UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA:
INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de:
Ingenieros de Sistemas

TEMA:
DESARROLLO DE UN PROTOTIPO DE CHATBOT QUE PROVEA INFORMACIÓN RELACIONADA CON TRÁMITES CIUDADANOS, CASO: AGENCIAS DE TRÁNSITO CUYA INFORMACIÓN SE ENCUENTRA EN SITIOS WEB.

AUTORES:
BRYAN ANIBAL LICTO FREIRE
MIGUEL ANGEL SARITAMA CAZA

TUTOR:
FRANKLIN EDMUNDO HURTADO LARREA

Quito, septiembre del 2020

CESIÓN DE DERECHOS DE AUTOR

Nosotros: BRYAN ANIBAL LICTO FREIRE, con documento de identificación N° 0503616286 y MIGUEL ANGEL SARITAMA CAZA, con documento de identificación N° 1718443128, manifestamos que voluntariamente otorgamos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de titulación con el tema: "DESARROLLO DE UN PROTOTIPO DE CHATBOT QUE PROVEA INFORMACIÓN RELACIONADA CON TRÁMITES CIUDADANOS, CASO: AGENCIAS DE TRÁNSITO CUYA INFORMACIÓN SE ENCUENTRA EN SITIOS WEB", mismo que ha sido desarrollado para optar por el título de INGENIEROS DE SISTEMAS en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

…… ……

BRYAN ANIBAL
LICTO FREIRE
CI: 0503616286

MIGUEL ANGEL
SARITAMA CAZA
CI: 1718443128

Quito, septiembre de 2020

DECLARACIÓN DE COAUTORÍA DEL DOCENTE TUTOR

Yo, FRANKLIN EDMUNDO HURTADO LARREA, declaro que bajo mi dirección y asesoría fue desarrollado el Proyecto Técnico, con el tema: "DESARROLLO DE UN PROTOTIPO DE CHATBOT QUE PROVEA INFORMACIÓN RELACIONADA CON TRÁMITES CIUDADANOS, CASO: AGENCIAS DE TRÁNSITO CUYA INFORMACIÓN SE ENCUENTRA EN SITIOS WEB", realizado por: BRYAN ANIBAL LICTO FREIRE y MIGUEL ANGEL SARITAMA CAZA, obteniendo un producto que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana, para ser considerados como trabajo final de titulación.

Quito, septiembre del 2020

.....

FRANKLIN EDMUNDO HURTADO LARREA

CI: 1713382016

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Resumen

En el Ecuador 12 de cada de 100 personas posee algún tipo de vehículo, lo que ha incrementado la cantidad de trámites a realizarse tales como matriculación, revisión técnica vehicular, etc. con el fin de evitar multas innecesarias que conlleva el incumplimiento de estos. Pero la falta de acceso a estos recursos de índole informativo, pues al no existir una claridad sobre los documentos necesarios para realizar los trámites, ha provocado que las personas necesiten un mayor número de interacciones para completar sus gestiones, teniendo un mayor impacto en las personas con discapacidad visual. Por lo tanto, para solventar estos inconvenientes dentro de un contexto digital se propuso la creación de un prototipo de chatbot para proporcionar a los usuarios información de forma rápida y estructurada mediante una interfaz cómoda e intuitiva.

Este documento detalla el desarrollo del prototipo aplicando el patrón arquitectónico MVC y haciendo uso de la librería Chatterbot que automatiza las conversaciones con los usuarios utilizando algoritmos de similitud y el procesamiento de lenguaje natural, para proporcionar información confiable y coherente acerca de los trámites almacenados en una base de conocimientos mediante una interfaz gráfica que incorpora funcionalidades para el reconocimiento y síntesis de voz.

Para el proceso de desarrollo se utilizó el marco de trabajo incremental e interactivo desde el levantamiento de los requerimientos hasta la construcción y pruebas del sistema desarrollado, donde se demostró que el prototipo tiene una precisión mayor al 80% al proporcionar una respuesta confiable a una petición del usuario.

Abstract

In Ecuador, 12 people out of 100 have a type of car, that has increased the proceedings to be carried out such as registrations, vehicle technical review, etc. to avoid unnecessary fines that non-compliance with these entails. But the lack of access to these informational resources, since there is no clarity about the documents necessary to do the proceedings, has caused people to need a greater number of interactions to complete their procedures, having a greater impact on people visually impaired. Therefore, to solve these drawbacks within a digital context, the creation of a chatbot prototype was proposed to provide users with information in a fast and structured way through a comfortable and intuitive interface.

This document details the development of the prototype using MVC architectural pattern and Chatterbot library that automates conversations with users using similarity algorithms and natural language processing to provide reliable and consistent information about the procedures stored in a knowledge base through a graphical interface that incorporates functionalities for speech recognition and synthesis.

The incremental and interactive framework was used to development process from the gathering of the requirements to the construction and testing of the developed system, where it was shown that the prototype has an accuracy greater than 80% by providing a reliable response to a request of the user.

INTRODUCCIÓN

Antecedentes

Por lo general, algunas agencias de transito proporcionan en sus sitios web la información sobre los tramites ciudadanos que se realizan en cada una de ellas de forma presencial, pero la falta de acceso a estos recursos de índole informativo o al no existir una claridad sobre los documentos necesarios para realizar dichos trámites, el usuario se ve obligado a encontrarla realizando preguntas en su entorno. De tal manera, la información requerida puede quedar expuesta por diversos factores, y de no ser la correcta, eventualmente puede generar que el usuario pueda cometer errores al realizar dichos trámites, necesitando un mayor número de interacciones para completar estas gestiones y a la vez provoca una gran pérdida de tiempo y quejas que afectan la calidad de atención al cliente que manejan las agencias de tránsito.

Problemática

En el Ecuador según el último estudio a escala nacional sobre número de autos por habitantes realizado en el año 2018, 12 de cada de 100 personas posee algún tipo de vehículo (Medina, 2018), lo que ha incrementado la cantidad de trámites que los ciudadanos están obligados a realizar tales como matriculación, revisión técnica vehicular, entre otros, estos pueden tener requisitos o procesos diferentes, de acuerdo a cada provincia en el país.

Este factor de incidencia, ha provocado que una gran cantidad de personas se acerquen a las agencias de tránsito a realizar dichos trámites, con el fin de evitar multas innecesarias que conlleva el incumplimiento de los mismos, pero existe un inconveniente, en muchas ocasiones los usuarios realizan estas diligencias parcialmente, un reciente estudio en relación a trámites ciudadanos, refleja que en el Ecuador solo el 60% de los trámites se realizan en la primera interacción (Benjamin

Roseth, Angela Reyes, Carlos Santiso, 2018 Junio), dentro de este contexto el otro 40% no se realizan en su totalidad, esto debido a diversos problemas, entre los de mayor relevancia se destaca la desinformación y la falta de acceso a los recursos de índole informativo, pues al no existir una claridad sobre los documentos necesarios para realizar los trámites, las personas necesitan un mayor número de interacciones para completar sus gestiones, esto provoca una gran pérdida de tiempo y por ende un sinnúmero de quejas por parte de los usuarios, lo que desfavorece la calidad en la atención al cliente que manejan las entidades públicas.

Otros puntos a considerar son las personas con discapacidad visual, dado que en el Ecuador existen 54956 personas registradas (CONADIS, 2019) y la falta de contenido informativo representa según un estudio del Plan Nacional de Gobierno Electrónico 2018-2021 (Ministerio de Telecomunicaciones y de la Sociedad de la Información, 2018) una barrera de acceso a la información. Barrera que afecta a dichas personas, debido a que por razones propias o familiares tienen un cierto grado de interés en los trámites relacionados con las agencias de tránsito para obtener beneficios como exenciones o exoneraciones que involucra la reducción de costos en procesos como la matriculación vehicular, entre otros.

Justificación

Con el avance de la tecnología y de la inteligencia artificial, nuevas plataformas han aparecido para facilitar el acceso a la información, siendo una de ellas los chatbots, por ejemplo en el Ecuador varias instituciones bancarias, de salud y de la industria automotriz han integrado esta tecnología para la atención inmediata a sus clientes, un claro ejemplo es Banco del Pacífico y Banco de Guayaquil que tienen chats virtuales para resolver inquietudes y dudas, puesto que este software de manera autónoma sin la ayuda de un ser humano, puede realizar diferentes tareas al simular una conversión

casi humana respecto a un tema en específico, mejorando la atención al cliente al momento de solicitar algún recurso informativo, además ofrece una atención 24/7 que permite a los usuarios obtener respuestas inmediatas los 7 días de la semana y las 24 horas del día, convirtiéndose así en una estrategia digital viable.

Es necesario recordar que en la problemática se vio reflejada que en el Ecuador solo el 60% de los trámites se completan en la primera interacción, esto debido a diversos factores como la lectura y búsqueda de información, el aprender a interactuar con un nuevo sistema o navegar por una página web, esto según un estudio realizado por Latino barómetro (Benjamin Roseth, Angela Reyes, Carlos Santiso, 2018 Junio). Por lo consecuente el tiempo requerido para completar diversas diligencias se incrementa; un informe proporcionado por el BID menciona que en el país se requiere por lo menos de 4.2 horas para completar un trámite (Benjamin Roseth, Angela Reyes, Carlos Santiso, 2018 Junio) y que fue empleado por los factores ya antes mencionados.

Con respecto a las personas con discapacidad, según el Plan Nacional de Gobierno Electrónico 2018-2021 menciona que el 20% las plataformas web de Gobierno no cumplen con características de accesibilidad (Ministerio de Telecomunicaciones y de la Sociedad de la Información, 2018) y además un documento publicado por el CONADIS refleja que 466 personas con discapacidad visual importaron vehículos entre 2008 y 2015, aunque no es una cantidad muy relevante de personas que adquirieron vehículos en 7 años, estas al poseer este tipo de discapacidad y que, al adquirir esos bienes van a tener inconvenientes si desean obtener información sobre cualquier trámite relacionado con el tránsito.

Por lo tanto, para solventar estos inconvenientes dentro de un contexto digital se propuso la creación de un prototipo de chatbot para entregar a los usuarios información

de forma rápida, estructurada y ágil mediante una interfaz cómoda, intuitiva y con accesibilidad. Esta aplicación de chatbot al ser un asistente digital será capaz de procesar el lenguaje natural de las personas mediante el uso de APIS reconocimiento y síntesis de voz, tecnologías que hoy en día tienen un alto grado de madurez y que son utilizadas en distintos ámbitos para mejorar la atención al cliente.

Objetivo General

Desarrollar un prototipo de chatbot que mediante una interfaz más cómoda e intuitiva y con la integración del reconocimiento de voz, sea capaz de simular una conversación con una persona en lenguaje natural, con el fin de proporcionar la información solicitada, de manera rápida y eficaz acerca de cada uno de los trámites cuya información se encuentra en los sitios web de las agencias de tránsito en el Ecuador.

Objetivos Específicos

Investigar sobre las tecnologías para el desarrollo de chatbots, aprendizaje automático y algoritmos para el procesamiento de la información.

Crear una base de datos que permita centralizar la información de cada uno de los servicios de las agencias de tránsito que estén presentes en sus sitios web, los cuales deben estar estructurados y organizados de acuerdo con cada provincia del Ecuador, para su almacenamiento en la Base de Datos e integración en la aplicación de chatbot.

Crear un prototipo de chatbot usando APIS de reconocimiento de voz y aprendizaje automático para dar respuestas rápidas y estructuradas, con el fin de mejorar la experiencia y usabilidad frente al usuario.

Entrenar al chatbot mediante un conjunto de declaraciones y respuestas conocidas y probar su funcionalidad, usando preguntas frecuentes y regulares de usuarios finales para que la información solicitada a través de la aplicación sea la adecuada.

Propuesta de solución y alcance

Para el desarrollo del prototipo de chatbot una vez definidos los requerimientos funcionales y no funcionales, se utilizó una arquitectura MVC (Modelo-Vista-Controlador) el cual permitió separar los datos, la interfaz de usuario y la lógica de control del sistema, para gestionar el desarrollo y el mantenimiento de esta.

La base de conocimiento es gestionada por MongoDB la cual contiene la información de los tramites de las agencias de tránsito obtenida mediante un servicio web a partir de documentos JSON generados por la técnica para extracción de datos web scraping. Cabe mencionar que el prototipo solo responde en base a la información de los trámites extraídos de la ciudad de Quito y Guayaquil.

En el parte del back-end del prototipo se utilizó la librería de código abierto Chatterbot para el procesamiento del lenguaje natural (NLP), búsqueda y selección de respuestas utilizando algoritmos de similitud, un proceso de aprendizaje supervisado en función de niveles de confianza y para la parte del front-end se utilizó el framework Flask para integrar estas tecnologías en una interfaz amigable al usuario.

Con la implementación de APIS de reconocimiento y síntesis de voz, el chatbot permite a los usuarios interactuar con él por medio de dispositivos de entrada y salida de audio, tales como manos libres, micrófonos, etc. sin la necesidad de escribir a través del teclado, es decir esta aplicación, transforma el mensaje de voz en texto para realizar un petición en lenguaje natural y una vez que esta petición sea procesada, el resultado nuevamente se convierte en una respuesta de voz para el usuario, facilitando así el medio de comunicación, esto no solo ayuda a las personas con falta de alfabetización digital , sino que tiene un mayor impacto en las personas con discapacidades motoras y visuales.

Marco metodológico

Para el marco metodológico de este proyecto se utilizó una metodología de investigación bibliográfica con el fin de obtener información basada en estudios, artículos, libros, reportajes, etc. sobre la falta de accesibilidad a la información web y población con discapacidad visual para comprender la esencia y particularidad de la problemática del caso estudio, es decir; apreciar su comportamiento en cuanto a los trámites en entidades públicas, centrándonos en los servicios de las agencias de tránsito dependiendo su localidad en el Ecuador, además de las herramientas tecnológicas para el desarrollo de chatbots tales como: algoritmos de similitud, procesamiento del lenguaje natural, etc.

Con respecto al desarrollo del chatbot, si bien es cierto una metodología ágil ayuda al desarrollo de proyectos que necesitan rapidez así como flexibilidad para adaptarse a las necesidades del cliente, realizar cambios si fuese necesarios y mostrar los avances realizados, esta metodología no fue idónea para este proyecto, pues en el contexto propuesto no se dispuso de un cliente específico, rol indispensable en estas metodologías, por este motivo, al tener relativamente bien definidos los requerimientos, se utilizó el marco de trabajo modelo incremental e interactivo (análisis, diseño, desarrollo y pruebas del software) para los procesos inmersos en el proyecto, de tal manera que al iterar en el desarrollo del mismo se obtuvo una retroalimentación sobre la estabilidad de la funcionalidad del prototipo en base a unas métricas definidas en relación al estándar de calidad ISO 9126, que establecieron un nivel de evolución de la aplicación desarrollada, minimizando el número de errores que se ocurrieron en el desarrollo, mejorando el nivel de calidad del chatbot.

CAPÍTULO 1

MARCO TEÓRICO

1.1. Trámites agencias de tránsito en el Ecuador

En el Ecuador la entidad pública que se encarga de garantizar la movilidad terrestre de forma segura y libre es la Agencia Nacional de Tránsito A.N.T, la misma ofrece varios servicios y procesos a seguir sobre tramitación vehicular. Dentro de los servicios más importantes que ofrece la institución son: mostrar un historial sobre las infracciones cometidas por el conductor, verificar el registro de pagos sobre infracciones de tránsito, solicitudes con respecto a licencias de conducir y solicitudes referentes a vehículos para certificaciones vehiculares, bloqueos y desbloqueos entre otros.

Si bien la A.N.T dentro de sus obligaciones como institución ofrece estos trámites, son los municipios de cada ciudad quienes asumen las competencias de tránsito, esto según el artículo 338 del Código Orgánico de Organización Territorial, Autonomía y Descentralización -COOTAD (PRESIDENCIA DE LA REPUBLICA DEL ECUADOR, 2010). Por ello la Agencia Nacional de Tránsito registra 3 tipos de modelo de gestión: A, B y C (Redacción Política, 2014). Los cabildos pueden ejecutar diversas operaciones de acuerdo con el modelo que se le ha sido asignado, en el caso A, las entidades están encargadas de la matriculación y revisión vehicular, además de controlar del tránsito por medio de agentes y también la posibilidad de otorgar permisos de operación para transporte Inter cantonales, taxis, carga liviana entre otros. En el caso B se deben hacer cargo de las mismas competencias que el primer modelo, con excepción del control de tránsito y por último el caso C, el cual solo otorgan permisos de operación. Los municipios han ido de poco tomando el control de las competencias, muchos de ellos ofrecen información detallada de sus servicios en un sitio web, incluso algunas permiten realizar trámites directamente de sus portales.

