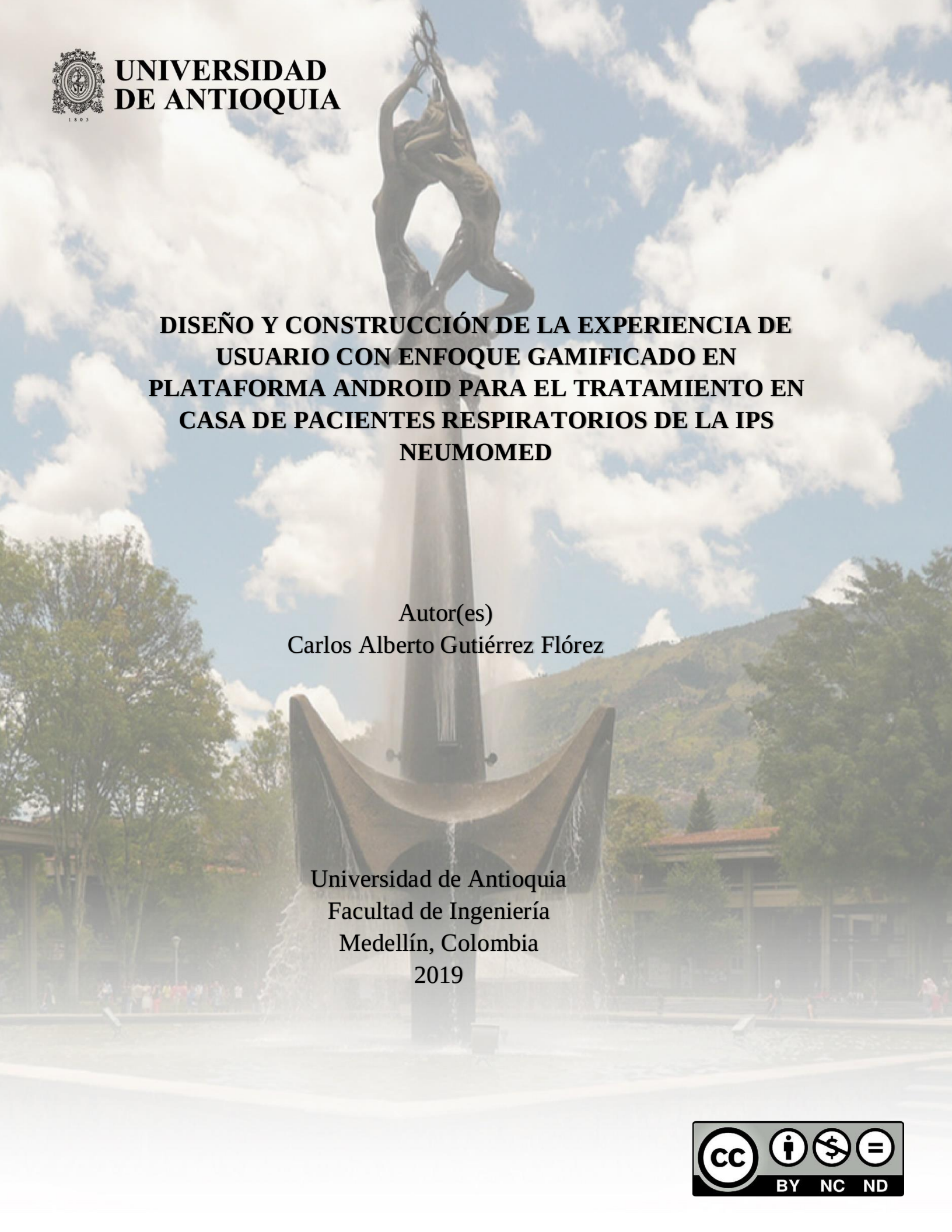

UNIVERSIDAD DE ANTIOQUIA

# DISEÑO Y CONSTRUCCIÓN DE LA EXPERIENCIA DE USUARIO CON ENFOQUE GAMIFICADO EN PLATAFORMA ANDROID PARA EL TRATAMIENTO EN CASA DE PACIENTES RESPIRATORIOS DE LA IPS NEUMOMED

Autor(es)
Carlos Alberto Gutiérrez Flórez

Universidad de Antioquia
Facultad de Ingeniería
Medellín, Colombia
2019

# Diseño y construcción de la experiencia de usuario con enfoque gamificado en plataforma Android para el tratamiento en casa de pacientes respiratorios de la IPS Neumomed

**Carlos Alberto Gutiérrez Flórez**

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
**Bioingeniero**

Asesores (a):
Maria Camila Niebles Reyes, Bioingeniera.
John Fredy Ochoa Gómez, PhD en Ingeniería Electrónica.

Línea de Investigación:
Desarrollo de Software

Universidad de Antioquia
Facultad de Ingeniería
Medellín, Colombia
2019.

# DISEÑO Y CONSTRUCCIÓN DE LA EXPERIENCIA DE USUARIO CON ENFOQUE GAMIFICADO EN PLATAFORMA ANDROID PARA EL TRATAMIENTO EN CASA DE PACIENTES RESPIRATORIOS DE LA IPS NEUMOMED

## 1. Resumen

**En la actualidad se desarrollan nuevas metodologías para la transformación de los servicios prestados en salud teniendo en cuenta el desarrollo digital y la revolución informática. La introducción de dispositivos móviles en el mercado mundial tiene un enorme potencial para cambiar la forma en que se brinda la atención médica, especialmente en la rehabilitación. La convergencia entre la salud y tecnología otorga acceso a recursos de atención médica a una amplia gama de personas, sin saturar el sistema, ya que reduce las consultas, las estadías en el hospital y los costos de atención médica. En este proyecto se plantea una solución para el acompañamiento del paciente de rehabilitación pulmonar a distancia de la IPS Neumomed, teniendo en cuenta conceptos como gamificación, patrones de diseño para el escalamiento, desarrollo transparente de la aplicación y usabilidad.**

## 2. Introducción

Los avances tecnológicos han sido un elemento importante para la evolución de la humanidad durante el siglo pasado. Básicamente se ha debido a la "revolución informática" que ha permitido cambiar todas las áreas de nuestra sociedad y revolucionar la medicina.

La irrupción de la tecnología en la sanidad es una realidad cada vez más evidente, impulsada por el auge de la transformación digital. De esta convergencia entre salud y tecnología digital nace la llamada *salud digital* ( *eHealth* ), que no solo nos brinda una caja de herramientas que respaldan el desarrollo de innovadores modelos de atención centrados en el paciente, impulsando la accesibilidad, la calidad, la seguridad y la eficiencia en todas las áreas de la salud, sino que es una transformación cultural de la atención médica tradicional, a través de tecnologías disruptivas como telemedicina, salud móvil ( *mHealth* ), aplicaciones, inteligencia artificial, sensores y otros dispositivos [1].

La introducción de dispositivos móviles en el mercado mundial tiene un enorme potencial para afectar la forma en que se brinda la atención de rehabilitación. El uso de teléfonos móviles ha crecido en los países en desarrollo. En 2017, las tasas de penetración para las suscripciones de teléfonos móviles alcanzaron el 98.7% de la población en los países en

desarrollo y el 70.4% en los países menos desarrollados. Como ocurre en otros campos, la medicina también ha experimentado los cambios traídos por esta revolución informática a través de la visión de la *e-Health*, basada en tecnologías de información y comunicación [2].

Las grandes ventajas que traen los dispositivos móviles en la salud marcan una visión representada en Mobile Health (*m-Health)* con aplicaciones potenciales en temas de salud pública ya que otorga acceso a recursos de atención médica a una amplia gama de personas, y sin saturar el sistema, ya que reduce las consultas, las estadías en el hospital y los costos de atención médica. En los países industrializados se considera una alternativa que puede resolver el problema de los costos y el acceso a la atención médica de una población cada vez más envejecida, mientras que en los países en desarrollo y gracias a la llegada de tecnologías móviles a estos países, otorgará acceso a la atención médica a la mayoría de las poblaciones de bajos ingresos que viven en zonas rurales[3].

Siendo suficiente evidencia de que *m-health* reduce costes sanitarios, mejora la salud de la población y la experiencia del cliente en el cuidado de su salud, la IPS Neumomed que ofrece una atención integral a pacientes con especialistas en neumología y neurología, siendo este grupo medico los más indicados para abordar al paciente respiratorio y dar solución a los trastornos del sueño, incursiona en *m-health* con una plataforma móvil ("Doctor Neumo") que permite una interacción con el paciente respiratorio y el grupo médico. La plataforma móvil se encuentra al servicio del paciente de rehabilitación ofreciendo registro de ejercicios y formularios, información importante para el monitoreo y prevención de casos adversos para pacientes que se encuentren en rehabilitación.

Al tener un mayor acompañamiento de la IPS hacia los pacientes, se busca que los sistemas de información sean un apoyo para los profesionales de la salud en su trabajo diario con los mismos. Lo cual implica proporcionar a los usuarios las funcionalidades adecuadas que siendo fáciles en su uso brinden calidad y eficiencia para la realización de las tareas clínicas. Por consiguiente, una de las principales preocupaciones en materia de adopción de sistemas de información para la atención en salud es la usabilidad. Actualmente, la industria no suele aplicar estrategias que aseguren usabilidad, sino que se continúa desarrollando plataformas sin tener consideración al usuario. Por lo tanto, en el presente trabajo, se conecta el concepto de usabilidad con la plataforma móvil, buscando metodologías para lograr un enganche de los pacientes con la rehabilitación pulmonar usando m-Health.

## 3. Objetivos

### 3.1 Objetivo general

Mejorar la usabilidad de la plataforma móvil "Doctor Neumo" introduciendo conceptos de gamificación para apoyar el proceso de rehabilitación en casa.

### 3.2 Objetivos específicos

- Recopilar la información considerada relevante para introducir la gamificación en "Doctor Neumo" buscando mejorar la experiencia para el paciente respiratorio.

- Adaptar la aplicación para que introduzca la estrategia de gamificación usando principios de buenas prácticas basadas en SOLID.

- Evaluar la mejora en la usabilidad de los ejercicios de rehabilitación en "Doctor Neumo" mediante un estudio piloto en un grupo de usuarios.

## 4. Marco Teórico

### 4.1 eHealth

La salud electronica o *e-Health,* según la definición de la Organización Mundial de la Salud (OMS), es "el uso de las Tecnologías de la Información y la Comunicación (TIC)" para la salud. Es evidente que la implementación de sistemas de eHealth, como los registros electrónicos de salud, la salud móvil (*m-Health*), los sistemas de telemedicina y los hospitales inteligentes, se ha convertido en una piedra angular para mejorar la accesibilidad, la capacidad de respuesta y la asequibilidad de los servicios de salud. Además, una implementación adecuada de los sistemas de eHealth puede mejorar dramáticamente estos servicios a nivel local, regional y mundial. Sin embargo, y similar a otras iniciativas nacionales como eLearing, el desarrollo exitoso de un sistema de eHealth eficiente se convierte en un verdadero desafío ya que incorpora diferentes campos, requiere experiencias interdisciplinarias y requiere un modelo de seguridad estricto para proteger la información de los pacientes [4].

### 4.2 m-Health

En la literatura, el sistema de asistencia sanitaria móvil ha sido conceptualizado como salud móvil (mHealth). Desafortunadamente, hasta ahora no se ha establecido una definición ampliamente aceptada de mHealth. El concepto de mHealth surgió del concepto anterior de salud electrónica (eHealth) que tenía la intención de proporcionar instalaciones de atención médica utilizando tecnologías de la información fija y periféricos informáticos.

El Observatorio Global de Salud definió mHealth como "práctica médica y de salud pública respaldada por dispositivos móviles, como teléfonos móviles, dispositivos de monitoreo de pacientes, asistentes digitales personales (PDA) y otros dispositivos inalámbricos". Recientemente, las tecnologías móviles mejoradas (por ejemplo: 3G, 4G y 5G) y la miniaturización de dispositivos informáticos (por ejemplo, Android, teléfonos móviles basados iOS) han transformado con éxito el sistema de eHealth orientado a las instalaciones en un sistema portátil de mHealth [3].

En comparación con eHealth, mHealth es más aplicable, debido a la portabilidad y la mayor flexibilidad del sistema. El sistema puede usarse en áreas rurales donde las instalaciones médicas convencionales son difíciles de alcanzar. mHealth puede considerarse un sustituto de bajo costo para la prestación de servicios de salud a los ciudadanos de un país. La eficacia de

los sistemas basados en mHealth en comparación con el sistema tradicional de prestación de servicios de salud es de hecho un tema discutible. Sin embargo, se espera que la tecnología mHealth traiga una revolución en el sector de la salud en el futuro debido a la flexibilidad que implica. mHealth puede convertirse en una herramienta complementaria importante para gestionar problemas relacionados con la salud en un futuro próximo [3].

En comparación con los proveedores de atención médica tradicionales, mHealth tiene el potencial de reducir el costo de los servicios de atención médica. Además, mHealth puede entregar toda la información relacionada con la salud al individuo que a su vez conducirá a un estilo de vida saludable. Se ha descubierto que mHealth es una herramienta útil para controlar enfermedades crónicas, reducir problemas de salud y aumentar el conocimiento de la salud de un individuo [3].

Un sistema basado en mHealth puede desempeñar un papel crucial en la prestación de servicios de salud a los seres humanos con el creciente número de usuarios de teléfonos inteligentes. Se espera que el sistema de prestación de servicios médicos basado en aplicaciones se convierta en un cambio de juego, especialmente en las zonas rurales donde las personas carecen de instalaciones médicas básicas. Según un estudio reciente, el número de usuarios de teléfonos inteligentes puede llegar a aproximadamente 3.800 millones en todo el mundo para 2021 por lo que mHealth será una aplicación innovadora para el siglo XXI [3].

Los estudios de mHealth se han llevado a cabo en las siguientes áreas: (a) adopción de tecnología de aplicaciones móviles de salud; (b) desarrollo de nuevas aplicaciones de software relacionadas con la salud y la forma física, y (c) ensayos clínicos sobre la adopción del sistema mHealth. Desafortunadamente, se han realizado estudios limitados sobre el desarrollo de marcos para seleccionar aplicaciones de mHealth. Debido a la mayor penetración de los teléfonos móviles basados en Android y Windows en el mercado en todo el mundo, la selección de la aplicación mHealth se ha convertido en una agenda importante. Hoy en día, las personas utilizan mucho las aplicaciones basadas en teléfonos inteligentes en su día a día. Sin embargo, en la literatura falta un marco adecuado para evaluar la eficacia de las aplicaciones móviles en el mercado[3].

### 4.4 Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de apps para Android, basado en intelliJ IDEA. Además del

potente editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece incluso más funciones que aumentan su productividad cuando desarrollas apps para Android, como las siguientes[5]:

- Un sistema de compilación flexible basado en Gradle
- Un emulador rápido y cargado de funciones
- Un entorno unificado donde puedes desarrollar para todos los dispositivos Android
- Aplicación de cambios para insertar cambios de códigos y recursos a la aplicación en ejecución sin reiniciar la aplicación
- Integración con GitHub y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de ejemplo
- Variedad de marcos de trabajo y herramientas de prueba
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de la versión, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada para Google Cloud Plataform, que facilita la integración con Google Cloud Messaging y App Engine

### 4.4.1 Estructura del proyecto en Android Studio

Cada proyecto de Android Studio incluye uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes [5]:

- Módulos de apps para Android
- Módulos de biblioteca
- Módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos del proyecto en la vista de proyecto de Android, como se ve en la Figura 1. Esta vista está organizada en módulos para que puedas acceder rápidamente a los archivos fuente clave del proyecto [5].

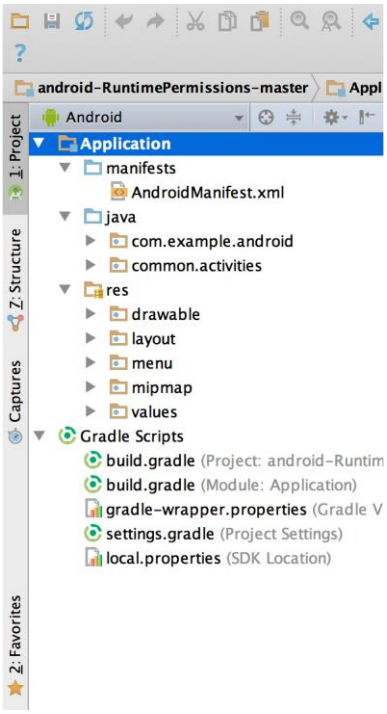


*Figura 1. Vista del proyecto en Android Studio*

Se pueden ver todos los archivos de compilación en el nivel superior de Secuencias de comando de Gradle y cada módulo de app contiene las siguientes carpetas[5]:

- manifests: Contiene el archivo AndroidManifest.xml.
- java: Contiene los archivos de código fuente Java, incluido el código de prueba de JUnit.
- res: Contiene todos los recursos sin código, como diseños XML, strings de IU e imágenes de mapa de bits.

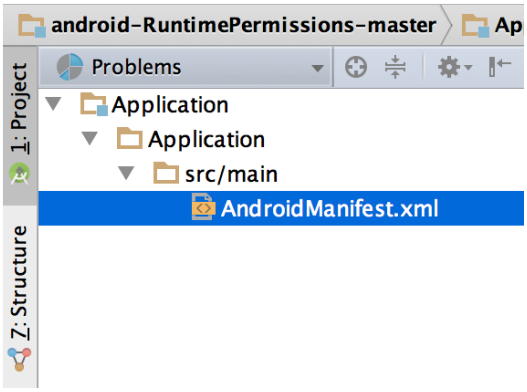


*Figura 2. Archivo manifest Android Studio*

### 4.4.2 Interfaz de usuario

La ventana principal de Android Studio consta de varias áreas lógicas identificadas en la Figura 3 [5].

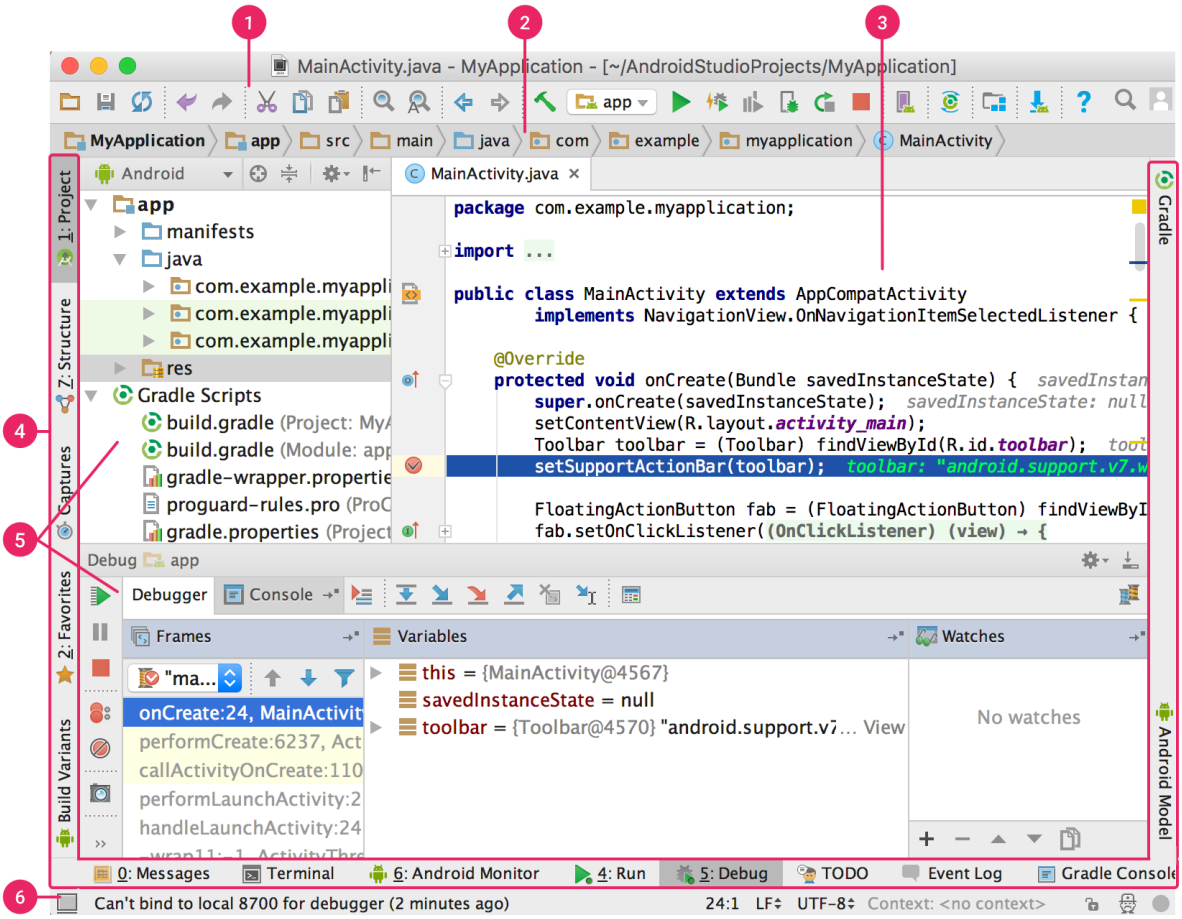


*Figura 3. Vista interfaz Android Studio*

1. La **barra de herramientas** permite realizar una gran variedad de acciones, como ejecutar la app e iniciar las herramientas de Android [5].

