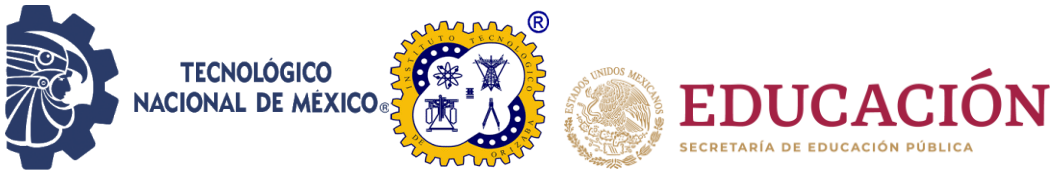

Instituto Tecnológico de Orizaba

División de Estudios de Posgrado e Investigación

Maestría en Sistemas Computacionales

TESIS

"Generación automática de interfaces de usuario de aplicaciones para dispositivos móviles para el ámbito de cuidado de la salud mediante comandos de voz"

PRESENTA:

I.S.C. Jesús Fernández Avelino

PARA OBTENER EL GRADO DE:

Maestro en Sistemas Computacionales

DIRECTOR:

Dr. Giner Alor Hernández

CO-DIRECTOR:

Dr. Mario Andrés Paredes Valverde

Contenido

Resumen ..... **VI**
Abstract ..... **VII**
Introducción ..... **VIII**
Capítulo I. Antecedentes **1**
1.1. Marco Conceptual **1**
1.1.1 Ingeniería de software **1**
1.1.2 El proceso del software **2**
1.1.3 Generación automática de código **5**
1.1.3.1 Lenguaje controlado **7**
1.1.3.2 Lenguaje de programación **7**
1.1.3.3 UML (Unified Modeling Language, Lenguaje de Modelado Unificado) **7**
1.1.3.4 Herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software asistida por Computadora) **7**
1.1.4 Agente conversacional **7**
1.1.5 Procesamiento de Lenguaje Natural **8**
1.1.6 Entendimiento del Lenguaje Natural **9**
1.1.7 Ontología **9**
1.1.8 Reconocimiento de entidades nombradas **10**
1.1.9 Administrador de diálogo **10**
1.1.9.1 Arquitectura tradicional de los sistemas de diálogo **10**
1.1.9.2 El rol de un administrador de diálogo **12**
1.1.10 Dialogflow **13**
1.1.11 Patrones de diseño de interfaz de usuario **15**
1.2 Planteamiento del problema **16**
1.3 Objetivo general y específicos **17**
1.4 Justificación **18**
Capítulo II. Estado de la práctica **19**
2.1 Trabajos relacionados **19**
2.2 Análisis comparativo de trabajos relacionados **29**
2.3 Solución propuesta **36**
2.4 Justificación de la solución seleccionada **37**

Capítulo III. Aplicación de la metodología 38
3.1 Análisis 38
3.1.1 Análisis de requerimientos funcionales 38
3.1.2 Flujo de trabajo para la generación de una aplicación móvil 40
3.1.3 Análisis de PDIU de aplicaciones móviles médicas 41
3.2 Diseño 46
3.2.1 Diseño de código intermedio XML 46
3.2.2 Diseño de la interfaz gráfica de usuario 47
3.2.3 Arquitectura del generador de aplicaciones móviles 49
3.3 Desarrollo 50
3.3.1 Proceso automático de generación de interfaces de usuario en el ámbito médico 50
3.3.2 Construcción de agente conversacional en Dialogflow 53
3.3.3 Reconocimiento de voz 62
3.3.4 Síntesis de voz 64
3.3.5 Construcción de XML 65
3.3.6 Desarrollo de aplicaciones médicas para las plataformas Android y iOS 67
3.3.7 Proceso de transformación de la intención del usuario 80
3.3.8 Identificación de los elementos de intención del usuario 80
3.3.9 Módulo de generación de código nativo de aplicaciones móviles 84
Capítulo IV. Resultados 86
4.1 Caso de estudio 1: Generación de aplicación para comunicación clínica para iOS 86
4.1.1 Generación de aplicación 86
4.2 Caso de estudio 2: Generación de aplicación de comunicación clínica para Android 91
4.1.1 Generación de aplicación 91
Capítulo V. Conclusiones 100
5.1 Conclusiones 100
5.2 Recomendaciones 101
Productos académicos 103
Referencias 104

Índice de tablas

Tabla 2.1 Análisis comparativo de trabajos relacionados con el reconocimiento de voz..... 29
Tabla 2.2 Análisis de la propuesta de solución 36
Tabla 3.1 Aplicaciones médicas según su uso..... 41
Tabla 3.2 UIDPs por tipo de aplicación de uso profesional 42
Tabla 3.3 UIDPs por tipo de aplicación de uso común..... 45

Índice de figuras

Figura 1.1 Capas de la ingeniería de software 2
Figura 1.2 Arquitectura tradicional de un sistema de diálogo..... 11
Figura 1.3 El rol de administrador de diálogo..... 13
Figura 3.1 Requerimientos funcionales del generador de aplicaciones móviles ... 39
Figura 3 2 Flujo de trabajo del generador de aplicaciones móviles..... 40
Figura 3.3 PDIU identificados en aplicaciones de uso profesional feedback, Navigation Tabs y Vertical Dropdown Menu..... 43
Figura 3.4 PDIU identificados en aplicaciones de uso profesional Gallery, Datalist y Search..... 43
Figura 3.5 PDIU identificados en aplicaciones de uso profesional Form, Navigation Tabs y Detail View..... 44
Figura 3.6 PDIU identificados en aplicaciones de uso profesional Stats, Dashboard y Login..... 44
Figura 3.7 PDIU identificados en aplicaciones de uso común Feedback, Maps y Splashscreen..... 45
Figura 3.8 PDIU identificados en aplicaciones de uso común Login, Form y Dashboard..... 45
Figura 3.9 Estructura de código intermedio XML 46
Figura 3.10 Elementos de IU del sistema..... 48
Figura 3.11 IU del historial de conversación..... 48
Figura 3.12 IU Generando la aplicación móvil..... 49

Figura 3.13 Arquitectura del generador de apliciones móviles 50
Figura 3.14 Descripción del proceso 51
Figura 3.15 Configuración de idioma en Dialogflow 53
Figura 3.16 Frases de entrenamineto en Dialogflow 54
Figura 3.17 Sección de acciones y parámetros de Dialogflow 55
Figura 3.18 Respuestas en Dialogflow 56
Figura 3.19 Entidades para el generador de aplicaciones móviles 58
Figura 3.20 Ejemplo de entrada de entidad para especificar el dominio 58
Figura 3.21 Pantalla inicial del chat 67
Figura 3.22 Pantalla de registro del chat 68
Figura 3. 23 Pantalla de inicio de sesión del chat 69
Figura 3.24 Historico de chats 70
Figura 3.25 Busqueda de usuarios del chat 71
Figura 3.26 Conversación con un usuario en el chat 71
Figura 3.27 Aplicación en iOS® para localización de entidades médicas 77
Figura 3.28 Identificación de elementos de la intención del usuario 81
Figura 3.29 Sinónimos para la intención del usuario 82
Figura 3.30 Sinónimos para el dispositivo 82
Figura 3.31 Sinónimos para el contenido 83
Figura 3.32 Sinónimos para el dominio 83
Figura 3.33 Plataformas compatibles 83
Figura 4.1 Pantalla de inicio del generador de código de aplicaciones móviles 86
Figura 4.2 Agente conversacional pide instrucciones al usuario 87
Figura 4.3 Usuario dicta instrucciones al agente conversacional 87
Figura 4.4 Pantalla de generación de la aplicación móvil 88
Figura 4.5 Pantalla de proceso de generación finalizado 88
Figura 4. 6 Descarga de la aplicación móvil que se generó 89
Figura 4.7 Estructura del proyecto para iOS 89
Figura 4. 8 Cargar el proyecto generado a Xcode® 90
Figura 4.9 Ejecución de aplicación de localización de entidades médicas para iOS® 90

Figura 4.10 Pantalla de inicio del generador de código de aplicaciones móviles .. 91
Figura 4.11 Agente conversacional escucha instrucciones del usuario 92
Figura 4.12 Agente conversacional pregunta el tipo de dispositivo 93
Figura 4. 13 Agente conversacional pregunta la plataforma 93
Figura 4.14 Agente conversacional pide dominio de la aplicación 94
Figura 4.15 Agente conversacional pregunta el contenido..... 94
Figura 4.16 Inicia el proceso de generación de la aplicación móvil 95
Figura 4.17 Proceso de generación de código finalizado..... 95
Figura 4. 18 Descarga del proyecto generado 96
Figura 4.19 Estructura del proyecto generado 96
Figura 4. 20 Carga del proyecto generado en Android Studio™ 97
Figura 4. 21 Aplicación cominicación clínica en ejecución, pantalla inicial..... 97
Figura 4. 22 Aplicación cominicación clínica en ejecución, pantalla de registro.... 98
Figura 4.23 Aplicación cominicación clínica en ejecución, pantalla de usuarios ... 98
Figura 4.24 Aplicación cominicación clínica en ejecución, pantalla de conversación
..... 99
Figura 4.25 Aplicación cominicación clínica en ejecución, pantalla de usuarios ... 99

Índice de código

Listado de código 3.1 consumo de servicio WEB de Dialogflow con CURL..... 59
Listado de código 3.2 estructura de la espuesta del servicio WEB de Dialogflow 59
Listado de código 3.3 interfaces de compatibilidad con Chrome..... 63
Listado de código 3.4 función para el reconocimiento de voz en el generador de aplicaciones..... 63
Listado de código 3.5 función para realizar la sintesis de voz en el generador de aplicaciones..... 64
Listado de código 3. 6 ejemplo de código intermedio XML 65
Listado de código 3.7 MessagesActivity.java 72
Listado de código 3. 8 MedicalEntities.swift 77

Resumen

Actualmente, existe una gran cantidad de aplicaciones móviles que solventan necesidades tanto generales como específicas en distintos dominios. El desarrollo de aplicaciones requiere de una gran inversión de tiempo y esfuerzo por parte de uno o más desarrolladores, los cuales son responsables de la especificación, diseño e implementación de la aplicación. Sin embargo, el esfuerzo de desarrollo y diseño a menudo se repite en muchas de estas aplicaciones. Este factor permite detectar patrones de diseño de interfaces de usuario los cuales dan lugar a la reutilización de código fuente de aplicaciones.

Teniendo en cuenta esta problemática, el presente proyecto de tesis busca desarrollar un componente de software que genere automáticamente interfaces de usuarios de dispositivos móviles en el contexto de cuidado de la salud, en conjunto de un agente conversacional por medio de detección de comandos de voz.

Se espera que con este componente de software se optimice el tiempo y esfuerzo que implica el proceso de desarrollo de software de manera eficaz y eficiente. Además, se consideran los principios operativos del software, se busca introducir al usuario una interfaz novedosa e innovadora por medio de comandos de voz y así facilitar el uso y experiencia de usuario del componente de software.

Abstract

Currently, there are a lot of mobile applications that solve general and specific needs in different domains. Application development requires investment of time and effort from one or more developers, who are responsible for the specification, design and implementation activities. However, the design and development effort are often repeated in many of these applications. This factor allows the detection of user interface design patterns which lead to the reuse of application source code.

Taking this problem into consideration, this project seeks to develop a software component that automatically generates user interfaces for mobile devices in the context of health care, supported by a conversational agent by voice recognition.

This software component is expected to optimize the time and effort involved in the software development process effectively and efficiently. Furthermore, the operating principles of the software are considered, the aim is to introduce the user to a new and innovative interface by voice recognition so it helps use and user experience of the software component.

Introducción

Existen numerosas aplicaciones en el contexto del cuidado de la salud que registran datos médicos en un esfuerzo de prevenir, controlar y monitorizar enfermedades tales como diabetes e hipertensión, por ejemplo, existen sistemas de dictado para ahorrar tiempo y esfuerzo.

Por otra parte, el reconocimiento de voz es la capacidad de una máquina para recibir e interpretar dictados o para comprender y ejecutar comandos de voz. El reconocimiento de voz es una de las formas de comunicación humano-computadora que se está sobreponiendo a otras formas de interacción tales como el teclado y el ratón o la funcionalidad táctil. Actualmente, ya es posible interactuar mediante comandos de voz con dispositivos móviles, automóviles, televisiones inteligentes y otras tecnologías domóticas para llevar a cabo tareas cotidianas. Sin embargo, es tipo de interacción no solo se da en dichos contextos. El reconocimiento de voz juega ya un papel importante en el sector empresarial y profesional.

Como contribuciones para la ingeniería de software de este trabajo de tesis, se tienen cuatro aspectos importantes: 1) aportar una nueva forma de generar código tomando en cuenta que el proceso de desarrollo de software sea mucho más intuitivo y que sea tan bueno como lo hace un ser humano, 2) diseñar un agente conversacional y un conjunto de reglas que permitan la generación automática de código, 3) tener un mejor procesamiento de lenguaje natural para aumentar la identificación de la intención del usuario y así generar una herramienta para agilizar el desarrollo del software y 4) enriquecer las aplicaciones que se sumen a las actividades diarias y aumente la motivación para el desarrollo de software.

Este documento está estructurado de la siguiente manera, en el capítulo 1 se detallan los antecedentes y su marco conceptual, en el capítulo 2 se presenta el estado del arte referente a los diversos trabajos relacionados al reconocimiento de voz. En el capítulo 3 la aplicación de la metodología, que incluye una descripción de los PDIU identificados, analizados y aplicados para la generación automática de aplicaciones del dominio médico. Se describe lo que implica el proceso de

generación de software. En el capítulo 4 se definen dos casos de estudio que hacen uso del generador de aplicaciones y la utilización del agente conversacional. Finalmente, el capítulo 5 se presenta las conclusiones del trabajo de investigación.

Capítulo I. Antecedentes

1.1. Marco Conceptual

A continuación, se definen algunos términos relevantes para el trabajo de investigación.

1.1.1 Ingeniería de software

La ingeniería de software es una tecnología con varias capas. Como se aprecia en la Figura 1.1, cualquier enfoque de ingeniería (incluso la de software) se basa en un compromiso organizacional con la calidad. La administración total de la calidad, Six Sigma y otras filosofías similares alimentan la cultura de mejora continua, y es esta cultura la que lleva en última instancia al desarrollo de enfoques cada vez más eficaces de la ingeniería de software. El fundamento en el que se apoya la ingeniería de software es el compromiso con la calidad.

El fundamento para la ingeniería de software es la capa proceso. El proceso de ingeniería de software es el aglutinante que une las capas de la tecnología y permite el desarrollo racional y oportuno del software de cómputo. El proceso define una estructura que se establece para la obtención eficaz de tecnología de ingeniería de software. El proceso de software forma la base para el control de la administración de proyectos de software, y establece el contexto en el que se aplican métodos técnicos, se generan productos del trabajo (modelos, documentos, datos, reportes, formatos, entre otros.), se establecen puntos de referencia, se asegura la calidad y se administra el cambio de manera apropiada.

Los métodos de la ingeniería de software proporcionan la experiencia técnica para elaborar software. Incluyen un conjunto amplio de tareas, como comunicación, análisis de los requerimientos, modelación del diseño, construcción del programa, pruebas y apoyo. Los métodos de la ingeniería de software se basan en un conjunto

de principios fundamentales que gobiernan cada área de la tecnología e incluyen actividades de modelación y otras técnicas descriptivas.

Las herramientas de la ingeniería de software proporcionan un apoyo automatizado o semiautomatizado para el proceso y los métodos. Cuando se integran las herramientas de modo que la información que se crea se utilice por otra, queda establecido un sistema llamado ingeniería de software asistido por computadora que apoya el desarrollo de software.

Figura 1.1 Capas de la ingeniería de software

1.1.2 El proceso del software

En el contexto de la ingeniería de software, un proceso no es una prescripción rígida de cómo elaborar software de cómputo. Por el contrario, es un enfoque adaptable que permite que las personas que hacen el trabajo (el equipo de software) busquen y elijan el conjunto apropiado de acciones y tareas para el trabajo. Se busca siempre entregar el software en forma oportuna y con calidad suficiente para satisfacer a quienes patrocinaron su creación y a aquellos que lo usarán [1].

La estructura del proceso establece el fundamento para el proceso completo de la ingeniería de software por medio de la identificación de un número pequeño de actividades estructurales que sean aplicables a todos los proyectos de software, sin importar su tamaño o complejidad. Además, la estructura del proceso incluye un conjunto de actividades sombrilla que son aplicables a través de todo el proceso del

software. Una estructura de proceso general para la ingeniería de software consta de cinco actividades:

- **Comunicación.** Antes de que comience cualquier trabajo técnico, tiene importancia crítica comunicarse y colaborar con el cliente (y con otros participantes). Se busca entender los objetivos de los participantes respecto del proyecto, y reunir los requerimientos que ayuden a definir las características y funciones del software.

- **Planeación.** Cualquier viaje complicado se simplifica si existe un mapa. Un proyecto de software es un viaje difícil, y la actividad de planeación crea un "mapa" que guía al equipo mientras viaja. El mapa llamado plan del proyecto de software define el trabajo de ingeniería de software al describir las tareas técnicas por realizar, los riesgos probables, los recursos que se requieren, los productos del trabajo que se obtendrán y una programación de las actividades.
- **Modelado.** Crear un "bosquejo" del objeto por hacer a fin de entender el panorama general de cómo se verá arquitectónicamente, cómo ajustan entre sí las partes constituyentes y muchas características más. Si se requiere, refina el bosquejo con más y más detalles en un esfuerzo por comprender mejor el problema y cómo resolverlo. Un ingeniero de software hace lo mismo al crear modelos a fin de entender mejor los requerimientos del software y el diseño que los satisfará.
- **Construcción.** Esta actividad combina la generación de código (ya sea manual o automática) y las pruebas que se requieren para descubrir errores en este.
- **Despliegue.** El software (como entidad completa o como un incremento parcialmente terminado) se entrega al consumidor que lo evalúa y que le da retroalimentación, misma que se basa en dicha evaluación.

Estas cinco actividades estructurales genéricas se usan durante el desarrollo de programas pequeños y sencillos, en la creación de aplicaciones WEB grandes y en

la ingeniería de sistemas enormes y complejos basados en computadoras. Los detalles del proceso de software serán distintos en cada caso, pero las actividades estructurales son las mismas.

Para muchos proyectos de software, las actividades estructurales se aplican en forma iterativa a medida que avanza el proyecto. Es decir, la comunicación, la planeación, el modelado, la construcción y el despliegue se ejecutan a través de cierto número de repeticiones del proyecto. Cada iteración produce un incremento del software que da a los participantes un subconjunto de características y funcionalidad generales del software. Conforme se produce cada incremento, el software se hace más y más completo.

Las actividades estructurales del proceso de ingeniería de software son complementadas por cierto número de actividades sombrilla. En general, las actividades sombrilla se aplican a lo largo de un proyecto de software y ayudan al equipo que lo lleva a cabo a administrar y controlar el avance, la calidad, el cambio y el riesgo. Es común que las actividades sombrilla sean las siguientes:

- **Seguimiento y control del proyecto de software:** permite que el equipo de software evalué el progreso comparándolo con el plan del proyecto y tome cualquier acción necesaria para apegarse a la programación de actividades.
- **Administración del riesgo:** evalúa los riesgos que afecten el resultado del proyecto o la calidad del producto.
- **Aseguramiento de la calidad del software:** define y ejecuta las actividades requeridas para garantizar la calidad del software.
- **Revisiones técnicas:** evalúa los productos del trabajo de la ingeniería de software a fin de descubrir y eliminar errores antes de que se propaguen a la siguiente actividad.
- **Medición:** define y reúne mediciones del proceso, proyecto y producto para ayudar al equipo a entregar el software que satisfaga las necesidades de los participantes; Se utiliza junto con todas las demás actividades estructurales y sombrilla.

- **Administración de la configuración del software:** administra los efectos del cambio a lo largo del proceso del software.
- **Administración de la reutilización:** define criterios para volver a usar el producto del trabajo (incluso los componentes del software) y establece mecanismos para obtener componentes reutilizables.
- **Preparación y producción del producto del trabajo:** agrupa las actividades requeridas para crear productos del trabajo, tales como modelos, documentos, registros, formatos y listas.

1.1.3 Generación automática de código

La generación automática de código a partir de las especificaciones formales determina dos grupos importantes. Uno en la parte académica, donde se definen conjuntos de reglas, y el otro grupo en el cual se vienen desarrollando herramientas CASE como apoyo al proceso.

Gomes et al. [2] propusieron una metodología para la generación automática de código en lenguaje C a partir de especificaciones formales, utilizando el Método-B. Este método provee un formalismo para el refinamiento de las especificaciones y se estructura en módulos que se clasifican de acuerdo con su nivel de abstracción: maquina, refinamiento e implementación.

Peckham et al. [3] propusieron un lenguaje natural controlado y unas plantillas para la especificación de casos de uso. Estos autores apoyan el desarrollo de software con una herramienta para la generación de procesos algebraicos a partir de los casos de uso.

Ramkarthik et al. [4] definieron un conjunto de reglas para generar la estructura básica de una aplicación en Java, tomando como punto de partida las especificaciones escritas en subconjunto de objetos Z (notación para las especificaciones formales).

El segundo grupo de generación automática de código se enfoca en el desarrollo de herramientas CASE, las cuales permiten mejorar el proceso de generación de código. Entre ellas se encuentran NL- OOPS (*Natural Language Object-Oriented Production System*, Sistema de Producción de Lenguaje Natural Orientado a Objetos), LIDA (*Linguistic Assistant for Domain Analysis*, Asistente Lingüístico para el Análisis de Dominio), CM-Builder (*Class Model Builder*, Generador de Modelos de Clases), RADD (*Rapid Application and Database Development*, Desarrollo Rápido se Aplicaciones y Base de datos) y NIBA. NL-OOPS es una herramienta CASE que se basa en un sistema de procesamiento del lenguaje natural denominado LOLITA (*Largescale Object-based Linguistic Interactor, Translator and Analyser*, Analista, Traductor e Interactor Lingüístico de Gran escala Orientado a Objetos), el cual contiene una serie de funciones para el análisis del lenguaje natural. NL-OOPS entrega como resultado una versión preliminar del diagrama de clases de OMT. LIDA es una herramienta CASE que analiza el texto en lenguaje natural y hace una clasificación en tres categorías gramaticales: sustantivos, verbos y adjetivos [5]; con esta información, el analista asigna a cada categoría, manualmente, un elemento del diagrama de clases y, de esta manera, LIDA permite trazar este diagrama. CM-Builder es una herramienta CASE que permite la elaboración del diagrama de clases a partir de textos en inglés, utilizando como modelo intermedio una red semántica [6]. RADD es una herramienta CASE que se enfoca en la obtención del diagrama entidad- relación a partir de un lenguaje controlado. Además, emplea una "herramienta de diálogo moderado" que posibilita la comunicación con el diseñador de la base de datos en lenguaje natural [7]. NIBA busca la elaboración de diferentes diagramas UML (especialmente clases y actividades, aunque se obtienen otros como secuencias y comunicación), empleando un conjunto de esquemas intermedios que denominaron KCPM Klagenfurt Conceptual Predesign Model [8]. KCPM posee formas diferentes de representación del conocimiento para diferentes diagramas de UML. NIBA, además de generar los diferentes diagramas UML, genera el código fuente en C++ [9].

1.1.3.1 Lenguaje controlado

Subconjunto del lenguaje natural en el que, generalmente, se restringe ya sea el vocabulario, la estructura o ambos. Se emplean para establecer unas condiciones iniciales para la generación del código, que permitan una transición suave hacia el código fuente [10].

1.1.3.2 Lenguaje de programación

Sistema de notación para describir conceptos en una forma legible para la máquina. Un lenguaje de programación es el fundamento para el código que se genera [8].

1.1.3.3 UML (Unified Modeling Language, Lenguaje de Modelado Unificado)

Lenguaje de modelado gráfico para visualizar, especificar, construir y documentar los elementos que forman un sistema software orientado a objetos. En la generación automática de código, cumple en la actualidad un papel fundamental, pues permite a los analistas expresar las necesidades en un lenguaje que, aunque técnico, se acerca al formalismo [11].

1.1.3.4 Herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software asistida por Computadora)

Conjunto de aplicaciones informáticas que dan asistencia a los analistas y desarrolladores, durante todo el ciclo de vida del software. Estas herramientas se destinan a aumentar la productividad y reducir costos en tiempo y dinero. Son importantes porque muchas de ellas contienen como una opción la generación de código, aunque en la actualidad lo hacen de forma incompleta [12].

1.1.4 Agente conversacional

De acuerdo con [13], un agente conversacional es cualquier sistema de diálogo que realiza procesamiento de lenguaje natural y responde usando lenguaje humano. Esos agentes representan la implementación práctica de la computación lingüística,

empleados en internet como *chatbots* y en los dispositivos móviles como asistentes. La interpretación y respuesta de los agentes conversacionales no necesariamente es por medio de texto, también, se utilizan medios como: reconocimiento de voz, imágenes y gestos virtuales.

El proceso que realiza un agente conversacional consta de tres fases:

1) En la primera fase se convierte la entrada del mundo real en un código de máquina universal utilizando algún tipo de reconocimiento automático del habla, o el reconocedor óptico de gesto o escritura a mano.
2) La entrada se interpreta usando alguna forma de NLU (*Natural Language Understanding,* Entendimiento de Lenguaje Natural), esta fase va más allá del procesamiento de lenguaje natural estándar al incluir la identificación del nombre correcto, parte del etiquetado de voz y un analizador sintáctico/semántico. Paralelamente a la interpretación, un administrador de diálogo rastrea el historial y el estado del diálogo, generalmente manteniendo la conversación en una pista lógica activando/desactivando los dominios de subtarea apropiados.
3) La respuesta se envía a un generador de salida, por lo general coincide con el formato de entrada.

1.1.5 Procesamiento de Lenguaje Natural

El PLN (Procesamiento de Lenguaje Natural) es el uso de algoritmos para determinar las propiedades del lenguaje natural y humano para que las computadoras entiendan lo que los humanos escribieron o dijeron. PLN incluye sistemas de computación para enseñar cómo extraer datos de cuerpos de texto escrito, traducir de un idioma a otro y reconocer palabras impresas o escritas a mano. En particular, PLN es el campo que permite el uso diario de asistentes virtuales como Siri [14].

1.1.6 Entendimiento del Lenguaje Natural

De acuerdo con Rouse [12] la comprensión del lenguaje natural es una rama de la Inteligencia Artificial que utiliza programas informáticos para comprender las aportaciones hechas en forma de oraciones en formato de texto o habla. NLU directamente habilita la interacción humano-computadora. La comprensión de NLU de los lenguajes humanos naturales permite que las computadoras entiendan los comandos sin la sintaxis formalizada de los lenguajes informáticos y que las computadoras se comuniquen con los humanos en sus propios idiomas.

El campo de NLU es un subconjunto importante y desafiante de procesamiento de lenguaje natural. Si bien ambos comprenden el lenguaje humano, NLU tiene la tarea de comunicarse con personas no capacitadas y de comprender su intención, lo que significa que NLU va más allá de la comprensión de las palabras e interpreta el significado, incluso cuenta con la capacidad de entender el significado a pesar de los errores humanos comunes, como las malas declaraciones, letras incorrectas o palabras transpuestas.

1.1.7 Ontología

En [16] se define que una ontología en AI (*Artificial Intelligence,* Inteligencia Artificial) es la representación del conocimiento de un dominio en específico que facilita la comprensión del mismo dominio. Es una combinación lógica de un dominio, conceptos y sus relaciones. Es una buena manera de representar gráficamente el conocimiento. La ontología para representar el conocimiento de dominio en una red gráfica consiste en algunos nodos y conexiones entre nodos. Los nodos representan conceptos de ese dominio mientras que las conexiones son las relaciones entre los conceptos. Este tipo de representación del conocimiento es valiosa, ya que facilita la comunicación entre las personas mediante una representación visual de los conceptos básicos de un dominio y la relación entre ellos, además, hace estándar la representación del conocimiento estructurándolo en una red. Esta representación de lenguaje natural del conocimiento no solo es amigable para el ser humano, sino que también para implementar en una computadora para otras aplicaciones. La

ontología se define como un diseño de máquina legible de una conceptualización compartida, que utiliza un tipo explícitamente definido de conceptos y restricciones que se aplican al uso de dichos conceptos, también, captura y representa el conocimiento que es aceptado por un grupo de masas, utiliza un vocabulario con el fin de representar el conocimiento en términos de objetos y su relación, este vocabulario hace fácil la comprensión de conocimiento entre dos o más sistemas inteligentes.

1.1.8 Reconocimiento de entidades nombradas

El término de “entidad nombrada” se acuñó en la sexta conferencia Message Understanding patrocinada inicialmente por DARPA (*Defense Advanced Research Projects Agency,* Agencia de Proyectos de Investigación Avanzada de Defensa), en ese tiempo, el término se enfoca a la extracción de información, tareas en donde información estructurada de las actividades de compañías y actividades de defensa relacionadas eran extraídas de texto no estructurado, como artículos en un periódico. Durante la implementación de dichas tareas la gente se dio cuenta que es esencial el reconocimiento de unidades de información, tales como: nombres, personas, organizaciones, nombres de lugares, y expresiones numéricas que incluyen tiempo, fechas, dinero y expresiones de porcentaje. La identificación de las referencias hacia ciertas entidades se reconoce como una subtarea de la extracción de la información y se nombró NERC (*Named Entity Recognition and Classification,* Reconocimiento y Clasificación de Entidades Nombradas) [17].