1.2. Chatbot

Hoy en día varias empresas y negocios ofrecen información sobre sus productos, servicios, actividades a través de sus portales web, incluso algunos utilizan herramientas adicionales para mejorar la experiencia y agilizar la búsqueda de recursos de los usuarios, una de estas herramientas son los chatbot.

Esta palabra hace referencia al término chat, charla en inglés, y bot a un robot y no son más que programas que poseen características que les permite interactuar con el usuario por medio de una conversación, interpretando el Lenguaje Natural de las personas (PLN). El desarrollo de los chatbots ha ido mejorando progresivamente sus funciones, desde tener un bot denominado Eliza, creado en la década de los 60, quien era capaz de buscar palabras claves y responder en forma oraciones, hasta tener asistentes virtuales robustos que interactúan de forma fluida e interactiva con los usuarios mediante texto o voz como Alexa, Google Assistant, Cortana entre otros. (Reyna, s.f.)

1.2.1. Procesamiento del lenguaje natural (NLP)

El NLP es un campo de la Inteligencia Artificial que ayuda a los computadores a interpretar, entender y manipular el lenguaje humano. Al existir una diversidad de dialectos, palabras, expresiones propias del ser humano, se hace uso de datos no estructurados para almacenar toda esa información.

Para que una máquina tenga la capacidad de tener conversión fluida con las personas, se tiene que usar un algoritmo de machine learning (aprendizaje automático) el cual procesa y ejecuta un análisis de los patrones de entrada y así generar una respuesta aceptable. Al completar este proceso, el algoritmo realiza un aprendizaje en base al

conjunto de entradas y respuestas con el fin de tener un mejor nivel de precisión para futuras consultas. (Indurkhya, Damerau, Graepel, & Herbrich, 2010)

1.2.2. Motor de inferencia

Es un programa de control basado en una serie de estrategias cuya función es procesar reglas de un problema y así alcanzar una solución determinada, es decir; es el componente encargado de aplicar procesos de análisis y deducción de manera consistente en función a reglas definidas, para tomar una decisión y seleccionar la respuesta a una pregunta y presentarla al usuario, considerando las variables de entrada, estado y de salida. (Mathivet, 2018)

Motor de Inferencia

Figura 1: Reglas y hechos como parte del motor de inferencia
Fuente: (Mathivet, 2018)

En un chatbot, el motor de inferencia se encarga del procesamiento de la información de la base de cocimiento, es decir; se encarga de gestionar la selección de respuestas a una petición realizado por un usuario mediante un conjunto de reglas o hechos como se puede observar en la figura 1.

1.3. Plataformas y bibliotecas para el desarrollo de chatbots

Actualmente existen una variedad de servicios, librerías, frameworks que facilitan el desarrollo e integración de chatbots.

1.3.1. Chatterbot

Es una librería de Python de código abierto para el desarrollo de chatbots, que utiliza algoritmos para la selección de respuestas a peticiones y para el aprendizaje automático, lo que alimenta su base de conocimiento e incrementa el número de conversaciones con los usuarios.

En primera instancia Chatterbot comienza sin saber comunicarse, por ende, cada vez que un usuario ingresa una petición, esta librería almacena el texto ingresado y al que respondía la petición, entonces a medida que se incrementa la información, aumenta la precisión de cada respuesta a una petición. Para ello se integra ciertos módulos de configuración que simplifican este proceso de manera rápida y eficaz como se muestran a continuación:

- **Adaptador lógico:** funciones para seleccionar una respuesta a una petición de entrada.
- **Adaptador de almacenamiento:** funciones que permiten al chatbot conectarse con diferentes gestores de almacenamiento tales como: MongoDB, MySQL, entre otras.
- **Preprocesadores:** funciones que modifican una petición, antes de que el adaptador lógico la procese.
- **Filtros:** consultas que se envía como parámatelos a los adaptadores de almacenamiento, para reducir la cantidad de declaraciones a procesar cuando se selecciona una respuesta.
- **Entrenadores:** funciones que reciben un conjunto de datos y crea las entradas necesarias en la base de conocimiento para que las entradas y respuestas de una declaración se asocien correctamente.

Capacitación Chatterbot

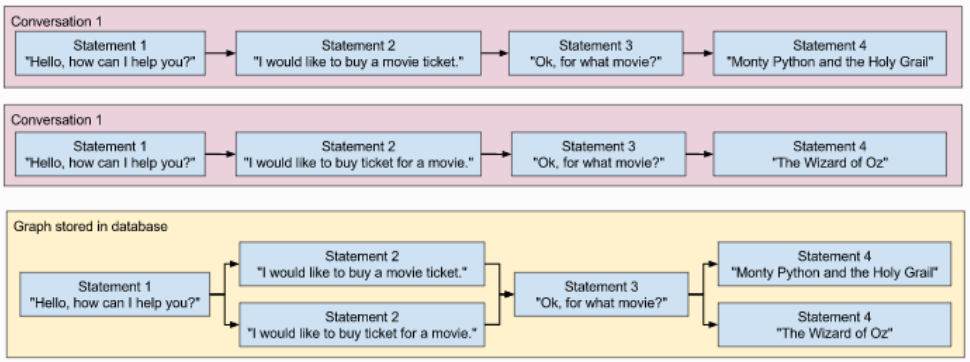

Figura 2: Modelo de capacitación de la librería Chatterbot
Fuente: (Gunther Cox Revision, 2019)

Como se puede observar en la figura 2, un chatbot puede recibir una misma respuesta para diferentes entradas para mejorar la conversión con el usuario.

- **Comparaciones:** son algoritmos para seleccionar una respuesta en base a su capacidad de comparar dos afirmaciones entre sí (Gunther Cox Revision, 2019). ChatterBot integra 4 algoritmos de similitud los cuales se describen a continuación:
    - **Jaccard Similarity:** es un algoritmo creado por Paul Jaccard en 1901, que mide la similitud de dos conjuntos A y B, donde el resultado se obtiene de la división del tamaño de la intersección del conjunto A y B (elementos comunes) sobre el tamaño de la unión del conjunto A y B (elementos únicos). (Rim Jallouli, 2017)
    - **Levenshtein Distance:** es una métrica para determinar la diferencia entre dos secuencias de texto, considerando el número mínimo de ediciones de un carácter (inserción, eliminación o modificación) requeridas para cambiar un texto en otro. Lleva el nombre de Vladimir Levenshtein, quien descubrió esta ecuación en 1965". (Vijay Nath, 2019)
    - **Synset Distance:** es un método para determinar la similitud de los sentidos de las palabras, a partir de los Synsets (palabras de WordNet) que ocurren

en el árbol de significados globales, donde se considera la distancia más corta entre los dos Synsets como el nivel de similitud (Perkins, 2014)

- **Sentiment Comparison:** mide la similitud de textos mediante reglas que identifican la subjetividad, la polaridad o el tema de una opinión. Las reglas incluyen técnicas de lingüística computacional, tales como: tokenización, etiquetado, análisis de parte del discurso y Léxicos (listas de palabras y expresiones). (MonkeyLearn, 2020)

1.3.2. Rasa Stack

Es una herramienta para el aprendizaje automático de código abierto que permite la creación de chatbots. Está basado en dos contextos:

- **Core:** un marco de chatbot con gestión de diálogo basada en aprendizaje automático.
- **NLU:** una biblioteca para el procesamiento del lenguaje natural que clasifica intenciones y extracción de entidades.

Estos recursos son consumidos mediante un API REST. (Stack, 2020)

1.3.3. DialogFlow

Es una plataforma de propiedad de la compañía Google, que permite el procesamiento del lenguaje natural, facilitando el diseño de una interfaz de usuario de conversación y su integración a las aplicaciones para dispositivos móviles y aplicación web, además ofrece el uso tecnologías de reconocimiento y síntesis de voz. (Google, 2020)

Está conformado por entents para clasificar la intención de búsqueda del usuario e intenciones que contiene la información relacionada a una búsqueda. Tanto el almacenamiento como las funcionalidades están disponibles como servicio en la nube previa creación de una cuenta.

1.3.4. IBM Watson Conversation Service

Es una plataforma de Inteligencia Artificial para la creación de conversaciones con los clientes a través de respuestas rápidas, directas y precisas en cualquier aplicación. IBM Watson hace uso de intenciones, entidades y un seguimiento de diálogo para centrarse en casos complejos y no en respuestas redundantes al establecer una conversación con el usuario. Además, es compatible con servicios de traducciones, reconocimiento y digitalización de voz y se lo puede implementar en la nube o de manera local. (Rothman, 2018)

1.3.5. Wit.ai

Es un servicio web, propio de Facebook, para desarrollar bots en diferentes plataformas como: slack, messenger, telegram, entre otras. Wit.ai usa historias y flujos de diálogo para la creación del agente conversacional a través del módulo de Inteligencia Artificial con NLP para interpretar y procesar las acciones del usuario. (Facebook, 2020)

1.3.6. Microsoft Bot Framework

Azure Bot Service es la plataforma Microsoft para el diseño y desarrollo de chatbots, que integra entornos específicos con herramientas de *Microsoft Bot Framework* y SDK's como interfaces para establecer la lógica de conversación con el usuario. A través del Bot Framework se utiliza recursos de Inteligencia Artificial, de Machine Learning para realizar una comunicación con datos REST, y el procesamiento del lenguaje natural. (Microsoft, 2020)

1.3.7. Amazon Lex

Es un servicio de AWS para crear interfaces de conversación de chatbots, que utiliza como motor de conversación Alexa y proporciona flexibilidad para el procesamiento

del lenguaje natural (NLU) y el reconocimiento automático de voz (ASR). Para crear un chatbot, solo es necesario especificar el flujo de conversación en la consola de Amazon Lex, para administrar el diálogo y ajustar las respuestas en la conversación. (Amazon, 2020)

1.4. Tecnologías y Lenguajes de Programación

1.4.1. Python

Python es un lenguaje de programación interpretado creado por Guido van Rossum, el cual se enfatiza en la simplicidad de la sintaxis y legibilidad de código. “Es compatible con paradigmas de programación incluyendo orientado a objetos, imperativo, funcional y de procedimiento y tiene una biblioteca completa” (Tudor, 2019). Este lenguaje de programación incluye un entorno de ejecución y un intérprete de líneas de comando en su instalación.

1.4.2. PHP

Es un lenguaje de programación de código abierto utilizado para desarrollar aplicaciones web y puede ser incluido en un código HTML. Este lenguaje de ejecuta en la parte del servidor y ofrece una serie de funcionalidades, correcciones y mejoras en cada una de sus versiones, siendo la más reciente la versión 7.4.8. Además, permite la conexión entre diversos servidores de base datos tales como PostgreSQL, MySQL, Oracle, Microsoft SQL Server entre otros. (PHP, 2020)

1.4.3. Servicio web

Es un sistema que permita la interoperabilidad entre dos aplicaciones mediante una red de comunicación. Para concretar el intercambio de datos se hace uso de distintas tecnologías tales como XML, WDSL, SOAP, REST independientemente del lenguaje de programación en que los sistemas hayan sido desarrollados. (W3C, 2020)

1.4.4. JSON

"JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos". (Json.org, 2020)

1.4.5. MongoDB

Es un gestor de base de datos NoSQL de código abierto, multiplataforma, el cual es orientado a documentos que los almacena en estructuras de datos BSON (similar a JSON). La diferencia respecto a las bases de datos relacionales es que no requiere seguir un esquema en específico para los datos de una colección.

1.4.5.1 Colección

En Gestores de Base de datos NoSQL, como lo es MongoDB es necesario conocer sobre las colecciones, las cuales son similares a una tabla en una base de datos relacional. La diferencia está en que las tablas almacenan registros en forma de filas y las colecciones almacenan documentos. (Aguilar, 2016)

1.4.6. SpeechSynthesis

Se trata de un componente que permite leer un texto y reproducirlos a través de un sintetizador de voz. Además, ofrece funcionalidades adicionales que permiten tener un mejor control de la herramienta al poder iniciar, detener o pausar la reproducción de voz. (Mozilla, 2020)

1.4.7. SpeechRecognition

Es una API que proporciona las funcionalidades de reconocimiento y síntesis de voz en un entorno web. Para el reconocimiento de voz la API una vez que recibe una

expresión, la compara con una lista de palabras o frases y cuando las reconoce, estas se convierten en una cadena de texto, lo contrario es la síntesis de voz, ya que recibe una expresión en forma de texto y lo convierte en voz para ser reproducida por el dispositivo.

La función del reconocimiento de voz es compatible solo con el navegador Chrome, en cambio la síntesis de voz es compatible con Chrome y Firefox en el cual se requiere habilitar permisos media. (Libby, 2020)

1.5. Herramientas para el desarrollo de software

1.5.1. NTLK

Es un kit de herramientas para procesamiento de lenguaje natural para Python, "que proporciona recursos corporales y léxicos como WordNet y un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización (segmentar palabras de un texto), derivación, etiquetado, análisis y razonamiento semántico para bibliotecas NLP." (Proyecto NLTK, 2020)

Los recursos ntkl que Chatterbot necesita para el funcionamiento de los algoritmos de similitud son:

- **nltk_averaged_perceptron_tagger:** etiqueta cada palabra de una oración en función a una lista de sustantivos, artículos, adjetivos, etc.
- **nltk_stopwords:** conjunto de palabras tales como: artículos, pronombres que son excluidos al comparar una oración.
- **nltk_wordnet:** base de datos léxica en ingles que contiene palabras como sinónimos, antónimos, etc.
- **nltk_vader_lexicon:** conjunto de palabras para determinar los sentimientos de un texto en las redes sociales.

- **nltk_punkt:** sirve para tokenizar oraciones. (Proyecto NLTK, 2020)

1.5.2. API REST

"Una interfaz de programación de aplicaciones (API) es un conjunto de herramientas, definiciones y protocolos que se utiliza para integrar los servicios y el software de aplicaciones." (RedHat, 2020).

La transferencia de estado representacional (REST) es un conjunto de principios arquitectónicos que permiten a las APIs comunicarse a través del Protocolo de transferencia de hipertexto (HTTP) para ejecutar las operaciones de inserción, consulta, actualización y eliminación de recursos.

1.5.3. PyCharm

Es un IDE multiplataforma que es utilizado con el lenguaje de programación Python, el cual pertenece a la empresa JetBrains. "PyCharm incluye en sus características un depurador y un ejecutor de pruebas integrados, perfilador Python, un terminal integrado, integración con los principales VCS y herramientas de base de datos integradas, capacidades de desarrollo remoto con intérpretes remotos." (JetBrains, 2020)

Este IDE tiene dos versiones, la Community que es gratuita y orientada al desarrollo puro ya que ofrece características fundamentales para cualquier programador en Python y la Professional, que adicionalmente tiene características que se enfocan en el desarrollo web con la implementación de frameworks como Django y Flask.

1.5.4. Flask

Es un framework para Python, que permite la creación de aplicaciones web de forma ágil y sencilla. Está basado en Werkzeug (una interfaz que permite la comunicación

entre un servidor y una aplicación web) que ofrece herramientas adicionales para crear nuevas funcionalidades en un proyecto. (Palletsprojects, 2020)

Flask ofrece el uso de plantillas que contiene una estructura HTML (HyperText Markup Language) conformada por bloques, los cuales podrán ser invocados dentro una plantilla esqueleto según sean requeridos. A este proceso se lo denomina "herencia de plantillas" y reduce el código HTML empleado.

1.5.5. JMeter

Es un software de código abierto diseñado en Java que permite cargar y medir el rendimiento de aplicaciones de las aplicaciones y servidores, aunque fue inicialmente diseñado para aplicaciones web, ha ido expandiendo a otros tipos de funciones tales como: HTTP, FTP (Protocolo de transferencia de archivos), bases de datos, servicios web entre otros. (Apache.org, 2020)

1.6. MVC

El patrón arquitectónico modelo-vista-controlador, es un modelo para organizar y estructurar los componentes de un sistema a desarrollar y las relaciones existentes entre cada uno de ellos.

