Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Modelo de navegación web usando voz, basado en contexto

TESIS

Que presenta

Citlalli Selene Avalos Montiel

para obtener el Grado de

Maestra en Ciencias

en Computación

Directores de la Tesis

Dr. José Guadalupe Rodríguez García

Dr. Sergio Víctor Chapa Vergara

Ciudad de México **Diciembre de 2023**

Resumen

La Web está constituida por documentos de hipertexto, enlaces, multimedia, entre otros elementos, que a través de navegadores pueden ser visualizados para aprovechar la información que contienen. Para encontrar información útil, un usuario debe realizar un proceso de búsqueda según sus intereses. Dicho proceso normalmente consiste en definir la búsqueda, filtrar, seleccionar, visualizar, analizar una o varias páginas web y decidir qué contenido le es más útil. Además, este proceso requiere la interacción del usuario con el navegador, mediante hardware como teclados, pantallas, pantallas táctiles, micrófonos, ratón, entre otros. Adicionalmente, se requiere de la atención visual y ciertas habilidades por parte del usuario. La navegación web puede ser una tarea compleja para algunas personas debido a diferentes razones, por ejemplo, el desconocimiento en el uso de TICs, falta de habilidad o alguna discapacidad. Estas dificultades provocan que las personas dejen de realizar esta actividad. En este proyecto se propone un modelo para navegación con base en el contexto del usuario y de las páginas web. La entrada al sistema es la petición del usuario en lenguaje natural, por voz. La salida es el contenido de mayor beneficio para el usuario, según sus intereses, en formato audible y/o gráfico. Lo que se busca es facilitar la navegación web y contribuir en la disminución de la brecha digital entre la población que, por diversas razones, tiene dificultades para navegar en la Web.

Palabras clave: Brecha digital, contexto, lenguaje natural, navegación web, procesamiento de voz.

Abstract

The Web contains hypertext documents, links, multimedia, among other elements, which can be viewed with the use of web browsers to take advantage of the information that these elements contain. In order to find useful information, a user must perform a search process according to his interests. This process usually consists of defining the search, filtering, selecting, viewing, and analysing one or several web pages, and deciding which content is most useful to the user. Also, this process requires the interaction of the user with the web browser through hardware such as keyboards, screens, touch screens, microphones, mouse, among others. In addition, it requires visual attention and certain user skills. Web browsing can be a difficult task for some people due to different reasons, for example, lack of knowledge in the use of ICT, lack of ability or some disability, causing people to stop doing this activity. In this project, a navigation model is proposed based on the context of the user and the web pages. The input to the system is the user's request in natural language, by voice. The output is the content that best fits the user's interests, in audible and/or graphic format. The aim of this project is to facilitate web browsing and contribute to reducing the digital divide among the population that, for various reasons, has difficulties for navigating the Web.

Keywords: Digital divide, context, natural language, web browsing, voice processing.

Agradecimientos

Agradezco al CONACYT por el apoyo económico brindado durante la realización de este proyecto.

Agradezco al CINVESTAV, esta institución de enorme calidad, que me brindó todo el apoyo durante mi estancia.

Agradezco a mis asesores de tesis, el Dr. José Guadalupe Rodríguez García y el Dr. Sergio Víctor Chapa Vergara, por su orientación y los conocimientos invaluables que me brindaron para llevar a cabo esta investigación, y sobre todo su gran paciencia para esperar a que este trabajo pudiera llegar a su fin.

Agradezco a los miembros del jurado, por las valiosas contribuciones que hicieron al trabajo final y por el tiempo que dedicaron para revisarlo, aun a pesar de las actividades que los ocupan.

Agradezco a los profesores del programa de maestría que hacen posible el conocimiento en las aulas del Departamento de Computación.

A todos los que alguna vez han compartido sus conocimientos para enriquecernos todos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

A lo largo de la historia, los seres humanos han buscado facilitar y optimizar sus actividades cotidianas, dando origen a diversas herramientas. Una de las herramientas que ha acompañado al hombre en los últimos años es la computadora. En las primeras tres generaciones de la computación, la interacción entre el hombre y la máquina era compleja, debido a que requería de que el usuario fuera experto en cuanto la estructura de la computadora y contara con conocimientos técnicos de su funcionamiento. A partir de la cuarta generación, se diseñaron computadoras personales que contaban con interfaces amigables pensadas para el usuario común, que no necesariamente tiene conocimientos técnicos en computación.

1.1 Motivación

En la sociedad de la información, la tecnología facilita las actividades de millones de individuos en todo el mundo. La tecnología ofrece soluciones a problemas cotidianos, académicos, culturales, sociales y económicos, por mencionar algunos, a través de la creación, acceso, manejo e intercambio de contenido digital [2]. Según Castells [3], las principales actividades económicas, sociales, políticas y culturales en todo el planeta se están estructurando por medio de Internet, en donde la población que permanece al margen de dichas

redes, vive una forma de exclusión muy grave. Es decir, la Internet y la Web son recursos indispensables para la sociedad de la información. En México, según las estadísticas de la Encuesta Nacional sobre Disponibilidad y Uso de Tecnologías de la Información en los Hogares (ENDUTIH) del año 2022, el 78.6 % de la población con edad mayor a 6 años, es usuaria de Internet, donde los principales usos son la comunicación y la búsqueda de información [4]. Estos datos revelaron un aumento de 3 puntos porcentuales en la población usuaria de Internet con respecto al año 2021.

La brecha digital divide a la población entre las personas que tienen acceso a las tecnologías de la información y la comunicación (TICs) y las que no. Según Van Dijk [5], la brecha digital también es causada por las motivaciones, actitudes y habilidades de los usuarios hacia las TICs, y estas causas tienen el mismo efecto de desigualdad en oportunidades y participación en la sociedad de la información. Actualmente, se está popularizando el uso de las TICs a través del uso de modelos visuales e interfaces por voz, que permiten realizar algunas actividades básicas en la web, por medio de comandos preestablecidos. Esta popularización contribuye en la disminución de la brecha digital en cuanto a las habilidades requeridas para la navegación web.

A pesar de este aumento en las estadísticas y los avances tecnológicos en el área de HCI, sigue existiendo una brecha digital que afecta el desarrollo integral, material y humano de algunos grupos de la sociedad, como adultos mayores, personas con analfabetismo, personas con analfabetismo digital y personas con alguna discapacidad, por ejemplo: motriz, adquirida por amputación y visual [6]. Con el fin de contribuir a la disminución de la brecha digital, es necesario desarrollar herramientas y/o proponer soluciones que permitan el uso de las TICs de una forma más natural y simple, por ejemplo, que una búsqueda de información sea como conversar con alguien.

1.2 Contexto de la investigación

De acuerdo con el sistema de clasificación de computación del la ACM [1], este proyecto se encuentra dentro de la clasificación **Sistemas de información** y **Computación centrada en el ser humano**, como categorías de mayor nivel y como categorías más específicas, el proyecto se encuentra dentro de las clasificaciones: interfaces de búsqueda, interfaces de lenguaje natural, interfaces gráficas de usuario y métodos de diseño y evaluación de HCI (ver Figura 1.1).

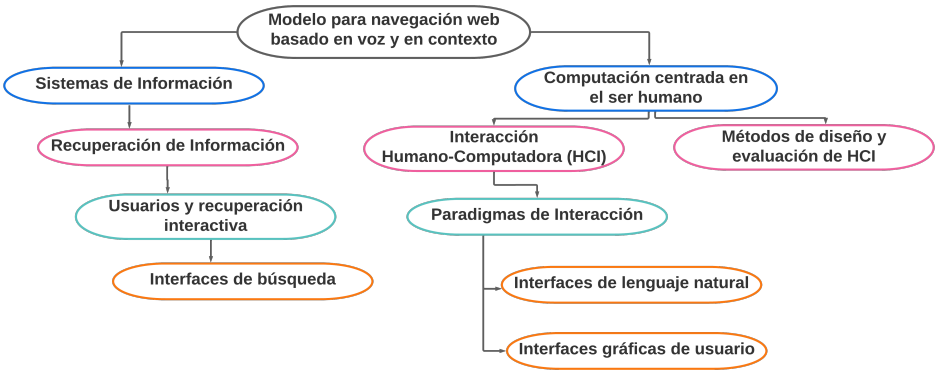


Figura 1.1: Clasificación ACM 2012

1.3 Planteamiento del problema

Según Mark Weiser "*... la computación ubicua hace que la tecnología ayude a la gente a vivir en el mundo real ...*"[7]. Sin embargo, en la actualidad, la mayoría de personas que trabajan de manera estrecha con TICs, usan interfaces con modelos visuales que demandan atención hacia los dispositivos para llevar a cabo sus tareas y, a veces, requieren ciertas habilidades para realizar una actividad específica, por ejemplo, navegar en la Web.

Por un lado, en los últimos años, se ha popularizado el uso de asistentes virtuales como Alexa de Amazon, Asistente de Google, Siri, entre otros. Éstos reciben peticiones de los usuarios mediante comandos de voz, y permiten realizar acciones relacionadas con la navegación web como reproducir música, realizar búsquedas, crear listas de compras,

[1] https://dl.acm.org/ccs

controlar aparatos electrónicos "inteligentes", establecer alarmas entre otras actividades básicas, que además dependen de las aplicaciones que se tienen instaladas en los dispositivos [8, 9, 10, 11].

Sin embargo, en la navegación web, sigue predominando el modo visual para la entrega y selección de contenido, debido a que para esta actividad se manejan documentos de hipertexto y un navegador o "visualizador" en el dispositivo del usuario. En el navegador, el usuario debe seleccionar los enlaces a los que desee dirigirse, y de la misma forma el usuario debe leer e interpretar qué páginas o sitios web son más útiles y descartar los que no le interesan.

Por otro lado, se ha desarrollado tecnología de asistencia dirigida a personas con discapacidad visual para navegar en la Web, como [12, 13, 14, 15, 16] que de forma general tienen las siguientes características.

- Lectores de pantalla que reciben instrucciones mediante combinaciones de teclas, gestos en pantallas o comandos específicos de voz y entregan contenido a los usuarios mediante voz o braille.

- Extensiones de navegador que trabajan con el lector de pantalla JAWS [17, 18] que ayudan a encontrar información considerada relevante mediante análisis estadístico o semántico de la página web. El objetivo principal es reducir el tiempo empleado en la navegación.

- Estos trabajos fueron desarrollados para ser empleados en entornos compuestos con computadoras de escritorio o laptops y periféricos de entrada como teclados en braille, micrófono, ratón, entre otros, para que el usuario interactúe con el sistema.

Los asistentes virtuales y las herramientas de asistencia, para personas con discapacidad visual, proveen una interfaz que facilita a los usuarios la interacción con las TICs. Sin embargo, comparten una limitante con los navegadores web más populares: no garantizan recuperar información, de manera efectiva, con respecto a las intenciones e intereses del

usuario. Para obtener resultados de mayor beneficio, los usuarios deben hacer uso de estrategias de búsqueda, por ejemplo: establecer filtros, pre-configurar el navegador web, usar operadores booleanos y caracteres especiales, entre otras. Estas estrategias pueden ser complicadas, por lo que una herramienta de navegación web que conozca el contexto del usuario y de las páginas web, permitirá encontrar coincidencias entre ambos contextos y filtrar resultados de real interés para el usuario [19].

Como se menciona en la Sección 1.1, una de las causas por las que se presenta la brecha digital es porque los usuarios no poseen las habilidades, motivaciones o actitudes necesarias para operar las TICs. Esto puede ser por desconocimiento o por la incapacidad de llevar a cabo una acción en un determinado momento. Por lo anterior, resultaría útil modelar una herramienta en cuya interfaz se fusionen la interpretación de instrucciones de los asistentes virtuales y la entrega de contenido de los lectores de pantalla y tecnología similar. Además, que la herramienta permita entregar contenido orientado a los intereses del usuario, con base en las coincidencias entre el contexto del usuario y de los sitios web, en formato gráfico o audible. Y de esta forma, contribuir en la disminución de la brecha digital, facilitando la navegación web.

1.4 Objetivos

General

Modelar una herramienta para la navegación web por voz, basada en contexto del usuario y de los recursos web, que permita presentar contenido de real interés para el usuario en formato audible o gráfico.

Específicos

1. Identificar los requerimientos necesarios para llevar a cabo la navegación web contextual por voz.

2. Definir qué funcionalidades y procesos de las propuestas existentes tienen relación

con la navegación web contextual por voz.

3. Proponer el diseño conceptual del modelo de navegación web contextual por voz, que cumpla con los requerimientos.

4. Definir las herramientas que permitan implementar un prototipo con base en el modelo.

5. Implementar un prototipo basado en el modelo.

6. Realizar pruebas al prototipo para validar el modelo.

7. Evaluar los resultados.

1.5 Metodología

De forma general, la metodología sigue las etapas de documentación, análisis, propuesta de solución, desarrollo, pruebas y resultados. La metodología de desarrollo para este trabajo de tesis sigue una serie de actividades que a continuación se enumeran.

1. Revisar el estado del arte.

2. Analizar las funcionalidades y procesos de las propuestas existentes que vayan de acuerdo con el objetivo general de la investigación.

3. Definir y analizar los requerimientos funcionales y no funcionales para el modelo de una herramienta de navegación web por voz, basado en contexto y entrega de contenido en formato gráfico o audible.

4. Definir el diseño conceptual del modelo.

5. Analizar, probar y elegir herramientas útiles para el desarrollo de un prototipo.

6. Implementar, probar y depurar cada uno de los componentes de la arquitectura del prototipo de forma individual.

7. Integrar los componentes del prototipo.
8. Definir experimentos, pruebas y espacio de pruebas.
9. Realizar pruebas al prototipo, para validar el modelo.
10. Evaluar los resultados.

1.6 Organización de la tesis

La tesis que se presenta en este documento consta de seis capítulos y las conclusiones. En el capítulo dos se presenta el marco teórico que da fundamento a este trabajo de tesis. En el capítulo tres se hace una revisión de los trabajos relacionados con esta propuesta, lo que nos permite obtener una visión más clara de este trabajo. En el capítulo cuatro se presenta la propuesta de solución. En el capítulo cinco se presenta el desarrollo de la solución, en el seis se presentan las pruebas y el análisis de resultados. Finalmente, se presentan las conclusiones y trabajos futuros.

Capítulo 2

Estado del arte

En este capítulo, se exploran los fundamentos teóricos esenciales para comprender el contexto y desarrollo de la tesis. Además, se lleva a cabo una revisión de los trabajos relacionados con la investigación. Se inicia con los antecedentes del grupo de trabajo, seguido por una presentación de los trabajos del estado del arte. Finalmente, se ofrece una discusión sobre el análisis de proyectos relevantes.

2.1 Marco teórico

2.1.1 Cómputo Ubicuo

Mark Weiser fue un visionario y padre de lo que se conoce como Cómputo Ubicuo. Weiser reconoce a esta área como la parte fundamental en la tercera ola de la computación. Donde el desafío es crear entornos tecnológicos compuestos de dispositivos con distintas funcionalidades, conectados entre sí, para manejar información y facilitar las actividades del hombre en el mundo real [7]. En el artículo "*El futuro de la computación ubicua en el campus*", publicado en 1998 por Weiser [20], las tres olas de la computación se describen como sigue.

- **Primera ola**

  Esta ola se presentó en la era del *mainframe*, donde existieron equipos de grandes dimensiones, instaladas en salas informáticas de universidades y empresas. Los equipos de cómputo eran aprovechados por estudiantes, investigadores y administradores, sin embargo, estos debían especializarse en el uso del equipo. Y en palabras de Weiser, en la primera ola "*... las computadoras eran raras, caras y requerían mantenimiento*" [20].

- **Segunda ola**

  Esta ola nació con el surgimiento de las computadoras personales, donde las computadoras eran más pequeñas y económicas, en comparación con las existentes en la primera ola, y además contaban con un monitor. Más personas lograron tener acceso a una computadora, incluso un usuario podía tener acceso a varias computadoras y la relación de usuario con la computadora era personal. Las computadoras personales aún eran difíciles de manejar, tenían poca o nula seguridad, eran ruidosas, pero ya eran necesarias para los usuarios en sus actividades diarias.

- **Tercera ola**

  Según la visión de Weiser, la tercera ola ya había comenzado en el año 1998, y esta sería la era del Cómputo Ubicuo. Las computadoras serían omnipresentes por su reducción en tamaño y en precio. Entonces, objetos como los relojes, automóviles, carteras, entre otras cosas, contarían con una computadora en su estructura, y estas computadoras estarían comunicadas entre sí, formando una infraestructura invisible para los usuarios.

En la actualidad, ya estamos un paso más cerca del Cómputo Ubicuo, debido al avance y reducción de tamaño en la tecnología, así como al incremento en el desarrollo de dispositivos informáticos para distintos propósitos. Además, estos dispositivos ya pueden comunicarse y compartirse datos entre sí, a través de redes alámbricas e inalámbricas.

Sin embargo, aún falta mucho por recorrer para lograr llegar a la propuesta de Weiser publicada en el artículo "La computadora del siglo XXI" en el cual menciona que:

- El propósito de las computadoras es ayudar a los humanos a realizar actividades en el mundo real y facilitar dichas actividades.
- La computadora debe ser o volverse invisible al extender el inconsciente de los usuarios.

El último punto se refiere a que la invisibilidad de la computadora es consecuencia del buen aprendizaje y la naturalidad en el uso de las tecnologías y las interfaces de las mismas, de tal forma que los usuarios dejan de ser conscientes de que las están usando.

2.1.2 Conciencia del contexto

El contexto, según la Real Academia Española (RAE)[1], es el entorno lingüístico del que depende el sentido de una palabra, frase o fragmento determinados o también se puede entender como el entorno físico o de situación, político, histórico, cultural o de cualquier otra índole, en el que se considera un hecho. En la comunicación entre humanos, es una práctica natural ser consciente del contexto, por ello, somos capaces de entender el sentido de una conversación, captar o transmitir ideas y en consecuencia, actuar de forma adecuada en una determinada situación o entorno.

A diferencia del humano, las computadoras tienen una conciencia del contexto limitada, debido a que dependen de su capacidad de detectar el entorno, captar eventos y la interfaz que ofrecen para la interacción entre el usuario y la máquina. En el área de la computación, algunos autores como Dey y Adwood definen el contexto como " *cualquier información que puede utilizarse para caracterizar la situación de una entidad*" donde "*una entidad puede ser una persona, lugar u objeto que se considere relevante para la interacción entre un usuario y una aplicación, incluido el usuario y las propias aplicaciones*" [19]. Pero,

[1] https://dle.rae.es/

¿qué diferencia hay entre una computadora consciente del contexto y una no consciente?, la respuesta está en que las computadoras conscientes del contexto aprovechan los datos del entorno y de la comunicación con el usuario, lo que les permite ofrecer un mejor servicio, ya que pueden responder de acuerdo a la situación e incluso tener un nivel más alto de entendimiento de las peticiones del usuario. Por esta razón, se ha popularizado el concepto de contexto en el área de computación y se ha incrementado el desarrollo de sistemas y aplicaciones conscientes del contexto.

2.1.3 Vida asistida por el entorno

El área de investigación Ambientes Inteligentes (AmI) es un campo multidisciplinario muy relacionado con la visión del Cómputo Ubicuo de Weiser. El campo de AmI tiene como objetivo mejorar la vida cotidiana, al cambiar la forma en que las personas viven, trabajan y socializan mediante la inserción y desarrollo de las TICs. En 1999, el Grupo Asesor de las Tecnologías de la Sociedad de la Información (ISTAG) introdujo el concepto de AmI, y en 2003 Lindwer *et al.* definen AmI como "*...la visión de que la tecnología sea invisible, integrada en el entorno natural, presente siempre que la necesitemos, habilitada por interacciones simples y sin esfuerzo, en sintonía con todos nuestros sentidos, autónomo y adaptable a los usuarios y al contexto*"[21].

Un subcampo de AmI es la vida asistida por el entorno (VAE), cuya aparición fue a raíz de que el sector político y los gobiernos de algunos países desarrollados se percataron de un cambio en su perfil demográfico, es decir, su población sería, en un futuro cercano, compuesta principalmente por adultos mayores. Por esta razón, propusieron la visión de VAE, cuyo objetivo es desarrollar sistemas para apoyar a los adultos mayores en sus actividades diarias para permitirles una vida más independiente y segura, el mayor tiempo posible. Actualmente, se ha extendido el desarrollo de proyectos basados en VAE para algunos sectores de la población como personas con discapacidad, personas con alguna enfermedad y adultos mayores, que necesitan asistencia tecnológica, médica, o de otro tipo [22].

2.2 Trabajo relacionado

2.2.1 Antecedentes

Como antecedentes se tienen dos estudios vinculados con esta investigación.

2.2.1.1 Diseño e implementación de mecanismos de búsqueda contextualizada y anotado a través de la Web Semántica

Este trabajo fue publicado por Romero en 2012 [23], en cuya motivación se expresa la necesidad de recuperar información de la Web, que esté enfocada a los requerimientos reales del usuario. Lo anterior, debido a que la Web está en constante crecimiento y los buscadores populares realizan extensos recorridos, a través de toda la Web, para ofrecer repuestas de miles o millones de sitios sugeridos para los tópicos buscados. A pesar de que los buscadores ofrecen ciertos filtros o recomendaciones, mediante técnicas de búsqueda o datos recolectados según la configuración y los permisos otorgados al navegador, no siempre facilitan la recuperación de información para el usuario.

El objetivo de esta investigación es construir un mecanismo de búsqueda que, mediante un conjunto de palabras clave y un tema en particular, recopile información de la Web referente al contexto establecido. El contexto está compuesto por palabras clave proporcionadas por el usuario. Además, este trabajo permite la construcción de una ontología, en donde se modela el dominio de conocimiento en cuestión.
Algunas de las contribuciones que aportan a la investigación actual son: la creación de un mecanismo de interpretación de la información, mediante la búsqueda de términos del dominio de conocimiento dentro de documentos (HTML o PDF) y la creación de un mecanismo para la explotación de la información obtenida. Romero concluyó que la combinación de técnicas de modelado de contexto, recuperación de información y anotado semántico permitieron lograr los objetivos planteados en su investigación. El usuario genera un contexto y con base en éste realiza búsquedas especializadas, acorde a las necesidades,

mediante la implementación de un "crawler contextual" que consiste en la adaptación del contexto base. Se logró una mejora significativa en comparación con métodos sintácticos popularmente usados.

2.2.1.2 Activación multimodal de servicios en dispositivos móviles

Este trabajo fue publicado por Ponce en 2012 [24], donde plantea la necesidad de contar con interfaces hombre-computadora, que permitan una interacción más natural entre el usuario y los dispositivos electrónicos, mediante el uso de las diferentes capacidades del usuario (motrices, voz y audiovisuales).

El objetivo de la investigación es diseñar e implementar mecanismos multimodales proactivos para la activación de servicios en dispositivos móviles. Se propone una interacción más natural para el usuario, aún cuando éste padezca de algún impedimento, y permitiendo al mismo tiempo explotar las capacidades de cómputo y conectividad mediante el uso de los recursos embebidos, como sensores e interfaces, disponibles en los dispositivos. Algunos de los objetivos particulares del trabajo de Ponce que se relacionan con la investigación actual son:

- Facilitar la interacción entre el usuario y los dispositivos móviles al brindar mecanismos de interacción/activación multimodal de servicios.
- Implementar mecanismos de análisis de contexto que asistan al sistema.

Con base en estos y otros objetivos, Ponce desarrolló un sistema multimodal para dispositivos móviles con el sistema operativo Android. La finalidad fue activar los servicios disponibles en los dispositivos, mediante el uso de comandos, en las modalidades de audio, gesto táctil, gesto físico y texto. Además, cuenta con un módulo de análisis de contexto, en cuanto a la conectividad a las redes celulares e inalámbricas del dispositivo.

Los trabajos descritos se vinculan con la investigación actual debido a que ambos ofrecen propuestas que apoyan el desarrollo de un sistema para la navegación web contextual,

visual y por voz. Por un lado, con el desarrollo de mecanismos que facilitan la búsqueda en la Web semántica con base en el contexto del usuario y, por otro lado, al desarrollar mecanismos multimodales para facilitar la interacción del usuario con dispositivos móviles.

2.2.2 Análisis de propuestas relevantes

Como menciona Pérez Zúñiga *et al.*, *"La sociedad de la información puede aplicar a cualquier persona que cuente con los recursos tecnológicos necesarios para acceder y obtener información en Internet"*[25]. Gracias al proyecto de hipertexto de Berners-Lee *et al.*, publicado en 1990 y popularmente conocido como la Web, es posible enlazar y acceder a información de diversa índole a través de la red de redes [26]. Por esto, se puede deducir que navegar en la Web es una actividad importante para la sociedad de la información. La brecha digital divide a las personas en dos grupos, las que tienen y las que no tienen acceso a las TICs. Dicha brecha digital puede ser consecuencia de la falta de habilidades, impedimentos físicos o inexperiencia de los usuarios. Por ende, si los usuarios no pueden usar recursos tecnológicos, entonces tampoco pueden obtener información de la Web.

Mark Weiser describe en su artículo "La computadora del siglo XXI", que las tecnologías deben ayudar a las personas a realizar actividades. Además menciona que las tecnologías más profundas son aquellas que desaparecen en el entorno, de tal forma, los usuarios dejan de ser consientes de que las están usando [7]. Apoyando al postulado de Weiser, Saha y Mukherjee comentan en su artículo "Pervasive Computing: A Paradigm for the 21st Century", que la invisibilidad o desaparición de las tecnologías se obtiene cuando existe la mínima intervención de los humanos en los sistemas y, si los humanos intervienen, debe ser sólo para actualizar el contexto y preferencias [27].

En el presente trabajo se plantea facilitar la navegación web, basándose en el emparejamiento entre el contexto del usuario y el contexto de las páginas web, para entregar contenido orientado a los intereses del usuario. Además de ofrecer una manera más natural de interacción entre los usuarios y las TICs al permitir navegar en la Web por medio de voz y recibir el contenido en formato audible o gráfico. En esta sección, se exponen diferentes

trabajos relacionados con esta propuesta de tesis, los cuales han sido organizados según las siguientes categorías. Primero se describen trabajos relacionados con asistentes virtuales, después trabajos relacionados con navegación web basada en contexto y finalmente navegación web por voz.

2.2.2.1 Asistentes virtuales o de voz

Un asistente virtual o asistente de voz es un programa informático capaz procesar el lenguaje natural del usuario. Además, interactúa con el usuario y dispositivos inteligentes para ofrecer información por medio de un diálogo verbal en tiempo de ejecución [28].

Actualmente, se ha desarrollado una variedad de asistentes virtuales. Por un lado, los asistentes más populares son: Alexa de Amazon [2], Asistente de Google [3], Siri de Apple [4], Cortana de Microsoft [5], por mencionar algunos. Los primeros tres, cuentan con dispositivos físicos como Echo Dot, Google Home y HomePod respectivamente. En el Cuadro 2.1, a manera de tabla comparativa, se enlistan las características de los asistentes virtuales, relacionadas con la propuesta de tesis. Estos asistentes permiten al usuario acceder a una gama de funcionalidades mediante voz o texto. De manera general, las funcionalidades que ofrecen son:

- Acceso a la Web para encontrar respuestas a preguntas cotidianas, como aquellas relacionadas con cálculos, traducciones, conversión de unidades, nutrición, diccionarios, por mencionar algunos.
- Gestión de contenido multimedia, por ejemplo: reproducción de música, visualización de videos y reproducción de audiolibros.
- Control de dispositivos compatibles, como: electrodomésticos, aparatos de iluminación, televisión, bocinas, entre otros.

[2] Alexa de Amazon, revisado en noviembre de 2022. URL: https://developer.amazon.com/es-ES/alexa.
[3] Asistente de Google, revisado en noviembre de 2022. URL: https://assistant.google.com/intl/es_es/.
[4] Siri, revisado en noviembre de 2022. URL: https://www.apple.com/mx/siri/.
[5] Cortana de Microsoft, revisado en noviembre de 2022. URL: https://support.microsoft.com/es−es/topic/-qué-es-cortana-953e648d-5668-e017-1341-7f26f7d0f825.

- Acceso a un organizador personal y administración de alarmas y temporizadores.
- Otras tareas como realizar llamadas, mandar mensajes, entre otras.
- Acceder a funcionalidades de aplicaciones compatibles, como comercio por voz, juegos, material educativo, entre otros.

Tabla 2.1: Asistentes virtuales populares

Característica	Alexa de Amazon	Asistente de Google	Siri de Apple	Cortana de Microsoft
Disponible en distintos dispositivos inteligentes.	✓	✓	✓	✓
Disponible en varios idiomas, entre ellos, Español.	✓	✓	✓	✓
Accede a funcionalidades a través de micro-servicios web o aplicaciones.	✓	✓	✓	✓
Cuenta con un módulo de reconocimiento automático de voz.	✓	✓	✓	✓
Cuenta con un módulo de comprensión de lenguaje natural.	✓	✓	✓	X
Interactúa con otros dispositivos inteligentes.	✓	✓	✓	X
Permite el desarrollo de funcionalidades a terceros.	✓	✓	✓	X
Considera datos como historial de navegación, ubicación, idioma y anuncios vistos, para ofrecer contenido di al usuario.	✓	✓	✓	✓
Compara contexto del usuario con el contexto de las páginas para ofrecer contenido al usuario.	X	X	X	X
Permite navegar dentro de las páginas web mediante instrucciones de voz.	X	X	X	X

Por otro lado, se han desarrollado asistentes virtuales con enfoques específicos en distintas

áreas. Las áreas con mayor cantidad de asistentes en los últimos años son: educación, salud, economía, por mencionar algunos. Por ejemplo, en el área educativa se tiene a Ubot [29] desarrollado en 2022 por Rubio *et al.*, un asistente enfocado en facilitar actividades administrativas y procedimentales de la Universidad Bernardo O'Higgins de Chile. En el área de salud, Vera *et al.*, desarrollaron SaminBot [30] en 2021, con el objetivo de recolectar datos y brindar información durante la pandemia del COVID-19 en la región Cusco del Perú. En la investigación documental no fueron encontrados trabajos, en el área de asistentes virtuales, relacionados directamente con la navegación web contextual. Sin embargo, a continuación se reportan algunos asistentes de propósito específico, que se relacionan con la propuesta de tesis o que podrían apoyar en el desarrollo de la misma.

MyrrorBot

Asistente digital basado en modelos holísticos de usuario[1], para el acceso personalizado a servicios en línea. Fue desarrollado en octubre del 2021 por Must *et al.* [31]. Este sistema invoca los servicios externos a través de sus respectivas aplicaciones, por ejemplo: Youtube, GoogleNews y Spotify, por mencionar algunos. Algunas de las funcionalidades de este trabajo son:

- Procesamiento de solicitudes expresadas con lenguaje natural en inglés.
- Autogestión del perfil de usuario.
- Procesamiento de dos clases de solicitudes: relacionadas con el usuario y de acceso a servicios en línea.
- Acceso a servicios en línea relacionados con música, videos, noticias y recomendaciónes de alimentos.

[1]El perfilado holístico de los usuarios, se basa en 7 facetas: datos demográficos, intereses, afectos, rasgos psicológicos, comportamiento, relaciones sociales y salud.

- Consulta y edición del modelo holístico y las características codificadas en cada faceta.

Las características de interés encontradas en este trabajo son: el modelado holístico del usuario y el mecanismo de procesamiento de solicitudes e intenciones.

Asistente de enseñanza virtual contextual impulsada por inteligencia artificial usando RASA

Este asistente, desarrollado en 2020 por Shekhar, D'Souza y Fernandes [32], tiene como objetivo ayudar en el proceso de aprendizaje de los alumnos, tomando en cuenta el contexto de cada alumno y los recursos del instructor. Las funcionalidades de este trabajo son:

- Detección del contexto de usuario por medio de preguntas generales.
- Generación de preguntas capciosas basadas en los materiales del instructor.
- Identificación de los estudiantes que necesitan ayuda adicional con base en el contexto de cada uno.
- Respuesta personalizada a las preguntas de los estudiantes.
- Recordatorio sobre los plazos de entrega de tareas.

Las características de interés encontradas en este trabajo son: el manejo del contexto de usuario, la toma de decisiones basadas en contexto y el mecanismo de generación de respuestas personalizadas.

Asistente personal, inteligente y virtual para investigadores de murciélagos

El trabajo desarrollado en 2018 por Ivanov y Orozova [33], es un asistente dedicado a facilitar el trabajo de campo de los investigadores de murciélagos. Su arquitectura está basada en un modelo de Creencias, Deseos e Intenciones (CDI). Las funcionalidades de este trabajo son:

- Recopilación de datos de campo personalizables.
- Procesamiento, validación y análisis de datos.
- Generación de reportes.
- Proactividad basada en el ajuste de objetivos.

Las características de interés encontradas en este trabajo son: la implementación del modelo basado en CDI y el mecanismo de comportamiento proactivo.

Asistente de escritorio personal consciente del contexto

Asistente desarrollado en 2008 por Bui *et al.* [34], con el objetivo de administrar la ejecución de proyectos (PExA), y ayudar al usuario con sus actividades dentro del escritorio de Windows. El asistente trabaja en función del conocimiento del flujo de trabajo de los usuarios, progreso y preferencias. El proyecto está basado en una arquitectura CDI, usando SPARK[2]. Las funcionalidades que se destacan de este trabajo son las siguientes:

- Atención de necesidades del usuario.
- Manejo de preferencias y objetivos del usuario.
- Manejo del contexto en el escritorio de Windows.
- Comportamiento proactivo.

[2]SPARK es un framework de agentes bajo el modelo BDI. Fue desarrollado por el Centro de Inteligencia Artificial del Instituto de Investigación de Stanford Internacional.

- Gestión de proyectos y tareas.
- Organización de la información.
- Preparación y generación de resúmenes de reuniones.

Las características de interés encontradas en este trabajo son: el mecanismo de manejo de contexto y el mecanismo de rastreo del patrón de comportamiento del usuario.

2.2.2.2 Navegación web basada en contexto

Siguiendo con la definición de contexto de la Sección 2.1.2, se puede observar que en la navegación web existen al menos tres tipos de entidades: usuario, página web y navegador. En este sentido, en la navegación web basada en contexto se puede aprovechar la información de las entidades involucradas para afinar la búsqueda y así presentar la información mejor relacionada con la petición, según el contexto del usuario. A continuación se muestran los trabajos relacionados con la navegación web basada en contexto.

Extracción de hechos cuantitativos contextualizados de las tablas web

Este trabajo fue desarrollado en abril del 2021 por Ho *et al.*. [35] y tiene como objetivo extraer automáticamente datos cuantitativos de tablas en la Web, mediante la contextualización del contenido. Las características destacables de este trabajo son las siguientes:

- Mecanismo de contextualización de contenido mediante reconocimiento del marcado estructural y pistas informativas dentro de la página web.
- Mecanismo para la clasificación de datos en tiempo de consulta basada en la similitud entre el contexto y la consistencia de los datos.

Sin embargo, presenta las siguientes desventajas:

- El modelo está basado en un formato específico de tablas.
- Únicamente se analiza el contexto de la tabla de interés.

- Está orientado a extraer información de tablas en formato HTML.
- No facilita la navegación web en otros aspectos.

Personalización de la búsqueda en la Web utilizando un contexto de navegación a corto plazo

En este trabajo, desarrollado en octubre de 2013 por Ustinovskiy y Serdyukov [36], se presenta un método para la personalización de preferencias de búsqueda en la Web basado en el contexto a corto plazo del usuario[1]. Las características destacables de este trabajo son:

- Uso de técnicas de aprendizaje automático para el filtrado y la clasificación de consultas en dos categorías: útil y no útil para personalización de búsqueda web.
- Realiza la personalización de búsquedas de un usuario nuevo basado en las experiencias de otros usuarios.
- El contexto está relacionado con el contenido web estadísticamente más visitado y el tipo de contenido preferido por el usuario.

Algunas desventajas que presenta este trabajo son:

- Únicamente atiende el problema de personalizar las primeras consultas en una sesión de búsqueda con falta de contexto.
- No analiza el contexto de las páginas web.
- El contexto a corto plazo es generado con base en las consultas y actividades recientes del usuario.

Búsqueda web contextualizada: clasificación dependiente de la consulta y búsqueda en medios sociales

[1] "Contexto a corto plazo" se refiere a la información de preferencias en búsquedas web de un usuario nuevo o que no cuenta con historial y/o preferencias de búsquedas.

En este trabajo, desarrollado en diciembre del 2010 por Bian [37], se implementaron algoritmos y técnicas para la adquisición efectiva de conocimiento a partir de redes sociales, mediante el uso de información de contexto dinámico. Las características destacables de este trabajo son:

- Búsqueda basada en la información del contexto del usuario que publica en la Web y en el contexto de contenido.
- La calidad del contenido y la reputación del usuario son calculadas mediante un mecanismo semi-supervisado.
- Proporciona una propuesta de técnicas para la extracción de la semántica estructurada del contenido en las redes sociales.

En este trabajo se pueden observar principalmente dos desventajas:

1. El estudio se centra en la búsqueda dentro del contenido de las redes sociales del año 2010, por ejemplo Yahoo Respuestas, Flickr, Youtube y Delicious.
2. No considera el contexto del usuario que realiza la búsqueda.

Búsqueda web contextual basada en relaciones semánticas: marco teórico, evaluación y prototipo de aplicación médica

En este trabajo desarrollado en 2006 por Zhang [38], se presenta una propuesta de marco conceptual para la clasificación de varios tipos de contexto en un entorno de búsqueda web. El objetivo es disminuir la ambigüedad de las consultas mediante el análisis del contexto y la relación entre los conceptos que se consultan. Algunos de los aspectos relevantes de este trabajo son:

- Generación del marco contextual mediante la clasificación de diferentes tipos de contexto en las búsquedas.

- Detección y uso de relaciones semánticas entre términos de consulta.
- Clasificación de contextos de las búsquedas.
- Procesamiento de consultas en lenguaje natural, en formato de texto.

Sin embargo, este trabajo presenta algunas desventajas como las siguientes:

- Está basado en motores de búsqueda del año 2006.
- Realiza las búsquedas mediante el análisis del contexto de la petición sin tomar en cuenta el contexto del usuario.
- El prototipo del sistema está orientado únicamente al área médica.

2.2.2.3 Navegación web por voz

En el área de investigación de Interfaces Humano-Computadora, mejor conocida como HCI, se busca facilitar, mejorar y hacer que la interacción entre humanos y máquinas sea de forma más natural. Munteanu y Penn comentan que "*el habla es la forma más natural de comunicación de los humanos y al mismo tiempo es una de las modalidades más difíciles de entender para las máquinas*" [39]. Pero gracias a los avances en interfaces de voz, se ha popularizado esta modalidad de interacción y se han desarrollado una gran variedad de herramientas bajo este enfoque, como los descritos en la Sección 2.2.2.1. A continuación se describen algunos de los trabajos relacionados con la navegación web, que utilizan las interfaces de voz y entregan contenido en formato audible, con el objetivo de mejorar la interacción de los usuarios o facilitar la accesibilidad a las personas en riesgo de exclusión tecnológica.

InSupport: interfaz de proxy para permitir una interacción no visual eficiente con registros de datos web

En este trabajo desarrollado por Ferdous, Lee, Jayarathna y Ashok [40] en marzo de 2022, se presenta una extensión para el navegador web Google Chrome como complemento para el lector de pantalla JAWS[1]. Su objetivo es facilitar la interacción a las personas con disminución parcial o total de la vista con los registros de datos en aplicaciones web modernas. Las características destacables de este trabajo son:

- Identificación y manejo de segmentos de interés en las páginas web.
- Despliegue de información filtrada en una interfaz web alterna.
- Entrega información por voz mediante el lector JAWS.

Sin embargo, algunas de las desventajas que se encontraron son:

- Depende del lector de pantalla JAWS para funcionar.
- Únicamente es compatible con el navegador Google Chrome.
- Funcional en el entorno de pruebas preestablecido.
- Se enfoca en la interacción con registros de datos.

SaIL: inyección impulsada por la prominencia de puntos de referencia ARIA

SaIL es un sistema automático, desarrollado en marzo del 2020 por Aydin, Feiz, Ashok y Ramakrishnan [18], que permite detectar la información importante de los sitios web y generar puntos de referencia en el estándar ARIA (ARIA Landmarks). Facilita la navegación a través de los lectores de pantalla JAWS, usando referencias que determinan la ubicación y secuencia de la información más importante. Las características más destacables de este trabajo son:

[1]JAWS es la sigla de Job Access With Speech, que es el lector de pantalla más popular del mundo, desarrollado para usuarios de computadoras cuya pérdida de visión les impide ver el contenido de la pantalla o navegar con un ratón.

- Algoritmo basado en aprendizaje automático para la detección de información importante.
- Está entrenado con información obtenida del contexto de búsqueda e información más visitada por los usuarios (gaze data).
- Entrega la información por voz mediante el lector JAWS.