1.1.9 Administrador de diálogo

1.1.9.1 Arquitectura tradicional de los sistemas de diálogo

El campo de sistemas de diálogo es un área de investigación que está creciendo rápidamente debido al incremento de tecnologías de reconocimiento de voz que hacen posible la construcción de sistemas que los humanos utilicen fácilmente por medio de lenguajes naturales. El administrador de diálogo realiza tareas muy importantes como: análisis del discurso, acceso a base de datos, manipulación de

errores y el control de acciones [18]. En general, la arquitectura de los sistemas de diálogo se muestra en la Figura 1.2.

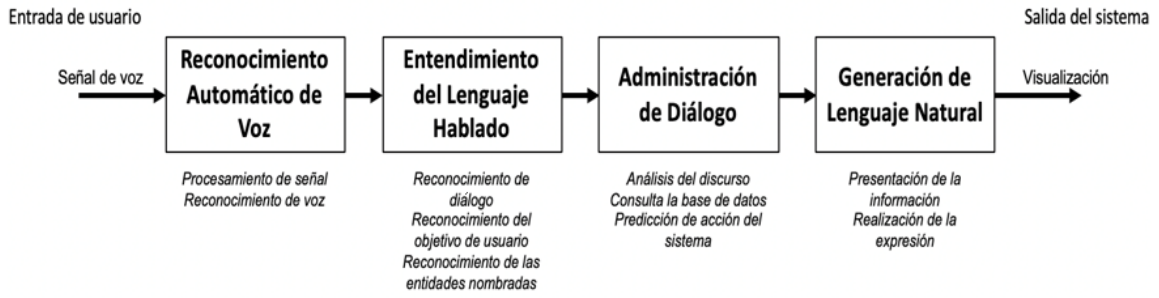

Figura 1.2 Arquitectura tradicional de un sistema de diálogo

- **Entrada de usuario:** La entrada del usuario usualmente es una señal de voz.
- **Reconocimiento Automático de Voz:** En el reconocimiento automático de voz, la señal de voz que se procesa para transformarla en una secuencia de vectores de parámetros. El reconocimiento de voz convierte el vector de parámetros en una entrada textual, por ejemplo, una secuencia de palabras.
- **Entendimiento de Lenguaje Hablado:** La entrada textual se analiza por un módulo de procesamiento de lenguaje natural, posteriormente, el módulo de SLU (*Spoken Language Understanding,* Entendimiento de Lenguaje Hablado) mapea el pre procesamiento de declaraciones a una representación con significado (por ejemplo, un marco semántico) en donde el acto del diálogo, objetivo del usuario y las entidades nombradas son extraídas por un mapeo semántico o un modelo estadístico.

- **Administración de Diálogo:** El administrador de diálogo es el núcleo de los sistemas de diálogo porque coordina la actividad de todos los componentes, controla el flujo del diálogo y comunica con las aplicaciones externas. El DM (*Dialog Management,* Administrador de diálogo) representa diferentes roles en el sistema los cuales incluyen análisis del discurso, consulta a la base de datos de conocimiento y la predicción de acciones de acuerdo con el contexto del discurso.

- **Generación de Lenguaje Natural:** A menudo, las respuestas del sistema se generan como un lenguaje natural con una lista de elementos de contenido de una parte de la base de datos de conocimientos externa que responden a una consulta del usuario en particular.
- **Salida del Sistema:** Finalmente, la salida del sistema se visualiza en una pantalla o se utiliza un módulo de TTS (Text-to-Speech, Texto a Voz) para transformar texto en voz.

1.1.9.2 El rol de un administrador de diálogo

A continuación, se enlistan los principales roles de un administrador de diálogo y se visualizan en la Figura 1.2 [15].

1. Buscar y proveer los resultados de consultas por medio de una base de datos de conocimientos externa, de acuerdo a la entrada y al contexto analizado.
2. Preguntar lo necesario para realizar una consulta apropiada.
3. Pedir al usuario clarificar la petición que realiza y re frasear la entrada del usuario si es necesario.
4. Predecir la siguiente acción del sistema a nivel conceptual para dar como salida una respuesta por medio de los módulos de NLG y text-to-speech.
5. Controlar mecanismos genéricos conversacionales para mantener una interacción común parecida a humano-humano.

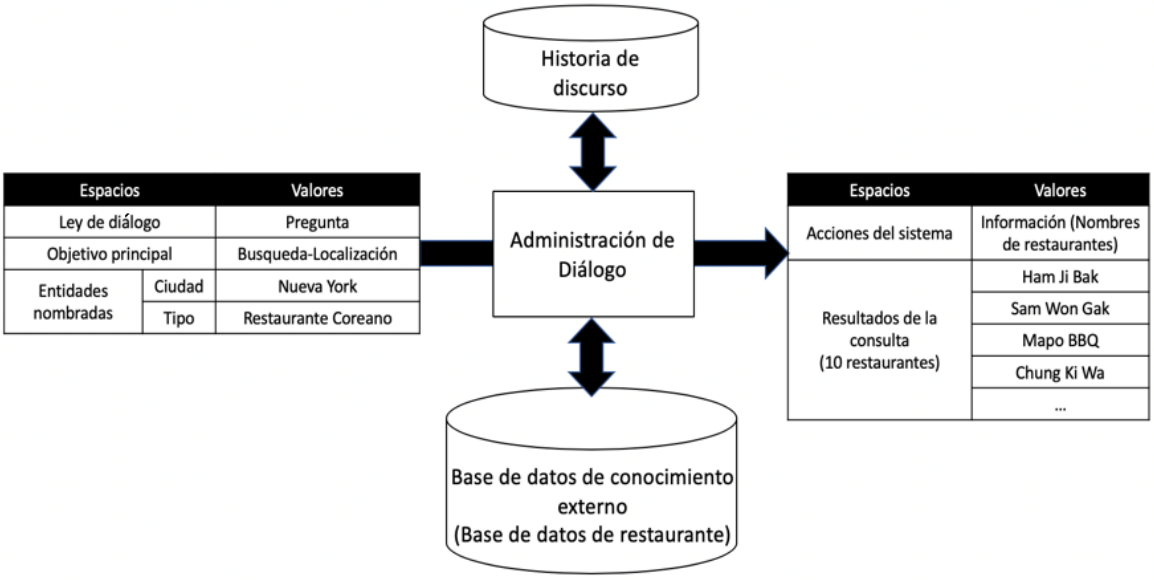

Figura 1.3 El rol de administrador de diálogo

1.1.10 Dialogflow

Dialogflow es un paquete integral de desarrollo que permite crear interfaces de conversación. Con solo crearlas una vez, es posible desplegarlas en cualquier sitio WEB, aplicación móvil, plataforma popular de mensajería y dispositivo de Internet de las cosas. Dialogflow se utiliza para crear interfaces, como bots de chat y respuestas de voz interactivas conversacionales, para bridar una interacción natural y llena de matices entre los usuarios y el software.

El procesamiento del lenguaje natural reconoce la intención del usuario y extrae entidades predefinidas, como el tiempo, la fecha y los números. Además, los agentes conversacionales tienen la capacidad de recibir un conjunto de datos como ejemplo para que se entrenen e identifiquen tipos de entidades personalizadas. Cabe mencionar que Dialogflow cuenta con más de 40 agentes predefinidos como plantillas.

Dialogflow proporciona un editor de código integrado para crear aplicaciones nativas sin servidor vinculadas a una interfaz de conversación a través de Cloud Functions

para Firebase. Sin embargo, la plataforma tiene la funcionalidad para utilizar un WEBhook propio y personalizado para alojarlo on-premise o en la nube.

Los conectores de conocimiento de Dialogflow, se encuentran en la fase beta, tienen la funcionalidad para añadir al agente datos en bloque de acuerdo a las reglas de negocio, como preguntas frecuentes y artículos de la base de conocimiento. Los conectores emplean algunas de las tecnologías que se utilizan en la Búsqueda y el Asistente de Google para extraer las respuestas correctas del corpus de datos proporcionado.

La funcionalidad de la telefonía de Dialogflow, que aún se encuentra en versión beta, tiene la funcionalidad para añadir al agente de Dialogflow un número de teléfono dedicado. Cuando los usuarios marquen ese nuevo número, hablarán directamente con el agente de Dialogflow. La funcionalidad de telefonía es un paquete fácil de usar que se basa en la inversión de Google en conectividad telefónica, la comprensión del lenguaje natural, el reconocimiento de voz, la síntesis de voz y otras características.

Dialogflow admite más de 20 idiomas y la integración con 14 plataformas distintas con un solo clic. Con Dialogflow se tiene la capacidad de crear un agente multilingüe con alcance mundial para interactuar con más audiencias.

Dialogflow ofrece más de una funcionalidad para incrementar la capacidad de una interfaz de conversación para que reconozca interacciones de voz y genere una respuesta vocal con una sola llamada a la API. Cuenta con las tecnologías de Transcripción de voz y Text-to-Speech de Google Cloud, y es compatible con los modos síncrono y de streaming en tiempo real.

El panel integrado de analíticas ofrece información valiosa sobre las interacciones de conversación; de esa forma, se tiene conocimiento para optimizar el bot para que reconozca mejor lo que el usuario quiere decir y le dé una respuesta más adecuada.

Ahora, Dialogflow examina todas las consultas de los usuarios con el análisis de opinión, una función con la tecnología de Natural Language de Cloud. Las valoraciones de este análisis no solo sirven para transferir a los usuarios insatisfechos a agentes humanos, sino también para comprender qué intents generan mejores opiniones.

En los chats, muchos usuarios escriben deprisa y no prestan atención a las faltas de ortografía ni a la gramática. Gracias a la función de Dialogflow para revisar la ortografía automáticamente, las faltas de ortografía se corrigen con una tecnología parecida a la que se utiliza en Google para la Búsqueda [16].

1.1.11 Patrones de diseño de interfaz de usuario

Los patrones de diseño de la interfaz de usuario son descripciones de las mejores prácticas dentro del diseño de la interfaz de usuario. Son soluciones generales y reutilizables para problemas comunes. Como tales, forman la columna vertebral del "soporte técnico". Sin embargo, como los patrones de diseño se aplican a una amplia variedad de casos, los diseñadores los adaptan al contexto específico de uso dentro de cada proyecto de diseño [17]. A menudo, un patrón de diseño de interfaz de usuario consiste de los siguientes elementos:

- **Problema:** El problema de la usabilidad cuando el usuario utiliza el sistema.
- **Contexto de uso:** La situación (en términos de tareas, usuarios y contexto de uso) que da lugar al problema de la usabilidad.
- **Principio:** Un patrón generalmente se basa en uno o más principios de diseño, como la gestión de errores o la coherencia de la guía del usuario.
- **Solución:** Una solución probada al problema. Una solución describe solo el núcleo del problema, y el diseñador tiene la libertad de implementarlo de muchas maneras.
- **Por qué:** Cómo y por qué el patrón trabaja así, incluyendo un análisis de cómo afectan ciertos atributos de usabilidad.

- **Ejemplos:** Cada ejemplo muestra cómo el patrón es aplicado en un sistema de la vida real. A menudo, el ejemplo es acompañado por imágenes del patrón y una breve descripción.
- **Implementación:** Algunos patrones proporcionan detalles de la implementación.

1.2 Planteamiento del problema

Ante la creciente adopción de dispositivos móviles, ha surgido una oportunidad de negocio que se enfoca en el desarrollo de aplicaciones para este tipo de dispositivos. Actualmente, existe una gran cantidad de aplicaciones móviles que solventan necesidades tanto generales como específicas en distintos dominios. Por ejemplo, en el contexto del cuidado de la salud existen numerosas aplicaciones que permiten registrar datos médicos en un esfuerzo de prevenir, controlar y monitorizar enfermedades tales como diabetes e hipertensión. El desarrollo de este tipo de aplicaciones requiere de una gran inversión de tiempo y esfuerzo por parte de uno o más desarrolladores, los cuales son responsables de la especificación, diseño e implementación de la aplicación. Sin embargo, el esfuerzo de desarrollo y diseño a menudo se repite en muchas de estas aplicaciones. Este factor permite detectar patrones de diseño de interfaces de usuario los cuales dan lugar a la reutilización de código fuente de aplicaciones.

Por otra parte, el reconocimiento de voz es la capacidad de una máquina para recibir e interpretar dictados o para comprender y ejecutar comandos de voz. El reconocimiento de voz es una de las formas de comunicación humano-computadora que se está sobreponiendo a otras formas de interacción tales como el teclado y ratón o la funcionalidad táctil. Actualmente, ya es posible interactuar mediante comandos de voz con dispositivos móviles, automóviles, televisiones inteligentes y otras tecnologías domóticas para llevar a cabo tareas cotidianas. Sin embargo, este tipo de interacción no solo se da en dichos contextos. El reconocimiento de voz juega ya un papel importante en el sector empresarial y profesional, por ejemplo, en

el dominio médico, donde existen sistemas de dictado que permiten ahorrar tiempo y esfuerzo.

En la literatura existen diversos esfuerzos de investigación enfocados en la generación automática de aplicaciones. Por ejemplo, Sánchez et al. [18] proponen un componente de software para la generación de interfaces de usuario a partir del procesamiento de una imagen. De forma similar, Cortes et al. [19] presentan un generador de aplicaciones educativas basado en patrones de diseño de interfaz de usuario. Finalmente, Modak et al. [20] presentan un sistema que proporciona una interfaz de lenguaje natural para automatizar la generación de páginas WEB. En el trabajo relacionado, ninguno de los enfoques presentados adopta un enfoque de reconocimiento de voz como medio de interacción para la generación automática de aplicaciones.

La idea central de esta propuesta consiste en desarrollar un componente de software capaz de generar automáticamente código fuente de interfaces de usuario de dispositivos móviles a partir de comandos de voz.

1.3 Objetivo general y específicos

A continuación, se muestra el objetivo general y específicos.

Objetivo general

Desarrollar un componente de software que permita la generación de código fuente de interfaces de usuario de aplicaciones móviles mediante comandos de voz.

Objetivos específicos

- Estudiar y analizar el estado de la práctica en los contextos de generación automática de código y reconocimiento de voz.
- Analizar las tecnologías y estándares para la generación automática de código y reconocimiento de voz para diseñar una arquitectura de solución.

- Desarrollar un componente de software para la generación automática de aplicaciones para dispositivos móviles en el contexto de cuidado de la salud por medio de comandos de voz.
- Diseñar el conjunto de prestaciones del componente de software para la generación automática de aplicaciones para dispositivos móviles.
- Evaluar el componente de software desarrollado en un caso de estudio para la generación de aplicaciones para dispositivos móviles en el dominio de cuidado de la salud.

1.4 Justificación

La idea central de esta propuesta consiste en desarrollar un componente de software capaz de generar automáticamente código fuente de interfaces de usuario de dispositivos móviles a partir de comandos de voz. El enfoque propuesto aplica técnicas de ingeniería de software y de procesamiento de lenguaje natural que permiten ocultar diversos aspectos de desarrollo a los usuarios y generar aplicaciones a través de una interfaz de usuario simple y fácil de usar. Entre los beneficios esperados se encuentran:

- Generar código fuente de interfaces de usuario de aplicaciones móviles mediante comandos de voz.
- El código fuente generado se ajusta a las necesidades de diversos dispositivos con distintos sistemas operativos manteniendo una adecuada funcionalidad de los componentes.

  Además, en la actualidad la mayoría de los agentes conversacionales se utilizan en idioma inglés, el componente de software propuesto se enfocará en el idioma español.

Capítulo II. Estado de la práctica

En este capítulo se presenta un resumen de los trabajos más significativos relacionados con el reconocimiento de voz, generación automática de código y aplicaciones orientadas al cuidado de la salud, los cuales permitieron obtener una gran cantidad de información valiosa, que se relaciona con la tesis.

2.1 Trabajos relacionados

Los sistemas de domótica están compuestos por varios dispositivos (foco, televisión, aire acondicionado, por mencionar algunos) que son controlados por una unidad central, conocida como *Gateway.* Erić et al. [21] realizaron una comparación entre las diferentes herramientas de reconocimiento de voz como Jasper platform, la API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) de Google speech, Alexa Voice Service y la API Bing speech, con el objetivo de implementarlas en sistemas de domótica. Así mismo, se menciona que el uso de herramientas específicas de reconocimiento de voz influye en la interfaz de comandos de voz y el rendimiento de la arquitectura que se implementa. La primera arquitectura que propusieron se basa en el servicio de reconocimiento de voz en la nube, servicio en la nube de domótica y dispositivos de grabación de voz de terceros. Por otra parte, la segunda arquitectura también utiliza un servicio de reconocimiento de voz en la nube, sin embargo, existe un *gateway* que interactúa con el servicio de reconocimiento de voz y los dispositivos finales. La tercera arquitectura solo está compuesta por un *gateway* que incluye una interfaz de reconocimiento de voz e interactúa directamente con los dispositivos finales, el *gateway* en esta arquitectura no necesita estar *online* a diferencia de las primeras dos arquitecturas. Por último, se comprobó el funcionamiento de las herramientas de control de voz en la domótica.

Los generadores de aplicaciones simplifican el desarrollo de software y hacen posible el desarrollo de una aplicación para diferentes plataformas usando el mismo proyecto. En este contexto, los patrones de diseño de interfaz de usuario facilitan la

interacción entre la aplicación y el usuario, además, brindan la oportunidad de hacer software de calidad. Cortes et al. [18] presentaron un generador de aplicaciones para dispositivos móviles basado en patrones de diseño de interfaz de usuario. El generador se utiliza a través de una plataforma WEB y está compuesto por cuatro capas: 1) presentación, 2) integración, 3) servicios y 4) de datos. Por otra parte, el proceso de una aplicación inicia por especificar en un archivo XML (*Extensible Markup Languague*, Lenguaje de Marcado Extensible) las características de la aplicación, tales como: el tipo de dispositivo, orientación, sistemas operativos, por mencionar algunas. El archivo XML se sube a través de la plataforma WEB en la capa de presentación, y posteriormente, se envía a la capa de integración en donde se valida la estructura del archivo XML. Después de que el archivo XML fue validado, pasa a la capa de servicios en donde se genera el proyecto a través de cuatro fases: procesamiento del archivo XML, sección de estructura base, archivos de configuración del proyecto y construcción de archivos del proyecto. Así mismo, se realizó un caso de estudio para generar una aplicación para las plataformas MACOSTM y WINDOWSPHONE® usado Atila. Se utilizaron los patrones de diseño *SplashScreen, Login, Dashboard, Menu, List* y *Video.* Finalmente, tienen la intención de continuar con la generación de aplicaciones orientadas a la educación y mejorar la compatibilidad con otros dispositivos.

El reconocimiento de voz es una tecnología que permite identificar y traducir un lenguaje hablado a datos que son interpretados por computadoras. Hoy en día los dispositivos móviles cuentan con un asistente de voz como Siri y Google Voice, esta tecnología brinda a los usuarios la funcionalidad de ejecutar comandos en los dispositivos, sin embargo, la ejecución de comandos por medio de voz representa una vulnerabilidad porque no se cuenta con un método de autenticación. Joon et al. [23] analizaron las vulnerabilidades de los sistemas de reconocimiento de voz en los dispositivos móviles. También, se presentó el modelo de ataque *Toilet-time* y se realizaron pruebas en diferentes escenarios utilizando la herramienta *BadVoice,* la cual simula las manos libres de un teléfono celular para ejecutar comandos del asistente de voz. El primer escenario fue *passive attack,* consiste en ejecutar comandos directamente como mandar un mensaje de texto, realizar una llamada,

hacer un *post* en la red social que se sincroniza con el celular, por mencionar algunos. El segundo escenario fue *active attack,* este ataque recupera información por medio del asistente de voz, por ejemplo, lectura automática de mensajes de texto de algún contacto. Al final de las pruebas se comprobó que las ejecuciones de los dos tipos de escenarios se realizaron en menos de 21 segundos y se concluye que existe un gran campo para trabajar en las vulnerabilidades de los sistemas de reconocimiento de voz.

Modak et al. [20] realizaron una aplicación de escritorio que implementa una interfaz de lenguaje natural para la generación de páginas WEB, el sistema está compuesto por cinco etapas: 1) recibir las palabras, 2) convertir las palabras a texto, 3) procesamiento de texto, 4) extracción de conocimiento y 5) generación de código. Los resultados de la implementación son prometedores, sin embargo, el reconocimiento de voz y la conversión de palabras a texto depende mucho del entorno, ruido en el entorno, calidad del micrófono, por mencionar algunos. Por último, se tiene la intención de implementar nuevas tecnologías como PHP (*Personal Home Page,* Página Personal) y *JavaScript* para generar páginas dinámicas, implementar Inteligencia Artificial para sugerir *templates* y así optimizar el tiempo de la generación de la página WEB.

Para realizar el diseño de una aplicación es importante conocer a los usuarios, las similitudes entre ellos nos permiten identificar los perfiles de los usuarios, sus necesidades y problemas en un contexto particular, sin tomar en cuenta las suposiciones y teorías para identificar alguna propuesta. Vittone et al. [23] mencionaron que las personas son una herramienta que se utiliza para definir modelos o arquetipos de usuarios, tomando como referencia sus necesidades y objetivos. Para conocer a los usuarios se requiere realizar una investigación que identifique cuáles son los patrones de comportamiento y pensamiento que tienen en común los usuarios. Por otra parte, Vittone et al. [23] recomendaron el uso de *wireframes* que son una representación muy simplificada de una pantalla individual, que sirven para tener una idea inicial de la organización de los elementos que contendrá, identificando y separando aquellos informativos de los interactivos. Para

concluir, hay diferentes maneras de utilizar *wireframes,* a mano, por medio de plantillas (*stencils)* y utilizar programas de *software* como Balsamiq, Omnigraffe y Axure.

El proceso de implementar *software* basado en un *mockup* de una interfaz gráfica de usuario que se crea por un diseñador significa una gran responsabilidad para los desarrolladores, representa dedicar demasiado tiempo en la interfaz gráfica en lugar de asegurar la funcionalidad y lógica del *software*. Beltramelli [24] presentó un software basado en redes neuronales convolucionales que realiza la generación de código para plataformas móviles y WEB, a partir de una imagen y un modelo de entrenamiento previo, su nombre es pix2code. Además, se presentó un modelo de entrenamiento para demostrar una implementación de pix2code. Los resultados de la implementación que se realizaron se consideraron relativos al conjunto de datos que se proporcionó para el entrenamiento del modelo. Se concluyó que la calidad de la generación de código se incrementa drásticamente cuando se realiza el entrenamiento con un modelo más grande.

La tecnología de reconocimiento de voz hizo posible que la computadora analizara y reconociera comandos de lenguajes naturales. Patil et al. [25] utilizaron LABView para la interpretación de señales de voz que se envían a través de *bluetooth* a un robot que realiza movimientos. La arquitectura del sistema se compone por dos unidades: de control y la del robot. La unidad de control captura la voz mediante un micrófono, se realiza el procesamiento de voz en LABView, por último, se envía a la unidad del robot a través del módulo *bluetooth* en modo maestro*.* La unidad del robot recibe los comandos por medio del módulo bluetooth en modo esclavo, los comandos son procesados por una tarjeta arduino y manda las señales pertinentes para que el motor de movimientos realice los giros correspondientes. Se encontró que usar la unidad de procesamiento de voz hizo que el sistema trabajara con mayor facilidad y exactitud. Además, la implementación de LABView hizo que el sistema fuera más barato y confiable.

En el campo de la manipulación de genes es necesario hacer más de una micro manipulación, especialmente para realizar operaciones con mayor complejidad. Terada et al. [26] desarrollaron un robot que se utiliza con reconocimiento de voz. Actualmente, el sistema del robot consiste de una coordenada rectangular que son dirigidas por aceite hidráulico, tienen dispositivos que realizan los movimientos con la mano izquierda y derecha, sin embargo, cuando un operador necesita realizar más de dos operaciones un tercer mando realiza los movimientos a través de reconocimiento de voz. Como resultados de las pruebas que realizaron al mando que se utiliza por medio de la voz encontraron que sí se controla de manera suave. Como trabajo futuro optimizarán el reconocimiento del orden de las palabras, por último, incrementarán el tiempo de respuesta de la ejecución de los movimientos por medio de comandos de voz.

Hoy en día, el desarrollo de dispositivos móviles crece rápidamente, especialmente los dispositivos Android. El sistema operativo Android necesita una alternativa para facilitar las operaciones cuando el usuario maneja un automóvil o cuando realiza otras actividades en donde el usuario utiliza sus manos. Sidiq et al. [27] presentaron una aplicación Android, su nombre es Vomma, esta aplicación se basó en reconocimiento de voz y ejecuta las aplicaciones que se instalan en el dispositivo. El proceso de la aplicación es bastante sencillo y consta de tres pasos: 1) vomma muestra una lista de aplicaciones instaladas en el dispositivo, después 2) el usuario inicia el reconocimiento de voz, para ejecutar los comandos preestablecidos, y finalmente, 3) vomma ejecuta la aplicación correspondiente.

Vomma tiene una lista de comandos preestablecidos para navegar y ejecutar acciones en la aplicación, tales como: up, down, stop, start, entre otros. Finalmente, realizaron una serie de pruebas para verificar el funcionamiento de la aplicación y concluyeron que hay varios factores que afectan el reconocimiento de voz, por ejemplo, el ruido en el entorno, la distancia entre el micrófono y el usuario, el ángulo del micrófono y la pronunciación.

La ingeniería de software para los sistemas WEB es un proceso complejo en donde la reutilización y productividad son aspectos deseables, además, involucra muchos aspectos como modelar la interfaz del usuario. A pesar de los avances recientes en la ingeniería de interfaz de usuario el proceso es costoso, laborioso y propenso a errores. Costa et al. [28] aplicaron un MDD (*Model-Driven Development,* Desarrollo Guiado por Modelo) en una investigación de interfaz de usuario, su nombre es UI Stereotype (*User Interface,* Interfaz de Usuario), para optimizar la construcción de portales WEB. Por otra parte, se propuso la implementación de un portal WEB utilizando UI Stereotype para generar diferentes portales WEB con la misma intención (apariencia y comportamiento) a partir de un mismo conjunto de meta-modelos. Como futuro trabajo se propuso migrar el concepto de UI Stereotype para IFML (*Interaction Flow Modeling Language,* Lenguaje de Modelado de Flujo de Interacción).

Hoy en día el campo de la ingeniería computacional involucra diferentes direcciones que afectan a las características del software. El procesamiento de imágenes está creciendo y evolucionando rápidamente, por esta razón, el software de análisis de imágenes requiere de organización de funciones, interfaz de usuario, administración de datos para el procesamiento de imágenes, entre otros. En [27] se propuso un esquema para el desarrollo de software de análisis y procesamiento de imágenes basado en la generación de tablas de sinónimos (*thesaurus tables*) en el *kernel* del intérprete. El esquema está basado en la combinación de características de bibliotecas dinámicas y en un intérprete que tienen un conjunto de funciones para el procesamiento de imágenes. Por otra parte, se realizó un caso de estudio para el análisis de imágenes histológicas en el área de medicina, se utilizó Lua como lenguaje intérprete, y para la interfaz de usuario, se hizo una combinación de Highgui y Qt. El desarrollo que se obtuvo a partir del caso de estudio consta de dos fases: 1) la primera fase consiste en evaluar y analizar la imagen para detectar características como la calidad, posteriormente, se realiza la 2) fase de generación de *scripts* que tiene como función la selección de objetos deseados, esta función se utiliza para determinar los objetos histológicos en una imagen. Para finalizar, se concluyó que la separación como un componente de software y parametrización del

kernel para la generación de *scripts* facilita la reutilización del software para resolver diferentes problemas en el análisis de imágenes.