Patrón arquitectónico MVC

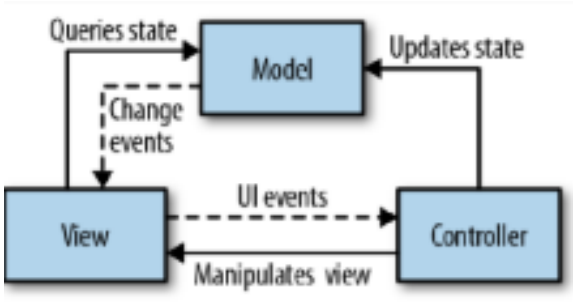

Figura 3: Componentes del patrón MVC
Fuente: (Elliott, 2014)

MVC presenta tres componentes para el desarrollo de sistemas informáticos, como se

puede observar en la figura 3, el modelo contiene los datos que opera el sistema, la vista es la interfaz gráfica que presenta al usuario la información del modelo y el controlador que funciona como intermediario entre el modelo y la vista, se encarga de gestionar las instrucciones de entrada y procesarlas, solicitando los datos necesarios para obtener resultados y enviarlos para que pueda ser mostrados. (Elliott, 2014)

1.7. ISO 9126

Publicado en 1991 y revisado en 2001, el ISO 9126 es un estándar internacional que permite la evaluación de la calidad del producto software. Con este propósito se define que los componentes pueden definirse en un conjunto de características en cuanto a la calidad interna y externa del sistema, estas características son: funcionalidad, usabilidad, portabilidad, usabilidad, eficiencia, facilidad de mantenimiento y portabilidad.

Aunque este estándar tiene varios años desde su publicación, es muy usado actualmente en diversos proyectos por la jerarquía multinivel que ofrece y que está compuesta por atributos y características que se hacen uso dependiendo con el tipo de aplicación. (Calero, Piattini, & Moraga, 2010)

1.7.1. Pruebas de usabilidad

Las pruebas de usabilidad permiten medir el grado de entendimiento, aprendizaje, satisfacción y manejo del software en distintas condiciones. Por lo general está enfocado a un grupo específicos de usuarios que puedan aportar detalles relevantes de uno o varias características del sistema de este modo se puede agregar, modificar o añadir funcionalidades que se adapten a los requerimientos de los usuarios finales. (Calero, Piattini, & Moraga, 2010)

1.7.2. Pruebas de fiabilidad

Se utilizan para medir la capacidad de un sistema para poder ejecutar sus funciones bajo condiciones específicas durante un periodo de tiempo determinado Un software debe tener la capacidad de seguir funcionando al ejecutar ciertas tareas que puedan generar problemas y además debe ofrecer soluciones que permitan su disponibilidad cuando se lo requiera. (Calero, Piattini, & Moraga, 2010)

1.7.3. Pruebas de eficiencia

Estas pruebas verifican la capacidad que tiene el sistema poder ofrecer un desempeño adecuado al utilizar una cantidad específica de recursos. Para ello se pueden ejecutar pruebas de rendimiento y estrés, en donde, se establece un rango de tiempo con una determinada cantidad de solicitudes al sistema que permitan saber hasta qué punto se ofrece un rendimiento óptimo o se produce algún fallo que lo haga inoperable. (Calero, Piattini, & Moraga, 2010)

1.7.4. Pruebas de funcionalidad

Permiten verificar que el software proporcione las funciones que permitan satisfacer las necesidades para las que fue diseñado. Estas pruebas se los realiza en base a los requisitos del usuario o grupo de usuarios que harán uso del sistema y que determinan si las funciones creadas cumplen o no con lo esperado. (Calero, Piattini, & Moraga, 2010)

CAPÍTULO 2

ANÁLISIS Y DISEÑO DEL SISTEMA

2.1. Definición y análisis de requerimientos

Para la definición y análisis de los requerimientos del prototipo, se consideró la funcionalidad de un chatbot y solventar ciertas necesidades de los usuarios respecto al manejo de información, dado que algunos carecen de alfabetización digital o tienen algún tipo de discapacidad visual, lo que hace un factor crítico al momento de solicitarla. De igual manera para garantizar la calidad del sistema se usó el estándar ISO 9126.

2.1.1. Requerimientos funcionales

Con el fin de obtener descripciones explicitas del funcionamiento que debe tener el prototipo y de la información a procesar, se definieron los siguientes requerimientos:

- El sistema deberá almacenar y procesar datos no estructurados (NoSQL) sobre trámites ciudadanos extraídos de los sitios web de las agencias de tránsito.
- El sistema deberá consumir datos JSON a través de un servicio web para la creación de la base de conocimiento.
- El prototipo permitirá a los usuarios utilizar herramientas como el reconocimiento de voz y reproductor de texto para facilitar y agilizar el procedimiento de búsqueda de información.
- Al no existir información sobre alguna consulta realizada, el sistema deberá responder acorde a la omisión de dicha información o generar alternativas como posibles respuestas.
- El aprendizaje automático se ejecutará al momento de encontrar alternativas como posibles respuestas a una búsqueda, las cuales serán mostradas por el prototipo y deberán ser seleccionadas por el usuario.

2.1.2. Requerimientos no funcionales

Con relación al estándar ISO 9126 para la evaluación de la calidad de software, se definieron los siguientes requerimientos:

- El prototipo deberá tener un diseño "responsive" para acceder de forma ágil y sencilla a su interfaz y a cada una de sus herramientas.
- El tiempo de respuesta del prototipo al realizar una consulta no deberá ser mayor a 3 segundos.
- El número de iteraciones realizadas en el prototipo hasta obtener una respuesta requerida por el usuario no deberá ser mayor a 5.
- El porcentaje de aprobación de las respuestas emitidas por el prototipo hacia el usuario deberá tener un margen de éxito de por lo menos el 80%.

2.2. Esquema PBS del prototipo

Una vez definidos los requerimientos y para observar de manera clara la distribución de los componentes del prototipo, se utilizó una PBS como se puede observar en la figura 4.

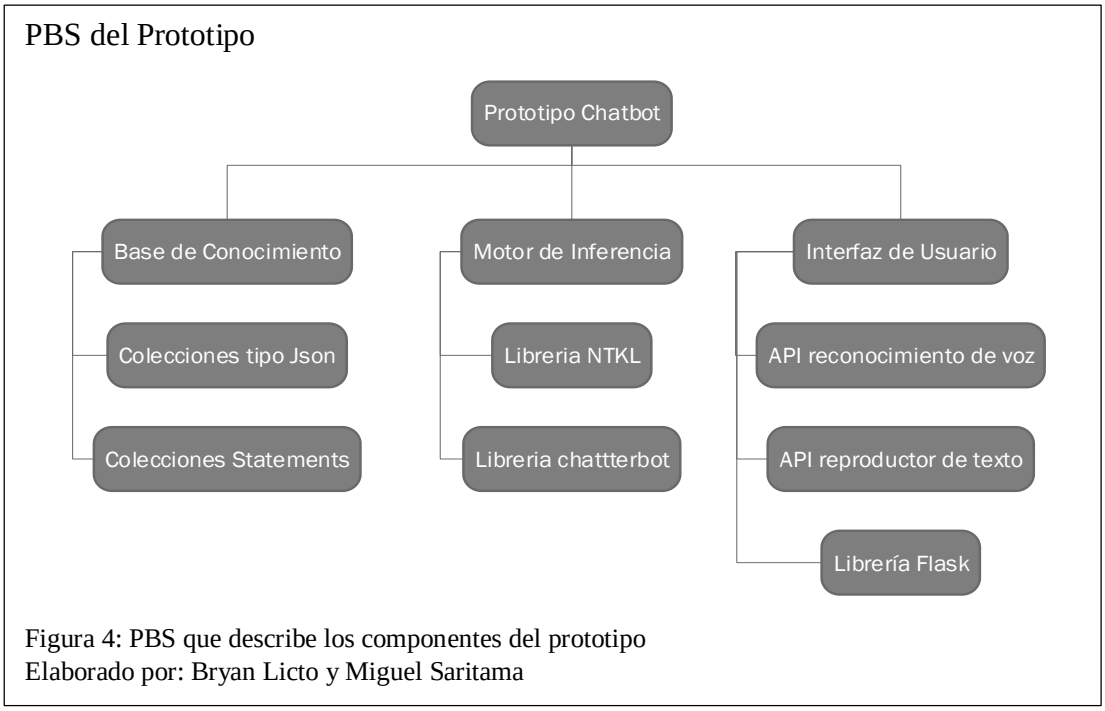

Figura 4: PBS que describe los componentes del prototipo
Elaborado por: Bryan Licto y Miguel Saritama

En donde:

- Base de Conocimiento: almacena las plantillas en un formato pregunta-respuesta en la colección statement a través del entrenamiento y el autoaprendizaje, además de la información tipo JSON obtenida de un proceso de Web scraping.
- Motor de Inferencia: procesa, genera y selecciona una respuesta a las entradas de los usuarios, a través del procesamiento de lenguaje natural y de adaptadores lógicos.
- Interfaz de usuario: sitio tipo chat, donde el usuario puede realizar consultas y recibir la información solicitada.

2.3. Análisis preliminar

2.3.1. Análisis y selección de plataformas o librerías a utilizar en la construcción del prototipo

Actualmente existen varias plataformas o librerías para el desarrollo y creación de chatbots, las cuales se pueden integrar mediante librerías, SDK o llamadas API REST. Aunque estas últimas son las más fáciles de encontrar, pues muchas empresas ofrecen estos servicios para una rápida creación, configuración e implementación en sitios web, pero esto implica un costo de operación para poder controlar completamente sus funcionalidades. Por otro lado, existen librerías y entornos de desarrollo de acceso libre que son similares a las de pago las cuales permiten tener un control total de la herramienta, al ofrecer un lenguaje de programación para su desarrollo o facilitar el uso de distintas bases de datos para el almacenamiento.

En la tabla 1 se detalla los aspectos más relevantes para el desarrollo del prototipo, de diversas herramientas tecnológicas utilizadas para la creación de chatbots.

Tabla 1. Comparativa plataformas o librerías para el desarrollo de chatbots

Herramientas Tecnológicas	**Plataforma de desarrollo**	**Herramienta de Implementación/ Desarrollo**	**Lenguaje Programación**	**Tipo de Base de Datos**	**Licencia**
Chatterbot	Local	Librería	Python	SQL, MongoDB	Código abierto
Rasa Stack	Local/ Cloud	SDK, API REST	Node.js, Ruby, PHP, Python	SQL, MongoDB, Redis, DynamoDB	Código abierto / pago
DialogFlow	Cloud	SDK, API REST	Android, iOS, Cordova, HTML, JavaScript, Node.js, .NET, Unity, Xamarin, C++, Python, Ruby, PHP (community supported), Epson Moverio, Botkit, Java	Firebase (Nube)	Servicio gratuito /pago
IBM Watson Conversation Service	Cloud	SDK, API REST	Node, Java, Python, iOS, Unity	Db2, PostgreSQL (Nube)	Servicio gratuito /pago
Wit.ai	Cloud	SDK, API REST	Node, Python client, Ruby	Nube	Servicio gratuito /pago
Microsoft Bot Framework	Cloud	SDK	C++, JS, Python, Java	Azure SQL	Código abierto SDK /pago
Amazon Lex	Cloud	SDK	Java, JS, Python, CLI, .Net, Ruby, PHP, Go y CPP	Amazon DynamoDB	Pago

Nota: Esta tabla contiene la comparación de las herramientas tecnológicas para la creación de chatbots.
Elaborado por: Bryan Licto y Miguel Saritama

Al realizar un análisis preliminar y considerando: el desarrollo local del prototipo, un lenguaje de programación de fácil aprendizaje como es el caso de Python, una base de datos que maneja datos no estructurados y una licencia que no implica costo de operación, se decidió utilizar la librería Chatterbot.

Esta librería contiene lo necesario para interpretar el Lenguaje de Procesamiento natural (NLP) como lo es NTKL y una serie algoritmos para la generación de respuestas a las entradas de los usuarios con el fin de automatizar las conversaciones con los mismos.

2.3.2. Análisis para la selección del algoritmo de similitud

Una parte importante de un chatbot es la forma de seleccionar una respuesta de una entrada mediante la capacidad de comparar una serie de afirmaciones. Es aquí donde Chatterbot facilita esta funcionalidad al integrar en su librería ciertos algoritmos de similitud.

Tabla 2. Tabla de comparación de algoritmos de similitud

Algoritmos de similitud	Métodos NTLK Necesarios	Frase A	Frase B	% Similitud
Jaccard Similarity	nltk_averaged_perceptron_ tagger nltk_stopwords nltk_wordnet	La situación de américa del sur es delicada, aunque ha mejorado	América del sur esta delicada, pero sigue mejorando	0,45
Levenshtein Distance	Ninguno	La situación de américa del sur es delicada, aunque ha mejorado	América del sur esta delicada, pero sigue mejorando	0,70
Sentiment Comparison	nltk_vader_lexicon	La situación de américa del sur es delicada, aunque ha mejorado	América del sur esta delicada, pero sigue mejorando	0,40
Synset Distance	nltk_punkt nltk_stopwords nltk_wordnet	La situación de américa del sur es delicada, aunque ha mejorado	América del sur esta delicada, pero sigue mejorando	1,38

Nota: En esta tabla se detalla los algoritmos de similitud que ofrece la librería Chatterbot que compara dos frases. Algunos requieren hacer uso de métodos adicionales para funcionar.
Elaborado por: Bryan Licto y Miguel Saritama

Una vez realizado un análisis comparativo, considerando los recursos necesarios para su ejecución y al aplicar un ejemplo práctico para determinar el porcentaje de similitud entre dos frases a cada uno de los algoritmos los cuales se pueden visualizar en la tabla 2. Se optó por utilizar el algoritmo de Levenshtein Distance al arrojar el mayor porcentaje de similitud y por no hacer uso de recursos adicionales NLTK para su ejecución. Aunque el método synset distance arroja un porcentaje elevado de similaridad, las frases comparadas no tienen un contenido idéntico, esto se produce por la forma de búsqueda que utiliza este método.

2.4. Diagrama de Caso de Uso

Al tratarse de un chatbot y al saber que su función es simular y mantener una conversación con un usuario, al proporcionar respuestas a las entradas realizadas por el mismo, se utilizó un diagrama de casos de uso UML con el fin de obtener una descripción más clara y visual de sus funciones, desde el punto de vista del usuario.

Casos de uso usuario y prototipo

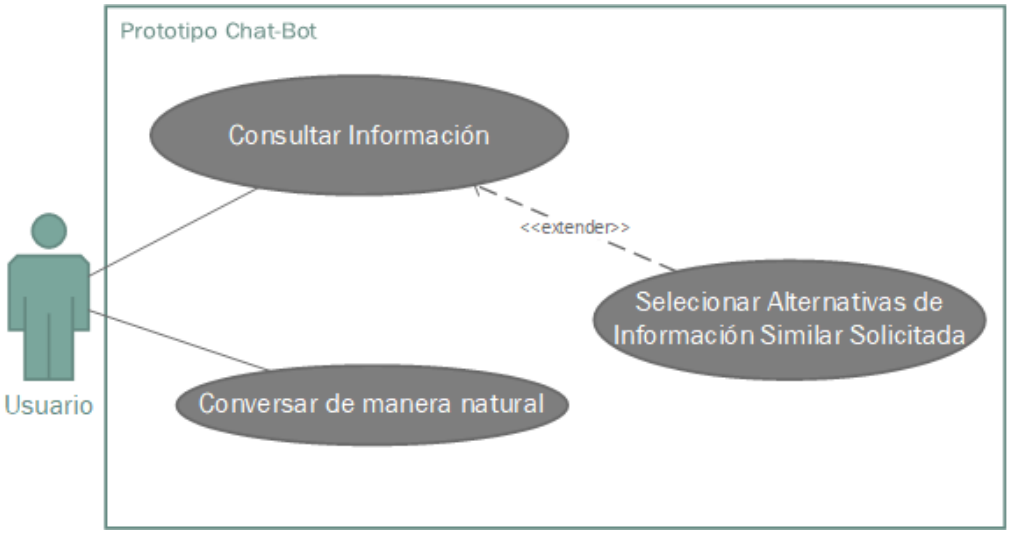

Figura 5: Caso de uso para consultar información e interactuar con el prototipo
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 5, se puede visualizar como un usuario únicamente puede entablar dos maneras de conversación, las cuales son: Consultar Información, la cual hace

referencia a la obtención de información de un tema en específico y Conversar de manera natural, que trata sobre los dialectos básicos en una conversación, es decir; responder a preguntas tales como: ¿Cómo estás?, ¿Cómo te ha ido?, ¿Cómo te va?, etc., o a la vez un simple saludo, por ejemplo, un Hola.