Sin embargo, algunas de las desventajas que se encontraron son:

- Es un complemento del lector de pantalla JAWS.
- Compatible con el navegador Google Chrome.
- Se enfoca en la detección y el marcado de la información más visitada en una página web.

iTOC: habilitación de una interacción no visual eficiente con documentos web

Extensión de navegador que identifica y extrae automáticamente los hipervínculos de tablas de contenido de los documentos web, y luego facilita el acceso instantáneo a pedido del lector de pantalla desde cualquier parte del sitio web. El objetivo de iTOC es mejorar la interacción de los usuarios del lector de pantallas JAWS con documentos web extensos. Desarrollado en 2020 por Lee, Uddin y Ashok [17]. Las características más destacables de este trabajo son:

- Identificación semántica en las páginas web.
- Extracción de enlaces de una página web.
- Clasificación de enlaces en una tabla de contenido.
- Entrega la información por voz mediante el lector JAWS.

Sin embargo, algunas de las desventajas que se encontraron son:

- Es un complemento del lector de pantalla JAWS.
- Compatible con el navegador Google Chrome.
- Se enfoca en detectar tablas de contenido y ofrecer un acceso rápido a cada elemento dentro de la tabla.

Asistencia en automatización de lectura de pantalla web mediante abstracción semántica

Sistema basado en la abstracción semántica cuya entrada son comandos en lenguaje natural, con el objetivo de localizar y acceder rápidamente a partes específicas de los sitios web relacionados con compras, reservaciones y registros. Desarrollado en 2017 por Ashok, Puzis, Borodin y Ramakrishnan [15]. Las características destacables de este trabajo son:

- Interpretación de comandos de voz mediante una biblioteca de términos de búsqueda.
- Segmentación de los elementos de las páginas web.
- Filtrado de los elementos útiles de las páginas web, con base en las interpretaciones de los comandos de voz.

Algunas desventajas que se encontraron son:

- Los comandos son específicos y predefinidos.
- No existe un análisis semántico de las instrucciones del usuario.
- El estudio está basado en las características generales de 100 sitios web populares.
- El sistema está orientado a acciones donde es necesario interactuar con botones, cursor y registros.

HO2IEV: técnica de extracción de información web basada en ontología pesada para usuarios invidentes

Mecanismo para la extracción de información de alta precisión utilizando ontologías pesadas y un sistema integrado de comandos de voz para usuarios de Internet con discapacidad visual. Desarrollado en Junio de 2011 por Bukhari, Fareedi y Kim. [41]. Las características destacables de este trabajo son:

- Extracción de información de alta precisión utilizando ontologías pesadas.
- Reconocimiento de comandos de voz, especializado para usuarios de Internet con discapacidades visuales.

Las desventajas que se encontraron en este trabajo se enlistan a continuación:

- Comunicación con el servidor web por medio del servicio de mensajes cortos (SMS).
- No realiza procesamiento de lenguaje natural.
- No permite interactuar con el contenido de las páginas web.

Hearsay: navegador web multimodal de nueva generación, de asistencia basado en el contexto

Navegador web no visual y multimodal cuyo objetivo es resolver algunos de los problemas de accesibilidad a la Web. Es una aplicación de escritorio que también se puede utilizar de forma remota a través de teléfono fijo y es compatible con la red de accesibilidad social de IBM. Desarrollado en 2010 por Lee, Uddin y Ashok [16]. Las características destacables de este trabajo son:

- Análisis de contenido de las páginas web.
- Identificación de información relevante basado en el modelo estadístico de aprendizaje automático máquina de vectores de soporte.

- Manejo de contexto de las páginas web.
- Detección de lenguaje mediante estadística, para el reconocimiento de comandos de voz.
- Salidas: audio, visual y Braille.
- Entradas: voz, texto y gestos en pantalla táctil.
- Convierte el contenido de la página web en diálogos interactivos para el usuario, con base en el estándar VoiceXML [2] .

Las desventajas que se encontraron se enlistan a continuación.

- Desarrollado con tecnología vigente al año 2010.
- No analiza semánticamente las peticiones del usuario.
- Se maneja mediante comandos de voz específicos y predeterminados.
- Basado en navegadores Firefox e Internet Explorer en sus versiones del año 2010.
- No cuenta con módulo de procesamiento de lenguaje natural.
- Basado en plantillas de diálogo VoiceXML.

Con base en el análisis realizado a lo largo del Capítulo 3, se determina que el proyecto planteado es realizable debido a lo siguiente:

- En los antecedentes se han establecido las bases para este proyecto, por un lado, con el desarrollo de mecanismos que facilitan la búsqueda en la Web con base en el contexto del usuario y, por otro lado, al desarrollar mecanismos multimodales para facilitar la interacción del usuario con dispositivos móviles.

[2]VoiceXML es un estándar de documentos digitales para especificar medios interactivos y diálogos de voz entre humanos y computadoras.

- En el análisis del estado del arte se determina que no se ha publicado ningún trabajo que facilite la navegación web tomando en cuenta el contexto del usuario y de las páginas web, y que además, ofrezca una interfaz de voz.
- Los trabajos analizados, al tener objetivos y funcionalidades relacionados con el proyecto de tesis, contienen técnicas y mecanismos que contribuyen en el desarrollo del tema de investigación.
- Las herramientas de asistencia para personas con discapacidad visual presentan áreas que se pueden mejorar e implementar usando tecnología actual.

Capítulo 3

Propuesta de Solución

En este capítulo se plasman los resultados obtenidos de las actividades de análisis y diseño del proyecto. En la primera sección se definen los requerimientos funcionales y no funcionales del modelo. En la segunda sección se encuentra la arquitectura del modelo. En la sección tres se describe el manejo del contexto y por último, en la sección cuatro, se encuentra el análisis de herramientas seleccionadas para desarrollar el prototipo.

3.1 Especificación de requerimientos

Los requerimientos son declaraciones que expresan con presición las necesidades, limitaciones, condiciones, suposiciones y atributos del usuario, del desarrollador, del sistema o de los elementos del sistema. El resultado principal de especificar requerimientos es un conjunto de observaciones que permiten el entendimiento entre las partes interesadas: el usuario y el desarrollador. La especificación de requerimientos también funciona como referencia para el diseño, planificación, ejecución de pruebas y la validación del modelo. Como se menciona en las secciones 1.1 y 1.3 del Capítulo 1, el presente trabajo es una propuesta de navegación web por voz para personas que son afectadas por la desigualdad que genera la brecha digital, específicamente, para personas con analfabetismo o analfabetismo digital, adultos mayores y personas con discapacidad visual, motriz y/o

adquirida por amputación. Por esta razón se requiere de una especificación minuciosa de los requerimientos y en consecuencia, esta sección está basada en la norma ISO IEEE 29148-2018[1].

3.1.1 Funciones del prototipo

En esta sección se resumen las principales funciones que debe realizar el prototipo del modelo de navegación por voz, basado en contexto.

1. El prototipo debe escuchar y entender las peticiones y respuestas por voz del usuario.

2. El prototipo debe encontrar contenido en la Web que se relacione con la consulta del usuario y su contexto.

3. El prototipo debe permitir navegar al usuario dentro de una página web mediante comandos de voz. Las actividades que el prototipo debe permitir al usuario son:

   - Interactuar con objetos que generen eventos como botones, casillas de verificación, barra de desplazamiento, por mencionar algunos.
   - Seleccionar texto o elementos de multimedia.
   - Moverse entre secciones de una página web.

4. El prototipo debe reproducir al usuario el contenido o información relevante de una página web.

5. El prototipo debe administrar el contexto del usuario y monitoreo de la navegación.

6. El prototipo debe permitir la interacción con el usuario principalmente por voz, pero también mediante gestos en pantalla y teclado virtual.

[1]La norma IEEE 29148-2018 especifica los procesos requeridos en las actividades de ingeniería que dan como resultado requisitos para sistemas y productos de software (incluidos los servicios) a lo largo del ciclo de vida. Disponible en: https://www.iso.org/standard/72089.html

3.1.2 Características del usuario

En esta sección son descritas las características generales de los grupos de usuarios para quienes está orientado el prototipo basado en el modelo de navegación por voz, basado en contexto. Como se menciona en la Sección 1.3, con este proyecto se busca facilitar la navegación web a la población que, por diversas razones, tiene dificultad para realizar dicha actividad. Aunque la propuesta puede ser usada por cualquier usuario que pueda comunicarse hablando, se considera que los usuarios a los que va dirigido esta propuesta tienen las siguientes características:

1. El usuario debe poder realizar peticiones por voz en español.
2. El usuario debe ser una persona oyente o vidente o ambos.
3. El usuario puede o no ser analfabeta o analfabeta digital o ambos.
4. El usuario puede o no tener una o varias de las siguientes condiciones:
   - discapacidad visual.
   - discapacidad motriz.
   - discapacidad adquirida por amputación.
5. El usuario puede tener experiencia básica a avanzada en cuando a la navegación web.
6. El usuario puede ser de cualquier edad, si cumple la característica 1.
7. El usuario puede o no tener algún nivel educativo.

3.1.3 Requerimientos funcionales

En esta sección se especifican todas las acciones y comportamientos externos que debe llevar a cabo la propuesta, y que deben ser percibidos por el usuario. Este apartado se desarrolla por objetivos, los cuales son servicios que ofrece el modelo. A continuación, se detallan los requerimientos funcionales necesarios para cumplir con cada objetivo.

Objetivo 1: entender la petición del usuario

El prototipo debe ejecutar las acciones que le ordene el usuario. Las peticiones del usuario son por voz en idioma español. Los requerimientos para este objetivo son los siguientes.

Tabla 3.1: Requerimiento funcional RF01.

ID de requerimiento	**RF01**
Nombre de requerimiento	Modo en espera y reconocimiento de palabra clave.
Propósito	Reconocer cuando el usuario pronuncie la palabra clave, la cual indica que el usuario comenzará una petición.
Descripción	El prototipo permanece activo en segundo plano y en espera de que el usuario pronuncie la palabra clave *Selene*. Al reconocer la palabra clave, el prototipo indica al usuario, por medio de un sonido breve, que puede comenzar a decir una petición y a partir de ese momento, el prototipo graba el audio hasta que el usuario no diga ninguna palabra durante dos segundos continuos.
Entrada(s)	Palabra clave y petición en voz del usuario.
Salida(s)	▪ Sonido de confirmación después de que el prototipo capte la palabra clave. ▪ Despliegue de la petición en formato de texto en la pantalla.

Tabla 3.2: Requerimiento funcional RF02

ID de requerimiento	**RF02**
Nombre de requerimiento	Convertidor de audio a texto.
Propósito	Convertir el audio, que contiene la petición del usuario, a texto.
Descripción	El prototipo reconoce las palabras que pronunció el usuario, mediante el análisis de un archivo de audio. El prototipo almacena el texto correspondiente a las palabras reconocidas.
Entrada(s)	Audio correspondiente a una petición hecha por el usuario.
Salida(s)	Texto correspondiente a una petición hecha por el usuario.

Objetivo 2: crear perfil y contexto preliminar del usuario

El prototipo debe crear el perfil de usuario en la primera interacción que exista entre usuario y el prototipo. Los datos del usuario, que se toman en cuenta para el perfil, se enlistan en la Sección 3.3.1.1. A continuación se enlistan los requerimientos para este objetivo.

Tabla 3.3: Requerimiento funcional RF04.

ID de requerimiento	**RF04**
Nombre de requerimiento	Cuestionario para el usuario.
Propósito	Recopilar datos del usuario mediante un cuestionario por voz.
Descripción	▪ El prototipo aplica un cuestionario predeterminado. ▪ El cuestionario se desarrolló previamente, de modo que el usuario pueda responder con respuestas cortas o elegir entre opciones que el prototipo debe sugerir al usuario. ▪ Las preguntas se presentan al usuario mediante audio y se deben repetir si el usuario así lo desea.
Entrada(s)	Primera interacción y respuestas del usuario en formato de audio.
Salida(s)	Respuestas del usuario en formato de texto.

Tabla 3.4: Requerimiento funcional RF03

ID de requerimiento	**RF03**
Nombre de requerimiento	Analizador del significado de la petición del usuario.
Propósito	Dar significado a la petición del usuario mediante la traducción de texto a instrucciones que el prototipo pueda ejecutar.
Descripción	El prototipo realiza acciones correspondientes a la petición del usuario. Por lo anterior, el prototipo analiza sintácticamente el texto correspondiente a la petición del usuario. El análisis sintáctico consiste en extraer y etiquetar cada uno de los componentes sintácticos que aparecen en la oración. Después, el prototipo analiza cómo se combinan las palabras y determina si es una petición válida o no. Dependiendo de la validez de la petición se forman instrucciones, a continuación se enlistan algunos ejemplos: ▪ Si la petición es válida, se forma una instrucción para el buscador (especificado en el requerimiento RF07 ubicado en el Cuadro 3.6). ▪ Si la petición no es válida, se notifica al usuario para que vuelva a formular la petición. Una petición es válida cuando está formada al menos por un verbo y un sustantivo en ese orden. Siempre y cuando el verbo se encuentre en la lista descrita en la Sección XX o sea sinónimo de algún verbo de la lista.
Entrada(s)	Texto correspondiente a una petición hecha por el usuario.
Salida(s)	Instrucción para el prototipo.

Tabla 3.5: Requerimiento funcional RF05.

ID de requerimiento	**RF05**
Nombre de requerimiento	Interpretación de respuestas del usuario.
Propósito	Registrar datos y preferencias del usuario en su perfil y contexto.
Descripción	El prototipo analiza el texto correspondiente a las respuestas del usuario y toma decisiones con base en las respuestas para, por ejemplo: preguntar nuevamente, ofrecer información al usuario acerca del uso de su respuesta o llenar directamente el perfil y el contexto del usuario.
Entrada(s)	Respuestas del usuario en formato de texto.
Salida(s)	Tabla con perfil y contexto de usuario.

Objetivo 3: ofrecer contenido que se relacione mejor con la petición y contexto del usuario

El prototipo debe entregar al usuario la información o el contenido mejor relacionado con su petición y su contexto. A continuación se presentan los requerimientos para este objetivo.

Tabla 3.6: Requerimiento funcional RF06.

ID de requerimiento	**RF06**
Nombre de requerimiento	Buscador
Propósito	Encontrar páginas web que se relacionen con la petición del usuario y su contexto.
Descripción	- El prototipo busca páginas web relacionadas con los términos de la petición del usuario, tomando en cuenta el contexto de usuario. - El prototipo toma en cuenta el contexto de las páginas web encontradas en el punto anterior para elegir la(s) página(s) que tenga(n) el contenido que se relacione más con la petición y el contexto del usuario.
Entrada(s)	Petición y contexto de usuario.
Salida(s)	Resultados relacionados con la petición y el contexto de usuario

Tabla 3.7: Requerimiento funcional RF07.

ID de requerimiento	**RF07**
Nombre de requerimiento	Lector de página web.
Propósito	Reproducir por voz el contenido de una página web.
Descripción	- El prototipo analiza la distribución de la página web e identifica los elementos que contiene. - El prototipo indica al usuario qué elementos existen dentro de la página web y ofrece opciones de interacción o de reproducción del contenido. - El prototipo captura la petición del usuario y la ejecuta. - El prototipo lee el contenido que el usuario elija en su petición.
Entrada(s)	Página web y petición del usuario.
Salida(s)	Lectura del contenido de la página web.

Tabla 3.8: Requerimiento funcional RF08.

ID de requerimiento	**RF08**
Nombre de requerimiento	Generador de respuestas en formato específico.
Propósito	Crear respuestas en un formato adecuado a la petición del usuario.
Descripción	De manera predeterminada, el prototipo genera respuestas y las entrega al usuario mediante voz sintetizada. Además, el prototipo es capaz de generar respuestas según la petición del usuario. Las peticiones son acciones, por ejemplo: buscar información, reproducir videos o canciones, visualizar imágenes o documentos, visitar una página web específica, entre otras. Dependiendo de la petición del usuario, el prototipo genera las respuestas como sigue: ▪ Para las peticiones que involucran acciones como reproducir, tocar, interpretar, entre otras; la respuesta es breve, en formato de voz y texto, que indique la fuente y autor, entre otros datos del elemento de multimedia, seguido de la reproducción del mismo. En el caso de que el elemento sea video, este se muestra en pantalla. ▪ Para las peticiones que involucran acciones como mostrar, visualizar, visitar, abrir, entre otras; la respuesta es desplegar el contenido en pantalla.
Entrada(s)	Contenido de las páginas web y petición del usuario
Salida(s)	Respuesta en formato adecuado.

Objetivo 4: interacción del usuario con los elementos de una página web

En la navegación web convencional, los usuarios pueden llenar formularios, dar clic en los elementos, navegar por los menús, acceder a enlaces dentro de una página web, entre otras actividades, haciendo uso del ratón y el teclado. El prototipo permite que el usuario realice actividades similares, pero mediante comandos de voz. A continuación se enlistan los requerimientos para este objetivo.

Tabla 3.9: Requerimiento funcional RF09.

ID de requerimiento	**RF09**
Nombre de requerimiento	Llenar formularios
Propósito	Permitir que el usuario llene formularios que se encuentren dentro de una página web, mediante el uso de voz.
Descripción	▪ El prototipo busca e identifica los elementos pertenecientes al formulario. ▪ El prototipo notifica al usuario el tema u objetivo del formulario y pregunta al usuario si desea llenarlo. ▪ El prototipo informa al usuario el nombre del campo, y captura la respuesta por voz del usuario para llenar los campos.
Entrada(s)	Página web, petición y respuestas del usuario.
Salida(s)	Formulario lleno

Tabla 3.10: Requerimiento funcional RF10.

ID de requerimiento	**RF10**
Nombre de requerimiento	Seleccionar y accionar elementos de una página web.
Propósito	Permitir al usuario seleccionar, y accionar elementos que se encuentren dentro de una página web.
Descripción	▪ El prototipo identifica los elementos que generen eventos dentro de la página web o que redireccionen a otro enlace. ▪ El prototipo notifica al usuario el nombre del elemento y pregunta al usuario si desea activarlo o seleccionarlo. ▪ El prototipo capta la petición del usuario y la ejecuta.
Entrada(s)	Página web y petición del usuario.
Salida(s)	Activar o seleccionar el elemento

3.1.4 Requerimientos de interfaz

Interfaz de usuario El usuario interactúa con el modelo a través de la voz y además, la interfaz de usuario responde en formato audible o gráfica a las peticiones del usuario. En esta sección se definen los requerimientos y las características lógicas de cada interfaz entre el modelo y el usuario.

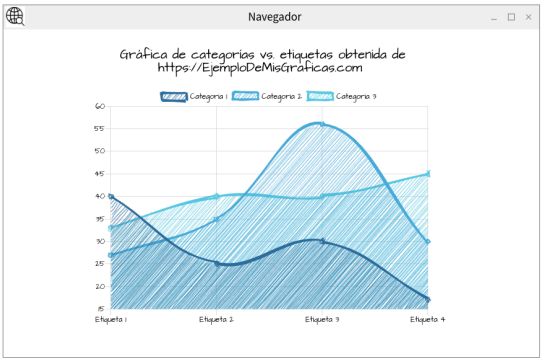
- **Interfaz de voz**: es la interfaz principal y no tiene elementos gráficos. Esta interfaz captura la voz del usuario, traduce a cadenas de texto para posteriormente interpretarlo en lenguaje natural y traducirlo a instrucciones que pueda ejecutar el prototipo.

- **Interfaz gráfica de inicio**: esta interfaz se despliega en la primera interacción que exista entre el usuario y el prototipo. De antemano, el prototipo desconoce qué habilidades o limitaciones posee el usuario, por lo tanto, la interfaz proporciona instrucciones e información en las modalidades gráfica y audible.

- **Interfaz gráfica para despliegue de resultados**: esta interfaz despliega resultados o contenido en pantalla cuando el usuario lo desee. Es capaz de desplegar distintos formatos de contenido multimedia, por ejemplo: texto, imágenes, y videos. En la figura 3.1 se muestra el bosquejo de esta interfaz, donde:

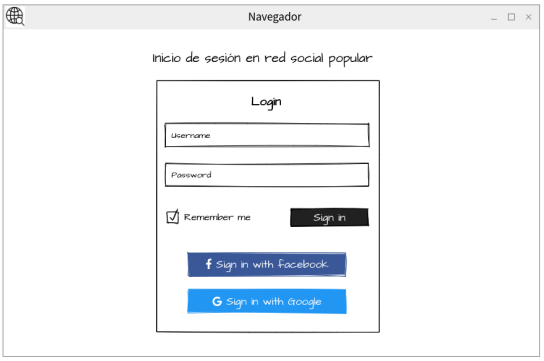
  - En la figura 3.1a se muestra la forma en que se visualizan las gráficas. En esta interfaz el usuario debe poder aumentar y disminuir el tamaño de las imágenes y desplazarse sobre ellas en las diferentes direcciones: izquierda, derecha, arriba y abajo, mediante comandos de voz.
  - En la figura 3.1b se muestra la forma en que se visualizan y reproducen videos y audio. Esta interfaz permite al usuario hacer pausa, repetir, retroceder, adelantar y subir o bajar el volumen del video mediante la voz.
  - En la figura 3.1c se muestra la forma en que se visualizan los formularios. Esta interfaz permite al usuario llenar formularios al interactuar con los componentes de entrada del formulario mediante la voz.
  - En la figura 3.1d se muestra la forma en que se visualiza una lista de resultados correspondientes a una búsqueda. Esta interfaz sintetiza el texto correspondiente a los resultados en voz. Además, permite al usuario seleccionar una opción.

- En la figura 3.1e se muestra la forma en que se visualiza un sitio o página web que el usuario desee observar. Esta interfaz permite al usuario interactuar con menús, contenido, formularios y otros elementos de la página web mediante la voz.



(a) Visualización de gráficos

(b) Visualización de video y audio



(c) Visualización de formulario

(d) Visualización de resultados de una búsqueda

(e) Visualización de una página especifica

Figura 3.1: Interfaz gráfica para despliegue de resultados.

Las modalidades de interacción que proporciona el modelo son mediante voz y texto.

Interfaz de hardware

En esta sección se especifican las características lógicas de cada interfaz que existe entre el modelo y los elementos de hardware del prototipo.

1. **Interfaz de voz**

   Como se mencionó en la sección 3.1.4 “Interfaz de usuario”, el modelo tiene una interfaz de voz que permite la interacción entre el usuario y el modelo.

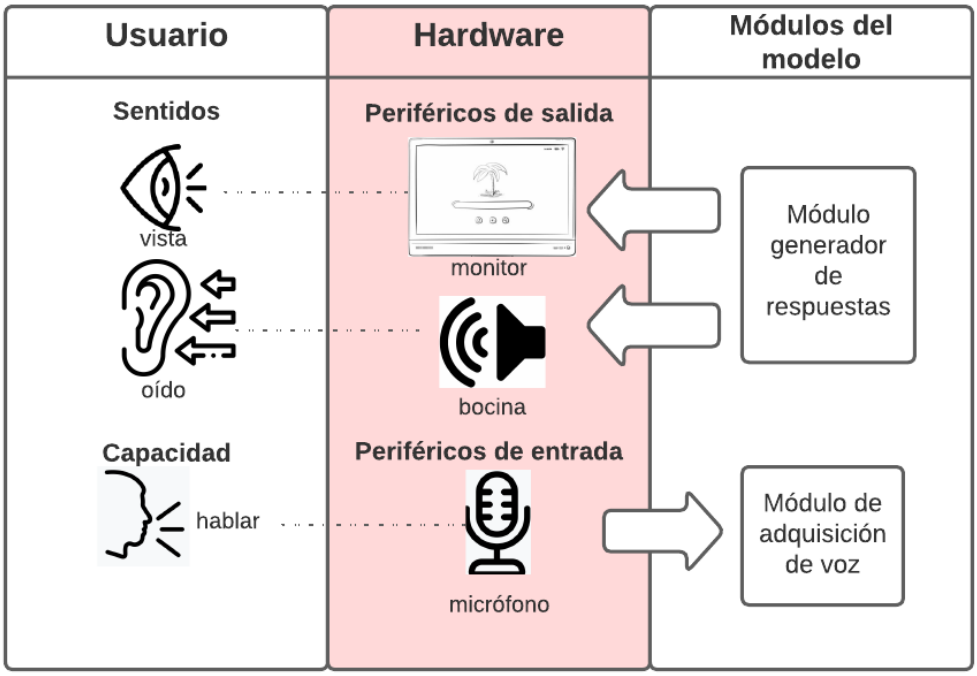


Figura 3.2: Hardware de la interfaz multimodal

En la figura 3.2 se puede observar que, por un lado, el usuario usará sus sentidos para recibir contenido gráfico y audible del prototipo. Por lo anterior, los periféricos de salida que emplea el prototipo son un monitor y una bocina, los cuales reciben el contenido desde el módulo generador de respuestas del modelo. Por otro lado,

el usuario puede ingresar datos o hacer peticiones mediante la voz, entonces, el periférico de entrada es un micrófono que interactúa con el módulo de adquisición de voz.

2. **Interfaz de comunicaciones**

   El modelo necesita una conexión a Internet y a periféricos como bocinas portátiles, por lo que un requisito fue que el prototipo fuera desarrollado en un dispositivo que cuente con conectividad bluetooth e Internet inalámbrico.

3.1.5 Requerimientos de usabilidad

La norma ISO 9241-11[1] define la usabilidad como el "*grado en que un producto puede ser utilizado por usuarios específicos para lograr objetivos específicos con eficacia, eficiencia y satisfacción en un contexto de uso específico*". Para el desarrollo de la primera versión del modelo de navegación web por voz, basado en contexto; se establecen los siguientes requerimientos de acuerdo a las características de usuario definidas en la Sección 3.1.2 y a los requerimientos funcionales definidos en la Sección 3.1.3.

ID de requerimiento	**RU01**
Nombre de requerimiento	Volumen o intensidad del audio.
Propósito	El nivel del volumen del audio que se le proporciona al usuario es el apropiado para una interacción efectiva entre el prototipo y el usuario.
Descripción	▪ El prototipo reproduce el audio a 40 decibeles de intensidad de manera predeterminada. Esta es la magnitud, registrada por la OMS como aceptable en una conversación entre dos personas [42]. ▪ El prototipo permite al usuario aumentar o disminuir la intensidad del sonido.

Tabla 3.11: Requerimiento de Usabilidad RU01.

[1] https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

ID de requerimiento	**RU02**
Nombre de requerimiento	Tamaño del contenido en pantalla.
Propósito	Mostrar el contenido de forma legible para el usuario
Descripción	- El prototipo muestra el contenido en pantalla, con proporciones similares a la Figura 3.1 en la Subsección 3.1.4, de forma predeterminada. - El prototipo permite al usuario reducir o aumentar el tamaño del texto e imágenes gradualmente y restablecer el tamaño predeterminado.

Tabla 3.12: Requerimiento de Usabilidad RU02.

ID de requerimiento	**RU03**
Nombre de requerimiento	Comandos para activación por voz.
Propósito	Activación mediante voz en cualquier momento.
Descripción	- El prototipo permanece en segundo plano hasta que el usuario lo active mediante la voz. - El prototipo cierra las ventanas de la pantalla cuando el usuario lo indique.

Tabla 3.13: Requerimiento de Usabilidad RU03.

3.1.6 Requerimientos de desempeño

En esta sección se especifican los requerimientos numéricos impuestos, para la delimitación del modelo o de la interacción del usuario con el modelo. Estos requerimientos se presentan en forma de lista.

1. El modelo está diseñado para una sola terminal.

2. El modelo atiende a un solo usuario.

3. El modelo no cuenta con memoria de las peticiones que haga el usuario, ni de su cronología.

4. El modelo guarda un historial de las últimas páginas web visitadas, por un periodo de diez días. El historial contiene datos como: fecha de última visita, URL, número de

veces que ha sido visitada, categoría del contenido (por ejemplo: deportes, noticias, etc.).

5. El modelo conserva en el historial la información correspondiente a las páginas web que cumplan lo siguiente:
   - La página web debe tener al menos cinco visitas en diez días.
   - La última fecha de visita a la página web debe tener una antigüedad no mayor a un mes.
   - La página web fue visitada el último día del periodo de diez días.
6. El modelo atiende una petición del usuario a la vez.
7. No se considera un límite de tiempo para la respuesta del modelo, debido a que se estudiará la relación entre los resultados que ofrezca el modelo contra la petición del usuario.
8. El modelo permite que el usuario interrumpa una tarea para realizar otra.

3.1.7 Requerimientos de la base de datos

En esta sección se especifican los requerimientos lógicos para los datos que se almacenarán. A continuación se presentan los requerimientos de la base de datos.

ID de requerimiento	**RD01**
Nombre de requerimiento	Archivo de texto para datos estáticos del usuario.
Propósito	Almacenar los datos del usuario que no cambian a corto o mediano plazo.
Descripción	- El prototipo tiene un archivo de texto donde almacena los datos del usuario, que no cambian a corto o mediano plazo, por ejemplo: nombre, edad, género, nivel de estudios, entre otros. - Los datos son guardados como texto, separados por comas (,).

Tabla 3.14: Requerimiento de Base de Datos RD01.

ID de requerimiento	**RD02**
Nombre de requerimiento	Tabla de páginas web favoritas del usuario.
Propósito	Almacenar los URL de las páginas web favoritas del usuario.
Descripción	- El prototipo cuenta con un archivo de texto, donde se almacena una tabla que contiene datos de las páginas web favoritas o recurrentes del usuario. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: categoría de contenido, URL, número de visitas y contexto de la página web. - Los datos de la tabla son actualizados cada diez días de forma automática o el usuario podrá definir que páginas agregar a esta lista en cualquier momento.

Tabla 3.15: Requerimiento de Base de Datos RD02.

ID de requerimiento	**RD03**
Nombre de requerimiento	Tabla de URLs semillas predeterminadas del usuario.
Propósito	Almacenar los URL seleccionados, según el contexto del usuario, los cuales funcionan como semillas para el rastreo de contenido.
Descripción	- El prototipo cuenta con un archivo de texto donde se encuentra almacenada una tabla que contiene los datos de las páginas web que funcionan como URL de inicio en el rastreo de contenido. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: categoría de contenido y URL.

Tabla 3.16: Requerimiento de Base de Datos RD03.

ID de requerimiento	**RD04**
Nombre de requerimiento	Historial de navegación del usuario.
Propósito	Almacenar datos de las páginas web que el usuario ha visitado.
Descripción	- El prototipo cuenta con un archivo de texto donde se almacena, en forma de tabla, los datos de las páginas web que el usuario visita. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: categoría de contenido, URL, número de visitas y fecha de última visita. - El archivo de texto es actualizado en cada sesión de búsqueda. - Cada diez días se remueven de la tabla, los datos de las páginas web que no cumplen con el requerimiento de desempeño número cinco, definido en la Sección 3.1.6.

Tabla 3.17: Requerimiento de Base de Datos RD04.

ID de requerimiento	**RD05**
Nombre de requerimiento	Contexto de páginas web.
Propósito	Almacenar los datos que describen el contexto de las páginas web resultantes del rastreo de contenido.
Descripción	- El prototipo tiene un archivo de texto, donde se almacena una tabla con los datos que describen el contexto de cada una de las páginas web rastreadas, que tengan contenido similar a la petición del usuario. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: URL y palabras que describan el contenido de la página web. - El archivo de texto se sobreescribe en cada sesión de búsqueda.

Tabla 3.18: Requerimiento de Base de Datos RD05.

ID de requerimiento	**RD06**
Nombre de requerimiento	Conjunto general de URLs.
Propósito	Contar con una tabla que contenga un conjunto de URLs con contenido de distintas categorías.
Descripción	- El prototipo cuenta con un archivo de texto donde se almacena una tabla con los datos de páginas web que tienen contenido de distintas categorías. Éstas sirven como semillas de inicio de rastreo de contenido y, dependiendo del contexto del usuario, se seleccionan datos de esta tabla y se agregan a la tabla definida en el requerimiento RD03 que se encuentra en el Cuadro 3.16. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: URL y categoría del contenido. - El archivo de texto no se actualiza automáticamente, ya que está predefinido.

Tabla 3.19: Requerimiento de Base de Datos RD06.

ID de requerimiento	**RD07**
Nombre de requerimiento	Diccionario de sinónimos de verbos.
Propósito	Contar con un diccionario de sinónimos de verbos. El diccionario se consulta para identificar si el verbo principal de una petición del usuario corresponde a una actividad que el modelo puede realizar. Por ejemplo: el verbo "poner"puede ser sinónimo de reproducir o mostrar.
Descripción	- El prototipo cuenta con un diccionario de sinónimos en español. - Los datos son guardados como texto, separados por comas (,). - Los datos que se almacenan son: actividad que puede realizar el modelo y sinónimos. - El archivo de texto no se actualiza automáticamente.

Tabla 3.20: Requerimiento de Base de Datos RD07.

3.2 Arquitectura del modelo

En la Figura 3.3 se muestra la estructura del modelo para navegación web, como se puede observar, el modelo se divide en diez módulos o áreas funcionales que se describen a continuación.

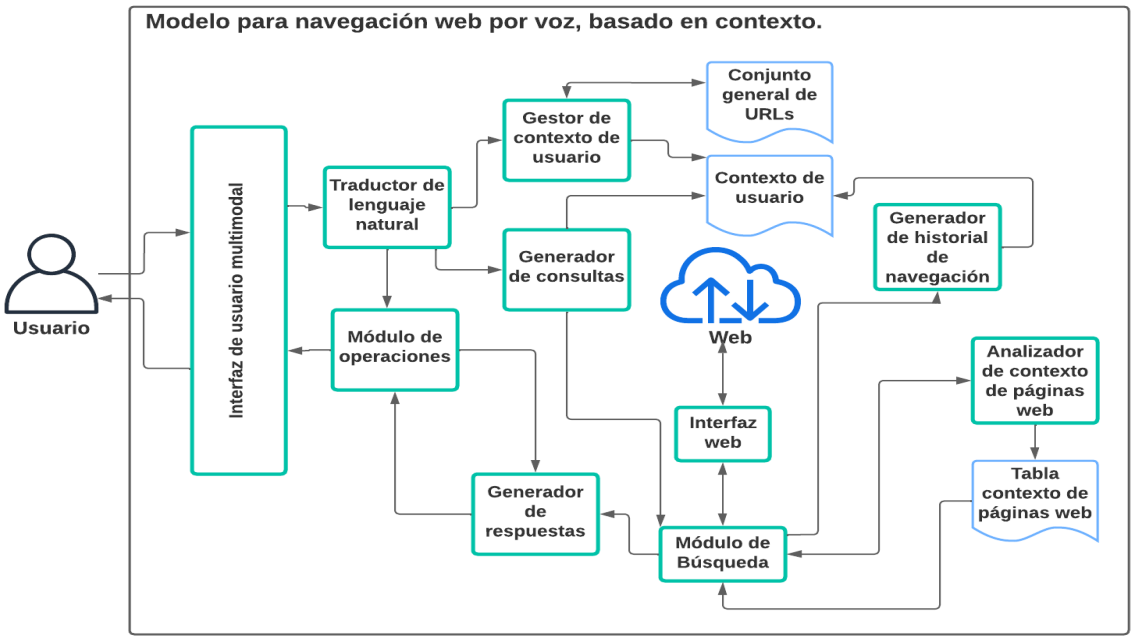


Figura 3.3: Arquitectura del modelo.

Interfaz de usuario multimodal (IUM)

Este módulo es el encargado de presentar información, resultados y acciones disponibles en formato audible o gráfico. Permite la interacción del usuario hacia el prototipo mediante voz, gestos táctiles y escritura. La IUM recibe datos desde el usuario en formato de audio, texto o señales obtenidas de gestos táctiles. Posteriormente, transforma los datos a cadenas de texto y envía dicho texto al **traductor de lenguaje natural**. También, la IUM recibe contenido en formato gráfico o audible desde el **módulo de operaciones** y después entrega el contenido al usuario.

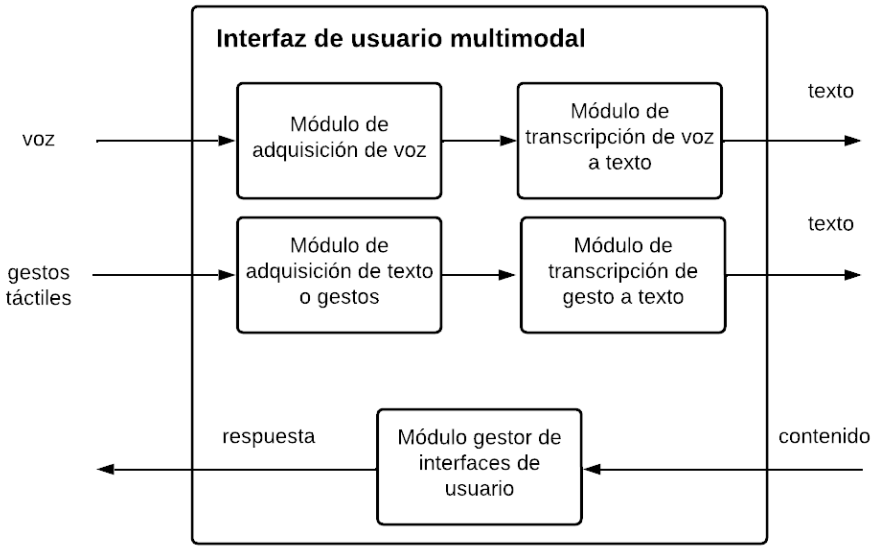


Figura 3.4: Interfaz de usuario multimodal.

Para lograr lo anterior, la IUM contiene los módulos que se muestran en la Figura 3.4 y se describen a continuación.

- **Módulo de adquisición de voz**: este módulo obtiene las señales de audio correspondientes a la voz del usuario.
- **Módulo de transcripción de voz a texto**: este módulo convierte en texto las peticiones por voz del usuario.

- **Módulo de adquisición de texto**: este módulo obtiene las señales correspondientes al texto introducido por el usuario en la pantalla táctil.

- **Módulo gestor de interfaces de usuario**: este módulo presenta al usuario el contenido, en una de las interfaces de despliegue de resultados, que se muestran en la Figura 3.1.

Traductor de lenguaje natural

Este módulo realiza dos tareas: primero, realiza el preprocesamiento del texto correspondiente a la petición del usuario. El preprocesamiento consta de limpieza, curación y almacenamiento breve de texto en lenguaje natural. Despúes, extrae unidades léxicas y realiza un proceso de extracción de información, a partir de las cadenas de texto que son recibidas en la IUM. Dependiendo de la actividad que esté realizando el usuario (establecer perfil de usuario o navegar), pasan las unidades léxicas o la información extraída al **gestor de contexto de usuario**, **generador de consultas** o al **módulo de operaciones**, respectivamente.

Gestor del contexto de usuario

Este módulo genera y actualiza la tabla correspondiente al contexto del usuario e historial de navegación. También, genera el vocabulario que contendrá términos descriptores del contexto del usuario y actualiza la tabla de URLs, con base al contexto de usuario. El gestor de contexto de usuario recibe datos desde el **traductor de lenguaje natural** que, posteriormente, procesa para tomar decisiones acerca de los datos, los cuales guarda, elimina o edita en las tablas **conjunto general de URL** y **contexto de usuario**. Las tablas mencionadas se describen a continuación.

- **Conjunto general de URL**: esta tabla contiene un compendio de URLs, abarcando sitios web de diversos dominios, que proporcionan información de diferentes

temas. Por ejemplo: sitios de difusión cultural, enciclopedias en línea, repositorios, corpus de textos, entre otros.

- **Contexto de usuario**: es una tabla que contiene datos y términos descriptores del perfil de usuario. Estos datos describen diversos aspectos del usuario.

Generador de historial de navegación

Este módulo permite crear y actualizar el historial de navegación, según la frecuencia de concurrencia en ciertos sitios y la frecuencia de búsqueda de un determinado tema. El módulo recibe las URLs de las páginas web que visita el usuario desde el **módulo de operaciones** y también, actualiza la tabla de **contexto de usuario**.

Analizador de contexto de páginas web

Este módulo obtiene los elementos de una página web que permiten conocer el tema, contenido, autor, fecha de publicación, palabras clave, elementos sintácticos, marcas y otra información relevante de la página. Con base a los elementos anteriormente mencionados, se generan términos descriptores del contexto de la página. Los términos descriptores del contexto de la página serán almacenados brevemente. Este módulo recibe del **módulo de búsqueda** una lista de URLs resultantes de un rastreo de páginas web y edita la **tabla de contexto de páginas web**. También, envía al **módulo de búsqueda** una bandera para indicarle si puede o no consultar los datos en **tabla de contexto de páginas web**.