Las aplicaciones móviles son aceptadas y distribuidas rápidamente por millones de personas alrededor del mundo. En los últimos años surgieron aplicaciones que almacenan datos críticos para solucionar problemas orientados a la salud. Sin embargo, en el proceso de desarrollo de las aplicaciones orientadas a la salud se encuentran muchas semejanzas, las cuales implican que el proceso sea repetitivo y que en algunos casos los *bugs* sean los mismos. Paschou et al. [30] realizaron un generador de aplicaciones orientados a la salud para el sistema operativo Android que se utiliza a través de una interfaz WEB y no es necesario tener conocimientos técnicos y/o de programación para utilizarlo. El generador se creó para que gente especializada en el área de la salud tenga la posibilidad de diseñar aplicaciones que son útiles para sus pacientes, con el fin de que los pacientes almacenen información importante que requiere una supervisión especial por parte del médico. Cabe mencionar que el proceso para generar una aplicación es muy sencillo y consta de cinco pasos: 1) el especialista de la salud especifica las características de la aplicación como los tipos de campos que tendrá la aplicación, tareas que realizará la aplicación, entre otros. 2) se especifica el diseño de los formularios con respecto a los campos creados en el paso anterior, 3) se procede a generar la aplicación con todas las características descritas en los pasos anteriores. Los tres pasos previos en algunos casos son repetitivos, depende del usuario. 4) los pacientes son notificados para descargarla. Por último, 5) la aplicación se inicializa en el dispositivo móvil del paciente. Por otra parte, se realizó una comparación entre aplicaciones que se crearon con el generador propuesto y aplicaciones existentes orientadas a la salud como Care@HOME. Los resultados fueron satisfactorios ya que las aplicaciones eran muy similares, sin embargo, la utilización del generador optimiza tiempo, recurso humano y previene errores constantes.
En los últimos años, el incremento de contenido de audio y el procesamiento de audio se convirtió en una necesidad para resolver tareas cotidianas. En el campo aeroespacial los comandos de voz están siendo adoptados rápidamente gracias a que son más efectivos que ejecutar simples comandos de computadora, ayudan a

los usuarios a ejecutar comandos sin necesidad de utilizar sus manos. Tabibian [31] propuso un marco de trabajo de reconocimiento de comandos de voz para el área aeroespacial, el marco de trabajo detecta los comandos de voz en un tiempo aceptable, además, el índice de detección de falsa alarma es muy bajo. El marco de trabajo propuesto funciona en dos partes fundamentales: 1) localización de palabras y 2) decodificador de comandos de voz. La salida de la primera parte son palabras discriminadas que se inyectan al decodificador como parámetros. La segunda parte cuenta con un decodificador que está basado en un modelo de lenguaje basado en reglas, la salida del decodificador da como resultado los comandos de voz. Por otra parte, el marco de trabajo incluye dos fases: la fase entrenamiento y la de prueba. La fase de entrenamiento contiene tres sub-fases; extracción, pos procesamiento y entrenamiento del modelo. La fase de pruebas incluye el pre procesamiento, extracción, post procesamiento, localización de palabras clave y la fase de modelo de lenguaje basado en reglas. Finalmente, se realizaron una serie de pruebas del marco de trabajo con diferentes métodos de reconocimiento de voz, los resultados fueron satisfactorios, sin embargo, la pronunciación y el ruido en el ambiente son factores que afectan el funcionamiento idóneo del marco de trabajo propuesto.

Estudios señalan la importancia de la interacción en el campo de visualizadores de información para realizar una buena visualización de datos. Las investigaciones de interacción en visualizadores de información fomentan el uso de interfaces no convencionales además del teclado y *mouse,* por ejemplo, los comandos de voz. Furtado et al. [32] utilizaron Coruja Software para realizar una serie de pruebas en un software que muestra un plano cartesiano en tres dimensiones para evaluar los aspectos de diseño, desarrollo e interacción con una interfaz de reconocimiento de comandos de voz. La primera parte del conjunto de pruebas que realizaron fue identificar los perfiles de los usuarios, después, se les asignó una serie de tareas y se tomó el tiempo en el que se realizaron, por último, se les aplicó un cuestionario para identificar la dificultad de cada escenario. Cada una de las tareas se midió con un estudio psicométrico de índice de la carga mental. Los resultados de las pruebas fueron que a la mayoría de los usuarios se les dificultó más la utilización del software

por comandos de voz en comparación con el teclado y *mouse,* el factor más relevante fue la frustración porque identifica la incomodidad del usuario al realizar la tarea asignada. Las pruebas en relación al tiempo demostraron que el *mouse* es eficiente, en cambio los comandos de voz son efectivos.

La generación de interfaces de usuario es una parte esencial para el proceso de desarrollo de software, el cual está en constante evolución. En los últimos años, enfoques y modelos de desarrollo demuestran desventajas e implica que el proceso de desarrollo se vuelva más complicado, sin embargo, existen técnicas de Inteligencia Artificial que son eficientes para contrarrestar la complejidad en el desarrollo de interfaces de usuario. Sánchez et al. [18] presentaron un componente de software para la generación de interfaces de usuario para aplicaciones móviles usando reconocimiento de patrones, procesamiento de imágenes y técnicas de redes neuronales. Por otra parte, el proceso para la generación de interfaces de usuario para aplicaciones móviles tiene tres fases: 1) análisis de imagen, 2) configuración y 3) generación de código fuente. La fase de análisis de imagen tiene tres etapas: la primera etapa recibe las imágenes, las imágenes admitidas para el análisis de imágenes son modelos abstractos de vistas de datos, cualquier herramienta de dibujo genera ADV (*Abstract Data Views,* Vistas de Datos Abstractos*).* En la segunda etapa se evalúa que las imágenes son interfaces de usuario correctas, los elementos que contiene una imagen correcta son representaciones de patrones de diseño de interfaz de usuario. En la tercera etapa se identifican cada uno de los elementos de la interfaz de usuario. Por otra parte, en la fase de configuración se elige el tipo de aplicación (WEB o móvil) y la configuración general en donde se especifica el título principal, lenguaje, por mencionar algunos. En la fase de generación de código se crea un archivo XML que posteriormente se traduce en la tecnología especificada, dependiendo la plataforma. Finalmente, se realizó un caso de estudio para la generación de un *login* para una aplicación Android, se concluyó que es posible la generación de interfaces de usuario a partir de una imagen, sin embargo, es necesario seguir realizando pruebas e incrementar el número de patrones de diseño de interfaz de usuario,

también, se desea implementar la funcionalidad de generar interfaces de usuario a partir de una foto o un *mockup*.

2.2 Análisis comparativo de trabajos relacionados

A continuación, la tabla 2.1 muestra un análisis comparativo de los factores más importantes de los artículos de investigación relacionados.

Tabla 2.1 Análisis comparativo de trabajos relacionados.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
Erić et al.[21]	Identificar una arquitectura óptima para la utilización de herramientas específicas de reconocimiento de voz en la domótica.	Tres arquitecturas de sistemas de domótica con implementación de reconocimiento de voz.	Jasper platform, de Google speech API, Alexa Voice Service y Bing speech API.	La identificación de las arquitecturas mediante una comparativa del uso de las herramientas del reconocimiento de voz específicas que determinan la arquitectura óptima para el uso de sistemas de domótica.	Finalizado
Cortes et al. [18]	Se identificó la existencia de generadores de aplicaciones que no son flexibles para el desarrollo de	Generador basado en patrones de diseño de interfaz gráfica de aplicaciones educativas.	XML	La obtención de proyectos nativos según la plataforma especificada.	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	aplicaciones en diferentes plataformas.				
Lee et al.[15]	Se encontró la existencia de vulnerabilidades en los asistentes de voz por falta de autenticación en los dispositivos móviles.	Un modelo de ataque que lleva por nombre *Toilet-time.*	BadVoice, Siri y Google Voice.	Se comprobó que el tiempo de ejecución de las pruebas de ataque fue crítico.	Finalizado
Modak et al. [20]	Se identificó que para el desarrollo y diseño de páginas WEB se requieren conocimientos y habilidades de un experto.	Generador de páginas WEB mediante el reconocimiento de voz.	Google Speech API, HTML5 (*Hypertext Markup Language revision five,* Lenguaje de Marcado de Hipertexto versión cinco) y	Los resultados son buenos pero el reconocimiento de voz y la conversión de palabras a texto dependen del entorno.	Finalizado

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
			CSS (*Cascading Style Sheets,* Hojas de Estilo en Cascada).		
Vittone et al. [23]	Se identificó el proceso incorrecto de modelado que se realiza al crear una interfaz gráfica de usuario.	Herramientas para la creación de *wireframes.*	No se especifican.	Se obtuvo un mejor proceso de modelado de interfaces de usuario.	Finalizado.
Beltramelli et al. [24]	Se encontró que se dedicaba demasiado tiempo en la codificación de una interfaz gráfica a partir de un *mockup*.	Modelo de entrenamiento para realizar una implementación en el *software* pix2code.	Pix2code.	Los resultados fueron relativos al conjunto de datos que se utilizaron en la etapa de entrenamiento.	Finalizado.
Patil et al.[25]	Se identificó que para utilizar una interfaz de reconocimiento de voz de manera óptima	Modelos de ejecución de LABView.	LABView.	Los resultados fueron buenos pero las condiciones del entorno influyen finalmente en el resultado.	Finalizado.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	depende de las condiciones del entorno.				
Terada et al. [26]	Se encontró la falta de control de multi-micro manipulaciones en la genética.	Robot que realiza manipulaciones por medio de reconocimiento de voz.	No se mencionan.	Se logró una manipulación suave, sin embargo, se optimizará el reconocimiento del orden de las palabras y tiempo de respuesta de los movimientos a partir de los comandos de voz.	Finalizado.
Sidiq et al. [27]	Se detectó la falta de control para ejecutar aplicaciones por medio de comandos de voz en dispositivos Android.	La aplicación Vomma se basó en reconocimiento de voz para ejecutar aplicaciones instaladas en el dispositivo.	Sistema operativo Android y la aplicación Vomma.	Los resultados fueron favorables, sin embargo, se concluyó que hay varios factores que afectan el reconocimiento de voz, por ejemplo, el ruido en el entorno, la distancia entre el micrófono y el usuario, el ángulo del micrófono y la pronunciación.	Finalizado.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
Costa et al. [28]	Se encontró que el proceso de construcción de portales WEB es costoso, repetitivo, laborioso y tedioso.	Modelo basado en la apariencia y comportamiento de los componentes, su nombre es UI Stereotype.	UML (*Unified Model Language,* Lenguage Unificado de Modelado) e IFML.	La optimización de tiempo en la construcción de portales WEB similares fue uno de los resultados más importantes.	Finalizado.
Nedzved et al. [29]	Se identificó que no separar y parametrizar el núcleo de un *software* implica más tiempo en el proceso de construcción y codificación.	Esquema para el desarrollo de software de análisis y procesamiento de imágenes basado en la generación de tablas de sinónimos (*thesaurus tables*) en el *kernel* del intérprete.	Lua, Highgui y Qt.	La implementación del esquema dió como resultado el desarrollo de un *software* para el análisis de imágenes histológicas.	Finalizado.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
Paschou et al. [30]	Se identificó que el proceso de desarrollo de aplicaciones orientadas a la salud es repetitivo, costoso y tedioso.	Generador de aplicaciones Android orientadas a la salud.	Android, HTML5, CSS3, JavaScript, Apache, MySQL, PHP y jQuery.	Los resultados fueron satisfactorios ya que las aplicaciones que se crearon con el generador propuesto son muy parecidas en funcionalidad y apariencia.	Finalizado.
Tabibian. [31]	Se detectó que los comandos de voz son más útiles que ejecutar comandos de computadora en el campo aeroespacial.	Marco de trabajo de reconocimiento de voz para aplicaciones aeroespaciales.	No se mencionan.	Los resultados de las pruebas fueron los esperados, sin embargo, factores como la pronunciación y el ruido en el ambiente afectan el funcionamiento correcto del marco de trabajo.	Finalizado.
Furtado et al. [32]	Se encontró que la implementación de nuevas interfaces para la interacción con visualizadores de información como los	Un conjunto de pruebas que identifica los problemas que implica usar un *software* por medio de comandos de	Coruja software.	Los resultados de las pruebas fueron que a la mayoría de los usuarios se les dificultó más la utilización del software por comandos de voz en comparación con el teclado y *mouse.*	Finalizado.

Artículo	Problema	Contribución	Tecnologías	Resultado	Estado
	comandos de voz se tiene que mejorar.	voz y medios tradicionales como el teclado y *mouse.*			
Sánchez et al. [18]	Se identificó que la implementación de técnicas de procesamiento de imágenes y de inteligencia artificial disminuyen la complejidad para la generación de interfaces de usuario.	Componente de software para la generación de interfaces de usuario a partir de una imagen.	Android, IOS, BlackBerry, HTML5, XML JQuery, Xencha y MATLAB.	Los resultados son parciales y relativos a lo que se espera del software, ya que es necesario realizar pruebas con más casos de estudio.	Finalizado.

Después de analizar los artículos de investigación citados, se identificó que el incremento y desarrollo de tecnologías de reconocimiento de voz dio como resultado herramientas eficientes que se implementaron por diferentes áreas de investigación en distintos ámbitos. Sin embargo, la mayoría de investigaciones coincidieron que la eficiencia de las interfaces de reconocimiento de voz depende de factores como: 1) el ruido en el ambiente, 2) la calidad del micrófono, 3) la pronunciación, 4) distancia y ángulo entre el usuario y el micrófono, por mencionar los factores más importantes. Por otra parte, los proyectos de investigación demostraron técnicas, métodos y tecnologías que son útiles para realizar reconocimiento de voz de la manera más eficiente. Finalmente, se concluyó que los artículos de investigación son de gran importancia para considerar los métodos y tecnologías que se utilizarán para desarrollar el componente de software de este anteproyecto que consiste en generar interfaces de usuario de aplicaciones móviles para el ámbito de cuidado de la salud mediante comandos de voz.

2.3 Solución propuesta

A continuación, se presenta la propuesta de solución para el desarrollo de la investigación, la cual se considera: Plataforma de agente conversacional, Conjunto de herramientas y marco de trabajo para el front-end, lenguaje de programación del lado del back-end, y, por último, el editor de código. Esta solución sustenta un desarrollo robusto, gratuito y mantenible. En la tabla 2.2 se muestra la propuesta de la solución que se seleccionó.

Tabla 2.2 Análisis de la propuesta de solución

Aspecto	Solución
Plataforma de agente conversacional	Dialogflow
Front-end	HTML5, CSS, JavaScript y AngularJs
Back-end	PHP
Editor de código	Visual Studio Code
Metodología para el desarrollo de software	SCRUM

2.4 Justificación de la solución seleccionada

Esta decisión se basa en el análisis de la relación costo beneficio de la combinación de tecnologías de software, y, sobre todo, la más adecuada para este proyecto. Se eligió la plataforma de Dialogflow para realizar la configuración del agente conversacional, esta plataforma a diferencia de otras, es la más económica relativamente a los servicios necesarios que ofrece, cuenta con soporte por diferentes medios y tiene la mejor accesibilidad al marco de trabajo seleccionado. Por otra parte, el lenguaje de programación seleccionado fue PHP por su flexibilidad de desarrollo, por ser gratuito y por la facilidad de crear aplicaciones robustas. Además, como Editor de código se eligió Visual Studio Code por sus dos características importantes: es ligero y extensible. De igual forma se trata de un editor de código ampliamente utilizado por desarrolladores. Finalmente, por el tiempo de desarrollo y al hecho de no pasar por alto la documentación, se seleccionó a SCRUM como metodología, ya que el enfoque ágil permite la inclusión de los artefactos que la ingeniería de software recomienda para sustentar los requerimientos y características de la solución que se plantea, enfocándose en la entrega del producto de software a desarrollar de forma iterativa e incremental teniendo en cuenta la documentación.

Capítulo III. Aplicación de la metodología

En este capítulo se describen cada una de las actividades de implementación para que se logre el objetivo de este trabajo. Este capítulo se divide en tres secciones acorde a las actividades de análisis, diseño y desarrollo.

3.1 Análisis

A continuación, se describe el proceso de análisis de este proyecto de tesis, los puntos que se presentan son tres: 1) análisis de los requerimientos funcionales, 2) el flujo de trabajo para especificar las características de la aplicación móvil y 3) Análisis de PDIU de aplicaciones móviles médicas para el uso común y profesional.

3.1.1 Análisis de requerimientos funcionales

Se identificaron los requerimientos funcionales para el desarrollo del generador de aplicaciones móviles por medio de reconocimiento de voz, a continuación, se describe cada requerimiento funcional:

1) **Especificar dispositivo:** Para este trabajo siempre será móvil ya que se generaron aplicaciones para dispositivos móviles.
2) **Especificar plataforma:** Android™ y iOS® son las plataformas compatible con las aplicaciones móviles que se generaron.
3) **Especificar dominio:** Las aplicaciones se enfocan al área del cuidado de la salud, por lo tanto, el dominio médico es el indicado.
4) **Especificar Contenido:** Son los PDIU que conforman la aplicación que se generó.

La Figura 3.1 ilustra un diagrama de casos de uso que representa los requerimientos funcionales listados anteriormente.

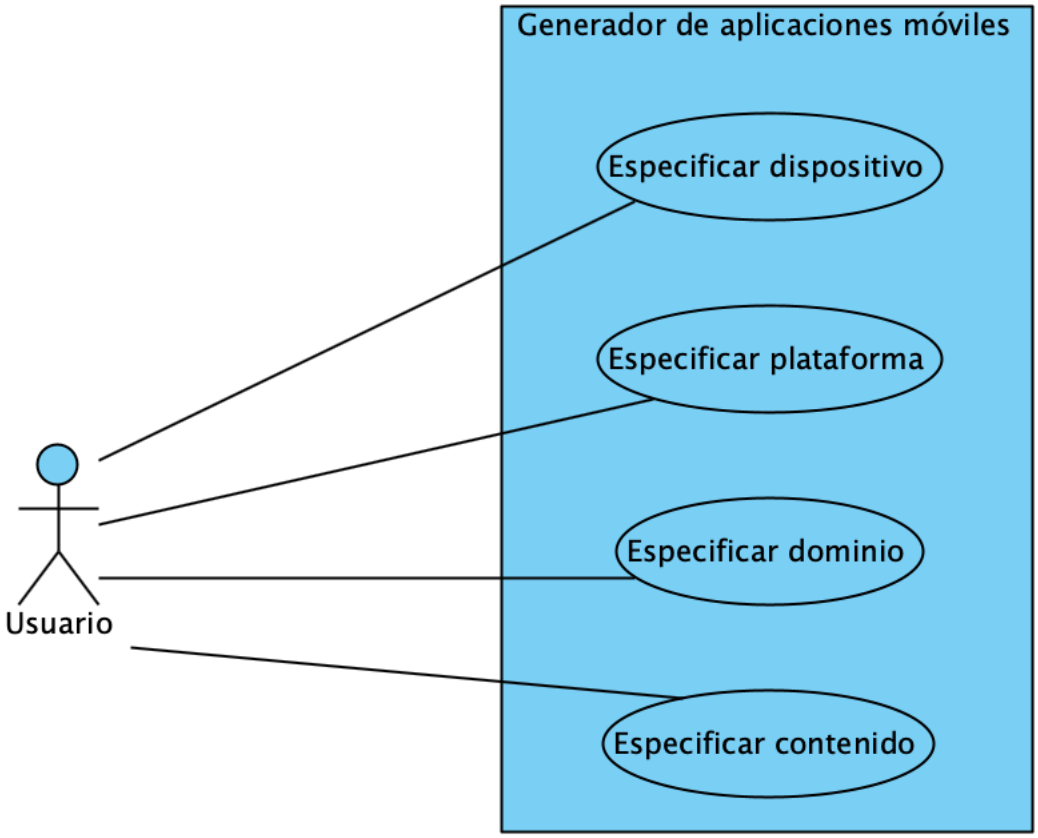

Figura 3.1 Requerimientos funcionales del generador de aplicaciones móviles

3.1.2 Flujo de trabajo para la generación de una aplicación móvil

A partir del análisis de requerimientos funcionales, se analizó un flujo de trabajo para que el generador de aplicaciones móviles cumpla con cada uno de ellos y así el usuario describa las características de la aplicación móvil que quiera generar. A continuación, se enlistan ocho pasos del flujo de trabajo del sistema para generar una aplicación móvil:

1. El sistema muestra pantalla de inicio
2. El usuario inicia la construcción
3. El sistema da la bienvenida
4. El sistema escucha las instrucciones
5. El usuario dicta las instrucciones
6. El sistema determina si en la intención del usuario se identifican los elementos necesarios para generar una aplicación móvil. Si el sistema identifica que la intención del usuario está incompleta entonces pide los elementos faltantes, de lo contrario, el sistema procede a la siguiente etapa.
7. El sistema construye la aplicación móvil con las características que el usuario especificó.
8. El sistema descarga la aplicación.
9. El sistema regresa a la pantalla de inicio.

La Figura 3.2 muestra un diagrama de actividades que ilustra los pasos del flujo de trabajo y la interacción del generador y el usuario para construir una aplicación móvil.

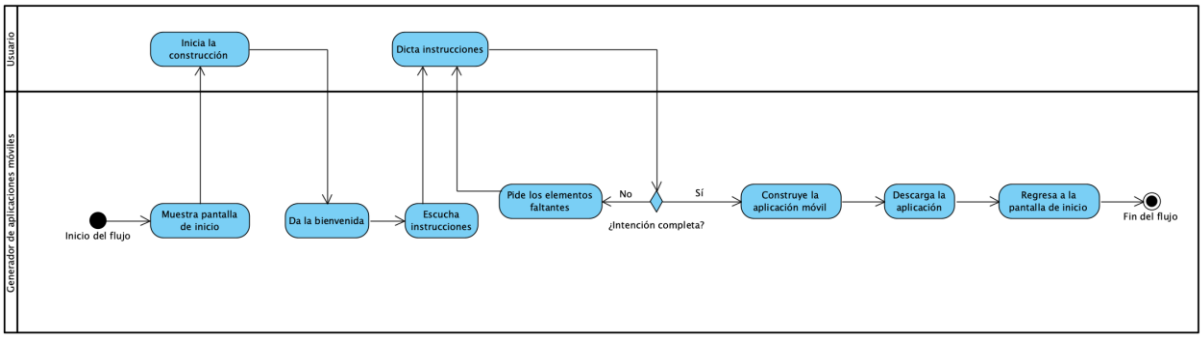

Figura 3.2 Flujo de trabajo del generador de aplicaciones móviles

3.1.3 Análisis de PDIU de aplicaciones móviles médicas

Para determinar los PDIU más concurrentes en las aplicaciones médicas se realizó un análisis que clasifica las aplicaciones por su uso: profesional y común. En la Tabla 3.1 se muestra los tipos de aplicaciones de acuerdo a la clasificación propuesta.

Tabla 3.1 Aplicaciones médicas según su uso

Categoría	Tipo de aplicaciones	Principales usuarios
Uso profesional	• Diagnóstico y tratamiento • Enciclopedia médica • Control de citas médicas • Administración de resultados de laboratorio • Cálculos médicos • Comunicación clínica • Sistemas de información hospitalaria • Entrenamiento médico • Farmacia • Valoraciones	• Profesionales de la salud • Médicos • Estudiantes del área
Uso común	• Localización de entidades médicas y profesionales de la salud • Calculadora médica • Administración de enfermedades crónicas • Recordatorio de medicación • Guía de primeros auxilios	• Pacientes • Familiares de los pacientes

Se analizó un conjunto de aplicaciones de uso profesional para identificar los PDIU de acuerdo a su interfaz gráfica. En la Tabla 3.2 se muestran los PDIU identificados por tipo de aplicación médica. Por otra parte, la Figura 3.3 ilustra los PDIU Feedback, Navigation Tabs y Vertical Dropdown Menu. En la Figura 3.4 se muestran los PDIU Gallery, Datalist y Search. Los PDIU Form, Navigation Tabs y Detail View se ilustran en la Figura 3.5 y en la Figura 3.6 se muestran los PDIU Stats, Dashboard y Login.

Tabla 3.2 PDIU por tipo de aplicación de uso profesional

Tipo de aplicación	PDIU
Diagnóstico y tratamiento	Feedback, Datalist, Gallery, Video, Vertical Dropdown Menu, Map, Menu, Navigation Tabs, Chat
Enciclopedia médica	Datalist, Dashboard, Form, Gallery, Panel, Search, Menu
Control de citas médicas	Datalist, Detail View, Form, Menu, Navigation Tabs, Search, Pagination, Panel
Administración de resultados de laboratorio	Feedback, Form, Dashboard, Datalist, Menu, Panel, Splashscreen, Stats, Login

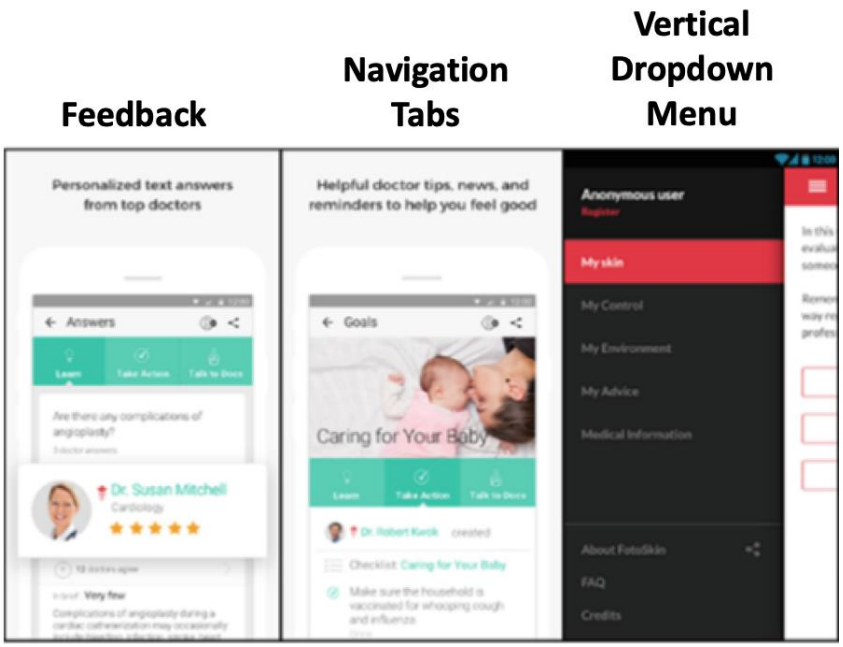

Figura 3.3 PDIU identificados en aplicaciones de uso profesional feedback, Navigation Tabs y Vertical Dropdown Menu

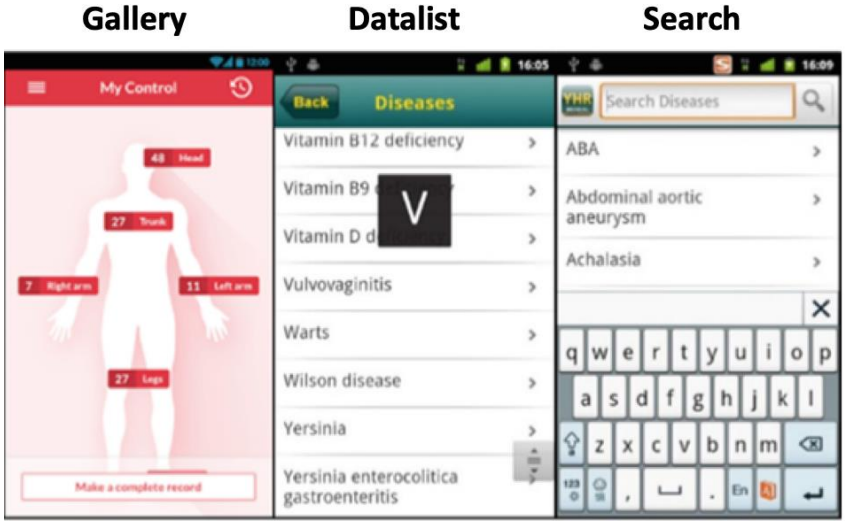

Figura 3.4 PDIU identificados en aplicaciones de uso profesional Gallery, Datalist y Search

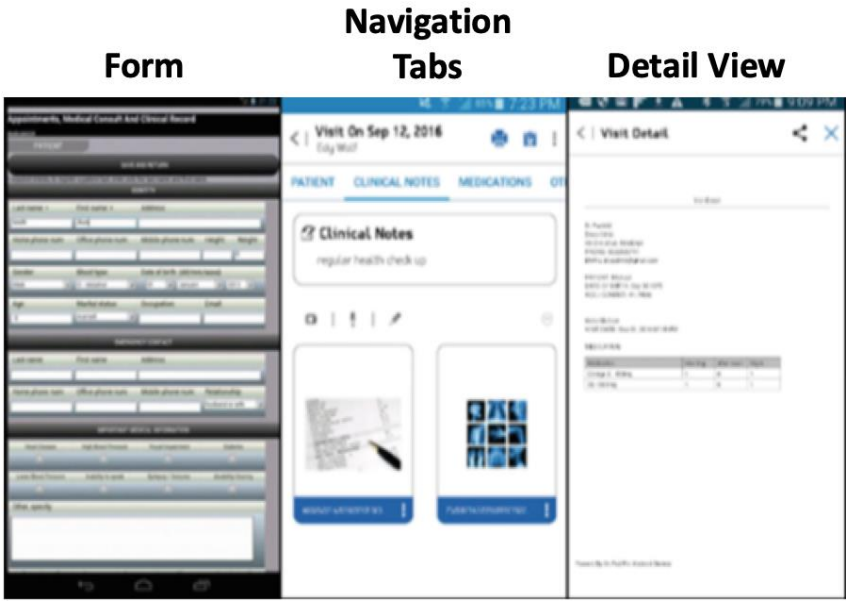

Figura 3.5 PDIU identificados en aplicaciones de uso profesional Form, Navigation Tabs y Detail View

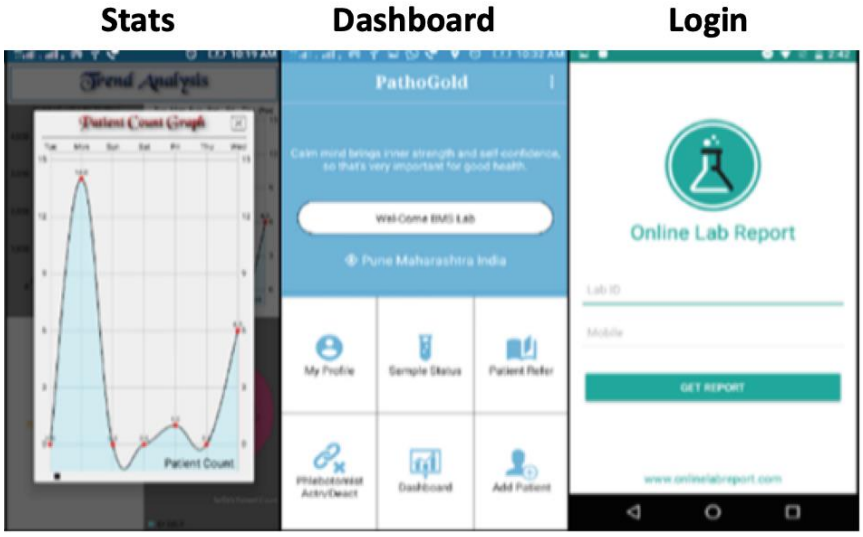

Figura 3.6 PDIU identificados en aplicaciones de uso profesional Stats, Dashboard y Login

Por otra parte, la Tabla 3.3 muestra los PDIU identificados en aplicaciones médicas de uso común. Además, la Figura 3.7 muestra los PDIU Feeback, Maps y Splashcreen que se encuentran en las interfaces gráficas de las aplicaciones médicas de uso común y en la Figura 3.8 los PDIU Login, Form y Dashboard.