2.5. Diagrama de Clases

Diagrama de clases del prototipo

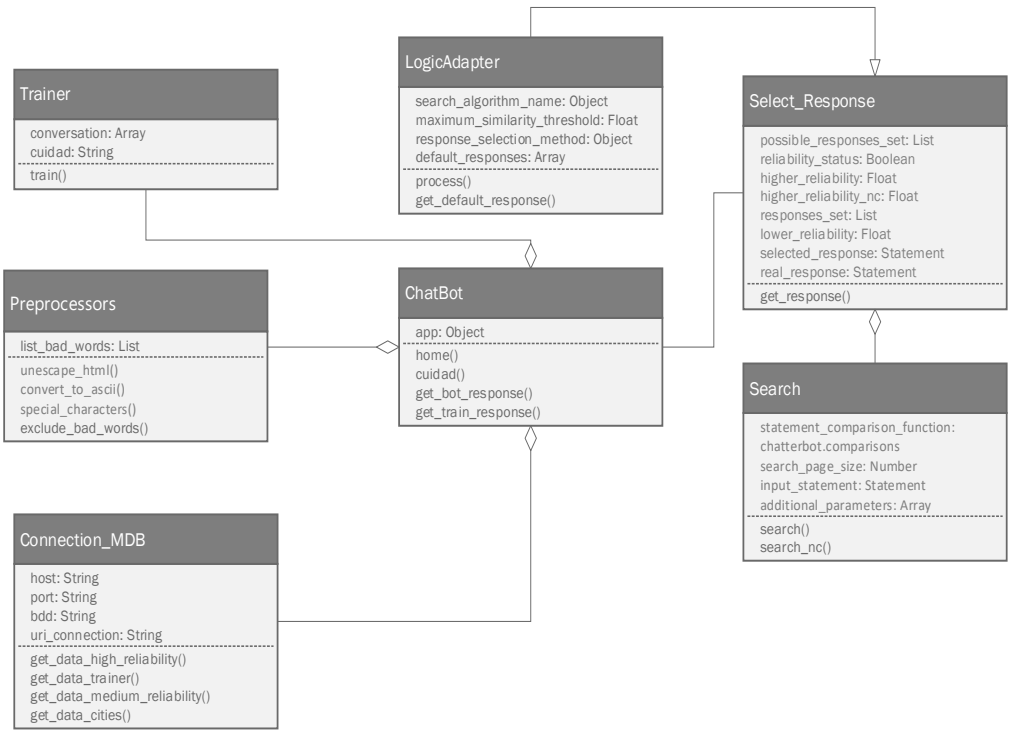

Figura 6: Diagrama de clases del prototipo
Elaborado por: Bryan Licto y Miguel Saritama

Con el fin de apreciar de manera estética y visual cada una de las clases modificadas que describen la estructura del Prototipo, se utilizó un diagrama de Clases UML como se observa en la figura 6. Este diagrama está compuesto por siete clases, de las cuales: la clase Chatbot, a más de contener el método main() para la ejecución del prototipo requiere de una cierta dependencia con las clases Trainer, Preprocessors y Conection_MongoDB, debido a que cada una de ellas contienen las variables y métodos para el aprendizaje de nuevas plantillas tipo pregunta-respuesta, para la

modificación de las declaraciones de entrada antes del procesamiento del adaptador lógico y para la conexión con la Base de Conocimiento, respectivamente.

De igual manera cabe mencionar la herencia de Select_Response sobre LogicAdapter, pues esta clase contiene las variables y métodos necesarios para seleccionar y medir el nivel de fiabilidad de una respuesta frente a cada declaración de entrada de los usuarios, a partir de un conjunto de posibles respuestas que se obtendrán previamente de la clase Search.

2.6. Diagrama de la Base de Datos

La tabla 3 muestra las colecciones creadas para la base de datos, por un lado, las colecciones iniciadas con el prefijo BC_, almacenarán la información acerca de los trámites de las agencias de tránsito separadas por el nombre de la ciudad, mientras que la colección Statements almacenará todas las plantillas tipo pregunta-respuesta.

Tabla 3. Colecciones Base de Conocimiento

Colección	**Descripción**
Statements	Almacena conversaciones de usuario tanto preguntas como sus respectivas respuestas.
BC_Quito	Almacena los datos tipo JSON obtenidos mediante el proceso de Web scraping del sitio web de la Agencia de Tránsito de Quito.
BC_Guayaquil	Almacena los datos tipo JSON obtenidos mediante el proceso de Web scraping de un sitio web de la Agencia de Tránsito de Guayaquil

Nota: En esta tabla se muestra las colecciones almacenadas en la base de conocimiento y su descripción.
Elaborado por: Bryan Licto y Miguel Saritama

2.6.1. Metamodelo conceptual

Metamodelo conceptual BC_Quito

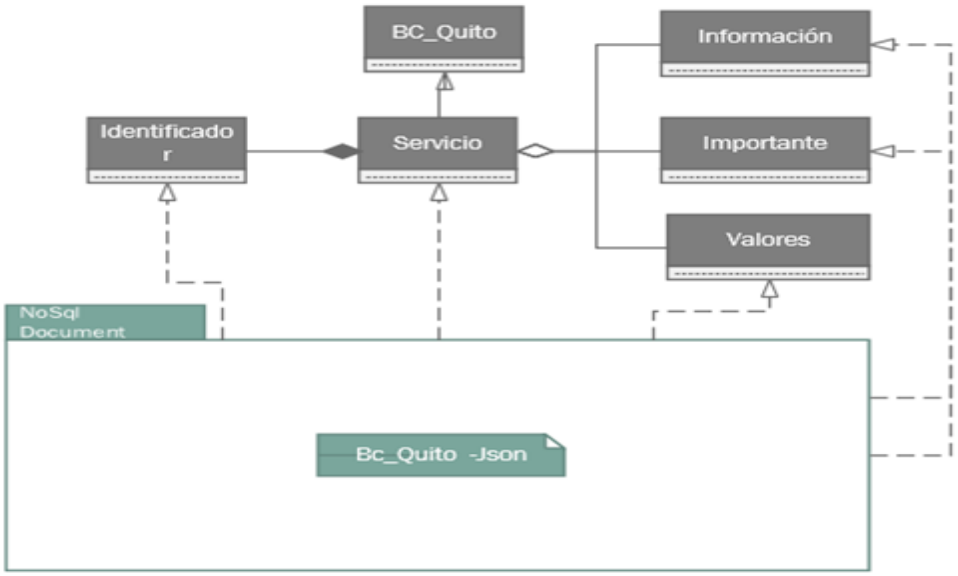

Figura 7: Metamodelo conceptual de la colección BC_Quito
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 7, se puede apreciar el tipo de documento creado en la colección BC_Quito, donde cada uno de ellos es un servicio compuesto por un identificador, al cual se le agrega los parámetros información, importante y valores, es decir, cada servicio debe tener al menos una identificación y puede o no tener contenido en información, importante y valores.

Cabe mencionar que los parámetros previamente mencionados, pueden variar dependiendo la información del sitio web de una agencia de tránsito.

2.6.2. Diseño Lógico de la Base de Datos

El diseño lógico de las colecciones determinadas en la figura 8, permite cumplir con las funcionalidades establecidas en los requerimientos y es la base de datos definida para el desarrollo de este prototipo en la cual se va a procesar y almacenar todo tipo de información.

Modelo lógico de la base de conocimiento

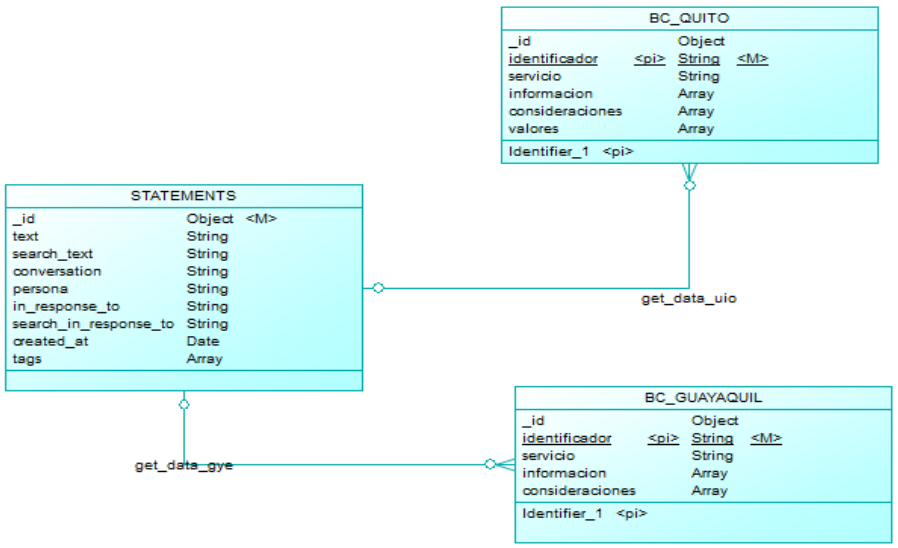

Figura 8: Modelo lógico de la base de conocimiento
Elaborado por: Bryan Licto y Miguel Saritama

2.7. Diseño del flujo del proceso conversación

Se establece un proceso de conversación, para que el prototipo pueda simular y mantener la conversación con un usuario, enviando respuestas acordes a las consultas realizadas por el mismo.

Proceso de conversación del prototipo

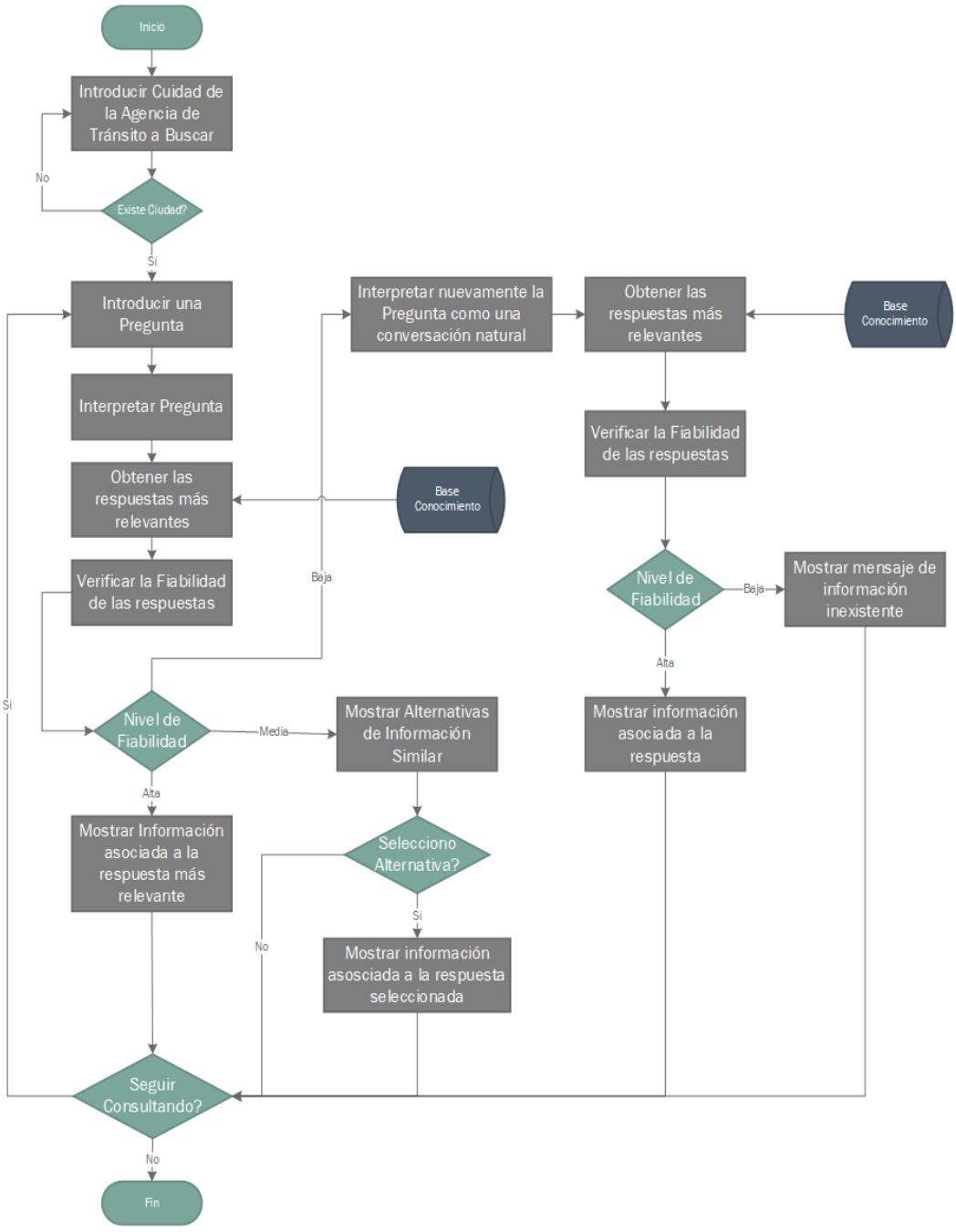

Figura 9: Diagrama de flujo del proceso de conversación
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 9, se observa el proceso de conservación diseñado, en el que, una vez seleccionada la ciudad por el usuario, se podrá generar una consulta al sistema. Este realizará una comparación de dicha consulta en la base de conocimiento y generará un conjunto de posibles respuestas, las cuales, serán comparadas entre sí a través de los niveles de fiabilidad detallados en la tabla 4.

Tabla 4. Nivel de fiabilidad

Nivel de fiabilidad	**Porcentaje**
Alta	Mayor al 90%
Media	Entre el 90% y 70%
Baja	Menor al 70%

Nota: Esta tabla contiene los niveles de fiabilidad al comparar la consulta del usuario con lo almacenado en la base de conocimiento.
Elaborado por: Bryan Licto y Miguel Saritama

Al tener un alto nivel de fiabilidad, se mostrará la información más similar a la solicitada por el usuario, por el contrario, con nivel medio de fiabilidad, se procede a mostrar una lista de alternativas similares a su consulta. Si alguna opción de esta lista es seleccionada por el usuario, se mostrará la información vinculada a la misma. Con ello si un usuario realiza una búsqueda que contenga una frase similar o exacta a la vinculada y guardada por otro usuario, se desplegará inmediatamente su información por su alto nivel de fiabilidad.

Con un nivel bajo de fiabilidad, se generará un nuevo conjunto de posibles respuestas manteniendo una conversación natural, las cuales están basadas en saludos y respuestas coherentes. Para ello se realiza una comparación para determinar el nivel de fiabilidad de la consulta detallados en la tabla 5.

Tabla 5 Nivel de fiabilidad en una conversación natural

Nivel de Fiabilidad (Conversación Natural)	Porcentaje
Alta	Mayor al 80%
Baja	Menor al 80%

Nota: Esta tabla contiene los niveles de fiabilidad con el fin mantener una conversación natural con el usuario si se tiene una fiabilidad baja obtenida de la tabla 4.
Elaborado por: Bryan Licto y Miguel Saritama

Para este caso en concreto, cuando la fiabilidad sea baja, se desplegará un mensaje mencionando que no existe información sobre lo requerido, caso contrario, el sistema mantendrá una conversación con el usuario en base a estados de ánimo o saludos.

2.8. Diseño del modelo de aprendizaje automático supervisado

Un chatbot contiene una serie de patrones o plantillas para una respuesta, que se combinan entre sí para encontrar la respuesta más adecuada a las entradas de los usuarios. Con el fin de que el prototipo no solo comprenda las conversaciones, sino también las aprenda, para incrementar el número y la precisión de cada respuesta en relación con la instrucción de entrada, surge la necesidad de diseñar un modelo de aprendizaje supervisado para el desarrollo del prototipo.