2. La **barra de navegación** ayuda a explorar el proyecto y abrir archivos para editar. Proporciona una vista más compacta de la estructura visible en la ventana **Project** [5].

3. La **ventana del editor** es el área en la que puedes crear y modificar código. Según el tipo de actividad actual, el editor puede cambiar. Por ejemplo, cuando ves un archivo de diseño, el editor muestra el Editor de diseño [5].

4. La **barra de la ventana de herramientas** se encuentra afuera de la ventana del IDE y contiene los botones que permiten expandir o contraer ventanas de herramientas individuales [5].

5. Las **ventanas de herramientas** brindan acceso a tareas específicas, como la administración de proyectos, la búsqueda, el control de versiones, entre otras [5].

6. En la **barra de estado**, se muestra el estado del proyecto y el IDE, además de advertencias o mensajes [5].

Se puede organizar la ventana principal para tener más espacio en pantalla si oculta o se desplaza las barras y ventanas de herramientas. También se puede usar combinaciones de teclas para acceder a la mayoría de las funciones del IDE [5].

En cualquier momento, se puede realizar búsquedas en el código fuente, las bases de datos, las acciones y los elementos de la interfaz de usuario, entre otros[5].

### 4.5 Patrón Arquitectónico

Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares al patrón de diseño de software pero tienen un alcance más amplio[6].

En aplicaciones móviles existen 10 patrones arquitectónicos comunes.

#### 4.5.1 Patrón de capas

Este patrón se puede utilizar para estructurar programas que se pueden descomponer en grupos de subtareas, cada una de las cuales se encuentra en un nivel particular de abstracción. Cada capa proporciona servicios a la siguiente capa superior[6].
Las 4 capas más comúnmente encontradas de un sistema de información general son las siguientes.

- Capa de presentación (también conocida como capa UI)
- Capa de aplicación (también conocida como capa de servicio)
- Capa de lógica de negocios (también conocida como capa de dominio)

- Capa de acceso a datos (también conocida como capa de persistencia)

**Uso:**

- Aplicaciones de escritorio generales.
- Aplicaciones web de comercio electrónico.

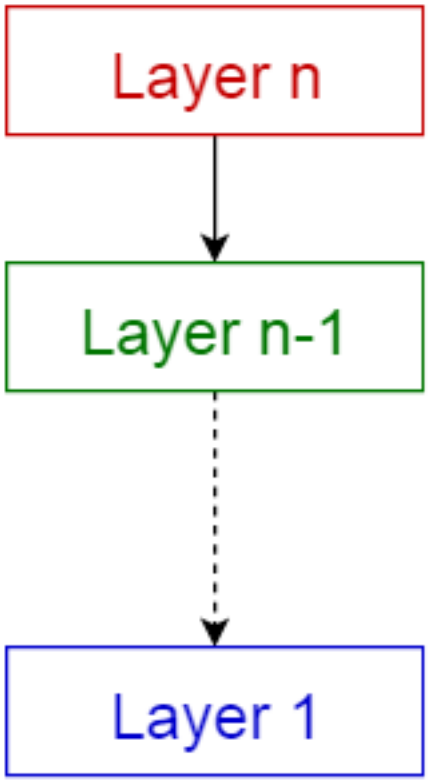


*Figura 4. Patrón de capas*

### 4.5.2 Patrón cliente-servidor

Este patrón consiste en dos partes; un **servidor** y múltiples **clientes**. El componente del servidor proporcionará servicios a múltiples componentes del cliente. Los clientes solicitan servicios del servidor y el servidor proporciona servicios relevantes a esos clientes. Además, el servidor sigue escuchando las solicitudes de los clientes[6].

**Uso:**

- Aplicaciones en línea como correo electrónico, uso compartido de documentos y banca.

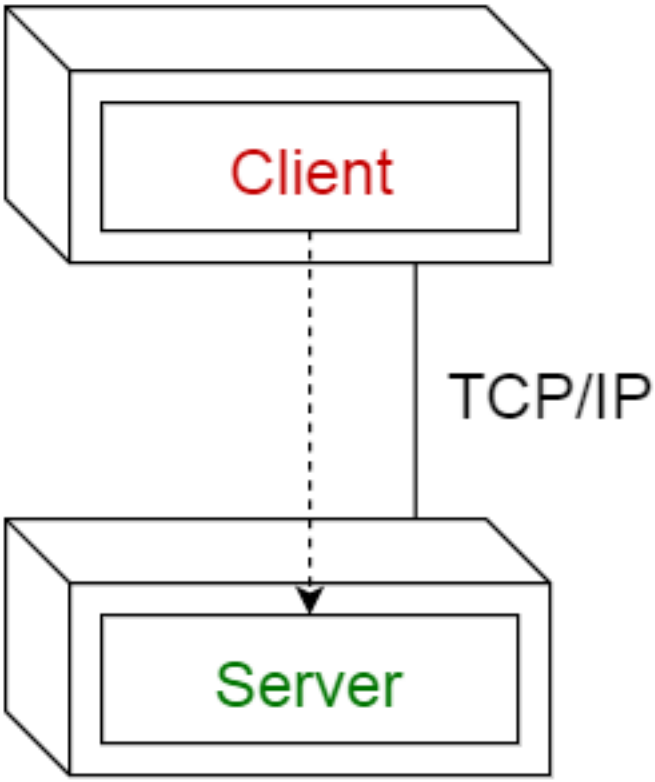


*Figura 5. Patrón cliente-servidor*

### 4.5.3 Patrón maestro-esclavo

Este patrón consiste en dos partes: maestro y esclavos. El componente maestro distribuye el trabajo entre componentes esclavos idénticos y calcula el resultado final de los resultados que devuelven los esclavos[6].

**Uso:**

- En la replicación de la base de datos, la base de datos maestra se considera como la fuente autorizada y las bases de datos esclavas se sincronizan con ella.
- Periféricos conectados a un bus en un sistema informático (unidades maestra y esclava).

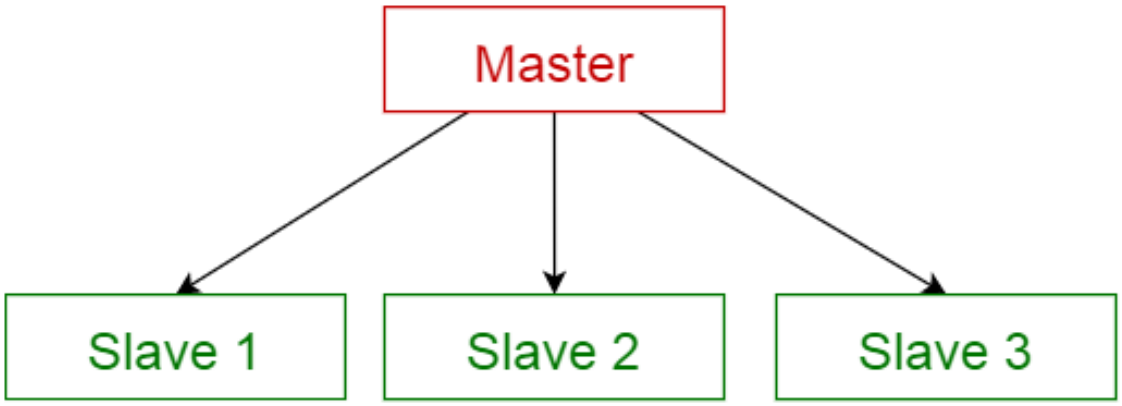


*Figura 6. Patrón maestro-esclavo*

### 4.5.4 Patrón de filtro de tubería

Este patrón se puede usar para estructurar sistemas que producen y procesan una secuencia de datos. Cada paso de procesamiento se incluye dentro de un componente de filtro. Los datos que se procesarán se pasan a través de las tuberías. Estas tuberías se pueden utilizar para el almacenamiento en búfer o con fines de sincronización[6].

**Uso:**

- Compiladores donde los filtros consecutivos realizan análisis léxico, análisis sintáctico y generación de código.
- Flujos de trabajo en bioinformática.

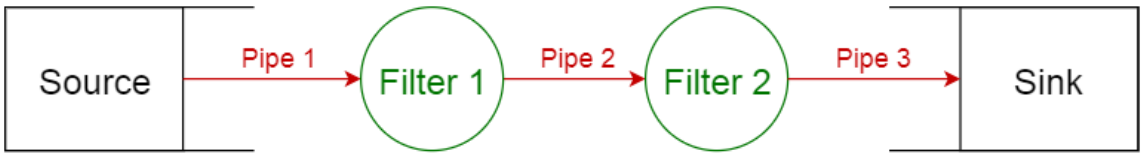


*Figura 7. Patrón de filtro de tubería*

### 4.5.5 Patrón de intermediario

Este patrón se usa para estructurar sistemas distribuidos con componentes desacoplados. Estos componentes pueden interactuar entre sí mediante invocaciones de servicios remotos. Un componente de intermediario es responsable de la coordinación de la comunicación entre los componentes [6].

Los servidores publican sus capacidades (servicios y características) a un intermediario. Los clientes solicitan un servicio del intermediario y el intermediario redirecciona al cliente a un servicio adecuado desde su registro [6].

**Uso:**

- Software de Message Broker Como Apache ActiveMQ , Apache Kafka , RabbitMQ y JBoss Messaging .

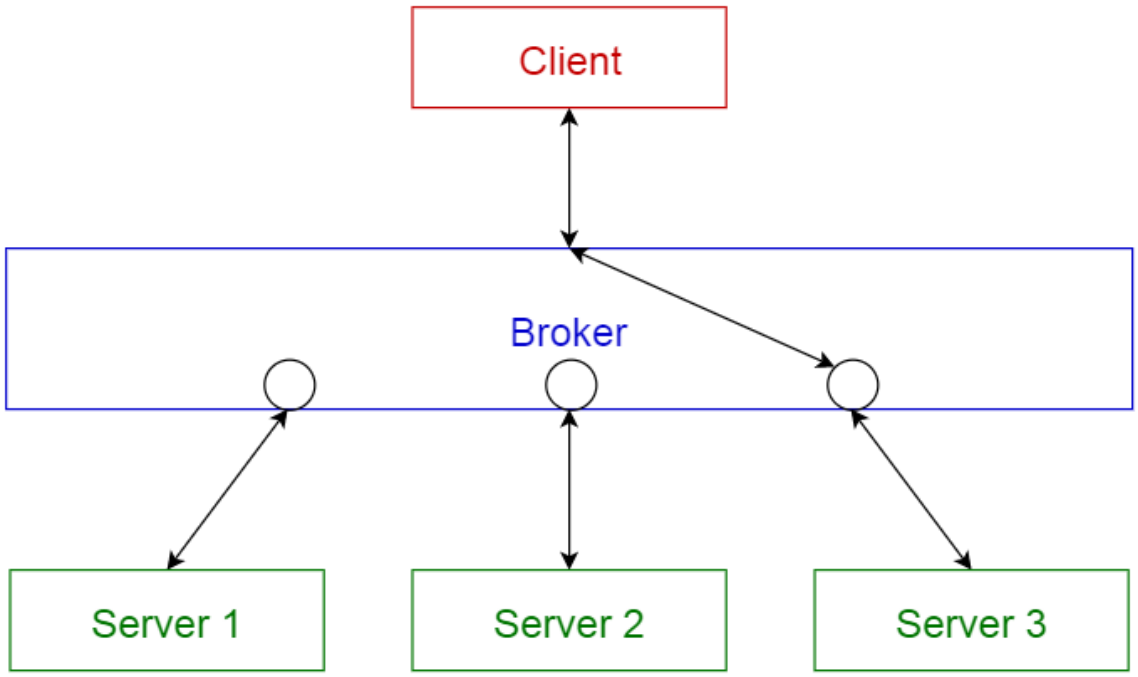


*Figura 8. Patrón de intermediario*

### 4.5.6 Patrón de igual a igual

En este patrón, los componentes individuales se conocen como pares. Los pares pueden funcionar tanto como un cliente, solicitando servicios de otros pares, y como un servidor, proporcionando servicios a otros pares. Un par puede actuar como un cliente o como un servidor o como ambos, y puede cambiar su rol dinámicamente con el tiempo [6].

**Uso:**

- Redes de intercambio de archivos como Gnutella y G2
- Protocolos multimedia como P2PTV y PDTP.

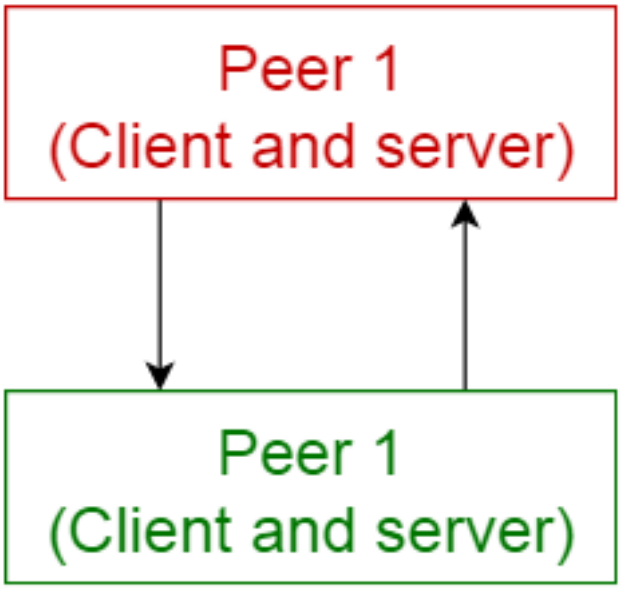


*Figura 9.Patrón de igual a igual*

### 4.5.7 Patrón de bus de evento

Este patrón trata principalmente con eventos y tiene 4 componentes principales: fuente de evento, escucha de evento, canal y bus de evento. Las fuentes publican mensajes en canales particulares en un bus de eventos. Los oyentes se suscriben a canales particulares. Los oyentes son notificados de los mensajes que se publican en un canal al que se han suscrito anteriormente [6].

**Uso:**

- Desarrollo de Android
- Servicios de notificación

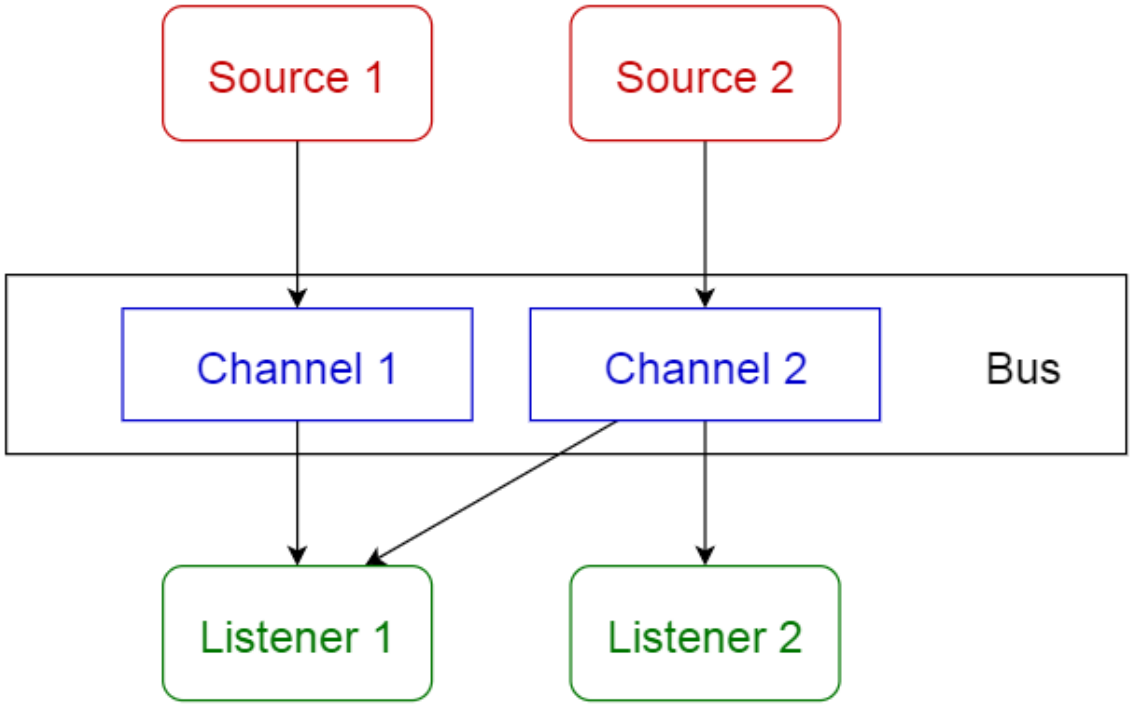


*Figura 10. Patrón de bus de evento*

### 4.5.8 Modelo-vista-controlador

Este patrón, también conocido como patrón MVC, divide una aplicación interactiva en 3 partes, como:

1. modelo — contiene la funcionalidad y los datos básicos
2. vista: muestra la información al usuario (se puede definir más de una vista)
3. controlador: maneja la entrada del usuario

Esto se hace para separar las representaciones internas de información de las formas en que se presenta y acepta la información del usuario. Desacopla los componentes y permite la reutilización eficiente del código[6].

**Uso:**

- Arquitectura para aplicaciones World Wide Web en los principales lenguajes de programación.
- Marcos web como Django y Rails.

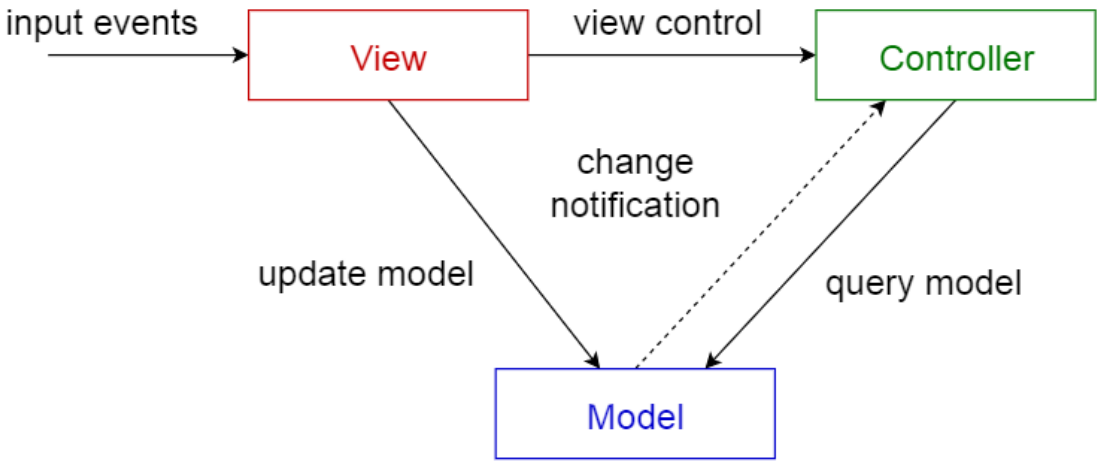


*Figura 11. Patrón MVC*

### 4.5.9 Patrón de pizarra

Este patrón es útil para problemas para los que no se conocen estrategias de solución deterministas. El patrón de pizarra consta de 3 componentes principales [6].