Tabla 3.3 PDIU por tipo de aplicación de uso común

Tipo de aplicación	PDIU
Localización de entidades médicas y profesionales de la salud	Datalist, Feedback, Gallery, Maps, Menu, Navigation Tabs, Splashscreen, Vertical Dropdown Menu
Calculadora médica	Datalist, Dashboard, Form, Login, Menu, Panel, Splashscreen

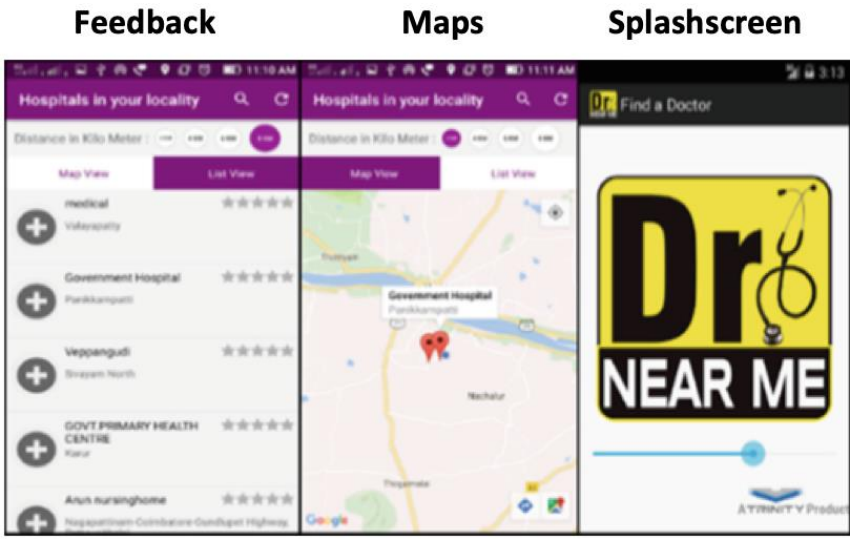

Figura 3.7 PDIU identificados en aplicaciones de uso común Feedback, Maps y Splashscreen

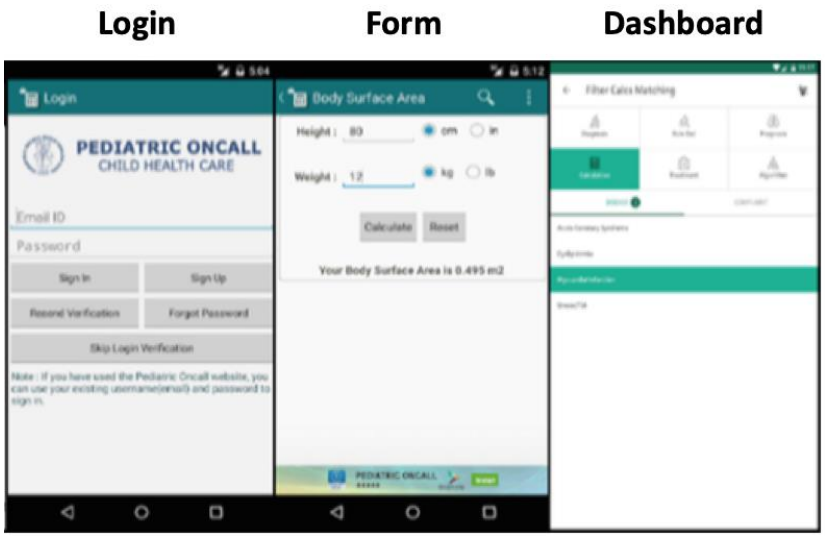

Figura 3.8 PDIU identificados en aplicaciones de uso común Login, Form y Dashboard

3.2 Diseño

En esta sección, se presentan las actividades que se relacionan al diseño del código intermedio para generar la aplicación móvil y el diseño de la interfaz gráfica de usuario.

3.2.1 Diseño de código intermedio XML

Con base al análisis de los requerimientos funcionales y PDIU identificados en aplicaciones médicas, se diseñó la estructura del código intermedio representado como un archivo XML, como se puede apreciar en la Figura 3.9.

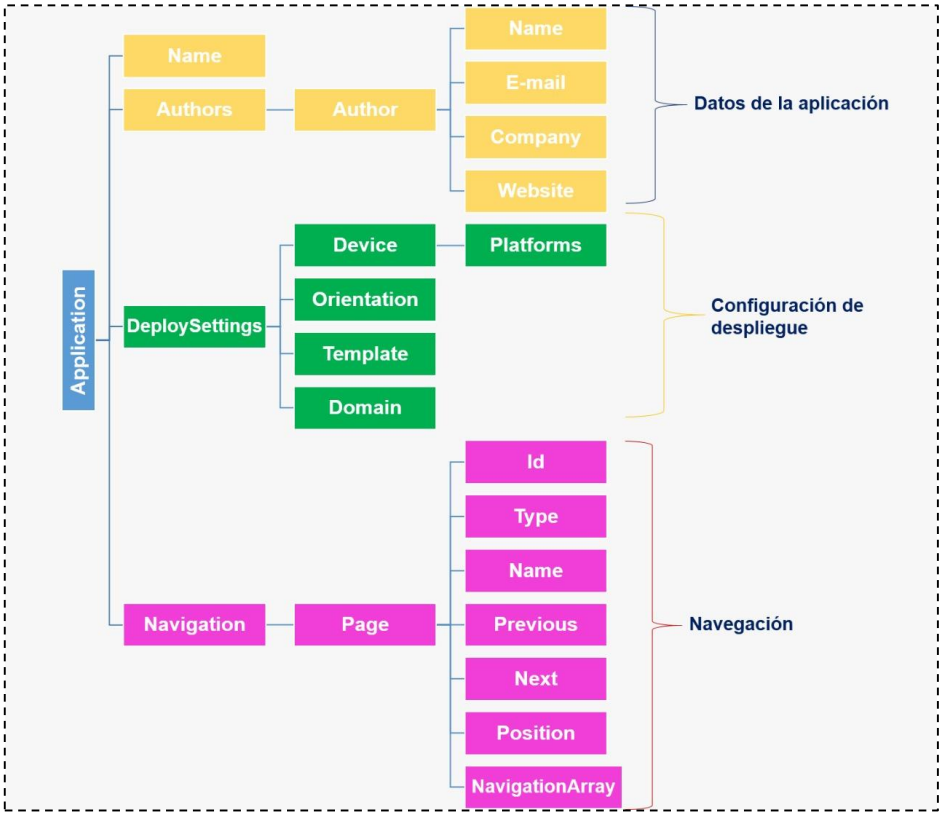

Figura 3.9 Estructura de código intermedio XML

El código intermedio está compuesto por tres partes importantes: 1) Datos de la aplicación, 2) Configuración de despliegue y 3) Navegación. La primera parte contiene información sobre la aplicación y datos de lo(s) autor(es), para cada autor se especifica nombre, correo electrónico, compañía y sitio WEB. En los datos de

configuración de despliegue se especifica el dispositivo y las plataformas correspondientes, orientación de visualización, plantilla de distribución y dominio, para este caso es el médico. Por último, en la información de navegación se especifica los datos de cada PDIU que conforma nuestra aplicación. Para cada PDIU se especifica un identificador único, tipo, nombre, elemento previo, elemento posterior, posición y un conjunto de PDIU a los cuales puede redirigir.

3.2.2 Diseño de la interfaz gráfica de usuario

Como se aprecia en la Figura 3.10, la interfaz de usuario se compone de tres elementos principales: 1) Pantalla principal, 2) Stack de elementos identificados y 3) Monitor de reconocimiento y síntesis de voz. A continuación, se describe cada uno de los elementos de la interfaz gráfica de usuario.

1. **Pantalla principal:** Tiene dos objetivos, el primero es mostrar el historial de la conversación entre el agente conversacional y el usuario como se puede apreciar en la Figura 3.11. El segundo objetivo es mostrar cuando la aplicación inicia el proceso de construcción, tal como se muestra en la Figura 3.12.
2. **Stack de elementos identificados:** La funcionalidad de éste elemento es mostrar al usuario los elementos que ya se identificaron por el sistema, así, el usuario tendrá una idea más clara de qué le falta especificar.
3. **Monitor de reconocimiento y síntesis de voz:** El sistema especifica cuando escucha instrucciones o cuando da indicaciones, entonces el usuario sabe en qué momento es su turno para dictar las instrucciones.

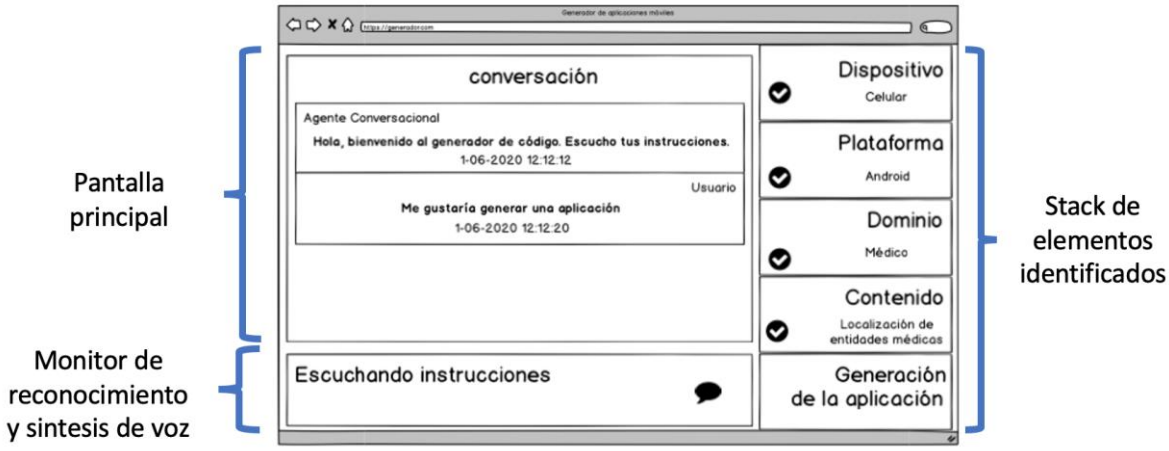

Figura 3.10 Elementos de IU del sistema

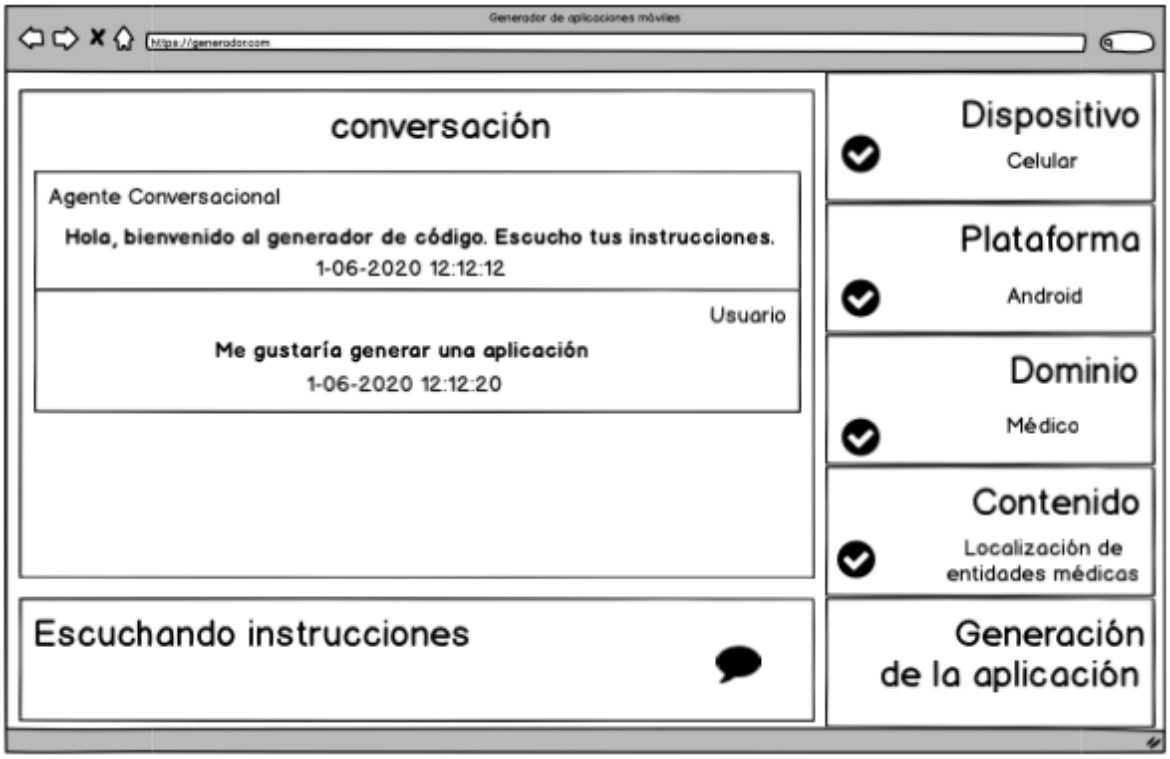

Figura 3.11 IU del historial de conversación

Figura 3.12 IU Generando la aplicación móvil

3.2.3 Arquitectura del generador de aplicaciones móviles

Como se visualiza en la Figura 3.13, la arquitectura del generador de aplicaciones móviles está compuesto por tres módulos principales:

1. **Administración de diálogo:** Consiste de una aplicación WEB del lado del cliente (Front-end) que se implementó con HTML 5, Bootstrap 4 y AngularJs un marco de trabajo para JavaScript. La principal tarea de este módulo es interactuar entre el usuario, la plataforma de PLN (Procesamiento de Lenguaje Natural) para identificar la intención del usuario y el módulo que genera el código nativo de aplicaciones móviles.
2. **Dialogflow:** Esta plataforma se utiliza para realizar el Procesamiento de Lenguaje Natural e identificar los elementos de la intención del usuario. La plataforma tiene un servicio WEB de tipo REST para consultar el agente conversacional, así, el administrador de diálogo determina si se identificaron todos los elementos para generar la aplicación.

3. **Módulo de generación de código:** El módulo se compone por dos aplicaciones WEB del lado del servidor (Back-end) que se implementaron en PHP, la primera genera el código intermedio XML y la segunda recibe el código intermedio y genera el código nativo de las aplicaciones móviles.

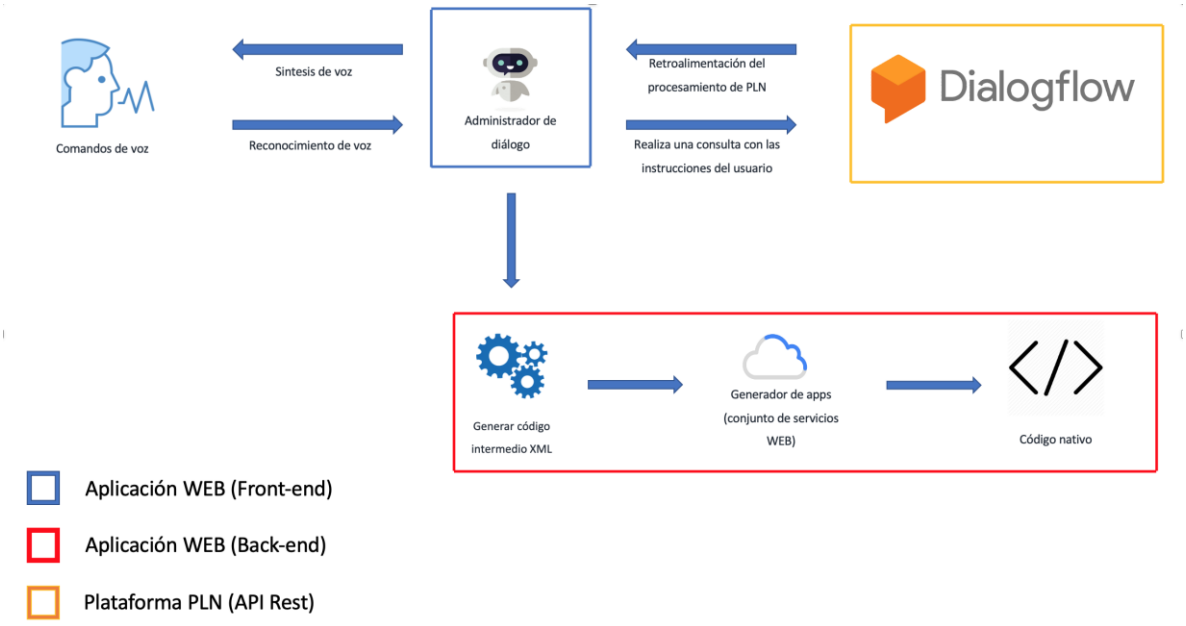

Figura 3.13 Arquitectura del generador de aplicaciones móviles

3.3 Desarrollo

En esta sección se describen todas las actividades de implementación para construir una aplicación WEB para generar aplicaciones para dispositivos móviles en el dominio médico por medio de comandos de voz.

3.3.1 Proceso automático de generación de interfaces de usuario en el ámbito médico

A continuación, se presenta el proceso automático de generación de interfaces de usuario en el ámbito médico. El proceso se divide en dos fases las cuales se componen de un conjunto de pasos en donde es posible implementar cualquier

lenguaje de programación, marcos de trabajo y técnicas de reconocimiento de voz. La interacción entre las dos fases se muestra en la Figura 3.14.

1. **Reconocimiento de voz.** En esta fase los diálogos del usuario se graban y se guardan como un archivo de audio para convertirlos en una secuencia de palabras la cual se analiza para identificar la intención del usuario. Tecnologías de PLN y semántica se utilizan para identificar los elementos de la intención del usuario, tales como tipo de dispositivo, plataforma móvil, dominio y contenido. Cuando no se identifican todos los elementos de la intención del usuario, el módulo de administración de diálogo se encarga de pedirlos al usuario para clarificar la intención del usuario. Este proceso es iterativo hasta que la intención del usuario está completa. Una vez que todos los elementos se identifican en la intención, un archivo XML es generado.
2. **Generación del proyecto.** Esta fase analiza el archivo XML generado en la fase previa. Después, el módulo de configuración de proyecto selecciona todos los elementos que contienen las aplicaciones móviles. Finalmente, se genera el código fuente de la aplicación móvil.

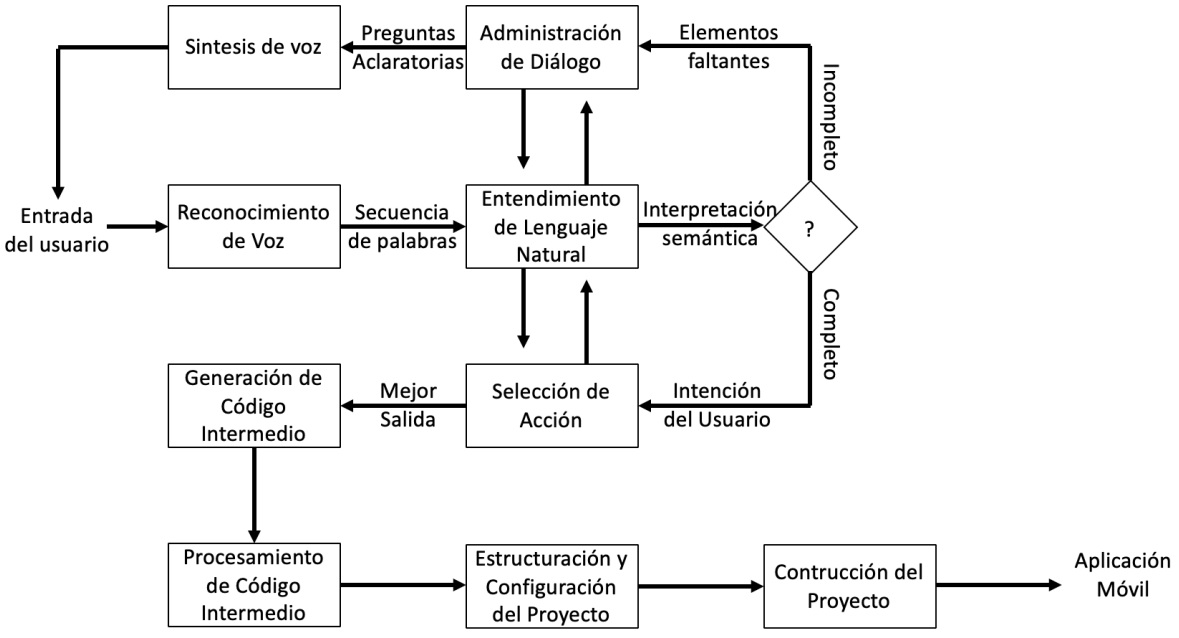

Figura 3.14 Descripción del proceso

Los principales módulos de las fases que se mencionan anteriormente son descritos a continuación:

- **Reconocimiento de voz:** En este módulo, el diálogo se procesa para analizar la secuencia de palabras y convertirlas a texto. El texto que se genera se procesa para determinar su semántica.
- **Procesamiento de Lenguaje Natural:** La semántica del texto se determina en este módulo para detectar las intenciones del usuario. Si la intención del usuario está completa entonces se envía al módulo de selección de acciones. En otro caso se notifica al módulo de administración de diálogo para que se pida al usuario con la ayuda de tecnología de síntesis de voz los elementos que no se identificaron.
- **Selección de acciones:** En este módulo se determina la acción dependiendo de la intención del usuario. Un archivo XML se genera para describir todas las características de la aplicación. Este documento se envía a la fase de generación de proyecto.
- **Administración de diálogo:** El objetivo de este módulo es pedir al usuario los elementos faltantes cuando el módulo de PLN no obtuvo todos los elementos para determinar la intención del usuario.
- **Síntesis de voz:** La principal función de este módulo es convertir las preguntas provenientes del módulo de administración de diálogo en un archivo de texto a un archivo de audio. Así, el módulo reproduce el archivo de audio para realizar una pregunta o dar una respuesta al usuario.
- **Generación de código intermedio:** En este módulo se realiza una validación semántica y sintáctica del archivo XML que describe la configuración del proyecto.
- **Configuración del proyecto:** Las características del proyecto se definen en este módulo. Esta configuración depende de la descripción del archivo XML.
- **Construcción del proyecto:** Finalmente, en este módulo la aplicación se construye de acuerdo a las características que se especifican durante todo el proceso.

3.3.2 Construcción de agente conversacional en Dialogflow

En esta sección se describe el proceso de construcción del agente conversacional en la plataforma de Dialogflow. El objetivo de esta plataforma es realizar el procesamiento de lenguaje natural, por medio de un servicio WEB estilo REST se consume para enviar las instrucciones del usuario.

El agente conversacional que se configuró en la plataforma Dialogflow se consideró para el idioma español como idioma por defecto, sin embargo, es posible configurar diferentes idiomas tal y como se muestra en la Figura 3.15.

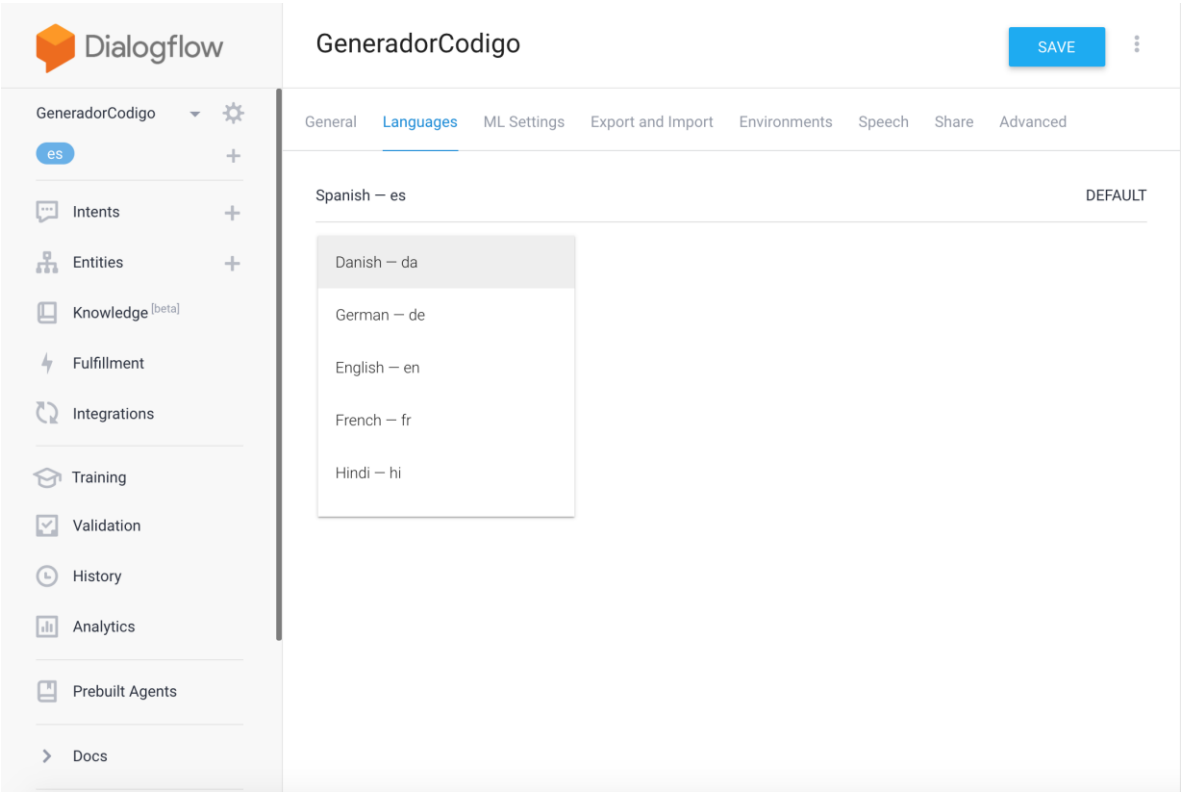

Figura 3.15 Configuración de idioma en Dialogflow

El agente conversacional que se configuró para generador de aplicaciones móviles tiene tres intenciones, lo cual significa que dependiendo de las instrucciones del usuario es el intento que se activará. Los intentos se configuran en tres partes requeridas: 1) Las frases de entrenamiento, 2) Acciones y parámetros y 3)

Respuestas. En la siguiente lista se describe cada una de las partes requeridas de un intento:

1. **Frases de entrenamiento:** Como su nombre lo indica, son aquellas frases que el agente espera por parte del usuario, es así como se identifica que intento está activo.
2. **Acciones y parámetros:** A partir de las fases de entrenamiento, se especifican entidades que determinan los elementos requeridos o no requeridos para que la intención proporcione una respuesta. Si un parámetro es requerido se configura una frase para pedirlo. Cada parámetro se representa por una entidad.
3. **Respuestas:** Una vez que el agente identifico que todos los elementos de la intención de usuario fueron proporcionados entonces el agente da una respuesta a la intención del usuario.

La Figura 3.16 ilustra ejemplos de frases de entrenamiento las cuales se utilizaron para el generador de aplicaciones móviles.