Tabla de contexto de páginas web

Esta tabla contiene términos descriptores del contexto de un conjunto de páginas web resultantes del rastreo de contenido. En la Tabla 3.21, se presenta la estructura general. En ella, la primera columna corresponde a la URL de la página web, mientras que las columnas T1 hasta Tn representan los términos descriptores asociados a cada página web de cada fila. La tabla puede contener una cantidad variable de filas correspondientes a las

páginas web que se analicen. Al concluir cada solicitud del usuario, la tabla se elimina y se genera nuevamente al inicio de una nueva solicitud.

Tabla 3.21: Tabla de contexto de páginas web

Página web	T1	T2	...	Tn
www.youtube.com	videos	tendencias	...	otro
www.cinvestav.mx	educación	posgrado	...	otro

Módulo de búsqueda

Este módulo se encarga de buscar y ofrecer un conjunto de URLs como resultado a una consulta específica. Permite la navegación dentro de sitios web y realiza el emparejamiento de términos de búsqueda y contexto de usuario de las páginas web realizando las siguientes actividades.

- Generación de una lista doblemente ligada de URLs a partir de una página web en específico. La lista de URLs servirá para la navegación dentro de un sitio elegido por el usuario.
- Comparación entre los términos descriptores del contexto del usuario y los términos de búsqueda con los términos descriptores del contexto de las páginas web.
- Selección de las páginas web que tienen mayor porcentaje de coincidencias con el contexto del usuario y los términos de su búsqueda.

El módulo de búsqueda recibe la consulta del usuario y la(s) URL, mediante cadenas de texto, enviadas desde el **generador de consultas**; inspecciona páginas web a partir de las consultas y la(s) URL. Por lo tanto, este módulo cuenta con conexión a Internet mediante la **interfaz web**. También, genera una lista con la(s) URL coincidentes con la consulta y envía dicha lista al **analizador de contexto de páginas web**. Posteriormente, el módulo recibe una bandera del **analizador de contexto de páginas web**, que

le indica si puede acceder o no a la **tabla de contexto de páginas web**. El módulo de búsqueda selecciona y envía al **generador de respuestas** las páginas web que tienen el contexto de mayor coincidencia con la petición del usuario y el contexto del usuario.

Interfaz Web

Este módulo se encarga de gestionar la comunicación entre el módulo de búsqueda y la Web.

Generador de respuestas

Este módulo genera respuestas u opciones de navegación en lenguaje natural, con base a los resultados de la búsqueda. También, sintetiza en voz las respuestas u opciones y presenta resultados en formato gráfico. El generador de respuestas recibe una lista de contenido en HTML del **módulo de búsqueda**, extrae información, genera respuestas en lenguaje natural del contenido con mayor similitud con el contexto de usuario y su petición, posteriormente, envía al **módulo de operaciones** las repuestas. También, recibe un dato del **módulo de operaciones** que le indica al generador de respuestas en qué formato enviar el contenido y que elemento de la lista de contenido en HTML debe enviar al módulo de operaciones.

Generador de consultas

Como lo dice su nombre, este módulo genera consultas basadas en la petición y el contexto del usuario. Una consulta está compuesta de operadores de búsqueda y filtros. Se encarga de seleccionar una URL semilla, desde la cual se parte para buscar contenido relacionado con la petición. Dicha URL se elige de la tabla **conjunto general de URLs**. El generador de consultas recibe datos del **traductor de lenguaje natural** y con base en estos datos genera una cadena de consulta, establece una o varias URL de inicio para comenzar la búsqueda de contenido. También, accede a la información de la tabla **contexto de**

usuario y envía al **módulo de búsqueda** la consulta y la(s) URL(s) semilla.

Módulo de operaciones

Este módulo se encarga de realizar tareas como ubicarse en alguna sección y accionar eventos dentro de la página web o del navegador. También, canaliza datos hacia el generador de consultas y envía los resultados a la **IUM** para presentarlos al usuario. El módulo de operaciones, por un lado, recibe respuestas en formato de audio, gráfico o texto, desde el **generador de respuestas**. También, recibe datos desde el **intérprete de lenguaje natural**, para transformarlos en instrucciones y después entregar el contenido mediante la **IUM** al usuario. Por otro lado, envía al generador de respuestas una bandera que indica la opción de la lista de resultados y el formato que requiere.

3.3 Manejo del contexto

Como se mencionó en la Sección 2.2, la conciencia del contexto es un tema relevante en el campo del Cómputo Ubicuo. Los sistemas conscientes del contexto aprovechan los datos del entorno y del usuario para ofrecer un mejor servicio. En la navegación web, intervienen diversas entidades, como el usuario, el navegador, las páginas web y el dispositivo en el que se aloja el navegador, entre otras. La información de estas entidades puede utilizarse para ofrecer mejores resultados y mejorar la experiencia del usuario durante esta actividad. En esta propuesta, se analiza del contexto del usuario y del contenido de las páginas web. A continuación, se explica cómo son manejados estos aspectos.

3.3.1 Contexto de usuario

Para manejar el contexto del usuario es necesario contar con una representación computacional de los aspectos que definen las características individuales del usuario y cómo se relaciona con su entorno. Esto permitirá que el sistema se adapte y ajuste su funcionalidad de acuerdo a dicha información. Actualmente, no existe un consenso para el modelado

del contexto de alguna entidad. Los modelos de contexto tienden a ser específicos del dominio del problema que se está abordando. Esto se debe a que el contexto puede variar significativamente dependiendo del ámbito en el que se aplique, ya sea salud, educación, transporte, entre otros.

3.3.1.1 Datos de usuario

En este trabajo se modela el contexto de usuario en el ámbito de la navegación web. Para ello se analizan las siguientes cinco facetas del usuario:

1. **Datos demográficos**: estos datos proporcionan información importante sobre características de la población a la que pertenece el usuario. Los datos demográficos que se usan en esta propuesta son:
    - **Edad:** permite comprender las distintas etapas de la vida que atraviesa el usuario y cómo éstas pueden influir en sus necesidades, comportamientos y preferencias.
    - **Género:** permite comprender la percepción del mundo que tiene el usuario, sus roles y patrones de consumo.
    - **Dirección:** brinda información sobre el entorno donde vive el usuario, características geográficas y culturales que pueden influir en su comportamiento y necesidades específicas.
    - **Nivel educativo:** permite comprender el nivel de conocimientos del usuario, habilidades y acceso a oportunidades, lo cual puede tener un impacto en su comportamiento, toma de decisiones y perspectivas.
    - **Estado civil**: proporciona información sobre las relaciones y la estructura familiar del usuario.
2. **Salud**: estos datos proporcionan información acerca del estado general de salud o capacidades físicas del usuario.

3. **Temas de interés:** en la Web, se puede encontrar una amplia variedad de contenido sobre diversos temas como política, ciencia, tecnología, arte, historia, entre otros. Conocer los temas de interés del usuario permite mejorar su experiencia de navegación al facilitar la búsqueda y acceso a la información.

4. **Afinidades personales:** estos datos permiten conocer los intereses, gustos o preferencias del usuario. En esta propuesta, se consideran los siguientes datos:

   - **Actividades y pasatiempos:** permite conocer las actividades que el usuario disfruta realizar en su tiempo libre, como deportes, música, lectura, arte, cine, viajes, cocina, entre otros.
   - **Música y películas favoritas:** permite conocer los géneros musicales y cinematográficos preferidos por el usuario, así como las películas y música que más consulta. Además, brinda la oportunidad de conocer los artistas, bandas o directores que el usuario aprecia.
   - **Libros y lecturas:** permite conocer los géneros literarios que prefiere el usuario, si tiene autores o libros favoritos, o si le gusta leer novelas, ensayos, biografías u otros géneros.
   - **Deportes y actividades físicas:** permite conocer si el usuario tiene interés en algún deporte en particular, si le gusta hacer ejercicio, participar en actividades al aire libre o si sigue algún equipo o competencia deportiva.
   - **Viajes y destinos:** permite conocer información sobre lugares que ha visitado el usuario o le gustaría visitar, sus experiencias de viaje o los tipos de destinos que encuentra interesantes.
   - **Comida y gastronomía:** permite conocer preferencias culinarias, si disfruta de algún tipo de cocina en particular o si tiene restaurantes o platillos favoritos.

5. **Preferencias de sitios web:** permite conocer las preferencias individuales del usuario al elegir utilizar ciertos sitios web para buscar información.

Los datos del contexto del usuario serán recopilados a través de una encuesta por voz llevada a cabo durante la fase de configuración del prototipo. La encuesta está conformada por preguntas en su mayoría cerradas, pero también contendrá preguntas abiertas.

3.3.1.2 Procesamiento de datos

Dado que las respuestas del usuario a la encuesta serán en formato de voz, en esta etapa se realiza el reconocimiento del habla. Posteriormente, se lleva a cabo el filtrado de los datos mediante el análisis y validación de las respuestas a las preguntas cerradas y finalmente se actualiza la tabla "Contexto Usuario". En el caso de las respuestas a las preguntas abiertas, éstas se analizan utilizando técnicas de procesamiento del lenguaje natural. En la Figura 3.5 se muestra el diagrama de flujo que representa el procesamiento de las respuestas a las preguntas abiertas, para generar términos descriptores del contexto del usuario. A continuación, se explican cada uno de los pasos en detalle.

Preprocesamiento: en este paso, se tiene como entrada la respuesta del usuario en formato de texto. Se convierte el texto a minúsculas, se divide el texto en palabras y se eliminan las palabras irrelevantes.

Análisis sintáctico: en esta etapa, se utiliza el texto resultante del preprocesamiento como entrada. En las respuestas que consisten en dos o más palabras, se etiquetan las palabras según su categoría gramatical, como sustantivos, verbos, adjetivos, adverbios, entre otros. Además, se realiza un análisis de las relaciones de dependencia entre las palabras. Como resultado, se genera un conjunto de datos que contiene las palabras con su categoría gramatical correspondiente y las etiquetas de las relaciones de dependencia que existen entre palabras.

Generación de términos descriptores de contexto: en este paso se tiene como entrada el conjunto de datos generado en el análisis sintáctico. Se filtran los sustantivos y

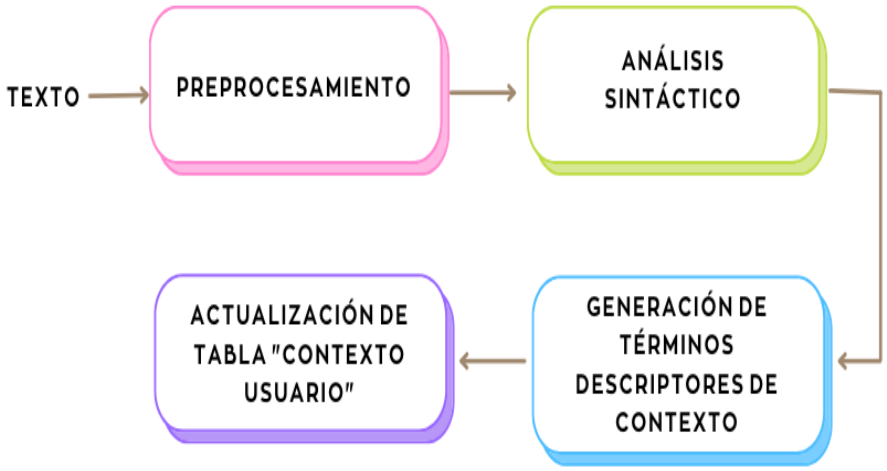


Figura 3.5: Procesamiento de datos para generar contexto de usuario.

las palabras que componen las relaciones de dependencia siguientes:

- **Modificador adjetival (amod)**: indica la relación de dependencia en la cual el adjetivo describe o califica al sustantivo. Por ejemplo, en la frase "me gusta escuchar música clásica", el modificador adjetival "clásica"modifica al sustantivo "música", describiendo el tipo de música que escucha el usuario.

- **Modificador nominal (nmod):** indica la relación de dependencia en la cual el modificador nominal agrega información adicional o específica características del sustantivo al que se refiere. Por ejemplo, en la frase "leer libros de historia", la relación nmod indica que "de historia"modifica o describe el tipo de libros que lee el usuario.

Actualización de tabla "Contexto de Usuario"(CU): por último, se seleccionan las palabras que conforman la relación de dependencia como términos descriptores y se guardan en la tabla del CU. La tabla del CU está diseñada de forma que cada fila contiene una categoría de datos específica y sus palabras clave correspondientes. En cada fila, la primera columna índica el tipo de dato y las demás columnas contienen las palabras clave

de ese tipo de dato (ver Tabla 3.22).

Tabla 3.22: Tabla contexto de usuario

Tema	Palabras clave			
	p_1	p_2	...	p_n
Edad	diez	niño	...	infantil
Género	masculino		...	
Dirección	CDMX	Colonia Centro	...	Rep. Salvador
.	.	.	...	.
.	.	.	...	.
.	.	.	...	.
Preferencias de sitios web	youtube.com	cinvestav.mx	...	mercadolibre.com.mx

3.3.2 Contexto de las páginas web

En este trabajo, se utiliza el término "contexto de página web" para referirse a los datos que se pueden extraer del contenido de una página, con el fin de determinar su utilidad y relevancia. Estos datos permiten tomar decisiones sobre qué contenido presentar al usuario, de acuerdo con sus intereses, preferencias o necesidades. En la Sección 3.2 se describe el funcionamiento del analizador de contexto del contenido web. A continuación, se presenta en detalle el proceso de obtención de datos y los aspectos clave que se incluirán en el contexto.

3.3.2.1 Datos de las páginas web

Existe variabilidad en la disponibilidad de datos que se pueden obtener de las diferentes páginas web. Esto se debe a la diversidad de enfoques y decisiones tomadas por los desarrolladores al diseñar y estructurar sus sitios web. Cada desarrollador tiene la libertad de determinar qué información incluir y cómo organizarla en sus páginas web, lo que resul-

ta en diferencias significativas en los datos disponibles para el análisis de contexto. Por ejemplo, en una página web específica es posible encontrar el nombre del autor y la fecha de publicación, mientras que en otra página estos datos pueden estar ausentes. Teniendo esto en cuenta, el análisis se enfoca en recuperar los siguientes datos:

- título
- autor
- fecha de publicación
- subtítulos
- palabras clave del contenido
- indicador de multimedia

Los elementos listados proporcionan una idea general del tema, enfoque y aspectos específicos que se tratan en la página. También, permiten identificar si la página está actualizada y si incluye imagénes, videos, u otros elementos multimedia.

3.3.2.2 Procesamiento de los datos

Para iniciar la extracción de datos, se parte de una lista de URLs de páginas web que están relacionadas tanto con las preferencias del usuario como con la petición que realiza. La lista de URLs se genera mediante un crawler para la búsqueda de contenido. Posteriormente, se comienza el proceso de extracción de datos de cada una de las páginas web que se encuentran en la lista de URLs. Cada página web se somete al proceso ilustrado en la Figura 3.6 que se explica a continuación.

Figura 3.6: Generación de términos descriptores del contexto de las páginas web.

Recopilación del contenido: se envía la solicitud HTTP al servidor para obtener el HTML de la página web, mientras se realiza el manejo de posibles errores.

Análisis de metadatos: el objetivo de este proceso es identificar y buscar el contenido

de las etiquetas HTML, tales como títulos, descripciones, introducciones, palabras clave definidas por el desarrollador de la página, nombre del autor y fecha de publicación. Esto se realiza con el fin de obtener información adicional relevante. Para lograrlo, se siguen los siguientes pasos.

- Identificar los elementos HTML específicos mediante clases, etiquetas o identificadores (IDs).
- Seleccionar los elementos HTML que contengan los datos de interés.
- Extraer y almacenar el contenido relevante como texto, imágenes, enlaces y videos.

Extracción de los datos: se aplican técnicas de procesamiento del lenguaje natural para comprender el contenido, identificar entidades relevantes y finalmente extraer las palabras clave que describan el contexto de la página web. Para lograr lo anterior, se realizan los siguientes pasos:

- Separar el texto por oraciones.
- Eliminar elementos no deseados como símbolos, etiquetas y signos de puntuación.
- Separar cada oración en palabras individuales (Tokenizar).
- Etiquetar las palabras según su categoría gramatical.
- Obtener las relaciones de dependencia entre palabras.
- Seleccionar las palabras que tengan relaciones de dependencia del tipo **amod**[1], **nmod**[2] y **nsubj**[3].
- Detectar entidades nombradas y seleccionar las que pertenezcan a los siguientes grupos:

[1]Indica que un adjetivo modifica a un sustantivo. Ejemplo:"manzana roja".
[2]Indica que un sustantivo modifica a otro sustantivo. Ejemplo:"libro de geografía"
[3]Indica que un sustantivo realiza la acción principal en una oración. Ejemplo:"El perro ladra"

- personas: nombres de individuos, como “Citlalli Avalos”.
- ubicaciones geográficas: nombres de lugares, como “México”.
- organizaciones: nombres de empresas, instituciones u organizaciones, como “Google” o “Cinvestav”.
- fechas: expresiones de tiempo o fechas, como“hoy” o “el 12 de julio de 2023”.

Después del proceso anterior, se tiene un conjunto de palabras seleccionadas y se eliminan las palabras repetidas, asegurando así que cada palabra se represente una sola vez en el conjunto final de palabras clave que describen el contexto del contenido de la página web. Por último, se agrega una fila en la tabla de contexto de páginas web con el URL y las palabras clave de la página analizada.

3.3.3 Emparejamiento de contextos

Como se menciona en las secciones anteriores, en esta propuesta se representa el contexto del usuario y de las páginas web mediante palabras clave. Además, se tiene en cuenta la petición del usuario, de la cual también se pueden extraer palabras clave relevantes. Con base en la comparación y análisis de los conjuntos de palabras clave, se busca encontrar el contenido más apropiado para el usuario. El objetivo de la comparación es determinar qué conjunto de palabras clave del contexto de páginas web tiene mayor similitud con las palabras clave del contexto y petición del usuario.

Las medidas de similitud y distancia son empleadas en diversos campos para cuantificar y comparar la similitud o diferencia entre conjuntos o elementos. Algunas medidas de similitud y distancia están acotadas en el rango de cero a uno. En la Tabla 3.23, se enlistan las medidas de similitud y distancia empleadas mayormente en la comparación de cadenas de texto (palabras u oraciones).

Medida de Similitud	**Descripción**
Distancia de Hamming	Cuenta el número de posiciones en las que las cadenas difieren.
Distancia de Levenshtein	Mide el número mínimo de operaciones para convertir una cadena en otra.
Coeficiente de Dice	Evalúa la similitud entre dos conjuntos de caracteres.
Coeficiente de Jaccard	Calcula la similitud entre conjuntos basada en la intersección y unión de elementos.
Coeficiente de similitud del coseno	Mide el ángulo entre dos vectores de términos en espacios vectoriales.
Distancia de Jaro Winkler	Cuantifica la similitud entre dos cadenas teniendo en cuenta prefijos comunes.

Tabla 3.23: Medidas de similitud y distancia.

Dado que en este problema se desean comparar conjuntos de palabras entre sí se eligió utilizar coeficiente de Jaccard por las siguientes razones.

- Es adecuado para comparar conjuntos.
- Permite considerar únicamente la presencia o ausencia de palabras en los conjuntos, sin importar el orden o la frecuencia de aparición.
- No toma en cuenta duplicados dentro de los conjuntos.
- Interpretación intuitiva.

El proceso de emparejamiento de contexto consta de dos fases, como se ilustra en la Figura 3.7 y se explica a continuación.

- **Fase 1:** identificación de coincidencias entre las palabras clave extraídas de la petición del usuario (**pcpu**) y las palabras clave del contexto del usuario (**pccu**). Se

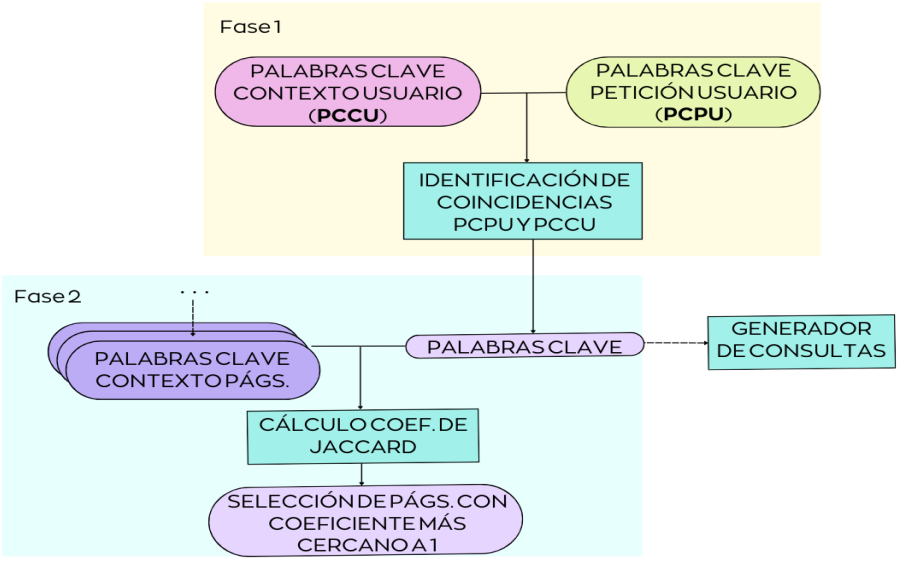


Figura 3.7: Fases del emparejamiento de contextos.

toman los valores de la primera columna de la tabla del CU (ver Tabla 3.22) y se comparan con las **pcpu**. Al encontrar una coincidencia, se agregan las **pccu** de determinada fila a las **pcpu**.

- **Fase 2:** cálculo de coeficiente de Jaccard entre las palabras clave resultantes de la identificación de coincidencias (**pcr**) y las palabras clave del contexto de cada una de las páginas web (**pccp**). Como se mencionó anteriormente, el coeficiente de Jaccard es una medida númerica que permite conocer el grado en que dos conjuntos de datos son similares. Los valores de este coeficiente están entre los valores cero, cuando no hay ninguna similitud, y uno, cuando hay similitud total. El coeficiente de Jaccard se define como el tamaño de la intersección de los conjuntos dividido por el tamaño de su unión [43]. Matemáticamente, se expresa de la siguiente manera:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Donde:

- A y B son conjuntos de datos.
- $|A \cap B|$ representa la cardinalidad de la intersección de A y B, es decir, el número de elementos que están presentes tanto en A como en B.
- $|A \cup B|$ representa la cardinalidad de la unión de A y B, es decir, el número de elementos únicos en ambos conjuntos.

Finalmente, son seleccionadas aquellas páginas web cuyo conjunto de **pccp** ha obtenido una alta similitud con el conjunto **pcr**. Es decir, que posean un coeficiente de Jaccard J(**pccp**, **pcr**) cercano a 1. Las páginas seleccionadas son catalogadas como posibles resultados para desplegar al usuario.

3.4 Análisis de herramientas

En esta sección se reportan los resultados del análisis de herramientas para el desarrollo del prototipo, contemplando sus características y considerando cómo se ajustan a las necesidades específicas del proyecto.

3.4.1 Sistema operativo Android

Según la ENDUTIH [4] al año 2022, el smartphone encabeza la lista de dispositivos preferidos por los usuarios para acceder a Internet debido a que el 97 % de las conexiones fueron a través de este equipo, seguido de las computadoras portátiles y/o tabletas con el 31 % de las conexiones. Cabe mencionar que muchos usuarios optan por utilizar más de un equipo para conectarse, lo que explica el hecho de que la suma de los porcentajes sea mayor al 100 %. La información anterior refleja que los dispositivos móviles, como smartphones y tabletas, se han convertido en una parte esencial de la vida cotidiana al brindar una amplia gama de funcionalidades y recursos. Por esto, se eligió desarrollar una aplicación para dispositivo móvil. En el núcleo de estos dispositivos se encuentran los sistemas operativos.

Figura 3.8: Cuota de mercado de sistemas operativos para móviles y tabletas en México 2022 a 2023 [1].

Según los datos recopilados por StatCounter[1] en el lapso de junio de 2022 a junio de 2023, el sistema operativo Android continúa dominando el mercado de dispositivos móviles en México, con una presencia del 77.35 %, seguido del sistema operativo iOS con una presencia de 22.34 % y el 0.31 % restante distribuido en otros sistemas operativos como se muestra en la Figura 3.8. Debido a la presencia dominante de Android en el mercado de dispositivos móviles, se tomó la decisión de desarrollar el prototipo mediante una aplicación móvil diseñada para este sistema operativo.

La plataforma Android va evolucionando mediante nuevas versiones, en las cuales se introducen mejoras de rendimiento, características, funcionalidades, nuevas API para los desarrolladores y actualizaciones de seguridad para los dispositivos móviles. Según el entorno de desarrollo integrado Android Studio, la gama de versiones disponibles hoy en día va desde la versión 4.4 Kitkat hasta la 13 T.

Como se muestra en la Figura 3.9, la versión API 21: Android 5.0 (Lollipop) se encuentra disponible en aproximadamente 99.5 % de los dispositivos disponibles, en otras palabras, si se desarrolla una aplicación tomando como base dicha versión, la aplicación se ejecutará en

[1]StatCounter es un sitio web de análisis de tráfico en tiempo real. Disponible en: https://statcounter.com/

aproximadamente en el 99.5 % de los dispositivos.

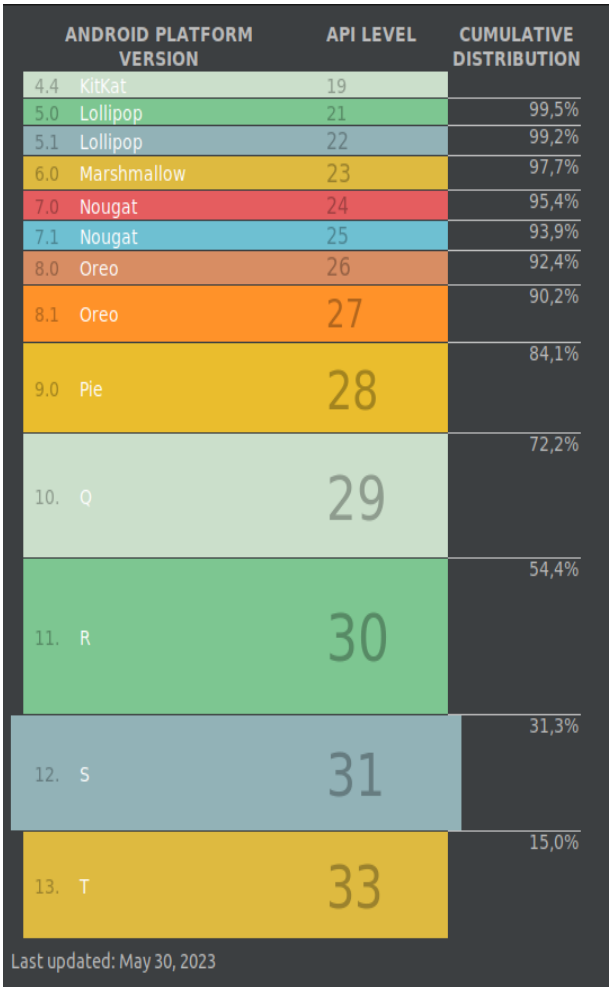


Figura 3.9: Versiones de Android

Sin embargo, el desarrollo en una versión reciente permite ofrecer una experiencia moderna, funcional y segura al usuario, además de aprovechar las últimas innovaciones en la plataforma Android.

Se decidió desarrollar el prototipo en la versión API 31: Android 12 S en función de su disponibilidad y accesibilidad para la investigación. Como se puede observar en la Figura 3.9, el prototipo es capaz de correr en aproximadamente el 33 % de los dispositivos y dado que las versiones anteriores eventualmente quedarán obsoletas, la aplicación se mantendrá vigente en el futuro cercano.

3.4.2 DialogFlow

DialogFlow es una herramienta de Inteligencia Artificial (IA) conversacional para el desarrollo de bots de chat y de voz, basada en la nube creada por Google. Esta herramienta usa tecnologías como procesamiento de lenguaje natural, reconocimiento del habla y reconocimiento de entidades nombradas (NER) para identificar o inferir la intención, las entidades y el contexto de lo que dice el usuario. De esta forma, el sistema que emplea es-

ta herramienta se vuelve capaz de proporcionar respuestas eficientes y precisas a través de la interfaz de usuario [44]. Algunos de los aspectos importantes de DialogFlow se enlistan a continuación:

- Permite crear una experiencia de usuario natural y conversacional.
- Utiliza técnicas de PLN y aprendizaje automático para entender las intenciones de los usuarios y responder de manera coherente.
- Proporciona una plataforma amigable para crear chatbots y agentes de conversación sin requerir un profundo conocimiento en IA o PLN.
- Recibe actualizaciones y soporte continuo de Google Cloud.
- Ofrece plantillas y reglas predefinidas para crear la interacción, lo que puede limitar el control total sobre la experiencia del usuario.
- Dependiendo del volumen de uso y la cantidad de interacciones, el uso de Dialogflow puede implicar costos adicionales.
- Funciona en la nube, por lo tanto, el prototipo requiere una conexión a Internet para comunicarse con el servicio.
- Soporta el idioma español.
- Se encuentra disponible para su integración en diversos entornos, entre ellos Android Studio[1].

En este proyecto, es fundamental que la interfaz proporcione una experiencia natural al usuario, evitando la necesidad de que éste tenga que aprender formas estructuradas o definidas para realizar una petición. Puede ocurrir que los usuarios olviden dichas formas para realizar una petición, lo que puede ocasionar molestias, confusión y una experiencia

[1]Entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones para dispositivos Android.

negativa en general. El objetivo es crear una interfaz que sea intuitiva. Se requiere que la interacción sea lo más fluida y natural posible, permitiendo que el usuario se exprese en lenguaje común y la interfaz pueda interpretar y comprender sus solicitudes sin dificultad. Tomando esto en cuenta, y después de analizar la herramienta, se determinó que esta herramienta sirve para el desarrollo del prototipo.

Nombre de la API	**Descripción o Características**
Google Text-to-Speech (TTS) API	API proporcionada por Google que permite convertir texto en voz en múltiples idiomas y voces generadas por WaveNet. Permite ajuste de tono, velocidad del habla, aumento de volumen y selección de formato de audio (MP3, Linear16, OGGOpus, entre otros) [45].
Amazon Polly	Servicio de síntesis de voz de Amazon Web Services (AWS) que ofrece varias voces y soporte para múltiples idiomas. Ofrece Neural Text-to-Speech (NTTS) que mejora la calidad del habla, que se refleja en voces más naturales y humanas [46].
IBM Watson Text-to-Speech	Servicio de síntesis de voz en la nube con soporte al idioma español. Produce audio en formatos populares [47].
Text to Speech Android	Clase de Android que permite generar texto en voz. Capaz de generar archivo de audio en formatos populares. Permite el ajuste de tono, velocidad del habla y aumento de volumen de audio [48].

Tabla 3.24: APIs de síntesis de voz disponible para Android

Tabla 3.25: APIs de reconocimiento de voz disponible para Android

API	**Descripción/Características**
Speech Recognizer	1. Captura y reconoce la entrada de voz del usuario en tiempo de ejecución. 2. Utiliza los servicios de reconocimiento de voz proporcionados por el sistema operativo Android. 3. Los métodos de esta clase solo pueden ser invocados desde el subproceso principal de la aplicación 4. Transmite el audio a servidores remotos. 5. No está diseñada para usare para reconocimiento continuo [49].
Speech to Text Google	1. Convierte voz en texto basado en tecnologías de IA de Google. 2. Procesa señales de audio captadas por el micrófono o enviadas desde un archivo de audio pregrabado. 3. Realiza tratamiento del ruido [50].
Recognizer Intent	1. Clase de Android que proporciona una forma sencilla de realizar reconocimiento de voz. 2. Realiza el reconocimiento de voz mediante el inicio de un "Intent". 3. No requiere de APIs extra o servicios en la nube. 4. Procesa señales de audio captadas por el micrófono [51].
Pocket Sphinx	1. Es una librería de código abierto que ofrece reconocimiento de voz sin necesidad de conexión a internet. 2. Puede ser utilizada en versiones antiguas de Android. 3. No requiere conexión a Internet para realizar el reconocimiento. 4. Diseñado para funcionar en dispositivos con recursos limitados, como teléfonos móviles y sistemas embebidos. 5. Soporte limitado en cuanto a lenguaje español [52].

3.4.3 Servicios de sintetización de texto a voz

El objetivo de los servicios de sintetización de texto a voz es mapear las cadenas de texto a formas de onda para posteriormente concatenarlas y crear una señal de audio. En esta propuesta se requiere que el contenido se entregue al usuario en formato de audio, por lo tanto, se analizaron herramientas que permitan agregar este servicio a una aplicación desarrollada para el sistema operativo Android, como se muestra en la Tabla 3.24.

3.4.4 Servicio de reconocimiento de voz

Antes de procesar la frase del usuario mediante técnicas de PLN, es necesario reconocer las palabras que expresa el usuario mediante la voz y convertirlas en texto. A esta tarea se le llama reconocimiento de voz. El reconocimiento de voz consiste en captar audio a través de un micrófono u otra fuente de audio, y convertirlo en texto utilizando algoritmos avanzados y/o modelos de aprendizaje automático. Para el sistema operativo Android, están disponibles diversas herramientas y APIs[1] que permiten agregar características de reconocimiento de voz a las aplicaciones móviles. En la tabla 3.25 se enlistan cuatro de las herramientas de reconocimiento de voz disponibles y sus características destacables.

3.4.5 Servicio de navegación y control por voz

Uno de los objetivos de la propuesta es la implementación de la función de comandos de voz en lenguaje natural, que permite a los usuarios interactuar con las páginas web sin necesidad de utilizar hardware convencional, como el ratón o las pantallas táctiles. En lugar de depender de clics y desplazamientos manuales, los usuarios simplemente podrán dar instrucciones a través de su voz, lo que ofrece una experiencia de navegación más intuitiva y amigable. Después de entender la intención de la petición del usuario se deben generar los eventos correspondientes, algunos de éstos pueden ser ejecutados mediante líneas de código que interactúan directamente con la IU. Sin embargo, algunas de las acciones tienen que ver con el contenido web que se presenta al usuario. Para este último

[1]Interfaz de Programación de Aplicaciones

tipo de acciones se propone utilizar alguna herramienta de automatización de pruebas disponibles para aplicaciones Android. Estas herramientas permiten a los equipos de desarrollo realizar pruebas de manera más eficiente, estandarizada, repetible y precisa, sin la necesidad de interacción manual constante del ingeniero de pruebas [53]. Algunas de las funciones comunes de estas herramientas y que son útiles para este proyecto son:

- Interacción con la interfaz de usuario: las herramientas de automatización de pruebas pueden interactuar con la interfaz de usuario de la aplicación, como hacer clic en botones, ingresar texto en campos de entrada, seleccionar elementos de listas, etc.
- Pruebas en emuladores y dispositivos reales: las herramientas pueden realizar pruebas tanto en emuladores como en dispositivos físicos.
- Admiten diversos lenguajes de programación, como Java, Python, JavaScript, entre otros.

En la Tabla 3.26 se enlistan algunas de las herramientas de automatización de pruebas que se ajustan a las necesidades del proyecto. Todas las herramientas mencionadas son de licencia libre y cuentan con foros y una comunidad activa de desarrolladores y usuarios que contribuyen, mantienen y mejoran las herramientas constantemente.

Tabla 3.26: Herramientas de automatización de pruebas para aplicaciones Android

Herramienta	**Interacción IU**	**Soporte multi-plataforma**	**Integración con lenguajes**
Espresso	Sí	No	Java/Kotlin
UI Automator	Sí	No	Java/Kotlin
Appium	Sí	Sí	Java/Kotlin y más.
Calabash	Sí	Sí	Ruby/Cucumber
Selendroid	Sí	Sí	Java
Detox	Sí	Sí	JavaScript

Capítulo 4

Desarrollo

En este capítulo, se describe el desarrollo del prototipo basado en la arquitectura propuesta en el capítulo anterior. Se abordan las actividades relacionadas con la implementación, prueba, depuración e integración de cada uno de los componentes de la arquitectura. Como se mencionó en el capítulo anterior, el prototipo se implementó en el sistema operativo Android, como una aplicación diseñada para la versión Android 12 S. La aplicación del prototipo se desarrolló utilizando el lenguaje de programación *Kotlin*. También se utilizó el entorno de desarrollo oficial Android Studio. La elección de esta plataforma de desarrollo permitió una integración más sencilla con el sistema operativo Android, y una mayor facilidad para adaptar la interfaz de usuario a las últimas directrices de diseño y funcionalidades de la versión 12 S. A lo largo de las siguientes secciones, se presentan en detalle los aspectos técnicos de la implementación, incluyendo las funcionalidades clave y los desafíos encontrados durante el desarrollo.

4.1 Implementación de la interfaz de usuario (IU)

Como se ha mencionado a lo largo de la tesis, la IU es una parte fundamental de la interacción entre el usuario y la aplicación. La interacción debe darse mediante la voz en lenguaje natural y se debe proporcionar al usuario contenido en formato audible o gráfico.

Por ello, el desarrollo de la IU se ha dividido en tres aspectos clave: reconocimiento de voz, síntesis de voz e interfaz gráfica.

4.1.1 Reconocimiento de voz

Según el requerimiento de usabilidad RU03 en la Tabla 3.13 y el requerimiento funcional 1 en la Tabla 3.1, el prototipo debe activarse mediante la voz en cualquier momento, por lo que el prototipo debe permanecer "escuchando" hasta que el usuario lo active mediante el comando de voz "*comando*". Para desarrollar este módulo, se exploraron las API de reconocimiento de voz disponible para Android, enlistadas en la Tabla 3.25, como se describe a continuación.

4.1.1.1 Primera implementación

En la primera implementación, se optó por desarrollar el módulo de reconocimiento de voz en primer plano. Esta decisión fue respaldada por la documentación oficial para desarrolladores de Android [54], donde se menciona que el enfoque de primer plano es adecuado cuando la funcionalidad, en este caso el reconocimiento de voz, es parte fundamental de la interacción del usuario con la aplicación y requiere una retroalimentación inmediata. Se empleó la clase `Speech Recognizer` que, como se menciona en la Tabla 3.25, sus métodos solo pueden ser invocados desde un hilo principal de la aplicación. El objetivo de esta implementación fue habilitar el reconocimiento de voz al iniciar la actividad principal, manteniéndolo activo para reconocer la voz del usuario y mostrar el texto reconocido en pantalla durante la ejecución de la actividad principal. Para lograr este propósito, se siguieron los siguientes pasos:

- **Configuración del proyecto**: primero se edita el archivo `Gradle` a nivel del módulo de la aplicación. Aquí se definen las dependencias específicas, las configuraciones de repositorios y otra configuración relacionada con la compilación del módulo. Para utilizar `Speech Recognizer`, se agrega la siguiente dependencia

de *Google Play Services* para reconocimiento de voz.

```
implementation 'com.google.android.gms:play-services-speech:20.1.0'
```

- **Solicitar permisos**: se incorporó la etiqueta que se muestra a continuación en el archivo de manifiesto `AndroidManifest.xml`, con el propósito de solicitar el permiso del usuario para la grabación de audio mediante el micrófono del dispositivo.

```
<uses-permission android:name="android.permission.RECORD\_AUDIO"/>
```

- **Implementación en la actividad principal**: en este paso, se sigue un orden específico en el código. Primero se importan las clases y paquetes necesarios para que el código funcione correctamente. Específicamente las clases necesarias para utilizar el Servicio de Reconocimiento de Voz (SRV) proporcionado por Android son:

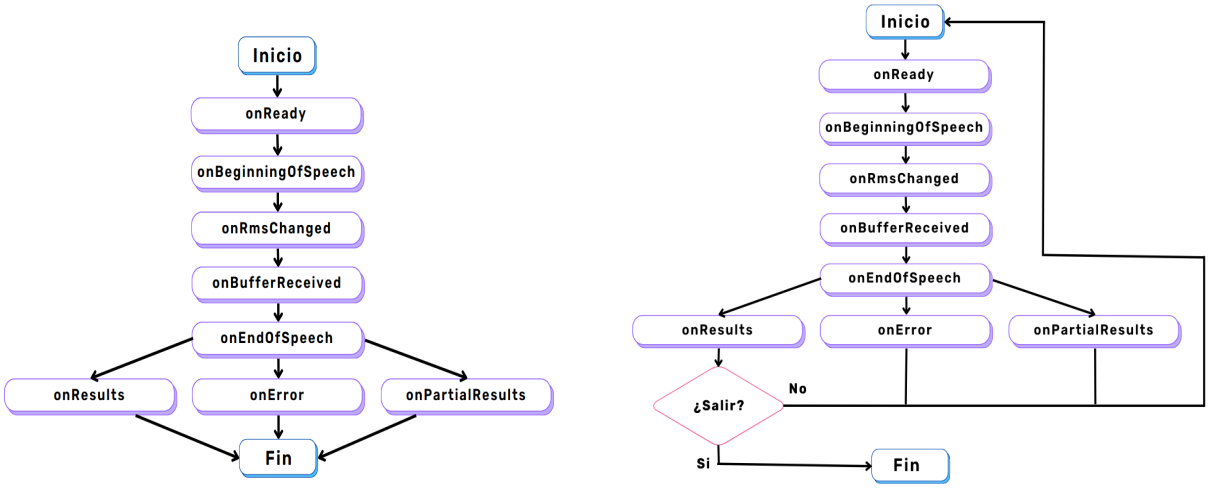
  - `android.speech.RecognitionListener`: contiene los métodos de *call-back* que se invocan cuando ocurren eventos en el reconocimiento de voz, por ejemplo: resultados, errores, notificaciones, por mencionar algunos.
  - `android.speech.Recognizer Intent`: contiene constantes y funciones que permiten configurar el reconocimiento.
  - `android.speech.SpeechRecognizer`: proporciona acceso al SRV de Android.