1. pizarra: una memoria global estructurada que contiene objetos del espacio de solución
2. fuente de conocimiento: módulos especializados con su propia representación
3. componente de control: selecciona, configura y ejecuta módulos.

Todos los componentes tienen acceso a la pizarra. Los componentes pueden producir nuevos objetos de datos que se agregan a la pizarra. Los componentes buscan tipos particulares de datos en la pizarra, y pueden encontrarlos por coincidencia de patrones con la fuente de conocimiento existente [6].

**Uso:**

- Reconocimiento de voz
- Identificación y seguimiento del vehículo
- Identificación de la estructura proteica
- Sonar señala la interpretación.

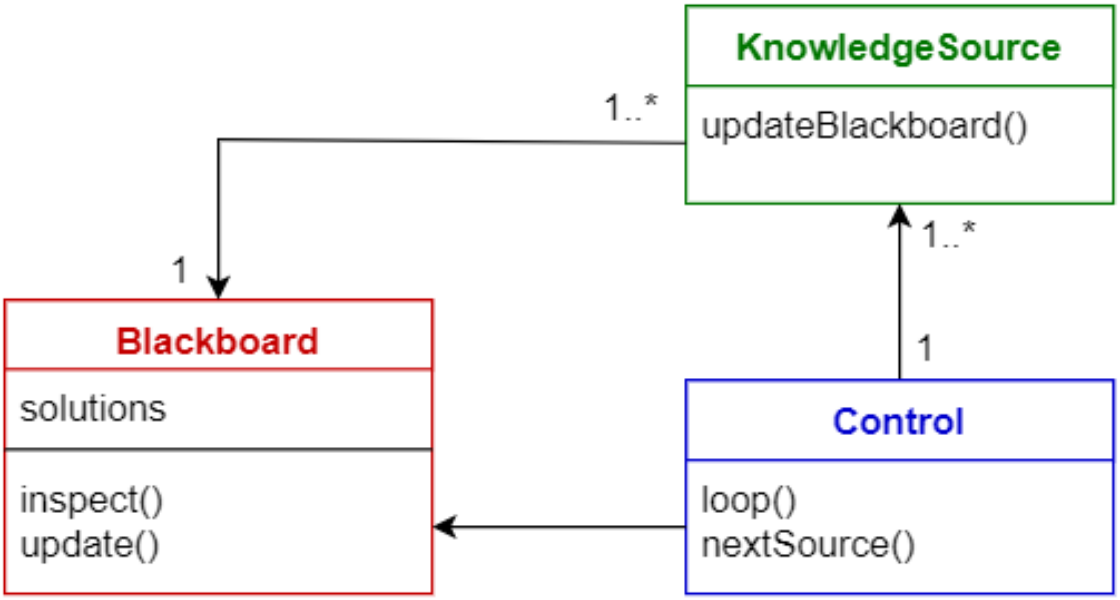


*Figura 12. Patrón de pizarra*

### 4.5.10 Patrón de intérprete

Este patrón se usa para diseñar un componente que interpreta programas escritos en un lenguaje dedicado. Especifica principalmente cómo evaluar las líneas de programas, conocidas como oraciones o expresiones escritas en un idioma particular. La idea básica es tener una clase para cada símbolo del idioma [6].

**Uso:**

- Lenguajes de consulta de base de datos como SQL.
- Idiomas utilizados para describir los protocolos de comunicación.

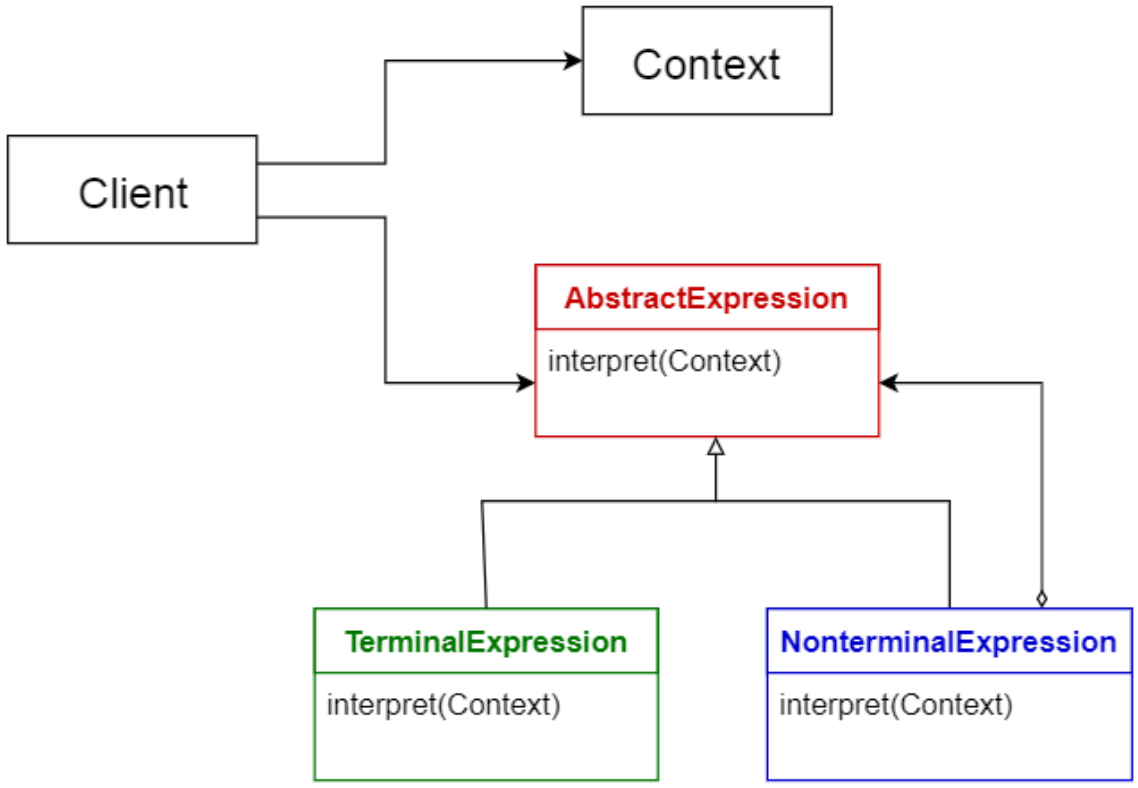


*Figura 13. Patrón de intérprete*

### 4.6 Gamificación

El concepto de gamificación es actualmente un tema candente en marketing. La gamificación toma elementos de videojuegos, como la mecánica y dinámica de los juegos, y los utiliza en contextos que no son juegos. Abarca la aplicación del pensamiento de juego y el diseño lúdico en áreas como el contexto de negocios, y varios estudios han demostrado su utilidad como herramienta motivadora [7].

### 4.7 Usabilidad

Usabilidad es un anglicismo que se ha introducido para referirse básicamente a la facilidad de uso de una aplicación o producto interactivo, definición que viene desde el campo de investigación relacionado con la interacción humano-computador (HCI). En el popular libro "No me hagas pensar" de Steve Krug, la usabilidad significa asegurarse de que alguna cosa trabaja bien, y que una persona con capacidad y experiencia promedio (o

incluso por debajo del promedio), puede usarla para el fin previsto sin que llegue a frustrarse en el intento [8].
La usabilidad es un concepto empírico, lo que significa que puede ser medida y evaluada, y por tanto no debe entenderse como un concepto abstracto, subjetivo o carente de significado. La mayoría de las investigaciones coinciden en que las métricas de usabilidad miden atributos o características de usabilidad que pueden ser medibles. En este sentido, se define que esos atributos son: la facilidad de aprendizaje, el recuerdo en el tiempo, la eficiencia en uso, la tasa de errores y la satisfacción [9].
La siguiente tabla muestra la relación de los atributos medibles de la usabilidad [9].

*Tabla 1. Metodologia de medicion para usabilidad*

| ATRIBUTO | SIGNIFICADO | FORMA DE MEDIR |
|---|---|---|
| Facilidad de aprendizaje | Implica cuán rápido y fácilmente los usuarios pueden comenzar a realizar un trabajo productivo con un sistema que usan por primera vez. | Tiempo que el usuario nuevo utiliza el sistema antes de alcanzar el nivel de eficiencia que tiene el usuario experto en el uso de la aplicación. |
| Recuerdo en el tiempo | Capacidad del sistema de permitir al usuario utilizar la aplicación siempre, sin tener que recordar su funcionamiento. | Tiempo requerido para concluir la actividad. |
| Eficiencia en su uso | Productividad del usuario con el uso del sistema. | Número de tareas por unidad de tiempo en que el usuario experto es capaz de utilizar el sistema. |
| Tasa de errores | Errores cometidos durante el uso del sistema y cuán fácil el usuario se recupera de ellos, tanto del número como el tipo de errores. | Número de errores cuando el usuario intenta hacer una tarea concreta y cómo se recupera del error. |
| Satisfacción | Opinión subjetiva que se forma el usuario acerca del tema. | Cuestionarios de satisfacción que llenan los usuarios. |

### 4.8 Experiencia de usuario

La noción de Experiencia del Usuario ha surgido como un nuevo concepto que hace énfasis en los aspectos emocionales, sensaciones, sentimientos,

valoración y satisfacción que resultan del uso del sistema (ISO:9241-210, 2010). La Experiencia del Usuario (UX) representa un cambio emergente del propio concepto de usabilidad, donde el objetivo no se limita a mejorar el rendimiento del usuario en la interacción, eficacia, eficiencia y facilidad de aprendizaje, sino que se intenta resolver el problema estratégico de la utilidad del producto y el problema psicológico del placer y el entusiasmo de su uso. Es importante señalar que la Experiencia del Usuario no constituye una disciplina cerrada y definida, sino un enfoque de trabajo abierto y multidisciplinar. La experiencia del usuario no se trata de un buen diseño industrial o interfaces de fantasía. Se trata de trascender la materia. Se trata de crear una verdadera experiencia a través de un dispositivo [8].

### 4.9 SOLID

En ingeniería de software, SOLID (Single responsaiblity, Open-closed, Liskov substitution, Interface segregation and Dependency inversion) es un acrónimo mnemónico introducido por Robert C. Martin a comienzos de la década del 2000 que representa cinco principios básicos de la programación orientada a objetos y el diseño. Cuando estos principios se aplican en conjunto es más probable que un desarrollador cree un sistema que sea fácil de mantener y ampliar en el tiempo. Los principios de SOLID son guías que pueden ser aplicadas en el desarrollo de software para eliminar código sucio provocando que el programador tenga que refactorizar el código fuente hasta que sea legible y extensible. Debe ser utilizado con el desarrollo guiado por pruebas y forma parte de la estrategia global del desarrollo ágil de software y programación adaptativa [10].

## 5. Metodología

La metodología propuesta para el desarrollo de la aplicación está basada en los requerimientos planteados por la IPS Neumomed y en la experiencia de investigaciones previas en aplicaciones móviles. El principal requerimiento dentro de la IPS es desarrollar una aplicación la cual sirva de canal para la comunicación del paciente con un asistente virtual desarrollado dentro de la IPS, que tenga la funcionalidad de favorecer la atención, seguimiento y control de los pacientes, además de facilitar la gestión de los pacientes dentro de la IPS Neumomed. Para lograr este objetivo la aplicación permite una interacción total del usuario con el bot, permitiendo registrar ejercicios, llenar formularios, recibir notificaciones push, recordatorios o mensajes del personal asistencial y hacer preguntas frecuentes.

La metodología se encuentra enmarcada en en cinco fases como se muestra en la figura _ denominadas: análisis, diseño, desarrollo, pruebas de funcionamiento y Despliegue. Esta metodología se propone de manera que se pueda llevar a cabo el cumplimiento de cada uno de los objetivos de este proyecto. La descripción de cada una de las actividades se muestra a continuación.

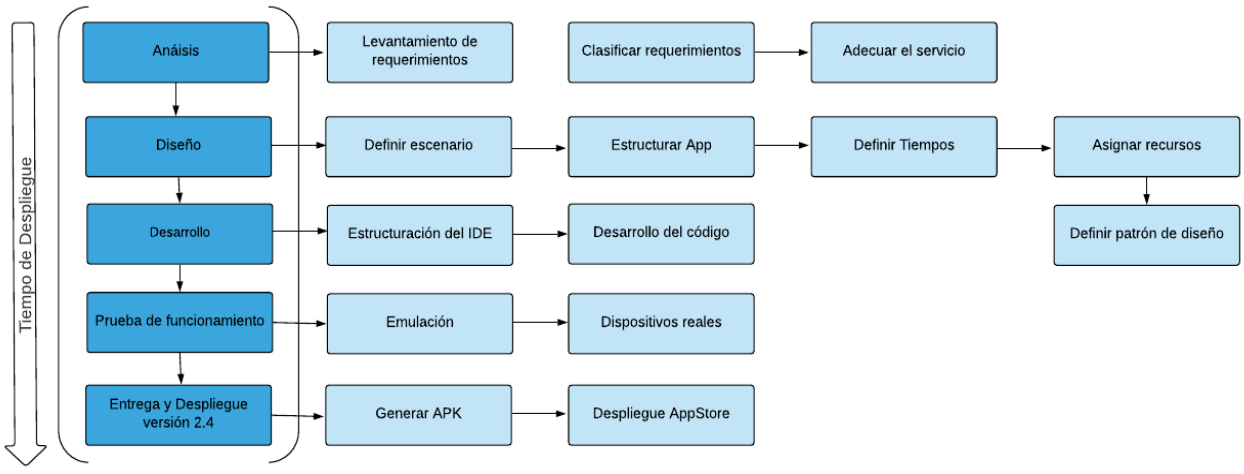


*Figura 14. Diagrama metodologia*

## 6. Resultados y análisis

Se muestra a continuación el desarrollo de cada uno de los pasos mostrados en la metodología.

### 6.1 Análisis

En la fase de Análisis se exponen detalladamente las peticiones o requerimientos de la IPS Neumomed, el propósito es definir las características del entorno de la aplicación con la información recopilada para mejorar la experiencia de usuario de la aplicación Doctor Neumo. Se realizaron tres tareas: levantamiento de requerimientos, clasificar los requerimientos y adecuar el servicio.

Para el levantamiento de los requerimientos se puso en evidencia las necesidades que se pretenden solucionar con las tecnologías móviles dentro de la IPS. Se expuso los recursos que se utilizaban en versiones anteriores y se señalan las características que debe tener la aplicación, principalmente para la comunicación con el asistente virtual.

Una vez identificados los requerimientos que debe tener la aplicacion Doctor Neumo se buscó clasificarlos. Una de las clasificaciones de los requerimientos se refiere a todo lo que rodea al servicio. Por ejemplo, las características técnicas del dispositivo móvil de los usuarios, el sistema operativo, la tecnología utilizada para la transferencia de información, el Sistema Manejador de Base de Datos, el formato de archivos, entre otros módulos que se utilizan en el servicio. Otra clasificación de requerimientos se basa en la interfaz gráfica de usuario y las adaptaciones que involucren la comunicación del usuario con la aplicación.

Por otra parte, se adecuó el servicio hacia los pacientes respiratorios que están dentro del programa de rehabilitación de Neumomed, teniendo en cuenta el público hacia el cual va dirigido, se genera una aplicación con una forma de uso intuitiva para garantizar la aceptación de esta en los pacientes.

### 6.2 Diseño

Se describieron cuatro actividades durante esta etapa. Las actividades corresponden a definir el escenario, estructurar la aplicación, asignar tiempos, definir recursos y definir un patrón de diseño.

El escenario definido para la aplicación se tomó orientado hacia la conexión de la información con el servidor. El dispositivo debe estar siempre conectado con el servidor para su correcto funcionamiento, no se almacenan datos o archivos en el móvil, la sincronización se realiza mediante la validación de formularios, usualmente se utiliza el Protocolo de Transferencia de Hipertexto (Hypertext Transfer Protocol, HTTP).
la estructuración de la aplicación se hace según las necesidades del proyecto. Se define las funcionalidades requeridas para la aplicación las cuales se muestran a continuación en la figura 15, estas son la base para el desarrollo de la aplicación.

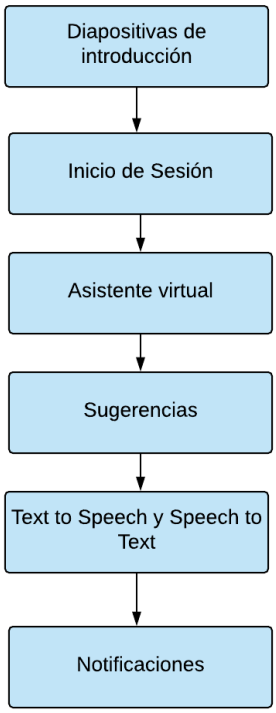


*Figura 15. Funcionalidades requeridas para la app*

A continuación se explica de forma breve cada funcionalidad mostrando los requerimientos de cada función.

### 6.2.1 Diapositivas de Introducción

La aplicación debe mostrarle al usuario, la primera vez que ingrese, un módulo con imágenes guiadas de cómo funciona la aplicación, el usuario debe ser capaz de moverse entre las diferentes imágenes guiadas y una vez finalice el módulo de introducción ingresar a realizar el login.

### 6.2.2 Inicio de Sesión

La aplicación debe permitir a los pacientes loguearse en la app para interactuar con el asistente virtual y acceder a los mensajes, formularios y sugerencias específicas para ese usuario. El módulo debe loguearse con los datos de inicio de sesión definidos que son cédula del paciente y año de nacimiento, cuando los datos sean incorrectos se muestra un mensaje de alerta al usuario.

### 6.2.3 Asistente Virtual

Esta es la tarea más importante de la aplicación ya que, la aplicación es uno de los canales por el cual las personas pueden contactar el asistente virtual. Por tal razón, la aplicación debe permitirle al usuario hablar con el asistente virtual, escribirle, etc. Todas estas interacciones deben estar orientadas a una conversación, por tanto, siempre debe estar la posibilidad que el asistente intervenga sobre la tarea que se esté ejecutando.

### 6.2.4 Sugerencias

El asistente virtual debe mostrar las sugerencias de acciones o respuesta al usuario siempre que existan, las sugerencias se deberán mostrar con iconos si los tiene o solo texto.

### 6.2.5 Text to Speech y Speech to Text

La aplicación debe ser capaz de implementar lectura de la respuesta del asistente virtual y procesar los comandos que el usuario realice por medio de dictado por voz.

### 6.2.6 Notificaciones

El Sistema de inteligencia artificial y personal administrativo de Neumomed, debe tener la posibilidad de enviar notificaciones push a una o más personas que cuenten con la aplicación. De esta manera en la aplicación se debe visualizar al asistente virtual recibiendo dicha notificación, gestionandola para el usuario y permitirle ver un historial de notificaciones no leídas o mensaje específico que el usuario solicite por medio de la interacción con el asistente virtual. La interrupción del bot debe tener en cuenta el proceso en el que se encuentre el usuario y así no interferir en la actividad o reporte que se está realizando.

Para asignar los tiempos se establecieron plazos para cada una de las actividades restantes, con el objetivo de terminar la aplicación a tiempo para su salida a producción.

Se asignan los recursos para realizar cada actividad y alcanzar los objetivos propuestos, se consideró recursos humanos como la disposición del personal asistencial de Neumomed para obtener información sobre la rehabilitación pulmonar, financieros como la disposición de servicios de google como el entrenamiento del bot por parte de Dialogflow y tecnológicos como el gasto computacional físico. Además, se deben seleccionar las herramientas para el desarrollo de la aplicación móvil.