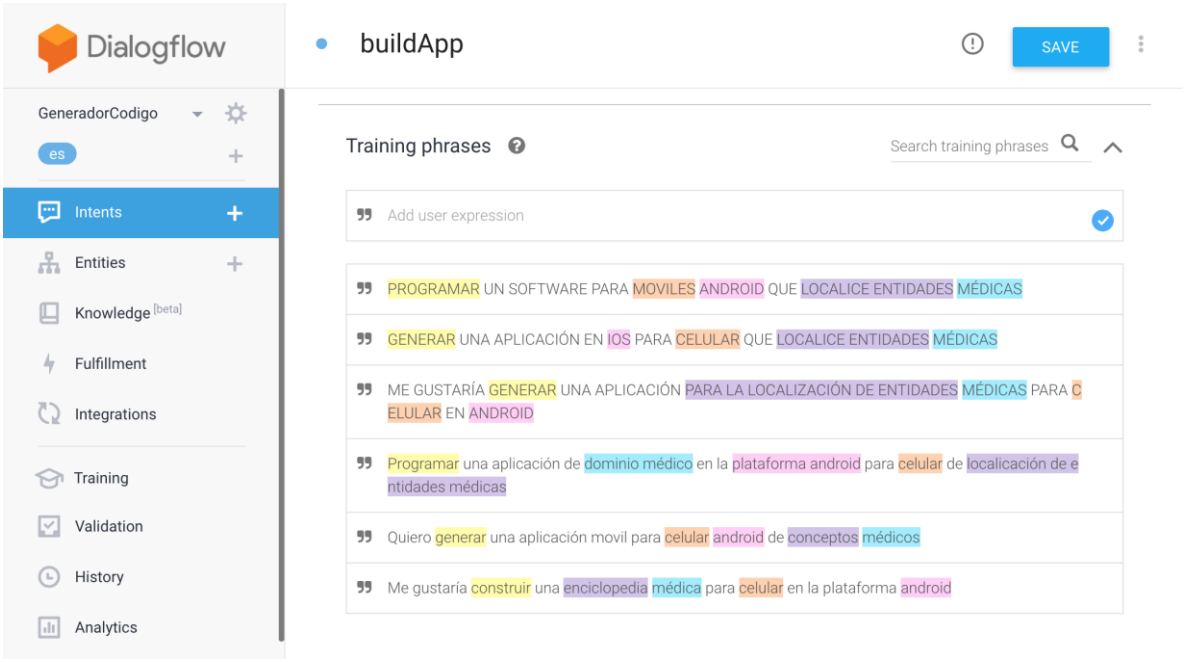

Figura 3.16 Frases de entrenamiento en Dialogflow

Por otro lado, la Figura 3.17 ilustra un ejemplo de acciones y parámetros de un *intent*, en esta sección se especifica si el parámetro es requerido, el nombre del parámetro, la entidad que representa el parámetro, el valor del parámetro (este valor es obtenido por el PLN, es decir, lo que el usuario pidió), se especifica si el parámetro es una lista, y por último, se especifica una oración para pedir el parámetro en caso de que sea requerido.

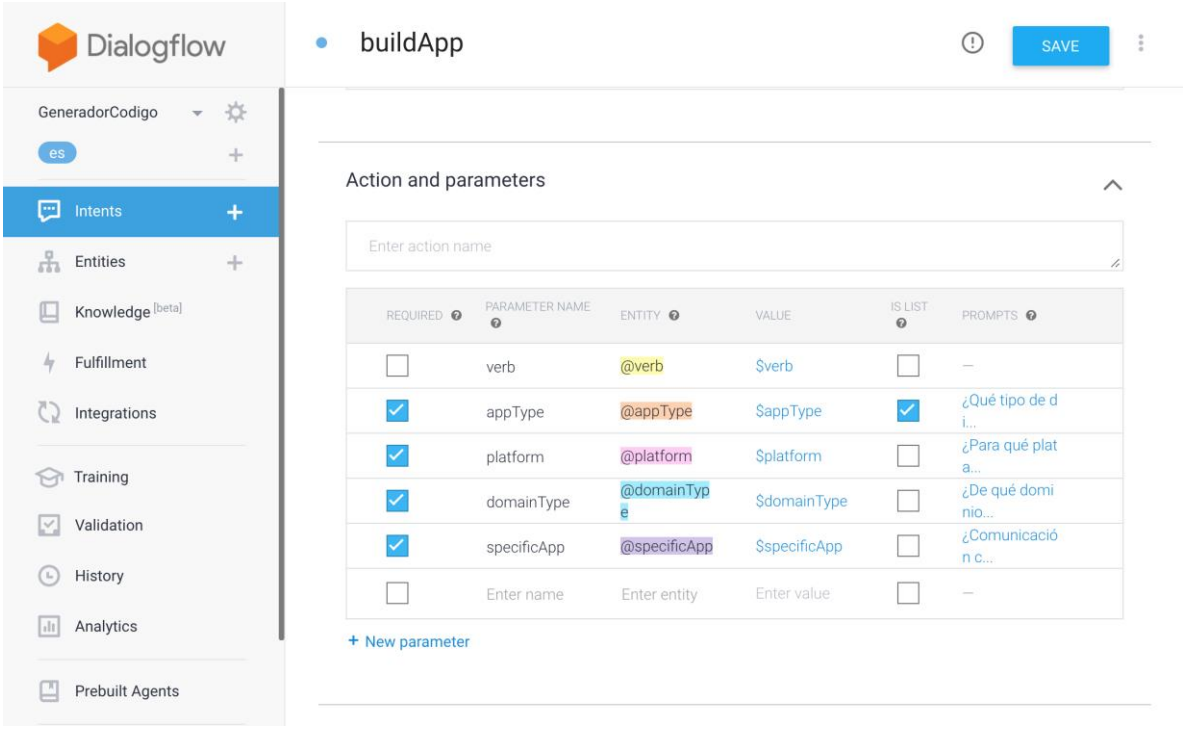

Figura 3.17 Sección de acciones y parámetros de Dialogflow

La Figura 3.18 se muestra ejemplos de las repuestas que se configuraron para el generador de aplicaciones móviles, cuando se identifican todos los parámetros requeridos de la intención del usuario entonces el agente proporciona una respuesta.

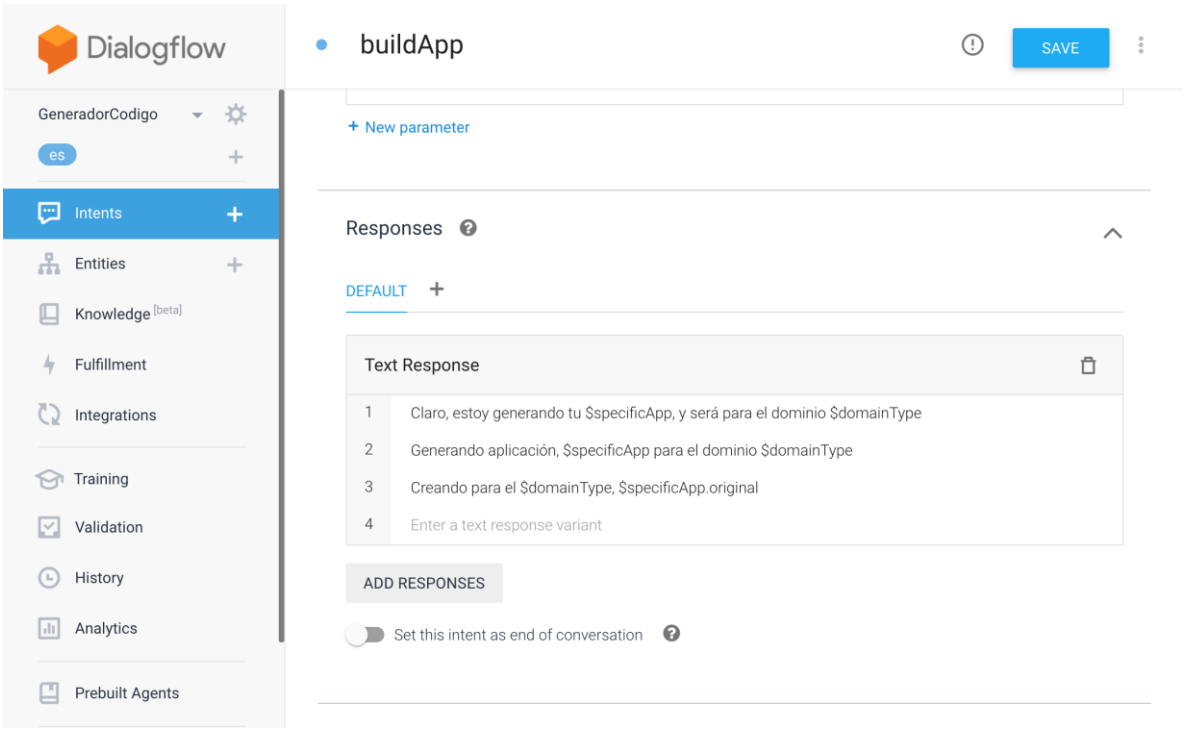

Figura 3.18 Respuestas en Dialogflow

Por otro lado, cada parámetro de un intento pertenece a un tipo, el cual se denomina tipo de entidad. Esto determina de forma exacta cómo se extraen los datos de una expresión de usuario final.

Dialogflow proporciona entidades del sistema predefinidas que detectan coincidencias con muchos tipos comunes de datos. Por ejemplo, hay entidades del sistema que detectan coincidencias con fechas, horas, colores, direcciones de correo electrónico, por mencionar algunos. También se crean entidades personalizadas para detectar coincidencias en datos personalizados. Por ejemplo, al definir una entidad verdura que detecte coincidencias con los tipos de verduras que se vendan mediante un agente de supermercado.

El término entidad se usa en referencia al concepto general de las entidades. A continuación, se describen específicamente los términos más importantes:

1. **Tipo de entidad:** Define el tipo de información que se extrae de la entrada del usuario. Por ejemplo, verdura es el nombre de un tipo de entidad. Si haces clic en Crear entidad en la consola de Dialogflow, se crea un tipo de entidad. Cuando se usa la API, el término tipo de entidad hace referencia al tipo EntityType.
2. **Entrada de entidad:** Por cada tipo de entidad, hay muchas entradas de entidad. Cada entrada de entidad proporciona un conjunto de palabras o frases que se consideran equivalentes. Por ejemplo, si verdura es un tipo de entidad, entonces los siguientes son ejemplos de entradas de entidad:

- zanahoria
- cebollín, cebolla verde
- pimentón, pimiento dulce

Cuando se edita un tipo de entidad en la consola de Dialogflow, cada fila de la pantalla corresponde a una entrada de entidad. Cuando se usa la API, el término entrada de entidad hace referencia al tipo Entity (EntityType.Entity o EntityType_Entity en algunos lenguajes de biblioteca cliente).

En la Figura 3.19 se muestran las entidades para el generador de aplicaciones móviles. Además, la Figura 3.20 ilustra un ejemplo de entrada de entidad, en este caso para el dominio que se considera en el generador de aplicaciones móviles.

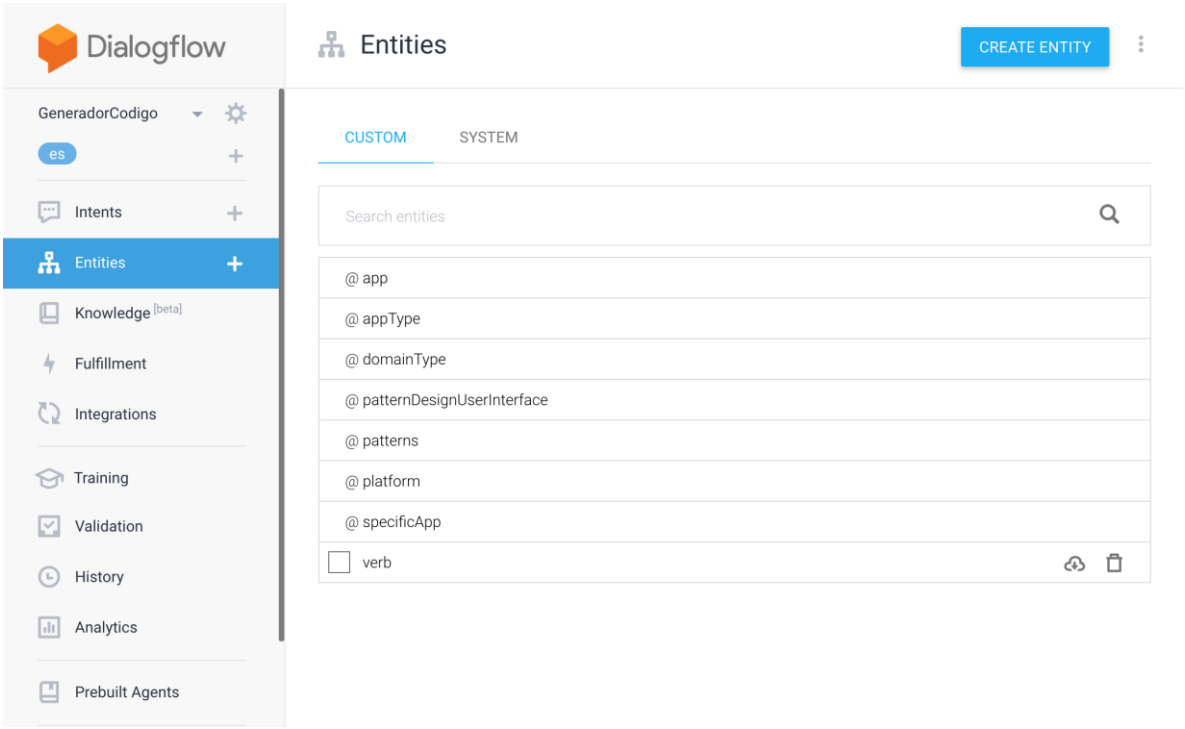

Figura 3.19 Entidades para el generador de aplicaciones móviles

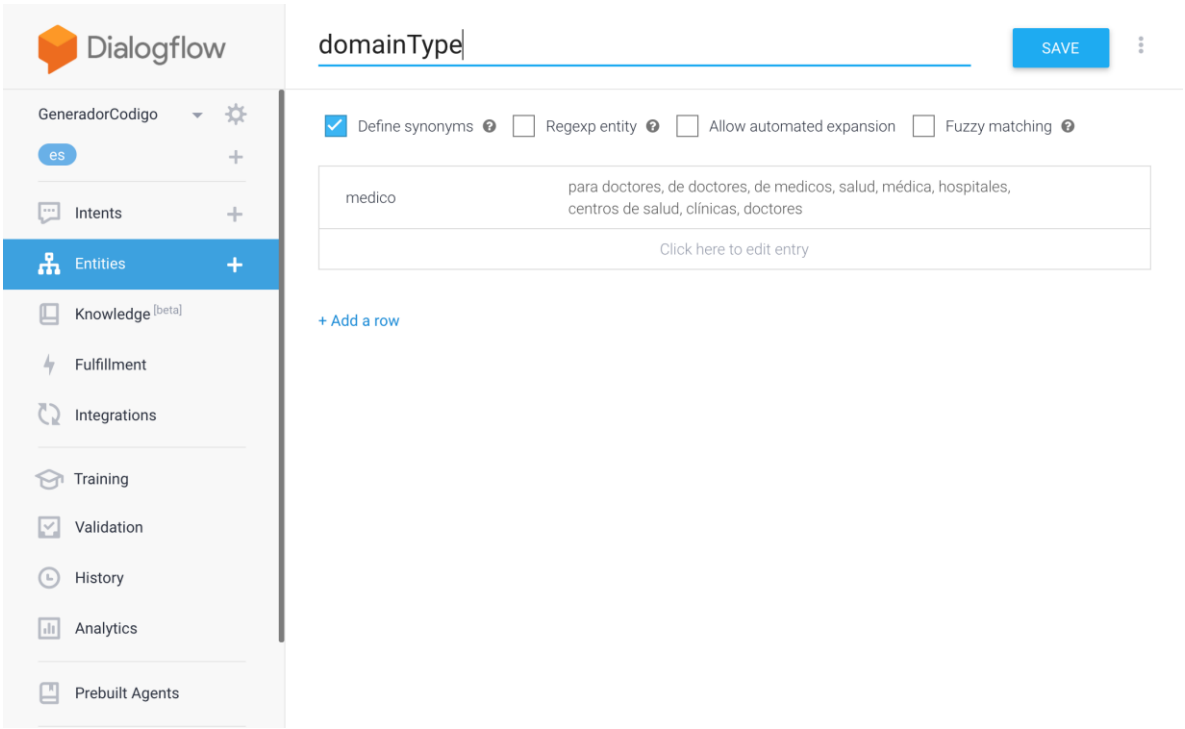

Figura 3.20 Ejemplo de entrada de entidad para especificar el dominio

Por defecto se tienen dos intentos, uno para dar la bienvenida y el otro para notificarle al usuario que no se identificó lo que desea realizar. Adicionalmente, se creó un intento para el generador de aplicaciones móviles, el cual se consume por medio de un servicio WEB tipo REST. El listado de código 3.1 es el consumo del servicio WEB en donde la expresión del usuario es “Me gustaría construir una del dominio médico, que tenga una lista de conceptos médico, que pueda buscar y pueda ver la definición de cada uno", se utiliza la herramienta CURL.

Listado de código 3.1 consumo de servicio WEB de Dialogflow con CURL

```
curl --location --request POST 'https://api.dialogflow.com/v1/query' \
--header 'Content-Type: application/json' \
--header 'Authorization: Bearer d17acd9d73604d12900fca33ed7b5c0a' \
--data-raw '{
"lang": "es",
"query": "Me gustaría construir una del dominio médico, que tenga una lista de conceptos médico, que pueda buscar y pueda ver la definición de cada uno",
"sessionId": "12345",
"v": "20150910"
}'
```

El listado de código 3.2 es un ejemplo de la estructura de la respuesta de la petición anterior.

Listado de código 3.2 estructura de la respuesta del servicio WEB de Dialogflow

```
{
"id": "f56fb145-380d-408d-ad0a-761a70d62346-0f0e27e1",
"lang": "es",
"sessionId": "12345",
"timestamp": "2020-05-29T13:34:07.458Z",
"result": {
"source": "agent",
"resolvedQuery": "Me gustaría construir una del dominio médico, que tenga una lista de conceptos médico, que pueda buscar y pueda ver la definición de cada uno",
"action": "",
"actionIncomplete": true,
```

```
"score": 0.3922297,
"parameters": {
"verb": "crear",
"domainType": "dominio médico",
"specificApp": "conceptos",
"appType": [],
"platform": ""
},
"contexts": [
{
"name": "b6f8cc32-833f-478c-879e-fca0804ae136_id_dialog_context",
"lifespan": 2,
"parameters": {
"verb": "crear",
"verb.original": "construir",
"domainType": "dominio médico",
"domainType.original": "dominio médico",
"specificApp": "conceptos",
"specificApp.original": "conceptos",
"appType": [],
"appType.original": [],
"platform": "",
"platform.original": ""
}
},
{
"name": "buildapp_dialog_context",
"lifespan": 2,
"parameters": {
"verb": "crear",
"verb.original": "construir",
"domainType": "dominio médico",
"domainType.original": "dominio médico",
"specificApp": "conceptos",
"specificApp.original": "conceptos",
"appType": [],
"appType.original": [],
"platform": "",
```

```
49. "platform.original": ""
50. }
51. },
52. {
53. "name": "buildapp_dialog_params_apptype",
54. "lifespan": 1,
55. "parameters": {
56. "verb": "crear",
57. "verb.original": "construir",
58. "domainType": "dominio médico",
59. "domainType.original": "dominio médico",
60. "specificApp": "conceptos",
61. "specificApp.original": "conceptos",
62. "appType": [],
63. "appType.original": [],
64. "platform": "",
65. "platform.original": ""
66. }
67. },
68. {
69. "name": "__system_counters__",
70. "lifespan": 1,
71. "parameters": {
72. "no-input": 0.0,
73. "no-match": 0.0,
74. "verb": "crear",
75. "verb.original": "construir",
76. "domainType": "dominio médico",
77. "domainType.original": "dominio médico",
78. "specificApp": "conceptos",
79. "specificApp.original": "conceptos",
80. "appType": [],
81. "appType.original": [],
82. "platform": "",
83. "platform.original": ""
84. }
85. }
86. ],
```

```
87. "metadata": {
88. "intentId": "b6f8cc32-833f-478c-879e-fca0804ae136",
89. "intentName": "buildApp",
90. "WEBhookUsed": "false",
91. "WEBhookForSlotFillingUsed": "false",
92. "isFallbackIntent": "false"
93. },
94. "fulfillment": {
95. "speech": "¿Qué tipo de dispositivo?",
96. "messages": [
97. {
98. "type": 0,
99. "speech": "¿Qué tipo de dispositivo?"
100.        }
101.        ]
102.        }
103.        },
104.        "status": {
105.        "code": 200,
106.        "errorType": "success"
107.        }
108.        }
```


3.3.3 Reconocimiento de voz

Para capturar las instrucciones del usuario y enviarlas a Dialogflow es necesario convertir el audio a texto. Para realizar este proceso se utilizó el WEB Speech API.

El reconocimiento de voz implica recibir voz a través del micrófono de un dispositivo, luego se verifica por un servicio de reconocimiento de voz contra una lista de palabras o frases (esta lista básicamente es el vocabulario a reconocer en una app particular). Cuando se reconoce con éxito una palabra o frase, esta se devuelve como una cadena de texto y así iniciar otras acciones.

Actualmente, la compatibilidad de la WEB Speech API para el reconocimiento de voz se limita a Chrome para escritorio y Android. Chrome tiene soporte desde la

versión 33 pero con interfaces “prefixed”, por lo que se incluye ese tipo de interfaces “prefixed”, por ejemplo, WEBkitSpeechRecognition.

Como se mencionó anteriormente, Chrome actualmente admite el reconocimiento de voz con propiedades “prefixed”, por lo tanto, como se muestra en el listado de código 3.3, al principio del código se incluye las siguientes líneas para usar los objetos que correspondan a Chrome, y así cualquier implementación en el futuro admitirá estas características sin ningún “prefixed”.

Listado de código 3.3 interfaces de compatibilidad con Chrome

```
1. var SpeechRecognition = SpeechRecognition || WEBkitSpeechRecognition
2. var SpeechGrammarList = SpeechGrammarList || WEBkitSpeechGrammarList
3. var SpeechRecognitionEvent = SpeechRecognitionEvent ||
   WEBkitSpeechRecognitionEvent
```

A continuación, se agrega el listado de código 3.4 en donde se muestra la implementación con la ayuda de AngularJs para el reconocimiento de voz del generador de aplicaciones móviles.

Listado de código 3.4 función para el reconocimiento de voz en el generador de aplicaciones

```
1. var recognition = new WEBkitSpeechRecognition();
2. recognition.onresult = function (event) {
3. $scope.$apply(function () {
4. for (var i = event.resultIndex; i < event.results.length; i++) {
5. if (event.results[i].isFinal) {
6. $scope.theText = event.results[i][0].transcript;
7. $scope.sentences.push({'text': $scope.theText, 'userType': 'user', 'date': new Date()});
8. $(document).scrollTop($(document).height(), 1000);
9. $scope.requestToDialogflow($scope.theText);
10. }
11. }
12. });
13. };
14. recognition.onstart = function(){
15. $scope.actionText = "Escuchando instrucciones";
```

```
16. $scope.$apply();
17. }
18. recognition.start();
```

3.3.4 Síntesis de voz

Para darle una retroalimentación al usuario de lo que necesita generar es necesario dar instrucciones o pedir los elementos faltantes en la intención del usuario, por ese motivo se implementa la síntesis de voz con WEB Speech API.

La síntesis de voz (también conocida como texto a voz o tts) implica recibir contenido en forma de texto dentro de una aplicación y convertirla en voz a través del altavoz del dispositivo o de la conexión de salida del audio.

El soporte para la síntesis de voz WEB Speech API solo llega a los navegadores más importantes y actualmente se limita a los siguientes:

- Firefox para escritorio y dispositivos móviles en Gecko 42+ (Windows)/44+, sin prefijos, y se puede activar configurando el flag media. WEBspeech. synth.enabled a true en about:config.
- Firefox OS 2.5+ lo soporta por defecto y sin ser necesario ningún permiso.
- Chrome para escritorio y Android tienen soporte desde la versión 33, sin prefijos.

El listado de código 3.5 muestra la función en AngularJs de la implementación de la síntesis de voz del generador de aplicaciones móviles.

Listado de código 3.5 función para realizar la síntesis de voz en el generador de aplicaciones

```
1. $scope.sayIt = function (textToSpeech) {
2. $scope.sentences.push({'text': textToSpeech, 'userType': 'boot', 'date': new Date()});
3. $scope.actionText = "Agente conversacional hablando";
4. $(document).scrollTop($(document).height(), 1000);
```

```
5. let speechSynthesis = new SpeechSynthesisUtterance(textToSpeech);
6. speechSynthesis.onend = function () {
7. $scope.dictateIt();
8. }
9. window.speechSynthesis.speak(speechSynthesis);
10. };
```

3.3.5 Construcción de XML

En esta sección se describe la estructura y generación del código intermedio para construir una aplicación móvil. El código intermedio se representa por un archivo XML, este se genera en una aplicación WEB en back-end se utiliza el lenguaje de programación PHP.

En el listado de código 3.6 se representa el código intermedio XML para una aplicación de comunicación clínica para la plataforma Android™.

Listado de código 3. 6 ejemplo de código intermedio XML

```
1. <?xml version="1.0"?>
2. <application xmlns="http://www.itodepi.edu.mx"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://www.itodepi.edu.mx define.xsd">
5. <userName>Jesus Fernández Avelino</userName>
6. <name>wiki</name>
7. <vendor>Instituto Tecnológico de Orizaba</vendor>
8. <developerAccount>ITO</developerAccount>
9. <WEBsite></WEBsite>
10. <shortName>com-cli</shortName>
11. <version>1.0</version>
12. <icon>icono.png</icon>
13. <typeApp>a3</typeApp>
14. <authors>
15. <author>
```

```
16. <name>Jesus Fernández Avelino</name>
17. <email>fdezjesus12@gmail.com</email>
18. <WEBsite>jesusfdez.com</WEBsite>
19. <company>ITO</company>
20. </author>
21. </authors>
22. <deploySettings>
23. <build>
24. <mobile>
25. <Android>true</Android>
26. </mobile>
27. </build>
28. <orientation>
29. <portrait>true</portrait>
30. <landscape>true</landscape>
31. </orientation>
32. </deploySettings>
33. <navigation>
34. <page>
35. <id>1</id>
36. <name>com-cli</name>
37. <title>Prueba comunicación clínica</title>
38. <position>
39. <head></head>
40. <body>
41. <uidp type="single" name="com-cli"></uidp>
42. </body>
43. <left></left>
44. <right></right>
45. <foot></foot>
46. </position>
47. </page>
48. </navigation>
49. </application>
```

3.3.6 Desarrollo de aplicaciones médicas para las plataformas Android y iOS

En esta sección se presenta el desarrollo de las aplicaciones médicas para comunicación clínica y localización de entidades médicas en las plataformas Android™ y iOS®, respectivamente.

La aplicación de comunicación médica consiste en el desarrollo de un chat, esta aplicación se desarrolló para el sistema operativo Android™, es compatible con el API de Android™ a partir de la versión 21 que equivale a Lollipop 5.0. A continuación, se describen las cuatro pantallas que conforman la aplicación:

Pantalla inicial: Como se aprecia en la Figura 3.21, la pantalla muestra dos botones, uno para realizar el registro por primera vez y el otro para iniciar sesión en caso de que el usuario ya esté registrado.

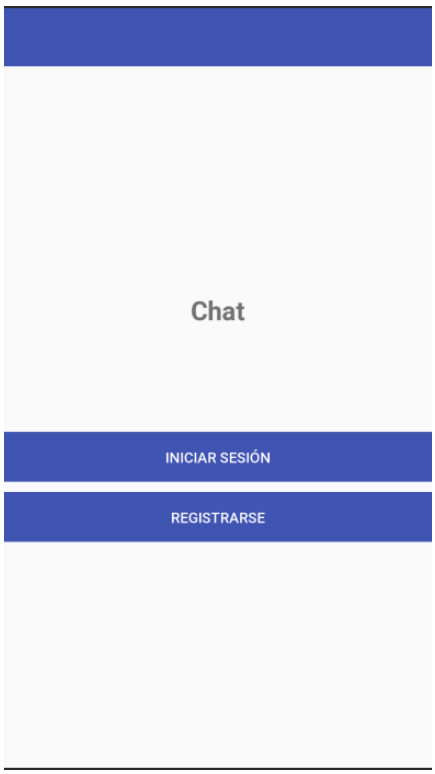

Figura 3.21 Pantalla inicial del chat

Registro: La pantalla de registro pide al usuario ingresar un nombre de usuario, un correo electrónico y una contraseña tal como se muestra en la Figura 3.22.

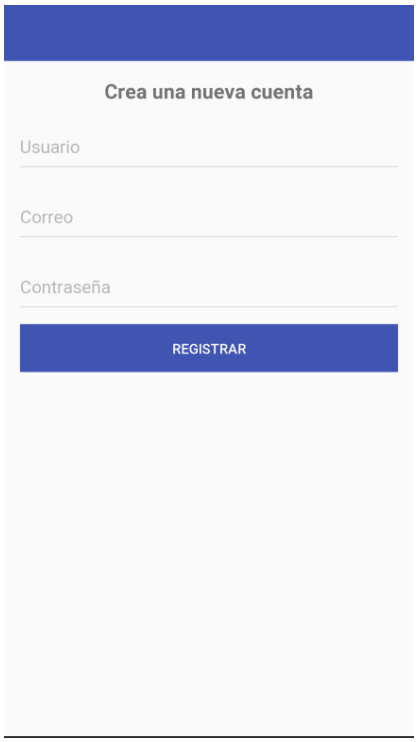

Figura 3.22 Pantalla de registro del chat

Inicio de sesión: El inicio de sesión se muestra en la Figura 3.23 en donde se pide correo y contraseña para iniciar sesión en el chat.