  Después, se declara la clase `MainActivity`, seguida de las variables utilizadas en la actividad como la instancia de `Speech Recognizer` y de `Speech Recognizer Intent`. Posterior a esto, se realiza la configuración inicial de la actividad con el método `onCreate()`, por ejemplo, asignar la interfaz de usuario (*layout*) con `setContentView()`, inicializar las variables y configurar el objeto de `Speech`

`Recognizer`. La configuración del objeto de tipo `Speech Recognizer Intent` consiste en crear una instancia de la clase `Intent` con la acción `ACTION_RECOGNIZE _SPEECH` para realizar el reconocimiento de voz. Además, se establecen los siguientes valores:

- `EXTRA_LANGUAGE_MODEL` en `LANGUAGE_MODEL_FREE_FORM`, para reconocer el habla libre sin restricciones específicas de formato o estructura.
- `EXTRA_LANGUAGE` en `Locale.getDefault()`, para utilizar el idioma predeterminado del dispositivo.

También, se creó una instancia de la clase `RecognitionListener`, para manejar los eventos relacionados con el reconocimiento de voz, como el inicio y fin del habla, los resultados del reconocimiento y los errores, entre otros.



(a) Original. (b) Primera implementación.

Figura 4.1: Diagrama de flujo de `RecognitionListener` para manejar eventos.

Es importante mencionar que el servicio de `Speech Recognizer` en Android fue diseñado para capturar y procesar frases o comandos cortos. Por esta razón, los métodos que se invocan cuando ocurren eventos siguen el comportamiento mostrado en la Figura 4.1a.

Para lograr que el SRV esté constantemente activo, hasta que el usuario decida detenerlo, se implementó un ciclo continuo, como se muestra en la Figura 4.1b. Durante este ciclo, el SRV permanece en funcionamiento, escuchando constantemente las entradas del usuario. Si el usuario indica que desea salir del SRV, entonces el ciclo finaliza y el servicio se detiene. En caso contrario, el ciclo se reinicia, permitiendo que el servicio continúe activo para seguir escuchando las entradas del usuario.

Finalmente, se verifica si la aplicación tiene el permiso `RECORD_AUDIO`. En caso positivo, se inicia el reconocimiento de voz utilizando `startListening(SpeechRecognizerIntent)`. Si no tiene el permiso, se solicita al usuario que lo conceda utilizando `ActivityCompat.requestPermissions()`. Si el usuario concede el permiso, se inicia el reconocimiento de voz utilizando `startListening(SpeechRecognizerIntent)`.

Resultados de primera implementación

A continuación, se presentan los resultados obtenidos durante la fase de pruebas de la primera implementación.

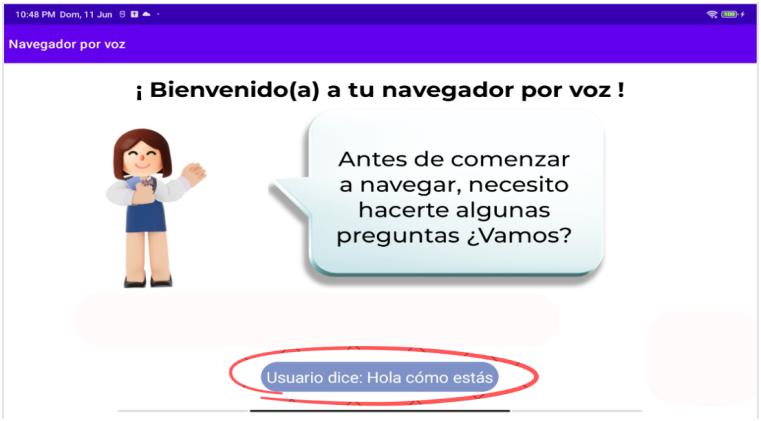


Figura 4.2: Primera implementación de reconocimiento de voz

- El prototipo realizó el reconocimiento de voz de forma constante, rompiendo el ciclo hasta que el usuario indicó finalizar con el comando “cerrar”.

- En la Figura 4.2 se muestra, resaltado en un óvalo rojo, un rectángulo azul acompañado de la leyenda 'Usuario dice:' con el cual se presenta en pantalla el texto reconocido. Esta representación visual se logró mediante la implementación de un *Toast*, que es un breve mensaje o notificación que aparece en la parte inferior de la pantalla durante un breve período de 3.5 segundos.

- Al iniciar cada ciclo de reconocimiento de voz (ver Fig. 4.1b) se activa una señal auditiva que se reproduce para notificar al usuario que el micrófono está activo y escuchando.

- Se intentó eliminar este sonido de notificación, pero no puede ser desactivado directamente por las aplicaciones, ya que es una función del sistema operativo. Dicha notificación forma parte de las políticas de seguridad de Android, cuyo objetivo es proporcionar transparencia y asegurarse de que los usuarios estén al tanto de las aplicaciones que acceden al micrófono del dispositivo. Sin embargo, este sonido puede ser intrusivo en ciertas situaciones, lo que puede afectar negativamente la experiencia del usuario, especialmente cuando se está reproduciendo un video o música.

4.1.1.2 Segunda implementación

El objetivo de esta implementación fue desarrollar el módulo de reconocimiento de voz bajo un Patrón de diseño Observador (PO). El PO es un patrón de comportamiento que se utiliza comúnmente para lograr una comunicación eficiente entre objetos [55]. El objetivo principal del PO es establecer una relación de uno a muchos entre objetos, de manera que cuando un objeto cambia su estado, todos los objetos dependientes u observadores sean notificados y actualizados automáticamente.

Como se muestra en la Figura 4.3, se utilizó el PO aprovechando cada uno de sus elementos y de acuerdo con las características específicas de esta propuesta, los elementos son los siguientes:

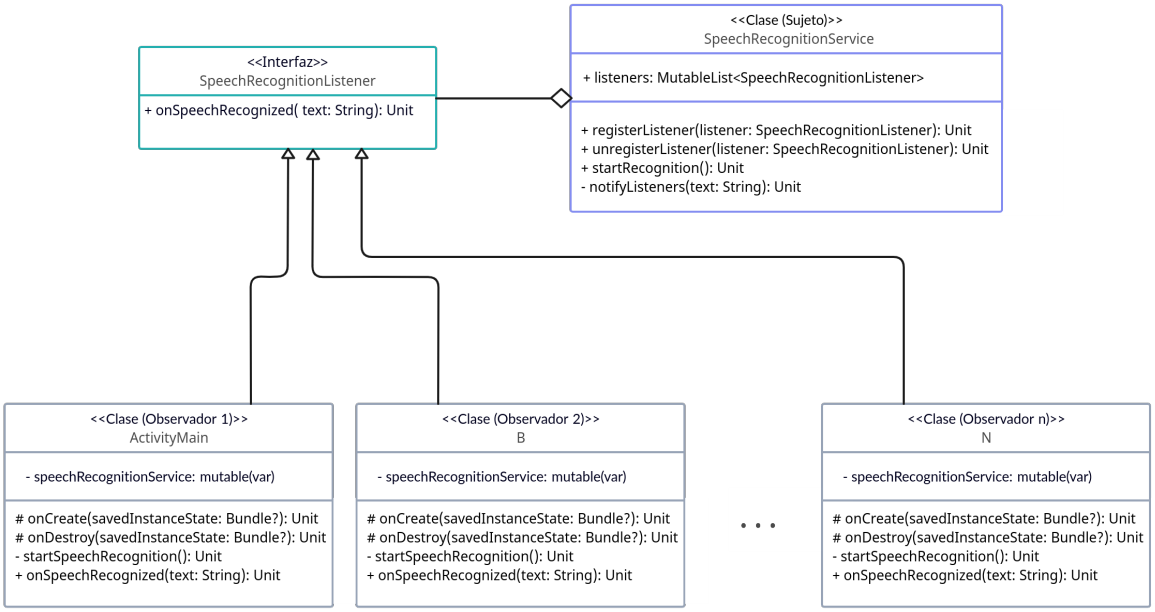


Figura 4.3: Diagrama de clases de PO de la segunda implementación.

- **Clase (Sujeto) - `SpeechRecognitionService`:** esta clase actúa como el '*sujeto*' en el PO, manteniendo una lista de observadores interesados en el reconocimiento de voz. Proporciona métodos para iniciar el reconocimiento de voz, agregar (`registerListener`), eliminar (`unregisterListener`) y notificar a los observadores (`notifyListener`) cuando se produce un cambio en su estado. En este contexto, el cambio de estado ocurre cuando se reconoce texto a partir del reconocimiento de voz.

- **Interfaz (Observador) - `SpeechRecognitionListener`:** interfaz que los observadores deben implementar. Contiene el método de actualización (`onSpeechRecognized`) que es llamado por el sujeto cuando se produce un cambio.

- **Clase (Observador i)**: las clases de ejemplo `ActivityMain`, `B` y `N` que se muestran en la Figura 4.3, son clases observadoras que implementan la interfaz `SpeechRecognitionListener`. Cada clase observadora se registra para recibir notificaciones y realiza acciones específicas cuando se actualiza el sujeto.

En esta versión también se solucionó el problema mencionado en la sección 4.1.1.1 referente al sonido de notificación de acceso al micrófono.

Código 4.1: Clase VolumeControl

```
import android.content.Context
import android.media.AudioManager
class VolumeControl(private val context: Context) {
    fun muteNotifications() {
        val audioManager = context.getSystemService(Context.AUDIO_SERVICE)
           as AudioManager
        audioManager.setStreamVolume(
            AudioManager.STREAM_NOTIFICATION,
            AudioManager.ADJUST_MUTE, AudioManager.FLAG_SHOW_UI)
    }
    fun unmuteNotificationsWithVolume(volume: Int) {
        val audioManager = context.getSystemService(Context.AUDIO_SERVICE)
           as AudioManager
        audioManager.setStreamVolume(AudioManager.STREAM_NOTIFICATION,
            volume, AudioManager.FLAG_SHOW_UI)
    }
}
```

La solución consistió en reducir el volumen de las notificaciones al inicio de cada ciclo del servicio de reconocimiento de voz (SRV) y, una vez que el SRV se ha iniciado, se restablece el volumen de las notificaciones a su estado original. Se creó la clase `VolumeControl` mostrada en el bloque de código 4.1 para realizar esta tarea, donde el método `muteNotifications` reduce a cero el volumen de las notificaciones y el método `unmuteNotificationsWithVolume`, restablece el volumen original. Se agregaron las líneas que invocan a estos métodos en código al inicio de cada ciclo del SRV como se muestra en el bloque de código 4.2.

Código 4.2: Solución al problema del sonido de notificación de uso de micrófono

```
volumeControl.muteNotifications()
startRecognition() //Comienza el reconocimiento de voz
volumeControl.unmuteNotificationsWithVolume(originalVolume)
```

Resultados de la segunda implementación

Inicialmente, la segunda implementación mostró un funcionamiento satisfactorio al ser probada de forma aislada. Sin embargo, al integrarla con la interfaz gráfica y otras funcionalidades como reproducción de videos y música, se identificó un problema que requirió una revisión más detallada. El problema surgió debido a la interacción entre el ciclo del SRV y la presentación de contenido al usuario, donde en cada inicio del ciclo del SRV, el proceso se pausaba en la actividad o interfaz gráfica mostrada al usuario. Estos efectos se volvieron más evidentes cuando el usuario solicitó la reproducción de videos y música. El ciclo del SRV se iniciaba cada 10 segundos, lo que provocaba una pausa recurrente en la reproducción del recurso multimedia, afectando directamente la experiencia del usuario.

4.1.1.3 Tercera implementación

El objetivo de esta implementación fue resolver el problema encontrado en la sección anterior. Se analizaron las posibles razones por las cuales se presentó el problema y se enuncian a continuación.

Problemas de audio concurrente

Pueden surgir dificultades técnicas cuando un dispositivo intenta realizar ambas tareas al mismo tiempo, como grabar y reproducir sonidos. Siguiendo esta idea, se realizaron pruebas donde se pidió al prototipo reproducir un video, pero sin volumen, los resultados obtenidos mostraron que la reproducción no se pausó mientras que el SRV se ejecutaba y mostraba resultados a la par. Para comprobar que el problema era causado por el audio concurrente, se decidió separar la captura y reproducción de audio mediante el

uso de audífonos sin micrófono. De esta manera, se utilizó el micrófono de la tableta sin interferencias del audio reproducido. Con esta configuración, se solicitó al prototipo reproducir un video con volumen; sin embargo, se presentó el mismo fenómeno de pausa en el video, incluso cuando el micrófono no capturaba ningún sonido. Por otro lado, cuando se pidió reproducir el video sin volumen, no hubo pausas en el video.

Bloqueo del hilo principal

En Android, el hilo principal es el responsable de manejar la interfaz de usuario y ejecutar todas las operaciones principales de la aplicación, como la interacción del usuario, la actualización de la interfaz gráfica y la respuesta a eventos. El bloqueo del hilo principal ocurre cuando una tarea tarda mucho tiempo en completarse o está realizando operaciones pesadas, lo que provoca que otras tareas que deberían ejecutarse en el hilo principal se detengan o se realicen con retraso [54].

Como se mencionó anteriormente, se utilizó la clase `Speech Recognizer`, cuyos métodos solo pueden ser invocados en el hilo principal. Según la documentación de Android, no es recomendable hacer reconocimiento de voz de forma constante debido a que esta clase no fue diseñada para el reconocimiento continuo, además de que esta funcionalidad consumiría una cantidad significativa de batería y ancho de banda.

Tomando en cuenta los riesgos mencionados, se realizaron las implementaciones anteriores y con base en los resultados obtenidos, se sospechó que el problema encontrado podría estar relacionado con el bloqueo del hilo principal, es decir, el reconocimiento de voz continuo no se maneja adecuadamente en el hilo principal. Para solucionar este problema, se propuso lo siguiente:

1. Ejecutar parte del servicio de reconocimiento de voz en un hilo secundario como se muestra en la Figura 4.4.

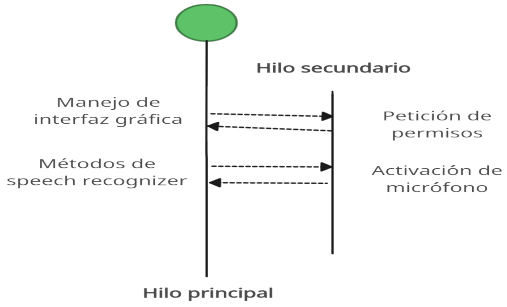


Figura 4.4: Primer propuesta de solución al bloqueo del hilo principal.

El propósito de esta solución fue determinar qué tarea era responsable del fenómeno que pausaba los procesos en la actividad principal: el reconocimiento de voz, la activación del micrófono o ambas. Durante el desarrollo de esta propuesta, se identificó que no es posible solicitar permisos de acceso ni activar el micrófono directamente desde un hilo secundario en Android. Esta limitación se basa en lo siguiente:

- **Políticas de seguridad y privacidad:** el sistema Android, exige transparencia al usuario en cuanto al uso de características del dispositivo, como el micrófono. Para cumplir con estas políticas, los desarrolladores deben solicitar y obtener explícitamente el permiso de acceso al hardware, e informar claramente al usuario sobre el propósito del acceso. Además, se requiere notificar al usuario mediante notificaciones cuando el micrófono esté activo, incluso después de que se haya otorgado el permiso [54]. El usuario otorga o deniega permisos a través de la interfaz de usuario, por lo tanto, la solicitud de permisos debe realizarse en el hilo principal, ya que dicho hilo es el responsable de manejar la interfaz de usuario y mostrar elementos en pantalla.
- **Acceso al hardware y recursos compartidos:** en Android, se establecen políticas y restricciones para el acceso al hardware y recursos compartidos desde hilos secundarios. El objetivo es evitar problemas de sincronización y bloqueos que podrían surgir si múltiples hilos acceden al hardware simultáneamente o

si las operaciones de la interfaz de usuario se realizan en hilos secundarios separados.

Debido a las limitaciones encontradas, no fue posible desarrollar la propuesta de solución ni determinar la causa del fenómeno de pausa en la interfaz. Sin embargo, se pudo concluir que la activación del micrófono siempre se realiza desde el hilo principal. La investigación también reveló que algunas API externas permiten que sus funciones o llamadas se realicen en hilos secundarios. De lo anterior, surgió la idea de implementar el reconocimiento de voz utilizando una de estas API externas, que no requiera ser invocada desde el hilo principal, y verificar si la activación del micrófono es responsable del fenómeno de pausa en la interfaz de usuario.

2. Ejecutar el reconocimiento de voz en un hilo secundario, mientras que la activación y la solicitud de permisos se llevan a cabo en el hilo principal.

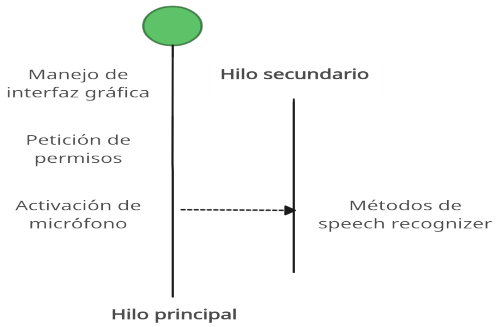


Figura 4.5: Segunda propuesta de solución al bloqueo del hilo principal.

El propósito de la solución ilustrada en la Figura 4.5, fue determinar qué tarea era responsable de pausar o bloquear el hilo principal utilizando el método de eliminación. Se planteó que si al ejecutar los métodos de reconocimiento de voz en un hilo secundario, se resolvía el fenómeno de bloqueo, entonces esa sería la solución al problema. Por otro lado, si a pesar de quitar los métodos de reconocimiento de voz del hilo principal, el bloqueo persistía, entonces el problema estaría relacionado

con la activación del micrófono. Se partió de las clases y APIs descritas en la Tabla 3.25. A continuación, se presenta el desarrollo y resultados obtenidos.

Recognizer Intent

Se comenzó consultando la documentación oficial de Android acerca de `Reconigzer Intent`, con el objetivo de determinar si presenta restricciones similares a `Speech Recognizer` en cuanto a la ejecución de sus métodos en el hilo principal. Al analizar la clase `Recognizer Intent`, se encontró que ésta se encarga de crear y enviar una intención al Sistema Operativo (SO). Posteriormente, el SO maneja una instancia de `Speech Recognizer` con la cual inicia el servicio de reconocimiento de voz, procesa el audio capturado por el micrófono y obtiene los resultados del reconocimiento. Aunque la documentación no menciona directamente restricciones en la ejecución de los métodos de `Recognizer Intent`, el análisis reveló que esta clase trabaja como una interfaz de alto nivel, que simplifica el uso de los métodos de `Speech Recognizer`. Por lo tanto, se concluyó que un servicio de reconocimiento de voz implementado mediante `Recognizer Intent` no puede ser ejecutado en un hilo secundario.

Speech to text Google

En la documentación proporcionada por Google sobre *Cloud Speech-to-Text V1* [56], se abordan diversos aspectos de la implementación de esta herramienta, como los tipos de solicitudes que se pueden realizar, la construcción de solicitudes y el manejo de respuestas. No se encontró información que indique que las funcionalidades de esta API deban ser ejecutadas en el hilo principal dentro de un entorno específico. Con base en esta información, se procedió a implementar la propuesta ilustrada en la Figura 4.5 siguiendo el patrón de diseño observador propuesto en la Figura 4.3.

El diagrama de clases que se muestra en la Figura 4.6 presenta la estructura de

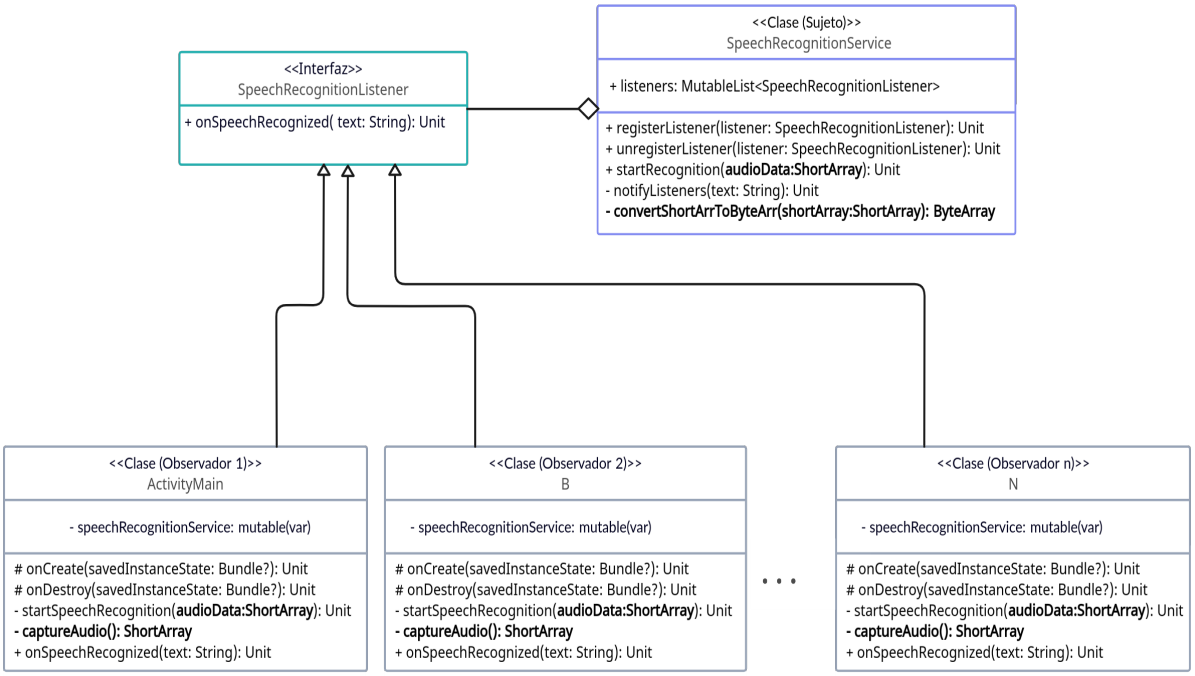


Figura 4.6: Modificaciones al PO.

la solución. Las modificaciones en relación con la Figura 4.3 se indican mediante letras en negritas. En dicho diagrama, se puede observar que en la clase `SpeechRecognitionService` y en las clases observadoras, se mantienen los métodos relacionados con el patrón de diseño observador. No obstante, es importante destacar los siguientes cambios significativos:

a) Cambios en la clase `SpeechRecognitionService`

- El método `startRecognition` (ver Anexo A.1), se ajustó para recibir datos de audio en formato *ShortArray*, debido a que el audio se captura desde el hilo principal en formato LINEAR16[1]. Esta función, al ser invocada, obtiene las credenciales necesarias para autenticar la API de reconocimiento de voz de Google Cloud, desde un archivo JSON. Utilizando

[1]Formato de codificación de audio utilizado por Android, también conocido como "PCM 16-bit".

estas credenciales, se configura un cliente `SpeechClient`. Luego, se establecen los detalles de la solicitud, como el formato del audio, el idioma y la tasa de muestreo. La función realiza la solicitud de reconocimiento de voz mediante `SpeechClient`, procesa los resultados y notifica el texto reconocido a los observadores.

- Los datos de audio se capturan en formato PCM (Modulación por Codificación de Pulso) y se almacenan en un arreglo de tipo *short*. Se introdujo el método `convertShortArrToByteArr` con el objetivo de convertir los datos de audio en un arreglo de tipo *Byte*. Esta conversión es esencial, ya que el servicio de reconocimiento de voz de Google Cloud requiere específicamente el uso de éste último formato.

b) Cambios en las clases observadoras

- Se agregó el método, llamado `captureAudio()`, con el objetivo de capturar datos de audio a través del micrófono del dispositivo. En esta función se obtiene el tamaño del búfer necesario para la grabación de audio. Se configura la grabación de audio, instanciando un objeto de tipo `AudioRecord`, para gestionar la captura de audio y se especifican los siguientes parámetros:
  - `MediaRecorder.AudioSource.MIC`: indica que el micrófono del dispositivo será la fuente de audio.
  - `SAMPLE_RATE`: la tasa de muestreo especificada para la grabación.
  - `AudioFormat.CHANNEL_IN_MONO`: indica que se grabará en formato mono (un solo canal).
  - `AudioFormat.ENCODING_PCM_16BIT`: el formato de codificación especificado (PCM 16-bit).
  - `bufferSize`: el tamaño del búfer.

  Luego, se crea el arreglo de tipo *short* de nombre `audioData`, con el

tamaño del búfer. En este arreglo se almacenan las mediciones de amplitud de las muestras de audio capturadas. La grabación se inicia y se capturan los datos. Posteriormente, se detiene la grabación y se liberan los recursos. Finalmente, la función devuelve el arreglo `audioData`.

- El método `startSpeechRecognition`, se ajustó para recibir como parámetro de entrada el arreglo `audioData`. En este método se coordina la inicialización y ejecución del proceso de reconocimiento de voz utilizando el servicio de reconocimiento de voz de Google Cloud a través de una instancia de la clase `SpeechRecognitionService`.

Código 4.3: Ciclo while para servicio de reconocimiento de voz

```
while (true) {
    if (permissionToRecordAccepted) {
        val audioData = captureAudio()
        // Iniciar el reconocimiento en un hilo secundario
        val recognitionThread = Thread {
            startRecognitionService(audioData)
            }
        recognitionThread.start()
        // Esperar a que el hilo de reconocimiento termine
        recognitionThread.join()
        // Esperar antes de continuar con la siguiente
           iteracion del ciclo
        Thread.sleep(1000)
     }
}
```

- En el método `onCreate` de cada clase observador, se implementó el ciclo que se muestra en el bloque de código 4.3. En este ciclo se captura continuamente el audio y se envía para su reconocimiento en un hilo secundario, con un intervalo de pausa entre iteraciones.

Los resultados obtenidos de las pruebas indicaron que es viable llevar a cabo el reconocimiento de voz en un hilo secundario utilizando *Speech-to-Text* de Google. Sin embargo, persistió el problema relacionado con el bloqueo del hilo principal que afectaba la reproducción de contenido multimedia. Se determinó que este problema se origina por la activación repetitiva del micrófono en cada iteración del ciclo. Por otro lado, se observó que al finalizar o pausar una actividad que actuaba como observador, se interrumpía el ciclo de captura y reconocimiento de voz. Este ciclo se reiniciaba al presentarse en pantalla una nueva actividad. Esta dinámica no cumple con el requerimiento funcional 3.1, lo que condujo a la búsqueda de una alternativa que permitiera abordar esta situación de manera más efectiva.

4.1.1.4 Cuarta implementación

De la tercera implementación, se pudo concluir que el problema con el servicio de reconocimiento de voz está directamente relacionado con el bloqueo del hilo principal. El bloqueo ocurría cuando el micrófono se activaba en cada iteración del ciclo, y se descartó que tuviera que ver con el procesamiento de la voz. También se identificó un problema asociado al ciclo de procesamiento de voz. Cuando se cierra una actividad que inicia este ciclo, el proceso se pausa y posteriormente se reinicia una vez que se vuelve a abrir la misma actividad u otra.

La cuarta implementación se basó en la idea de activar el micrófono una única vez al inicio, durante la creación de la actividad principal de la aplicación. De esta manera, el micrófono queda activado en una transmisión continua de audio. Tanto la transmisión como el procesamiento de voz se llevan a cabo en segundo plano[1], sin embargo, la activación del micrófono y las notificaciones permanecen en primer plano. Esta decisión se tomó con el propósito de evitar que estas tareas se pausen en caso de que se cierre alguna actividad.

En el entorno de Android, existe un componente llamado *service* el cual tiene la capa-

[1]En Android, "segundo plano" se refiere a los procesos que no son visibles para el usuario y pueden o no ejecutarse en el hilo principal.

cidad de llevar a cabo operaciones de larga duración en segundo plano, sin presentar una interfaz de usuario o notificaciones visibles. Además, cualquier otro componente dentro de la aplicación puede iniciar un servicio, permitiendo que éste funcione en segundo plano incluso si el usuario cambia a otra actividad. Se implementó el servicio personalizado `AudioCaptureService` (ver Anexo A.2), que se encarga de capturar audio desde el micrófono del dispositivo, procesar el audio capturado y enviarlo al servicio de *Speech-to-text* de Google Cloud para realizar el reconocimiento de voz continuo.

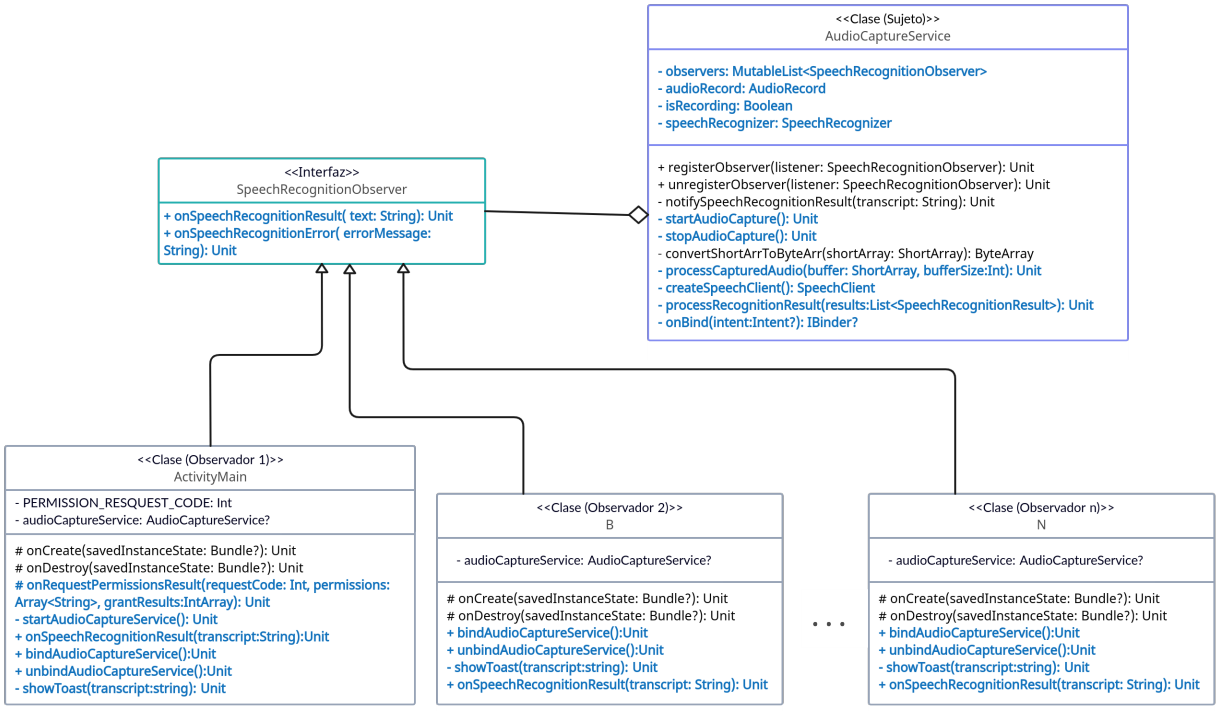
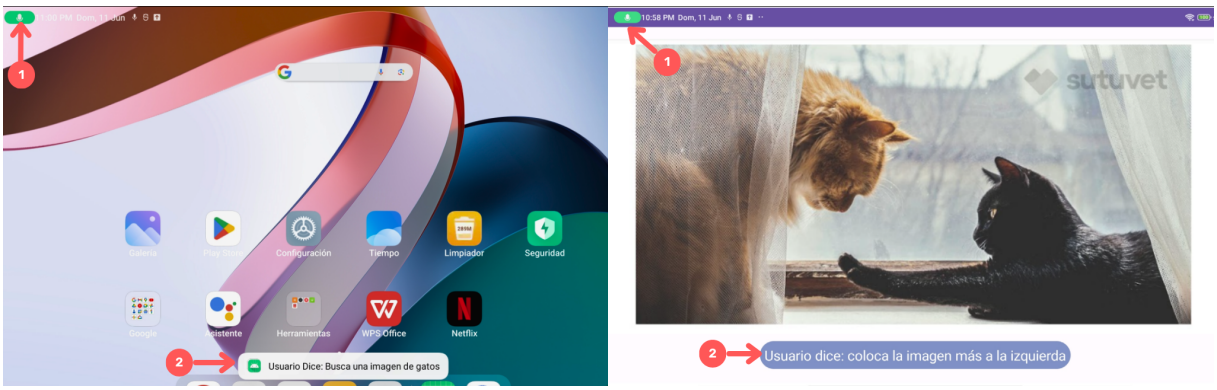


Figura 4.7: Diagrama de actividades de la cuarta implementación.

En la Figura 4.7, se presenta el diagrama de actividades correspondiente a esta implementación. Se han destacado las modificaciones mediante el uso de color azul y formato de texto en negrita. Se continuó con la adopción del patrón de diseño observador, realizando las adaptaciones requeridas en cada clase involucrada.



(a) Reconocimiento de voz activo en segundo plano.

(b) Reconocimiento de voz activo en primer plano.

Figura 4.8: Resultados de la cuarta implementación

El análisis de los resultados de esta implementación reveló que los recursos multimedia, tales como videos y música, ya no experimentaban interrupciones en su reproducción. Además, se constató que el servicio de reconocimiento de voz continuaba operando incluso en ausencia de actividades visibles en la pantalla. La Figura 4.8 ilustra dos escenarios de la implementación.

- En la Subfigura 4.8a, se puede observar que el SRV opera en segundo plano mientras el dispositivo muestra la pantalla de inicio. En otras palabras, no se presenta ninguna actividad visible de la aplicación en ese momento.

- En la Subfigura 4.8b, se puede observar que el SRV opera en primer plano mientras se muestra una imagen en pantalla a través de una interfaz.

En ambas figuras se presentan un ícono resaltado con una flecha y el número 1 en color rojo. Este ícono se coloca como notificación, en conformidad con las políticas de seguridad de Android. Su propósito es indicar visualmente al usuario que el micrófono de su dispositivo está activado. De manera análoga, se incluye un mensaje a través de un *Toast*, marcado con una flecha y el número 2 en color rojo. Este mensaje, que refleja el texto reconocido por el SRV a partir de la voz del usuario, se visualiza durante 3.5 segundos

en pantalla y luego desaparece. La interfaz gráfica no experimenta interrupciones en la entrega de contenido multimedia, y el proceso de reconocimiento de voz se lleva a cabo de manera fluida.

4.1.2 Interfaz gráfica

En la Sección 3.1.4, se definieron los requisitos de diseño y funcionalidad de las interfaces gráficas. Cada interfaz se diseñó con el objetivo de mostrar contenido específico y proporcionar retroalimentación visual de las solicitudes al usuario. En el contexto del desarrollo de aplicaciones Android, la interfaz gráfica de usuario se define en el archivo XML correspondiente a una actividad específica. En este archivo, se establecen los elementos visuales[1] de la interfaz. A continuación, se detallan los aspectos importantes de la implementación.

- **Diseño adaptable:** cada interfaz se diseñó teniendo en cuenta su apariencia en las dos orientaciones que puede adoptar el dispositivo. Esto asegura que el usuario tenga una experiencia consistente tanto en orientación vertical como horizontal. En el archivo XML de cada interfaz, se definió la estructura de diseño *ConstraintLayout* con la etiqueta que se muestra en el bloque de código 4.4.

  Mediante el empleo de esta etiqueta, es posible establecer relaciones y restricciones entre los elementos visuales. Esto garantiza que los componentes se posicionen y ajusten de manera automática ante cambios en la orientación de la pantalla del dispositivo, siguiendo las reglas definidas dentro de la etiqueta.