Aprendizaje automático supervisado

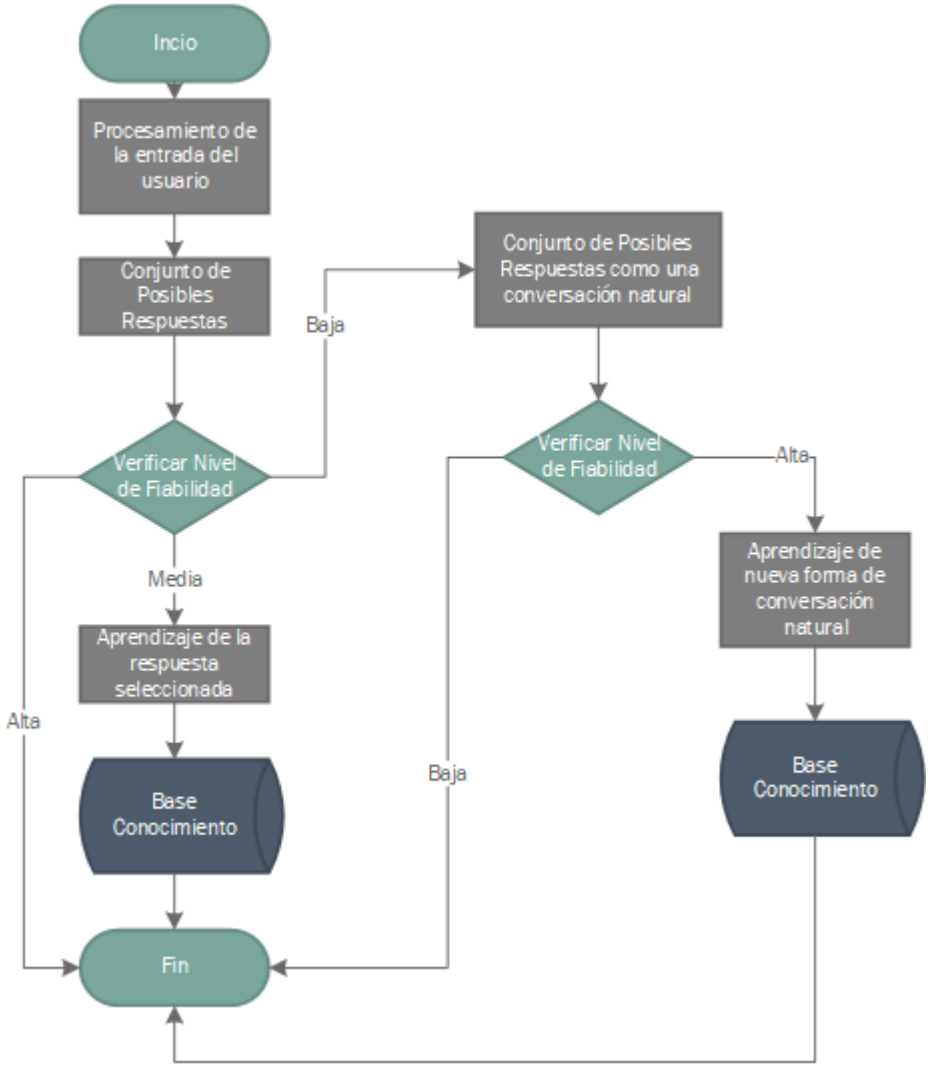

Figura 10: Diagrama del proceso de aprendizaje automático supervisado
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 10, se observa el modelo de aprendizaje supervisado del prototipo a partir del cual se puede apreciar que, una vez que las entradas de los usuarios son procesadas, este generará un conjunto de posibles respuestas obtenidas de la base de conocimiento, las cuáles serán comparadas entre sí según el nivel de fiabilidad detallados en la tabla 6.

Tabla 6. Nivel de fiabilidad del aprendizaje

Nivel de Fiabilidad	Porcentaje
Alta	Mayor al 90%
Media	Entre el 90% y 70%
Baja	Menor al 70%

Nota: Esta tabla contiene los niveles de fiabilidad al comparar la consulta del usuario con lo almacenado en la base de conocimiento
Elaborado por: Bryan Licto y Miguel Saritama

Al tener un nivel de fiabilidad alta, el prototipo no aprenderá una nueva plantilla para una respuesta, pues hay un alto grado de similitud a las plantillas preexistentes en la base de conocimiento, por el contrario, con un nivel medio de fiabilidad, este aprenderá una nueva plantilla de tipo pregunta-respuesta siempre y cuando se seleccione una opción de la lista de alternativas similares a la entrada del usuario.

Por último, al tener un nivel de fiabilidad baja, el prototipo nuevamente generará un conjunto de posibles respuestas obtenidas de la base de conocimiento considerando una conversación natural, las cuáles serán comparadas entre sí de acuerdo con un nivel de fiabilidad detallados en la tabla 7.

Tabla 7 Nivel de fiabilidad para el aprendizaje al mantener una conversación natural

Nivel de Fiabilidad Conversación Natural	Porcentaje
Alta	Mayor al 80%
Baja	Menor al 80%

Nota: Esta tabla contiene los niveles de fiabilidad al tener una fiabilidad baja obtenida de la tabla 4.
Elaborado por: Bryan Licto y Miguel Saritama

En este nuevo proceso, al tener un alto nivel de fiabilidad, el prototipo aprenderá una nueva plantilla para una respuesta, con el fin de obtener un mayor conocimiento frente a esta forma de interacción y la vez incrementar la precisión de selección de las respuestas, por el contrario, con un nivel bajo de fiabilidad, no aprenderá ningún tipo

de plantilla, pues este modelo de aprendizaje al ser supervisado está completamente limitado considerando los niveles de fiabilidad mencionados anteriormente.

Cabe recalcar que se verificó la poca fiabilidad del modelo de aprendizaje de Chatterbot, debido a que esta librería en cuanto a la interacción con el usuario, cada vez que este ingresaba una declaración, se guardaba tanto el texto ingresado como el que probablemente responde la declaración, es decir se almacenaba de manera lineal y sin considerar la fiabilidad o relación preexistente entre las declaraciones.

2.9. Diseño de la interfaz

Para el diseño de la interfaz se ha creado un botón tipo widget en la parte inferior derecha el cual, al presionarlo, permite acceder a las funcionalidades disponibles para la interacción del usuario con el prototipo. El diseño y la posición están basados en directrices sobre la experiencia de usuario con los servicios de chat.

Interfaz del prototipo

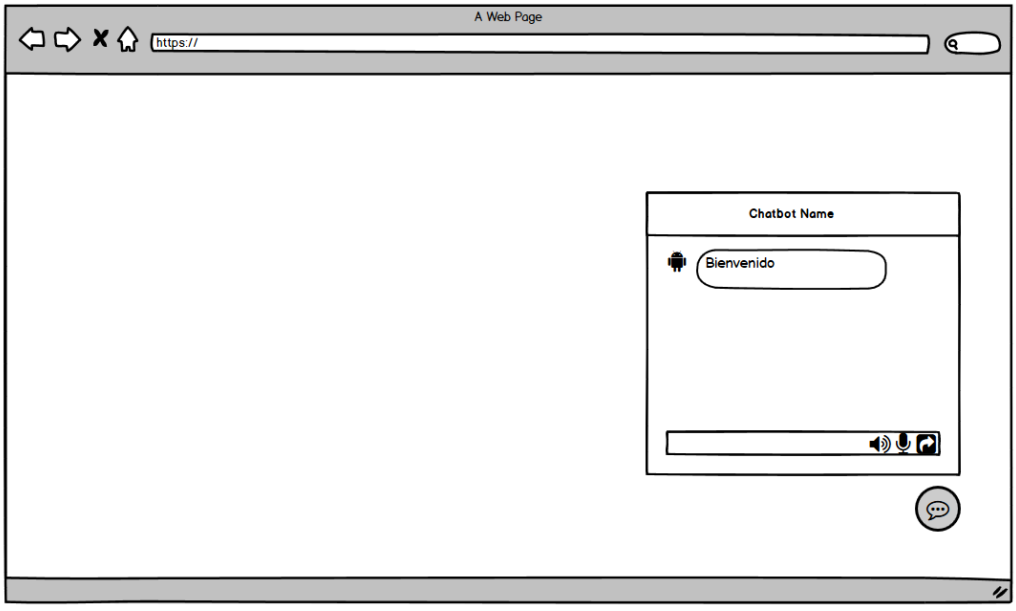

Figura 11: Diseño de la interfaz del prototipo
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 11 se establece la posición de widget en base al patrón de lectura en F especifico por el Grupo Nielsen Norman, en donde se menciona que, el modo de lectura de un usuario al entrar a un sitio formando una letra F es desde la esquina superior izquierda, en donde se posicionan los elementos importantes o relevantes del autor y en la esquina inferior derecha los elementos flotantes informativos.

CAPÍTULO 3

CONSTRUCCIÓN DEL SISTEMA

3.1. Estándares del prototipo

Para el desarrollo del prototipo se utilizó una serie de estándares que determinan la forma en que se codificó y configuró las clases, métodos, variables, colecciones y ficheros para mostrar de manera legible el programa. De esta forma otros desarrolladores pueden evidenciar el trabajo realizado o utilizarlo como referencia para futuras mejoras e integrarlos con otros proyectos.

3.1.1. Estándares para ficheros

Para los ficheros que contienen la información sobre los trámites de las agencias de tránsito, los cuales son consumidos a través de un servicio web, se definieron los siguientes estándares:

- El fichero debe ser del tipo JSON.
- El fichero debe ser nombrado en minúsculas acorde a la ciudad de la que se obtuvo la información.
- Cada objeto JSON debe tener dos claves imprescindibles: el nombre del trámite y su identificador el cual no puede repetirse.

3.1.2. Estándares de programación

Para la codificación del sistema se establecieron los siguientes estándares:

- Los nombres de las clases deben tener el estilo de escritura CamelCase (la primera letra de cada una de las palabras es mayúscula).
- Uso del guion bajo para el nombramiento de variables o funciones que contengan dos o más palabras, por ejemplo, get_response, responses_set.

- Los imports se colocan en la parte superior, en distintas líneas y antes de las variables o funciones del archivo.py.
- Uso de espacios en blanco cerca de los operadores aritméticos.
- Usar 'self' como primer parámetro de los métodos instanciados.
- Las colecciones de las ciudades deben tener el prefijo inicial “BC_” seguido de su nombre.
- Utilizar comentarios para describir las funcionalidades del sistema.

3.2. Arquitectura del prototipo

Para el desarrollo del chatbot se utilizó el estilo arquitectónico MVC (Modelo - Vista - Controlador), para separar la interfaz de usuario, la base de conocimiento y la lógica de la aplicación. Además, la librería Chatterbot maneja su propia estructura funcional para procesar, almacenar y gestionar las conversaciones.

Arquitectura del prototipo

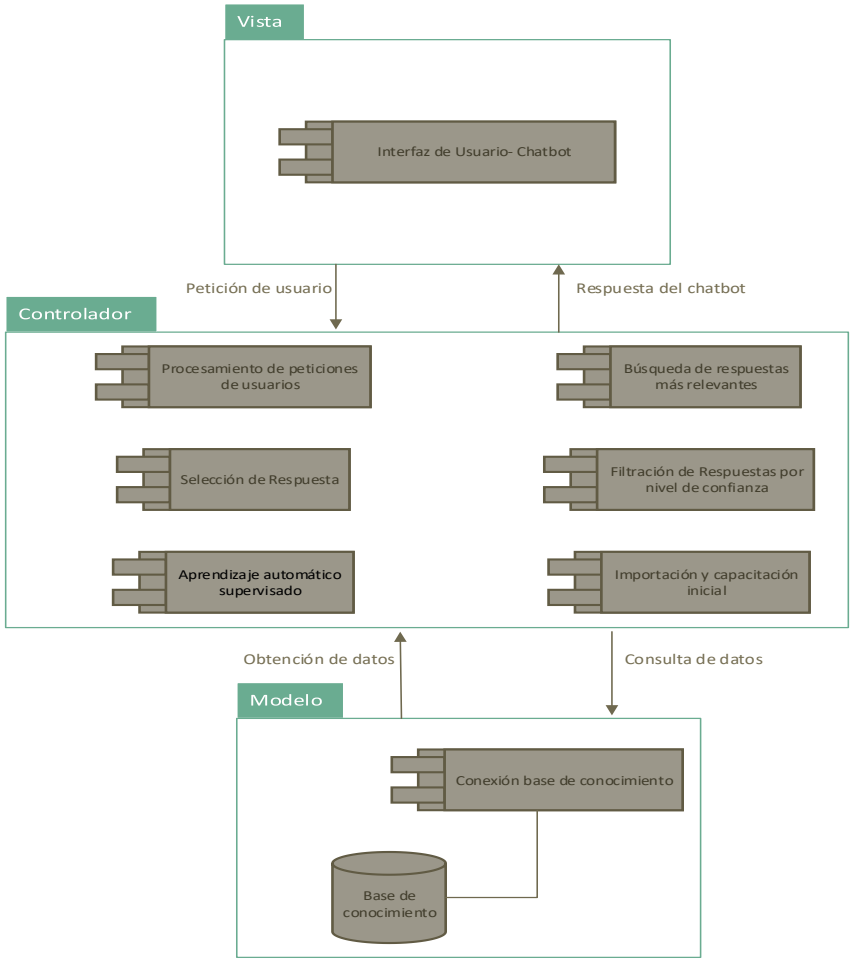

Figura 12: Diagrama de componentes del prototipo
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 12 se puede observar que a través de la interfaz de usuario se realiza una petición de tipo GET la cual es comparada mediante el algoritmo de similitud con la información de la base de conocimiento. Este proceso establece un nivel de confianza que se utiliza para seleccionar la respuesta más acorde a la petición y enviarla al usuario, por ejemplo; una persona ingresa una búsqueda sobre un trámite que desee conocer y una vez determinado el nivel de confianza, el sistema puede generar una respuesta única, un conjunto de alternativas similares (en este proceso se realiza el autoaprendizaje) o determinar que no existe información.

Cabe mencionar que se requiere de un proceso de importación de datos de los trámites por agencia para la capacitación inicial de la base de conocimiento, el cual se lo realiza al consumir un servicio web.

3.3. Código Relevante

3.3.1. Modificación de las entradas de usuario para el procesamiento del adaptador lógico

Estas funciones, a las cuales llamamos preprocesadores, modifican todas las entradas de usuarios que recibe el prototipo antes de que el adaptador lógico las procese y seleccione una respuesta.

Preprocesadores del prototipo

```
def clean_whitespace(statement):
    import re

    # Replace linebreaks and tabs with spaces
    statement.text = statement.text.replace('\n', ' ').replace('\r', ' ').replace('\t', ' ')

    # Remove any leeding or trailing whitespace
    statement.text = statement.text.strip()

    # Remove consecutive spaces
    statement.text = re.sub(' +', ' ', statement.text)

    return statement


def unescape_html(statement):


def convert_to_ascii(statement):

    import unicodedata

    text = unicodedata.normalize('NFKD', statement.text)
    text = text.encode('ascii', 'ignore').decode('utf-8')

    statement.text = str(text)
    return statement
```

Figura 13: Código para la modificación de las entradas de usuario
Elaborado por: Bryan Licto y Miguel Saritama

Como se observa en la figura 13, se realiza un proceso en segundo plano, en el cual toda petición de usuario es depurada, eliminando caracteres de espacios en blanco consecutivos, tabulaciones o caracteres especiales con el fin de evitar cualquier error en su procesamiento.

3.3.2. Algoritmo de Similitud

Una parte importante de cómo el prototipo selecciona una respuesta, es su capacidad de comparar dos peticiones entre sí, es decir; comparar las entradas de usuarios y los datos de la base de conocimiento. Hay varias maneras de realizar este proceso, una de ellas son los algoritmos de similitud de texto.

Algoritmo Levenshtein Distance

```
class LevenshteinDistance(Comparator):


    def compare(self, statement, other_statement):
        """
        Compare the two input statements.

        :return: The percent of similarity between the text of the statements.
        :rtype: float
        """

        # Return 0 if either statement has a falsy text value
        if not statement.text or not other_statement.text:
            return 0

        # Get the lowercase version of both strings
        statement_text = str(statement.text.lower())
        other_statement_text = str(other_statement.text.lower())

        similarity = SequenceMatcher(
            None,
            statement_text,
            other_statement_text
        )

        # Calculate a decimal percent of the similarity
        percent = round(similarity.ratio(), 2)

        return percent
```

Figura 14: Código del algoritmo de similitud de LevenshteinDistance
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 14, se observa el algoritmo de similitud "LevenshteinDistance", el cual compara dos objetos que contienen información, a los cuales llamamos statement, por medio de la función de Python SequenceMatcher, para obtener un porcentaje de similitud entre ellos.

3.3.3. Búsqueda de Respuestas a las declaraciones de entrada

Con la integración del algoritmo "LevenshteinDistance", el prototipo obtiene a partir de un porcentaje de confianza previamente definido, las respuestas más relevantes al comparar la entrada del usuario con los datos de la base de conocimiento.