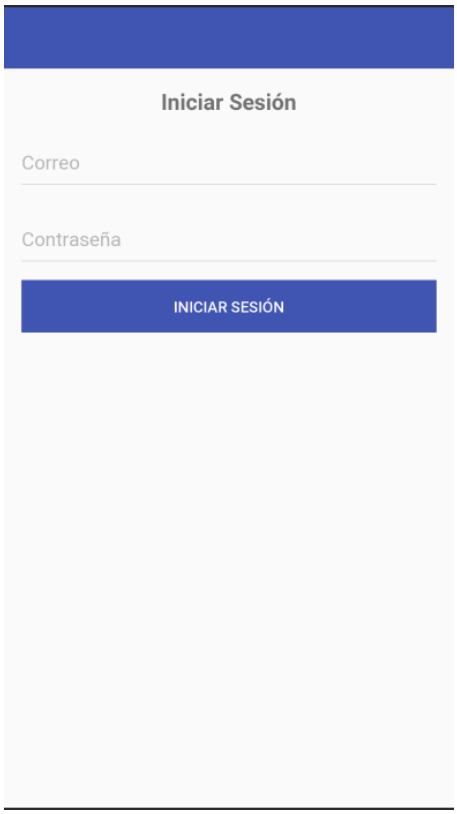

Figura 3. 23 Pantalla de inicio de sesión del chat

Conversaciones y contactos: Esta pantalla está dividida en dos pestañas: Chats y Usuarios. La pestaña de Chats se enlistan las conversaciones que se realizan con cada uno de los usuarios como se visualiza en la Figura 3.24. Por otro lado, La Figura 3.25 muestra la pestaña de Usuarios, en esta pestaña se muestran los usuarios registrados en el chat, además, se facilita la búsqueda con una barra de búsqueda en la parte inferior, al tocar un contacto se muestra una pantalla de conversación, tal y como lo muestra la Figura 3.26.

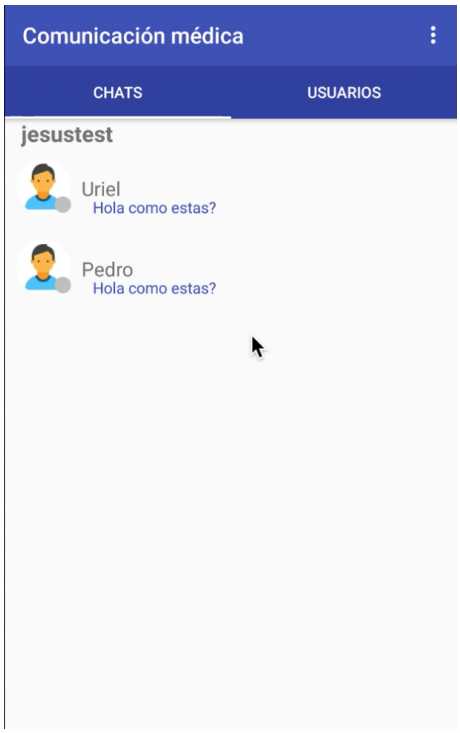

Figura 3.24 Histórico de chats

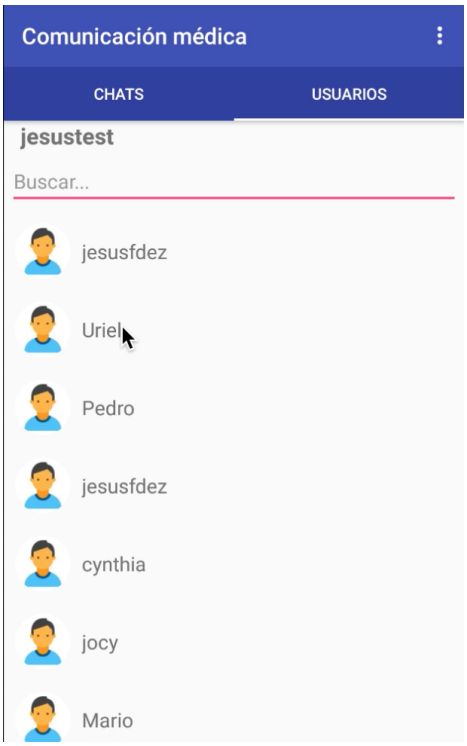

Figura 3.25 Búsqueda de usuarios del chat

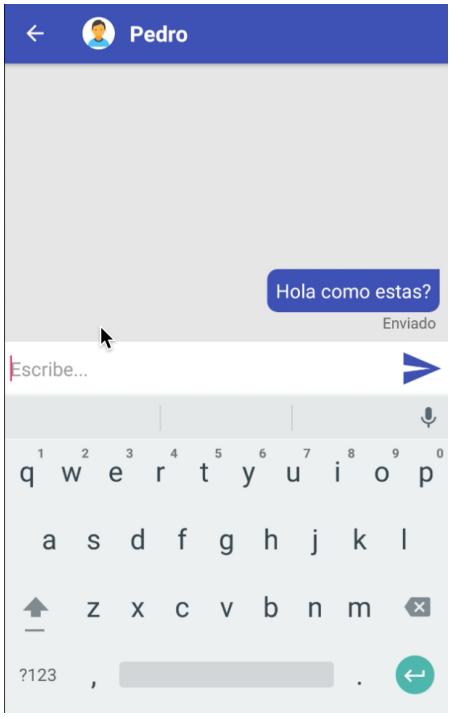

Figura 3.26 Conversación con un usuario en el chat

En el listado de código 3.7 se muestra la clase en Java de la actividad que envía los mensajes.

Listado de código 3.7 MessagesActivity.java

```
1. package com.example.composicion03;
2. import androidx.annotation.NonNull;
3. import androidx.appcompat.app.AppCompatActivity;
4. import androidx.appcompat.widget.Toolbar;
5. import androidx.recyclerview.widget.LinearLayoutManager;
6. import androidx.recyclerview.widget.RecyclerView;
7. import android.content.Intent;
8. import android.os.Bundle;
9. import android.view.View;
10. import android.widget.EditText;
11. import android.widget.ImageButton;
12. import android.widget.TextView;
13. import android.widget.Toast;
14. import com.example.composicion03.Adapter.MessageAdapter;
15. import com.example.composicion03.Model.Chat;
16. import com.example.composicion03.Model.User;
17. import com.google.firebase.auth.FirebaseAuth;
18. import com.google.firebase.auth.FirebaseUser;
19. import com.google.firebase.database.DataSnapshot;
20. import com.google.firebase.database.DatabaseError;
21. import com.google.firebase.database.DatabaseReference;
22. import com.google.firebase.database.FirebaseDatabase;
23. import com.google.firebase.database.ValueEventListener;
24. import java.util.ArrayList;
25. import java.util.HashMap;
26. import java.util.List;
27. import de.hdodenhof.circleimageview.CircleImageView;
28. public class MessageActivity extends AppCompatActivity {
29. CircleImageView profile_image;
30. TextView username;
31. FirebaseUser fuser;
32. DatabaseReference reference;
33. ImageButton btn_send;
```

```
34. EditText text_send;
35. Intent intent;
36. MessageAdapter messageAdapter;
37. List<Chat> mchat;
38. RecyclerView recyclerView;
39. ValueEventListener seenListener;
40. @Override
41. protected void onCreate(Bundle savedInstanceState) {
42. super.onCreate(savedInstanceState);
43. setContentView(R.layout.activity_message);
44. Toolbar toolbar = findViewById(R.id.toolbar);
45. setSupportActionBar(toolbar);
46. getSupportActionBar().setTitle("");
47. getSupportActionBar().setDisplayHomeAsUpEnabled(true);
48. toolbar.setNavigationOnClickListener(new View.OnClickListener() {
49. @Override
50. public void onClick(View view) {
51. startActivity(new Intent(MessageActivity.this,
    Main2Activity.class).setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP));
52. }
53. });
54. recyclerView = findViewById(R.id.recycler_view);
55. recyclerView.setHasFixedSize(true);
56. LinearLayoutManager linearLayoutManager = new
    LinearLayoutManager(getApplicationContext());
57. linearLayoutManager.setStackFromEnd(true);
58. recyclerView.setLayoutManager(linearLayoutManager);
59. profile_image = findViewById(R.id.profile_image);
60. username = findViewById(R.id.username);
61. btn_send = findViewById(R.id.btn_send);
62. text_send = findViewById(R.id.text_send);
63. intent = getIntent();
64. final String userid = intent.getStringExtra("userid");
65. fuser = FirebaseAuth.getInstance().getCurrentUser();
66. btn_send.setOnClickListener(new View.OnClickListener() {
67. @Override
68. public void onClick(View view) {
69. String msg = text_send.getText().toString();
```

```
70. if(!msg.equals("")){
71. sendMessage(fuser.getUid(), userid, msg);
72. }else{
73. Toast.makeText(MessageActivity.this, "No puedes enviar mensaje vacio.",
    Toast.LENGTH_SHORT).show();
74. }
75. text_send.setText("");
76. }
77. });
78. reference = FirebaseDatabase.getInstance().getReference("Users").child(userid);
79. //Imagen y nombre del toolbar
80. reference.addValueEventListener(new ValueEventListener() {
81. @Override
82. public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
83. User user = dataSnapshot.getValue(User.class);
84. username.setText(user.getUsername());
85. profile_image.setImageResource(R.drawable.icon_1);
86.  Glide.with(Main2Activity.this).load(user.getIamgeURL()).into(profile_image);
87. readMessage(fuser.getUid(), userid, user.getImageURL());
88. }
89. @Override
90. public void onCancelled(@NonNull DatabaseError databaseError) {
91. }
92. });
93. seenMessage(userid);
94. }
95. private void seenMessage(final String userid){
96. reference = FirebaseDatabase.getInstance().getReference("Chats");
97. seenListener = reference.addValueEventListener(new ValueEventListener() {
98. @Override
99. public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
100.        for (DataSnapshot snapshot : dataSnapshot.getChildren() ){
101.        Chat chat = snapshot.getValue(Chat.class);
102.        if (chat.getReceiver().equals(fuser.getUid()) && chat.getSender().equals(userid)){
103.        HashMap<String, Object> hashMap = new HashMap<>();
104.        hashMap.put("isseen", true);
105.        snapshot.getRef().updateChildren(hashMap);
106.        }
```

```
107.        }
108.        }
109.        @Override
110.        public void onCancelled(@NonNull DatabaseError databaseError) {
111.        }
112.        });
113.        }
114.        private void sendMessage (final String myid, final String userid,final String msg){
115.        DatabaseReference reference = FirebaseDatabase.getInstance().getReference();
116.        HashMap<String, Object> hashMap = new HashMap<>();
117.        hashMap.put("sender", myid);
118.        hashMap.put("receiver", userid);
119.        hashMap.put("message", msg);
120.        hashMap.put("isseen", false);
121.        reference.child("Chats").push().setValue(hashMap);
122.        final DatabaseReference chatRef =
    FirebaseDatabase.getInstance().getReference("Chatlist").child(fuser.getUid()).child(userid);
123.        //Agregar usuario al fragment chat
124.        chatRef.addListenerForSingleValueEvent(new ValueEventListener() {
125.        @Override
126.        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
127.        if (!dataSnapshot.exists()){
128.        chatRef.child("id").setValue(userid);
129.        }
130.        }
131.        @Override
132.        public void onCancelled(@NonNull DatabaseError databaseError) {
133.        }
134.        });
135.        }
136.        private void readMessage(final String myid, final String userid, final String
    imageurl){
137.        mchat = new ArrayList<>();
138.        reference = FirebaseDatabase.getInstance().getReference("Chats");
139.        reference.addValueEventListener(new ValueEventListener() {
140.        @Overridepublic void onDataChange(@NonNull DataSnapshot dataSnapshot) {
141.        mchat.clear();
142.        for (DataSnapshot snapshot : dataSnapshot.getChildren()){
```

```
143.        Chat chat = snapshot.getValue(Chat.class);
144.        if (chat.getReceiver().equals(myid) && chat.getSender().equals(userid) ||
    chat.getReceiver().equals(userid) && chat.getSender().equals(myid)){
145.        mchat.add(chat);
146.        }
147.        messageAdapter = new MessageAdapter(MessageActivity.this,  mchat, imageurl);
148.        recyclerView.setAdapter(messageAdapter);
149.        }
150.        }
151.        @Override
152.        public void onCancelled(@NonNull DatabaseError databaseError) {
153.        }
154.        });
155.        }
156.        private void status(String status){
157.        reference =
    FirebaseDatabase.getInstance().getReference("Users").child(fuser.getUid());
158.        HashMap<String, Object> hashMap = new HashMap<>();
159.        hashMap.put("status", status);
160.        reference.updateChildren(hashMap);
161.        }
162.        @Override
163.        protected void onResume() {
164.        super.onResume();
165.        status("En linea");
166.        }
167.        @Override
168.        protected void onPause() {
169.        super.onPause();
170.        reference.removeEventListener(seenListener);
171.        status("Desconectado");
172.        }
173.        }
```

Por otra parte, la aplicación de localización de entidades médicas se desarrolló para el sistema operativo iOS® con el objetivo de buscar en un mapa hospitales, centros de salud y consultorios médicos particulares. La aplicación es compatible a partir de la versión del sistema operativo 9.3. En la Figura 3.27 se visualiza la aplicación.

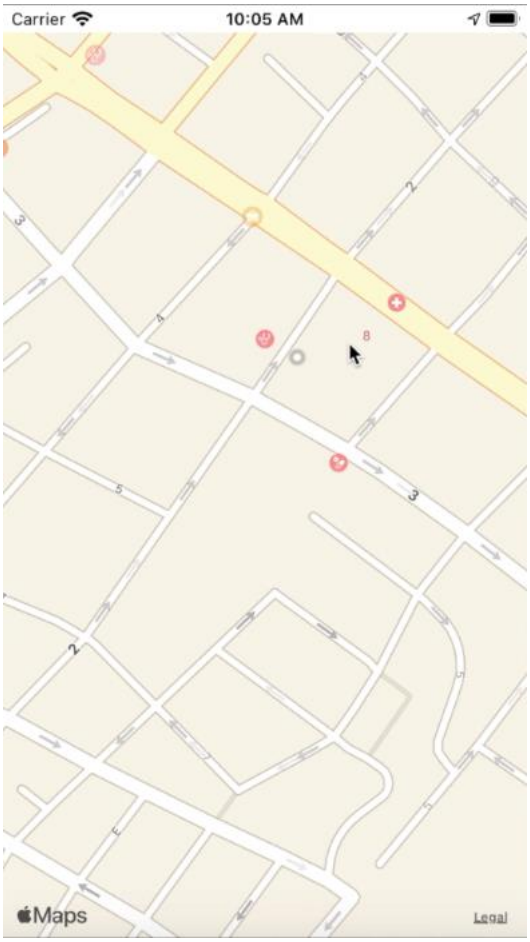

Figura 3.27 Aplicación en iOS® para localización de entidades médicas

El listado de código 3.8 muestra la clase principal en el lenguaje Swift de la aplicación de localización de entidades médicas.

Listado de código 3. 8 MedicalEntities.swift

```
1. //
2. //  ViewController.swift
3. //  Componente Mapa iOS
```

```
4.  //
5.  //  Created by Jesus Fernández Avelino on 5/16/19.
6.  //  Copyright © 2019 Jesus Fernández Avelino. All rights reserved.
7.  //
8.  import UIKit
9.  import MapKit
10. import CoreLocation
11. class ViewController: UIViewController {
12. @IBOutlet weak var viewMap: MKMapView!
13. let locationManager = CLLocationManager()
14. //metros a la redonda del dispositivo
15. let regionInMeters: Double = 10000
16. override func viewDidLoad() {
17. super.viewDidLoad()
18. // Do any additional setup after loading the view.
19. checkLocationService()
20. }
21. func setupLocationManager(){
22. locationManager.delegate = self
23. locationManager.desiredAccuracy = kCLLocationAccuracyBest
24. }
25. func centerViewOnUserLocation(){
26. if let location = locationManager.location?.coordinate{
27. let region = MKCoordinateRegion.init(center: location, latitudinalMeters: regionInMeters, longitudinalMeters: regionInMeters)
28. viewMap.setRegion(region, animated: true)
29. }
30. }
31. func checkLocationService(){
32. if CLLocationManager.locationServicesEnabled(){
33. setupLocationManager()
34. checkLocationAuthorization()
35. } else{
36. //alerte al usuario que tiene que encenderlo
37. }
38. }
39. func checkLocationAuthorization(){
40. switch CLLocationManager.authorizationStatus() {
```

```
41. case .authorizedAlways:
42. viewMap.showsUserLocation = true
43. centerViewOnUserLocation()
44. //locationManager.startUpdatingLocation()
45. break
46. case .authorizedWhenInUse:
47. viewMap.showsUserLocation = true
48. centerViewOnUserLocation()
49. //locationManager.startUpdatingLocation()
50. break
51. case .denied:
52. break
53. case .notDetermined:
54. locationManager.requestWhenInUseAuthorization()
55. break
56. case .restricted:
57. break
58. }
59. }
60. }
61. extension ViewController: CLLocationManagerDelegate{
62. func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]){
63. guard let location = locations.last else { return }
64. let center = CLLocationCoordinate2D(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)
65. let region = MKCoordinateRegion.init(center: center, latitudinalMeters: regionInMeters, longitudinalMeters: regionInMeters)
66. viewMap.setRegion(region, animated: true)
67. }
68. func locationManager(_ manager: CLLocationManager, didChangeAuthorization status: CLAuthorizationStatus) {
69. checkLocationAuthorization()
70. }
71. }
```

3.3.7 Proceso de transformación de la intención del usuario

En este punto se describe el proceso para identificar la intención del usuario para generar una aplicación móvil. Como primer punto se presenta la arquitectura del generador de aplicaciones, posteriormente, se describe la interacción entre el usuario, la aplicación WEB y el PLN para identificar la intención del usuario y construir el archivo de código intermedio XML. Por último, se describe el módulo de generación de código nativo de aplicaciones móviles a partir de lo descrito en el archivo de código intermedio XML.

3.3.8 Identificación de los elementos de intención del usuario

En esta sección se describe el proceso que se realiza para la identificación de los elementos que conforman la intención del usuario mediante PLN. La intención del usuario está dividida en cuatro elementos, descritos a continuación:

5) **Dispositivo:** Para este trabajo siempre será móvil ya que se generaron aplicaciones para dispositivos móviles.
6) **Plataforma:** Android™ y iOS® son las plataformas compatibles con las aplicaciones móviles que se generaron.
7) **Dominio:** Las aplicaciones se enfocan en el área del cuidado de la salud, por lo tanto, el dominio médico es el indicado.
8) **Contenido:** Son los PDIU que conforman la aplicación que se genera.

El proceso para identificar la intención del usuario inicia cuando el usuario dicta instrucciones a la aplicación WEB de administrador de diálogo a través de reconocimiento de voz. Este proceso es iterativo hasta que todos los elementos de la intención del usuario se identifican, cada elemento se identifica en una solo oración o el administrador de diálogo pide al usuario dependiendo de qué elemento falte por medio del WEB API de síntesis de voz. La Figura 3.28 ilustra un ejemplo para generar una aplicación del dominio médico para dispositivos iOS para la localización de entidades médicas.

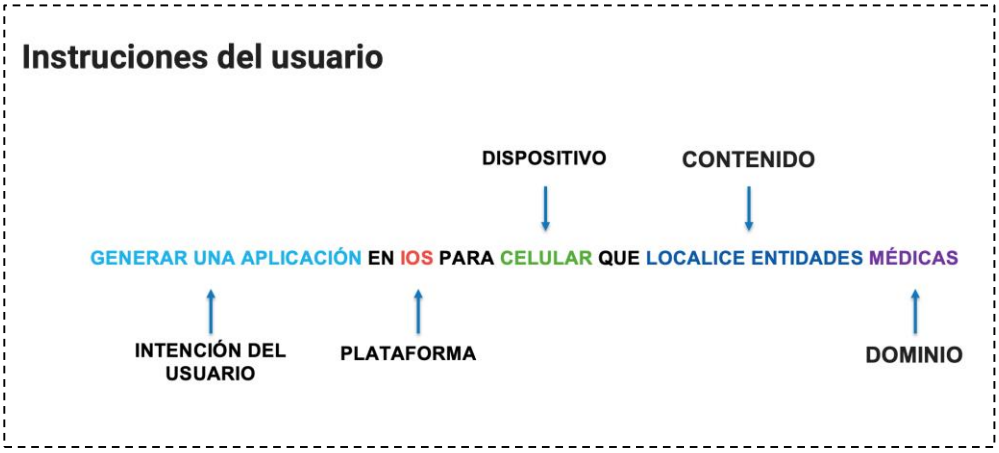

Figura 3.28 Identificación de elementos de la intención del usuario

Cabe mencionar que para cada elemento de la intención del usuario se consideró un conjunto de sinónimos, porque que existe la posibilidad de que el usuario dicte las instrucciones con diferentes palabras, esto se especificó en el agente conversacional que se configuró en la Dialogflow. Tomando el ejemplo ilustrado en la Figura 3.28, en la Figura 3.29 se ilustra el conjunto de palabras para detectar la intención del usuario. Las palabras que se consideraron para el dispositivo de muestran en la Figura 3.30. Por otro lado, la Figura 3.31 muestra el conjunto de palabras que se consideraron para detectar el contenido de la aplicación. Las palabras para especificar el dominio de la aplicación se ilustran en la Figura 3.32. Por último, la Figura 3.33 se ilustran las dos plataformas para desarrollar las aplicaciones móviles.

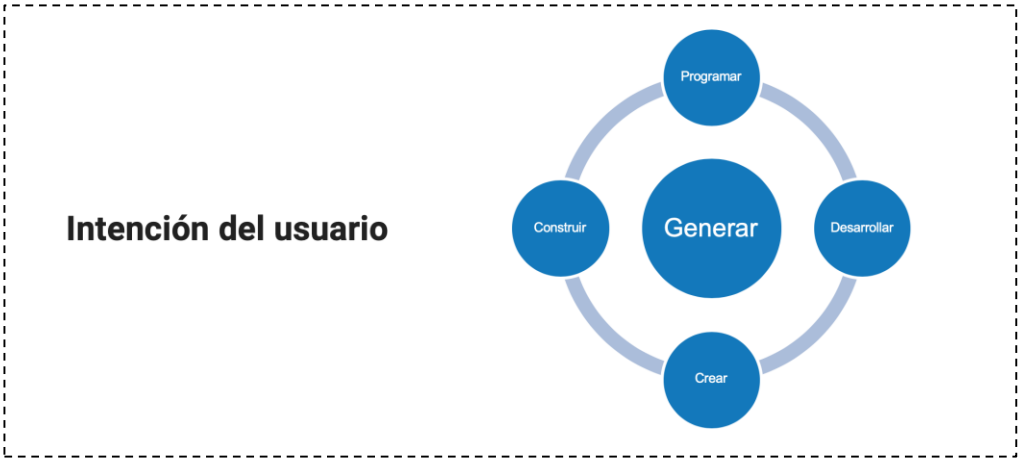

Figura 3.29 Sinónimos para la intención del usuario

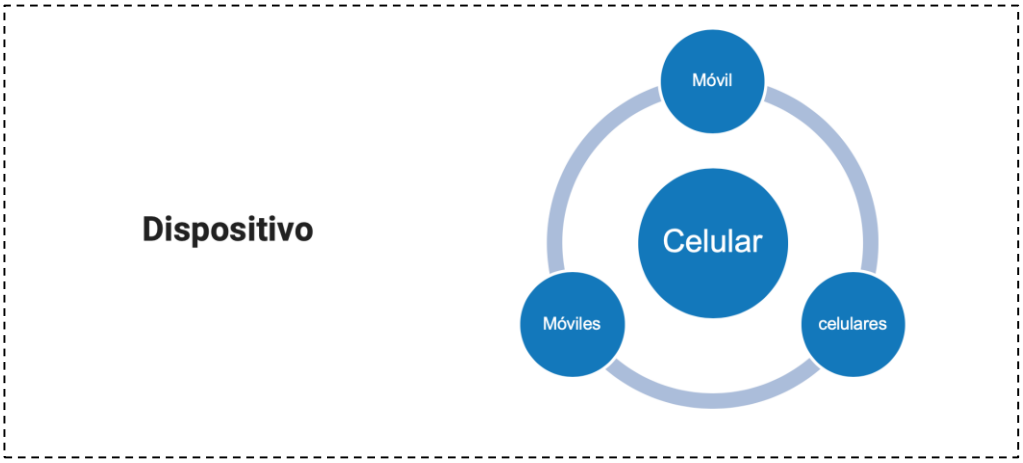

Figura 3.30 Sinónimos para el dispositivo

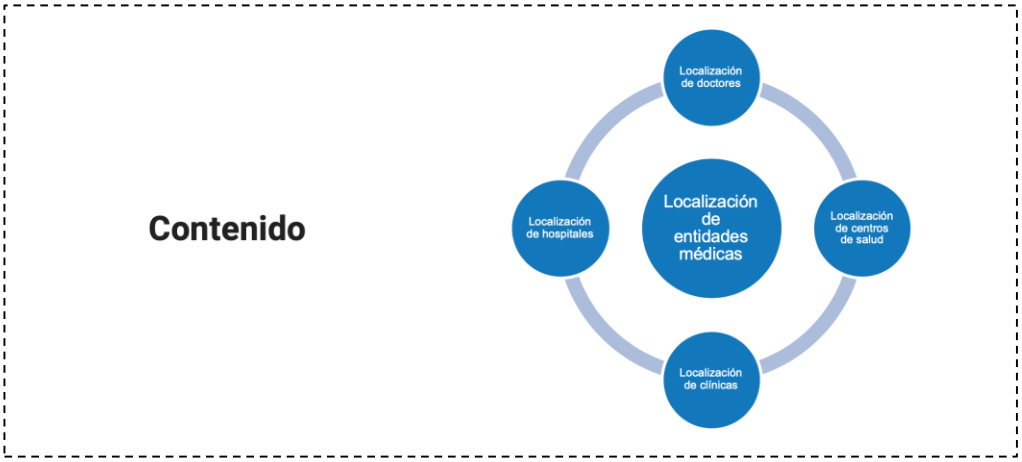

Figura 3.31 Sinónimos para el contenido

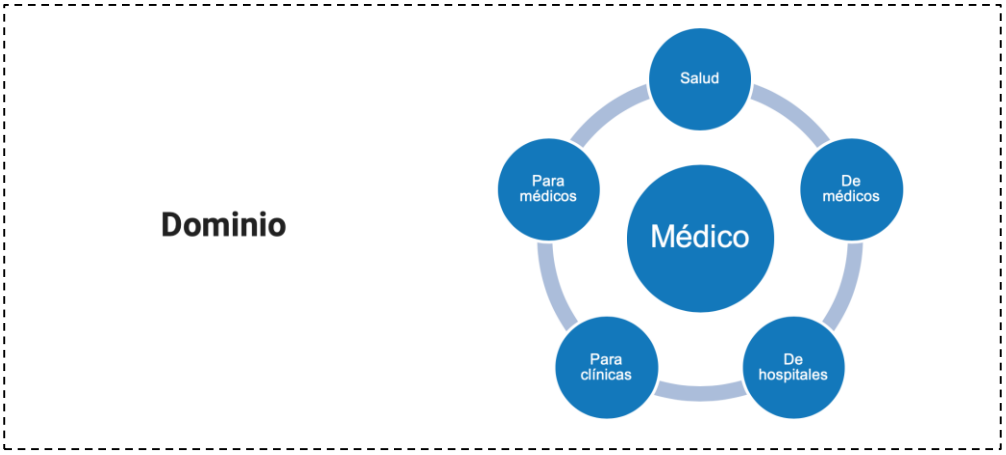

Figura 3.32 Sinónimos para el dominio

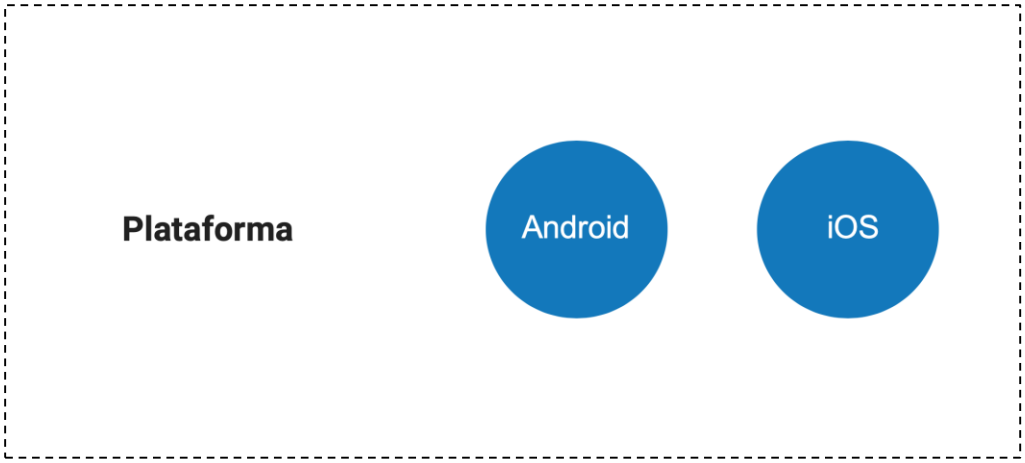

Figura 3.33 Plataformas compatibles

El módulo de administración de diálogo se encarga de realizar el reconocimiento de voz a través del API WEB ya que facilita el proceso de convertir el audio captado a texto. Después el texto se envía a la plataforma de Dialogflow a través de una petición HTTP para realizar el procesamiento de lenguaje natural e identificar los elementos de la intención del usuario. Dialogflow regresa una respuesta con o sin los elementos completamente identificados, sin embargo, el administrador de diálogo pide al usuario por medio del WEB API de síntesis de voz los elementos faltantes.