Código 4.4: ConstraintLayout

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="
   http://schemas.android.com/apk/res/android">
    (Elementos visuales y restricciones)
</androidx.constraintlayout.widget.ConstraintLayout>
```

Se ocuparon dos tipos de reglas principalmente: conexiones para alineación entre elementos y dimensiones/límites.

[1] Componentes gráficos que conforman la apariencia y la interacción de una interfaz de usuario en una aplicación. Por ejemplo: botones, campos de texto, iconos, contenedores, entre otros.

Conexiones para alineación: estas reglas se utilizaron para alinear elementos con los bordes o características del contenedor principal u otros elementos. Las etiquetas empleadas para definir las reglas son:

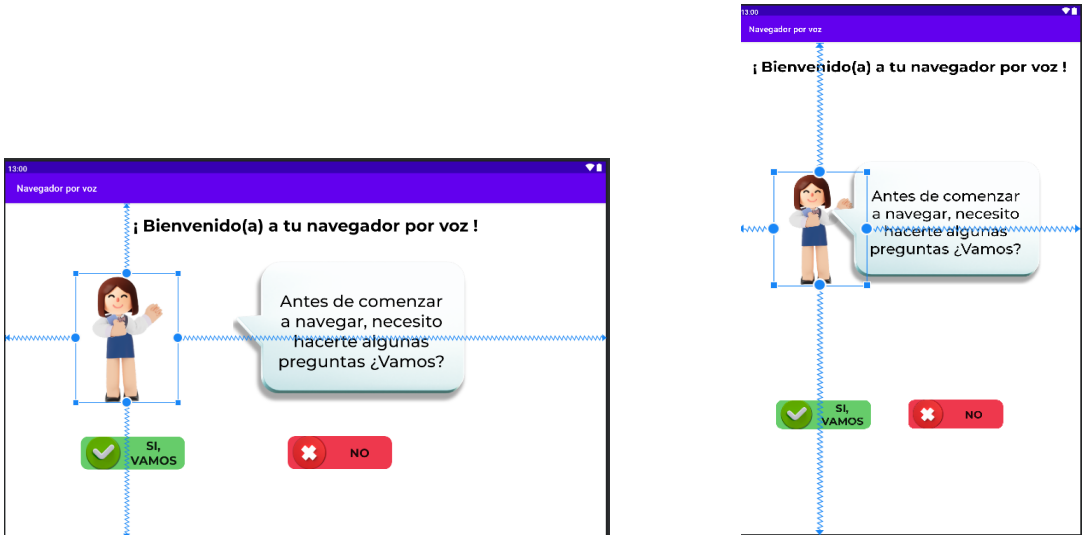
- *layout_constraintTop_toBottomOf*: alinea el borde superior de un elemento con el borde inferior de otro elemento.
- *layout_constraintBottom_toTopOf*: alinea el borde inferior de un elemento con el borde superior de otro elemento.
- *layout_constraintStart_toEndOf*: alinea el borde de inicio de un elemento con el borde de fin de otro elemento.
- *layout_constraintEnd_toStartOf*: alinea el borde de fin de un elemento con el borde de inicio de otro elemento.
- *layout_constraintHorizontal_bias*: permite ajustar la posición horizontal relativa entre dos elementos.

Por ejemplo, en la primera interfaz que se le muestra al usuario (ver Figura 4.2), se encuentra una imagen, que se colocó mediante el uso del elemento *ImageView*. Para este elemento se establecieron las reglas que se muestran en los renglones 6-11 del bloque de código 4.5.

Código 4.5: Reglas para un elemento ImageView

```
1 <ImageView
2        android:id="@+id/imageView"
3        android:layout_width="218dp"
4        android:layout_height="275dp"
5        android:foregroundGravity="center"
6        app:layout_constraintBottom_toBottomOf="parent"
7        app:layout_constraintEnd_toEndOf="parent"
8        app:layout_constraintHorizontal_bias="0.14"
9        app:layout_constraintStart_toStartOf="parent"
10       app:layout_constraintTop_toTopOf="parent"
11       app:layout_constraintVertical_bias="0.341"
12       app:srcCompat="@drawable/ic_saludo" />
```

Con estas especificaciones, la imagen resaltada con un recuadro azul, toma las posiciones que se muestran en la Figura 4.9, según la orientación de la pantalla.



(a) Horizontal. (b) Vertical.

Figura 4.9: Posiciones de ImageView

Dimensiones/límites: estas reglas se utilizaron para establecer o limitar el ancho y la altura de los elementos. En el bloque de código 4.5, se muestran en las líneas 3 y 4, las reglas usadas para establecer las dimensiones del *ImageView*.

- **Elementos visuales:** como se mencionó al inicio de esta sección, cada interfaz está encargada de desplegar en pantalla el contenido de forma gráfica, como se puede ver en la Figura 3.1. Por ello, se emplearon los siguientes elementos:

  - *TextView*: para mostrar texto en la pantalla, como títulos, descripciones, preguntas o instrucciones.
  - *ImageView*: para mostrar imágenes y gráficos en la pantalla.
  - *Button*: para que los usuarios pueden tocar una representación de un botón para activar un evento.

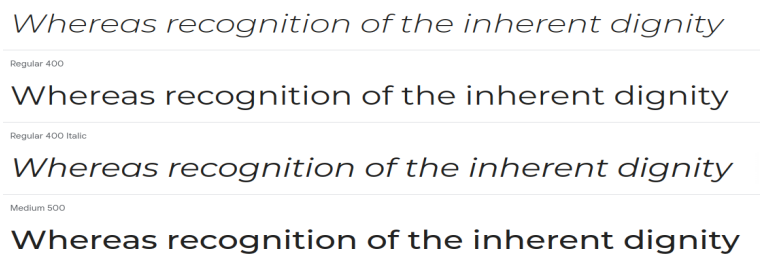


Figura 4.10: Fuente Monserrat de Google Fonts

- *WebView*: para mostrar videos alojados en alguna plataforma (por ejemplo, videos de Youtube) o el contenido de las páginas y sitios web.
- *VideoView*: para reproducir videos almacenados, en sitios web, en formatos .mp4, .avi, .mov, entre otros.

Las propiedades específicas de cada elemento, como su identificador, texto y fondo, entre otras, fueron ajustadas con el objetivo de controlar tanto su apariencia como su comportamiento inicial dentro de cada interfaz. Es importante destacar que el comportamiento posterior es gestionado por el módulo de operaciones.

- **Colores y Fuente:** estos aspectos fueron elegidos por la desarrolladora con la finalidad de establecer una apariencia legible y clara en este primer prototipo. En cuanto a los colores, se eligieron tonos contrastantes para garantizar una buena legibilidad y accesibilidad. Los colores elegidos son distinguibles en el fondo blanco (ver Figuras 4.8 y 4.9). Se eligió la tipografía Monsserrat[1], la cual presenta caracteres legibles que facilitan la lectura (Ver Figura 4.10).

4.2 Implementación del traductor de lenguaje natural

Como se mencionó anteriormente en la Sección 3.1, uno de los principales objetivos de esta propuesta es conocer y dar significado a la petición del usuario. En línea con esta

[1]Obtenida de: https://fonts.google.com/specimen/Montserrat

meta y el requerimiento 3.4, se desarrolló el Traductor de Lenguaje Natural (TLN). La importancia de este módulo radica en su capacidad para analizar la validez de las solicitudes de los usuarios y detectar la intención subyacente en cada una. Además, permite la extracción de datos relevantes y el establecimiento de parámetros basados en dichas solicitudes. Posteriormente, el módulo de operaciones utiliza esta información para generar e implementar instrucciones que entreguen contenido al usuario. La capacidad de extracción de datos abarca no solo las solicitudes del usuario, sino también las respuestas suministradas durante el proceso de creación del perfil de usuario. En esta Sección se explica a detalle cómo se desarrolló este módulo.

4.2.1 Reconocimiento de intención

Para llevar a cabo esta tarea, se empleó la herramienta *Dialogflow*. Como se explicó en la Sección 3.4.2, esta plataforma facilita la comprensión de las intenciones subyacentes en las palabras o frases introducidas por un usuario. A continuación, se presentan los conceptos básicos que se abordaron durante la implementación de esta funcionalidad utilizando *Dialogflow*. Estos conceptos se mencionan con el propósito de facilitar la comprensión del proceso de desarrollo:

- *Agente*: es la entidad central en *Dialogflow* (DF) encargada de interpretar las interacciones con los usuarios. Se entrena para responder a diversas situaciones de conversación esperadas. Un agente contiene todas las configuraciones, intenciones, entidades y respuestas necesarias.

- *Intención*: es un elemento que permite representar una acción o solicitud del usuario. Un agente contiene varias intenciones, que combinadas permiten manejar una conversación. Una intención básica contiene los siguientes elementos:

  - *Frases de entrenamiento*: estas son frases de ejemplo utilizadas para entrenar al agente en cómo podría expresarse un usuario con una intención específica. La

plataforma de DF indica que no es necesario definir todos los ejemplos posibles, ya que su aprendizaje automático amplía la lista con frases similares.

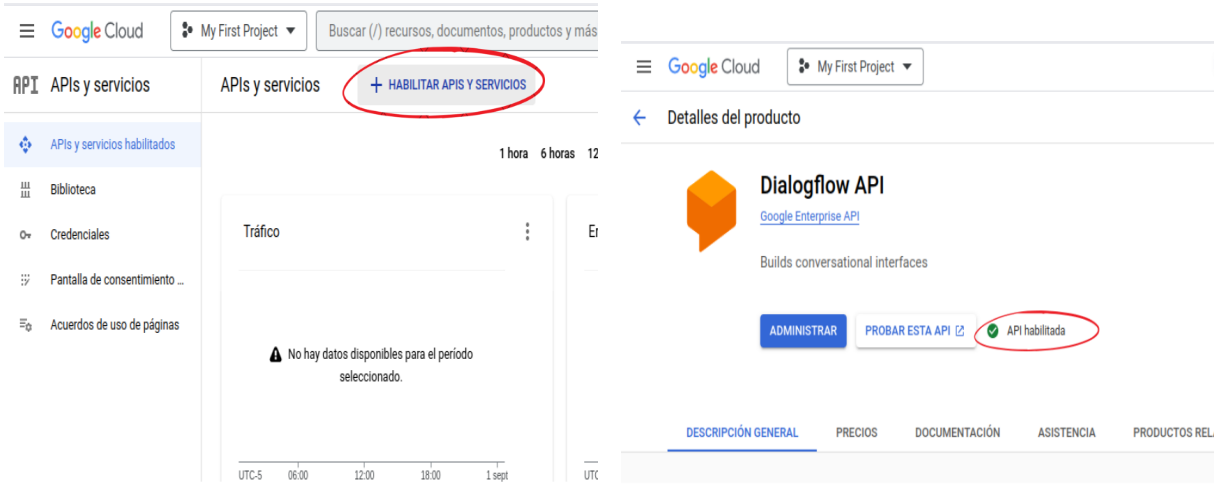
- *Acción*: es una bandera que se define para indicar qué acción debe realizarse cuando se clasifique una frase en determinada intención.
- *Parámetros*: son los valores que DF extrae de la expresión del usuario. Cada parámetro tiene un tipo de entidad que especifica qué datos deben ser extraídos. Estos datos pueden ser utilizados para realizar alguna lógica o generar respuestas relacionadas con la expresión del usuario.
- *Respuestas*: de acuerdo a cada intención se pueden definir respuestas en texto, de voz o visuales.

- *Entidad*: se refiere a conjuntos de palabras que pueden ser clasificadas en una categoría y que representan valores relevantes o comunes en el contexto de la conversación. Cada parámetro de una intención tiene un tipo de entidad asociado. DF proporciona algunas entidades predefinidas, como colores, horas y apellidos, por mencionar algunas. Sin embargo, también es posible crear entidades personalizadas según las necesidades específicas del proyecto.

- *Contexto*: es una bandera que ayuda al agente a comprender las expresiones del usuario con base en el flujo de la conversación. El contexto influye en la interpretación de las solicitudes y en la generación de respuestas.

En primer lugar, se inició sesión en la consola de Google Cloud[1], la cual es una interfaz de usuario web que permite administrar y configurar, entre otras cosas, los productos y APIs de Google Cloud. Se creó y configuró el proyecto asignándole un nombre, un identificador, la ubicación geográfica donde se guardan los recursos y se aceptaron los términos y condiciones de Google Cloud. Tras completar la creación del proyecto, se desplegó la página de inicio correspondiente. Como se puede apreciar en la Figura 4.11a,

[1]Disponible en https://console.cloud.google.com/



(a) Opción para habilitar API. (b) DialogFlow habilitado.

Figura 4.11: Habilitación de DialogFlow.

se resalta en color rojo la opción de habilitar las APIs o servicios disponibles. En este caso, se procedió a buscar y habilitar Dialogflow, tal como se muestra en la Figura 4.11b. Posteriormente, se accedió al sitio web de DF Essentials y se seleccionó la opción *crear agente*. Con esta opción DF configuró el agente y desplegó en pantalla el entorno de desarrollo (ver Figura 4.12).

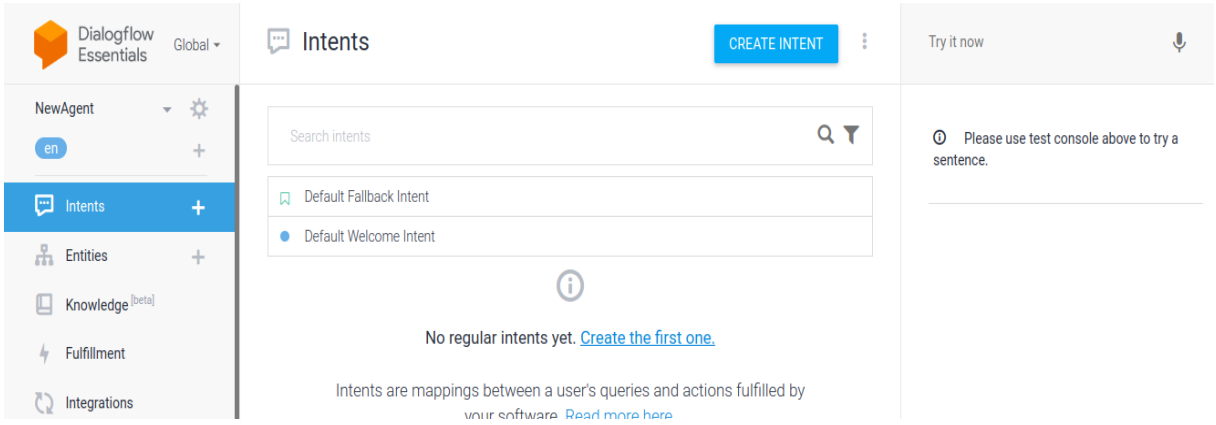


Figura 4.12: Entorno de desarrollo de DF.

Luego, se procedió a crear las intenciones y entidades necesarias.

4.2.1.1 Primera configuración

Se agruparon acciones similares bajo una sola intención. Por ejemplo, las acciones *reducir imagen* y *ampliar imagen* se incluyeron en la intención llamada *Tamaño imagen*. A continuación se enlistan las intenciones definidas y las acciones que abarcan:

- *Buscar Imagen*: utilizada para detectar cuándo el usuario solicita encontrar una imagen específica en la Web.

- *Tamaño imagen*: utilizada para detectar cuándo el usuario solicita modificar el tamaño de una imagen. Esta intención engloba reducir y aumentar el tamaño de la imagen.

- *Mover imagen*: utilizada para detectar cuándo el usuario solicita mover la imagen en el plano y rotaciones de 90 grados.

- *Buscar AudioVideo*: utilizada para detectar cuándo el usuario solicita encontrar videos o música en la Web.

- *Reproductor AudioVideo*: utilizada para detectar cuándo el usuario solicita pausar, reproducir, adelantar, retroceder y repetir el video o música.

- *Página Web*: utilizada para detectar cuándo el usuario solicita ir a una página web en específico.

- *Acciones PágWeb*: utilizada para detectar cuándo el usuario solicita dar clic, seleccionar, llenar un formulario y mover, acceder o buscar algún elemento específico dentro de una página web.

- *Término*: utilizada para detectar cuándo el usuario solicita buscar en la web un término específico.

- *General*: utilizada para detectar cuándo el usuario solicita realizar alguna acción como repetir respuesta, parar la reproducción de la respuesta y seleccionar una opción.
- *Volumen*: utilizada para detectar cuándo el usuario solicita subir o bajar el volumen de la reproducción de multimedia.
- *Cerrar*: utilizada para detectar cuándo el usuario solicita cerrar pantalla o la aplicación.

Junto con la definición de las intenciones, se detallaron las entidades con el propósito de identificar parámetros de tipo entidad en las frases del usuario. Algunas de las entidades que se implementaron en el agente son las siguientes:

- *Cantidad abstracta*: palabras como poco, mucho, algo.
- *Tamaño*: palabras como pequeño, chico, grande, entre otras.
- *Acciones VideoAudio*: palabras como pausar, reproducir, adelantar, retroceder, entre otras, que pueden ser usadas cuando el usuario interactúa con un reproductor de video o música.
- *Dirección*: palabras como arriba, abajo, izquierda, derecha, entre otras.

Las entidades y los parámetros están relacionados, debido a que las entidades se utilizan para definir tipos de datos que pueden extraerse de las frases del usuario, mientras que los parámetros se utilizan para capturar esos valores de las frases y utilizarlos en la lógica de la conversación o en las acciones del agente. En esta propuesta, los parámetros se emplean para determinar la acción que debe llevar a cabo la aplicación. Por ejemplo, si el usuario expresa la frase *" Haz la imagen un poco más pequeña"*, DF detecta la intención *tamaño imagen* y almacena los parámetros *"un poco"* y *"pequeña"* categorizándolos como tipos de entidad *"cantidad abstracta"* y *tamaño* respectivamente. Luego, utilizando estos

parámetros, la aplicación determina que la acción que debe realizarse es reducir el tamaño de la imagen.

También, fueron establecidos los contextos entre intenciones para determinar la intención del usuario en función de su interacción anterior. Por ejemplo, si el usuario solicita "*muestra una imagen de un paisaje*" y luego sigue con "*aumenta el tamaño*", el contexto permite al agente comprender que la segunda solicitud se refiere al tamaño de la imagen.

Esta estructura en el agente, se diseñó con el objetivo de simplificar la fase de entrenamiento, especialmente debido a las notables similitudes entre las frases de entrenamiento y las posibles respuestas. Por ejemplo, las solicitudes para cambiar el tamaño de una imagen a menudo seguían patrones similares como las frases "*reducir la imagen*" o "*ampliar la imagen*". Siguiendo con esta idea, las respuestas se definieron para adaptarse a solicitudes con la misma intención, pero con diferentes comportamientos en el prototipo. Este enfoque se basó en el análisis de frases comunes y en la consideración de ciertas palabras como tipos de entidad en cada intención. De esta manera, cuando el usuario emitía comandos relacionados, estas palabras se detectaban como parámetros, lo que permitía la generación de respuestas automáticas adecuadas para cada situación.

Los resultados de aplicar esta configuración en el agente revelaron la generación de errores gramaticales en las respuestas. Estos errores surgieron debido a la estructura genérica de las respuestas, la cual no siempre se adaptaba de manera adecuada a las variaciones lingüísticas que el agente reconocía en las solicitudes de los usuarios. Esto se debió a que DF tiene la capacidad de reconocer como parámetros a los sinónimos y variaciones de las palabras que se definieron en las entidades. Para ejemplificar este comportamiento, se toma el ejemplo del cambio del tamaño de una imagen, donde una de las respuestas adaptables fue estructurada de la siguiente forma:

"Entendido, voy a [*$acciónTamanoImagen*] la imagen"

Donde *[$acciónTamanoImagen]* es un parámetro del tipo de entidad "*accionTamanoImagen*". La entidad "*accionTamanoImagen*" contiene palabras como "*reducir*" y "*am-*

pliar", por lo tanto, cuando dichas palabras se identificaron en la solicitud del usuario, fueron configuradas las respuestas:

"Entendido, voy a *ampliar* la imagen" o "Entendido, voy a *reducir* la imagen"

Pero en el caso donde se usó la variación lingüística "*reduce la imagen*", se generó la respuesta:

"Entendido, voy a *reduce* la imagen"

Esta respuesta presenta un error gramatical, específicamente una conjugación verbal incorrecta. Se buscaron distintas alternativas de respuestas para resolver los problemas gramaticales, sin embargo, no fue posible encontrar estructuras que abarcaran todos los sinónimos o variaciones en las frases.

También, se observó que agrupar acciones dentro de una misma intención conlleva un procesamiento adicional de los parámetros detectados por el agente en las solicitudes de los usuarios. Este procesamiento es necesario para determinar el comportamiento específico que la aplicación debe exhibir cuando se identifica dicha intención junto con ciertos parámetros. Esto se debe a que, incluso cuando las acciones están relacionadas, desencadenan diferentes comportamientos en el prototipo.

4.2.1.2 Segunda configuración

Para abordar los problemas identificados en la primera configuración, se optó por definir cada acción dentro de una única intención. De esta manera, cada acción ahora cuenta con respuestas específicas asociadas. Por ejemplo, en la intención relacionada con la reducción del tamaño de una imagen, se han definido respuestas como:

"*Reduciendo imagen*" o "*Entendido, reduciendo el tamaño de la imagen*"

Asimismo, cuando se detecta la intención correspondiente, el agente devuelve los parámetros junto con una bandera que indica directamente la acción que la aplicación debe

ejecutar. Esto elimina la necesidad de realizar un procesamiento adicional de los parámetros detectados por el agente.

En la Tabla 4.1, se enlistan las intenciones que se crearon en esta configuración y se han categorizado por contexto en el que el usuario podría emplear frases que generen dichas intenciones. Con esta configuración se pudieron detectar las intenciones de cada acción que puede realizar el usuario y además se generaron respuestas automáticas congruentes con los parámetros y las solicitudes del usuario.

Tabla 4.1: Intenciones creadas en la segunda configuración.

Audio y video	**Imágenes**	**Páginas web**
- Buscar - Pausar - Reproducir - Repetir - Adelantar - Retroceder	- Buscar - Aumentar - Reducir - Girar en ángulo de 90 grados - Mover en el plano	- Buscar - Dar clic - Seleccionar - Ir a ... - Mover en el plano
Términos o nóticias	**Otros**	
- Buscar	- Repetir respuesta - Cerrar aplicación - Seleccionar opción - Gestionar el volumen	

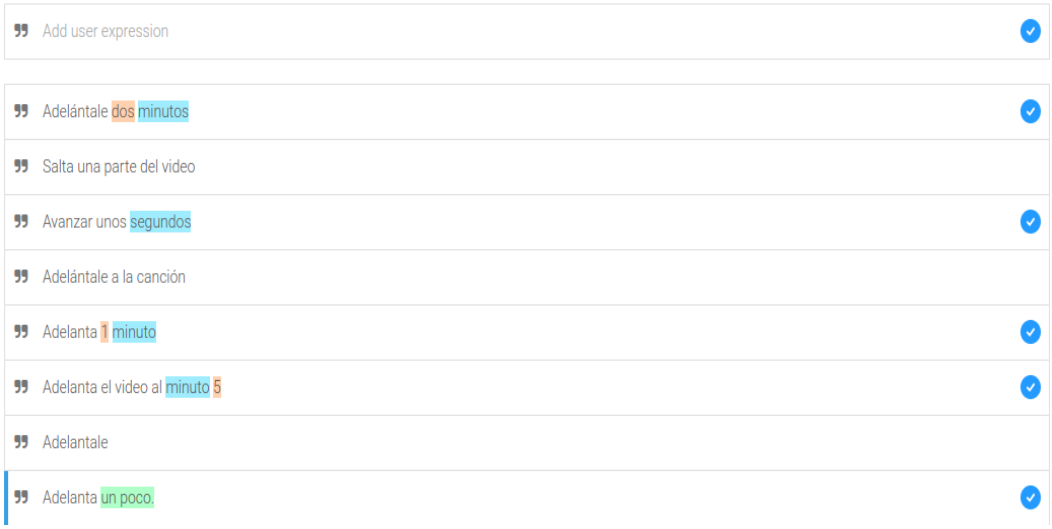
Otro de los cambios realizados en la configuración se centró en las frases utilizadas para entrenar la detección de intenciones. En estas frases, se definió específicamente el tipo de entidad al que pertenecían algunas palabras clave, en el contexto de la intención. Esta modificación fue necesaria porque, cuando no se especifican correctamente estos tipos de entidad, el agente de DF entrega valores que requieren un procesamiento adicional o no suministra información útil para realizar acciones en la aplicación. Por ejemplo,

cuando el usuario decía la frase "*Adelanta el video un minuto, por favor*" el sistema detectaba el parámetro del tipo *date-time*, originalmente definido para capturar datos de fechas o intervalos de tiempo, como horas. Sin embargo, bajo esta configuración, el agente respondía a la aplicación con un intervalo de tiempo capturado en tiempo de ejecución.

```
1 Resultado de reconocimiento de voz: adelanta el video un minuto por favor,
2 Responde DialogFlow: Adelantando video, Action: adelantarVideo,
3 Parameters: { cantidadAbstracta = string_value: "",
4 number = string_value: "",
5 date-time = struct_value {
6         fields{
7                 key: "endDateTime"
8                         value{
9                                 string_value: "2023-09-12T19:09:03-05:00"
10                        }
11                }
12        fields{
13                key: "startDateTime"
14                        value{
15                                string_value: "2023-09-12T19:08:03-05:00"
16                        }
17                }
18        }
19 } Contexto: []
```

Figura 4.13: Resultados del ejemplo "*Adelanta el video un minuto, por favor*" con la primera configuración.

En la Figura 4.13, se presenta la respuesta del agente ante la solicitud "*Adelanta el video un minuto, por favor*". En la línea uno, se muestra la cadena de texto detectada por la interfaz de voz. En la línea dos, se encuentra la respuesta predeterminada que proporciona el agente de DF y la acción que debe realizar la aplicación en función de la intención detectada, en este caso es "*adelantarVideo*". En las líneas 3-18, se encuentran los parámetros obtenidos y en los óvalos de color rojo se indican los parámetros que el agente de DF detecta cuando se menciona la palabra "*minuto*". Estos parámetros, denominados *startDateTime* y *endDateTime*, contienen valores en formato ISO 8601 que representan una fecha y hora, incluyendo la información de la zona horaria. A pesar de que representan un minuto, no son datos relevantes en el contexto de la intención. Debido a casos similares a éste, se realizaron ajustes en las entidades de cada intención, se crearon nuevas y se etiquetaron palabras en las frases de entrenamiento. Estos cambios permitieron obtener datos relevantes para cada una de las acciones.



(a) Frases de entrenamiento de la intención "*AdelantarVideo*".

```
Resultado de reconocimiento de voz: adelanta el video un minuto por favor,
Responde DialogFlow: Adelantando video, Action: adelantarVideo,
Parameters: { cantidadAbstracta = string_value: "",
cantidadTiempo = string_value: "minuto",
number = number_value: 1.0,
} Contexto: []
```

(b) Parámetros obtenidos con la segunda configuración.

Figura 4.14: Segunda configuración y resultados.

Continuando con el ejemplo anterior, en la Figura 4.14a se presentan las frases de entrenamiento utilizadas para la intención "*AdelantarVideo*". En estas frases, se pueden observar las modificaciones que se realizaron para mejorar la detección de los parámetros. Las palabras configuradas para identificarse con el tipo de entidad "*cantidadTiempo*", que incluye palabras como *hora(s)*, *minuto(s)* y *segundo(s)*, se resaltan en color azul. Aquellas palabras o caracteres identificados como el tipo de entidad "*número*" se destacan en color naranja, mientras que las palabras identificadas con el tipo de entidad "*cantidadAbstracta*" se resaltan en verde. Con esta configuración, se lograron obtener los parámetros, 1 y *minuto*, como se muestra resaltado en rojo en la Figura 4.14b.

4.2.1.3 Conexión del agente de DF con la aplicación

Después de haber abordado los aspectos esenciales de la configuración en DF, el siguiente

paso fue establecer la conexión entre DF y la aplicación. A continuación, se proporciona una descripción detallada de este proceso. Se creó la clase `LNTranslator`, que se utiliza para enviar consultas al agente y recibir respuestas. También, se encarga de la configuración de las credenciales, la creación de sesiones, el envío de consultas y la extracción de datos relevantes de las respuestas. Para el desarrollo de esta clase, se realiza la importación de las clases y bibliotecas necesarias para trabajar con DF y las credenciales de Google Cloud, como se muestra en el bloque de código 4.6.

Código 4.6: Importaciones en la clase LNTranslator

```
1 import android.content.Context
2 import com.google.api.gax.core.FixedCredentialsProvider
3 import com.google.auth.oauth2.GoogleCredentials
4 import com.google.cloud.dialogflow.v2.DetectIntentRequest
5 import com.google.cloud.dialogflow.v2.QueryInput
6 import com.google.cloud.dialogflow.v2.SessionName
7 import com.google.cloud.dialogflow.v2.SessionsClient
8 import com.google.cloud.dialogflow.v2.SessionsSettings
9 import com.google.cloud.dialogflow.v2.TextInput
```

A continuación, se explica la utilidad de cada una de estas importaciones.

- Se importa la clase `Context` para acceder al contexto de la aplicación.
- Se importa la clase `FixedCredentialsProvider` de Google Cloud para proporcionar credenciales fijas a las API de Google Cloud.
- Se importa la clase `GoogleCredentials` de la biblioteca de autenticación de Google, para trabajar con credenciales.
- Se importa la clase `DetectIntentRequest` para crear solicitudes de detección de intenciones.
- Se importa la clase `QueryInput` para crear objetos de entrada de consulta.

- Se importa la clase `SessionName` para gestionar nombres de sesión.
- Se importa la clase `SessionsClient` para interactuar con las sesiones de DF.
- Se importa la clase `SessionsSettings` para configurar la configuración de las sesiones de DF.
- Se importa la clase `TextInput` para representar datos de entrada de texto.

Después, se generaron tres métodos principales que se describen a continuación.

- **`DialogflowManager`**: este método toma el contexto de la aplicación como parámetro para permitir que la instancia de `LNTranslator` tenga acceso a recursos y funcionalidades específicas de la aplicación en la que se está utilizando. Posteriormente, se intenta abrir un archivo JSON que contiene las credenciales de Google Cloud desde el directorio `assets` de la aplicación. Las credenciales se utilizan para configurar la instancia de `SessionsClient` para la comunicación con DF. Se crea un objeto `SessionName` con el nombre del agente y un ID de sesión. Si ocurren excepciones durante estos pasos, se manejan mediante la impresión de errores en consola.
- **`closeGrpcChannel`**: este método se utiliza para cerrar el canal de comunicación de gRPC[1] después de su uso. Toma un objeto `AutoCloseable` como parámetro y se encarga de cerrarlo, manejando posibles excepciones.
- **`sendQuery`**: este método toma un mensaje de texto como entrada y lo envía a DF para su procesamiento, por medio de las siguientes acciones:

    1. Se construye un objeto `TextInput` con el mensaje de texto y el código de idioma.

[1] "*Google Remote Procedure Call*": concepto de programación que permite que un programa solicite la ejecución de un procedimiento en un servidor remoto

2. Se construye un objeto `QueryInput` utilizando el `TextInput` creado.
3. Se crea una solicitud `DetectIntentRequest` con el nombre de sesión y el `QueryInput`.
4. Se usa el `SessionsClient` para enviar la solicitud y obtener una respuesta.
5. Se llama al método `closeGrpcChannel` para cerrar el canal.
6. Se extrae la respuesta, los parámetros, la acción y contexto. Se colocan en una instancia de la clase `DialogflowResponse`.
7. Se devuelve la instancia de `DialogflowResponse`.

La clase `DialogflowResponse` se utiliza para estructurar y devolver la información obtenida de DF. Tiene cuatro propiedades: respuesta, acción, parámetros y contexto, que contienen la información devuelta por el agente después de procesar una solicitud.

4.2.2 Extracción de datos

Para llevar a cabo este proceso, se implementó la función `processTLN`, la cual recibe como parámetros de entrada la frase del usuario en formato de texto y el contexto de la aplicación. Luego, la función devuelve una meta lista compuesta por listas mutables[2]. Estas listas, a su vez, contienen elementos de tipo string, y es en ellas donde se almacenan los datos extraídos durante el proceso. Antes de comenzar con la extracción de datos, se identifica el estado de la aplicación. Esta comprobación es esencial, ya que el comportamiento de la función `proccessTLN` varía significativamente según si se encuentra en la fase de creación de perfil o en la fase de navegación en la web.

4.2.2.1 Comprobación del estado de la aplicación

Para realizar esta tarea, se definió la clase `AppState` bajo el patrón de diseño *singleton*. Según Montero *et al.* [57], el *singleton* es un patrón que permite que una clase tenga una

[2]Estructura de datos, que puede ser modificada después de su creación

instancia única y proporciona un punto de acceso global a esa instancia. Las características más destacables de la clase `AppState` se enlistan a continuación:

- Cuenta con un constructor privado con el propósito de evitar la creación de nuevas instancias fuera de la clase.
- Tiene la propiedad `currentProcess` de tipo *string*, con la cual se identifica el estado de la aplicación.
- Cuenta con un `companion object` para proporcionar un punto de acceso global a la instancia única de la clase. Dentro de este objeto se declara lo siguiente:
  - La variable privada `instance`, que almacena la instancia única de la clase y se inicializa en nulo.
  - La función `getInstance`, con la cual se obtiene la instancia única de la clase `AppState`.
- Cuenta con la anotación `@Synchronized`, para el manejo de la instancia en el entorno multi-hilo.

La configuración previa permite garantizar que exista una sola instancia del estado de la aplicación. Además, asegura que siempre se obtenga la versión actualizada. Para acceder y editar el valor del estado, desde cualquier parte de la aplicación, se deben implementar las líneas de código que se muestran en el bloque 4.7, las cuales se explican a continuación.

Código 4.7: Acceder y editar el estado de la aplicación

```
1   val appState = AppState.getInstance()
2   val estadoActual = appState.currentProcess
3   appState.currentProcess = "NuevoValor"
```

- En la línea 1, se obtiene la instancia única de la clase `AppState`, mediante la función `getInstance()`.

- En la línea 2, se accede al valor del estado a través de la propiedad `currentProcess`.
- En la línea 3, se edita el valor del estado de la aplicación.

4.2.2.2 Clase auxiliar para la extracción de entidades nombrabas, sustantivos y relaciones sintácticas a partir de texto

Para llevar a cabo esta tarea, se desarrolló la clase `CoreNLPHandler`. Esta clase se encarga del procesamiento de texto haciendo uso de la biblioteca *Stanford CoreNLP*. Su principal objetivo es la extracción de entidades nombradas, relaciones sintácticas y sustantivos. A continuación, se detallan los aspectos relevantes de la clase y sus funciones.

- La clase cuenta con un constructor que recibe dos parámetros de entrada para inicializar la clase. El primer parámetro es una cadena de texto que representa la URL de un servidor de *CoreNLP*. El segundo parámetro es un entero que representa el número de puerto en el que el servidor recibe solicitudes de procesamiento de texto.
- La clase tiene una función llamada `processText`, con la cual se inicia el procesamiento del texto. Esta función recibe tres parámetros de entrada:

  1. `text`: es una cadena de texto que representa el contenido que se va a procesar.
  2. `relationTypes`: es una lista mutable de elementos tipo `string`, que contiene los tipos de relaciones sintácticas que se desean extraer del texto.
  3. `onCompleteListener`: es un objeto que implementa la interfaz `OnCompleteListener` la cual se explica más adelante.

  La función `processText` crea una instancia de la clase `CoreNLPAsyncTask` con los parámetros `text`, `relationTypes` y `onCompleteListener` como argumentos. Luego, con la instancia, se ejecuta el método `execute()` de `AsyncTask`, lo que inicia la tarea de forma asíncrona. Según la documentación de Android [54],

esta clase permite ejecutar tareas en un hilo separado del hilo principal y cuenta con métodos para ejecutar código en el hilo secundario y actualizar el hilo principal. Algunos de estos métodos fueron utilizados y se explican en las siguientes viñetas.

Código 4.8: Método doInBackground

```
1 override fun doInBackground(vararg voids: Void): Annotation {
2    try {
3       val props= Properties()
4       props.setProperty("annotators","tokenize,pos,lemma,ner,depparse")
5       val pipeline= StanfordCoreNLPClient(props,coreNLPUrl,coreNLPPort,2)
6       val document= Annotation(text)
7       pipeline.annotate(document)
8       return document
9    } catch (e: Exception) {
10       e.printStackTrace()
11       throw e
13   }
14 }
```

- `doInBackground`: este método se ejecuta en un hilo secundario y es donde se realiza la tarea. Como lo muestra el bloque de código 4.8, se inicia un *try-catch* para manejar excepciones que puedan ocurrir durante el procesamiento. Dentro del `try`, se crea una instancia de la clase `Properties`. Con esta variable, se configuran las propiedades del procesador de lenguaje natural de *Stanford CoreNLP*. En la línea cuatro del código 4.8, se establecen las propiedades para indicar qué anotadores se utilizan. En este caso, se configuran los anotadores correspondientes para tokenización, análisis de partes de la oración (POS), lematización, reconocimiento de entidades nombradas y análisis de dependencias.

Código 4.9: Método onPostExecute

```
1 override fun onPostExecute(result: Annotation) {
2   try {
3      val namedEntities = extractNamedEntities(result)
4      val relations= extractRelationsByTypes(result,relationTypes)
5      val nouns= extractNouns(result)
6      if (namedEntities != null) {
7           onCompleteListener.onResult(namedEntities,relations,nouns)
8       }
9   } catch (e: Exception) {
10    onCompleteListener.onError("Excepcion:${e.message}")
11  }
12 }
```

Posteriormente, en la línea cinco del bloque 4.8, se crea una instancia de `StanfordCoreNLPClient` utilizando las propiedades configuradas en el constructor de la clase (la URL y el puerto). El número dos, en los parámetros, se refiere al número de hilos utilizados por el servidor *CoreNLP*. En la línea seis, se crea un objeto de tipo `Annotation`, que almacena el texto a procesar. En la línea siete, se emplea el pipeline de CoreNLP para realizar anotaciones en el documento, aplicando los anotadores configurados.

Finalmente, en la línea ocho, se devuelve el objeto `document`, que ahora contiene el texto procesado y las anotaciones generadas. Cuando se produce una excepción, se captura e imprime para su posterior manejo, tal como se muestra en las líneas 9-11.

- `onPostExecute`: este método se ejecuta en el hilo principal una vez que el método `doInBackground` ha finalizado su ejecución. En él, se extraen las entidades nombradas, relaciones y sustantivos del texto procesado a través de llamadas a funciones locales. Además, se invocan los métodos definidos en la interfaz

`OnCompleteListener` para notificar los resultados o errores (ver bloque de código 4.9).

- La clase, contiene una interfaz llamada `OnCompleteListener` que define dos métodos: `onResult` y `onError`. Estos métodos se llaman cuando se completan las tareas de procesamiento y permiten manejar los resultados o errores.

Dentro de la clase `CoreNLPHandler`, se definieron tres funciones para filtrar las anotaciones generadas. A continuación, se detallan estas funciones:

- **`extractRelationsByTypes`**: esta función se encarga de extraer relaciones sintácticas específicas a partir de un objeto de tipo `Annotation` (ver bloque de código 4.10). Recibe dos parámetros de entrada:

  1. El primer parámetro es un objeto de tipo `Annotation` que contiene el documento de texto procesado por *Stanford CoreNLP*.
  2. El segundo parámetro es una lista mutable de elementos de tipo `string` que especifica los tipos de relaciones que se desean extraer.

  Dentro de la función, se realizan operaciones específicas. En la línea dos del código 4.10, se inicializa una lista mutable llamada `relations` que se utiliza para almacenar las relaciones sintácticas extraídas. En la línea tres, se accede a las sentencias del documento procesado y se obtienen las oraciones del texto. Luego, en las líneas cuatro y cinco, se accede a las dependencias colapsadas de la oración, que representan las relaciones entre las palabras. A continuación, en las líneas seis a once, se realiza la extracción de relaciones sintácticas siguiendo estos pasos:

  - Se itera a través de las dependencias y se verifica si el tipo de relación coincide con alguno de los tipos especificados en la lista `relationTypes`. Si hay una coincidencia, extrae la relación.

- Se agrega la relación a la lista `relations` en el orden siguiente: tipo de relación seguido de las palabras que conforman la relación.

Finalmente, la función retorna la lista mutable con las relaciones encontradas.

Código 4.10: Función extractRelationsByTypes

```
1 private fun extractRelationsByTypes(document: Annotation, relationTypes:
   MutableList<String>): MutableList<String> {
2   val relations = mutableListOf<String>()
3   val sentences = document.get(CoreAnnotations.SentencesAnnotation::class
   .java)
4   for (sentence in sentences) {
5     val collapsedDependencies = sentence.get(SemanticGraphCoreAnnotations
   .CollapsedDependenciesAnnotation::class.java)
6     for (collapsedDependency in collapsedDependencies.typedDependencies()
   ) {
7         if (relationTypes.contains(collapsedDependency.reln().shortName))
    {
8             val governor = collapsedDependency.gov()
9             val dependent = collapsedDependency.dep()
10            val relation = "${collapsedDependency.reln().shortName},${
   dependent.word()},${governor.word()}"
11            relations.add(relation)
12        }
13    }
14  }
15  return relations
16 }
```

- **`extractNamedEntities`**: esta función se encarga de extraer entidades nombradas a partir de un objeto de tipo `Annotation`. En la línea dos del código 4.11, se crea una lista mutable llamada `namedEntities` en donde son almacenadas las entidades nombradas extraídas. De forma similar a la función `extractRelations-`

`ByTypes`, esta función utiliza dos bucles anidados para iterar a través de las oraciones y tokens del documento.

En la línea cuatro, comienza el bucle interior. En la línea cinco, se obtiene el *tag* de entidad nombrada (`nerTag`) de cada token. El `nerTag` indica si el token es parte de una entidad nombrada y especifica el tipo de entidad[1]. Luego, en la línea seis, se verifica si el `nerTag` del token actual es diferente a "`O`", ya que este símbolo se utiliza para representar que un token no es parte de ninguna entidad nombrada. Si el `nerTag` no es "`O`", entonces se considera como una entidad nombrada válida y se agrega el token a la lista `namedEntities`, junto con su tipo de entidad.

Al finalizar la función, se devuelve la lista `namedEntities`, que contiene las entidades nombradas extraídas del documento.

Código 4.11: Función extractNamedEntities

```
1 private fun extractNamedEntities(document: Annotation): MutableList<
   String>? {
2   val namedEntities = mutableListOf<String>()
3   for (sentence in document.get(CoreAnnotations.SentencesAnnotation::
   class.java)) {
4       for (token in sentence.get(CoreAnnotations.TokensAnnotation::class.
   java)) {
5           val nerTag = token.get(CoreAnnotations.NamedEntityTagAnnotation
   ::class.java)
6           if (nerTag != "O") {
7               val word = token.word()
8               namedEntities.add("$word: $nerTag")
9           }
10      }
11  }
12  return namedEntities
13 }
```

[1]Por ejemplo: persona, lugar, organización, por mencionar algunas.

Código 4.12: Función extractNouns

```
1 private fun extractNouns(document: Annotation): MutableList<String> {
2   val nouns = mutableListOf<String>()
3   val sentences = document.get(CoreAnnotations.SentencesAnnotation::class
    .java)
4   for (sentence in sentences) {
5       val wordsWithPOS = sentence.get(CoreAnnotations.TokensAnnotation::
    class.java)
6       for (word in wordsWithPOS) {
7           if (word.get(CoreAnnotations.PartOfSpeechAnnotation::class.java
    ).startsWith("NOUN")) {
8               val noun = word.word()
9               nouns.add(noun)
10          }
11      }
12  }
13 return nouns
14 }
```

- **`extractNouns`**: esta función se encarga de extraer sustantivos a partir de un objeto de tipo `Annotation`. En la línea dos del código 4.12, se crea una lista mutable llamada `nouns` en la que se almacenan los sustantivos extraídos. Al igual que las funciones anteriores, esta función utiliza dos bucles anidados para iterar a través de las oraciones y tokens del documento. En las líneas 7-10, se utiliza la anotación de Part-of-Speech (POS) para determinar si una palabra es un sustantivo. Si se encuentra una palabra que cumple con la condición de ser un sustantivo, se agrega a la lista de sustantivos (`nouns`). Finalmente, la función devuelve la lista `nouns`, que contendrá todos los sustantivos extraídos.

4.2.2.3 Configuración e implementación del servidor Stanford CoreNLP

Según la documentación de *Stanford CoreNLP* [58], *CoreNLP* es una herramienta de procesamiento del lenguaje natural que permite derivar una amplia variedad de anotaciones

lingüísticas a partir de texto. Estas anotaciones incluyen tokens, oraciones, partes del discurso, entidades nombradas, análisis de dependencia, análisis de sentimientos y más. *CoreNLP* presenta dos formas con las cuales se puede integrar esta herramienta en proyectos; mediante una biblioteca local o un servidor en línea.

Implementación con biblioteca local

En las primeras implementaciones del prototipo, fue utilizada la biblioteca de forma local en la tablet. Se detectó que la biblioteca, en su versión ligera con el conjunto limitado de funcionalidades que requiere el prototipo, utiliza 385 MB de memoria. Además, se experimentaron problemas de bloqueo que resultaron en ralentizaciones de aproximadamente 14 segundos, en las tareas de extracción de datos a partir de una frase. Se investigaron las posibles causas de este problema, y se llegó a la conclusión de que tenía principalmente dos fuentes: la optimización en el uso de recursos de la aplicación para reducir la carga en la memoria RAM y las incompatibilidades entre las versiones de la biblioteca y el sistema Android.

Uno de los experimentos realizados para identificar el consumo de recursos en la memoria RAM, demostró que el Servicio de Reconocimiento de Voz (SRV) consume memoria RAM repetidamente, ya que procesa audio de manera continua en tiempo de ejecución. Por esta razón, se llevó a cabo un experimento en el que se ejecutaron tanto el SRV como el método de extracción de datos (*CoreNLP*) de manera independiente y aislada de otros procesos. La conclusión obtenida fue que, incluso al ejecutarlos juntos de forma independiente, seguían provocando bloqueos.

Además, se encontró información en blogs de desarrolladores de aplicaciones para Android que indicaba que la versión ligera de la biblioteca no funciona adecuadamente en dispositivos móviles, ya que no está optimizada para este entorno. Sin embargo, no se pudo confirmar esta información a través de la documentación oficial de *Stanford CoreNLP*, ya que ésta solo menciona que la biblioteca está diseñada para su implementación en paque-

tes y lenguajes de programación determinados, sin hacer referencia a tipos específicos de dispositivos o arquitecturas.

Implementación final con servidor remoto

Se optó por implementar un servidor remoto que permite enviar solicitudes de análisis de texto a través de una interfaz web y recibir las respuestas correspondientes. Algunas de las ventajas de usar un servidor local son:

- Al ser una solución basada en la web, el servidor es compatible con una variedad de plataformas y lenguajes de programación.
- Se aprovechan los recursos compartidos en lugar de depender de los limitados recursos del dispositivo móvil.

Los pasos que se siguieron para ejecutar el servidor de *CoreNLP* en una computadora personal se enlistan a continuación:

- Se descargó el archivo ZIP correspondiente al idioma español desde la página oficial de Stanford CoreNLP[1]. Este archivo ZIP contiene los siguientes elementos.
  - El archivo *JAR* de código principal de *CoreNLP*.
  - El archivo *JAR* de modelos de procesamiento de lenguaje natural en español.
  - Las bibliotecas necesarias para la ejecución del servidor.
  - La documentación asociada al servidor y sus funcionalidades.
- Se descargó e instaló *Java* en la computadora utilizando los siguientes comandos en la terminal.

[1] https://stanfordnlp.github.io/CoreNLP/download.html

Código 4.13: Comandos para instalar Java en Ubuntu

```
- sudo apt update
- sudo apt install openjdk-17-jdk
- sudo apt install openjdk-17-jdk
```

- Se configuraron las variables de entorno del sistema para que el algoritmo de *CoreNLP* pueda encontrar *Java*.
- Se descomprime el archivo ZIP y mediante línea de comandos, se accede al directorio donde se encuentra la carpeta de *Stanford CoreNLP*.
- Dentro del directorio se ejecuta la siguiente línea de comando:

Código 4.14: Comandos para ejecutar el servidor

```
java -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -
   serverProperties StanfordCoreNLP-spanish.properties -port 9000
   -timeout 10000
```

A continuación se explica la línea del bloque de código 4.14:

- `java`: comando para ejecutar programas Java desde la línea de comandos.
- `-cp ``*''`: indica la ruta de las clases que se utiliza para buscar las clases y bibliotecas necesarias para ejecutar el servidor. El "*" indica que se deben incluir todas las bibliotecas *JAR*.
- `edu.stanford.nlp. pipeline.StanfordCoreNLPServer`: clase que inicia el servidor.
- `serverProperties StanfordCoreNLP-spanish.properties`: parámetro que especifica las configuraciones del servidor para procesar texto en idioma español.

- `port 9000`: parámetro que especifica el número de puerto en el cual el servidor recibe solicitudes.
- `timeout 10000`: parámetro que especifica el tiempo de espera máximo. En este caso se establecen 10 segundos.

Después de seguir estos pasos, en la terminal aparecen las líneas que se muestran en la Figura 4.15, las cuales indican que el servidor está ejecutándose correctamente y en espera de solicitudes. Estas solicitudes pueden ser generadas mediante el uso de la función `processText` de la clase mencionada en la sección 4.2.2.2.