Método de búsqueda (search)

```
def search(self, input_statement, **additional_parameters):
    self.chatbot.logger.info('Empezando Busqueda')
    input_search_text = input_statement.search_text
    ciudad= input_statement.conversation

    if not input_statement.search_text:
        self.chatbot.logger.warn(
            'No value for search_text was available on the provided input')

        input_search_text = self.chatbot.storage.tagger.get_bigram_pair_string(
            input_statement.text )

    city_conversation= ciudad+"_Conversation'"
    search_parameters = {
        'conversation': city_conversation,
        'search_in_response_to': '',
        'page_size': self.search_page_size
    }

    if additional_parameters:
        search_parameters.update(additional_parameters)

    statement_list = self.chatbot.storage.filter(**search_parameters)

    closest_match = Statement(text='')
    closest_match.confidence = 0
    self.chatbot.logger.info('Procesando Resultados de Busqueda')
    # Find the closest matching known statement
    for statement in statement_list:
        confidence = self.compare_statements(input_statement, statement)

        if confidence > 0.70:

            statement.confidence = confidence
            closest_match = statement
            self.chatbot.logger.info('Texto similar encontrado: {} con fiabilidad {}'.format(
                closest_match.text, confidence
            ))

            yield closest_match
```

Figura 15: Código para la búsqueda de repuestas más relevantes a una petición de usuario
Elaborado por: Bryan Licto y Miguel Saritama

Como se muestra en la figura 15, el método search recibe como parámetro la entrada del usuario a la cual llamamos input_statement, esta contiene el nombre de la cuidad en la que se va a realizar la búsqueda de información. A partir de esto y al definir un porcentaje de similitud del 70% como mínimo para el algoritmo de LevenshteinDistance, se obtiene las respuestas más relevantes y su nivel de similitud, los cuales son almacenados en la variable closets_match para su posterior procesamiento.

3.3.4. Selección de Respuestas

El adaptador lógico se utiliza para comparar las peticiones de los usuarios con las declaraciones conocidas del prototipo y una vez que se encuentra una o varias

coincidencias en función del porcentaje de confianza, se utiliza otra función para seleccionar una de las respuestas conocidas a esta declaración.

Código filtración de respuestas

```
def process(self, input_statement, additional_response_selection_parameters=None):
    #Objeto - Lista de las respuestas
    responses_set = self.search_algorithm.search(input_statement)
    # Usa la entrada como respuesta cuando no hay resultados
    selected_response = next(responses_set, input_statement)
    responses_set = self.search_algorithm.search(input_statement)
    # Busca la respueta para el valor de entrada
    reliability_status=True
    higher_reliability= selected_response.confidence
    lower_reliability= selected_response.confidence
    real_response= selected_response
    #posibles respuestas en el caso de no encontrar una mayor fiabilidad
    possible_responses_set = []
    for response in responses_set:
        # Detener la busqueda si la confidencialidad es mayor a 0.95
        # print('Posibles Respuestas:',respuesta, 'Confidencia:', respuesta.confidence )
        self.chatbot.logger.info('-->Posible Respuesta "{}"'.format(response.text))
        if response.confidence < lower_reliability:
            lower_reliability = response.confidence
        if 0.50 <= response.confidence < 0.90:
            possible_responses_set.append(response)
        if response.confidence > higher_reliability:
            higher_reliability = response.confidence
            real_response= response
        # print('Indice de Confianza:',higher_reliability)
        if response.confidence >= self.maximum_similarity_threshold:
            selected_response = response
            reliability_status=True
            break
        # print('ver:', reliability_status)
        reliability_status= False

    if not reliability_status:
        selected_response= real_response

    response_selection_parameters = create_response(self,selected_response, input_statement,additional_response_selection_parameters)

    response_list=[]
    alternative_responses_list=[]
    natural_conversation_list=[]
```

Figura 16: Código para filtrar las respuestas más relevantes a una petición de usuario
Elaborado por: Bryan Licto y Miguel Saritama

A partir del conjunto de respuestas obtenidas del método search, se agrupa las respuestas más relevantes en relación del porcentaje de similitud, como se puede observar en la figura 16. Este puede generar una respuesta única, un conjunto de alternativas similares o determinar que no existe información en la base de conocimiento. Cada una de estas opciones posteriormente se almacenan en una nueva lista acorde a la situación.

Código selección de respuestas

```
if not response_list:
    self.chatbot.logger.info('No se encontraron respuestas. ')

if response_list:
    self.chatbot.logger.info(
        'Seleccionamos respuesta  de {} respuetas opcionales.'.format(
            len(response_list)))

    response = self.select_response(
        input_statement,
        response_list,
        self.chatbot.storage)

    response.confidence = selected_response.confidence
    self.chatbot.logger.info('Respuesta seleccionada. Usando "{}"'.format(response.text))

elif alternative_responses_list:
    services_list=[]
    services=""
    for auxi in possible_responses_set:
        response_selection_parameters= get_response(self, auxi, input_statement,additional_response_selection_parameters)

        response_list2 = list(self.chatbot.storage.filter(**response_selection_parameters))
        dato = response_list2[0]
        dato = str(dato.text)
        dato = dato.replace("-informacion", "")
        dato = dato.replace("-valores", "")
        services_list.append(dato)

    new_services_list = []
    for i in services_list:
        if i not in new_services_list:
            new_services_list.append(i)

    for ra in new_services_list:
        services=ra+","+services
        services=services.rstrip(",")

    response = chatterbot.conversation.Statement(services)
    response.confidence = higher_reliability
    elif natural_conversation_list:

        self.chatbot.logger.info(
            'Seleccionamos respuesta  de {} respuetas opcionales.'.format(
                len(natural_conversation_list)))

        response = self.select_response(
            input_statement,
            natural_conversation_list,
            self.chatbot.storage)

        nc_statement="nc,"+response.text
        response = chatterbot.conversation.Statement(nc_statement)
        response.confidence = 0.1
        self.chatbot.logger.info('Respuesta seleccionada. Usando "{}"'.format(response.text))

    else:
        response = self.get_default_response(input_statement)

    return response
```

Figura 17: Código para seleccionar una respuesta en función de una lista
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 17, se puede observar que, tras generar las nuevas listas en función del porcentaje de similitud, se procede a la selección de una sola respuesta a través del método select_reponse, a excepción de la lista de alternativas similares, a la cual llamamos alternative_responses_list, donde se crea un nuevo statement con los

servicios asociados a cada alternativa, para mejorar la interacción del usuario cuando este no conozca con exactitud un servicio en específico.

Código obtención de respuestas

```
def get_response(self, selected_response, input_statement, additional_response_selection_parameters):
    self.chatbot.logger.info('Usando "{}" como respuesta de "{}" con una confidencialidad de {}'.format(
        selected_response.text, input_statement.text, selected_response.confidence
    ))

    recent_repeated_responses = filters.get_recent_repeated_responses(
        self.chatbot,
        input_statement.conversation
    )

    for index, recent_repeated_response in enumerate(recent_repeated_responses):
        self.chatbot.logger.info('{}. excluyendo las respuestas repetitivas de "{}"'.format(
            index, recent_repeated_response
        ))

    # Realizada la busqueda en la base de datos con el parametro search_text
    response_selection_parameters = {
        'search_in_response_to': selected_response.search_text,
        'exclude_text_words': self.excluded_words
    }

    alternate_response_selection_parameters = {
        'search_in_response_to': self.chatbot.storage.tagger.get_bigram_pair_string(
            input_statement.text
        ),
        'exclude_text_words': self.excluded_words
    }

    if additional_response_selection_parameters:
        response_selection_parameters.update(additional_response_selection_parameters)
        alternate_response_selection_parameters.update(additional_response_selection_parameters)

    return response_selection_parameters
```

Figura 18: Código para obtener la respuesta de un statament
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 18, se define el método get_response para obtener una respuesta a la petición del usuario una vez seleccionado el statement con más porcentaje de confianza al cual se lo denomina selected_response. Para ello se realiza la búsqueda en función del parámetro search_in_response_to en la base de conocimiento.

Código obtención de una respuesta

```
def get_random_response(input_statement, response_list, storage=None):

    from random import choice
    logger = logging.getLogger(__name__)
    logger.info('Selecting a response from list of {} options.'.format(
        len(response_list)
    ))
    return choice(response_list)
```

Figura 19: Código para seleccionar una sola respuesta de manera aleatoria
Elaborado por: Bryan Licto y Miguel Saritama

Cuando exista más de una respuesta a una petición, el método get_random_response como se observa en la figura 19, permite seleccionar una respuesta de manera aleatoria a dicha petición.

3.3.5. Entrenamiento de nuevas plantillas pregunta-respuesta

Una parte importante del prototipo es su capacitación, es decir; el proceso mediante el cual aprende y almacena nuevas plantillas tipo pregunta-respuesta en su base de conocimiento para generar diferentes tipos de respuestas a las peticiones de los usuarios.

Clase Entrenamiento del prototipo

```
class ListTrainer(Trainer):


    def train(self, conversation,city):

        previous_statement_text = None
        previous_statement_search_text = ''

        statements_to_create = []

        for conversation_count, text in enumerate(conversation):
            if self.show_training_progress:
                utils.print_progress_bar(
                    'CHAT-BOT TRAINING .......',
                    conversation_count + 1, len(conversation)
                )

            statement_search_text = self.chatbot.storage.tagger.get_bigram_pair_string(text)
            text_conversation=city+"_Conversation"
            statement = self.get_preprocessed_statement(
                Statement(
                    text=text,
                    search_text=statement_search_text,
                    in_response_to=previous_statement_text,
                    search_in_response_to=previous_statement_search_text,
                    conversation=text_conversation

                )
            )

            previous_statement_text = statement.text
            previous_statement_search_text = statement_search_text

            statements_to_create.append(statement)

        self.chatbot.storage.create_many(statements_to_create)
```

Figura 20: Código para el realizar el entrenamiento del prototipo en función de una conversación
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 20, se configuró el método train para que reciba como parámetros una plantilla pregunta-respuesta a la cual denominamos conversation y la ciudad a la que pertenece esta información, estos en conjunto con ciertos parámetros adicionales crean un nuevo statement que se almacena en la base de conocimientos.

Cabe recalcar que este método es ejecutado de dos formas: la primera de manera automática considerando un nivel de confianza en el procesamiento de respuestas cuando el prototipo interactúa con el usuario y la segunda al formar la base de conocimiento inicial.

3.3.6. Parámetros Iniciales del Adaptador Lógico

Adaptador lógico del prototipo

```
def __init__(self, chatbot, **kwargs):
    super().__init__(chatbot, **kwargs)
    from chatterbot.response_selection import get_first_response
    from Adaptador_Logico.response_selection import get_random_response

    self.search_algorithm_name = kwargs.get(
        'search_algorithm_name',
        IndexedTextSearch.name
    )

    self.search_algorithm = self.chatbot.search_algorithms[
        self.search_algorithm_name
    ]

    self.maximum_similarity_threshold = kwargs.get(
        'maximum_similarity_threshold', 0.90
    )

    # By default, select the first available response
    self.select_response = kwargs.get(
        'response_selection_method',
        get_random_response
    )

    default_responses = kwargs.get('default_response', [])

    # Convert a single string into a list
    if isinstance(default_responses, str):
        default_responses = [
            default_responses
        ]

    self.default_responses = [
        Statement(text=default) for default in default_responses
    ]
```

Figura 21: Código para definir los parámetros iniciales del adaptador lógico
Elaborado por: Bryan Licto y Miguel Saritama

Como se muestra en la figura 21, se define los parámetros iniciales para la ejecución del Adaptador lógico, siendo estos:

- **search_algorithm_name:** nombre de los métodos de búsqueda search para la obtención las respuestas más relevantes.
- **maximum_similarity_threshold:** porcentaje máximo de confianza
- **response_selection_method:** método para seleccionar una respuesta en el caso de que el prototipo encuentra más de una.
- **default_responses:** respuesta que es devuelta cuando el adaptador lógico no encuentra resultados a una petición.

3.3.7. Respuestas del chatbot basado en el nivel de confianza

Esta funcionalidad determina el nivel de confianza que se obtiene de la entrada proporcionada por un usuario al realizar una búsqueda y generar una respuesta dependiendo del rango alcanzado.

Código respuestas del chatbot

```
@app.route("/get")
def get_bot_response():
    userText = request.args.get('msg')
    res_ciud= request.args.get('ciudad')
    cuid = res_ciud
    urlcoleccion = "BC_"+cuid
    search_text = userText
    userText = Statement(userText)
    userText.conversation=cuid
    userText.search_text = search_text
    if userText == 'salir':
        return str("Espero haberte ayudado con tus inquietudes. Gracias por usar Chat-Bot")
    else:
        # parametro de salida una vez realizada la busqueda
        bot_response = bot.get_response(userText)
        # print("data: " + str(bot_response.confidence))
        if bot_response.confidence >= 0.90:
            # impresion de la respuesta
            # print("Bot--> ", bot_response, '-----', bot_response.confidence)
            ad = str(bot_response.text)
            mydoc = ""
            if get_data_high_reliability(urlcoleccion, ad) is not False:
                mydoc = get_data_high_reliability(urlcoleccion, ad)
            listaservicios = {'detalles': []}
            # impresion de la consulta
            for x in mydoc:
                listaservicios['detalles'].append(x)
            enviar_sms = json.dumps(listaservicios)

        elif 0.50 <= bot_response.confidence and 0.90 > bot_response.confidence:
            adicional = str(bot_response.text)
            adicional = adicional.split(',')
            listaservicios = {'tramites': []}
            for ad in adicional:
                mydoc = ""
                if get_data_medium_reliability(urlcoleccion, ad) is not False:
                    mydoc = get_data_medium_reliability(urlcoleccion, ad)
                for x in mydoc:
                    listaservicios['tramites'].append({ad:[x['servicio']]})
            listaservicios['tramites'].append({'res_usuario': [userText.search_text]})
            enviar_sms = json.dumps(listaservicios)
        else:
            nc_bot_response = bot_response.text.split('-')
            print(nc_bot_response)
            if nc_bot_response[0] == "nc":
                respuesta_bot = {"respuestas": [{"opciones": [nc_bot_response[1]]}]}
                enviar_sms = json.dumps(respuesta_bot)
                print(userText.text)
                if float(nc_bot_response[2]) < 0.90:
                    # Entrenamiento nc basado en el nivel de confianza
                    trainer = ListTrainer(bot)
                    trainer.train([userText.text, str(nc_bot_response[1])], "Natural")

            else:
                agencia=""
                if cuid=='quito':
                    agencia="Agencia Metropolitana de Tránsito"
                if cuid == 'guayaquil':
                    agencia = "Autoridad de Tránsito Municipal"
                lista_res = [
                    'Disculpa. ¿Puedes ser más específico?. Recuerda que solo te puedo ayudar con temas relacionadas a la '+agencia,
                    'Que lástima, no he podido encontrar nada relacionado con tu busqueda. Recuerda que solo te puedo ayudar con temas relacionadas
                    '¿Podrías ser un poco más específico con tu pregunta?. Recuerda que solo te puedo ayudar con trámites a la '+agencia]
                respuesta_bot = {"respuestas": [{"opciones": [random.choice(lista_res)]}]}
                enviar_sms = json.dumps(respuesta_bot)
        return str(enviar_sms)
```

Figura 22: Código para el envío de respuestas según el nivel de confianza
Elaborado por: Bryan Licto y Miguel Saritama

Como se puede observar en la figura 22 al recibir una solicitud de tipo GET, el sistema obtiene el texto de búsqueda del usuario y la ciudad donde se ejecuta la consulta, seguido de esto se hace un llamado al método get_response que establece tres niveles de confianza: alta (mayor a 90), media (mayor o igual 70 y menor que 90) o baja (menor que 70). En el primer caso se procede a enviar toda la información del trámite solicitado, en el segundo se genera una lista de los trámites similares a al texto de entrada y en el último caso al no encontrar coincidencia alguna, se envía una respuesta aleatoria almacenada en una lista que hacen referencia a que el chatbot no pudo encontrar lo solicitado por el usuario.

3.3.8. Entrenamiento al chatbot basado en la entrada proporcionada por el usuario

Este código permite obtener los parámetros necesarios para el entrenamiento del chatbot mediante el método GET. Este método solo se ejecuta cuando el usuario elije una de las opciones presentadas por el sistema al encontrar varias coincidencias en su búsqueda.