A continuación se proporciona una descripción general de los recursos utilizados para lograr cada funcionalidad.

*Tabla 2. Recursos para construcción app nativa Android*

| TECNOLOGÍA | APLICACIÓN | VERSIÓN |
|---|---|---|
| Android SDK | Proporciona las API disponibles en el marco de Android, incluidos los widgets de UI y varios frameworks de servicios | Min SDK: 17<br>Target SDK: 27 |
| Firebase | Firebase Cloud Messaging (FCM) es una solución de mensajería multiplataforma que te permite enviar mensajes de forma segura y gratuita. | 11.8.0 |
| Retrofit | Retrofit es un cliente REST para Android que permite hacer peticiones y gestionar diferentes tipos de parámetros y parsear automáticamente la respuestas. | 2.5.0 |
| Glide | Glide es una librería de administración de medios y carga de imágenes rápida y eficiente, que envuelve la decodificación de medios, la memoria caché del dispositivo y el disco, y la agrupación de | 4.4.0 |

| | | |
|---|---|---|
| | recursos en una interfaz simple y fácil de usar. | |
| Lottie | Lottie es una librería para que toma las animaciones de Adobe After Effects exportadas como json con Bodymovin y las reproduce de forma nativa en el móvil y en la web. | 2.5.0 |
| Realm | Realm es un motor de base de datos que permite crear bases de datos relacionales de forma sencilla. | 2.5.0 |
| Gson | Una biblioteca de serialización / deserialización de Java para convertir objetos Java en JSON y viceversa. | 2.5.0 |
| Gradle | Sistema de automatización de construcción. | 3.2.1 |
| RxAndroid | Facilita el manejo de flujos de datos y eventos por medio de la programación reactiva, a partir de una combinación de el patrón Observer, el patrón Iterator, y características de la Programación Funcional simplificando la programación asíncrona. | 2.1.16 |

Finalizando esta etapa, se definió un patrón de diseño para flexibilizar, modular y reutilizar lo desarrollado; la selección del patrón de diseño es acorde con el escenario del servicio. El Patrón de arquitectura que se permite implementar en esta aplicación se llama **Modelo – Vista – Presentador** (MVP) que es una derivación del patrón arquitectónico modelo-vista-controlador (MVC). MVP es utilizado mayoritariamente para construir interfaces de usuario y el presentador asume la funcionalidad de mediador para la comunicación entre interfaces, toda la lógica de presentación es colocada al presentador. Las capas están definidas por:

- **Modelo:** Esta capa gestiona los datos. Son las clases que denominaremos de lógica de negocio.

- **Vista:** Se encarga de mostrar los datos. Aquí se encontrarán nuestros Fragmentos y Vistas.

- **Presentador:** Se sitúa entre el modelo y la vista, permitiendo conectar la interfaz gráfica con los datos.


*Figura 16. Patrón Modelo-Vista-Presentador*

Este modelo permite una vista desacoplada del resto de componentes, una lógica de presentación testable de forma unitaria y vistas y presenters reusables.

### 6.3 Desarrollo

En esta etapa se realizan dos actividades, estas son la estructuración del IDE y el desarrollo del código encaminadas a adaptar la aplicación para que introduzca la estrategia de gamificación.

En la estructuración del IDE, El proyecto a nivel de paquetes de Android Studio se estructura de la siguiente manera:

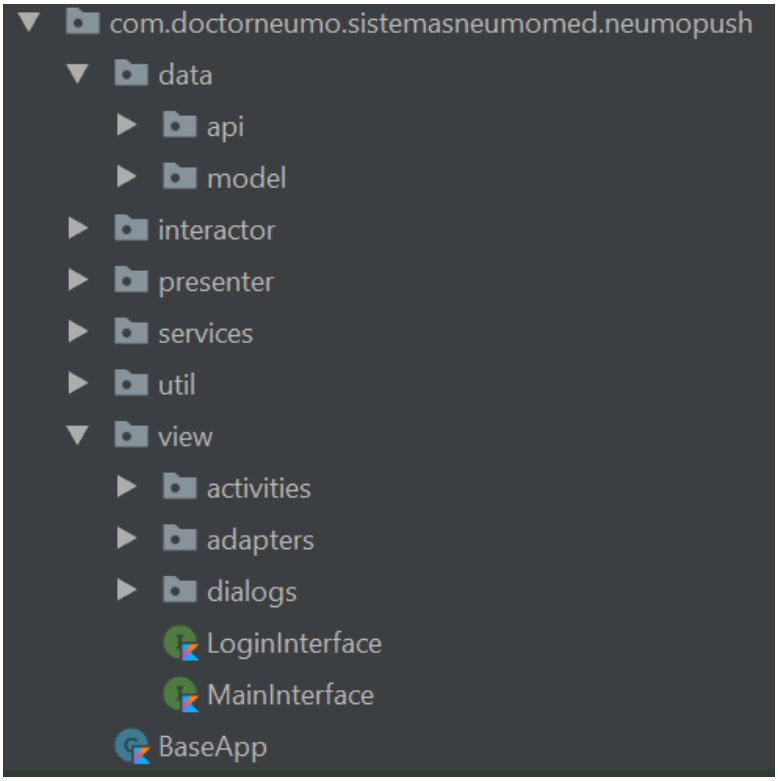


*Figura 17. Distribución de la arquitectura en el proyecto Doctor Neumo*

- **Data:** paquete donde se estructura todo el componente "módelo" del patrón MVP, en este paquete se debe configurar la fuente de datos, el cliente para consumir esa fuente de datos y los objetos o entidades necesarios para parsear los datos. Cómo se ve en la imagen, el paquete data a su vez está compuesto por el paquete api y el paquete model, en el paquete api se configura el cliente Retrofit junto con la interfaz de las apis a consumir que proveerán los datos de la aplicación y en el paquete model, se crean las entidades /objetos de los datos de la aplicación. Se muestra a continucion la configuracion para el cliente Retrofit y las apis que se utiliza. El codigo fue escrito utiizando buenas practicas de programacion basadas en SOLID.

```
class NeumobotClient : ApiClient(), NeumobotService {
    override fun graphicRhb(documento: String, labelTime: String): Single<GraphicResponse> {
        Timber.i("El documento consultado es : $documento")
        return getApiService()!!.graphicRhb(GraphicBody(documento,labelTime))
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
    }


    override fun revisarMensajes(documento: String): Single<ReviewMessageResponse> {
        return getApiService()!!.revisarMensajes(ReviewMessageBody(documento))
```

```
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
        }

        override fun registerFCM(documento: String, token: String): Single<TokenResponse> {
            Timber.i("registerFCM documento: ${documento}, token: ${token}")
            return getApiService()!!.registerTokenFCM(RegisterFCMBody(documento, "create_login", token,
    "paciente"))
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
        }

        override fun login(contrasena: String, documento: Int): Single<LoginResponse> {
            Timber.i("login contraseña: $contrasena y documento : $documento")
            return getApiService()!!.login(LoginBody(contrasena, documento))
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
        }

        override fun comunucacionBot(message: String, session_id: String): Single<DialogFlowResponse>
    {

            return getApiService()!!.comunicacionBot(DialogflowBody("text", message, session_id, "app"))
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
            Timber.i("El mensaje es: $message, session_id es: $session_id")

        }

        override fun markNotificationAsRead(messageId: String, cedula: String): Single<TokenResponse>
    {

            return getApiService()!!.markNotificationAsRead(NotificationReadBody("read_confirmation",
    messageId, cedula))
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
        }

        override fun bandejaEntrada(documento: String): Single<ReviewMessageResponse> {
            return
    getApiService()!!.revisarBandeja(ReviewMessageBody(documento)).subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
        }


    }
```

- **Interactor:** Paquete que contiene los archivos que comunicaran a la capa de servicio o modelo con la capa presentador, por medio de los archivos de este paquete el presentador se subscribirá a una tarea para traer datos y por medio de un observador el hilo responderá cuando los datos se encuentren disponibles.

```
class MainInteractor(val mainInteractorInterface: MainInteractorInterface) {
    val neumobotService: NeumobotService = NeumobotClient()

    fun comunicacionBot(message: String, session_id: String) {

        neumobotService.comunucacionBot(message, session_id).subscribeWith(object :
```

```
DisposableSingleObserver<DialogFlowResponse>() {
            override fun onSuccess(response: DialogFlowResponse) {
                Timber.i("Consulta exitosa ${response.response}")
                mainInteractorInterface.onSuccessDialogFlowResponse(response)
            }

            override fun onError(e: Throwable) {
                if (e is HttpException) {
                    val errorJsonString = e.response().errorBody()?.string()
                    val message = JsonParser().parse(errorJsonString).asJsonObject["message"].asString
                    Timber.i("Error es $message")
                    mainInteractorInterface.onErrorDialogFlowResponse(message)
                } else{
                    Timber.i("error en la consulta es ${e.message}")
                    //mainInteractorInterface.onErrorDialogFlowResponse(e.message)
                    mainInteractorInterface.onErrorDialogFlowResponse("Se ha perdido la comucación. Por favor
vuelve a enviar un Hola.")
                }



            }
        })
    }

    fun markNotificationAsRead(messageId: String, cedula: String) {

        neumobotService.markNotificationAsRead(messageId, cedula).subscribeWith(object :
DisposableSingleObserver<TokenResponse>() {
            override fun onSuccess(response: TokenResponse) {
                Timber.i("Consulta exitosa ${response.message}")
                mainInteractorInterface.onSuccessMarkNotification(response)
            }

            override fun onError(e: Throwable) {
                mainInteractorInterface.onErrorMarkNotification(e.message)
            }


        })

    }

    fun revisarMensajes (documento : String){
        Timber.i("ConsultaSessssion $documento")
        neumobotService.revisarMensajes(documento).subscribeWith(object :
DisposableSingleObserver<ReviewMessageResponse>(){
            override fun onError(e: Throwable) {
                Timber.i("Revisa mensajes: negativo")
                if (e is HttpException) {
                    val errorJsonString = e.response().errorBody()?.string()
                    val message = JsonParser().parse(errorJsonString).asJsonObject["message"].asString
                    Timber.i("Error es $message")
                    mainInteractorInterface.onErrorNotReadYet(message)
                } else{
                    Timber.i("error en la consulta es ${e.message}")
                    mainInteractorInterface.onErrorNotReadYet(e.message)
                }
            }
            override fun onSuccess(messageReview: ReviewMessageResponse) {
                Timber.i("Revisa mensajes: positivo")
                mainInteractorInterface.onSuccessNotReadYet(messageReview)
            }
```

```
        })
    }

    fun bandejaEntrada (documento: String){
        neumobotService.bandejaEntrada(documento).subscribeWith(object:
DisposableSingleObserver<ReviewMessageResponse>(){
            override fun onError(e: Throwable) {
                if(e is HttpException){
                    val errorJsonString = e.response().errorBody()?.string()
                    val message = JsonParser().parse(errorJsonString).asJsonObject["message"].asString
                    mainInteractorInterface.onErrorNotReadYet(message)
                }else{
                    mainInteractorInterface.onErrorNotReadYet(e.message)
                }

            }

            override fun onSuccess(messageReview: ReviewMessageResponse) {
                mainInteractorInterface.onSuccessNotReadYet(messageReview)
            }
        })
    }

    fun graphicRhb (documento: String, labelTime : String){
        Timber.i("datos de entrada son $documento y $labelTime")
        neumobotService.graphicRhb(documento,labelTime).subscribeWith(object:
DisposableSingleObserver<GraphicResponse>(){
            override fun onSuccess(graphicResponse: GraphicResponse) {
                Timber.i("graphic on succes")
                mainInteractorInterface.onSuccessGraphicRhb(graphicResponse)
            }

            override fun onError(e: Throwable) {
                if(e is HttpException){
                    val errorJsonString = e.response().errorBody()?.string()
                    val message = JsonParser().parse(errorJsonString).asJsonObject["message"].asString
                    mainInteractorInterface.onErrorNotReadYet(message)
                    Timber.i("graphic on error $message")
                }else{
                    mainInteractorInterface.onErrorNotReadYet(e.message)
                }

            }
        })


    }

}
```

- **Presenter:** Paquete que contiene la estructura o archivos del componente "presentador" del patrón MVP, es en esta capa donde se crea la comunicación entre la vista y el módelo, acá se ejecutan los casos de uso.

```
class MainPresenterImp(val view: MainInterface, val context: Context) : MainInteractorInterface, MainPresenter
{



    val interactor: MainInteractor = MainInteractor(this)


    override fun comunicacionBot(message: String, session_id: String) {
        Timber.i("El session_id es: $session_id el mensaje es: $message")
        interactor.comunicacionBot(message, session_id)
    }

    override fun markNotificationAsRead(documento: String, id_notificacion: String) {
        interactor.markNotificationAsRead(documento,id_notificacion)
    }

    override fun onSuccessDialogFlowResponse(dialogFlowResponse: DialogFlowResponse) {

        if(dialogFlowResponse.variables_front.data != null){

            Timber.i("Salida inminente ${dialogFlowResponse.variables_front.data.action}")
            Timber.i("sugerencias notificaciones ${dialogFlowResponse.sugerencias}")


            if(dialogFlowResponse.variables_front.data.action != null){
                Timber.i("el action es ${dialogFlowResponse.variables_front.data.action}")

                when (dialogFlowResponse.variables_front.data.action) {

                    ACTION_EA_SELECT -> {
//                        view.setChatBot(dialogFlowResponse)
                        view.hideSplash()
//                        view.textToSpeech(dialogFlowResponse.response)
//                        view.showSugerencias(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
                        view.eAerobicSelected(dialogFlowResponse)
                    }

                    ACTION_INICIO1 -> {
                        val mDataUser = LoginResponseBody().queryLast()
                      interactor.comunicacionBot(mDataUser!!.documento, mDataUser!!.documento)
                      // interactor.comunicacionBot("1022096890", mDataUser!!.documento)
//                        interactor.comunicacionBot("1017 185696", mDataUser!!.documento)

//                        view.showSplash()

                    }

                    ACTION_INICIO2 -> {
                        view.showSugerencias(dialogFlowResponse, ACTION_INICIO2)
                        view.showUI(dialogFlowResponse)
                        view.hideSplash()
//                        view.textToSpeech(dialogFlowResponse.response)
                    }

                    ACTION_RG->{
                        view.setChatBot(dialogFlowResponse)
                        view.hideSplash()
//                        view.textToSpeech(dialogFlowResponse.response)
                        view.showSugerencias(dialogFlowResponse, ACTION_INICIO3)
                    }
```

```
                ACTION_SEE_NOTIFICATIONS->{
                    val mDataUser = LoginResponseBody().queryLast()
                    Timber.i("Mensaje review es : ${mDataUser!!.documento}")
                    interactor.revisarMensajes(mDataUser!!.documento)
//                    nryinteractor.revisarMensajes(mDataUser!!.documento)
                    view.showSugerencias(dialogFlowResponse, ACTION_SEE_NOTIFICATIONS)
//                    view.setChatBot(dialogFlowResponse)
                }

                ACTION_EJERCICIO_AEROBICO  ->{
                    view.resetAerobics(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
//                    view.AerobicsSug(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
                }

                ACTION_EJERCICIO_AEROBICO_FB  ->{
//                    view.setChatBot(dialogFlowResponse)
                    view.resetAerobics(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
//                    view.AerobicsSug(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
                }

                ACTION_EJER_RESPIRATORIO -> {
                    view.resetAerobics(dialogFlowResponse)
                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_EJER_RESPIRATORIO)
                }

                ACTION_PESO_ADICIONAL_BRAZOS -> {
                    view.resetDisnea(dialogFlowResponse)
                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_PESO_ADICIONAL_BRAZOS)
                }

                ACTION_EJER_BRAZOS -> {
                    view.resetDisnea(dialogFlowResponse)
                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_EJER_BRAZOS)
                }



                ACTION_END_RHB ->{

                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_END_RHB)
                    view.resetDisnea(dialogFlowResponse)
//                    view.graphicRhb(dialogFlowResponse)
//                    view.dyspneaMeter(dialogFlowResponse, ACTION_DISNEA)
```

```
                }

                ACTION_EJ_PIERNAS -> {

                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_EJ_PIERNAS)
                    view.resetDisnea(dialogFlowResponse)
//                    view.dyspneaMeter(dialogFlowResponse, ACTION_DISNEA)
                }

                ACTION_DISNEA -> {
//                    val mDataUser = LoginResponseBody().queryLast()
//                    interactor.graphicRhb(mDataUser!!.documento,"semanas")
//                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
//                    view.showSugerencias(dialogFlowResponse, ACTION_DISNEA)
                    view.selectDyspneaMeter(dialogFlowResponse)


                }

                ACTION_DISNEA_FB ->{
                    view.setChatBot(dialogFlowResponse)
                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_DISNEA_FB)
                    view.resetDisnea(dialogFlowResponse)
//                    view.graphicRhb(dialogFlowResponse)
//                    view.dyspneaMeter(dialogFlowResponse, ACTION_DISNEA_FB)
                }

                ACTION_MENSAJES->{
                    val mDataUser = LoginResponseBody().queryLast()
                    interactor.bandejaEntrada(mDataUser!!.documento)
                    view.showSugerencias(dialogFlowResponse, ACTION_MENSAJES)

                }

                else -> {

                    if(dialogFlowResponse != null){
                        view.setChatBot(dialogFlowResponse)
                    }

                    view.hideSplash()
//                    view.textToSpeech(dialogFlowResponse.response)
                    view.showSugerencias(dialogFlowResponse, ACTION_INICIO3)
                }
            }
        }else{
            Timber.i("El action es null")
//            view.textToSpeech(dialogFlowResponse.response)
            view.setChatBot(dialogFlowResponse)
            view.showSugerencias(dialogFlowResponse, ACTION_INICIO3)
            view.hideSplash()
        }


    }else{
```

```
//            view.textToSpeech(dialogFlowResponse.response)
            view.setChatBot(dialogFlowResponse)
            view.showSugerencias(dialogFlowResponse, ACTION_RG)
            view.hideSplash()
        }

    }

    override fun onErrorDialogFlowResponse(mensaje: String?) {
        view.showError(mensaje)
        Timber.i(mensaje)
    }

    override fun onSuccessMarkNotification(tokenResponse: TokenResponse) {
        view.updateNotificationAdapter()
    }

    override fun onErrorMarkNotification(mensaje: String?) {

    }
    override fun onSuccessNotReadYet(messageReview: ReviewMessageResponse) {
        view.setChatBotMessageReview(messageReview)
    }

    override fun onErrorNotReadYet(mensaje: String?) {

    }

    override fun obtenerDocumento(): String {
        val dataUser = LoginResponseBody().queryLast()
        var documento = dataUser!!.documento
        return documento
    }

    override fun onSuccessGraphicRhb(graphicResponse: GraphicResponse) {
        view.graphicRhbL(graphicResponse)
    }

}
```