```
[main] INFO CoreNLP - Threads: 8
[main] INFO CoreNLP - Starting server...
[main] INFO CoreNLP - StanfordCoreNLPServer listening at /[0:0:0:0:0:0:0:0]:9000
```

Figura 4.15: Servidor *CoreNLP* ejecutándose.

4.2.2.4 Extracción de datos en la fase de creación de perfil de usuario

Para llevar a cabo esta tarea, se utiliza la función `processTLN` de la clase `LNTranslator`. En esta función, se identifica el estado de la aplicación, como se describe en la Sección 4.2.2.1. Si el estado de la aplicación tiene el valor "*Perfil*", esto indica que se está ejecutando el proceso de creación de un perfil de usuario. En este punto, se inicia la extracción de datos. Como se menciona en la Sección 3.3.1.2, las respuestas obtenidas durante la creación de un perfil son procesadas para extraer una serie de términos descriptores de contexto. Este proceso se lleva a cabo utilizando las funcionalidades del servidor remoto llamado *CoreNLP*. A continuación, se explican los pasos necesarios para este proceso.

1. Google ha modificado las políticas de seguridad en las aplicaciones desarrolladas para versiones de Android posteriores a la 9.0. Estas nuevas políticas promueven el uso de conexiones HTTPS cifradas en lugar de HTTP. Esto se debe a que el tráfico no cifrado a través de HTTP, puede ser vulnerable a ataques de intermediarios y a

la exposición de datos confidenciales [54]. En esta primera versión del prototipo, el servidor *CoreNLP* se implementó utilizando conexiones HTTP. Por esta razón, se incluye la línea de código que se muestra en el bloque 4.15 para permitir el tráfico de red no cifrado para el nombre de host del servidor.

Código 4.15: Comando para configuración de políticas de seguridad de red.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.P) {
                android.security.NetworkSecurityPolicy.getInstance
                    ().isCleartextTrafficPermitted(hostname)
            }
```

2. Para conectarse al servidor *CoreNLP*, se crea una instancia de la clase `CoreNLPHandler` y se inicializa con el nombre de host del servidor y el puerto por el que recibe solicitudes.

3. Se declara la lista mutable que contiene los tipos de relaciones sintácticas que se desean extraer de la respuesta del usuario. En este caso, se incluyen las relaciones *amod*, *nmod* y *dobj*.

4. Se verifica que la variable contiene la respuesta del usuario tenga un valor distinto a nulo. Si el valor es nulo, se envía una advertencia. En caso contrario, se pasa al siguiente paso.

5. Se llama a la función `processText` de la clase `coreNLPHandler` para realizar la solicitud de procesamiento. Se le pasan tres argumentos a la función:

   a) El texto que se va a procesar.

   b) La lista de tipo de relaciones a extraer.

   c) Un objeto de tipo `CoreNLPHandler.OnCompleteListener` para el manejo de resultados y errores de la solicitud.

Los resultados que se obtienen del método `processText` son 3 listas mutables: entidades, relaciones sintácticas y sustantivos. Estas listas se almacenan en una lista mutable, que al final de la función, se devuelve a la instancia correspondiente. En el caso de obtener errores en la solicitud, se devuelve una notificación del error a la instancia correspondiente.

4.2.2.5 Extracción de datos en la fase de navegación

Al igual que en la fase de creación de perfil de usuario, se utiliza la función `processTLN` de la clase `LNTranslator`. En esta función, se identifica el estado de la aplicación. Si el estado no es "*Perfil*", esto indica que se está llevando a cabo una tarea relacionada con la navegación web. Cuando se cumple la condición mencionada anteriormente, se procede a realizar el reconocimiento de la intención a partir de la frase del usuario de la siguiente manera.

- Se invoca la función `DialogflowManager`, la cual se encarga de configurar la comunicación con DF, como se detalló en la Sección 4.2.1.3.
- Se envía una consulta al agente de DF con el texto de la frase del usuario como parámetro. La respuesta del agente es almacenada en una variable llamada `respuesta`. La variable `respuesta` consta de cuatro propiedades: respuesta textual de agente, acción detectada, parámetros y contexto.

Una vez detectada la intención de la frase del usuario, que se encuentra en la propiedad acción (`respuesta.action`), se procede a manejar los siguientes casos:

Caso 1. Búsqueda

Se clasificaron como intenciones de *búsqueda*, a aquellas relacionadas con la búsqueda en la Web de imágenes, videos, términos y páginas web. Para verificar que la intención de la frase del usuario corresponde a esta clasificación, se realizó la comparación del bloque 4.16.

Código 4.16: Condición de intención de búsqueda.

```
if (respuesta.action== "Videos" || respuesta.action== "verFotos" ||
   respuesta.action== "Termino" ||  respuesta.action=="PagWeb")
```

Si la condición se cumple, entonces se comienza con la extracción de datos. El proceso utiliza el servidor *CoreNLP* de manera similar, en los primeros dos pasos, al proceso explicado en la Sección 4.2.2.4. Sin embargo, a partir del tercer paso, se presentan las diferencias que se detallan a continuación.

- En este caso, la lista de los tipos de relaciones sintácticas, que se desean extraer de la frase del usuario, incluyen las siguientes relaciones: *amod*[1], *nmod*[2], *nsubj*[3],*dobj*[4] y *case*[5].
- Se verifica que la variable que contiene la frase del usuario no sea nula. En caso de que no sea nula, se procede al siguiente paso.
- De manera similar al paso 5 de la Sección 4.2.2.4, se invoca la función `processText`. Los resultados se almacenan en una lista mutable con el siguiente orden y elementos:
  - El primer elemento es una lista mutable que contiene una bandera que indica que se trata de una intención de búsqueda, la respuesta del agente de DF, la intención detectada y la frase del usuario.
  - Los elementos siguientes son las listas obtenidas de la función `processText`, que corresponden a las entidades, relaciones sintácticas y sustantivos. Sin embargo, si la intención es igual a "*Termino*", que se refiere a la búsqueda de un término específico, se envían solo dos elementos, además del primero, que corresponden a las entidades y relaciones sintácticas.

[1]Indica que un adjetivo modifica a un sustantivo.
[2]Indica que una palabra o frase modifica un sustantivo.
[3]Indica que el sustantivo es el sujeto del verbo en una oración.
[4]Indica que un sustantivo recibe la acción del verbo.
[5]Indica la relación entre una preposición y su complemento.

- Finalmente, se devuelve la lista de resultados a la instancia correspondiente de procesamiento.

Caso 2. Modificación de IU

Se identificaron como intenciones de *modificación de la IU*, a aquellas que implican cambios en la interfaz. Estas intenciones se dividieron en cinco listas diferentes. Cada una clasifica las intenciones en función del tipo de modificación que realiza en la interfaz. A continuación, se detallan las categorías y las intenciones que incluyen.

- `AudioVideo`: intenciones relacionadas con la interacción del usuario con audio y video, como pausar, reproducir, repetir, adelantar y retroceder.

- `Imagen`: intenciones relacionadas con la interacción del usuario con imágenes, como cambiar su tamaño, rotar 90° hacia la izquierda o derecha y mover la imagen en el plano.

- `Termino`: intención relacionada con la interacción del usuario en el menú que muestra respuestas a una búsqueda de término.

- `AccionesPagWeb`: intenciones relacionadas con la interacción del usuario en una página web, como hacer clic en enlaces o navegar por menús.

- `Otras`: intenciones que abarcan acciones como ajustar el volumen, cerrar la aplicación, repetir respuestas o navegar entre opciones anteriores y siguientes.

Una vez que se identifica que la intención pertenece al tipo que modifica la IU, y se determina a qué lista corresponde, los datos se extraen de los parámetros proporcionados por el agente de DF. Estos parámetros se obtienen a través de la propiedad `respuesta.parameters`. Cada tipo de intención tiene parámetros específicos que recibe del agente de DF. Se detallan los parámetros de cada intención en la Tabla 4.2.

Tabla 4.2: Parámetros de intenciones

Intención	Nombre	Tipo	Utilidad
Adelantar y retroceder video	cantidadTiempo	string	Identificar si el usuario habla de minuto(s) o segundo(s).
	cantidad	entero	Identificar la cantidad de minuto(s) o segundo(s).
	cantidadAbstracta	string	Identificar si el usuario dice cantidades abstractas como *mucho*, *poco*, entre otras.
Girar imagen	dirección	string	Identificar la dirección del giro, derecha o izquierda.
Aumentar y reducir imagen	factor	entero	Identificar el factor de escala con el cual se desea modificar el tamaño de una imagen
	porcentaje	entero	Identificar el porcentaje de escala con el cual se desea modificar el tamaño de una imagen.
Mover imagen	dirección	string	Identificar la dirección hacia donde se desea mover la imagen: derecha, izquierda, arriba y abajo.
	cantidadAbstracta	string	Identificar si el usuario dice cantidades abstractas como *mucho*, *poco*, entre otras.
Seleccionar opción	opción	entero	Identificar la opción que se desea seleccionar.
	ordinal	entero	Identificar la opción deseada a través de expresiones numéricas en forma de números ordinales.
Volumen	dirección	string	Identificar expresiones como *subir*, *bajar*, entre otras.
	cantidad	entero	Identificar el nivel de volumen deseado por el usuario, que varía en un rango de 1 a 15.
	tamaño	string	Identificar expresiones como *disminuye*, *aumenta*, entre otras.

Luego, los resultados se almacenan en una lista mutable con el siguiente orden y elementos:

- El primer elemento es una lista mutable que contiene: una bandera que indica que se trata de una intención de modificación de la IU, la respuesta del agente de DF y

la intención detectada.

- El segundo elemento es una lista mutable que contiene los parámetros correspondientes a la intención detectada.

Finalmente, los resultados son devueltos a la instancia correspondiente.

4.3 Implementación del módulo de operaciones

Como se explicó en la Sección 3.2, el módulo de operaciones es responsable de gestionar la ejecución de eventos, coordinar la presentación de resultados a través de la IU y mantener la comunicación de datos con el módulo generador de respuestas. En esta sección, se describen las funciones de la clase `OperationModule` que se utilizan para llevar a cabo estas tareas.

4.3.1 Procesamiento de intenciones de tipo búsqueda

La función `processNavigationAction` se encarga de gestionar las intenciones relacionadas con búsquedas, como buscar imágenes, videos, términos o páginas web. Recibe como parámetros el contexto de la aplicación y una lista mutable, obtenida durante la extracción de datos en el caso de búsqueda (ver Sección 4.2.2.5). A partir de esta lista, se extrae la intención de la frase del usuario.

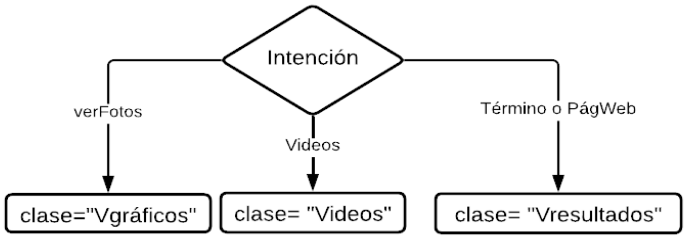


Figura 4.16: Diagrama de flujo para la toma de decisiones en la función `processNavigationAction`

El control de flujo se realiza mediante la instrucción `when`, tal como se ilustra en la

Figura 4.16, con el propósito de determinar la acción que se llevará a cabo en función de la intención identificada. En este contexto, se introduce una variable denominada `clase`, la cual almacena el nombre de la actividad encargada de mostrar los resultados que corresponden a la intención. Después, los resultados generados por el módulo generador de respuestas (consultar la Sección 4.4 para más detalles) se almacenan en una lista mutable de nombre `respuestas`. En el caso de la intención "*Termino*", se agrega a esta lista la frase del usuario.

Posteriormente, se concatena el nombre del paquete de la aplicación con el valor de la variable `clase` y el resultado se asigna nuevamente a la variable `clase`. Esto se hace para obtener el nombre completo de la clase que se abrirá. Luego, en la línea de código 4.17, se utiliza la función `Class.forName(clase)` para obtener una referencia a la clase cuyo nombre está contenido en la variable clase. Esto permite la creación de una instancia de la clase correspondiente.

Código 4.17: Línea para crear una instancia de clase.

```
val intentClass = Class.forName(clase)
```

Finalmente, se llama a la función `openActivitywithData` con los siguientes parámetros:

- contexto de la aplicación,
- instancia de la clase que se va a abrir,
- lista mutable `respuestas`

4.3.2 Procesamiento de intenciones para la modificación de la IU

La función `processIUAction` tiene como objetivo coordinar la transmisión de datos relacionados con eventos que ocasionan modificaciones en la IU. Esta función recibe como parámetros el contexto de la aplicación y una lista mutable de datos, que se obtiene

previamente durante el proceso de extracción detallado en la Sección 4.2.2.5. Se crea un `intent` con el nombre "*actualizando*", para enviar una notificación de actualización a las interfaces de la aplicación.

Después, la lista de datos y una clave se incorporan al `intent`. En la línea 2 del bloque de código 4.18, se observa que la clave corresponde a la variable `currentProcess`, la cual contiene el estado actual de la aplicación. Se seleccionó esta clave para garantizar que solo la interfaz que conozca la cadena que coincida con el estado actual, pueda recibir los datos almacenados en el `intent`. Finalmente, mediante la línea 5, se transmite el `intent` con los datos actualizados a través de un `broadcast`, posibilitando la actualización de las interfaces abiertas.

Código 4.18: Líneas empleadas para almacenar datos y enviar un intent.

```
1 intent.putExtra(
2                 currentProcess,
3                 ArrayList(datosEnviar)
4                )
5 context.sendBroadcast(intent)
```

4.3.3 Iniciar una nueva actividad en la aplicación

La función `openActivitywithData` se encarga de iniciar una nueva actividad y transmitir datos a la actividad de destino. Esta función tiene cuatro parámetros de entrada: el contexto de la aplicación, una referencia a la clase de la actividad de destino, una etiqueta opcional (`clave`) y una lista de datos.

Código 4.19: Función openActivitywithData.

```
  private fun openActivitywithData(context: Context, claseDestino:
     Class<*>?, clave: String?, datos:ArrayList<String>) {
        val intent = Intent(context, claseDestino)
        intent.putExtra(clave, datos)
        intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
        context.startActivity(intent)
    }
```

La función, representada en el bloque 4.19, realiza las siguientes tareas: crea un `intent` que incluye el contexto de la aplicación y la clase de destino como argumentos. Luego, adjunta los datos a enviar junto con sus claves correspondientes en el `intent`. Finalmente, inicia la nueva actividad, permitiendo la transferencia de datos entre los componentes de la aplicación.

4.3.4 Recepción y procesamiento en las actividades

Cada actividad posee una interfaz y funciones particulares, pero comparten una serie de pasos para recibir datos del módulo de operaciones. Estos pasos se llevan a cabo tanto al iniciar la actividad como al recibir datos mediante un broadcast enviado por el módulo.

4.3.4.1 Recepción de datos en inicio de actividad

Cuando una actividad inicia su ciclo de vida, el método `onCreate` entra en ejecución. Este método se encarga de establecer el diseño de la interfaz de usuario y configurar el estado inicial. En este contexto, se usó la clase `AppState` para asignar un valor a la variable `currentProcess`, actualizando así el estado actual de la aplicación. Posteriormente, se llevaron a cabo los siguientes pasos para manejar la recepción de datos provenientes del `intent` que inició la actividad.

- Se guarda el `intent` en una variable local.
- Se verifica si el `intent` contiene *extras*, como se indica en la línea uno del bloque de código 4.20.
- En caso afirmativo, lo que indica que el `intent` incluye datos, se procede a almacenarlos para su posterior procesamiento, tal como se ilustra en las líneas dos y tres.

Código 4.20: Condición para verificar si el Intent tiene extras.

```
1 if (intent.hasExtra(clave)) {
2   var datos = intent.getStringArrayListExtra(clave)
3   datosMutableList = ArrayList(datos)  }
```

4.3.4.2 Recepción de datos enviados mediante un broadcast

Para que la actividad pueda recibir los datos que el módulo de operaciones envía a través de un broadcast, se implementaron los siguientes pasos:

- En el método `onResume`[1], se creó un `IntentFilter` para especificar el tipo de intento que la actividad está interesada en recibir. En este caso, se filtran los intentos con el nombre "*actualizando*" como se muestra en el bloque 4.21. Y se registra el receptor de transmisiones *BroadcastReceiver* en el contexto de la aplicación.

Código 4.21: Filtro de intents.

```
1 override fun onResume() {
2   super.onResume()
3   val filter = IntentFilter("com.example.modelonavweb2.actualizando")
4   registerReceiver(broadcastReceiver, filter)
5   }
```

- Dentro de la actividad, se implementó un `BroadcastReceiver`. Este receptor está diseñado para recibir y gestionar mensajes de un broadcast.
- Para extraer los datos del `intent`, se utiliza el método `getStringArrayList-Extra` en la línea tres del bloque de código 4.22. En este ejemplo, se espera que los datos se hayan enviado con la clave "*VideoWeb*".
- Los datos extraídos se almacenan en la variable `nuevosDatos`, permitiendo así su procesamiento subsiguiente por parte de la actividad.

[1]Método que se ejecuta cuando la actividad se reanuda después de un estado de pausa o detención.

Código 4.22: BroadcastReceiver.

```
1 private val broadcastReceiver: BroadcastReceiver = object :
   BroadcastReceiver() {
2       override fun onReceive(context: Context?, intent: Intent) {
3          val nuevosDatos = intent.getStringArrayListExtra("VideoWeb")
.
.    // PROCESAMIENTO DE DATOS RECIBIDOS**
.
n   }
```

- El `BroadcastReceiver` se desconecta en los métodos `onPause` y `onDestroy` siguiendo las prácticas de gestión de recursos en Android.

4.3.4.3 Procesamiento de datos en las actividades

En este primer prototipo, fueron implementadas cinco actividades con su interfaz correspondiente. Estas actividades son:

- `VideoWeb`: se encarga de la reproducción de videos e interactúa con el reproductor o la página web que contiene el video.

- `Vresultados`: su función principal es mostrar los resultados obtenidos de una búsqueda, además de gestionar acciones relacionadas con la selección y visualización de cada opción.

- `Vgraficos`: está diseñada para mostrar imágenes obtenidas de la web y ejecutar acciones solicitadas por el usuario, como cambiar el tamaño de la imagen, entre otras.

- `Web`: su propósito es presentar una página web y facilitar la interacción con los elementos presentes en dicha página.

Por un lado, los datos que llegan cuando una actividad es inicializada, son principalmente los resultados de búsqueda y respuestas proporcionadas desde el módulo de operaciones. Entonces, el método `onCreate` entra en ejecución, establece el diseño de la interfaz de usuario y configura el estado inicial usando los datos recibidos.

Por otro lado, los datos recibidos por el `BroadcastReceiver` consisten principalmente en una intención y los parámetros extraídos de una frase proporcionada por el usuario. A través de una estructura de control de flujo, se identifica el evento desencadenado por la intención y, utilizando los parámetros correspondientes, se ejecuta una o varias funciones diseñadas para llevar a cabo las acciones. Cabe mencionar que las acciones que puede realizar el prototipo, corresponden con las intenciones que se encuentran en la Tabla 4.1.

4.4 Implementación del generador de respuestas

En este primer prototipo, el módulo generador de respuestas se encarga de sintetizar respuestas en lenguaje natural. Estas respuestas se fundamentan en los resultados obtenidos mediante la detección de intenciones, así como en la extracción de contenido. Las respuestas son generadas a partir de distintas fuentes:

- Agente de DF: como se mencionó en la sección 4.1.1.4, se definieron una serie de respuestas que se adaptan, según la intención y parámetros detectados en la frase del usuario. Por ejemplo:

  "*Girando la imagen a la* `$direccion`"

  o

  "*Aquí tienes algunos resultados, recuerda que puedes decir "siguiente" para que te muestre otra opción.*"

  Donde `$direccion` es un parámetro.

- Analizador del contexto de las páginas web: cuando el usuario selecciona una opción de entre los resultados presentados por el prototipo, se utiliza la función

`extractTextURL` (ver Sección 4.8) para recuperar el texto introductorio de la página web asociada a la elección. Este texto, junto con el título y la URL fuente, se integra para formar el texto que se sintetiza y presenta como respuesta.

- Función `processRecognitionResult` de la clase `AudioCaptureService`: en esta función se establecieron respuestas relacionadas con errores y notificaciones que puedan surgir en cualquiera de los módulos del prototipo. Su propósito es mantener al usuario informado acerca de eventos relevantes y brindar asistencia en la corrección de posibles errores cometidos por el usuario. Por ejemplo:

"*La acción que solicitaste no está disponible en esta interfaz, prueba con otra*"

o

"*Lo siento, no entendí... ¿Podrías repetirlo?*"

Para sintetizar las respuestas, se creó la clase `TextToSpeechService`. Esta clase facilita la síntesis de voz en la aplicación mediante el servicio nativo de Android, `TextToSpeech`. La clase cuenta con un constructor que recibe dos parámetros: el contexto de la aplicación y una función de devolución de llamada que se ejecutará cuando el servicio TextToSpeech esté inicializado. Posteriormente, se inicializa una variable con una instancia de `TextToSpeech`. También, se implementaron las funciones, cuyo funcionamiento se detalla a continuación:

- `onInit`: este método se llama cuando la inicialización del servicio `TextToSpeech` se completa. Si la inicialización es exitosa, se configura el idioma (en este caso, español) y se ejecuta la función de devolución de llamada que se pasó al constructor.

- `sintetizartexto(texto)`: este método permite sintetizar y reproducir un texto dado a través del servicio `TextToSpeech`.

- `shutdown`: este método detiene la reproducción de voz actual y apaga el servicio `TextToSpeech`.

La clase `TextToSpeechService` es instanciada en diversos componentes de la aplicación, como en el módulo de operaciones; para convertir a voz las respuestas generadas antes de presentarlas al usuario, en el servicio `AudioCaptureService`; para informar al usuario sobre eventos o errores y en las actividades encargadas de mostrar los resultados en la interfaz de usuario; para ofrecer una experiencia auditiva al usuario al presentar contenido.

4.5 Generación de consultas

Este módulo se encarga de generar cadenas de consulta, basadas en la frase y el contexto del usuario. Esta tarea se logra mediante los siguientes pasos:

- El módulo de operaciones transmite al módulo generador de consultas las entidades nombradas, relaciones sintácticas y sustantivos extraídos de la frase del usuario.
- Se crea un conjunto de datos general con las entidades nombradas, relaciones sintácticas y sustantivos para evitar repeticiones de palabras.
- Las palabras clave del perfil del usuario se obtienen utilizando la función `getUserProfile` de la clase `UserProfileDatabaseHelper` (consultar Sección 4.6.3).
- Se revisa la tabla del perfil del usuario para verificar si algún tipo de dato coincide con el conjunto de datos general. En caso afirmativo, se extraen las palabras clave asociadas al tipo de dato y se incorporan al conjunto de datos general.
- Se agrega una palabra "*infantil*", si el usuario es menor de edad.
- Se buscan las palabras *imagen(es)* y *video(s)*, en el conjunto de datos general, para establecer un filtro de búsqueda, mediante las cadenas de texto ``&searchType=image'' y ``&q=site:youtube.com'' respectivamente.
- Se genera una cadena de texto `query`, que contiene las palabras del conjunto de datos general.

- Finalmente, se retorna una lista con dos elementos: la cadena `query` y el filtro de búsqueda

4.6 Implementación del gestor del contexto de usuario

Como se menciona en la Sección 3.2, el gestor del contexto de usuario (GCU) se encarga de generar y actualizar la tabla correspondiente al contexto del usuario e historial. Para ello, se realizan las tareas que se detallan en esta Sección.

4.6.1 Comprobación del perfil de usuario

Al iniciar la actividad de lanzamiento de la aplicación, se verifica el estado del perfil de usuario como se muestra en el bloque de código 4.23. A través de la función `checkUserProfileStatus`, se determina si el perfil del usuario está completo. Según el resultado de esta verificación, se ejecutan acciones específicas: si el perfil no está completo, se inicia la actividad `UserProfileActivity` (ver Sección 4.6.2), para que el usuario pueda completarlo; en caso contrario, si el perfil está completo, se abre la actividad principal de la aplicación, permitiendo al usuario comenzar a navegar por la web.

Código 4.23: Comprobación del perfil de usuario

```
1 val profileCompleted = checkUserProfileStatus()
2           if (!profileCompleted) {
3               val intent = Intent(this, UserProfileActivity::class.java)
4               startActivity(intent)
5               finish()
6               return
7           }
```

La función `checkUserProfileStatus` (ver bloque de código 4.24), verifica el estado del perfil de usuario utilizando `SharedPreferences`[1] para obtener el valor asociado

[1]Interfaz de Android que se utiliza para almacenar y recuperar datos en pares clave-valor.

a la clave "*profileCompleted*". Si el valor es `true`, significa que el perfil del usuario está completado; de lo contrario, devuelve false.

Código 4.24: Función checkUserProfileStatus

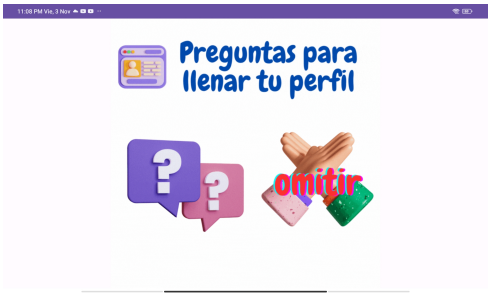
```
1 private fun checkUserProfileStatus(): Boolean {
2       val sharedPreferences = getSharedPreferences("MyAppPrefs", Context.
   MODE_PRIVATE)
3       return sharedPreferences.getBoolean("profileCompleted", false)
4   }
```

4.6.2 Creación del perfil de usuario

Para formar el contexto de usuario, se creó la clase `UserProfileActivity`, con la cual se extraen los datos a partir de un cuestionario por voz. El cuestionario cubre los temas mencionados en la sección 3.3.1.1. En esta primera versión del prototipo, se establecieron 12 preguntas (ver Anexo B). También, se creó una interfaz que reproduce las preguntas del cuestionario en formato de audio y muestra imágenes relacionadas con cada pregunta en pantalla, con el objetivo de guiar al usuario.

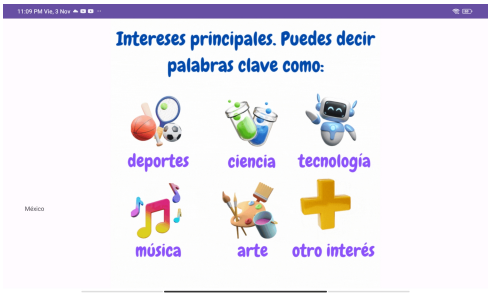
En la figura 4.17 se muestran algunas de las pantallas que se muestran al usuario. Antes de comenzar el cuestionario, se muestra la figura 4.17a en pantalla. Al mismo tiempo, el usuario recibe instrucciones verbales a través de audio para completar el cuestionario. El usuario tiene la opción de omitir las preguntas que no desee contestar, utilizando la palabra "*omitir*", como respuesta. La conversión de voz a texto de las respuestas del usuario, se realizó empleando el método documentado en la sección 4.1.1.1.

Posteriormente, se definió la variable `currentQuestionIndex`, con el fin de almacenar el índice de la pregunta actual que se presenta al usuario en determinado momento y así procesar correctamente cada respuesta y asignar los datos extraídos en las variables correspondientes en el perfil de usuario.



(a) Pantalla introductoria para la creación del perfil de usuario.

(b) Pantalla para la pregunta: ¿cuántos años tienes?



(c) Pantalla para la pregunta: ¿cuáles son tus intereses?

Figura 4.17: Interfaz gráfica para la creación del perfil de usuario.

Para procesar las respuestas del usuario, se implentó la función `procesarRespuestaPAbierta`. A continuación, se describen los aspectos relevantes de esta función.

- Recibe como entrada la respuesta del usuario en formato de texto.
- Devuelve una variable booleana con el valor *true* si la respuesta contiene un valor esperado, o *false* si la respuesta no es válida.
- Analiza las respuestas según el índice de la pregunta actual utilizando la expresión de control de flujo `when`.
- Procesa las respuestas de forma individualizada, adaptando su lógica a las características específicas de cada pregunta. A continuación, se presentan algunos ejemplos.

- En el caso de la pregunta ilustrada en la figura 4.17b, "*¿Cuántos años tienes?*", la función se encarga de extraer un valor numérico en formato entero a partir de una cadena de texto. Esto incluye situaciones en las que la edad se representa mediante símbolos numéricos, como '8', o a través de palabras, como 'ocho'. También, verifica que el valor esté dentro de un rango de valores que represente una edad válida.
- En el caso de la pregunta ilustrada en la figura 4.17c, "*¿Cuáles son tus intereses?*", la función divide la respuesta en palabras individuales y crea una lista con ellas. Posteriormente, filtra las palabras clave que coinciden con las palabras clave de intereses como deporte, viajes, lectura, tecnología, música, etc. Finalmente, convierte el resultado del filtro en un conjunto para evitar palabras repetidas.

El procesamiento de cada respuesta, presenta sus propias particularidades dependiendo de la pregunta a la que corresponde. Sin embargo, comparten el siguiente procesamiento:

- Se convierte toda la respuesta del usuario a minúsculas, con el fin de que la búsqueda de coincidencias de palabras clave sea insensible a las mayúsculas y minúsculas.
- Se verifica si la respuesta contiene la palabra "*omitir*", de ser el caso, se asigna el valor "*NA*" al campo correspondiente.
- Los datos extraídos de las respuestas se guardan en variables de la *data class* `userProfile`.

4.6.3 Creación y actualización de las tablas para el contexto de usuario y el historial de navegación

Para crear la tabla del contexto de usuario, se implementó la clase `UserProfileData-`

`baseHelper`, la cual es una implementación de *SQLiteOpenHelper* en Android, que facilita la gestión de una base de datos *SQLite* localizada en el dispositivo. Esta clase contiene métodos esenciales para la creación y gestión de la base de datos, así como para la manipulación de datos relacionados con el perfil del usuario. A continuación, se detallan algunas de las funciones clave de `UserProfileDatabaseHelper`:

- Se establecen los parámetros iniciales para crear y gestionar una base de datos SQLite local llamada "`UserProfile.db`". Como se muestra en el bloque de código 4.25, los parámetros que se definen son el nombre y versión de la base de datos y el nombre de la tabla.

Código 4.25: Inicialización de parámetros iniciales de la base de datos.

```
1 SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
2   companion object {
3       private const val DATABASE_NAME = "UserProfile.db"
4       private const val DATABASE_VERSION = 1
5       private const val TABLE_NAME = "UserProfile"
6   }
```

- `onCreate`: este método se encarga de crear la estructura inicial de la base de datos cuando se ejecuta por primera vez. En este caso, se utiliza para crear la tabla que almacenará la información del perfil de usuario y la tabla correspondiente al historial de navegación. Como se muestra en el bloque de código 4.26, la tabla `UserProfile` tiene campos relacionados con la información personal del usuario, mientras que la tabla `Historial` registra la actividad de navegación web, incluyendo la dirección de la página web y la frecuencia de visita.

- `userProfile`: es una *data class* que representa un perfil de usuario y contiene propiedades como la edad, género, dirección, listas de intereses, pasatiempos, etc. Cada propiedad tiene un tipo de datos específico y de solo lectura.

Código 4.26: Función onCreate

```
1 override fun onCreate(db: SQLiteDatabase) {
2   // Crea la tabla UserProfile
3   db.execSQL(
4       "CREATE␣TABLE␣$TABLE_NAME␣(" +
5            "edad␣INTEGER,"+"genero␣TEXT,"+"direccion␣TEXT," +
6            "intereses␣TEXT,"+"pasatiempos␣TEXT,"+"estiloMusica␣TEXT,"
   +
7            "estiloPeliculas␣TEXT,"+"estiloLibros␣TEXT,"+
8            "deportes␣TEXT,"+"estiloComida,"+ "sitiosWeb␣TEXT," +
9            "educacion␣TEXT" + ")"
10  )
11  // Crea la tabla Historial
12  db.execSQL(
13      "CREATE␣TABLE␣Historial␣(" +
14              "pagina_web␣TEXT," +
15              "frecuencia_visita␣INTEGER" +
16              ")"
17  )
18 }
```

- `onUpgrade`: para actualizar la tabla `Historial`, este método se encarga de manejar la inserción de una nueva fila a la tabla, como se muestra en la línea 3 del bloque de código 4.27.

Código 4.27: Función onUpgrade

```
1 override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
   newVersion: Int) {
2  if (oldVersion < 2) {
3    db.execSQL("ALTER␣TABLE␣Historial␣ADD␣COLUMN␣nueva_columna␣TEXT;")
4  }
5 }
```

- `insertUserProfile`: esta función permite insertar un nuevo perfil de usuario en la base de datos. Este prototipo es monousuario, por ello, antes de la inserción, borra cualquier perfil existente, asegurando que solo haya un perfil en la base de datos en un momento dado. Como se muestra en el bloque de código 4.28, se emplea la *data class* `userProfile`, para acceder a los datos obtenidos al aplicar el cuestionario explicado en la Sección 4.6.2.

Código 4.28: Función insertUserProfile

```
1 fun insertUserProfile(userProfile: UserProfile) {
2  val db = writableDatabase
3  val values = ContentValues().apply {
4    put("edad", userProfile.edad)
5    put("genero", userProfile.genero)
6    put("direccion", userProfile.direccion)
7    put("intereses", userProfile.intereses.joinToString(","))
8    put("pasatiempos", userProfile.pasatiempos.joinToString(","))
9    put("estiloMusica", userProfile.estiloMusica.joinToString(","))
10   put("estiloPeliculas", userProfile.estiloPeliculas.joinToString(",
   "))
11   put("estiloLibros", userProfile.estiloLibros.joinToString(","))
12   put("deportes", userProfile.deportes.joinToString(","))
13   put("estiloComida", userProfile.estiloComida.joinToString(","))
14   put("sitiosWeb", userProfile.sitiosWeb.joinToString(","))
15   put("educacion", userProfile.educacion)
16  }
17  db.insert(TABLE_NAME, null, values)
18  db.close()
19}
```

- `getUserProfile`: recupera el perfil de usuario almacenado en la base de datos. Retorna un objeto `userProfile` si se encuentra un perfil, o `null` si no hay

ninguno.

- `updateFrequencyVisit`: edita el valor de la frecuencia de visitas en una fila específica de la tabla *Historial*. Como se muestra en la línea 5 del bloque de código 4.29, se llama a la función `update` con los siguientes parámetros de entrada.

  1. El nombre de la tabla.
  2. Los valores a actualizar, representados por un objeto `ContentValues`.
  3. La cláusula `WHERE` que especifica qué fila(s) actualizar. En este caso, se actualizarán las filas donde la columna `pagina_web` coincida con el valor del arreglo `paginaWeb`.

Código 4.29: Función updateFrequencyVisit

```
1 fun updateFrequencyVisit(paginaWeb: String, nuevaFrecuencia: Int) {
2  val db = writableDatabase
3   val valores = ContentValues()
4   valores.put("frecuencia_visita", nuevaFrecuencia)
5   db.update("Historial", valores, "pagina_web = ?", arrayOf(paginaWeb
   ))
6   db.close()
7 }
```

4.7 Implementación del módulo de búsqueda

Como se detalla en la Sección 3.2, el módulo de búsqueda tiene como objetivo buscar y proporcionar un conjunto de URLs, como resultado a una consulta específica. En la Sección 4.3.2.2, se propuso realizar las búsquedas de contenido mediante un crawler; sin embargo, surgieron algunos inconvenientes que obstaculizaban la entrega de contenido relacionado con una búsqueda. En esta sección, se aborda la implementación inicial propuesta y se presenta la versión final realizada para el prototipo.

4.7.1 Primera implementación

En esta etapa, se optó por utilizar una tabla de URLs, donde se almacenaron pares de palabras clave y sus URLs correspondientes. El propósito principal fue contar con una tabla consultable que sirviera como punto de partida para el procesamiento del crawler. La creación de esta tabla se realizó mediante un código similar al presentado en la Sección 4.6.3, con algunas diferencias. Principalmente, se implementaron las siguientes funciones adicionales:

- `insertKeywordUrlPair`: esta función permite la inserción de pares de palabras clave y URLs en la tabla.
- `getUrlbyKeyword`: se implementó esta función para obtener la URL asociada a una palabra clave específica.

Se implementó la clase, denominada `WebCrawler`, con el propósito de iniciar la búsqueda utilizando el crawler a partir de la URL obtenida mediante la función `getUrlbyKeyword`. La función `crawlWebsite` de la clase `WebCrawler`, se encargaba de explorar la página web y seleccionar hasta 10 páginas que contuvieran las palabras clave de la consulta. Una vez obtenidos los resultados, son transferidos al analizador de páginas web para su posterior presentación al usuario.