3.3.9 Módulo de generación de código nativo de aplicaciones móviles

El módulo para generar el código nativo se compone de dos aplicaciones WEB, la primera es para construir el código intermedio XML y la segunda recibe el archivo XML y genera el código.

El objetivo de la primera aplicación WEB es configurar el tipo de aplicación a generar por medio de un archivo XML para este caso una configuración para generar una aplicación para el dominio médico. Los pasos de este proceso son los siguientes.

1. **Selección del dominio:** la selección del dominio de la aplicación a generar se basa en el contenido identificado en la etapa de PLN, ya que no todos los PDIU se validan para el dominio médico.
2. **Selección del tipo de aplicación a generar:** En este caso, se permite generar aplicaciones para dispositivos móviles. Esta decisión se toma con la finalidad de que la generación de código se realice acorde las características de cada aplicación en específico. Para la selección del tipo de aplicación se toma en cuenta el contenido y la intención del usuario, por lo tanto, se infiere el tipo de aplicación a generar.
3. **Configuración de la aplicación:** de acuerdo con el dominio y tipo de aplicación que se selecciona en los pasos previos, se selecciona la configuración general de la aplicación. Algunos de los datos de configuración incluyen título principal, una cadena compuesta para 50 caracteres; lenguaje de desarrollo una cadena de 20 caracteres, los lenguajes soportados son

para WEB con HTML5, Android® con Java y iOS® con Objetive –C. Otro dato de configuración es la versión de la aplicación, donde se aceptan solo números, puntos y guiones. Finalmente, datos del autor de la aplicación como nombre, apellidos, e-mail, empresa y e-mail de la empresa.

La segunda aplicación WEB se encarga de la generación de código, durante esta etapa se realiza la transformación de la información que se recolecta a lo largo del proceso de identificación y configuración necesarios para generar el código fuente de las aplicaciones. Este proceso consta de un par de pasos que se describen a continuación.

1. **Transformación del documento XML a código fuente**: la transformación consiste primero en procesar el documento XML por medio de un documento XML Schema con el objetivo de validar la estructura y garantizar la correcta generación del código fuente de la aplicación. Si la estructura es correcta se genera el equivalente en código por plataforma de los PDIU que se describen en el documento XML. La estructura que se genera se agrupa en clases y se almacenan en una carpeta por plataforma de acuerdo con el lenguaje de programación que el usuario seleccionó.
2. **Archivo ZIP con el código fuente generado:** con el código fuente que se generó se empaquetan las carpetas con el proyecto para la plataforma y se crea un único archivo ZIP, el usuario descarga el código fuente que se modifique o adecue conforme a sus requerimientos y como mejor le convenga. En este sentido, los proyectos que se generan tienen la ventaja de exportarse a entornos de desarrollo de acuerdo a la plataforma de desarrollo. Los entornos de desarrollo válidos para exportar los proyectos son para iOS® el entorno de Xcode™ y Android Studio™ para el proyecto Android.

Capítulo IV. Resultados

En este capítulo, se presenta dos casos de estudios para dispositivos móviles en las plataformas Android™ y iOS® para el dominio médico. La primera aplicación para la plataforma Android™ es para comunicación clínica y la segunda aplicación para iOS® es para la localización de entidades médicas.

4.1 Caso de estudio 1: Generación de aplicación para comunicación clínica para iOS

Para el primer caso de estudio, se propone la generación de una aplicación móvil para la plataforma iOS® que permita la localización de entidades médicas. Se presenta la interfaz del generador de aplicaciones en donde el usuario expresa las entidades una por una para generar la aplicación móvil. Por último, la aplicación se descarga y se carga como proyecto en el IDE Xcode® con el objetivo de comprobar el funcionamiento.

4.1.1 Generación de aplicación

El proceso de generación de la aplicación inicia cuando el usuario entra al sistema y presiona el botón con la leyenda "¡Iniciar con la construcción!", tal como se muestra en la Figura 4.1.

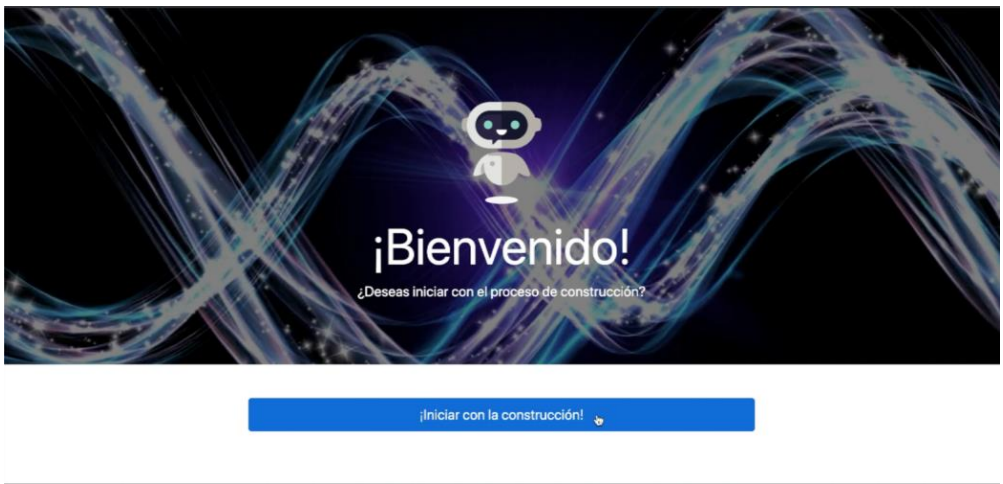

Figura 4.1 Pantalla de inicio del generador de código de aplicaciones móviles

Después, el asistente da la bienvenida y escucha las instrucciones del usuario como se muestra en la Figura 4.2. Después, el usuario expresa la siguiente oración “generar una aplicación en iOS para celular que localice entidades médicas” como se muestra en la Figura 4.3.

Figura 4.2 Agente conversacional pide instrucciones al usuario

Figura 4.3 Usuario dicta instrucciones al agente conversacional

El generador de aplicaciones móviles identifica todos los elementos que determinan la intención del usuario para generar la aplicación, entonces se inicia con la generación de código por parte del generador como se ilustra en la Figura 4.4.

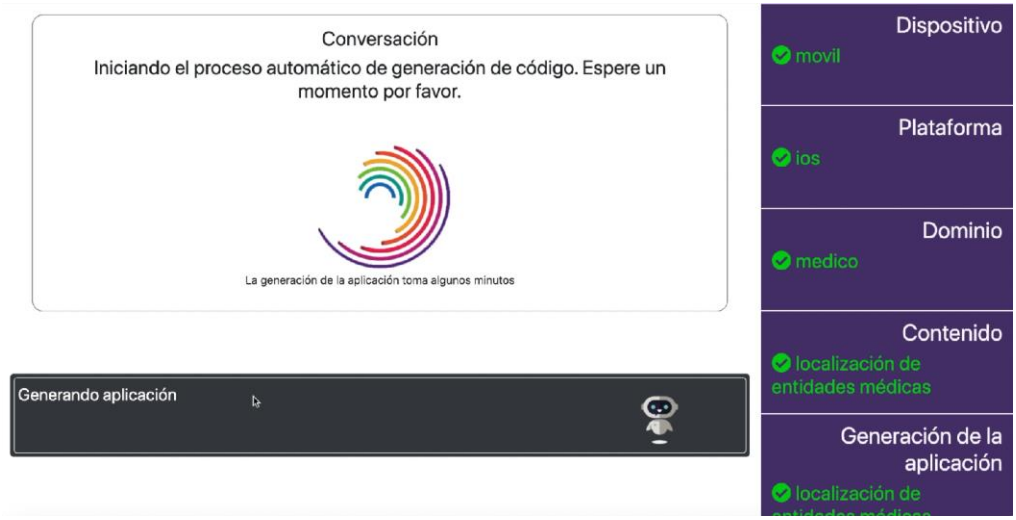

Figura 4.4 Pantalla de generación de la aplicación móvil

Una vez que el proceso de generación de la aplicación móvil se realiza satisfactoriamente el sistema notifica al usuario como se visualiza en la Figura 4.5. Finalmente, en la Figura 4.6 se muestra como el sistema descarga la aplicación y redirige a la pantalla inicial.

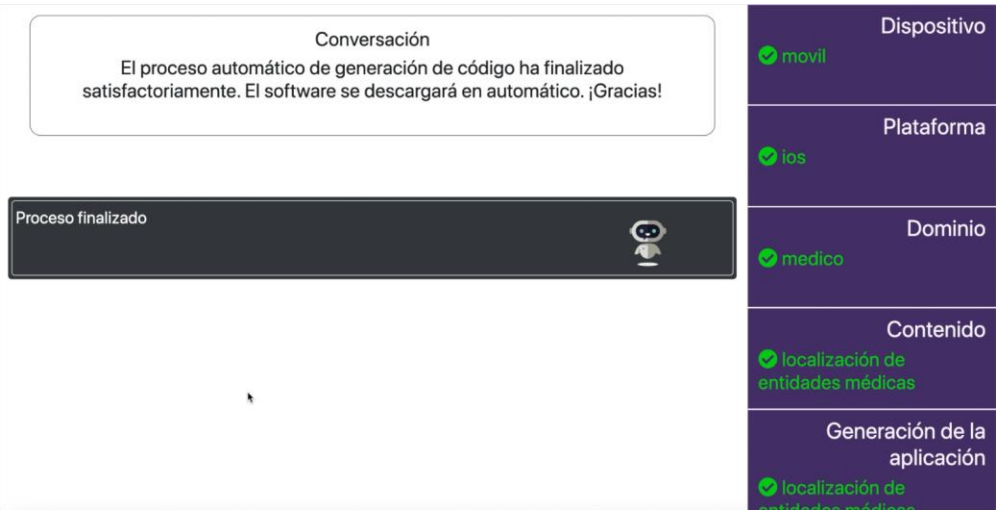

Figura 4.5 Pantalla de proceso de generación finalizado

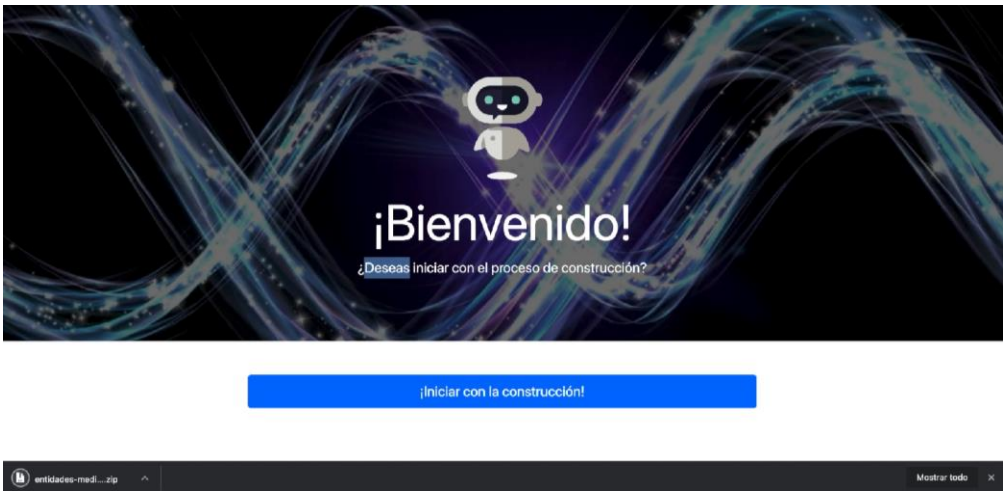

Figura 4. 6 Descarga de la aplicación móvil que se generó

Una vez que se descargó la aplicación y se descomprime el proyecto que se generó, la Figura 4.7 ilustra la estructura del proyecto. Posteriormente se carga desde Xcode® como se ilustra en la Figura 4.8, posteriormente, se ejecuta la aplicación en un emulador como se muestra en la Figura 4.9.

Nombre	Fecha de modificación	Tamaño	Clase
Componente Mapa iOS	31 may 2019 13:21	--	Carpeta
AppDelegate.swift	17 may 2019 0:07	2 KB	Swift Source
Assets.xcassets	17 may 2019 0:07	--	Xcode...Catalog
Base.lproj	31 may 2019 13:15	--	Carpeta
Info.plist	17 may 2019 1:34	2 KB	Property List
ViewController.swift	31 may 2019 13:21	3 KB	Swift Source
Componente Mapa iOS.xcodeproj	28 may 2019 17:47	76 KB	Xcode Project

Figura 4.7 Estructura del proyecto para iOS

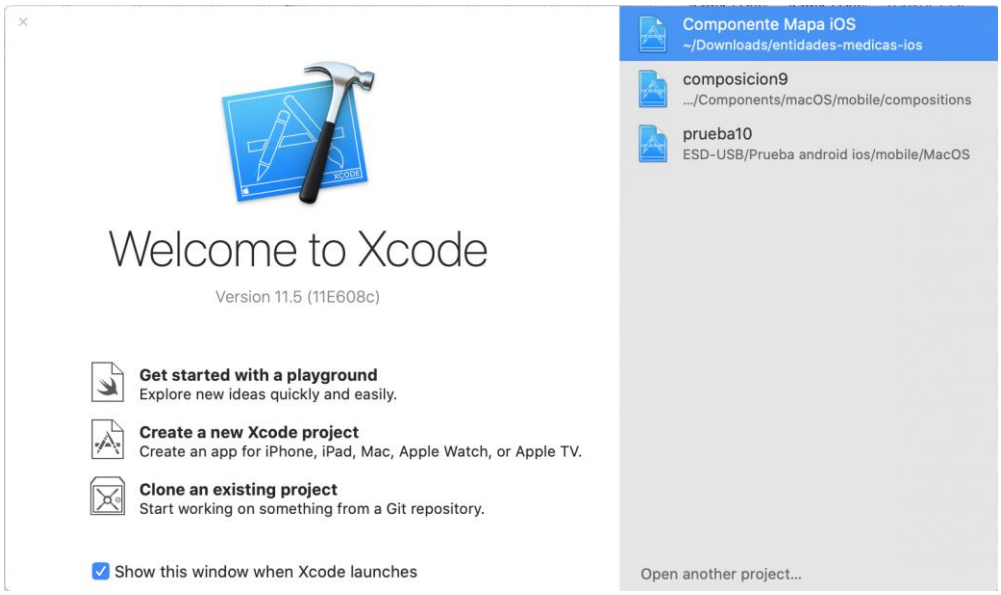

Figura 4. 8 Cargar el proyecto generado a Xcode®

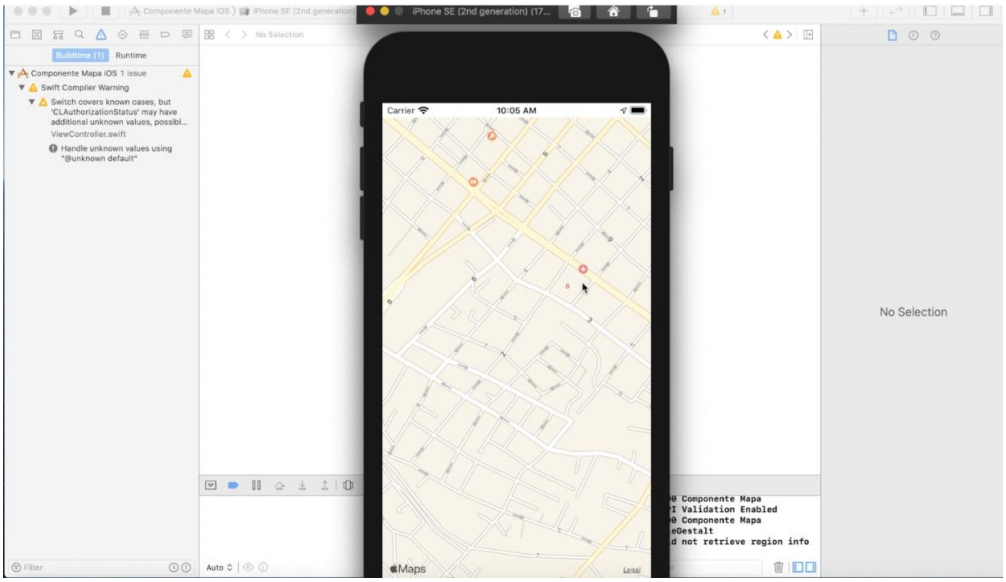

Figura 4.9 Ejecución de aplicación de localización de entidades médicas para iOS®

Como conclusión, el proceso de generación de la aplicación móvil para iOS® se realizó satisfactoriamente, comprobando la funcionalidad del sistema y cumpliendo con el objetivo principal de este trabajo. Utilizando el agente conversacional, el usuario solo tiene que dictar las características de la aplicación en el idioma español estableciendo una conversación fluida con el agente conversacional de acuerdo con el flujo de trabajo establecido, por lo tanto, el usuario no especifica explícitamente los requerimientos funcionales de la aplicación gracias a la identificación y composición de los patrones de diseño de interfaz de usuario que componen dicha

aplicación. Por otra parte, el módulo proporciona un proyecto con todos los archivos correspondientes para que el usuario personalice las características y funcionalidades de la aplicación, por lo tanto, el proyecto está listo para que se despliegue.

4.2 Caso de estudio 2: Generación de aplicación de comunicación clínica para Android

Para el segundo caso de estudio, se propone la generación de una aplicación móvil para la plataforma Android que permita comunicación clínica, también es conocido como chat. Se presenta la interfaz del generador de aplicaciones en donde el usuario da elemento por elemento para generar la aplicación móvil. Por último, la aplicación se descarga y se carga como proyecto en el IDE Android Studio™ con el objetivo de comprobar el funcionamiento correcto del sistema.

4.1.1 Generación de aplicación

La generación de la aplicación inicia cuando el usuario entra al sistema y presiona el botón con la leyenda “¡Iniciar con la construcción!”, tal como de muestra en la Figura 4.10.

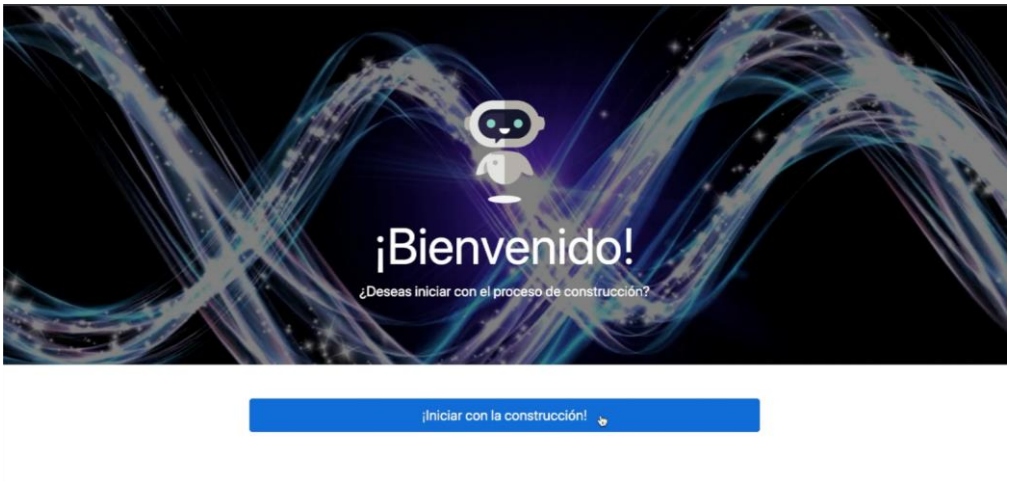

Figura 4.10 Pantalla de inicio del generador de código de aplicaciones móviles

Después, el agente conversacional da la bienvenida y escucha las instrucciones del usuario. A continuación, se muestran los pasos que se realizan para generar la aplicación ya que el usuario desde el inicio no especifica los elementos suficientes para que el agente conversacional detecte la intención del usuario, por lo tanto, el agente conversacional pide los elementos faltantes:

El usuario dice “generar una aplicación”, como se aprecia en la Figura 4.11.

Figura 4.11 Agente conversacional escucha instrucciones del usuario

En la Figura 4.12 se ilustra que el agente conversacional pregunta el tipo de dispositivo y el usuario responde “móvil”.

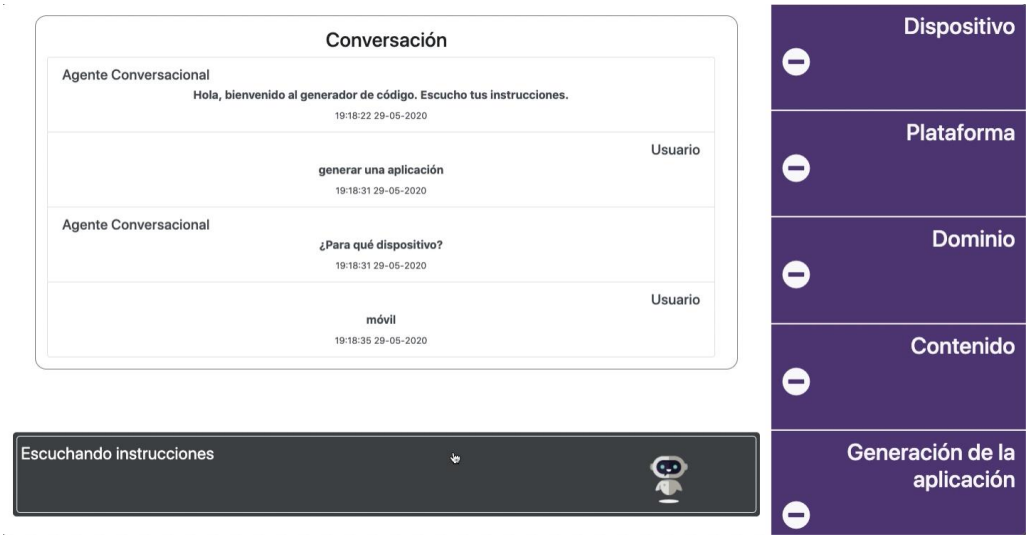

Figura 4.12 Agente conversacional pregunta el tipo de dispositivo

El agente conversacional pregunta la plataforma y el usuario contesta que para Android™, como se muestra en la Figura 4.13.

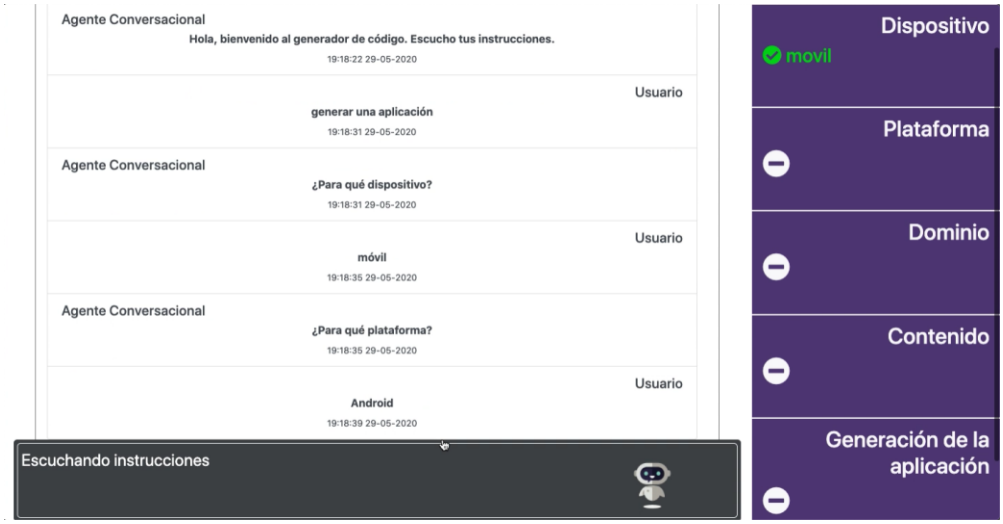

Figura 4. 13 Agente conversacional pregunta la plataforma

Después, el agente conversacional pide el dominio entonces el usuario especifica dominio médico Figura 4.14.

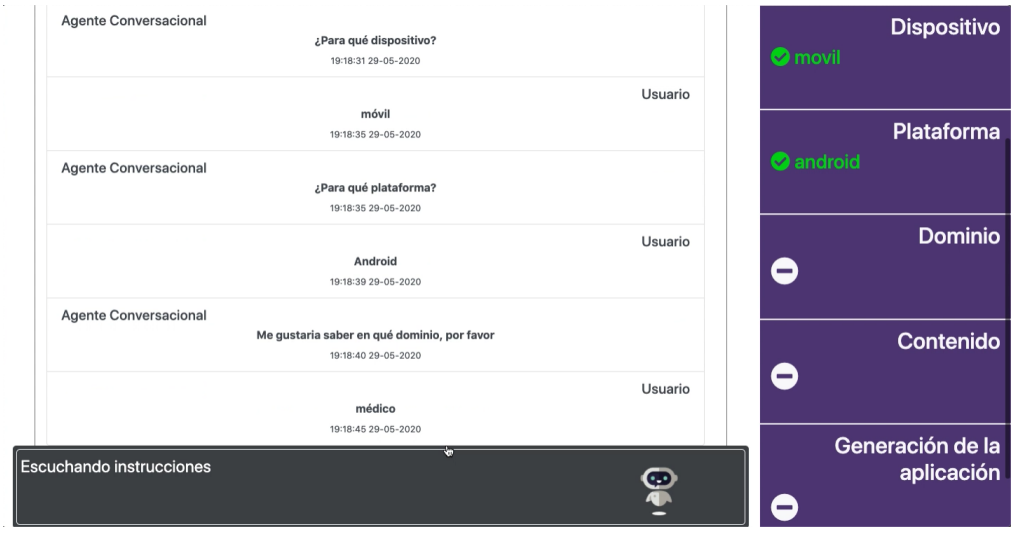

Figura 4.14 Agente conversacional pide dominio de la aplicación

En este punto el agente conversacional tiene más clara la intención del usuario y pregunta el contenido de la aplicación móvil: Comunicación clínica o localización de entidades médicas. El usuario pide comunicación clínica como se muestra en la Figura 4.15.

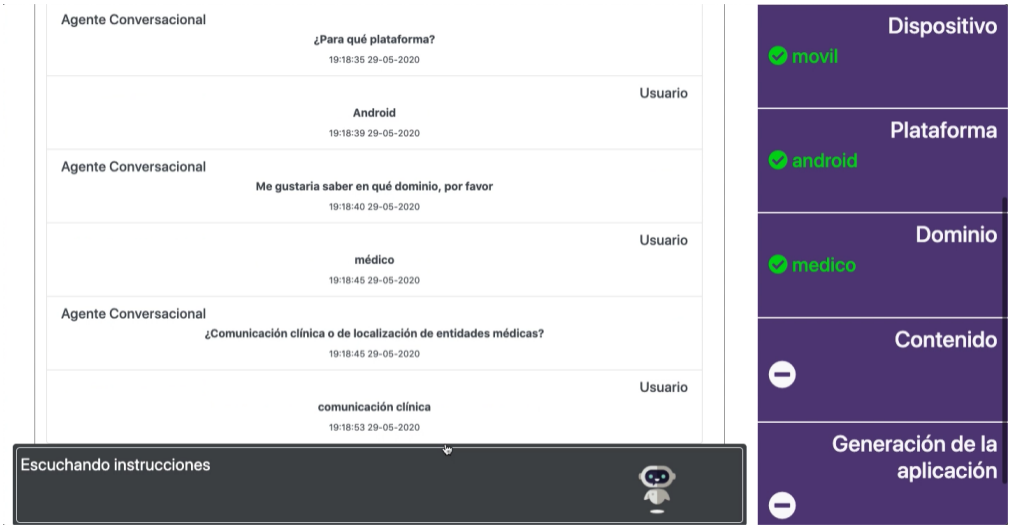

Figura 4.15 Agente conversacional pregunta el contenido

Una vez que el usuario especifico todos los elementos del flujo de trabajo, el sistema inicia el proceso de generación de la aplicación móvil tal y como se visualiza en la Figura 4.16.

Figura 4.16 Inicia el proceso de generación de la aplicación móvil

Finalmente, el agente conversacional notifica que el proceso de generación de código finalizó como se visualiza en la Figura 4.17.