- **Vista:** Paquete que contiene todos los archivos correspondientes a UI o interacción con el usuario, estos son adapters, fragments, diálogos o activities.

```
class MainActivity : AppCompatActivity(), MainInterface{


    private var adapter: MessageAdapter? = null
    private var mensajes = ArrayList<DatosChat>()


    private lateinit var mRealm: Realm
    private var mDataUser: LoginResponseBody? = null
    private val presenter: MainPresenter = MainPresenterImp(this, this)
    private val mSplashDialog: SplashDialog by lazy {
        SplashDialog(this)
    }

    private lateinit var mTextToSpeech: TextToSpeech
    var adapter_sugerencias: SugerenciasAdapter? = null
```

```
    val  arrayListSugPosition = ArrayList<Int>()
    val arrayListSug = ArrayList<String>()
    private var audioInterrupted = false
    private lateinit var speechRecognizerViewModel: SpeechRecognizerViewModel
    private lateinit var mSnackbarInternet: Snackbar

    private var textOrSpeech = TEXT_ON
    private var oneTime = 1

    private var width: Int = 0
    private var height: Int = 0
    var fecha: String = ""
    var tiempo: String = ""
    var notification_option: String = ""
    var notification_video: String = ""
    var notification_preview: String = ""
    var size_notificacion: Int = 0
    var notifications_messages = ArrayList<DatosChat>()
    var removeAction: Int = 0
    var notification_speech: Int = 0
    var position: Int = 0
    var multimedia: String? = ""
    var tipoMultimedia: String? = ""
    var eAerobic: Int = 0
    var eDisnea: Int = 0
    var p1 = 0
    var p2 = 0
    var sugerenciasAerobicas = 0
    var sugerenciasDisnea = 0
    var selectItemOnClick = 0
    var mensajeSegmentado = ""




    private val mediaPlayer = MediaPlayer()
    private var runnable: Runnable? = null
    private var pause: Boolean = false
    private lateinit var dialog: ProgressDialog
    private lateinit var dialog1: ProgressDialog
    private var i = 0
    companion object {
        var notification: Boolean = false
        var estado_audio: Int = 0
    }

    private var mcontext: Context? = this
    private lateinit var estado_vista: View
    private lateinit var solo_vista: View
    private var sendMessage: Int = 0

    private var x: Double = -5.0
    private var y: Double = 0.0
    private lateinit var firebaseAnalytics: FirebaseAnalytics

    private var bandInit: Boolean = true
    private var Userid: String = ""

    private var actionMensaje: String = ""

    var ejercicio_card = ArrayList<DatosChat>()
    var disnea_card = ArrayList<DatosChat>()
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mcontext = this
        //Fabrics
//        Fabric.with(this, Crashlytics())
       // scroller.post(Runnable { scroller.fullScroll(ScrollView.FOCUS_DOWN) })
        //Block Doze
        window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON)

        // Global
        application
        mRealm = Realm.getDefaultInstance()
        mDataUser = LoginResponseBody().queryLast()

        // Init
        configureRecycler()
        configureTTSClient()
        botInit()
        setupSpeechViewModel()

        //scrollStateChanged

        recyclerView_chatbot.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView?, dx: Int, dy: Int) {
                super.onScrolled(recyclerView, dx, dy)
                Timber.i("se captura el evento scroll $dx, $dy")
                if(dy < -1 ) {
                    if (speechRecognizerViewModel.isListening) {
                        speechRecognizerViewModel.stopListening()
                    }

                }
            }
        })


        //Screen Size Recognition
        val metrics = DisplayMetrics()
        windowManager.defaultDisplay.getMetrics(metrics)
        width = metrics.widthPixels
        height = metrics.heightPixels

        // Events
        activity_main_boton_hablar.setOnClickListener {
            textOrSpeech = SPEECH_ON
            recognizeVoice()
        }
        activity_main_boton_teclado.setOnClickListener {
            if (speechRecognizerViewModel.isListening) {
                speechRecognizerViewModel.stopListening()
            }
            enableWrite()
        }

        activity_main_boton_neumo.setOnClickListener { mostrarAyuda() }
        activity_main_enviar_mensaje.setOnClickListener { sendMessage(activity_main_edit_text.text.toString())
}
        activity_main_edit_text.setOnEditTextImeBackListener(object : EditTextImeBackListener {
```

```
            override fun onImeBack(ctrl: CustomEditText, text: String) {
                if (buttons_teclado.visibility == View.VISIBLE) {
                    buttons_teclado.gone()
                    buttons_navigate.visible()
                }
            }
        })
        if(notification == true){
            new_notification.gone()
        }

        // Obtain the FirebaseAnalytics instance.
//        firebaseAnalytics = FirebaseAnalytics.getInstance(this)
//
//        if(bandInit == true){
//            bandInit = false
//            Userid = presenter.obtenerDocumento()
//        }
//
//        firebaseAnalytics.setUserProperty("userId", Userid)


    }

    override fun onStart() {
        super.onStart()
        mcontext = this
        configureSnackBarInternet()
        // Broadcast that receives the incoming notifications
        LocalBroadcastManager.getInstance(this).registerReceiver(mMessageReceiver,
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION))
    }

    override fun onStop() {
        super.onStop()
        // Unregisters from the receiver before ending the activity
        LocalBroadcastManager.getInstance(this).unregisterReceiver(mMessageReceiver)
    }

    private fun configureRecycler() {
        recyclerView_chatbot.setHasFixedSize(true)
        recyclerView_chatbot.addOnLayoutChangeListener { p0, left, top, right, bottom, oldLeft, oldTop, oldRight,
oldBottom ->
            if (bottom < oldBottom) {
                recyclerView_chatbot.postDelayed(Runnable {
                    recyclerView_chatbot.smoothScrollToPosition(
                            recyclerView_chatbot.adapter.itemCount - 1)
                }, 900)
            }
        }
    }


    /**
     * Receives the incoming notifications and takes the needed action
     */
    private val mMessageReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) {
            Timber.i("BroadcastReceiver onReceive")
            if (intent.action == ConnectivityManager.CONNECTIVITY_ACTION) {
                connectivityChanges(intent)
            }
```

```
        }
    }

    /**
     * Method that checks if extras received are related to any type of notification
     */
    private fun connectivityChanges(intent: Intent) {
        val networkInfo =
intent.getParcelableExtra<NetworkInfo>(ConnectivityManager.EXTRA_NETWORK_INFO)
        if (networkInfo != null && networkInfo.detailedState == NetworkInfo.DetailedState.CONNECTED) {
            Timber.i("Broadcast internet")
            mSnackbarInternet.dismiss()
        } else if (networkInfo != null && networkInfo.detailedState ==
NetworkInfo.DetailedState.DISCONNECTED) {
            Timber.i("Broadcast sin  internet")
            mSnackbarInternet.show()
        }
    }

    private fun configureSnackBarInternet() {
        mSnackbarInternet = Snackbar.make(
                activity_main_presentation, // Parent view
                getString(R.string.error_internet_corto), // Message to show
                Snackbar.LENGTH_INDEFINITE // How long to display the message.
        )

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR1)

mSnackbarInternet.view.findViewById<TextView>(android.support.design.R.id.snackbar_text).textAlignment =
View.TEXT_ALIGNMENT_CENTER
        else
            mSnackbarInternet.view.findViewById<TextView>(android.support.design.R.id.snackbar_text).gravity =
Gravity.CENTER_HORIZONTAL

        if (isOnlineNet(this)) {
            mSnackbarInternet.dismiss()
        } else {
            mSnackbarInternet.show()
        }

    }

    private fun hideAnimationListener() {
        lottieAnimationView_micToListener.gone()
        lottieAnimationView_listenertoMic.gone()
    }

    private fun configureTTSClient() {
        mTextToSpeech = configureTTS(this)
        mTextToSpeech.setOnUtteranceProgressListener(object : UtteranceProgressListener() {
            override fun onStart(utteranceId: String) {
                audioInterrupted = false
                hideAnimationListener()
            }

            override fun onDone(utteranceId: String) {
                Timber.i("var audio interrupted es $audioInterrupted")
                this@MainActivity.runOnUiThread(Runnable {
                    if (!audioInterrupted) {
                        if (!speechRecognizerViewModel.isListening) recognizeVoice()
//                        val handler = Handler()
//                        handler.postDelayed({
```

```
//                        Timber.i("el mensaje se termina de leer")
//                        if (!speechRecognizerViewModel.isListening) recognizeVoice()
//                    }, 10)

                }


            })
        }

        override fun onError(utteranceId: String) {
        }

        override fun onStop(utteranceId: String?, interrupted: Boolean) {
            hideAnimationListener()
            Timber.i("se llama el metodo onstop de reproducit")
            audioInterrupted = true
        }
    })
}

private fun botInit() {
    if (isOnlineNet(this)) {
        presenter.comunicacionBot(TRIGGER_HOLA, mDataUser?.documento!!)
    } else {
        showDialogError(resources.getString(R.string.error_internet))
    }
}

override fun showError(error: String?) {
    showDialogError(error)
}

private fun showDialogError(error: String?) {
    if (!error.isNullOrEmpty()) {
        val dialog = AlertDialog()
        var arguments = Bundle()
        arguments.putString(DIALOG_MESSAGE, error)
        dialog.arguments = arguments
        dialog.show(supportFragmentManager, "dialogo")
    }
}

private fun setupSpeechViewModel() {
    speechRecognizerViewModel =
ViewModelProviders.of(this).get(SpeechRecognizerViewModel::class.java)
    speechRecognizerViewModel.getViewState().observe(this, Observer<ViewState> { viewState ->
        render(viewState)
    })
}

private fun render(uiOutput: ViewState?) {
    Timber.i("render es ${uiOutput?.spokenText}")
    Timber.i("Lo que se debe visualizar es: ${uiOutput!!.spokenView}")
    Timber.i("oneTime $oneTime")
    item_textView_speech.visible()
    item_textView_speech.text = uiOutput.spokenView

    if (uiOutput.complete_silence == TIME_SILENCE) {
        Timber.i("onEndOfSpeech")
        hideAnimationListener()
    }
```

```
        if (!uiOutput.spokenView.isNullOrEmpty()) {
            recyclerView_sugerencias.gone()
        }
        if (uiOutput == null) return
        if (uiOutput.spokenText.isNullOrEmpty()) {
            return
        } else {
            if (uiOutput != null && oneTime == 1) {
                sendMessage(uiOutput.spokenText)
                notification_speech = SPEECH_ON
                textOrSpeech = SPEECH_ON


                item_textView_speech.invisible()
                speechRecognizerViewModel.stopListening()
                oneTime += 1
            }
            if (!uiOutput.isListening) {
//                item_textView_speech.invisible()
                Timber.i("esta escuchando? : ${uiOutput.isListening}")
            }
            if (uiOutput.spokenText == "adiós") {
                item_textView_speech.invisible()
            }
        }
    }

    override fun recognizeVoice() {
        if (isOnlineNet(this)) {
            if (checkRecordPermission()) {
                if (textOrSpeech == SPEECH_ON) {
                    if (speechRecognizerViewModel.isListening) {
                        Timber.i("pare de escuchar")
                        speechRecognizerViewModel.stopListening()
                        item_textView_speech.invisible()
                    } else {
                        Timber.i("start escuchar")
                        speechRecognizerViewModel.startListening()
                        oneTime = 1
                        lottieAnimationView_micToListener.visible()

lottieAnimationView_micToListener.setAnimation("Microfono_Propuesta_B_Waiting_Load_V2.json")
                        lottieAnimationView_micToListener.setOnClickListener {
                            speechRecognizerViewModel.stopListening()
                            lottieAnimationView_micToListener.gone()
                        }
                    }
                }else if(textOrSpeech == TEXT_ON){
                    speechRecognizerViewModel.stopListening()
                }
            } else {
                makePermissionRequest()
            }
        } else {
            showDialogError(resources.getString(R.string.error_internet))
        }
    }

    private fun checkRecordPermission(): Boolean {
        val permission = ContextCompat.checkSelfPermission(this,
                Manifest.permission.RECORD_AUDIO)
```

```
        return permission == PackageManager.PERMISSION_GRANTED
    }

    private fun makePermissionRequest() {
        ActivityCompat.requestPermissions(this,
                arrayOf(Manifest.permission.RECORD_AUDIO),
                RECORD_REQUEST_CODE)
    }

    override fun onRequestPermissionsResult(requestCode: Int,
                                            permissions: Array<String>, grantResults: IntArray) {
        when (requestCode) {
            RECORD_REQUEST_CODE -> {

                if (grantResults.isEmpty() || grantResults[0] != PackageManager.PERMISSION_GRANTED) {

                } else {
                    recognizeVoice()
                }
            }
        }
    }

    override fun setChatBotMessageReview(messageResponse: ReviewMessageResponse) {
        size_notificacion = messageResponse.data.size
        Timber.i("tamaño de notificaciones: $size_notificacion ")
        Timber.i("json enviado: ${messageResponse.data} ")
        if (size_notificacion != 0) {
            for (i in 0 until size_notificacion) {

                if(messageResponse.data[i].notification_date != null){

                    fecha = getDate(messageResponse.data[i].notification_date).toString()
                }
                if (messageResponse.data[i].notification_option != null) {
                    notification_option = messageResponse.data[i].notification_option
                }
                if(messageResponse.data[i].notification_video != null){
                    notification_video = messageResponse.data[i].notification_video
                }
                if(messageResponse.data[i].notification_preview != null){
                    notification_preview = messageResponse.data[i].notification_preview
                }
                if(messageResponse?.data[i].notification_audio != null){
                    loadChatAdapter(messageResponse!!.data[i].name_program, NOTIFIER, height,
                            messageResponse!!.data[i].notification, fecha, notification_option,
messageResponse!!.data[i].messageId, notification_video, notification_preview,
messageResponse.data[i].notification_audio, "", "", "", "", "",null,"")
                }else
                loadChatAdapter(messageResponse!!.data[i].name_program, NOTIFIER, height,
                        messageResponse!!.data[i].notification, fecha, notification_option,
messageResponse!!.data[i].messageId, notification_video, notification_preview, "", "", "", "", "", "", null, "")
            }
            textOrSpeech = TEXT_ON
            activity_main_boton_hablar.isClickable = false
            recognizeVoice()
            hideAnimationListener()
            divider.visible()
            val animation_bot = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_right)
            recyclerView_sugerencias.visible()
            recyclerView_sugerencias.startAnimation(animation_bot)
        }
```

```
    }


    fun mostrarAyuda(){
        notification = true
        new_notification.gone()
        val intent = Intent(this, WelcomeSlidesActivity::class.java)
        intent.putExtra("activity", 1)
        startActivity(intent)
    }


    override fun showSugerencias(dialogFlowResponse: DialogFlowResponse, action: String) {

        if (dialogFlowResponse.sugerencias == null || dialogFlowResponse.sugerencias.isEmpty()) {
            Timber.i(" sugerencias null or empty")
            recyclerView_sugerencias.gone()
        }
        loadAdapterSug(dialogFlowResponse)
        Timber.i("El action es ${action}")
        actionMensaje = action
    }

    private fun build_adapter_sugerencias(dialogFlowResponse: DialogFlowResponse) {
        activity_main_boton_hablar.isClickable = true
        adapter_sugerencias = SugerenciasAdapter(dialogFlowResponse.sugerencias, height) {
            Timber.i("el mensaje seleccionado es $it")
            if (speechRecognizerViewModel.isListening) speechRecognizerViewModel.stopListening()
            sendMessage(it)
        }
        recyclerView_sugerencias.adapter = adapter_sugerencias
    }

    fun loadAdapterSug(dialogFlowResponse: DialogFlowResponse) {
        var size_sugerencias = dialogFlowResponse.sugerencias.size
        if (size_sugerencias != 0) {
            for (i in 0..size_sugerencias - 1) {
                Timber.i(" las sugerencias son: ${dialogFlowResponse.sugerencias[i].title}")
                build_adapter_sugerencias(dialogFlowResponse)

            }
        }
        if (dialogFlowResponse.sugerencias.isEmpty()) {
            recyclerView_sugerencias.gone()
            //activity_main_boton_neumo.isClickable = false
        }
    }

    override fun showUI(dialogFlowResponse: DialogFlowResponse) {
        eAerobic = 0
        eDisnea = 0
        activity_main_boton_hablar.isClickable = true
        if(dialogFlowResponse.variables_front.data.action == ACTION_INICIO2 && include_bot_init.visibility ==
View.INVISIBLE || include_bot_init.visibility == View.GONE){
            oneTime = 1
            if(textOrSpeech == SPEECH_ON){
                textOrSpeech = SPEECH_ON
            }else {
                textOrSpeech = TEXT_ON
                notification_speech = TEXT_ON
            }
        }
```

```
        if (include_bot_init.visibility == View.VISIBLE) {
            if (recyclerView_chatbot.visibility == View.VISIBLE) {
                val animation_bot = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_right)
                recyclerView_sugerencias.startAnimation(animation_bot)
                loadChatAdapter(dialogFlowResponse.response[0], RECEIVER, height, "", "", "", "", "", "", "", "",
multimedia!!, tipoMultimedia!!, "", "", dialogFlowResponse.variables_front.data.aerobico_ejer, null, "")
            }
            activity_main_tw_welcome.text = dialogFlowResponse.response[0]
            if(textOrSpeech == SPEECH_ON){
                textOrSpeech = SPEECH_ON
            }else{
                textOrSpeech =  TEXT_ON
                notification_speech = TEXT_ON
            }
            speechRecognizerViewModel.startListening()
            lottieAnimationView_micToListener.visible()
            lottieAnimationView_micToListener.setAnimation("Microfono_Propuesta_B_Waiting_Load_V2.json")
            lottieAnimationView_micToListener.setOnClickListener {
                speechRecognizerViewModel.stopListening()
                lottieAnimationView_micToListener.gone()
            }
            buttons_navigate.visible()
            buttons_teclado.gone()
            val animation = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_left)
            val animation_bot = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_right)
            divider.visible()
            if (width <= WIDTH_SCREEN || height <= HEIGHT_SCREEN) {
                activity_main_tw_welcome.setTextSize(TypedValue.COMPLEX_UNIT_PT, 9F)
                activity_main_tw_welcome.startAnimation(animation)
                lottieAnimationView.startAnimation(animation_bot)
                recyclerView_sugerencias.visible()
            } else {
                activity_main_tw_welcome.startAnimation(animation)
                lottieAnimationView.startAnimation(animation_bot)
                recyclerView_sugerencias.visible()
            }
        }
        else if(recyclerView_chatbot.visibility == View.VISIBLE){
            setChatBot(dialogFlowResponse)
        }

    }

    override fun setChatBot(dialogFlowResponse: DialogFlowResponse) {

        if(dialogFlowResponse.variables_front.data.multimedia != null &&
dialogFlowResponse.variables_front.data.tipoMultimedia != null ){
            multimedia = dialogFlowResponse.variables_front.data.multimedia
            tipoMultimedia = dialogFlowResponse.variables_front.data.tipoMultimedia
        }else{
            multimedia = ""
            tipoMultimedia = ""
        }
        if(dialogFlowResponse.sugerencias.isNullOrEmpty() == true){
            textOrSpeech = TEXT_ON
            notification_speech = TEXT_ON
        }

        Timber.i("tamaño de dialogflowresponse ${dialogFlowResponse.response.size}")

        if(mensajeSegmentado != ""){mensajeSegmentado = ""}
```

```
        if(!(dialogFlowResponse.response.size != 1 && !mTextToSpeech.equals(false))){
            mensajeSegmentado = mensajeSegmentado + dialogFlowResponse.response[0]
            loadChatAdapter(dialogFlowResponse.response[i], RECEIVER, height, "", "", "", "", "", "", "", "",
multimedia!!, tipoMultimedia!!, "", "", "", null, "")
            recyclerView_sugerencias.visible()
            val animation = AnimationUtils.loadAnimation(applicationContext, R.anim.slide_in_from_right)
            recyclerView_sugerencias.startAnimation(animation)
        }else{
            val rnds = (20 .. 100).random()
            val rnds_2 = (100..200).random()
            for (i in 0 .. dialogFlowResponse.response.size -1) runBlocking{
                if(i == 0){
                    mensajeSegmentado = mensajeSegmentado + dialogFlowResponse.response[i]
                    loadChatAdapter(dialogFlowResponse.response[i], RECEIVER, height, "", "", "", "", "", "", "", "",
multimedia!!, tipoMultimedia!!, "", "", "", null, "")
                    recyclerView_sugerencias.gone()
                }else if(i > 0 && i%2 == 0){
                    mensajeSegmentado = mensajeSegmentado + dialogFlowResponse.response[i]
                    var delayMessage = dialogFlowResponse.response[i].length * rnds_2
                    lottieAnimationView_waitingResponse.visible()
                    lottieAnimationView_waitingResponse.setAnimation("waitingResponse.json")
                    val handler = Handler()
                    handler.postDelayed({
                        loadChatAdapter(dialogFlowResponse.response[i], RECEIVER, height, "", "", "", "", "", "", "", "",
multimedia!!, tipoMultimedia!!, "", "", "", null, "")
                        lottieAnimationView_waitingResponse.gone()
                        if(i == dialogFlowResponse.response.size -1){
                            lottieAnimationView_waitingResponse.gone()
                            recyclerView_sugerencias.visible()
                        }
                        val animation = AnimationUtils.loadAnimation(applicationContext, R.anim.slide_in_from_right)
                        recyclerView_sugerencias.startAnimation(animation)
                    }, delayMessage.toLong())
                    hideAnimationListener()
                }else if(i > 0 && i%2 != 0) {
                    mensajeSegmentado = mensajeSegmentado + dialogFlowResponse.response[i]
                    var delayMessage = dialogFlowResponse.response[i].length * rnds
                    lottieAnimationView_waitingResponse.visible()
                    lottieAnimationView_waitingResponse.setAnimation("waitingResponse.json")
                    val handler = Handler()
                    handler.postDelayed({
                        loadChatAdapter(dialogFlowResponse.response[i], RECEIVER, height, "", "", "", "", "", "", "", "",
multimedia!!, tipoMultimedia!!, "", "", "", null, "")
                        if(i == dialogFlowResponse.response.size -1){
                            lottieAnimationView_waitingResponse.gone()
                            recyclerView_sugerencias.visible()
                        }
                        val animation = AnimationUtils.loadAnimation(applicationContext, R.anim.slide_in_from_right)
                        recyclerView_sugerencias.startAnimation(animation)
                    }, delayMessage.toLong())
                    hideAnimationListener()
                }
            }
        }
        Timber.i("Mensaje segmentado es: $mensajeSegmentado")
        textToSpeech(mensajeSegmentado)
    }

    override fun textToSpeech(text: String) {
        if (textOrSpeech == SPEECH_ON) {
            if (text == null) {
                Toast.makeText(this, "No hay respuesta de NeumoBot", Toast.LENGTH_SHORT).show()
```

```
            } else {
                text.tts(mTextToSpeech, this)
                mensajeSegmentado = ""
            }
        } else {
            stopSpeak()
        }
    }

    override fun onPause() {
        super.onPause()
        stopSpeak()
        hideSplash()
    }

    override fun hideSplash() {
        mSplashDialog.dismissDialog(this)
    }

    override fun showSplash() {
        mSplashDialog.showDialog(this)
    }

    fun stopSpeak() {
        Timber.i("stop speak")
        if (mTextToSpeech.isSpeaking) {
            mTextToSpeech.stop()
        }
    }

    fun hideKeyboard() {
        val inputMethodManager = getSystemService(Activity.INPUT_METHOD_SERVICE) as
InputMethodManager
        inputMethodManager.hideSoftInputFromWindow(activity_main_edit_text.windowToken, 0)
        recyclerView_sugerencias.visibility = View.VISIBLE
    }

    fun showKeyboard() {
        activity_main_edit_text.requestFocus()
        val inputMethodManager = getSystemService(Activity.INPUT_METHOD_SERVICE) as
InputMethodManager
        inputMethodManager.showSoftInput(activity_main_edit_text,
InputMethodManager.HIDE_IMPLICIT_ONLY)
        recyclerView_sugerencias.visibility = View.GONE
    }

    fun enableWrite() {
        buttons_navigate.invisible()
        buttons_teclado.visible()
        textOrSpeech = TEXT_ON
        notification_speech = TEXT_ON
        Timber.i("text or speech: $textOrSpeech")
        showKeyboard()
        if (recyclerView_chatbot.adapter != null) {
            recyclerView_chatbot.smoothScrollToPosition(recyclerView_chatbot.adapter.itemCount)
        }
    }

    private fun sendMessage(mensaje: String) {
        if (!mensaje.isNullOrEmpty() && !mensaje.isNullOrBlank()) {
            if (isOnlineNet(this)) {
                activity_main_edit_text.setText("")
```

```
            item_textView_speech.invisible()
            buttons_navigate.visible()
            buttons_teclado.gone()
            recyclerView_sugerencias.gone()
            hideKeyboard()
            hideSplash()

            if (recyclerView_chatbot.visibility == View.GONE || recyclerView_chatbot.visibility ==
View.INVISIBLE) {
                include_bot_init.invisible()
                recyclerView_chatbot.visible()
                recyclerView_sugerencias.gone()
                build_adapter()
            }

            if(actionMensaje == ACTION_MENSAJES){
                notifications_messages.forEach{
                    mensajes!!.remove(it)
                    Timber.i("Esto no debe entrar ahí")
                }

            }
            if(sendMessage == 0 || actionMensaje == ACTION_MENSAJES){
                sendMessage = 1
                if(eAerobic == 0){
                    Timber.i("el mensaje se va a enviar como sender $eDisnea y $eAerobic")
                    loadChatAdapter(mensaje, SENDER, height, "", "", "", "", "", "", "", "", "", "", "", "", "", null, "")
                    messageOnEnd(mensaje)
                    presenter.comunicacionBot(mensaje, mDataUser?.documento!!)
                }else if(eDisnea == 1 || eAerobic == 2 || eAerobic == 1){
                    presenter.comunicacionBot(mensaje, mDataUser?.documento!!)
                    recyclerView_sugerencias.gone()
                }
            }
        } else {
            showDialogError(resources.getString(R.string.error_internet))
        }
    }
  }

  override fun AerobicsSug(dialogFlowResponse: DialogFlowResponse, action: String){
    setChatBot(dialogFlowResponse)
    if(action == ACTION_EJERCICIO_AEROBICO || action == ACTION_EJERCICIO_AEROBICO_FB){
        if(dialogFlowResponse.variables_front.data.aerobicos == "no"){
            eAerobic = 0
        } else eAerobic = 1

        var size_sugerencias = dialogFlowResponse.sugerencias.size
        Timber.i("sugerenciasAerobicas $sugerenciasAerobicas")
        if (size_sugerencias != 0 && size_sugerencias > 2) {
//            itemRemove(AEROBICS, dialogFlowResponse)
            for (i in 0..size_sugerencias - 1) {
                Timber.i("las sugerencias aerobicas son: ${dialogFlowResponse.sugerencias[i].title}")
                if(dialogFlowResponse.sugerencias[i].title == "Montar bicicleta" && sugerenciasAerobicas == 0){
//                    itemRemove(AEROBICS, dialogFlowResponse).execute()
                    loadChatAdapter("0", RECEIVER, height, "", "", "", "", "", "", "", "", RESET, "", "", "", null, "")

                    val handler = Handler()
                    handler.postDelayed({
                        loadChatAdapter("Montar bicicleta", AEROBICS, height, "", "", "", "", "", "", "", "", "", AEROBICS,
dialogFlowResponse.sugerencias[i].iconoUrl, dialogFlowResponse.sugerencias[i].title,
dialogFlowResponse.variables_front.data.aerobico_ejer, null, "")
```

```
                    }, 10)
                }
            }
            sugerenciasAerobicas=sugerenciasAerobicas+1
        }
        recyclerView_sugerencias.gone()
        if(size_sugerencias <= 2){
            recyclerView_sugerencias.visible()
        }
    }
}

override fun eAerobicSelected (dialogFlowResponse: DialogFlowResponse){
    Timber.i ("aerobico_ejercicio es ${dialogFlowResponse.variables_front.data.aerobico_ejer}")
    if(textOrSpeech == SPEECH_ON || textOrSpeech == TEXT_ON && selectItemOnClick == 0){
        Timber.i("seleccionó itemAerobico diferente on click")
        loadChatAdapter("aerobicVoice", AEROBICS, height, "", "", "", "", "", "", "", "", "", AEROBICS,"", "",
dialogFlowResponse.variables_front.data.aerobico_ejer, null, "")
    }else if (selectItemOnClick == 1 ){
        Timber.i("seleccionó itemAerobico on click")
        val handler = Handler()
        handler.postDelayed({
            selectItemOnClick = 0
        }, 10)
    }
    setChatBot(dialogFlowResponse)
    showSugerencias(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
    recyclerView_sugerencias.visible()
    eAerobic = 0
}

override fun selectDyspneaMeter(dialogFlowResponse: DialogFlowResponse){
    eAerobic = 0
    eDisnea = 0
    if(dialogFlowResponse.variables_front.data.disnea.isNullOrEmpty() == false && textOrSpeech ==
SPEECH_ON || textOrSpeech ==  TEXT_ON && selectItemOnClick == 0){
        loadChatAdapter("1", RECEIVER, height, "", "", "", "", "", "", "", "", "", RESET, "", "", "", null, "")
        val handler = Handler()
        handler.postDelayed({
            loadChatAdapter("5", DISNEA, height, "", "", "", "", "", "", "", "", "", "disnea", "", "", "", null,
dialogFlowResponse.variables_front.data.disnea)
            setChatBot(dialogFlowResponse)
            showSugerencias(dialogFlowResponse, ACTION_DISNEA)
            recyclerView_sugerencias.visible()
        }, 10)

    }else if (selectItemOnClick == 1){
        setChatBot(dialogFlowResponse)
        showSugerencias(dialogFlowResponse, ACTION_DISNEA)
        recyclerView_sugerencias.visible()
        val handler = Handler()
        handler.postDelayed({
            selectItemOnClick = 0
        }, 10)
    }
}

override fun graphicRhbL(graphicResponse: GraphicResponse) {
//        loadChatAdapter("", RECEIVER, height, "", "", "", "", "", "", "", "", "", "", "", "", "", "", graphicResponse, "")
    recyclerView_sugerencias.visible()
}
```

```
    override fun dyspneaMeter(dialogFlowResponse: DialogFlowResponse, action: String) {
        if(dialogFlowResponse.sugerencias.size >= 4 ){
            Timber.i("Medidor de disnea ${dialogFlowResponse.sugerencias[0].title},
${dialogFlowResponse.sugerencias[1].title}")
            var disnea = ArrayList<Int>()
            eDisnea = 1
            eAerobic = 2
            disnea.add(dialogFlowResponse.sugerencias[0].title.toInt())
            disnea.add(dialogFlowResponse.sugerencias[1].title.toInt())
            disnea.add(dialogFlowResponse.sugerencias[2].title.toInt())
            disnea.add(dialogFlowResponse.sugerencias[3].title.toInt())
            disnea.add(dialogFlowResponse.sugerencias[4].title.toInt())
            for(i in 0 .. 4){
                if(disnea[i] == 0 && sugerenciasDisnea == 0){
                    Timber.i("SugerenciasDisnea: $sugerenciasDisnea, SugerenciasAerobicos:
$sugerenciasAerobicas")
                    loadChatAdapter("1", RECEIVER, height, "", "", "", "", "", "", "", "", RESET, "", "", "", null, "")
//                    itemRemove(DISNEA, dialogFlowResponse).execute()
                    val handler = Handler()
                    handler.postDelayed({
                        loadChatAdapter("0", DISNEA, height, "", "", "", "", "", "", "", "", "disnea", "", "", "", null, "")
                    }, 10)
                }
            }
            sugerenciasDisnea = sugerenciasDisnea +1
        } else {Timber.i("No es medidor de disnea")}
    }

    override fun resetAerobics(dialogFlowResponse: DialogFlowResponse) {
        Timber.i("RESETAEROBICS $sugerenciasAerobicas")
        if(dialogFlowResponse.variables_front.data.action == ACTION_EJERCICIO_AEROBICO){
            if(sugerenciasAerobicas != 0) sugerenciasAerobicas = 0
            AerobicsSug(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
        }else if(dialogFlowResponse.variables_front.data.action == ACTION_EJERCICIO_AEROBICO_FB){
            sugerenciasAerobicas = 0
            AerobicsSug(dialogFlowResponse, ACTION_EJERCICIO_AEROBICO)
        }
    }
    override fun resetDisnea(dialogFlowResponse: DialogFlowResponse) {
        Timber.i("RESETDISNEA $sugerenciasDisnea")
        if(dialogFlowResponse.variables_front.data.action == ACTION_END_RHB ||
dialogFlowResponse.variables_front.data.action == ACTION_EJ_PIERNAS ||
dialogFlowResponse.variables_front.data.action == ACTION_DISNEA_FB){
            if(sugerenciasDisnea != 0)sugerenciasDisnea = 0
            dyspneaMeter(dialogFlowResponse, ACTION_DISNEA)
        }
    }



    private fun build_adapter() {
        recyclerView_sugerencias.gone()
        adapter = MessageAdapter(mensajes!!, object: MessageAdapter.ListenerInterface{
            override fun deleteResponse(position: Int){
                Timber.i("posicion a eliminar en voice = $position")
                val handler = Handler()
                handler.postDelayed({
                    mensajes!!.removeAt(position)
                    adapter!!.notifyItemRemoved(position)
                }, 10)
            }
```

```
        override fun responseDyspnea(i: Int, datosChat0: DatosChat?, datosChat1: DatosChat?, datosChat2:
DatosChat?, datosChat3: DatosChat?) {
            if(i != 6 ){
                presenter.comunicacionBot(i.toString(), mDataUser?.documento!!)
                selectItemOnClick = 1
            }
        }

        override fun aerobicExercise(sender: String, datosChat1: DatosChat?, datosChat2: DatosChat?) =
runBlocking {
            if (sender != ""){
                presenter.comunicacionBot(sender, mDataUser?.documento!!)
                selectItemOnClick = 1
            }
        }
        override fun playVideo(notification_video: String) {
            goToVideoActivity(notification_video)
        }

        override fun stopAudio(itemView: View) {
            stopAudioNotification(itemView)
        }

        override fun playAudio(urlAudio: String, itemView: View, estado: Int) {
            estado_audio = estado
            playAudioNotification(urlAudio, itemView)
        }

        override fun readNotification(datosChat: DatosChat, mensaje: String, position: Int, messageId: String,
nameProgram: String, urlAudio: String) {
            noReadNotification(datosChat, mensaje, position, messageId, nameProgram, urlAudio)
        }

        override fun viewImage(urlImage: String){
            goToImageActivity(urlImage)
        }

        override fun playGif(urlGif: String) {
            goToGifActivity(urlGif)
        }

        override fun hideSugerencias() {
            hideSugerenciasAhora()
        }


    })
    recyclerView_chatbot.adapter = adapter
}



fun hideSugerenciasAhora() {
    if(actionMensaje == ACTION_SEE_NOTIFICATIONS){
        recyclerView_sugerencias.gone()
    }else if(actionMensaje == ACTION_MENSAJES){

    }
}
```

```
    private fun noReadNotification(datosChat: DatosChat, mensaje: String, position: Int, messageId: String,
nameProgram: String, urlAudio: String) {
        Timber.i("la posicion es: $position y messageID: $messageId")
        Timber.i("textorspeech es: $textOrSpeech y notification speech es: $notification_speech")
        removeAction = removeAction + 1
        adapter!!.notifyItemRemoved(position)
        mensajes!!.removeAt(position)
        mensajes!!.remove(datosChat)
        presenter.markNotificationAsRead(messageId,mDataUser?.documento!!)
        if (notification_speech == TEXT_ON) {
            textOrSpeech = TEXT_ON
        }

        if(notification_speech == SPEECH_ON){
            textOrSpeech = SPEECH_ON
//            textToSpeech("Mensaje enviado por " + nameProgram +", dice: "+ "..." + mensaje)
            loadChatAdapter("Mensaje enviado por " + nameProgram, RECEIVER, height, "", "", "", "", "", "", "", "",
"", "", "", "", null, "")
            loadChatAdapter(mensaje, NOTIFIER_BODY, height, datosChat.notificacion, "",
datosChat.notification_option, "", datosChat.notification_video, datosChat.notification_preview, urlAudio, "", "",
"", "", "", null, "")

        }else{
            loadChatAdapter("Mensaje enviado por " + nameProgram, RECEIVER, height, "", "", "", "", "", "", "", "",
"", "", "", "", null, "")
            loadChatAdapter(mensaje, NOTIFIER_BODY, height, datosChat.notificacion, "",
datosChat.notification_option, "", datosChat.notification_video, datosChat.notification_preview, urlAudio, "", "",
"", "", "", null, "")

            textOrSpeech = TEXT_ON
        }
        textOrSpeech = TEXT_ON
        Timber.i("Mira este dato ${size_notificacion}")
        if (removeAction == size_notificacion) {
//            loadChatAdapter("Has leido todos tus mensajes, ¿Que mas quieres hacer?" , RECEIVER, height, "",
"", "", "")
            val animation_bot = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_right)
            recyclerView_sugerencias.visible()
            recyclerView_sugerencias.startAnimation(animation_bot)
            activity_main_boton_hablar.isClickable = true
            if (notification_speech == TEXT_ON) {
                textOrSpeech = TEXT_ON
            } else {
                textOrSpeech = SPEECH_ON
            }
        } else {
            val animation_bot = AnimationUtils.loadAnimation(this, R.anim.slide_in_from_right)
            recyclerView_sugerencias.visible()
            recyclerView_sugerencias.startAnimation(animation_bot)
        }

    }


    override fun updateNotificationAdapter() {
       // adapter!!.notifyItemRemoved()

    }


    fun goToVideoActivity(urlVideo: String){
```

```
        val intent = Intent(this, VideoPlayActivity::class.java)
        intent.putExtra("video", urlVideo)
        startActivity(intent)
    }

    fun goToImageActivity(urlImage: String){
        val intent = Intent(this, ImageViewActivity::class.java)
        intent.putExtra("image", urlImage)
        startActivity(intent)
    }

    fun goToGifActivity(urlGif: String){
        val intent = Intent(this, GifPlayActivity::class.java)
        intent.putExtra("gif", urlGif)
        startActivity(intent)
    }


    fun playAudioNotification(urlAudio: String, itemView: View)=with (itemView){
        if(estado_audio == 0){
            estado_vista = itemView.audio_notifier_play
            solo_vista = audio_notifier_play
        }else if(estado_audio == 1){
            estado_vista = itemView.audio_notifier_play_message
            solo_vista = audio_notifier_play_message
        }

        if (pause) {
            mediaPlayer.seekTo(mediaPlayer.currentPosition)
            mediaPlayer.start()
            pause = false
        } else {
            try {
                mediaPlayer.setDataSource(urlAudio)
                mediaPlayer.prepareAsync ()
//                mediaPlayer.start()
            } catch (e: IOException) {
            }
            dialog = ProgressDialog(context)
            dialog.setMessage("Cargando audio, por favor espere...")
            dialog.setCancelable(false)
            dialog1 = ProgressDialog(context)
            dialog1.setCancelable(false)
            dialog.show()
            Timber.i("nowPlaying $urlAudio")
            mediaPlayer?.setOnPreparedListener(object : MediaPlayer.OnPreparedListener {
                override fun onPrepared(mp: MediaPlayer?) {
                    Timber.i("duration ${mediaPlayer.duration}")
                    initializerSeekBar(estado_vista)
                    mediaPlayer.start()
                    dialog.dismiss()
//                    dialog1.dismiss()
                    if(mediaPlayer.isPlaying){
                        mediaPlayer.setOnBufferingUpdateListener { mp, percent ->
                            solo_vista.progressBar.setProgress(percent)
//                            if(percent <= 80){
//                                dialog1.setMessage("Cargando audio: $percent %")
//                                dialog1.show()
//                            }else dialog1.dismiss()
                        }
                    }
                }
```

```
            })

            mediaPlayer.setOnCompletionListener {
//                dialog.dismiss()
                pause = false
                solo_vista.seek_bar_notification.setProgress(0)
                solo_vista.progressBar.setProgress(0)
                mediaPlayer.reset()
               // handler.removeCallbacks(runnable)

                solo_vista.reproducir_notificacion_audio.visible()
                solo_vista.pausar_notificacion_audio.invisible()

            }
        }
    }

    private fun initializerSeekBar(itemView: View)= with(itemView) {

        estado_vista.seek_bar_notification.max = mediaPlayer.duration
        Timber.i("maximo  ${mediaPlayer.duration}")
        runnable = Runnable {
            estado_vista.seek_bar_notification.setProgress(mediaPlayer.currentPosition)
            estado_vista.seek_bar_notification.setOnSeekBarChangeListener(object :
SeekBar.OnSeekBarChangeListener {
                override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser: Boolean) {
                    if (fromUser) {
                        mediaPlayer.seekTo(progress)
                    }
                    estado_vista.seek_bar_notification.post(object : java.lang.Runnable {
                        override fun run() {
                            updateTime(progress.toFloat(), estado_vista)
                        }
                    })
                }

                override fun onStartTrackingTouch(seekBar: SeekBar?) {
                    Timber.i("onStartTrackingTouch")
                }

                override fun onStopTrackingTouch(seekBar: SeekBar?) {
                    Timber.i("onStopTrackingTouch")

                }
            })
            val getHandler : Handler = Handler()
            getHandler!!.postDelayed(runnable!!,500)

//            handler!!.postDelayed(runnable!!,500)
        }
        val getHandler : Handler = Handler()
        getHandler!!.postDelayed(runnable!!,1000)
//        handler1.postDelayed(runnable, 1000)
    }

    fun updateTime(progress: Float, itemView: View)= with(itemView) {
        Timber.i("Progress es ${progress/1000}")
        tiempo = getTime(progress).toString()
        Timber.i("El tiempo es: $tiempo")
        estado_vista.tv_due.text = tiempo

    }
```

```
    fun stopAudioNotification(itemView: View)= with (itemView) {
        if(mediaPlayer.isPlaying){
            mediaPlayer.pause()
            pause = true
        }

    }


    fun stopAudioOnFunction(itemView: View) = with(itemView){
        if(estado_audio == 0){
            estado_vista = itemView.audio_notifier_play
        }else if(estado_audio == 1){
            estado_vista = itemView.audio_notifier_play_message
        }
        pause = false
        estado_vista.seek_bar_notification.setProgress(0)
        estado_vista.progressBar.setProgress(0)
        mediaPlayer.reset()
    }


    private fun loadChatAdapter(mensaje: String, type: String, screenSize: Int, notification: String, fecha: String,
video: String, id: String, notification_video: String, notification_preview: String, audio: String,
multimedia_adapter: String, tipoMultimedia_adapter: String, iconoUrl: String, tittle: String, aerobico_ejer: String,
graphicResponse: GraphicResponse?, dyspneaSelection: String) {
        item_textView_speech.text = " "
        recyclerView_sugerencias.gone()
        mensajes!!.add(DatosChat(mensaje, type, screenSize, notification, fecha, video, id, notification_video,
notification_preview, audio, multimedia_adapter, tipoMultimedia_adapter, iconoUrl, tittle, aerobico_ejer,
graphicResponse, dyspneaSelection))
        adapter!!.notifyDataSetChanged()
        recyclerView_chatbot.smoothScrollToPosition(recyclerView_chatbot.adapter.itemCount)
        Timber.i("textOrSpeech: $textOrSpeech")
        messageOnEnd(mensaje)
        multimediaEnd(multimedia_adapter, tipoMultimedia_adapter)
        if(actionMensaje == ACTION_MENSAJES){
            notifications_messages.add(DatosChat(mensaje, type, screenSize, notification, fecha, video, id,
notification_video, notification_preview, audio, multimedia!!, tipoMultimedia!!, iconoUrl, tittle, aerobico_ejer,
graphicResponse, dyspneaSelection))
        }
        if(actionMensaje == ACTION_EJERCICIO_AEROBICO || actionMensaje ==
ACTION_EJERCICIO_AEROBICO_FB){
            ejercicio_card.add(DatosChat(mensaje, type, screenSize, notification, fecha, video, id,
notification_video, notification_preview, audio, multimedia!!, tipoMultimedia!!, iconoUrl, tittle, aerobico_ejer,
graphicResponse, dyspneaSelection))
        }

        if(actionMensaje == ACTION_DISNEA || actionMensaje == ACTION_DISNEA_FB){
            disnea_card.add(DatosChat(mensaje, type, screenSize, notification, fecha, video, id, notification_video,
notification_preview, audio, multimedia!!, tipoMultimedia!!, iconoUrl, tittle, aerobico_ejer, graphicResponse,
dyspneaSelection))
        }
        if(type == RECEIVER && speechRecognizerViewModel.isListening==false){sendMessage = 0}
    }

    private fun multimediaEnd(multimedia_adapter: String, tipoMultimedia_adapter: String) {
        if(multimedia != null && tipoMultimedia != null){
            multimedia = multimedia_adapter
            tipoMultimedia = tipoMultimedia_adapter
        }else {
```

```
            multimedia = ""
            tipoMultimedia = ""

        }

    }

    private fun messageOnEnd(mensaje: String) {
        if(mensaje == TRIGGER_ADIOS || mensaje == TRIGGER_ADIOS1 || mensaje == TRIGGER_ADIOS2 ||
mensaje == TRIGGER_ADIOS3){
            if(textOrSpeech == SPEECH_ON){
                textOrSpeech = TEXT_ON
                notification_speech = TEXT_ON
            }
            recyclerView_sugerencias.gone()
        }
    }

    override fun attachBaseContext(newBase: Context) {
        super.attachBaseContext(CalligraphyContextWrapper.wrap(newBase))
    }

    override fun dispatchTouchEvent(ev: MotionEvent): Boolean {
        super.dispatchTouchEvent(ev)
            stopSpeak()

        return false
    }

    override fun onBackPressed() {

    }
    internal inner class itemRemove(type: String, dialogFlowResponse: DialogFlowResponse) :
AsyncTask<Void, Void, String>() {

        val type = type
        val dialogFlowResponse = dialogFlowResponse
        var resetAero = 0
        var resetDis = 0


        override fun onPreExecute(){
            super.onPreExecute()
            Timber.i("asyncTask remove items $sugerenciasAerobicas sugerenciasDisnea: $sugerenciasDisnea
type : $type action: ${dialogFlowResponse.variables_front.data.action}")

            if(type == AEROBICS && sugerenciasAerobicas == 0 && resetAero == 0){
                Timber.i("resetAero es: $resetAero")
                loadChatAdapter("0", RECEIVER, height, "", "", "", "", "", "", "", "", RESET, "", "", "", null, "")
                resetAero = resetAero +1
            }
            if(type == DISNEA && sugerenciasDisnea == 0 && resetDis == 0){
//                loadChatAdapter("1", RECEIVER, height, "", "", "", "", "", "", "", "", RESET, "", "", "", null, "")
                resetDis = resetDis +1
            }
    }
        override fun doInBackground(vararg params: Void?): String?{
            Timber.i("params removeItem $params")
            return null
        }
        override fun onPostExecute(result: String?){
            super.onPostExecute(result)
```

```
            Timber.i("mensaje a setChatBot $result")

            if(type == DISNEA && sugerenciasDisnea == 0){
                sugerenciasDisnea = sugerenciasDisnea +1
//                loadChatAdapter("0", DISNEA, height, "", "", "", "", "", "", "", "", "", "disnea", "", "", "", null, "")
            }
            if(type == AEROBICS && sugerenciasAerobicas == 0){
                sugerenciasDisnea = sugerenciasDisnea +1
                loadChatAdapter("Montar bicicleta", AEROBICS, height, "", "", "", "", "", "", "", "", "", AEROBICS,
dialogFlowResponse.sugerencias[i].iconoUrl, dialogFlowResponse.sugerencias[i].title,
dialogFlowResponse.variables_front.data.aerobico_ejer, null, "")
            }

        }

    }
}
```