Limitaciones encontradas

En este enfoque, se proporcionaba una base para la búsqueda de contenido. Sin embargo, presenta limitaciones en comparación con sistemas más avanzados, como motores de búsqueda convencionales. A continuación se presentan dichas limitaciones.

1. Conjunto limitado de URLs predefinidas: en la primera implementación, los resultados de las consultas del usuario estaban restringidos a un conjunto predefinido de URLs, lo que limitaba significativamente la diversidad y amplitud de la información disponible.

2. Indexación: a diferencia de los motores de búsqueda convencionales, que emplean sistemas de indexación avanzados para organizar y clasificar dinámicamente el contenido, la primera implementación no cuenta con un sistema indexación. Esta limitación afecta negativamente la velocidad y eficiencia en la recuperación de resultados de búsqueda.

3. Desafíos de seguridad en *crawling*: algunas de las páginas y sitios web utilizados en esta primera implementación cuentan con medidas de seguridad para protegerse contra actividades de *crawling* y *scraping* no autorizadas. Estas medidas incluyen restricciones de acceso, verificación de bots y otras técnicas que dificultaron o, en algunos casos, impidieron el acceso a ciertos sitios web mediante el proceso de *crawling*.

Estas limitaciones resaltaron la necesidad de implementar una estrategia distinta que permitiera superar estos desafíos y proporcionar una mejor experiencia de búsqueda al usuario.

4.7.2 Implementación final

Debido a restricciones temporales, no fue factible incorporar mecanismos adicionales, como indexación, actualización, y otros módulos, para integrar un motor de búsqueda en

el modelo de navegación. Como alternativa, se tomó la decisión de realizar consultas al motor de búsqueda de Google. Dado que Google implementa técnicas de protección y *anti-scraping* en su sitio web, resultó imposible enviar solicitudes HTTP directas a sus servicios. Por esta razón, se optó por utilizar la API de Google Custom Search. Así, el módulo de búsqueda fue implementado siguiendo los siguientes pasos:

- Configuración del buscador: se accedió al panel de control de Google Search[1], para crear una instancia del motor de búsqueda programable que ofrece en su API. Aquí se llena un formulario donde se proporciona información básica, para personalizar las configuraciones del buscador como idiomas, regiones, entre otras configuraciones, como se muestra en la Figura 4.18.

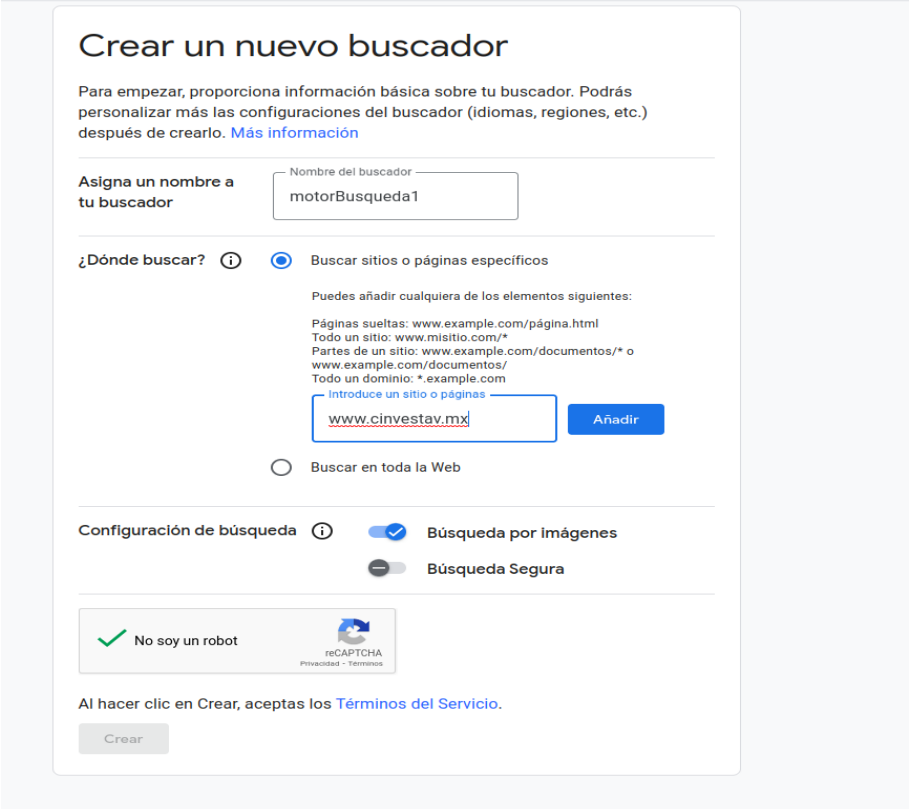


Figura 4.18: Configuración del motor de búsqueda programable

[1]Disponible en https://programmablesearchengine.google.com/controlpanel/

Una vez concluido este paso, se adquirió la clave de API de Google en formato de documento JSON. Esta clave es necesaria para la integración de la API en el contexto de la aplicación.

- En el paquete de la aplicación, se generó una *data class* llamada `SearchResult` que sirve para almacenar los resultados de búsqueda, cada uno representado por un título y un enlace.

- Se implementó la clase `GoogleSearchHelper`, que contiene la lógica para realizar búsquedas en la API de Google Custom Search.

- También, fue implementada la función `performSearch`, que se encarga de realizar búsquedas en la API de Google Custom Search. Sus parámetros de entrada incluyen la clave de la API, la identificación del motor de búsqueda, una función de devolución de llamada que se ejecutará cuando la búsqueda esté completa, la consulta y el tipo de búsqueda[1]. Al concluir la ejecución de la función, se obtienen los resultados en forma de una lista de objetos de tipo `SearchResult`.

Finalmente, los resultados son transmitidos al analizador de contexto de las páginas web.

4.8 Analizador de contexto de páginas web

En esta sección, se detalla como se implementaron las tareas relacionadas con el análisis del contexto de las páginas web.

4.8.1 Generación de contexto a través de palabras clave

Este módulo recibe una lista de objetos del tipo `SearchResult`. Cada objeto contiene un enlace y el título de la página, que posteriormente se analizan. Para realizar el análisis del contexto de las páginas web, fueron implementadas dos funciones principales:

[1]Parámetros obtenidos del generador de consultas

- La función `extractTextURL`, se encarga de extraer texto introductorio de una página web. Primeramente, la función se conecta una la URL especificada como parámetro de entrada y obtiene el documento HTML de la página. Luego, utilizando un selector CSS proporcionado como parámetro, se elige el elemento HTML deseado que contiene el texto a analizar. Se utilizan varios selectores para acceder al texto introductorio en una página web, por ejemplo:

  - `.contenido p:first-child`: selecciona el primer párrafo dentro de un elemento con la clase `contenido`.
  - `#seccion-intro p`: selecciona el párrafo dentro de una división con el identificador `seccion-intro`.
  - `div[data-tipo='intro'] p`: selecciona el párrafo dentro de una división con el atributo de datos `'intro'`.

  Cabe mencionar que estos selectores no son universales, ya que cada diseñador web puede establecer los selectores según sus preferencias. En caso de no encontrar información con los selectores específicos, se utiliza como última opción el selector `p` para seleccionar todos los elementos de párrafo en el documento HTML. Luego, con la función `firstOrNull()`, se obtiene el primer párrafo encontrado. Finalmente, se extrae el texto de ese elemento seleccionado y se devuelve como resultado. En caso de cualquier error durante el proceso, se maneja la excepción y se retorna un mensaje indicando la ocurrencia del error.

- Se implementó la función `getKeywords` (ver bloque de código 4.30), que recibe como parámetro de entrada, un texto obtenido usando la función `extractTextURL`. También, utiliza las funcionalidades del servidor *CoreNLP* para analizar un texto y extraer entidades nombradas y/o palabras que componen relaciones sintácticas específicas. La función realiza una solicitud *POST* al servidor que se ejecuta en `http://localhost:9000/`. Después, se procesa el resultado de la solicitud que

se recibe en formato JSON y se analiza para extraer información relevante de la siguiente forma:

Código 4.30: Función getKeywords

```
fun getKeywords(texto: String): List<String> {
    val url = "http://localhost:9000/annotate"
    val payload = mapOf(
        "annotators" to "tokenize,ssplit,pos,lemma,depparse",
        "outputFormat" to "json", "timeout" to "30000",
        "ssplit.eolonly" to "true", "input" to texto
    )
    val response = post(url, data = payload)
    val json = response.jsonObject
    val sentences = json.getJSONArray("sentences")
    val palabrasClave = mutableListOf<String>()
    for (i in 0 until sentences.length()) {
        val tokens = sentences.getJSONObject(i).getJSONArray("tokens")
        val dependencies = sentences.getJSONObject(i).getJSONArray("
           basicDependencies")
        for (j in 0 until tokens.length()) {
            val palabra = tokens.getJSONObject(j).getString("word")
            val lemma = tokens.getJSONObject(j).getString("lemma")
            val depType = dependencies.getJSONObject(j).getString("dep"
               )
            if (depType in listOf("amod", "nmod", "nsubj")) {
                palabrasClave.add(lemma)
            }
        }
        val entidades = detectarEntidades(texto)
        for ((_, entidadesEnGrupo) in entidades) {
            palabrasClave.addAll(entidadesEnGrupo)
        }

    }
    return palabrasClave.distinct()
}
```

- La función itera sobre las oraciones y *tokens* del análisis.
- Para cada *token*, se obtienen la palabra, el lema[1] y la información de dependencia sintáctica.
- Se seleccionan los lemas correspondientes a las palabras, o las propias palabras[2], que forman parte de relaciones sintácticas específicas, tales como *amod*, *nmod* y *nsubj*.
- Se extraen entidades nombradas pertenecientes a los siguientes grupos: personas, ubicaciones geográficas, organizaciones y fechas.
- Se combinan los conjuntos de palabras obtenidos en los dos puntos anteriores.
- Se eliminan las palabras duplicadas y se retorna el conjunto final.

El conjunto final está integrado con las palabras clave detectadas a partir del análisis de entidades nombradas y la selección específica de relaciones sintácticas. Utilizando las funciones anteriores, se examinan cada una de las páginas web en la lista de objetos del tipo `SearchResult`. Los resultados son utilizados para generar o actualizar la tabla de contexto de páginas web. Los campos de dicha tabla son el título y la URL de la página web y palabras clave.

4.8.2 Navegación dentro de una página web

Después de identificar la intención que señala el deseo del usuario de navegar dentro de la página web, el módulo de operaciones envía la cadena correspondiente a la petición del usuario a la actividad `Web`. Dentro de la actividad, se procesa la petición del usuario. La función privada `extractKeywords` se encarga de extraer la información relevante de la frase del usuario, específicamente la parte que sigue a un comando específico relacionado con acciones de clic o navegación. Por ejemplo, dado el comando "*da clic en servicios*", la función se encargará de extraer la palabra "*servicios*".

[1]Forma base o raíz de las palabras.

[2]Algunos términos, como nombres propios, acrónimos o palabras técnicas, pueden carecer de un lema.

Código 4.31: Función getKeywords de actividad Web

```
private fun getKeywords(frase: String): String {
        val patron = "(da␣clic␣in|da␣click␣in|dale␣clic␣a|Comando␣ve␣a|
            da␣clic␣en|da␣click␣en|ve␣al␣menu|dale␣click␣en|Haz␣clic␣en
            |Realiza␣un␣clic␣en|Accede␣a|Visita|Navega␣hacia|Explora|
            Selecciona|Ve␣a␣la␣secci n|Abre|Consulta)␣(.+)"
        val regex = Pattern.compile(patron, Pattern.CASE_INSENSITIVE)
        Log.d(TAG,"peticion␣frase:␣$frase␣")
        val matcher: Matcher = regex.matcher(frase)
        val textoDespuesDeComando = StringBuilder()
        while (matcher.find()) {
            val texto = matcher.group(2)
            if (textoDespuesDeComando.isNotEmpty()) {
                textoDespuesDeComando.append("␣")
            }
            textoDespuesDeComando.append(texto)
        }
        return textoDespuesDeComando.toString()
    }
```

Como se muestra en el bloque de código 4.31, se ha definido una serie de posibles comandos de clic o navegación, como "*da clic en*", "dale clic a", "ve al menú", entre otras. Para esta tarea, se utiliza el operador "`(.+)`" para capturar cualquier texto que siga al comando. Luego, se implementó un bucle `while` para encontrar todas las coincidencias en la cadena de texto. La información extraída se almacena en una cadena resultante (`textoDespuesDeComando`), que se construye concatenando las palabras y separándolas por espacios, si hay más de una palabra.

Código 4.32: Función buscarYHacerClic

```
private fun buscarYHacerClic(palabrasClave: String) {
    var webView1 = findViewById<WebView>(R.id.vview1)
    palabrasClave.trim().toLowerCase()
    val javascript = """
        var elementos = document.getElementsByTagName("a");
        for (var i = 0; i < elementos.length; i++) {
            if (elementos[i].textContent.trim().toLowerCase() === '${
               palabrasClave}') {
                var claseElemento = elementos[i].classList;
                elementos[i].click();
                break;
                }
            }
            """.trimIndent()
            webView1.evaluateJavascript(javascript, null)
        }
```

Una vez identificada o identificadas la o las palabra(s) clave que indican hacia qué dirección desea navegar el usuario, se emplea la función `buscarYhacerClic` (ver bloque de código 4.32). Esta función toma como parámetro la cadena obtenida al ejecutar la función `extractKeywords`. Dentro de esta función, se realiza una búsqueda en la página web, por medio de un bloque de código *JavaScript* que se ejecuta en el contexto del *WebView*.

- Utiliza la función `document.getElementsByTagName(''a'')` para obtener todos los elementos de enlace en la página.
- Itera sobre estos elementos y compara el texto contenido en cada enlace con las palabras clave proporcionadas.
- Si se encuentra un enlace que coincide, se simula un clic en el enlace.

Capítulo 5

Pruebas, resultados y análisis

La presente investigación tiene como objetivo principal modelar una herramienta de navegación web por voz, fundamentada en el contexto del usuario y los recursos web. Este capítulo se enfoca en la evaluación práctica del prototipo generado a partir de la arquitectura propuesta, la cual surge de los objetivos específicos delineados en el primer capítulo. Un video con la ejecución del modelo puede ser visto en: https://youtu.be/MpjGf7d-fFI.

5.1 Métricas de evaluación

El propósito de esta evaluación es conocer el nivel de satisfacción con respecto a los resultados obtenidos en una búsqueda y la comodidad percibida por los usuarios al utilizar el prototipo. Para llevar a cabo estas evaluaciones, se utilizaron dos instrumentos diseñados para evaluar la experiencia del usuario: el *System Usability Scale* (SUS) y el *Questionnaire for User Interaction Satisfaction* (QUIS).

- **System Usability Scale**

  La Escala de usabilidad de sistemas, según Brooke [59], ofrece una perspectiva general de las evaluaciones subjetivas de usabilidad que puede ser aplicado en diversos contextos. Esta escala consta de diez preguntas estandarizadas, que permiten obtener una medida cuantitativa de la facilidad de uso percibida por el usuario.

- **Questionnaire for User Interaction Satisfaction**

  El cuestionario de satisfacción de la interacción de usuario, desarrollado por Harper y Norman [60], tiene como propósito medir la satisfacción subjetiva del usuario en relación con la interfaz de la computadora. Este cuestionario busca obtener opiniones sobre áreas clave, tales como la facilidad de uso, coherencia, capacidad del sistema, entre otras.

Se optó por utilizar tanto SUS como QUIS en la evaluación de la usabilidad debido a sus características específicas. Por un lado, el uso de SUS proporciona una evaluación rápida y generalizada de la facilidad de uso percibida y la satisfacción general del usuario. Por otro lado, QUIS se empleó debido a su enfoque más extenso, que permite evaluar una variedad de cualidades del sistema de manera detallada. Esta combinación de herramientas brindó una visión amplia de la experiencia del usuario. Se eligió incluir la versión completa del SUS, y en cuanto al QUIS, se optó por la última versión, la número siete, obtenida directamente del sitio oficial de la Universidad de Maryland[1]. Con la finalidad de adaptar el cuestionario a las especificidades de nuestro contexto de estudio, se llevó a cabo la selección de preguntas y secciones del QUIS, que están alineadas con las características del prototipo. Las secciones y preguntas elegidas comprenden temas como: la experiencia previa del usuario, interfaces gráficas, información sobre el sistema y capacidad del sistema.

5.2 Escenario de pruebas

5.2.1 Usuarios de prueba

Para evaluar este primer prototipo, se seleccionó a una población objetivo que contara con experiencia en la navegación web y con individuos que no presentaran necesidades particulares. Lo anterior, con el propósito de capturar percepciones de usuarios que po-

[1] https://www.umventures.org/technologies/quis%E2%84%A2-questionnaire-user-interaction-satisfaction

seen conocimiento previo de prácticas convencionales de navegación web. Este enfoque sirve como punto de partida para futuras versiones del prototipo enfocadas a otro tipo de usuarios.

Se utilizó un muestreo por conveniencia [61] en la selección de los participantes para las pruebas. Es importante mencionar que esta técnica, al depender de la accesibilidad de elementos de investigación disponibles, puede introducir sesgos y no garantiza que la muestra sea representativa de toda la población objetivo. Pero, a pesar de que es una muestra pequeña, se encuentra dentro del promedio para este tipo de pruebas [62].

5.2.2 Configuración del escenario

Para las pruebas de evaluación, se configuró un entorno libre de distracciones acústicas, con acceso a servicios de internet mediante conexión inalámbrica. En este espacio, participaron exclusivamente el usuario de prueba y el aplicador. Los dispositivos esenciales para las pruebas incluyeron una computadora portátil para ejecutar el servidor *CoreNLP* y una tablet conectada al servicio de internet, con la aplicación (prototipo) instalada. Opcionalmente, se sugirió el uso de audífonos inalámbricos para mejorar la experiencia auditiva del usuario durante la interacción con la aplicación.

5.3 Procedimiento

Para llevar a cabo las evaluaciones, se siguieron los siguientes pasos.

5.3.1 Preparación del entorno

- Se aseguró de que la sala estuviera libre de distracciones acústicas y se confirmó la disponibilidad del servicio de internet inalámbrico.
- La computadora portátil se encendió, se verificó su conexión a la red local, se extrajo su dirección IP y se ejecutó el servidor *CoreNLP*.
- Se configuró en el código del prototipo la IP del servidor *CoreNLP*.

- La tablet fue conectada a la red local y se le instaló la aplicación.
- En el caso de utilizar audífonos, estos se conectaron a la tablet.

5.3.2 Tareas de evaluación

Durante las pruebas de evaluación, se asignaron diversas tareas al usuario, diseñadas para evaluar la usabilidad del prototipo. Estas tareas incluyeron:

- Configuración de perfil: con el objetivo de optimizar el proceso de evaluación, se predeterminaron siete perfiles. Cada uno con características distintivas. Esta medida permitió a los usuarios, si así lo preferían, seleccionar uno de estos perfiles en lugar de dedicar tiempo a completar su perfil. Se les solicitó a los usuarios que, al elegir un perfil predeterminado, actuaran como si tuvieran los gustos y características asociados con ese perfil. De esta manera, pudieron interactuar con la aplicación en función de las preferencias establecidas para cada perfil específico. Esta opción se presentó considerando el tiempo que podría requerir el llenado del perfil. No obstante, para aquellos usuarios que así lo deseaban, se permitió y alentó la opción de llenar su propio perfil.
- Búsqueda de una imagen.
- Interacción con la imagen encontrada: cambio de tamaño, movimiento en el plano y rotación.
- Búsqueda de un video.
- Interacción con el video encontrado: pausar, gestionar el volumen, repetir, avanzar o retroceder.
- Búsqueda de una página web
- Navegación en la página web: desplazamiento hacia arriba y hacia abajo e interacción con menús.

- Búsqueda de un término o definición.
- Selección de una opción entre las sugerencias proporcionadas.
- Ajuste del volumen de la aplicación.
- Solicitar la repetición de las frases pronunciadas por la aplicación.
- Solicitar la presentación de la siguiente opción en cualquier búsqueda realizada.
- Cerrar la aplicación.

5.3.3 Instrucciones

- Antes de iniciar las pruebas de evaluación, se brindó al usuario una breve presentación del proyecto de investigación. Durante esta introducción, se resaltaron los objetivos y detalles relacionados con el proceso de evaluación.
- También, el usuario recibió una breve demostración del funcionamiento del prototipo. Con el objetivo que el usuario comprendiera la manera en que la herramienta de navegación web por voz abordaría sus interacciones.
- Posteriormente, se entregó al usuario una hoja que contenía las tareas específicas a realizar durante la evaluación. Cada tarea estaba acompañada de frases de ejemplo que servían como guía para la interacción con el prototipo.
- Al concluir cada búsqueda o interacción, se cuestionaba al usuario acerca de la idoneidad de la respuesta obtenida, buscando obtener retroalimentación inmediata.
- Finalmente, se le pidió al usuario que, al concluir las tareas de evaluación, respondiera el cuestionario detallado en la Sección 5.1.

5.4 Resultados

El conjunto de prueba estuvo compuesto de 20 participantes, de los cuales 9 eran mujeres y 11 eran hombres. La edad promedio de los participantes fue de 26 años, con una edad máxima de 52 y una edad mínima de 16. En cuanto al nivel educativo de los participantes, se observó lo siguiente:

- El 15 % de los participantes indicaron haber concluido o estar cursando el nivel educativo básico.
- El 40 % manifestó encontrarse actualmente en el nivel medio superior o haberlo completado.
- El 10 % de los participantes indicó estar cursando o haber concluido el nivel superior.
- El resto de los participantes (el 35 %) informó estar cursando o haber completado un posgrado.

Estos datos se visualizan en la Figura 5.1.

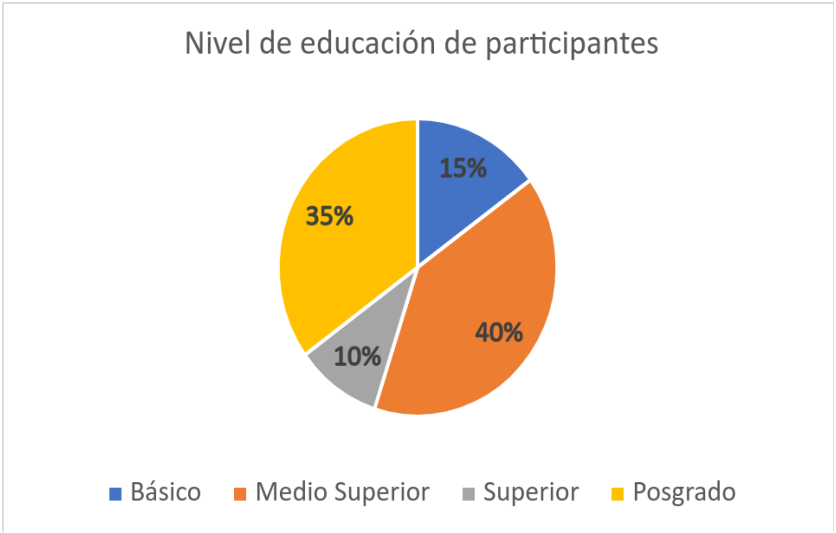


Figura 5.1: Nivel educativo de los participantes.

5.4.1 Evaluación del modelo usando QUIS 7

5.4.1.1 Experiencia previa

Con el objetivo de validar la experiencia de los usuarios con asistentes virtuales (AV) que operan mediante comandos de voz, se les consultó acerca de su historial de interacción con dichas tecnologías. Como se aprecia en la Figura 5.2, se obtuvieron las siguientes respuestas: dos usuarios indicaron no tener ninguna experiencia previa con asistentes virtuales, mientras que otros cuatro confirmaron haber utilizado al menos un AV. Adicionalmente, diez participantes expresaron haber interactuado con dos tipos diferentes de asistentes virtuales, y finalmente, cuatro usuarios afirmaron haber utilizado de tres a cuatro asistentes virtuales distintos. Además de indagar sobre su experiencia con asistentes virtuales, se consultó a los participantes acerca de las tecnologías con las que se sienten familiarizados y utilizan regularmente.

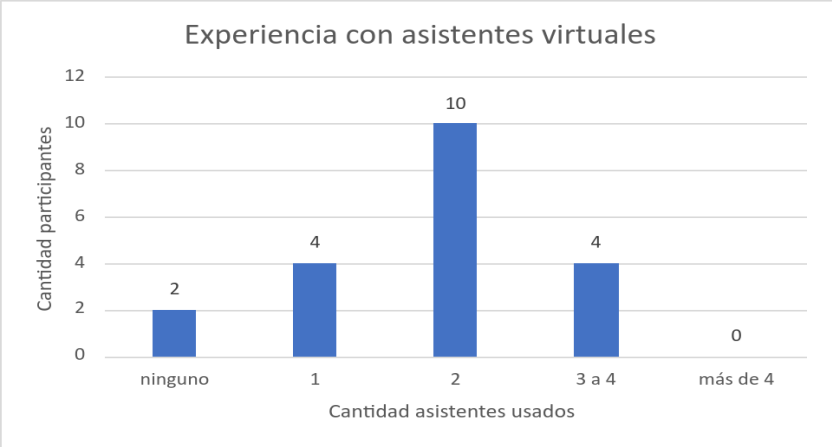


Figura 5.2: Experiencia con asistentes virtuales.

Los resultados indican que el 100 % de los usuarios afirmó utilizar dispositivos con pantalla táctil, acceder a Internet y gestionar correos electrónicos. En cuanto a otras tecnologías, el 90 % de los participantes expresaron estar familiarizados con el uso de monitores a color, el 85 % de ellos indicaron saber utilizar el teclado, el 80 % manifestó

conocer o haber utilizado un ratón, el 75 % afirmó tener habilidades en el manejo de una computadora personal, y el 60 % expresó haber utilizado sistemas de reconocimiento de voz. La muestra de usuarios exhibió una diversidad educativa intermedia, en promedio. Este perfil educativo, junto con el hecho de que el 90 % de los usuarios tienen experiencia utilizando al menos un asistente virtual y que el 100 % accede a internet, destaca que las percepciones y opiniones recopiladas provienen de individuos con sólido conocimiento y experiencia en este ámbito.

5.4.1.2 Evaluación de las interfaces gráficas

Se realizaron 11 preguntas destinadas a evaluar la usabilidad de las interfaces gráficas (IG). Los resultados en promedio de estas evaluaciones, se presentan en la Figura 5.3, donde *"IG"* representa la cualidad específica evaluada.

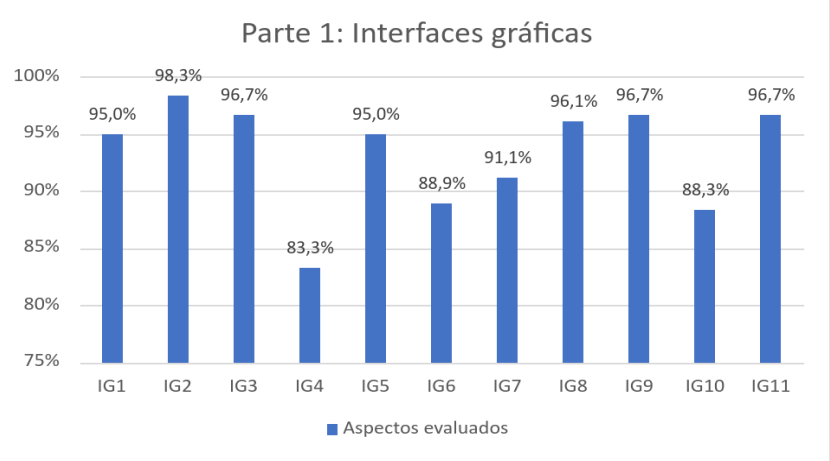


Figura 5.3: Interfaces gráficas.

En todas las evaluaciones realizadas, el porcentaje se interpreta en una escala que va desde 0 % hasta 100 %. En este contexto, un puntaje del 100 % representa la totalidad o la máxima calidad de la cualidad evaluada, como totalmente bueno, completamente nítido, absolutamente claro, entre otros. Por otro lado, un 0 % indicaría el extremo opuesto, es decir, la ausencia o la más baja calidad en relación con el aspecto evaluado. De esta

manera, los puntajes intermedios reflejan grados proporcionales de calidad en función de la escala establecida. A continuación, se proporciona la interpretación de estos resultados.

Principales cualidades de la IG

- **IG1**: facilidad de lectura de las letras en la pantalla: 95 %.
- **IG2**: nitidez de las imágenes: 98.3 %.
- **IG3**: legibilidad de las letras: 96.7 %.
- **IG4**: utilidad del uso de negritas en la aplicación: 83.3 %.
- **IG5**: utilidad del formato de las pantallas: 95.1 %.
- **IG6**: suficiencia en la cantidad de información en las pantallas: 88.9 %.
- **IG7**: lógica en la distribución espacial de la información: 91.1 %.
- **IG8**: claridad en la secuencia entre pantallas: 96.1 %.
- **IG9**: siguiente pantalla en una secuencia es predecible: 96.7 %.
- **IG10**: capacidad de volver a pantallas anteriores: 88.3 %.
- **IG11**: claridad de la secuencia de actividades en la navegación: 96.7 %.

A partir de la evaluación de los 11 aspectos, se obtuvo un porcentaje promedio del 93.3 %. Adicionalmente, algunos usuarios presentaron comentarios referentes a lo siguiente:

- Incluir versión de modo oscuro.
- Mantener un tamaño de letra uniforme en toda la aplicación, por ejemplo, en las páginas web que se visitan.

- Mejorar la distribución del contenido, debido a que: "*Algunas veces se amontonan las palabras*".
- Incluir un tutorial en la aplicación.
- Permitir cambiar el tamaño de las letras.
- Permitir el uso de la pantalla táctil en las pantallas.

La evaluación de las interfaces gráficas reveló, en general, una buena aceptación por parte de los usuarios. Entre los aspectos más destacados se encuentran la facilidad de lectura, la claridad en la secuencia de actividades de navegación, la utilidad del formato de las pantallas y la capacidad de predecir que pantalla es la siguiente en una secuencia. Además, estos resultados sugieren áreas de mejora, como la adaptación del tamaño de letra según las preferencias de cada usuario y la personalización de la interfaz en función de la cantidad de contenido. En general, las interfaces cumplen correctamente su función de presentar contenido en formato gráfico.
Sin embargo, se observó que los usuarios, al estar acostumbrados a utilizar aplicaciones que emplean hardware como dispositivos táctiles, expresaron algunas recomendaciones orientadas hacia mejoras que asemejan el modelo de navegación a la forma convencional. Es importante destacar que en este trabajo se busca implementar una navegación por voz para lograr una interacción natural, diferenciándose así de los modelos convencionales de interacción basados en hardware.

5.4.1.3 Evaluación de calidad de los resultados

Se realizaron diez preguntas con respecto a la información, notificaciones y mensajes auxiliares que muestra el prototipo. Los resultados en promedio de estas evaluaciones, se presentan en la Figura 5.4, donde *"IS"* representa la característica específica evaluada. De forma similar a la Sección anterior, los porcentajes se interpretan en una escala del 0 al 100 %. Donde: 100 % representa la máxima calidad del aspecto evaluado, mientras que

un 0 % indica la más baja calidad o ausencia del aspecto. A continuación, se proporciona la interpretación de estos resultados.

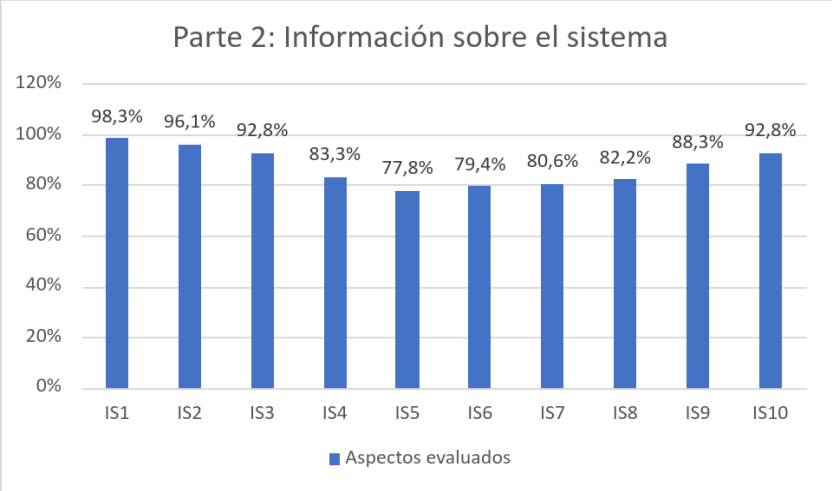


Figura 5.4: Información sobre el sistema.

- **IS1**: consistencia en los mensajes de las interfaces: 98.3 %.
- **IS2**: consistencia en la ubicación de los mensajes: 96.1 %.
- **IS3**: claridad de los mensajes: 92.8 %.
- **IS4**: claridad de las indicaciones que proporciona la aplicación: 83.3 %.
- **IS5**: consistencia en las indicaciones para corregir errores: 77.8 %.
- **IS6**: comunicación de las actividades que realiza la aplicación: 79.4 %.
- **IS7**: latencia aceptable entre operaciones: 80.6 %.
- **IS8**: utilidad de mensajes de error: 82.2 %.
- **IS9**: claridad de los mensajes de error: 88.3 %.
- **IS10**: consistencia en los mensajes de error: 92.8 %.

A partir de la evaluación de los 10 aspectos, se obtuvo un porcentaje promedio del 87.2 %. Se observó que, en términos generales, la información que se proporciona contribuye al correcto uso y funcionamiento del prototipo. Así mismo, indica una evaluación positiva de la usabilidad. Sin embargo, se identifica la necesidad de mejorar los siguientes aspectos:

- Claridad y simplicidad de las instrucciones verbales: una posible mejora podría ser la inclusión de ejemplos auditivos que faciliten la comprensión y ejecución de las tareas por parte de los usuarios.

- Claridad y consistencia en indicaciones para corregir errores: proporcionar instrucciones detalladas y precisas sobre cómo corregir el error.

- Comunicación de las acciones que realiza el prototipo: el prototipo sí cuenta con notificaciones que indican la actividad que realiza en determinado momento, por ejemplo:"*Estoy buscando imágenes que correspondan con tu petición*"; sin embargo, en ocasiones, estas notificaciones tardaron en reproducirse un tiempo considerable. Analizando este comportamiento, se determinó que ocurre en función de los siguientes factores:

  - Ruido ambiental: como se detalla en la sección 4.1.1.4, el sistema de reconocimiento de voz transmite datos de manera continua a la API de *speech to text*. En algunas ocasiones, la API puede demorar la entrega de resultados de transcripción hasta que el entorno se encuentre en un estado de silencio. Esto implica que, aunque el usuario haya realizado una solicitud, el prototipo puede tardar en responder hasta que detecte un período de silencio.

  - Conexión a internet: la demora en la respuesta de la API *speech to text*, a veces se debe a una conexión a Internet deficiente o con alta latencia. Esta latencia impacta directamente en el tiempo que el prototipo toma en generar

la notificación, ya que esta última es generada a partir de la transcripción de la solicitud del usuario.

El retraso generado en estos casos, provoca una incertidumbre para el usuario, ya que no queda claro si el prototipo ha captado o no su solicitud durante ese intervalo. Este inconveniente representa un área de mejora significativa.

Adicionalmente, algunos usuarios presentaron comentarios referentes a lo siguiente:

- Agregar una notificación que muestre el progreso de la tarea actual.
- Mostrar de forma gráfica los errores.
- Agregar indicaciones en pantalla para corregir errores.

Como se muestra en la lista anterior, los usuarios proporcionaron algunas recomendaciones relacionadas con la navegación web convencional. Es importante reiterar que, el objetivo principal de este trabajo es la implementación y uso de una interfaz de voz para lograr una interacción natural.

5.4.1.4 Evaluación de la capacidad del sistema

Para evaluar la capacidad del sistema, se aplicaron 3 preguntas a los usuarios. Los resultados en promedio de estas evaluaciones, se presentan en la Figura 5.3, donde *"CS"* representa el aspecto específico evaluado. A continuación, se proporciona la interpretación de estos resultados. De forma similar a las secciones anteriores, 100 % representa la totalidad del aspecto evaluado, por ejemplo, totalmente confiable. Por el contrario, 0 % representa la mínima puntuación, por ejemplo, nada confiable.

- **CS1**: confiabilidad de la aplicación: 94.4 %.
- **CS2**: facilidad de uso de acuerdo a la experiencia: 80 %.
- **CS3**: permite completar tareas sabiendo pocas instrucciones: 95 %.

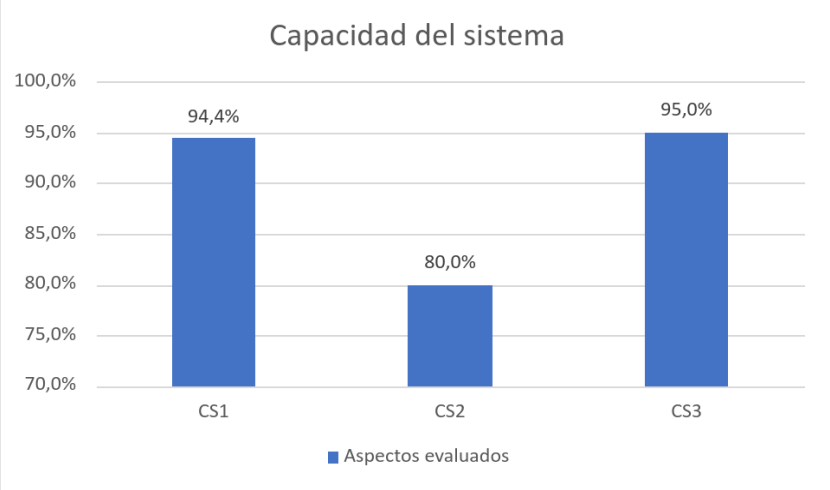


Figura 5.5: Capacidad del sistema.

Los usuarios percibieron el prototipo como bastante confiable para navegar en la web, ya que observaron que ejecutaba correctamente las peticiones que realizaban. Además, destacaron la facilidad de uso del prototipo, incluso con instrucciones mínimas de uso. Estos hallazgos sugieren que los usuarios pueden interactuar con el navegador de forma simple y natural. Aunque se recomienda mejorar la facilidad de uso y para ello se requiere una evaluación más detallada en cuanto a esta característica.

5.4.1.5 Impresión general del usuario

Se evaluó la impresión general del usuario mediante 3 preguntas. A continuación, se muestra el promedio de las calificaciones obtenidas.

1. Impresión general del prototipo:
    - 8.45, donde uno es muy mala y nueve es muy buena.
    - 8.4, donde uno es frustrante y nueve es agradable.
2. El uso del prototipo fue:
    - 8.55, donde uno es aburrido y nueve es estimulante.

- 8.6, donde uno es difícil y nueve es fácil.

3. La interacción con el prototipo fue:

   - 7.95, donde uno es rígida y nueve es flexible.
   - 8.15, donde uno es capacidad inadecuada y nueve es capacidad adecuada.

En esta sección, se obtuvo un puntaje promedio de 8.35, lo cual refleja una evaluación mayormente positiva en cuanto a la impresión que el prototipo generó en los usuarios. Las cualidades mejor evaluadas indican que el prototipo es fácil de usar, estimulante, altamente competente en las tareas que puede realizar y agradable.

5.4.2 Evaluación del modelo usando SUS

La escala de usabilidad del sistema, según Brooke [59], genera un único número (de 0 a 100) que representa una medida compuesta de la usabilidad general del sistema en cuestión. En la Figura 5.6, se presenta la puntuación de usabilidad por usuario. Al calcular el promedio de estas puntuaciones, se obtiene un valor de 76.87 puntos, señalando un nivel de usabilidad aceptable, aunque con áreas de mejora.

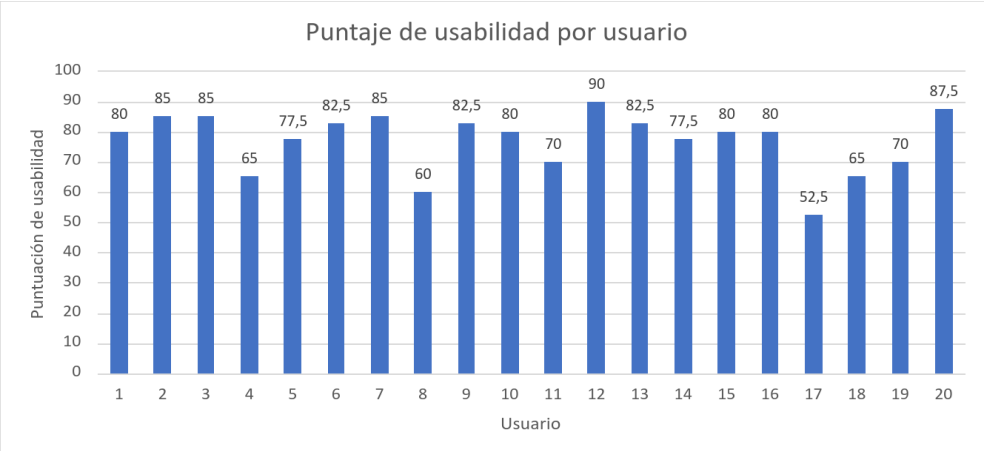


Figura 5.6: Puntuación de usabilidad por usuario.