Código para el entrenamiento del prototipo

```
@app.route("/train")
def get_train_response():
    respUser = request.args.get('respUser')
    res_ciud = request.args.get('ciudad')
    id = request.args.get('id')
    dict=convertion_dict(respUser)
    urlcoleccion = "BC_" + res_ciud
    # Entrenamiento basado en el entrada de texto realizada por el usuario
    if(grammar(str(dict))):
        trainer = ListTrainer(bot)
        trainer.train([respUser, id], res_ciud)
    # Respuesta despues del entrenamiento
    mydoc = ""
    if get_data_trainer(urlcoleccion, id) is not False:
        mydoc = get_data_trainer(urlcoleccion, id)
    listaservicios = {'detalles': []}
    # impresion de la consulta
    for x in mydoc:
        listaservicios['detalles'].append(x)
    enviar_sms = json.dumps(listaservicios)
    # print(json.dumps(listaservicios))
    return enviar_sms
```

Figura 23: Código del entrenamiento al chatbot al seleccionar una alternativa de información
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 23 se muestra como el sistema recibe tres variables importantes que son enviadas desde la interfaz de usuario. La primera es respUser, quien contiene la entrada del usuario cuando realiza una búsqueda y genera una lista de posibles coincidencias, la segunda es res_ciudad quien hará referencia a la ciudad y la última es el id que hace referencia al identificador del trámite seleccionado por el usuario en la lista de coincidencias. Una vez definidas estas variables se procede a llamar al método train para ejecutar el entramiento en base a la ciudad donde se realizó la consulta.

3.3.9. Servicio web

El servicio web está construido en PHP en base al protocolo REST, el cual mediante el método GET se proporciona el acceso a los ficheros tipo JSON separados por ciudad que contienen la información relacionada a sus trámites.

Verificación de datos JSON a través del servicio web

```php
//obtenemos los ficheros sucesivamente
while ($file = readdir($dir))
{
    if (!is_dir($file))
    {
        //Verificamos si existe o no el recurso solicitado
        if (strcmp($file, $city)== 0)
            echo json_decode(json_file($city));
        else
            header("Status: 404 Not Found");
    }
}
```

Figura 24: Obtención de recursos tipo JSON mediante un servicio web
Elaborado por: Bryan Licto y Miguel Saritama

En el código de la figura 24 se muestra cómo se envía la información de la ciudad solicitada en formato JSON. Si el recurso no es localizado se envía un mensaje de error.

3.3.9.1 Obtención de datos de las agencias mediante un servicio web

El objetivo de este código es obtener la información sobre los trámites de las agencias de tránsito por ciudad mediante un servicio web. Estos ficheros se encuentran en formato JSON y tienen como nombre su ciudad.

Obtención de datos JSON a través del servicio web

```
cities = ['quito', 'guayaquil']
base_url = 'https://transitoec.herokuapp.com/servicio.php?ciudad='
# Realiza una busqueda de los recursos por ciudad
for city in cities:
    r = requests.get(base_url + city)
    # Verificar la disponibilidad del recurso HTTP
    if r.status_code == 200:
        # Ejecutar la creación de colecciones
        create_collec_info(city)
        data_train(city)
    else:
        print("No se encontró el recurso para: " + city)
```

Figura 25: Código para la obtención de datos de las agencias por ciudad mediante un servicio web.
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 25 se crea una lista de ciudades a obtener información, el sistema realiza una solicitud por ciudad para verificar si existe o no el recurso, la cual, si se obtiene sin problemas se genera un mensaje con código 200 de lo contrario muestra un mensaje de error.

3.3.9.2 Creación de colecciones y entrenamiento inicial

Para la creación de las colecciones por ciudad se hace uso de los métodos create_collecion_info y data_statement, los cuales procesan la información obtenida mediante el servicio web.

Código del Servicio web

```
# Método para el creación de la Coleccion Statement basasa en la plantilla pregunta-respuesta
def data_statement(city):
    my_dict = json.loads(r.content)
    cont = 0
    for i in my_dict:
        data = []
        for (k, v) in my_dict[cont].items():
            if k == 'identificador':
                data.append(v)
            elif k == 'servicio':
                data.append(v)
        trainer.train([data[1], data[0]], city)
        cont += 1

# Método para la creación de la coleccion que contendrá la informacion de cada ciudad
def create_collec_info(city):
    city_name = prefix + city
    # Asignar el json decoficado del recurso HTTP obtenido
    my_dict = json.loads(r.content)
    # Verificar la existencia de colecciones similares
    if city_name not in db.list_collection_names():
        if isinstance(my_dict, list):
            try:
                db[city_name].insert_many(my_dict)
                print("Se ha creado la coleccion " + city_name + " correctamente.")
            except pymongo.errors.BulkWriteError as e:
                print(e.details['writeErrors'])
        else:
            db[city_name].insert_one(my_dict)
    else:
        print("No se pudo completar lo solicitado. Ya existe una colección con el nombre: " + city_n
```

Figura 26: Código para la creación de colecciones y statament por el servicio web
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 26 se muestra como ambos métodos reciben como parámetro el nombre de la ciudad, pero tienen distintas funciones. En el caso de create_collec_info, este crea una colección con un nombre referente a la ciudad, el cual contiene toda la información relacionada a los trámites de la agencia de esa localidad, por otro lado, data_statement realiza una búsqueda de los campos identificador y servicios que hacen referencia al id y al nombre del trámite respectivamente, a continuación, se ejecuta el método train para el entrenamiento inicial del prototipo a partir de los campos obtenidos previamente.

CAPÍTULO 4

PRUEBAS

Las pruebas realizadas al prototipo permiten verificar que todas sus funcionalidades actúen de manera adecuada acorde a los requerimientos planteados, cada una de ellas incluyen distintos métrica y escenarios con la finalidad de verificar su nivel de calidad y desempeño.

4.1. Plan de pruebas

Con el fin de evaluar la calidad del prototipo desarrollado, en cuanto a sus funcionalidades se seleccionó un conjunto de características tales como la usabilidad, confiabilidad, funcionalidad, portabilidad y eficiencia, en base al estándar ISO 9126.Cada una de ellas es medida por una serie de métricas y en distintos escenarios como se puede observar en la tabla 8.

Tabla 8. Plan de pruebas

Característica	Métrica	Tipo de Prueba	Técnica	Instrumento
Usabilidad	Nivel de entendimiento del manejo de las funciones del chatbot	Escala de Usabilidad del Sistema	Encuesta	Cuestionario
Eficiencia	Tiempo de respuesta del prototipo	Pruebas de Rendimiento	Herramienta Testing	Jmeter Apache
Eficiencia	Número de peticiones máximas	Pruebas de Estrés	Herramienta Testing	Jmeter Apache
Portabilidad	Número de funcionalidades del chatbot compatibles con los navegadores web.	Pruebas de Compatibilidad	Tabla Comparativa	Tabla Comparativa
Confiabilidad	Porcentaje de exactitud de palabras recibidas a través del comando de voz con un mínimo aceptable del 90%	Pruebas Reales	Escenarios de Pruebas	Escenarios
Confiabilidad	Numero de iteraciones realizadas con el agente hasta obtener la respuesta requerida por el usuario con un máximo de 5	Pruebas Reales	Escenarios de Pruebas	Escenarios

Confiabilidad	Tiempo necesario para obtener una respuesta especifica requerida por el usuario con un máximo de 2min	Pruebas Reales	Escenarios de Pruebas	Escenarios
Confiabilidad	Porcentaje de fiabilidad de las respuestas obtenidas del prototipo con relación a un trámite en específico con un mínimo del 80%	Pruebas Reales	Escenarios de Pruebas	Escenarios
Funcionalidad	Funcionamiento completo de las características del prototipo	Escala de Usabilidad del Sistema	Encuesta	Cuestionario

Nota: Esta tabla contiene las características en base al estándar ISO 9126, métricas, técnicas e instrumentos utilizados para las pruebas del sistema.
Elaborado por: Bryan Licto y Miguel Saritama

4.2. Pruebas de usabilidad

Para medir la experiencia del usuario en cuanto a las funcionalidades del prototipo y obtener una retroalimentación para el mejoramiento de este, se realizó una encuesta, en la cual participaron 10 personas, cada una de ellas debía por lo menos tener un mínimo de conocimiento sobre los trámites que se realizan en las agencias de tránsito.

Pregunta 1: Basado en su experiencia en relación con el manejo del chatbot. Califique el uso del reconocimiento de voz**.**

Resultado de la pregunta 1, encuesta sobre usabilidad

Figura 27: Resultados a encuesta realiza sobre el uso del reconocimiento de voz
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 27 se observa que el 80% de los usuarios calificaron esta funcionalidad como excelente y solo un 20% como muy buena. Ningún usuario lo consideró como bueno o regular, lo que demuestra que la funcionalidad del reconocimiento de voz desarrollada cumple con las expectativas del usuario.

Pregunta 2: Basado en su experiencia con relación al manejo del chatbot. Califique el uso del reproductor de texto.

Resultado de la pregunta 2, encuesta sobre usabilidad

Figura 28: Resultados a encuesta realiza sobre el uso del reproductor de texto
Elaborado por: Bryan Licto y Miguel Saritama

Se puede apreciar en la figura 28, que todos los usuarios calificaron esta funcionalidad como excelente, lo que demuestra que la funcionalidad del reproductor de texto en cuanto a las repuestas enviadas por el chatbot cumple con las expectativas del usuario.

Pregunta 3: Basado en su experiencia en relación con el manejo del chatbot, califique la forma de acceso al sistema.

Resultado de la pregunta 3, encuesta sobre usabilidad

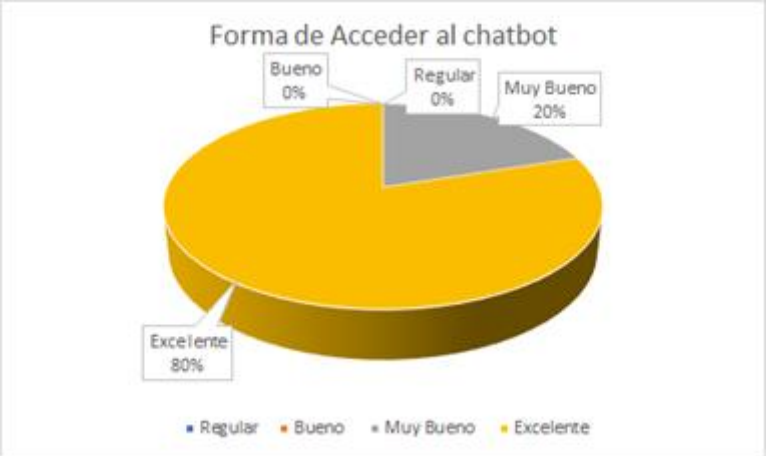

Figura 29: Resultados a encuesta realiza sobre el acceso al sistema
Elaborado por: Bryan Licto y Miguel Saritama

Como se muestra en la figura 29, el 80% de los usuarios calificaron como excelente la forma de acceder al sistema y solo un 20% lo consideró como muy buena, lo que demuestra que la forma en que se accede al prototipo a través de un clic cumple con las expectativas del usuario.

Pregunta 4: Basado en su experiencia con relación al manejo del chatbot, ¿Los elementos tales como: botones, iconos y colores le resultaron familiares?

Resultado de la pregunta 4, encuesta sobre usabilidad

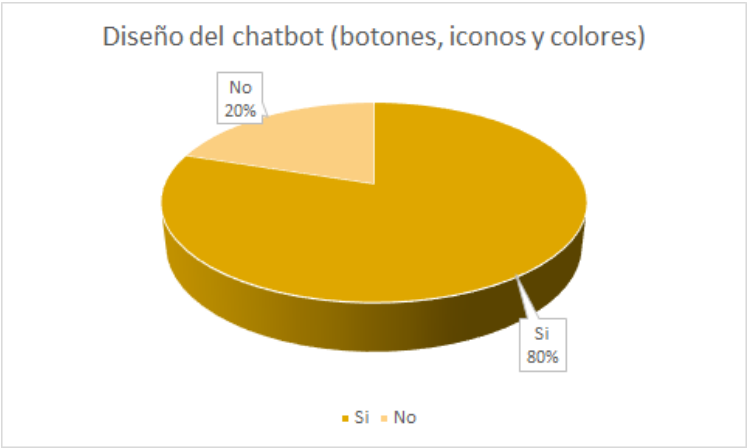

Figura 30: Resultados a encuesta realiza sobre el diseño del sistema
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 30 se puede apreciar que el 80% de los usuarios están familiarizados con los elementos del sistema y con el 20% se realizó una retroalimentación para la modificación de los iconos del prototipo, pues estos debían ser más acordes a sus funcionalidades.

Pregunta 5: Basado en su experiencia en relación con el manejo del chatbot, califique la facilidad en la que usted obtuvo información del sistema.

Resultado de la pregunta 5, encuesta sobre usabilidad

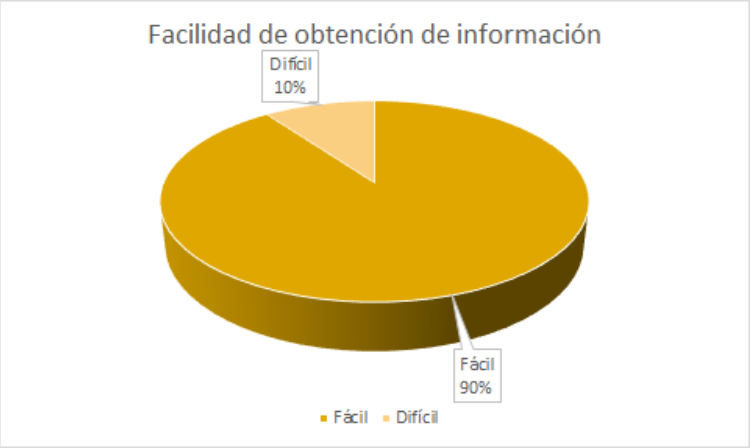

Figura 31: Resultados a encuesta realiza sobre la facilidad en la obtención de información
Elaborado por: Bryan Licto y Miguel Saritama

Se puede observar en la figura 31, que el 90% de los usuarios calificaron como "fácil", la forma de obtener información debido a que el prototipo ayuda con alternativas similares en una búsqueda.

4.3. Pruebas de confiabilidad

Estas pruebas son realizadas en distintos escenarios, para determinar el nivel de confiabilidad de la información proporcionada por el prototipo al realizar una búsqueda.

Métrica: Porcentaje de exactitud de palabras recibidas a través del comando de voz con un mínimo aceptable del 90%

Esta métrica se evaluó para medir la exactitud de palabras recibidas a través de la función del comando de voz del prototipo. Para ello se definió una tabla con los campos que se muestran continuación:

- **N°:** Hace Referencia al número de prueba.
- **Frase o palabra pronunciada por el usuario:** Entradas de usuarios a través del comando de voz.
- **Frase o palabra reconocida por el chatbot:** Entradas reconocidas por el prototipo al usar el comando de voz.
- **% deseado:** Porcentaje mínimo establecido para la evaluación de la métrica.
- **% de similitud:** Porcentaje obtenido al comparar las entradas de usuarios con las entradas reconocidas por el prototipo al usar el comando de voz.

Tabla 9. Pruebas de fiabilidad del comando de voz

N°	Frase o palabra pronunciada por el usuario	Frase o palabra reconocida por el chatbot	% deseado	% de similitud
1	Disculpa me puede ayudar con información para la renovación de la matricula	Disculpa me puede ayudar con información para la renovación de la matricula	90	100
2	Necesito saber cuáles son los pasos para la revisión vehicular	Necesito saber cuáles son pasos para la revisión vehicular	90	96
3	Soy dueño de transporte comercial entonces necesito saber cuáles son requisitos para el cambio de socio	Soy dueño de transporte comercial entonces necesito saber cuáles son requisitos para el cambio de socio	90	99
4	Necesito saber cómo puedo liberar mi vehículo que estaba mal estacionado	Necesito saber cómo puedo liberar mi vehículo que estaba mal estacionado	90	100
5	Oye bot que necesito para obtener duplicado de mi matricula que se me perdió	Oye bot que necesito para obtener duplicado de mi matricula que se perdió	90	97

6	Que necesito para matricular mi vehículo	Quisiera matricular mi vehículo	90	100
7	Cuáles son los requisitos para obtener un salvoconducto	Cuáles son los requisitos para obtener un salvoconducto	90	100
8	Cuánto es la multa por mal estacionamiento	Cuánto es la multa por mal estacionamiento	90	100
9	Quisiera saber los pasos para la matriculación de mi vehículo	Quisiera saber los pasos para matriculación del vehículo	90	96
10	Necesito saber los requisitos para la revisión técnica	Necesito saber los requisitos para revisión técnica	90	98

Nota: Esta tabla contiene el porcentaje de similitud entre la frase o palabra ingresada por usuario y la reconocida por el API de reconocimiento de voz.
Elaborado por: Bryan Licto y Miguel Saritama

Análisis de Resultados:

Una vez realizadas las pruebas de confiabilidad, en cuanto a la funcionalidad del comando de voz y al obtener un promedio de los porcentajes de los escenarios de pruebas detallados en la tabla 9, se verificó que el prototipo cumple con la métrica, pues alcanza un promedio del 98,6%, el cual es superior al margen del 90% considerado como mínimo en la evaluación.