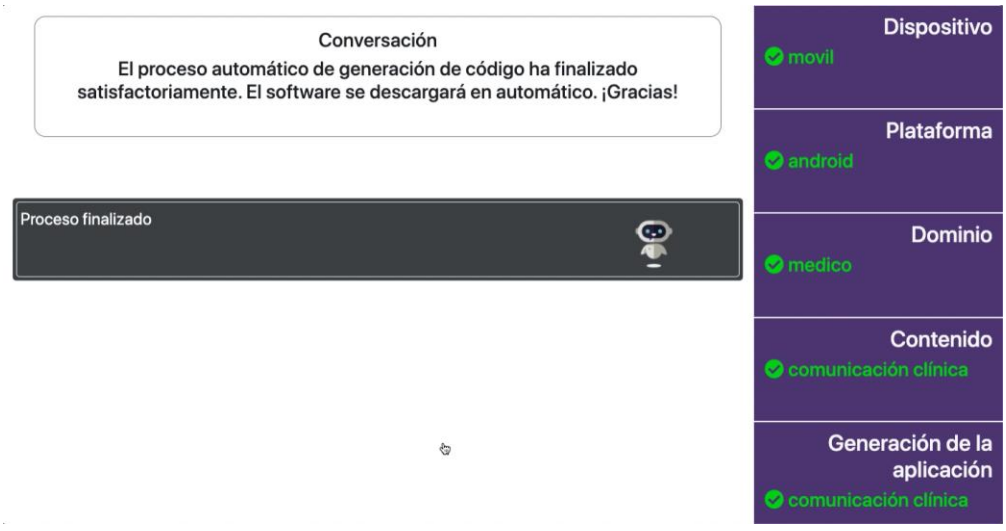

Figura 4.17 Proceso de generación de código finalizado

En la Figura 4.18 se muestra cómo se descarga el proyecto de la aplicación y se redirige a la pantalla principal del sistema.

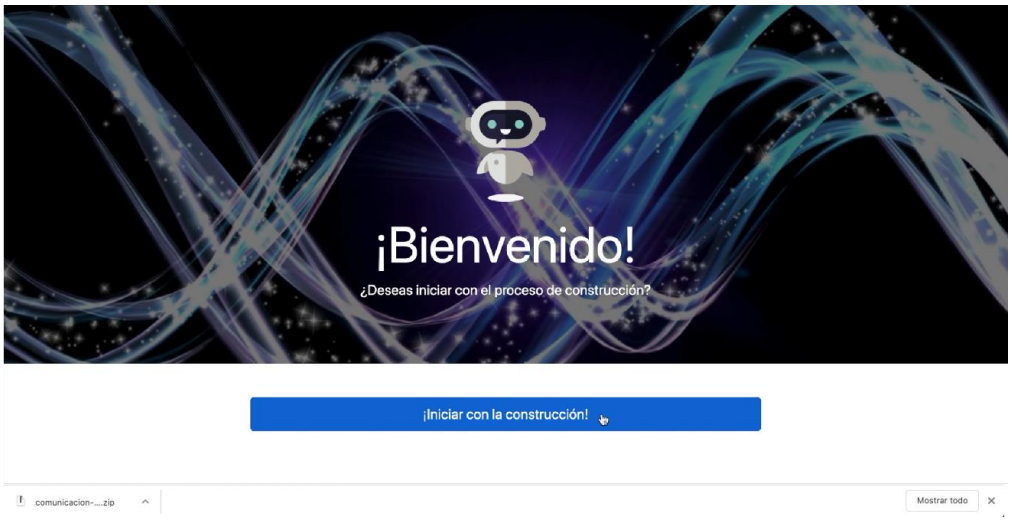

Figura 4. 18 Descarga del proyecto generado

En la Figura 4.19 se ilustra la estructura del proyecto generado después de que se descomprime el archivo que se descargó. Posteriormente se carga el proyecto en Android Studio™ como se ilustra en la Figura 4.20.

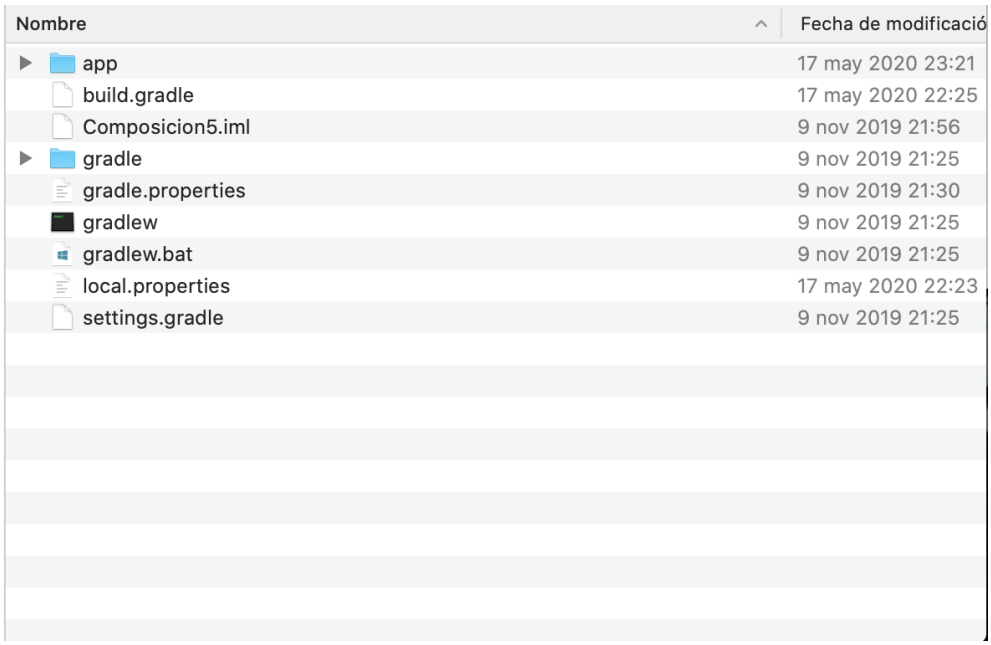

Figura 4.19 Estructura del proyecto generado

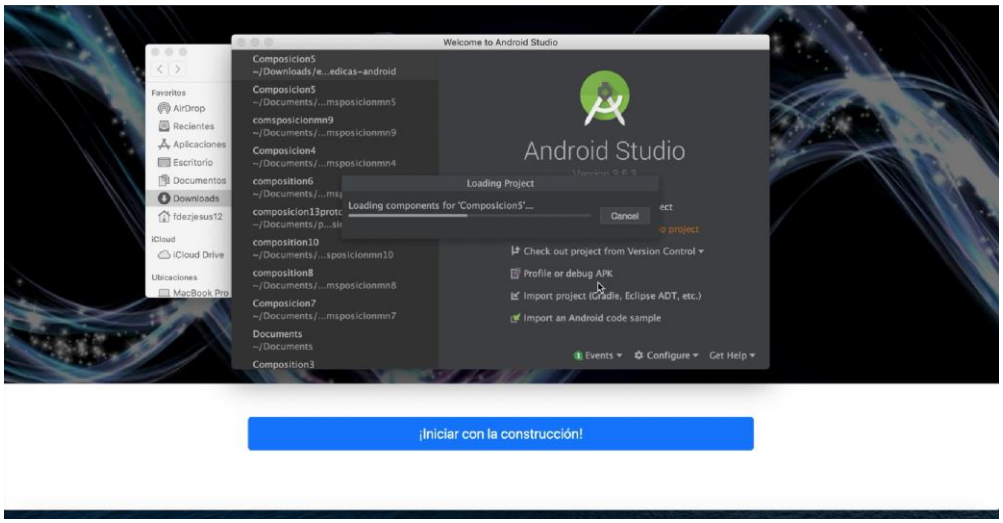

Figura 4. 20 Carga del proyecto generado en Android Studio™

En la Figuras 4.21se ilustra la pantalla inicial de la aplicación de comunicación clínica, en esta pantalla se muestran dos opciones: 1) iniciar sesión y 2) registrarse. La funcionalidad de registro de usuario de muestra en la Figura 4.22. Por otra parte, en la Figura 4.23 se ilustra la pantalla para buscar usuarios registrados. La funcionalidad de enviar un mensaje un usuario se ilustra en la Figura 4.24, mientras que en la Figura 4.25 se muestra la pantalla de conversaciones con los usuarios.

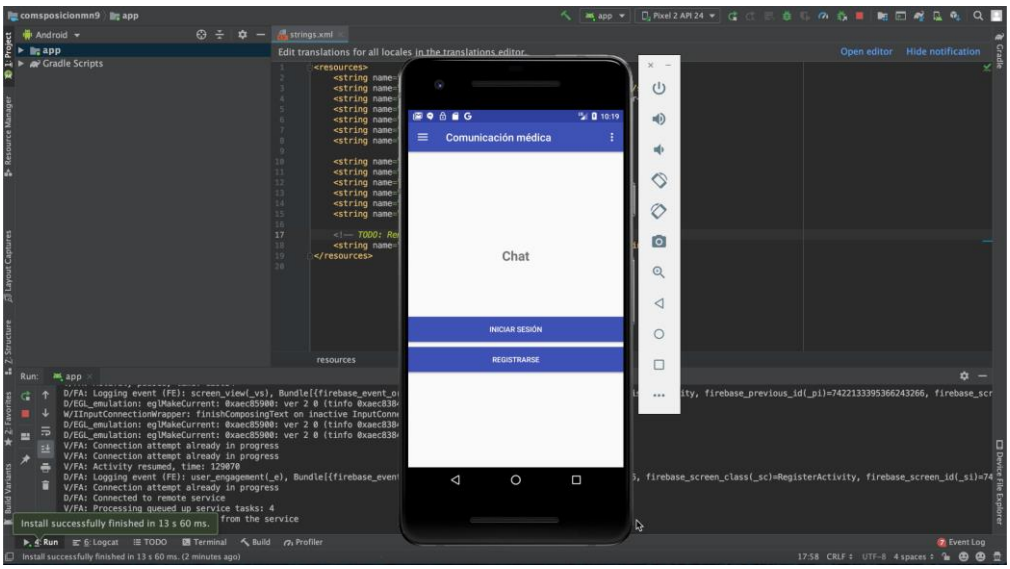

Figura 4. 21 Aplicación comunicación clínica en ejecución, pantalla inicial

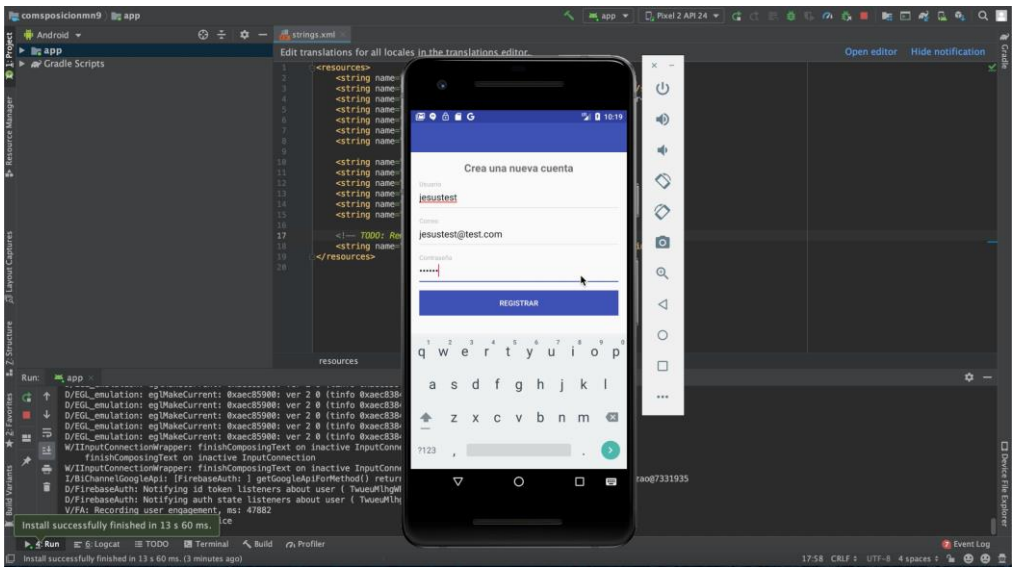

Figura 4. 22 Aplicación comunicación clínica en ejecución, pantalla de registro

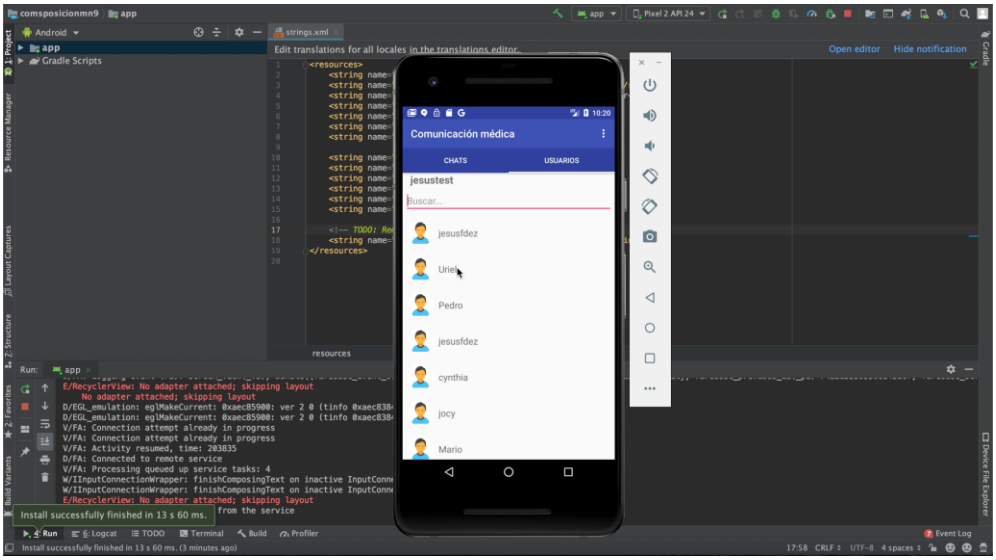

Figura 4.23 Aplicación comunicación clínica en ejecución, pantalla de usuarios

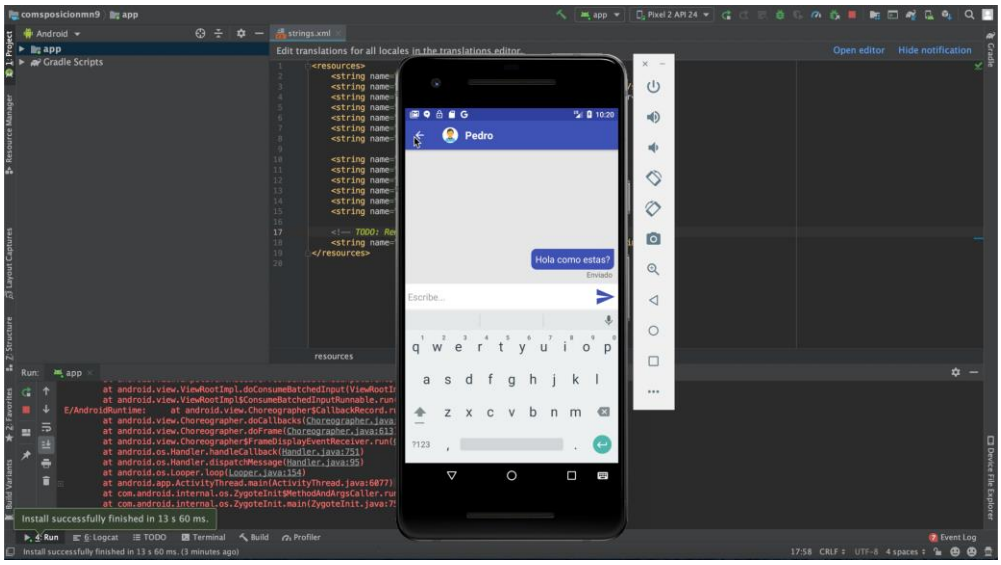

Figura 4.24 Aplicación comunicación clínica en ejecución, pantalla de conversación

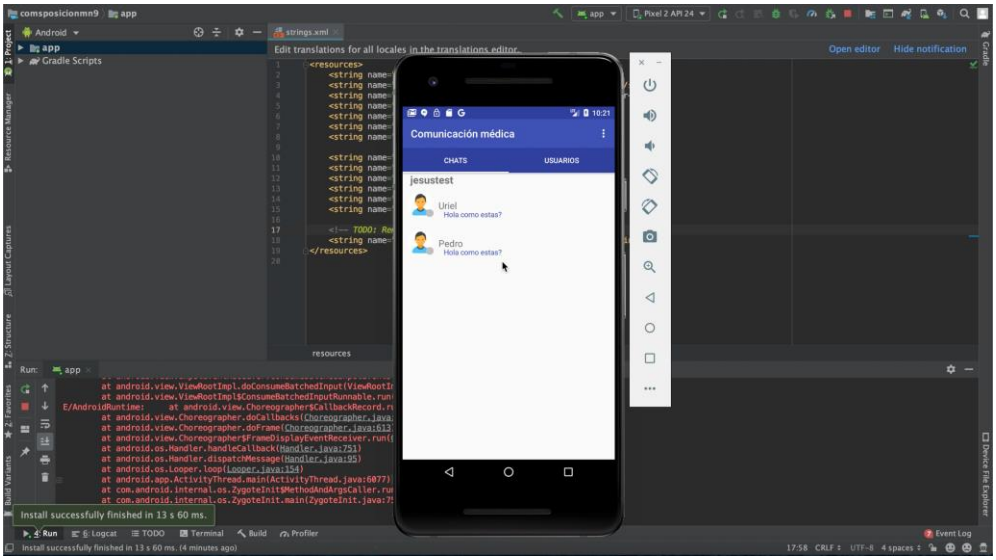

Figura 4.25 Aplicación comunicación clínica en ejecución, pantalla de usuarios

Cabe mencionar que la aplicación de comunicación clínica para dispositivos móviles Android™ se generó correctamente. Además, el asistente virtual cumple con la tarea de pedir cada uno de los elementos de la intención del usuario para continuar con la generación de la aplicación móvil, esto evita errores en el proceso de generación ya que el módulo del asistente virtual funciona como filtro para el módulo de generación de código.

Capítulo V. Conclusiones

5.1 Conclusiones

El proceso de desarrollo de software tiene actividades iterativas que se automatizan para optimizar el tiempo de desarrollo, por lo tanto, las personas involucradas invertirán tiempo y esfuerzo en actividades tales como reglas de negocios, análisis y diseño. El reconocimiento de voz es una interfaz de usuario innovadora para tener una mejor interacción con los usuarios.

En busca de métodos y técnicas para mejorar el proceso de desarrollo de software, este trabajo sugiere un proceso de soporte por un flujo de trabajo para generar proyectos nativos para dos plataformas móviles mediante el uso de reconocimiento de voz. Este enfoque aplica técnicas de ingeniería de software y procesamiento de lenguaje natural que ocultan aspectos técnicos a los usuarios finales mediante una interfaz de usuario amigable. Además, las aplicaciones médicas han adquirido gran importancia hoy en día, especialmente en el cuidado de la salud. Debido a esto, es importante investigar el uso de estas aplicaciones para desarrollar interfaces gráficas de usuario apropiadas para este dominio.

Por otra parte, los agentes conversacionales son el comienzo de una interfaz entre los humanos y la Inteligencia Artificial general. La arquitectura y las herramientas que se analizaron fueron clave para el desarrollo de este trabajo ya que los casos de estudio demuestran la capacidad del agente conversacional para determinar la intención del usuario sin importar la estructura gramatical. El flujo de trabajo establecido para que el agente conversacional identifique cada característica de la aplicación evita que la generación de código intermedio sea un proceso complejo ya que solo es necesario pasar los valores identificados en el agente conversacional sin realizar una validación extra en el proceso de construcción del código intermedio (archivo XML).

Gracias a estas nuevas técnicas de desarrollo de software, se logra la generación de proyectos base para las plataformas Android y IOS y con esto los desarrolladores tienen la facilidad de personalizar las características del proyecto base, tales como: colores, textos, tamaños, por mencionar alunas características. Sin embargo, el proyecto que se genera está listo para que se despliegue en un ambiente de producción siempre y cuando se considere la configuración pertinente en cuanto a infraestructura. Para los usuarios y los desarrolladores es importante siempre contar con herramientas que faciliten el desarrollo de software y permitan además una interacción mucho mayor con el entorno o ambiente que se manipula.

5.2 Recomendaciones

Al revisar los puntos observados, desarrollados, y generados del proceso de desarrollo de aplicaciones determinan los siguientes puntos especiales para atender las siguientes recomendaciones:

Es claro mencionar que este trabajo se enfocó al dominio médico, por tal motivo, el generar nuevos patrones compuestos que delimiten nuevas funcionalidades que aprovechen resolver algunos problemas específicos como integrar más patrones para hacer una aplicación más completa al generar una nueva aplicación, a la par, da la oportunidad de trabajar con otros dominios un ejemplo de ello es el dominio educativo, ya que cuenta con otro tipo de patrones aún por conocer y analizar.

Para el procesamiento de lenguaje natural se utilizó la plataforma Dialogflow, por lo tanto, es importante considerar los cambios que se realicen a la plataforma a lo largo de su evolución para evitar incompatibilidades al consumir su servicio WEB. Por otra parte, se debe considerar un mecanismo de seguridad para mantener las sesiones separadas al consumir el servicio WEB de la plataforma con el objetivo de evitar colapsos y confusiones de la conversación de la plataforma y los usuarios.

Es importante mantener los componentes de las aplicaciones móviles actualizados con el objetivo de evitar errores de compatibilidad y/o de funcionalidad, por ello es

necesario realizar seguimiento de la actualización de las versiones más recientes para las plataformas Android™ y iOS®.

Productos académicos

En este apartado, se presentan los trabajos derivados de este proyecto de investigación.

Artículo

Jesús Fernández-Avelino, Giner Alor-Hernández, Mario Andrés Paredes-Valverde, Lisbeth Rodríguez-Mazahua, María Antonieta Abud-Figueroa. A Process for Automatic Generation of Medical Mobile Applications using Voice Recognition. **Estatus:** Publicado.

Capítulo

Jesús Fernández-Avelino, Giner Alor-Hernández, Mario Andrés Paredes-Valverde. Developing chatbots for supporting health self-management. **Estatus:** Aceptado.

Proyecto de investigación

Generación automática de interfaces de usuario de aplicaciones médicas para dispositivos móviles mediante técnicas de Procesamiento de Lenguaje Natural. Convocatoria Proyectos de Investigación Científica 2020 del Tecnológico Nacional de México. **Rol:** Colaborador. **Clave:** 7651.20-P. **Estatus:** En proceso.

Estancia académica

Instituto Tecnológico Superior de Teziutlán. Teziutlán, Pue. Periodo: 4 de mayo al 4 de junio del 2020. Dr. Mario Andrés Paredes Valverde.

Referencias

[1] R. S. Pressman, *Ingeniería del software: un enfoque práctico*. 2013.

[2] C. M. Zapata y J. J. Chaverra, «A CONCEPTUAL APPROACH TO AUTOMATIC GENERATION OF CODE», *Rev. EIA*, vol. 7, n.º 13, pp. 143-154, sep. 2013.

[3] J. Peckham y B. MacKellar, «Generating code for engineering design systems using software patterns», *Artif. Intell. Eng.*, vol. 15, n.º 2, pp. 219-226, abr. 2001.

[4] S. Ramkarthik. and C. Zhang, «Generating Java Skeletal Code with Design Contracts from Specifications in a Subset of Object Z», en *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06)*, 2006, pp. 405-411.

[5] S. P. Overmyer, B. Lavoie, y O. Rambow, «Conceptual Modeling Through Linguistic Analysis Using LIDA», en *Proceedings of the 23rd International Conference on Software Engineering*, Washington, DC, USA, 2001, pp. 401–410.

[6] H. M. Harmain y R. Gaizauskas, «CM-Builder: an automated NL-based CASE tool», en *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*, 2000, pp. 45-53.

[7] E. Buchholz y A. Raab-Düsterhöft, «Using Natural Language for Database Design», en *KRDB*, 1994.

[8] Y. Wand y R. WEBer, «Research Commentary: Information Systems and Conceptual Modeling—A Research Agenda», *Inf. Syst. Res.*, vol. 13, n.º 4, pp. 363-376, dic. 2002.

[9] C. M. Zapata y J. J. Chaverra, «A CONCEPTUAL APPROACH TO AUTOMATIC GENERATION OF CODE», *Rev. EIA*, n.º 13, pp. 155-169, jul. 2010.

[10] «Conversational Agent», *DeepAI*. [En línea]. Disponible en: https://deepai.org/machine-learning-glossary-and-terms/conversational-agent. [Accedido: 15-mar-2019].

[11] «Natural Language Processing», *DeepAI*. [En línea]. Disponible en: https://deepai.org/machine-learning-glossary-and-terms/natural-language-processing. [Accedido: 17-mar-2019].

[12] M. Rouse, «What is natural language understanding (NLU)? - Definition from WhatIs.com», *SearchEnterpriseAI*. [En línea]. Disponible en: https://searchenterpriseai.techtarget.com/definition/natural-language-understanding-NLU. [Accedido: 17-mar-2019].

[13] Z. Rehman y S. Kifor, «Teaching Natural Language Processing (PLN) Using Ontology Based Education Design», *Balk. Reg. Conf. Eng. Bus. Educ.*, vol. 1, n.º 1, nov. 2015.

[14] S. Sekine y E. Ranchhod, *Named Entities: Recognition, classification and use*. John Benjamins Publishing, 2009.

[15] C.-J. Lee, S.-K. Jung, K.-D. Kim, D.-H. Lee, y G. G.-B. Lee, «Recent Approaches to Dialog Management for Spoken Dialog Systems», *J. Comput. Sci.*

Eng., vol. 4, n.º 1, pp. 1-22, 2010.
[16] «Dialogflow | Google Cloud». https://cloud.google.com/dialogflow/?hl=en (accedido ene. 26, 2020).
[17] «What are User Interface (UI) Design Patterns?», *The Interaction Design Foundation*. [En línea]. Disponible en: https://www.interaction-design.org/literature/topics/ui-design-patterns. [Accedido: 18-mar-2019].
[18] L. N. Sánchez-Morales, G. Alor-Hernández, R. Miranda-Luna, V. Y. Rosales-Morales, y C. A. Cortes-Camarillo, «Generation of User Interfaces for Mobile Applications Using Neuronal Networks», en *New Perspectives on Applied Industrial Tools and Techniques*, J. L. García-Alcaraz, G. Alor-Hernández, A. A. Maldonado-Macías, y C. Sánchez-Ramírez, Eds. Cham: Springer International Publishing, 2018, pp. 211-231.
[19] C. A. Cortes-Camarillo, V. Y. Rosales-Morales, L. N. Sanchez-Morales, G. Alor-Hernández, y L. Rodríguez-Mazahua, «Atila: A UIDPs-based educational application generator for mobile devices», en *2017 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, 2017, pp. 1-7.
[20] S. Modak, S. Vikmani, S. Shah, y L. Kurup, «Voice driven dynamic generation of WEBpages», en *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, 2016, pp. 1-4.
[21] T. Erić, S. Ivanović, S. Milivojša, M. Matić, y N. Smiljković, «Voice control for smart home automation: Evaluation of approaches and possible architectures», en *2017 IEEE 7th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2017, pp. 140-142.
[22] and y and, «BadVoice: Soundless voice-control replay attack on modern smartphones», en *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2016, pp. 882-887.
[23] J. Vittone y J. Cuello, «Definiendo la propuesta», en *Diseñando apps para móviles*, 1.ª ed., 2013, pp. 66-79.
[24] T. Beltramelli, «Pix2Code: Generating Code from a Graphical User Interface Screenshot», en *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, New York, NY, USA, 2018, pp. 3:1–3:6.
[25] S. Patil, A. Abhigna, A. -, D, y P, «Voice Controlled Robot Using Labview», en *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, 2018, pp. 80-83.
[26] H. Terada, K. Makino, H. Nishizaki, E. Yanase, T. Suzuki, y T. Tanzawa, «Positioning Control of a Micro Manipulation Robot Based on Voice Command Recognition for the Microscopic Cell Operation», en *Advances in Mechanism Design II*, 2017, pp. 73-79.
[27] M. Sidiq, W. T. A. Budi, y S. Sa'adah, «Vomma: Android application launcher using voice command», en *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015, pp. 49-53.
[28] S. L. d Costa, V. V. G. Neto, y J. L. d Oliveira, «A User Interface Stereotype to Build WEB Portals», en *2014 9th Latin American WEB Congress*, 2014, pp. 10-18.
[29] A. Nedzved, I. Gurevich, Yu. Trusova, y S. Ablameyko, «Software development technology with automatic configuration to classes of image processing problems», *Pattern Recognit. Image Anal.*, vol. 23, n.º 2, pp. 269-277,

jun. 2013.
[30] M. Paschou, E. Sakkopoulos, y A. Tsakalidis, «easyHealthApps: e-Health Apps Dynamic Generation for Smartphones & Tablets», *J. Med. Syst.*, vol. 37, n.º 3, p. 9951, may 2013.
[31] S. Tabibian, «A voice command detection system for aerospace applications», *Int. J. Speech Technol.*, vol. 20, n.º 4, pp. 1049-1061, dic. 2017.
[32] L. Furtado, A. Marques, N. Neto, M. Mota, y B. Meiguins, «IVOrpheus 2.0 - A Proposal for Interaction by Voice Command-Control in Three Dimensional Environments of Information Visualization», en *Human Interface and the Management of Information: Information, Design and Interaction*, 2016, pp. 347-360.