- **Útil:** Paquete transversal a toda la aplicación, en este paquete se incluyen las utilidades, constantes, funciones de extensión, entre otras.

```
    /**
     * View visible
     */
    fun View.visible() {
        visibility = View.VISIBLE
    }

    /**
     * View gone
     */
    fun View.gone() {
        visibility = View.GONE
    }

    /**
     * View invisible
     */
    fun View.invisible() {
        visibility = View.INVISIBLE
    }

    /**
     * Translates String into MD5 hex
     */
    fun String.md5(): String {
        val md = MessageDigest.getInstance("MD5")
        val digested = md.digest(toByteArray())
        return digested.joinToString("") {
            String.format("%02x", it)
        }
    }

    /**
     * Text changed listener as lambda expression
     */
    fun EditText.afterTextChanged(afterTextChanged: (String) -> Unit) {
        this.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
```

```
        }

        override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
        }

        override fun afterTextChanged(editable: Editable?) {
            afterTextChanged.invoke(editable.toString())
        }
    })
}

/**
 * Displays a Toast
 */
fun Any.showToast(context: Context) {
    Toast.makeText(context, this.toString(), Toast.LENGTH_LONG).show()
}

/**
 * Displays a Dialog
 */
fun Dialog.showDialog(activity: Activity) {
    if (!activity.isFinishing && !this.isShowing) {
        this.show()
        this.setCanceledOnTouchOutside(false)
    }
}

/**
 * Dissmises a Dialog
 */
fun Dialog.dismissDialog(activity: Activity) {
    if (!activity.isFinishing && this.isShowing) {
        this.dismiss()
    }
}

fun EditText.hashBCrypt(): String{
    val hashString= BCrypt.hashpw(this.text.toString(), SALT)
    return hashString.substring(SALT.length, hashString.length)
}

fun configureTTS(context: Context): TextToSpeech{
    var textToSpeech: TextToSpeech? = null
     textToSpeech = TextToSpeech(context.applicationContext, TextToSpeech.OnInitListener { status ->
         if (status != TextToSpeech.ERROR) {
            textToSpeech!!.language = Locale.getDefault()
            textToSpeech!!.setSpeechRate(1.8f)
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
                textToSpeech!!.voice = Voice("com.google.android.tts", Locale("es", "CO", ""), 400, 200, true, null)
            }
        } //else {
          //  conversionCallaBack.onErrorOccurred("Failed to initialize TTS engine")
        // }
     })

    return textToSpeech
}

fun String.tts(textToSpeech: TextToSpeech, context: Context){
```

```
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            textToSpeech!!.speak(this, TextToSpeech.QUEUE_FLUSH, null, "MessageId")
        } else {
            val hash = HashMap<String, String>()
            hash[TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID] = "MessageId"
            hash[TextToSpeech.Engine.KEY_PARAM_STREAM] =
    AudioManager.STREAM_NOTIFICATION.toString()
            textToSpeech!!.speak(this, TextToSpeech.QUEUE_FLUSH, hash)
        }
    }

    /**
     * Comprueba si hay conexión a internet
     *
     * @return
     */
    fun isOnlineNet(context: Context): Boolean {
        val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE)
        return if (connectivityManager is ConnectivityManager) {
            val networkInfo: NetworkInfo? = connectivityManager.activeNetworkInfo
            networkInfo?.isConnected ?: false
        } else false
    }


    fun getDate(milliSeconds: Float): String? {
        val formatter =  SimpleDateFormat("d 'de' MMMM 'de' yyyy")
        val calendar = Calendar.getInstance()
        calendar.timeInMillis = milliSeconds.toLong()
        return formatter.format(calendar.time)
    }

    /**
     * Muestra el tiempo que pasa en minutos y seg
     *
     * @return
     * */

    fun getTime(milliSeconds: Float): String?{
        val formatter =  SimpleDateFormat("mm:ss")
        val calendar = Calendar.getInstance()
        calendar.timeInMillis = milliSeconds.toLong()
        return formatter.format(calendar.time)
    }
```