De esta puntuación se puede concluir que el prototipo alcanzo un valor aceptable que

sugiere que es fácil de aprender, fácil de recordar cómo usar, congruente en su funcionamiento y proporciona una experiencia de usuario generalmente agradable.

5.4.3 Evaluación de los resultados de búsqueda

Se consultó a los usuarios, después de cada una de las 4 búsquedas realizadas, si los resultados obtenidos coincidían con su perfil de usuario y petición. Los resultados fueron los siguientes:

- Búsqueda de una imagen: según lo evidenciado en la Figura 5.7a, el 60 % afirmó que los resultados eran acordes. Un 20 % indicó que no coincidían, mientras que el restante 20 % expresó que más o menos correspondían.

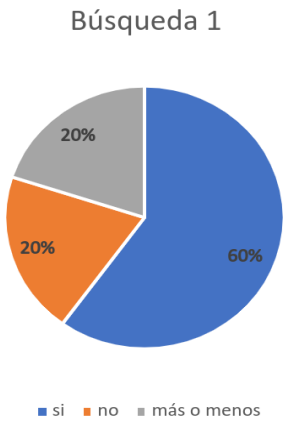
- Búsqueda de un video: conforme se observa en la Figura 5.7b, el 41 % aseguró que los resultados eran acordes. Un 36 % comentó que no correspondían, y el restante 23 % expresó que más o menos correspondían.

- Búsqueda de una página web: como se puede apreciar en la Figura 5.7c, el 75 % afirmó que los resultados eran acordes. Un 5 % comentó que no correspondían, mientras que el restante 20 % expresó que más o menos correspondían.

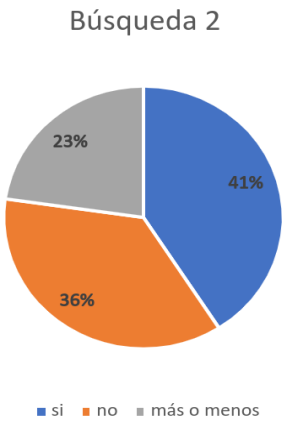
- Búsqueda de término: según se evidencia en la Figura 5.7d, el 70 % aseguró que los resultados eran acordes, y el 30 % comentó que más o menos correspondían.

En términos generales, y considerando las cuatro búsquedas realizadas, el 61.5 % de los usuarios afirmó que los resultados coincidían con sus perfiles y peticiones. Un 23.25 % expresó que los resultados más o menos correspondían, mientras que el 15.25 % indicó que los resultados no se ajustaban a sus perfiles y peticiones. Se concluye que el emparejamiento de contextos y la petición lograron que el 84.75 % de los usuarios obtuvieran contenido relevante en la primera petición. No obstante, se reconoce la oportunidad de mejora a través de la implementación de un método dinámico para el análisis y emparejamiento de

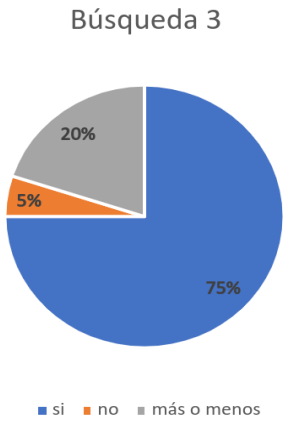
contextos.Esto posibilitará la expansión del contexto, permitiendo una mejor comprensión de las necesidades del usuario y las características de los recursos web. Asimismo, se contempla la integración de un motor de búsqueda dedicado que opere considerando el contexto del usuario.



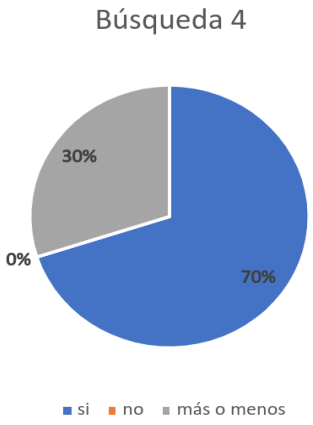
(a) Búsqueda de una imagen.



(b) Búsqueda de un video.



(c) Búsqueda de una página



(d) Búsqueda de un término

Figura 5.7: Resultados de búsqueda.

Conclusiones y trabajo a futuro

A pesar del aumento en el uso de las TICs en México, hay individuos que se encuentran excluidos de los beneficios y oportunidades que estas tecnologías ofrecen. La existencia de esta brecha digital no solo refleja una disparidad en el acceso a la información, sino que también contribuye a una marcada desigualdad social. Por esta razón, surge la necesidad de abordar la brecha digital.

La implementación de interfaces de voz en herramientas como asistentes virtuales, chatbots y lectores de pantalla juega un papel importante al simplificar la interacción y contribuir a combatir la brecha digital. A pesar de su contribución, estas soluciones presentan limitaciones significativas en la navegación web. Por ejemplo, algunos chatbots están diseñados exclusivamente para un sitio web o área específica, y los asistentes virtuales no pueden navegar en páginas web, lo que restringe su utilidad en esta tarea específica.

Es por esto que cobra relevancia una propuesta que transforme la experiencia digital para aquellos que enfrentan barreras significativas en la era de la información, tales como la falta de capacidades, aptitudes, habilidades, motivación, entre otras limitaciones. Este proyecto busca ayudar a disminuir la brecha digital, al permitir que el usuario tenga una interacción por voz y asegurar que las personas que no pueden navegar de manera convencional tengan acceso a información relevante que se ajuste a sus necesidades específicas. A continuación, se presentan las conclusiones derivadas de la investigación, destacando los resultados significativos y delineando posibles vías para futuras investigaciones.

Conclusiones

En este proyecto, se modeló una herramienta para la navegación web por voz. Este modelo utiliza el contexto de los recursos web, las peticiones y el contexto del usuario para navegar. El objetivo del modelo es ofrecer al usuario contenido que se ajuste con el perfil y las intenciones de búsqueda. Además, busca proveer una interacción simple y natural entre el usuario y la herramienta.

El diseño conceptual se basó en un estudio del estado del arte con el propósito de identificar los requisitos y funcionalidades asociados con la navegación web por voz, teniendo en cuenta el contexto de usuario y/o de los recursos web. Como resultado de este análisis, se identificaron seis funcionalidades principales, nueve requerimientos funcionales, tres requerimientos de usabilidad, ocho requerimientos de desempeño y siete requerimientos de bases de datos.

Con el objetivo de validar el modelo propuesto, se llevó a cabo la implementación de un prototipo, en forma de aplicación, diseñada para un dispositivo móvil (Tablet) con sistema operativo Android. Se concluye que el prototipo cumplió en general de manera positiva con la mayoría de los requerimientos, ofreciendo completamente cinco de las seis funcionalidades básicas de navegación propuestas. La única funcionalidad que no se cumplió por completo fue la capacidad de navegar dentro de una página web mediante comandos de voz, ya que el prototipo no puede interactuar con objetos que generen eventos como casillas de verificación o elementos multimedia incrustados, debido a limitaciones de tiempo por problemas encontrados en el desarrollo. Sin embargo, se logró implementar la navegación entre secciones de una página web y el desplazamiento vertical en la página.

Las evaluaciones de este primer prototipo, realizadas con individuos que poseen conocimientos intermedios en cuanto a la navegación web convencional y el uso de tecnologías con interfaz de voz, validaron el modelo propuesto al conseguir evaluaciones mayormente positivas en cuanto a las métricas de usabilidad evaluadas. Se reconoce que existen numerosas áreas de mejora para implementar en futuros prototipos. Además, durante este

proceso, se identificaron características clave que se detallan como trabajo futuro, con la esperanza de contribuir a optimizar el emparejamiento entre las intenciones, contexto del usuario y el contenido encontrado en la web a través de la implementación de un motor de búsqueda dedicado, así como realizar mejoras en la interfaz de voz.

También, las pruebas realizadas revelaron que los usuarios mantienen una fuerte conexión con los métodos de navegación convencionales que implican el uso de hardware, como el ratón, el teclado y la pantalla táctil, para interactuar con los dispositivos. Este sesgo se reflejó claramente en sus comentarios y sugerencias durante la evaluación del prototipo. A pesar de esta preferencia arraigada, es destacable que el prototipo fue bien recibido por los usuarios, lo que sugiere una apertura y adaptabilidad positiva hacia nuevas formas de interacción, como la navegación por voz.

Es importante mencionar que este modelo de navegación por voz, presenta diferencias significativas en comparación con herramientas que emplean interfaces de voz, como los asistentes virtuales. Mientras que los asistentes virtuales se centran en proporcionar una amplia gama de servicios y funciones, desde realizar tareas cotidianas como establecer recordatorios, enviar mensajes, hasta interactuar con dispositivos inteligentes en el hogar mediante comandos preestablecidos, la propuesta de este trabajo tiene un enfoque distinto.

Este modelo busca proporcionar contenido de interés genuino para el usuario, en formato audible y gráfico, mediante el análisis activo de las páginas web para su emparejamiento con el contexto de usuario y la petición. Adicionalmente, la propuesta incluye la capacidad de navegar por los resultados y las páginas web mediante comandos flexibles de voz. Esta característica facilita la accesibilidad para aquellos que no están familiarizados con las tecnologías de la información actuales, lo que contribuye a la disminución de la brecha digital. El primer prototipo validó los requerimientos y funcionalidades para la navegación por voz, sentando las bases para futuros prototipos que aspiran a hacer una contribución significativa a la reducción de la brecha digital.

Conclusiones respecto al desarrollo

Durante el desarrollo, se enfrentaron diversos desafíos, entre los cuales se incluyeron la curva de aprendizaje asociada al desarrollo de aplicaciones, configuración de entornos de desarrollo, el análisis de políticas de desarrollo y seguridad, entre otros. También, se identificaron problemas de compatibilidad entre versiones de bibliotecas que proveían de funcionalidades al prototipo, así que se buscaron alternativas para hacer compatibles y lograr su coexistencia. Al solucionar estos contratiempos se resaltó la importancia del control de versiones. En el transcurso de este proyecto, nos enfrentamos a un desafío al actualizar la herramienta de construcción Gradle. Esta actualización conllevó cambios significativos en la estructura del proyecto, en las dependencias y otros aspectos que resultaron ser irreversibles en este contexto específico. Intentar regresar a una versión anterior no fue suficiente para restaurar el comportamiento original de nuestro prototipo. Para abordar estos problemas, fue necesario depurar cada error identificado, lo que consumió tiempo de desarrollo.

De esta experiencia, se destaca la importancia de seguir buenas prácticas de desarrollo, tales como:

- Asegurarse de utilizar una versión compatible de Gradle y configurar el proyecto de manera apropiada para esa versión.
- Evitar actualizaciones innecesarias y realizarlas solo cuando sea necesario.
- Revisar las notas de versión de Gradle y cualquier documentación relacionada para asegurar que no haya cambios significativos que afecten la compatibilidad con versiones anteriores.
- Verificar las dependencias y asegurarse de que estén configuradas correctamente.

- En caso de necesitar una actualización, llevar a cabo el proceso con precaución para garantizar la compatibilidad entre los diversos módulos integrados en la aplicación.

Estas prácticas ayudan a evitar problemas al trabajar con herramientas y tecnologías en constante evolución. Aprender de esta experiencia refuerza la importancia de la configuración de entornos de desarrollo, la planificación y la gestión cuidadosa de las actualizaciones para mantener la estabilidad y consistencia del proyecto. En relación con la integración de servicios externos en el prototipo, se subraya la necesidad de adquirir conocimientos y profundizar en la comprensión del funcionamiento de cada servicio para lograr una integración efectiva en la aplicación. También se reconoce la opción de explorar otras herramientas que pudieran ofrecer el comportamiento deseado o en caso de dificultades significativas en la integración.

Limitaciones

A lo largo de esta investigación, se encontraron las siguientes limitaciones

- Las políticas de seguridad de Android en relación con el uso del micrófono y la grabación han presentado desafíos significativos en la implementación de un servicio de reconocimiento de voz continuo. Por un lado, el servicio de reconocimiento de voz nativo de Android limita la duración de las tareas de reconocimiento, dificultando su aplicación continua. Además, el uso continuo de este servicio afecta el rendimiento de la interfaz multimodal de esta propuesta, generando problemas como el bloqueo de la interfaz de usuario, como se reportó en la etapa de implementación. Por otro lado, Android requiere que la activación del micrófono se realice en primer plano. Esta tarea afecta la presentación de contenido. Este efecto es más notorio cuando se trata de audio o video, ya que se pausa cada vez que el micrófono se activa. Esta limitación impide la implementación de un ciclo donde se encienda y apague el micrófono para simular un reconocimiento de voz continuo.

- Las medidas específicas de seguridad implementadas por las páginas y sitios web

incluyen la utilización de archivos `robots.txt`, que indican a los rastreadores web qué áreas del sitio no deben ser exploradas, y la ejecución de scripts mediante *JavaScript* dinámico. Además, algunas plataformas web implementan técnicas avanzadas para protegerse contra la extracción ilegal de datos, como el uso de desafíos CAPTCHA para verificar la autenticidad de los usuarios y limitar la automatización no autorizada. Estas estrategias buscan salvaguardar la integridad y la disponibilidad de los recursos web, pero al mismo tiempo, presentan desafíos significativos al intentar acceder a su contenido mediante métodos de análisis estático como el utilizado en este prototipo. Por ejemplo, en pruebas realizadas en el desarrollo del prototipo, la IP del dispositivo fue bloqueada por un periodo por el sitio de YouTube y Amazon.

Esta limitación impactó directamente en el análisis del contexto de las páginas web y, consecuentemente, en la presentación de contenido adaptado al contexto del usuario. La incapacidad para acceder a ciertos contenidos desde algunas páginas web restringió la capacidad del prototipo para proporcionar información contextual específica. Esto destaca la importancia de considerar las restricciones de seguridad y las políticas de acceso al contenido al desarrollar sistemas de navegación web por voz que utilizan análisis estático.

- La herramienta de código abierto *Stanford CoreNLP* mostró limitaciones en cuanto a su optimización para dispositivos móviles o aquellos con recursos limitados. Para superar este desafío, se optó por implementar un servidor remoto, lo que resultó en una dependencia de otro dispositivo y la introducción de latencia en el proceso.

Trabajo a futuro

En el desarrollo y la evaluación de esta primera implementación, se revelaron varios aspectos a mejorar.

1. Mejorar la claridad de los mensajes de información que se le presentan al usuario.

2. Desarrollar una nueva estrategia para el reconocimiento de voz continuo. Específicamente, se busca optimizar el proceso de obtención de la frase del usuario, incorporando mecanismos para discriminar el ruido ambiental. En caso de no ser posible eliminar completamente el ruido, se plantea establecer un tiempo determinado para obtener una solicitud por voz y realizar este proceso de manera cíclica. Este enfoque tiene como objetivo mitigar la latencia experimentada en la primera implementación del prototipo.

3. Encontrar, optimizar o desarrollar herramientas que posibiliten que el modelo pueda operar de manera independiente sin depender de otros dispositivos, dado que en la configuración actual, existe una dependencia del dispositivo que ejecuta el servidor *CoreNLP*.

4. Si se opta por continuar utilizando servidores externos, se sugiere adquirir un certificado de seguridad para asegurar una comunicación segura, especialmente al manejar información sensible o personal en la aplicación.

5. Se plantea la posibilidad de integrar el uso de *transformers* para generar respuestas en lenguaje natural a partir del texto encontrado en las páginas web. Esta mejora permitiría evitar la lectura extensiva del contenido y proporcionar a los usuarios información precisa y específica relacionada con sus búsquedas.

6. Se considera la implementación de más prototipos mediante el uso de una metodología de diseño basada en el usuario, lo que implicaría continuar evaluando y mejorando el modelo en función de las necesidades y retroalimentación directa de

los usuarios. Este enfoque garantizaría una mayor adaptabilidad y eficacia del sistema en entornos del mundo real, considerando la diversidad de usuarios y sus características particulares.

7. Abordar las limitaciones relacionadas con las medidas específicas de seguridad implementadas por las páginas web, se propone investigar y desarrollar estrategias más sofisticadas de manejo y elusión de obstáculos. Esto podría incluir la implementación de técnicas avanzadas para manejar desafíos CAPTCHA, optimizar el manejo de archivos `robots.txt` y scripts dinámicos. Se sugiere examinar enfoques que respeten las políticas de seguridad de los sitios web mientras se busca acceder a contenido relevante para la navegación por voz. Este trabajo futuro se orientaría hacia la mejora de la capacidad del prototipo para adaptarse y recuperar información contextual específica a pesar de las restricciones de seguridad implementadas por algunas páginas web.

8. Se propone la exploración y desarrollo de un motor de búsqueda personalizado o la implementación de mecanismos de rastreo, indexación y actualización dinámica. Este enfoque se orientaría a lograr una mayor autonomía en la obtención de información relevante para el modelo de navegación por voz basado en contexto. La idea es diseñar un sistema que se adapte específicamente a los objetivos y requisitos del modelo, permitiendo una experiencia de navegación más eficiente y personalizada para los usuarios

9. Se sugiere la expansión del módulo analizador de contexto de páginas web. Actualmente, el sistema realiza un análisis sintáctico del texto presente en las páginas web y, en aquellas que lo permiten, efectúa un análisis estático de HTML, scripts de Java y CSS. La propuesta consiste en incorporar un análisis semántico y dinámico de la estructura y contenido de la página web. Este enfoque permitirá una comprensión más profunda y contextualizada del contenido, mejorando así la capacidad del

sistema para adaptarse a las necesidades y preferencias de los usuarios.

10. Se propone realizar un análisis comparativo para verificar si el modelo proporciona, y en qué medida, resultados más relevantes o mejor alineados con el contexto del usuario y su petición en comparación con modelos de navegación convencionales.

11. Se propone incorporar funcionalidades de navegación adicionales en las páginas web, lo cual está directamente relacionado con el punto nueve, que aborda el análisis del contexto de las páginas web. Esta ampliación en las capacidades de navegación contribuirá a mejorar la experiencia del usuario al proporcionar opciones más completas y específicas en la interacción con el contenido web.

12. Se propone implementar el módulo de reconocimiento de voz (*speech to text*) y síntesis de voz (*text to speech)* mediante el uso de herramientas de software libre. Actualmente, se emplean soluciones de Google, las cuales generan un costo mensual. La adopción de herramientas de software libre busca reducir los costos asociados con estas funcionalidades, promoviendo una solución más sostenible y accesible económicamente.

Anexos

A Códigos de la Sección 4.1

A.1 Método startRecognition con speech-to-text Google

```
1 fun startRecognition(audioData: ShortArray) {
2  val credentialsStream:InputStream? = this::class.java.
   getResourceAsStream(resources.openRawResource(R.raw.
   credenciales.json))
3  credentialsStream?.use {
4       val credentials = GoogleCredentials.fromStream(it)
5       SpeechClient.create().use { speechClient ->
6           val recognitionConfig = RecognitionConfig.newBuilder()
7               .setEncoding(RecognitionConfig.AudioEncoding.
   LINEAR16)
8               .setSampleRateHertz(SAMPLE_RATE)
9               .setLanguageCode("en-US")
10              .build()
11          val recognitionAudio = RecognitionAudio.newBuilder()
12              .setContent(ByteString.copyFrom(
   convertShortArrayToByteArray(audioData)))
13              .build()
14          val response = speechClient.recognize(
   recognitionConfig, recognitionAudio)
15          val results:List<SpeechRecognitionResult> = response.
   resultsList
16          if (results.isNotEmpty()) {
17              val alternative:SpeechRecognitionAlternative =
   results[0].alternativesList[0]
```

```
18              val recognizedText:String = alternative.transcript
19              notifyListeners(recognizedText)
20          }
21      }
22  }
23}
```

A.2 Servicio AudioCaptureService

```
class AudioCaptureService : Service() {
 private val observers: MutableList<SpeechRecognitionObserver>
    = mutableListOf()
 private lateinit var audioRecord: AudioRecord
 private var isRecording = false
 private lateinit var speechRecognizer: SpeechRecognizer
 companion object {
  private const val TAG = "AudioCaptureService"
        private const val SAMPLE_RATE = 16000
        private const val CHANNEL_CONFIG = AudioFormat.
           CHANNEL_IN_MONO
        private const val AUDIO_FORMAT = AudioFormat.
           ENCODING_PCM_16BIT
        private var instance: AudioCaptureService? = null
        fun getInstance(): AudioCaptureService? {
                return instance
            }
        }
    override fun onCreate() {
        super.onCreate()
        instance = this
    }
    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    override fun onStartCommand(intent: Intent?, flags: Int,
       startId: Int): Int {
```

```
        startAudioCapture()
        return START_STICKY
    }
    override fun onDestroy() {
        stopAudioCapture()
        super.onDestroy()
        instance = this
    }

    @SuppressLint("MissingPermission")
    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    private fun startAudioCapture() {
        isRecording = true

        // Calcular el tama\~no del b\'ufer de audio
        val bufferSize = AudioRecord.getMinBufferSize(
            SAMPLE_RATE,
            CHANNEL_CONFIG,
            AUDIO_FORMAT
        )
        val buffer = ShortArray(bufferSize)

        // Inicializar el objeto AudioRecord para capturar audio
        audioRecord = AudioRecord(
            MediaRecorder.AudioSource.MIC,
            SAMPLE_RATE,
            CHANNEL_CONFIG,
            AUDIO_FORMAT,
            bufferSize
        )

        // Iniciar la captura de audio en un hilo separado
        Thread {
            audioRecord.startRecording()
```

```
            processCapturedAudio(buffer, bufferSize)
        }.start()
    }

    private fun stopAudioCapture() {
        isRecording = false
        audioRecord.stop()
        audioRecord.release()
    }
    private fun processCapturedAudio(buffer: ShortArray,
       bufferSize: Int) {
        try {
            // Crear las credenciales de Google Cloud a partir
               del archivo JSON de las credenciales
            val credentials = GoogleCredentials.fromStream(
               resources.openRawResource(R.raw.speechtotext))

            // Crear el cliente de Speech-to-Text de Google
               Cloud
            val speechClientSettings = SpeechSettings.newBuilder
               ()
                .setCredentialsProvider(FixedCredentialsProvider
                   .create(credentials))
                .build()
            val speechClient = SpeechClient.create(
               speechClientSettings)

            // Configurar el reconocimiento de voz
            val recognitionConfig = RecognitionConfig.newBuilder
               ()
                .setEncoding(RecognitionConfig.AudioEncoding.
                   LINEAR16)
                .setSampleRateHertz(SAMPLE_RATE)
                .setLanguageCode("es-MX")
```

```
            .build()

        // Crear una instancia de ApiStreamObserver para
           recibir los resultados del reconocimiento
        val responseObserver = object : ApiStreamObserver<
           StreamingRecognizeResponse> {
            override fun onNext(response:
               StreamingRecognizeResponse) {
                // Procesar los resultados del
                   reconocimiento de voz
                for (result in response.resultsList) {
                    for (alternative in result.
                       alternativesList) {
                        val transcript = alternative.
                           transcript
                        //Log.d(TAG, "Resultado del
                           reconocimiento de voz:
                           $transcript")
                        notifySpeechRecognitionResult(
                           transcript)
                        Log.d(TAG, "Envi el resultado!")
                    }
                }
            }

            override fun onError(throwable: Throwable) {
                val errorMessage = throwable.message ?: "
                   Error en el reconocimiento de voz"
                notifySpeechRecognitionError(errorMessage)
            }

            private fun notifySpeechRecognitionError(
               errorMessage: String) {
                for (observer in observers) {
```

```
                observer.onSpeechRecognitionError(
                    errorMessage)
            }
        }
    }

    // Crear el stream de reconocimiento de voz
    val requestObserver = speechClient.
        streamingRecognizeCallable().bidiStreamingCall(
        responseObserver)

    // Enviar los datos de audio al stream de
        reconocimiento de voz
    val streamingConfig = StreamingRecognitionConfig.
        newBuilder()
        .setConfig(recognitionConfig)
        .build()
    val streamingRequest = StreamingRecognizeRequest.
        newBuilder()
        .setStreamingConfig(streamingConfig)
        .build()
    requestObserver.onNext(streamingRequest)

    // Procesar el audio capturado en tiempo real
    while (isRecording) {
        val bytesRead = audioRecord.read(buffer, 0,
            bufferSize)
        if (bytesRead != AudioRecord.
            ERROR_INVALID_OPERATION && bytesRead !=
            AudioRecord.ERROR_BAD_VALUE) {
            val audioData = ByteString.copyFrom(
                convertShortArrayToByteArray(buffer,
                bytesRead), 0, bytesRead * 2)
            val streamingContent =
```

```
                    StreamingRecognizeRequest.newBuilder()
                        .setAudioContent(audioData)
                        .build()
                    requestObserver.onNext(streamingContent)
                }
            }

            // Finalizar el stream de reconocimiento de voz
            requestObserver.onCompleted()

            // Cerrar el cliente de Speech-to-Text
            speechClient.close()
        } catch (e: Exception) {
            Log.e(TAG, "Error en el reconocimiento de voz: ${e.
                message}")
        }
    }

    private fun convertShortArrayToByteArray(shortArray:
        ShortArray, size: Int): ByteArray {
        val byteArray = ByteArray(size * 2)
        for (i in 0 until size) {
            val shortValue = shortArray[i]
            byteArray[i * 2] = (shortValue and 0xFF).toByte()
            byteArray[i * 2 + 1] = (shortValue.toInt() shr 8 and
                0xFF).toByte()
        }
        return byteArray
    }

    private fun createSpeechClient(): SpeechClient {
        val credentials = GoogleCredentials.fromStream(resources
            .openRawResource(R.raw.speechtotext))
        val speechClientSettings = SpeechSettings.newBuilder()
```

```
            .setCredentialsProvider { credentials }
            .build()
        return SpeechClient.create(speechClientSettings)
    }

    private fun processRecognitionResults(results: List<
        SpeechRecognitionResult>) {
        // Procesar los resultados del reconocimiento de voz
        for (result in results) {
            val alternatives = result.alternativesList.toList()
            for (alternative in alternatives) {
                val transcript = alternative.transcript
                Log.d(TAG, "Resultado del reconocimiento de voz:
                    $transcript")
            }
        }
    }

    fun registerObserver(observer: SpeechRecognitionObserver) {
        observers.add(observer)
    }

    fun unregisterObserver(observer: SpeechRecognitionObserver)
        {
        observers.remove(observer)
    }

    private fun notifySpeechRecognitionResult(transcript: String
        ) {
        Log.d(TAG, "Resultado del reconocimiento de voz NM!: $transcript") // Agregar este log
        Log.d(TAG, "Observadores registrados: ${observers.size}") // Agregar este log
        for (observer in observers) {
```

```
                observer.onSpeechRecognitionResult(transcript)
            }
        }

        inner class LocalBinder : Binder() {
            fun getService(): AudioCaptureService =
                this@AudioCaptureService
        }
        private val binder = LocalBinder()
        override fun onBind(intent: Intent?): IBinder? {
            return binder
        }
}
```

B Cuestionario empleado para la creación del perfil de usuario

En la primera versión del prototipo basado en el modelo propuesto, se emplearon las siguientes preguntas para la recolección de datos y la creación del perfil de usuario.

1. ¿Cuántos años tienes?
2. ¿Cuál es tu género?
3. ¿Cuál es tu dirección?
4. Menciona tus intereses principales. Puedes decir palabras clave como música, deportes, tecnología, cocina, viajes, o cualquier otro interés que tenga.
5. ¿Cuáles son tus pasatiempos o actividades recreativas favoritas?
6. ¿Cuál es tu estilo de música favorito?
7. ¿Tienes algún género de películas que te guste especialmente?
8. ¿Qué géneros de libros prefieres?
9. ¿Prácticas algún deporte?, ¿Cuál?
10. ¿Hay un tipo de comida que te guste más que otros?, ¿Cuál es?
11. Por favor, menciona los sitios web o páginas que visites con frecuencia:
12. ¿Cuál es el nivel más alto de educación que has completado? Básico, Medio Superior, Superior o Posgrado.

Bibliografía

[1] StatCounter Global Stats. Cuota de mercado de sistemas operativos para móviles y tabletas en méxico. `https://gs.statcounter.com/`, Junio 2023. Consultado en Julio de 2023.

[2] Ricardo Pérez Zúñiga, Paola Mercado Lozano, Mario Martínez García, Ernesto Mena Hernández, and José Ángel Partida Ibarra. La sociedad del conocimiento y la sociedad de la información como la piedra angular en la innovación tecnológica educativa. *RIDE. Revista Iberoamericana para la Investigación y el Desarrollo Educativo*, 8(16):847–870, 2018.

[3] Manuel Castells and Paul Chemla. La galaxia internet.

[4] INEGI. Encuesta nacional sobre disponibilidad y uso de tecnologías de la información en los hogares (endutih) 2022, Junio 2023. Consultado en Junio de 2023.

[5] Jan A. G. M. Van Dijk. Digital divide: Impact of access. In *The International Encyclopedia of Media Effects*, pages 1–11. John Wiley y Sons, Chichester, UK, primera edition, Marzo 2017. doi: https://doi.org/10.1002/9781118783764.wbieme0043.

[6] INEGI. *Comunicado de prensa núm. 258/22: Estadísticas a propósito del día mundial del Internet (17 de mayo)*, Mayo 2022.

[7] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, Julio 1999.

[8] Amazon®. Amazon alexa. https://developer.amazon.com/es-ES/alexa, Septiembre 2022. Consultado en Septiembre de 2022.

[9] GooglePlay®. Asistente de google. https://play.google.com/store/apps/details?id=com.google.android.apps.googleassistant, Septiembre 2022. Consultado en Septiembre de 2022.

[10] Microsoft®. Microsoft cortana. https://support.microsoft.com/es-es/topic/-qu\%C3\%-es-cortana-953e648d-5668-e017-1341-7f26f7d0f825, 2020. Consultado en Octubre de 2022.

[11] Apple®. Siri. https://www.apple.com/mx/siri/, Octubre 2022. Consultado en Octubre de 2022.

[12] Freedom Scientific. Jaws®. https://www.freedomscientific.com/products/software/jaws/, Septiembre 2022. Consultado en Septiembre de 2022.

[13] Apple-Support. Introducción a voiceover. https://www.apple.com/es/voiceover/info/guide/1121.html, Septiembre 2022. Consultado en Septiembre de 2022.

[14] Aurelio De Rosa and Donovan Justice. Web reader: A screen reader for everyone, everywhere. In *Proceedings of the 13th International Web for All Conference*, W4A '16, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Vikas Ashok, Yury Puzis, Yevgen Borodin, and I.V. Ramakrishnan. Web screen reading automation assistance using semantic abstraction. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, IUI '17, page 407 a 418, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Yevgen Borodin, Faisal Ahmed, Muhammad Islam, Yury Puzis, Valentyn Melnyk, Song Feng, Iv Ramakrishnan, and Glenn Dausch. Hearsay: A new generation context-driven multi-modal assistive web browser. pages 1233–1236, 01 2010.

[17] Hae-Na Lee, Sami Uddin, and Vikas Ashok. itoc: Enabling efficient non-visual interaction with long web documents. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3799–3806, 2020.

[18] Ali Selman Aydin, Shirin Feiz, Vikas Ashok, and IV Ramakrishnan. Sail: Saliency-driven injection of aria landmarks. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI '20, pages 111–115, New York, USA, 2020. Association for Computing Machinery.

[19] Anind K. Dey and Gregory Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC), Karlsruhe, Germany*, pages 304–307, 1999.

[20] Mark Weiser. The future of ubiquitous computing on campus. *Commun. ACM*, 41(1):41–42, Enero 1998.

[21] Menno Lindwer, Diana Marculescu, Twan Basten, Rainer Zimmermann, Radu Marculescu, Stefan Jung, and Eugenio Cantatore. Ambient intelligence visions and achievements: Linking abstract ideas to real-world concepts. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, page 10010, USA, 2003. IEEE Computer Society.

[22] N.M. Garcia and J.J.P.C. Rodrigues. *Ambient Assisted Living*, pages 7–8. Rehabilitation Science in Practice Series. CRC Press, 2015.

[23] José Manuel Romero Chávez. Diseño e implementación de mecanismos de búsqueda contextualizada y anotado a través de la web semántica. Tesis de maestría, Centro de

Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Septiembre 2012.

[24] Viridiana Ponce Angulo. Activación multimodal de servicios en dispositivos m óviles. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Septiembre 2012.

[25] Ricardo Pérez Zúñiga, Paola Mercado Lozano, Mario Martínez García, Ernesto Mena Hernández, and José Ángel Partida Ibarra. La sociedad del conocimiento y la sociedad de la información como la piedra angular en la innovación tecnológica educativa. *RIDE Revista Iberoamericana para la Investigación y el Desarrollo Educativo*, 2018.

[26] Tim Berners-Lee and R. Cailliau. Worldwideweb: propuesta de proyecto de hipertexto. `https://www.w3.org/Proposal.html`, Noviembre 1990. Consultado en octubre de 2022.

[27] Debashis Saha and Amitava Mukherjee. Mukherjee, a.: Pervasive computing: A paradigm for the 21st century. computer 36(3), 25-31. *Computer*, 36:25–31, 04 2003.

[28] Nicol González Ávila, Ignacio López Martínez, and Noe Hernández García. Aproximación al análisis de benchmark sobre asistentes virtuales. *Interconectando Saberes*, (9), Marzo 2020.

[29] José M. Rubio, Tanya Neira-Peña, Danilo Molina, and Cristian Vidal-Silva. Proyecto ubot: asistente virtual para entornos virtuales de aprendizaje. *Información Tecnológica*, 33(4):85 – 92, 2022.

[30] Harley Vera O., Ana Rocío Cárdenas M., Meluni Daney Palomino F., Jonathan Ricardo Vasquez C., Rosa Virginia Encinas Q., GGrover Enrique Castro G., Yanina Leon U., Jhony Lucia Huallparimachi, Luis Antonio Quispe C., and Lauro Enci-

so. Saminbot: un asistente virtual para recolectar datos durante la pandemia del covid-19. *Interfases*, (014):138 – 162, 2021.

[31] Cataldo Musto, Fedelucio Narducci, Marco Polignano, Marco De Gemmis, Pasquale Lops, and Giovanni Semeraro. Myrrorbot: A digital assistant based on holistic user models for personalized access to online services. *ACM Trans. Inf. Syst.*, 39(4), aug 2021.

[32] Gaurav Shekhar, Rhea DSouza, and Kevin Fernandes. Ai-driven contextual virtual teaching assistant using rasa. In *Proceedings of the 21st Annual Conference on Information Technology Education*, SIGITE '20, page 346, New York, NY, USA, 2020. Association for Computing Machinery.

[33] Angel Ivanov and Daniela Orozova. Virtual intelligent personal assistant for bat researchers. In *Proceedings of the 19th International Conference on Computer Systems and Technologies*, CompSysTech́18, pages 38–41, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Hung H. Bui, Federico Cesari, Daniel Elenius, David N. Morley, Sriraam Natarajan, Shahin Saadati, Eric Yeh, and Neil Yorke-Smith. A context-aware personal desktop assistant. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers*, AAMAS '08, pages 1679–1680, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[35] Vinh Thinh Ho, Koninika Pal, Simon Razniewski, Klaus Berberich, and Gerhard Weikum. Extracting contextualized quantity facts from web tables. In *Proceedings of the Web Conference 2021*, WWW '21, pages 4033–4042, New York, NY, USA, 2021. Association for Computing Machinery.

[36] Yury Ustinovskiy and Pavel Serdyukov. Personalization of web-search using short-term browsing context. 10 2013.

[37] Jiang Bian. *Contextualized Web Search: Query-Dependent Ranking and Social Media Search*. PhD thesis, USA, 2010. AAI3451218.

[38] Limin Zhang. *Contextual Web Search Based on Semantic Relationships: A Theoretical Framework, Evaluation and a Medical Application Prototype*. PhD thesis, USA, 2006. AAI3213284.

[39] Cosmin Munteanu and Gerald Penn. Speech and hands-free interaction: Myths, challenges, and opportunities. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, pages 1–4, New York, NY, USA, 2018. Association for Computing Machinery.

[40] Javedul Ferdous, Hae-Na Lee, Sampath Jayarathna, and Vikas Ashok. Insupport: Proxy interface for enabling efficient non-visual interaction with web data records. In *27th International Conference on Intelligent User Interfaces*, IUI '22, pages 49–62, New York, NY, USA, 2022. Association for Computing Machinery.

[41] Ahmad C. Bukhari, Abid Ali Fareedi, and Yong Gi. Kim. Ho2iev: Heavyweight ontology based web information extraction technique for visionless users. In *The 7th International Conference on Networked Computing and Advanced Information Management*, pages 90–95, 2011.

[42] World-Health-Organization. Occupational and community noise. "https://www2.d125.org/physical_welfare/health/pdf/Environmental\%20health/WHO_noise.pdf", Febrero 2021. Consultado en Febrero de 2023.

[43] P.N. Tan, M. Steinbach, A. Karpatne, and V. Kumar. *Introduction to Data Mining*. What's New in Computer Science Series. Pearson, 2019.

[44] L. Boonstra. *The Definitive Guide to Conversational AI with Dialogflow and Google Cloud: Build Advanced Enterprise Chatbots, Voice, and Telephony Agents on Google Cloud*. Apress, 2021.

[45] Google Cloud. Cloud text-to-speech. `https://cloud.google.com/text-to-speech?hl=es`, Julio 2023. Consultado en Julio de 2023.

[46] Amazon Web Services. Software de texto a voz-amazon polly. `https://aws.amazon.com/`, Julio 2023. Consultado en Julio de 2023.

[47] IBM. Watson text to speech. `https://www.ibm.com/mx-es/cloud/watson-text-to-speech`, Julio 2023. Consultado en Julio de 2023.

[48] Developers Android. Documentación: Text to speech. `https://developer.android.com/reference/android/speech/tts/TextToSpeech`, Julio 2023. Consultado en Julio de 2023.

[49] Developers Android. Documentación: Speech recognizer'. `https://developer.android.com/reference/android/speech/SpeechRecognizer`, Junio 2023. Consultado en Julio de 2023.

[50] Google Cloud. Cloud speech-to-text. `https://cloud.google.com/speech-to-text?hl=es`, Julio 2023. Consultado en Julio de 2023.

[51] Developers Android. Documentación: Recognizerintent. `https://developer.android.com/reference/android/speech/RecognizerIntent`, Julio 2023. Consultado en Julio de 2023.

[52] Fil Alleva, Robert Brennan, Hsiao wen Hon, Ravishankar Mosur, Eric Thayer, Kevin Lenzo, Alan W Black, Evandro Gouvea, David Huggins-Daines, Alexander Solovets, and Vyacheslav Klimkov. Pocketsphinx. `https://github.com/cmusphinx/pocketsphinx`, Julio 2023. Consultado en Julio de 2023.

[53] M.G.P. Velthuis. *Fábricas de Software: Experiencias, Tecnologías y Organización. 2ª Ed.*, pages 50–55. RA-MA S.A. Editorial y Publicaciones, 2012.

[54] Developers Android. Documentación para desarrolladores de apps. `https://developer.android.com/docs?hl=es-419`, Julio 2023. Consultado en Julio de 2023.

[55] H.M. Deitel, P.J. Deitel, and A.V.R. Elizondo. *Cómo programar en Java*, pages 444–446. Pearson Educación. Pearson Educación, 2003.

[56] Google Cloud. Conceptos básicos de speech-to-text. `https://cloud.google.com/speech-to-text/docs/basics?hl=es`, Julio 2023. Consultado en Julio de 2023.

[57] S. Montero, T. Zarraonadía, P. Díaz, I. Aedo, P.S. Antonio, A. Lorenzo Pérez, M.A. Allidem Caluza, C.G. Ana, and A. Estévez Funes. *Patrones de diseño aplicados al desarrollo de objetos digitales educativos (ODE).* Serie Informes. Ministerio de Educación y Formación Profesional, 2011.

[58] Stanford NLP Group. Corenlp. `https://stanfordnlp.github.io/CoreNLP/`, 2023.

[59] John Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[60] P. Harper and Kent Norman. Improving user satisfaction: The questionnaire for user interaction satisfaction version 5.5. 01 1993.

[61] Julio Mejía Navarrete. El muestreo en la investigación cualitativa. *Investigaciones Sociales*, 4:165–180, 06 2014.

[62] Ignacio Díaz-Oreiro, Gustavo López, Luis Quesada, and Luis A. Guerrero. Standardized questionnaires for user experience evaluation: A systematic literature review. *Proceedings*, 31(1), 2019.