("Success!", 200, headers)
```

Figura 8.24: Definición del endpoint creado para pruebas de estrés en el backend

Este endpoint puede ser llamado asignando en el campo *num_notifications* del JSON que se le envía, el número de notificaciones que quieren enviarse una tras otra. Para las pruebas de estrés que se hicieron, se llamó 100 veces el endpoint, en lotes de 20 llamados (de manera simultánea) y con 10 segundos de diferencia entre cada lote, cada llamado con 10.000 notificaciones. Esto para un total de 1.000.000 de notificaciones enviadas desde el backend al dispositivo seleccionado.

En la Figura 8.25 se puede evidenciar la manera en la que se iban enviando de manera concurrente las diferentes notificaciones. Es importante resaltar que, al hacerlo por lotes y llamados, puede que las impresiones se hagan unas primero que otras.

GRAVEDAD	MARCA DE TIEMPO	RESUMEN
>	2023-02-14 17:27:12.044 EST	notifications_subscriber vru3z92pctco [LOG] LOTE 2 -> INVOCACION # (13), NOTIFICACION # 504
>	2023-02-14 17:27:12.056 EST	notifications_subscriber epxhl3img93y [LOG] getting secret
>	2023-02-14 17:27:12.117 EST	notifications_subscriber 7fv6f8j1z6w8 [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…
>	2023-02-14 17:27:12.117 EST	notifications_subscriber 7fv6f8j1z6w8 [LOG] LOTE 4 -> INVOCACION # (13), NOTIFICACION # 2472
>	2023-02-14 17:27:12.119 EST	notifications_subscriber m4hs66a83c5f [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…
>	2023-02-14 17:27:12.119 EST	notifications_subscriber m4hs66a83c5f [LOG] LOTE 2 -> INVOCACION # (15), NOTIFICACION # 3199
>	2023-02-14 17:27:12.119 EST	notifications_subscriber m4hs66a83c5f [LOG] getting secret
>	2023-02-14 17:27:12.126 EST	notifications_subscriber j6y1xkinnhp0 [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…
>	2023-02-14 17:27:12.126 EST	notifications_subscriber j6y1xkinnhp0 [LOG] LOTE 1 -> INVOCACION # (20), NOTIFICACION # 9383
>	2023-02-14 17:27:12.126 EST	notifications_subscriber obl70o9tra1a [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…
>	2023-02-14 17:27:12.126 EST	notifications_subscriber obl70o9tra1a [LOG] LOTE 4 -> INVOCACION # (2), NOTIFICACION # 4337
>	2023-02-14 17:27:12.126 EST	notifications_subscriber obl70o9tra1a [LOG] getting secret
>	2023-02-14 17:27:12.133 EST	notifications_subscriber h10gtbvy73h0 [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…
>	2023-02-14 17:27:12.134 EST	notifications_subscriber h10gtbvy73h0 [LOG] LOTE 3 -> INVOCACION # (9), NOTIFICACION # 5115
>	2023-02-14 17:27:12.157 EST	notifications_subscriber vru3z92pctco [LOG] getting secret
>	2023-02-14 17:27:12.160 EST	notifications_subscriber epxhl3img93y [LOG] Device id -> eEZ4q9gaH4XjAdvCtWMGKl:APA91bGZYC72io5RwYsOulKxf2Jvhnahwi2BHMwKsQ6a0J_R5eHhcMkiS4GV5s…

Figura 8.25: Registros del Notifications Subscriber durante las pruebas de estrés

Anexo #8: Pruebas de escalabilidad

Con la finalidad de evaluar el comportamiento de la aplicación dentro de un entorno real, se le pidió a los agentes del cliente más grande de Piyion que usaran la aplicación para atender los chats de su empresa durante un día en específico. Una empresa que, en sus días más concurridos, ha tenido poco más de 4.000 conversaciones. En la Figura 8.26 se muestra el tráfico de conversaciones de la empresa del día 14 de febrero del 2023. Este día, en donde los agentes probaron la aplicación, se atendieron un total de 2.166 conversaciones, teniendo hasta 250 conversaciones activas en una misma hora (entre 9 y 10 de la mañana).

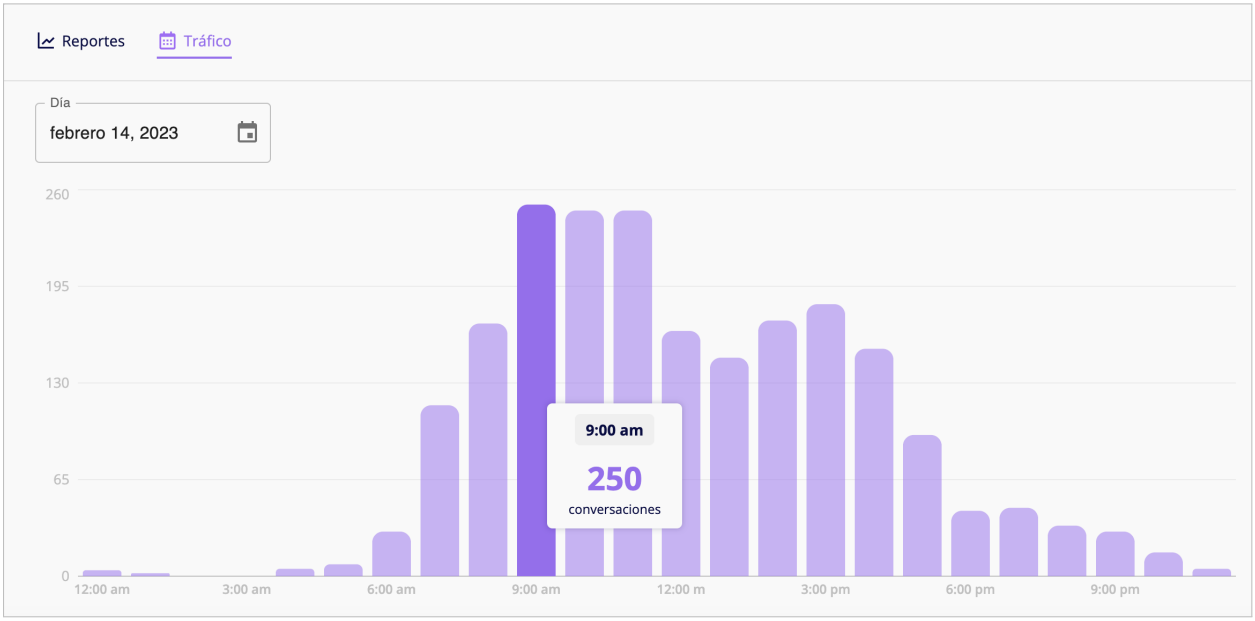


Figura 8.26: Tráfico de la empresa con la que se evaluó la escalabilidad de la app

Es importante destacar que los agentes han expresado una gran satisfacción con el rendimiento de la aplicación móvil de Piyion, tras haberla probado. Los comentarios recibidos indican que la aplicación es intuitiva y fácil de usar, además de que su rendimiento es rápido y eficiente. Estos agentes han destacado la calidad de las funciones y características que ofrece la aplicación, las cuales les han permitido realizar tareas de manera más efectiva y eficiente, sin necesidad de estar en sus computadores.

Además, es relevante destacar que Google se encarga de la escalabilidad del backend, su infraestructura es tan robusta que es capaz de responder ante picos de uso, un ejemplo de esto puede verse en el Anexo #7.