- **Services:** Paquete que contiene las clases de los servicios a los que subscribe la aplicación como FirebaseMessaging, NetworkingReceiver y los que aplique.

```
lass NeumoFirebaseMessagingService: FirebaseMessagingService() {

    override fun onMessageReceived(remoteMessage: RemoteMessage) {
        // ...

        // TODO(developer): Handle FCM messages here.
        // Not getting messages here? See why this may be: https://goo.gl/39bRNJ

        // Check if message contains a data payload.
        if (remoteMessage.data.isNotEmpty()) {
```

```
            Timber.i("Message data payload: " + remoteMessage.data)

            if (/* Check if data needs to be processed by long running job */ true) {
                // For long-running tasks (10 seconds or more) use Firebase Job Dispatcher.
            //   scheduleJob()
            } else {
                // Handle message within 10 seconds
             //   handleNow()
            }

            sendNotification(remoteMessage)

        }

        // Check if message contains a notification payload.
        if (remoteMessage.notification != null) {
            Timber.i( "Message Notification Body: " + remoteMessage.notification!!.body!!)
        }

        // Also if you intend on generating your own notifications as a result of a received FCM
        // message, here is where that should be initiated. See sendNotification method below.
    }


    private fun sendNotification(remoteMessage: RemoteMessage){
        var m: Int = ((Date().time / 1000L) % Integer.MAX_VALUE).toInt()
        val intent = Intent(this, LoginActivity::class.java)
        intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP
        val uniqueInt = (System.currentTimeMillis() and 0xffffff).toInt()
        val pendingIntent = PendingIntent.getActivity(this, uniqueInt, intent,
                PendingIntent.FLAG_UPDATE_CURRENT)

        val channelId = getString(R.string.channel_id)
        val notificationBuilder = NotificationCompat.Builder(this, channelId)
                .setSmallIcon(R.drawable.ic_logo_neumomed)
                .setContentTitle(remoteMessage.notification?.title)
                .setContentText(remoteMessage.notification?.body)
                .setAutoCancel(true)
                .setContentIntent(pendingIntent)

        val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        // Since android Oreo notification channel is needed.
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(channelId,
                    getString(R.string.channel_id),
                    NotificationManager.IMPORTANCE_DEFAULT)
            notificationManager.createNotificationChannel(channel)
        }

        notificationManager.notify(m, notificationBuilder.build())
    }


}
```

Esta forma de estructuración del proyecto nos ayuda a que el código desarrollado sea fácil de mantener, reusar y extender en el tiempo. Al tener

una arquitectura planificada se muestra los resultados del desarrollo de los requerimientos.

Para el desarrollo del código, fue seleccionado el lenguaje de programación Kotlin para desarrollar cada una de las partes definidas en los diagramas realizados en la etapa de diseño.

se propuso modularizar en diferentes componentes. Los componentes son módulos de aplicación que cumplen una función específica, estos vienen de dos tipos:

- **Componentes de interfaz:** responsables de exponer el tipo específico de funcionalidad.

- **Componentes de implementación:** implementación de la funcionalidad específica expuesta por los componentes de una interfaz.

Cuando una aplicación desea interactuar con un componente específico, la inyección se realiza a través de la interfaz y el cableado de la implementación debe lograrse a través de una inyección de dependencia.

La descripción de cada componente, de acuerdo al patrón de arquitectura se realiza en la siguiente tabla:

*Tabla 3. Capas patrón Modelo-Vista-presentador*

| CAPA | ROLES Y RESPONSABILIDADES |
|---|---|
| Capa UI | Responsable de la presentación de la interfaz y manejo de las acciones del usuario. La arquitectura de UI se basa en el modelo Modelo-Vista-Presentador. Esta capa está compuesta por las interfaces de vista que serían implementadas por una Vista, Fragmento o Actividad de Android. El presentador es responsable de establecer una comunicación entre la vista y la capa de lógica de negocio. Es aquí donde los datos se transforman (modelos) de un lado a otro en algo que las reglas de negocio pueden |

| | |
|---|---|
| | comprender en función del flujo de eventos.<br>Esta capa también es responsable de recopilar las entradas del usuario, las validaciones de entrada, la gestión de recursos de la aplicación, la gestión del TTS y STT para la entrada por voz y salida de audio. |
| Capa de negocio | Una capa con implementación pura de Kotlin, sin dependencias, capa con todas las reglas de negocios. Los casos de uso se ejecutan y comunican los eventos a la capa superior (UI / presentador) utilizando devoluciones de llamada. El propósito de los interactores o casos de uso es resolver problemas específicos. Esto se ajusta al principio de responsabilidad única. La capa de dominio también puede tener sus propios modelos de datos que pueden ser manipulados por la lógica de negocios. |
| Capa de servicio | Todos los datos que necesita la aplicación provienen de esta capa. Es aquí donde se consultan los datos a la fuente de datos. La capa de datos contendrá sus propias entidades que se transformarían en modelos cuando se devuelvan. |

Básicamente, los principios SOLID ayudan a conseguir escalabilidad y evitar que el código se rompa cada vez que aparece un cambio.

### 6.4 Pruebas de funcionamiento

El objetivo de esta fase es verificar el funcionamiento de la aplicación en diferentes escenarios y condiciones; para esto se realizan las tareas de prueba en emulador y dispositivos reales.

En la prueba en emulador, se realizaron pruebas simulando el escenario y emulando el dispositivo móvil, explorando todas las utilidades y funciones de la aplicación, introduciendo diferentes datos, inclusive erróneos, para medir la funcionalidad y el nivel de robustez del software. Si se encuentran algunas fallas, se debe regresar a la etapa de codificación en la fase de desarrollo para solucionar los problemas, si las pruebas son satisfactorias se procede a la etapa de pruebas con dispositivos reales. Deben hacerse pruebas de campo en equipos reales para medir el desempeño y el rendimiento del aplicativo. Si se encuentran fallas en el tiempo de ejecución, si el software no cumple con los requerimientos especificados, o si el cliente solicita un cambio de última hora, hay que regresar a la fase de diseño para reestructurar y solucionar el inconveniente presentado.

### 6.5 Despliegue

Terminada la depuración de la aplicación y atendidos todos los requerimientos de última hora del cliente se da por finalizada la aplicación y se procede a generar un APK para el despliegue en la PlayStore.

Al implementar la metodología en Doctor Neumo se busca hacer un nuevo versionamiento en la PlayStore. Se pretende con herramientas de Google obtener información que nos de cuenta de las mejoras de usabilidad.

A continuación se muestra cada una de las partes definidas en los diagramas realizados en la etapa de diseño después de ser desplegada en producción.

- **Diapositivas de Introducción**

Las diapositivas de introduccion cuenta con la información necesaria para que el usuario tenga un conocimiento preliminar acerca de lo que puede hacer dentro de la app y como se da la comunicación con el bot. Estas diapositivas componen imágenes que se deslizan en forma de carrusel y anteceden al login para iniciar la comunicación con el asistente.

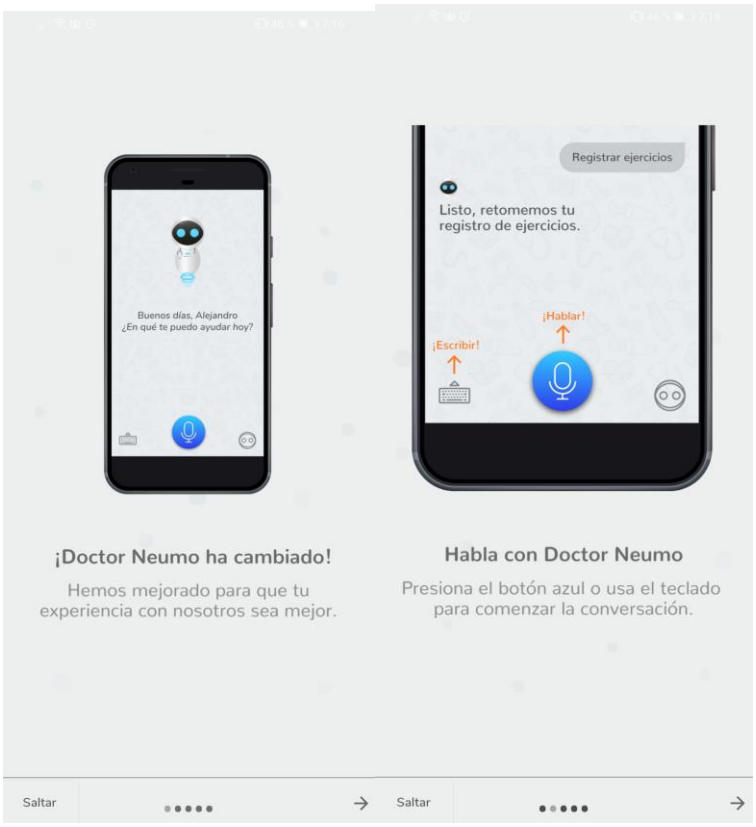


*Figura 18. Diapositivas de introducción*

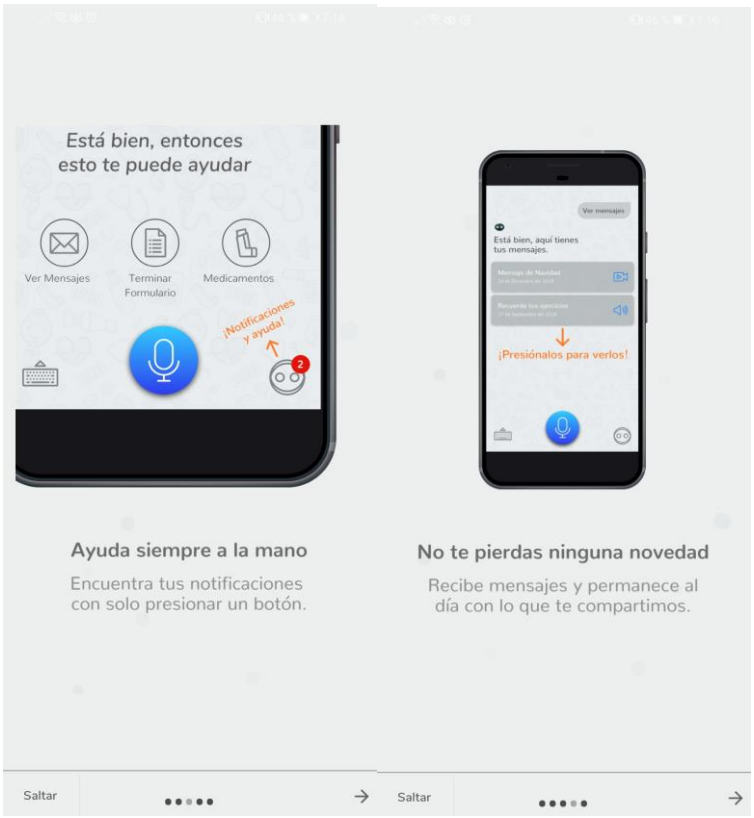


*Figura 19. Diapositivas de Introducción*

- **Inicio de Sesión**

El inicio de sesión consta de poner dos datos importantes que están registrados en la base de datos una vez el paciente hace su cita de ingreso al programa de rehabilitación pulmonar. Los datos necesarios son la cédula del paciente y su año de nacimiento. El año de nacimiento se toma como clave de acceso a la aplicación, esta viaja de forma encriptada y se valida, si esta es correcta se inicia la comunicación con el asistente, si es incorrecta la información suministrada, la aplicación recibirá como respuesta un error y esta será mostrada al usuario en una alerta. La alerta depende de la respuesta a la petición de login.


*Figura 20. Login aplicación Doctor Neumo*

- **Asistente Virtual, sugerencias y notificaciones.**

En la comunicación con el asistente virtual se da unas sugerencias en la parte inferior a cerca de las acciones que puede hacer el asistente. Las sugerencias son registrar ejercicios, llenar formularios, ver mensajes y preguntas frecuentes. En el proyecto se introduce la idea de desarrollar conceptos de gamificación que son válidos en el momento en que el usuario hace el registro de ejercicios.

La rehabilitación pulmonar es compleja. Implica una interacción siempre cambiante del paciente de rehabilitación con diferentes entornos clínicos y la IPS. La gamificación para la rehabilitación va más allá de simplemente crear una aplicación o entorno "divertido" y "emocionante" en el que se debe lograr que el paciente complete ejercicios e intervenciones de rehabilitación. La introducción de métodos de gamificación en Doctor Neumo permitió un mejor "ajuste" con cada paciente, proporcionó datos apropiados a terapeutas y médicos, y eventualmente conducirá a juegos efectivos para la rehabilitación.

Dentro del registro de ejercicios se incorporó un selector de ejercicios sugeridos en la consulta de rehabilitación dentro de la IPS. Entre los ejercicios sugeridos se encuentra subir escaleras, caminar, montar bicicleta, bailar y nadar. En el funcionamiento natural de la aplicación y la comunicación con el asistente se deja abierta la estructura para implementar conceptos de gamificación propios para la parte de ejercicios aerobios. Este medidor responde a las sugerencias y a las necesidades tanto del usuario interno como el usuario externo. Se implementan animaciones y transiciones dentro de la IU para mejorar la experiencia del usuario externo al registrar los ejercicios. Cabe resaltar que esta información queda consolidada en bases de datos y se puede disponer de ella para futuras implementaciones y registros gráficos y estadísticos para la observación del avance del paciente en su rehabilitación.

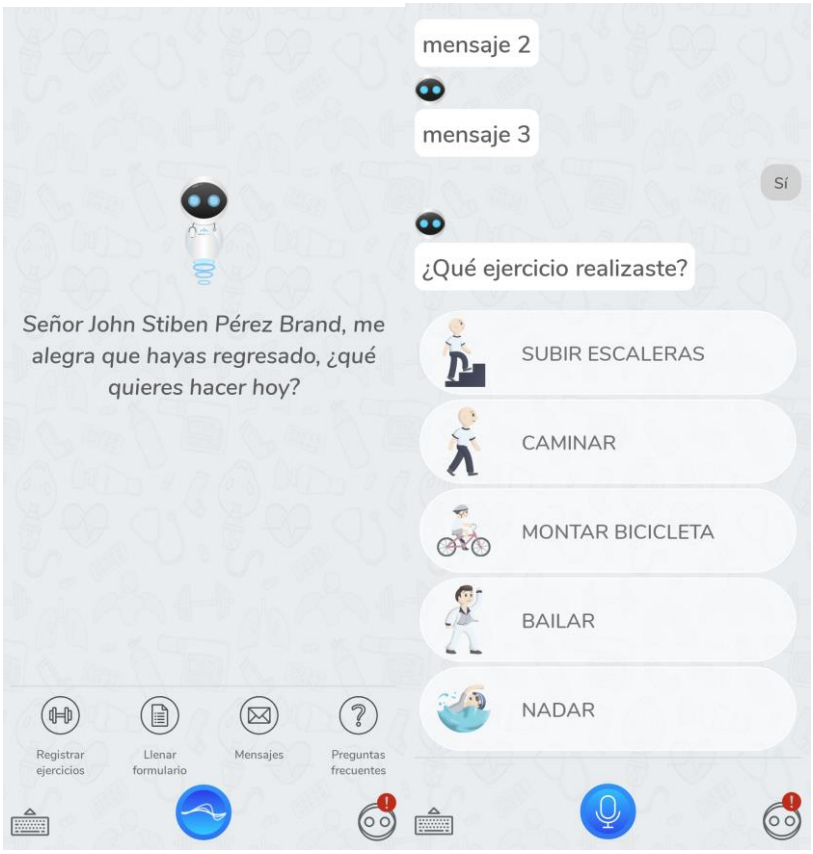


*Figura 21. Asistente Virtual y sugerencias ejercicios Aeróbicos.*

En el registro de ejercicios se implementó un medidor de disnea. Es importante este medidor ya que captura la información de nivel de disnea que tiene el paciente a la hora de finalizar los ejercicios. Este es uno de los parámetros medidos a la hora de hacer un ejercicio de rehabilitación y su escala es de 1 a 5. Es pertinente recopilar esta información porque se observa el registro del paciente en su rehabilitación y el cambio de su

patología en el histórico. Asimismo, estas ayudas permiten que el usuario esté en constante monitoreo, pues el usuario interno de la aplicación tiene acceso a los datos del registro del medidor de disnea y así el personal obtiene información pertinente del estado del paciente en la distancia. El mensaje que reciben de Doctor Neumo como respuesta depende del nivel de disnea que se seleccione, los mensajes en su mayoría son alentadores. Se implementa métodos de gamificación en este medidor, pues la medición se hace con variables más amigables e intuitivas, la pregunta ¿Que tan cansado te sientes? Trae como sugerencias de respuesta la escala de 1 a 5, pero en vez de mostrar la escala numérica se muestra sugerencias como Poco que hace referencia a la escala (1), Moderado (2), Bastante (3), Agotado (4) y Exhausto (5). Durante el registro de ejercicios están presentes animaciones y transiciones que permiten dinamizar y mejorar la experiencia del usuario externo con la aplicación.


*Figura 22. Medidor de disnea y registro de ejercicios*

La información a través del tiempo se muestra en las siguientes gráficas las cuales muestran el cambio del versionamiento y una curva positiva que indica la usabilidad de la aplicación en los pacientes. Se muestra que la versión 2.4 de la aplicación es una versión estable con un mínimo de fallos. Estas estadísticas fueron extraídas de la consola de google play el cual monitorea el funcionamiento de la aplicación. Actualmente la aplicación tiene 264 sesiones iniciadas y a partir del 5 de julio de 2019 y con esto un aumento en el registro de ejercicios de los pacientes.

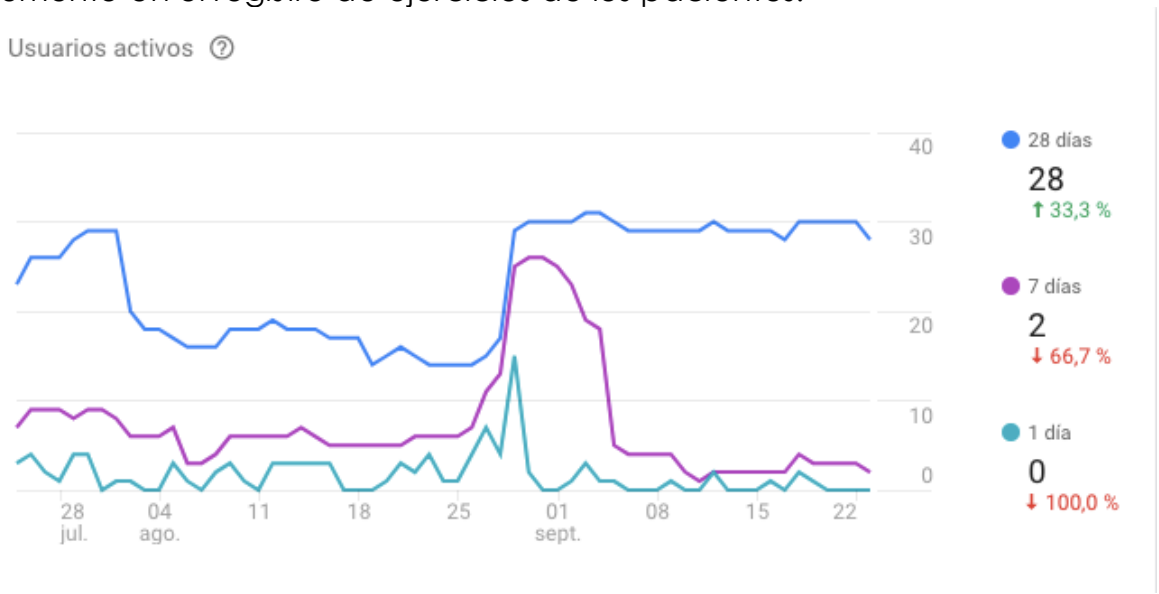


*Figura 23. Gráfica de uso de la aplicación después del lanzamiento de la versión 2.4*

Actualmente la aplicación se encuentra desplegada en una versión sin el medidor de disnea y las sugerencias aeróbicas, pero se pretende realizar una actualizacion despues de que se muestre mejoras en la usabilidad. Según se muestra en las gráficas de Google Play Console, para la mayoría de los usuarios que están dentro del proceso de rehabilitación pulmonar se ha optimizado su rendimiento y usabilidad.

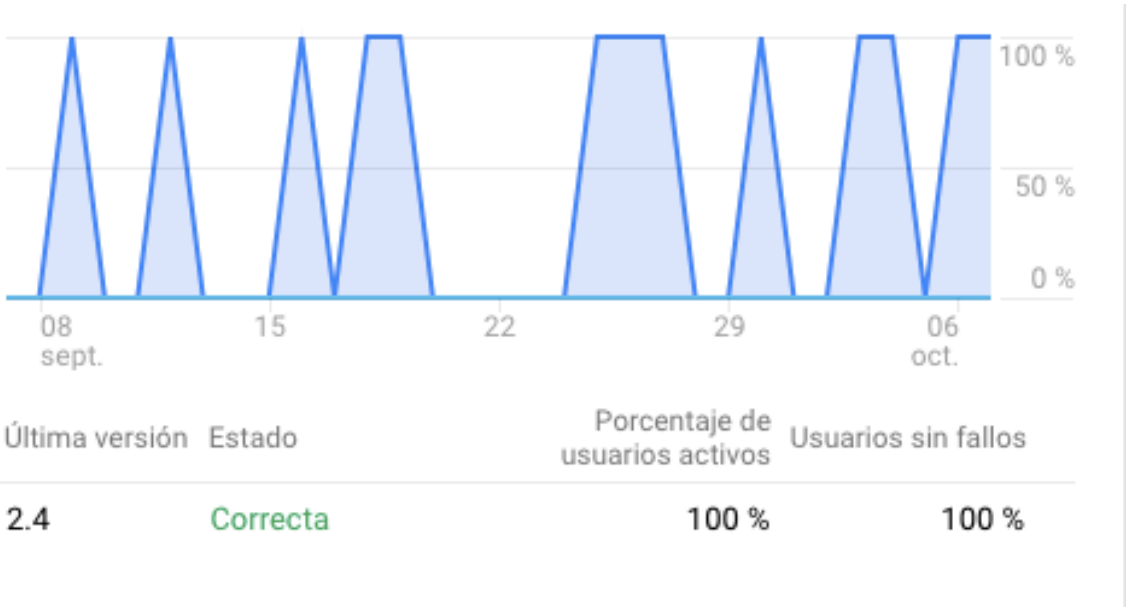


*Figura 24. Fallo de la aplicación después del lanzamiento de la versión 2.4 de la aplicación*

La prueba de usabilidad dispuesta se hizo posterior al despliegue. Se realizó la actualización de la nueva versión de la App dentro de la IPS Neumomed durante las citas de ingreso para pacientes aceptados en el programa de rehabilitación pulmonar. En estas citas de ingreso algunos pacientes realizaron un registro de ejercicios acompañados del personal de rehabilitación utilizando el Doctor Neumo instalado en sus teléfonos personales y evidenciando una mejora en los atributos de la aplicación que se muestran a continuación.

*Tabla 4. Evaluacion de usabilidad*

| Atributo | Definición | Calificación | Justificación |
|---|---|---|---|
| Momento | Un servicio que cuente con este atributo debe estar disponible en cualquier instante de tiempo en que el usuario desee usar dicho servicio. | 5 | El paciente puede ingresar a registrar sus ejercicios en cualquier momento, la obtención de la información y la consolidación de la información se da en forma correcta. |
| Movilidad | Un servicio móvil debe ser "móvil" por naturaleza, la ubicación debe ser una parte integral | 4 | El usuario se puede desplazar a cualquier lugar y realizar sus registros, siempre y cuando |

| | | | |
|---|---|---|---|
| | del servicio. | | tenga cobertura del operador o acceso a una red. |
| Dinero | Como cualquier acción comercial, un servicio móvil tiene un fin lucrativo, ya sea para el operador, para el proveedor del servicio o para el usuario. | 5 | Aunque la aplicación es de descarga gratuita, ésta genera ingresos al operador al ocasionar tráfico en la red de datos. La IPS Neumomed reduce costos por la atención de pacientes pulmonares a distancia, lo que permite redirigir los ingresos a casos de mayor relevancia. El paciente reduce los costos de traslado a la EPS |
| Yo | Se refiere al nivel de personalización de un servicio. | 5 | El servicio presenta un grado de personalización porque permite que el usuario seleccione: registrar ejercicios, ver mensajes, llenar formularios y hacer preguntas frecuentes. Sugiere respuestas a preguntas del asistente virtual. |
| Máquina | La tecnología (terminal o redes) siempre es el factor que posibilita o limita; el atributo máquina busca añadir potencia a los dispositivos de última generación que cada vez tienen mayores prestaciones a nivel de hardware y | 4 | El servicio solo puede ser soportado por celulares que posean Java o Android, y conexión a la red de datos. El usuario puede hacer un buen uso de la aplicación sin importar la resolución de la pantalla y el tipo del |

| | | | |
|---|---|---|---|
| | software. | | teclado. |
| Multiusuario | Busca extenderse dentro de la comunidad, que el servicio sea interactivo y que pueda utilizarse por múltiples usuarios de manera simultánea | 5 | la aplicación permite una comunicación en dos direcciones. Paciente←→ Bot |

## 7. Conclusiones

La introducción de estas nuevas tecnologías digitales permite construir una relación distinta con el paciente, centrada en sus necesidades, transparente, ágil y continua, donde el usuario goza de mucha más información, participación y autonomía. Tienen el potencial de simplificar tanto procesos administrativos como asistenciales, para mejorar la calidad y reducir el coste de la atención médica.

La aplicación es una forma útil para la comunicación de los pacientes en rehabilitación pulmonar con el asistente virtual y favorece una vez más a la participación del paciente en el desarrollo de los ejercicios a distancia.

La gamificación utiliza la predisposición natural humana hacia la competición y el juego para hacer menos aburridas determinadas tareas. Unas tareas que, con este método, pasan a ser realizadas de forma más dinámica y efectiva.

Los conceptos de gamificación introducidos en la aplicación en el medidor de disnea y en los ejercicios aeróbicos sugeridos son un gran avance para mejorar la usabilidad en la aplicación. La mejora en la usabilidad se observa desde las gráficas provenientes de firebase y de google play console.

La estructuración del desarrollo de la aplicación en lenguaje nativo para Android, deja la posibilidad de intervenir más en conceptos de gamificación para otras actividades que se realizan con los pacientes respiratorios.

La arquitectura implementada es una base estable e importante para el crecimiento de la aplicación y el escalamiento con futuros desarrollos.

## 8. Referencias Bibliográficas


[1] R. Nussbaum, C. Kelly, E. Quinby, A. Mac, B. Parmanto, B.E. Dicianno, Systematic Review of Mobile Health Applications in Rehabilitation, Arch. Phys. Med. Rehabil. 100 (2019) 115–127. doi:10.1016/j.apmr.2018.07.439.

[2] N. Xi, J. Hamari, International Journal of Information Management Does gamification satisfy needs ? A study on the relationship between gamification features and intrinsic need satisfaction, Int. J. Inf. Manage. 46 (2019) 210–221. doi:10.1016/j.ijinfomgt.2018.12.002.

[3] M. Rajak, K. Shaw, Technology in Society Evaluation and selection of mobile health ( mHealth ) applications using AHP and fuzzy TOPSIS, Technol. Soc. 59 (2019) 101186. doi:10.1016/j.techsoc.2019.101186.

[4] S. Al-sharhan, E. Omran, K. Lari, An integrated holistic model for an eHealth system : A national implementation approach and a new cloud-based security model, Int. J. Inf. Manage. 47 (2019) 121–130. doi:10.1016/j.ijinfomgt.2018.12.009.

[5] A. Developers, Android Studio, (n.d.). https://developer.android.com/studio/intro/index.html?hl=es-419.

[6] W.C. Huaman, Los 10 patrones comunes de arquitectura de software, (n.d.).

[7] C.L. Hsu, M.C. Chen, How does gamification improve user experience? An empirical investigation on the antecedents and consequences of user experience and its mediating role, Technol. Forecast. Soc. Change. 132 (2018) 118–129. doi:10.1016/j.techfore.2018.01.023.

[8] Y.M.F. Morán, Usabilidad de los Sistemas de Información en Salud dentro de escenarios de atención crítica: un estudio de los sistemas de historia clínica en IPS de alta complejidad Colombianas., Bogotá, Colombia, 2013. http://bdigital.unal.edu.co/43054/1/59801625.2013.pdf.

[9] L. P. Cancio and Mercedes Moráguez Bergues, Usabilidad de los sitios Web, los métodos y las técnicas para la evaluación/Usability of Web sites, methods and evaluation techniques., (n.d.).

[10] J. G. Carmona, SOLID Y GRASP. Buenas prácticas hacia el éxito en el desarrollo de software, (2012).

[11] J. Martí, Gamificación, ¿la gran solución para el aprendizaje?, XarxaTIC. (2019).