Métrica: Numero de iteraciones realizadas con el prototipo hasta obtener la respuesta requerida por el usuario con un máximo de 5.

La evaluación de esta métrica permitió contabilizar el número de iteraciones realizadas en el prototipo hasta obtener la respuesta requerida por el usuario y su confiabilidad al tener un previo conocimiento de la información solicitada. Para ello se definió una tabla con los campos que se muestran continuación:

- **N°:** hace referencia al número de prueba.
- **Trámite requerido:** nombre del trámite solicitado.

- **Iteraciones del usuario:** iteraciones del usuario con el prototipo hasta encontrar la información solicitada.
- **Número de iteraciones:** número de iteraciones realizadas por el usuario.
- **Observaciones:** describe cada una de las acciones relevantes o posibles errores en la ejecución de la prueba.

Tabla 10 Número de iteraciones realizadas por el usuario

N°	Trámite requerido	Iteraciones del usuario	Número de iteraciones	Observaciones
1	Renovación de matricula	**1) Ingreso cuidad: Quito** **2) Ingreso:** Disculpa me puede ayudar con información para la renovación de la matricula **3)** Selección de alternativas a la información solicitada	3	Ninguna
2	Pasos para la revisión vehicular	**1) Ingreso cuidad: Quito** **2) Ingreso:** Necesito saber cuáles son pasos para la revisión vehicular **3)** Selección de alternativas a la información solicitada	3	Ninguna
3	Transporte comercial requisitos para el cambio de socio	**1) Ingreso cuidad: Quito** **2) Ingreso:** Soy dueño de transporte comercial entonces necesito saber cuáles son requisitos para el cambio de socio **3)** Selección de alternativas a la información solicitada	3	Ninguna
4	Liberación de vehículos por abandono o sin documentos o mal estacionado	**1) Ingreso cuidad:** Quito **2) Ingreso:** Necesito saber cómo puedo liberar mi vehículo que estaba mal estacionado **3)** Selección de alternativas a la información solicitada	3	Ninguna
5	Duplicado de matrícula por pérdida o robo servicio público o comercial	**1) Ingreso cuidad:** Quito **2) Ingreso:** Disculpa bot que necesito para obtener duplicado de mi matricula que se perdió **3)** Selección de alternativas a la información solicitada(Ninguna) **4) Ingreso:** Me puedes ayudar con información para obtener un duplicado de matrículas por perdida	5	Se planteó nuevamente la pregunta debido a que el chatbot no reconoció la primera entrada

		5) Selección de alternativas a la información solicitada		
6	Revisión vehicular	**1) Ingreso cuidad:** Quito **2) Ingreso:** Que necesito para pasar la revisión vehicular mi auto **3)** Selección de alternativas a la información solicitada	3	Ninguna
7	Transporte comercial requisitos para el cambio de socio	**1) Ingreso cuidad:** Quito **2) Ingreso:** Cuales son los requisitos para el cambio de socio de un trasporte comercial **3)** Selección de alternativas a la información solicitada	3	Ninguna
8	Renovación de matricula	**1) Ingreso cuidad: Quito** **2) Ingreso:** Que necesito para la renovación de la matricula	2	Ninguna
9	Certificado único vehicular	**1) Ingreso cuidad:** Quito **2) Ingreso:** Como obtengo el certificado único vehicular	2	Ninguna
10	Cambio de características motor	1) **Ingreso cuidad:** Quito 2) **Ingreso:** Como puedo realizar el cambio de las características de motor de mi vehículo **3)** Selección de alternativas a la información solicitada	3	Ninguna

Nota: Esta tabla muestra el número de iteraciones necesarias para tener información coherente a una solicitud.
Elaborado por: Bryan Licto y Miguel Saritama

Análisis de Resultados:

Una vez finalizadas las pruebas detalladas en la tabla 10, se comprobó que el prototipo requiere de por lo menos 2 a 3 iteraciones con el usuario para proporcionar una información coherente a las solicitudes realizadas. Sin embargo, cuando no encontró información relacionada a la petición, se tuvo que replantear la pregunta y con ello se incrementó el número de iteraciones a 5.

Métrica: Tiempo necesario para obtener una respuesta especifica requerida por el usuario con un máximo de 2min.

La evaluación de esta métrica permitió conocer el tiempo de iteración con el prototipo hasta obtener una respuesta requerida por el usuario, de igual manera su confiabilidad al tener un previo conocimiento de la información solicitada. Para ello se definió una tabla con los campos que se muestran continuación:

- **N°:** hace referencia al número de prueba.
- **Preguntas planteadas:** entradas de usuarios como consultas al prototipo.
- **Proceso:** iteraciones del usuario con el prototipo hasta encontrar la información solicitada.
- **Tiempo:** tiempo máximo hasta obtener la información requerida por el usuario.
- **Observaciones:** describe cada una de las acciones relevantes o posibles errores en la ejecución de la prueba.

Tabla 11 Tiempo de iteración del usuario con el sistema hasta hallar una respuesta

N°	Trámite requerido	Iteraciones del usuario	Tiempo de iteración	Observaciones
1	Renovación de matricula	**1) Ingreso cuidad:** Quito **2) Ingreso:** Disculpa me puede ayudar con información para la renovación de la matricula **3)** Selección de alternativas a la información solicitada	00:00:56	Ninguna
2	Pasos para la revisión vehicular	**1) Ingreso cuidad:** Quito **2) Ingreso:** Necesito saber cuáles son pasos para la revisión vehicular **3)** Selección de alternativas a la información solicitada	00:00:31	Ninguna
3	Transporte comercial requisitos para el cambio de socio	**1) Ingreso cuidad:** Quito **2) Ingreso:** Soy dueño de transporte comercial entonces necesito saber cuáles son requisitos para el cambio de socio **3)** Selección de alternativas a la información solicitada	00:00:54	Ninguna

4	Liberación de vehículos por abandono o sin documentos o mal estacionado	**1) Ingreso cuidad:** Quito **2) Ingreso:** Necesito saber cómo puedo liberar mi vehículo que estaba mal estacionado **3)** Selección de alternativas a la información solicitada	00:00:43	Ninguna
5	Duplicado de matrícula por pérdida o robo servicio público o comercial	**1) Ingreso cuidad:** Quito **2) Ingreso:** Disculpa bot que necesito para obtener duplicado de mi matricula que se perdió **3)**Selección de alternativas a la información solicitada(Ninguna) **4)Ingreso:** Me puedes ayudar con información para obtener un duplicado de matrículas por perdida **5)**Selección de alternativas a la información solicitada	00:01:19	Se planteó nuevamente la pregunta debido a que el chatbot no reconoció la primera entrada
6	Revisión vehicular	**1) Ingreso cuidad:** Quito **2) Ingreso:** Que necesito para pasar la revisión vehicular mi auto **3)** Selección de alternativas a la información solicitada	00:0:29	Ninguna
7	Transporte comercial requisitos para el cambio de socio	**1) Ingreso cuidad:** Quito **2) Ingreso:** Cuales son los requisitos para el cambio de socio de un trasporte comercial **3)** Selección de alternativas a la información solicitada	00:0:36	Ninguna
8	Renovación de matricula	**1) Ingreso cuidad: Quito** **2) Ingreso:** Que necesito para la renovación de la matricula	00:0:29	Ninguna
9	Certificado único vehicular	**1) Ingreso cuidad:** Quito **2) Ingreso:** Como obtengo el certificado único vehicular	00:0:28	Ninguna
10	Cambio de características motor	**1) Ingreso cuidad:** Quito **2) Ingreso:** Como puedo realizar el cambio de las características de motor de mi vehículo 3) Selección de alternativas a la información solicitada	00:0:32	Ninguna

Nota: Esta tabla muestra el tiempo de iteración del usuario con el sistema hasta hallar una respuesta.
Elaborado por: Bryan Licto y Miguel Saritama

Análisis de Resultados:

Una vez finalizadas las pruebas definidas en la tabla 11, se verificó que un usuario requiere de por lo menos 29 segundos de iteración con el sistema hasta obtener una respuesta a su petición. En los casos en que la pregunta ingresada no generó ningún resultado, se tuvo que realizar iteraciones adicionales, por ende, hubo un incremento en el tiempo de iteración considerando un máximo de 1 minuto con 29 segundos.

Métrica: Porcentaje de fiabilidad de las respuestas obtenidas del prototipo con relación a un trámite en específico con un mínimo del 80%.

Al evaluar esta métrica se pudo apreciar el porcentaje de fiabilidad de las respuestas procesadas por el prototipo, al tener un previo conocimiento de la información solicitada. Para ello se definió una tabla con los campos que se muestran continuación

- **N°:** hace referencia al número de prueba.
- **Preguntas planteadas:** entradas de usuarios como consultas al prototipo.
- **Proceso:** iteraciones del usuario con el prototipo hasta encontrar la información solicitada.
- **Estado:** dependiendo de la respuesta hallada en cada prueba, se definió dos estados: encontrada cuando se obtiene la información que se solicitó y no encontrada cuando no se obtuvo la información.
- **Observaciones:** describe cada una de las acciones relevantes o posibles errores en la ejecución de la prueba.

Escenario 1

Como escenario de prueba se utilizó al trámite: cambio de servicio de particular a público de la agencia de tránsito.

Tabla 12 Fiabilidad de las respuestas obtenidas del prototipo con relación a un trámite específico

Trámite buscado:		Cambio de servicio particular a público		
N°	**Preguntas Planteadas**	**Iteraciones del usuario**	**Estado**	**Observaciones**
1	Cambio de servicio particular a público	**1) Ingreso:** cambio de servicio particular a público - Respuesta hallada	Encontrada	Ninguna
2	Necesito información sobre el proceso para el cambio de particular a público	**1) Ingreso:** Necesito información sobre el proceso para el cambio de particular a público 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrada	Ninguna
3	Cuál es el proceso para el cambio de servicio de particular a público	**1) Ingreso:** Cual es el proceso para el cambio de servicio de particular a público 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrada	Ninguna
4	Para el cambio de particular a público que no mas necesito saber	**1) Ingreso:** Para el cambio de particular a público que no mas necesito saber 2) Menú de alternativas a la información solicitada - Respuesta hallada	Encontrada	Ninguna
5	Soy el dueño de un vehículo particular como puedo cambiar a público	**1) Ingreso:** Soy el dueño de un vehículo particular como puedo cambiar a público **2)** Selección de alternativas a la información solicitada - Respuesta hallada	Encontrada	Ninguna
6	Soy el dueño de un vehículo particular	**1)Ingreso:** Soy el dueño de un vehículo particular como puedo cambiar a taxi	No encontrada	Para hallar la información solicitada se

	como puedo cambiar a taxi	**2)** Selección de alternativas a la información solicitada **3)** Buscar Nuevamente		tuvo que replantear la preguntar dado que ninguna las opciones fueron las requeridas.
7	Como puedo cambiar de particular a público mi auto	**1) Ingreso**: cómo puedo cambiar de particular a público mi auto **2)** Selección de alternativas a la información solicitada - Respuesta hallada	Encontrada	Ninguna

Nota: Esta tabla muestra la fiabilidad del sistema en base a la búsqueda del trámite: cambio de servicio de particular a público de la agencia de tránsito.
Elaborado por: Bryan Licto y Miguel Saritama

Análisis de Resultados:

Una vez finalizadas las pruebas desarrolladas en la tabla 12, se verificó la confiabilidad de un 85.7% en relación con las respuestas procesadas por el prototipo, dado que, en 6 de las 7 preguntas realizadas sobre un mismo trámite, se pudo obtener la información esperada.

Escenario 2

Como escenario de prueba se utilizó al trámite: duplicado de matrícula por pérdida o robo servicio público o comercial de la agencia de tránsito

Tabla 13 Fiabilidad de las respuestas obtenidas del prototipo con relación a un trámite específico

Trámite buscado		**duplicado de matrícula por pérdida o robo servicio público o comercial**		
N°	**Preguntas Planteadas**	**Iteraciones del usuario**	**Estado**	**Observaciones**
1	Me puedes ayudar con información sobre el duplicado de matrículas por perdida	1) Ingreso: **Me puedes ayudar con información sobre el duplicado de matrículas por perdida** 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrado	Ninguna
2	Duplicado de matrícula por perdida	1) Ingreso: **Duplicado de matrícula por perdida** 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrado	Ninguna
3	Sabes que necesito para obtener el duplicado de una matrícula perdida	1) Ingreso: s**abes que necesito para obtener el duplicado de la matricula que está perdida** 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrado	Ninguna
4	Duplicado de matrícula por pérdida o robo servicio público o comercial	1) Ingreso: **Duplicado de matrícula por pérdida o robo servicio público o comercial** - Respuesta hallada	Encontrado	Ninguna
5	Duplicado de matrícula para servicio público	1) Ingreso: **Duplicado de matrícula para servicio público** 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrado	Ninguna
6	cómo puedo sacar un duplicado de una	1) Ingreso: **Como puedo sacar un duplicado de una matrícula robada un vehículo público**	Encontrado	Ninguna

	matrícula robada un vehículo público	2) Selección de alternativas a la información solicitada - Respuesta hallada		
7	Duplicado de matrícula por robo vehículo comercial	1) Ingreso: **Duplicado de matrícula por robo vehículo comercial** 2) Selección de alternativas a la información solicitada - Respuesta hallada	Encontrado	Ninguna

Nota: Esta tabla muestra la fiabilidad del sistema en base a la búsqueda del trámite: duplicado de matrícula por pérdida o robo servicio público o comercial de la agencia de tránsito.
Elaborado por: Bryan Licto y Miguel Saritama

Se realizó un promedio del número de iteraciones del usuario con el prototipo, donde se obtuvo como resultado 2, el cual está dentro del margen 5 iteraciones como máximo para la evaluación.

Análisis de Resultados:

Tras finalizar las pruebas definidas en la tabla 13, se comprobó la confiabilidad al 100% en relación con las respuestas procesadas por el prototipo, dado que, en las 7 preguntas realizadas sobre un mismo trámite, se pudo obtener la información esperada.

Cabe recalcar que la confiabilidad se determina de dos maneras: la primera es cuando el prototipo envía una respuesta acorde al trámite y la segunda cuando se muestra alternativas similares a la información del trámite solicitado.

4.4. Pruebas de funcionalidad

Este tipo de pruebas se realizó con el fin de comprobar si el prototipo procesa y cumple de forma adecuada cada una de sus funciones en base a los requerimientos a partir de los cuales fue desarrollado. Para ello se definió la tabla 14 con los campos que se muestran continuación:

- **N°:** hace referencia al número de prueba.
- **Prueba:** principales funciones y procesos del sistema.
- **Acciones:** describe el procedimiento necesario para poder ejecutar la prueba.
- **Estado:** dependiendo del proceso que conlleva cada prueba, se definió dos estados: completo cuando se finaliza con éxito la prueba e incompleto cuando hay errores o inconvenientes en su ejecución.
- **Observaciones:** describe cada una de las acciones relevantes o posibles errores en la ejecución de la prueba.

Tabla 14 Pruebas de funcionalidad

N°	Prueba	Acciones	Estado	Observaciones
1	Ingreso al Chatbot	Acceder al prototipo mediante un botón tipo widget ubicado en la parte inferior derecha del sitio web	Completo	Ninguna
2	Seleccionar una ciudad	Seleccionar o ingresar de una lista el nombre de la ciudad, de la cual se va a realizar la búsqueda de información	Completo	Si no se selecciona una cuidad valida, el prototipo no le deja continuar con su funcionamiento
3	Botón de reconocimiento de voz	Utilizar el botón y verificar que se procese el texto hablado en el prototipo	Completo	Ninguna
4	Botón de reproductor de texto	Utilizar el botón y verificar que se escuche el texto procesado por el prototipo	Completo	Ninguna
5	Buscar Información	Ingresar una consulta en el prototipo y verificar los resultados procesados utilizando un criterio de fiabilidad	Completo	Si la información no existe envía un mensaje de que el prototipo no puede entender la consulta planteada

6	Alternativas de información similar	Ingresar una consulta en el prototipo y verificar las opciones procesadas y presentadas tengan relación a la consulta, utilizando un criterio de fiabilidad	Completo	Si la información no existe envía un mensaje de que el prototipo no puede entender la consulta planteada
7	Aprendizaje del prototipo	Seleccionar una opción de las alternativas de información similar	Completo	Si no escoge una opción de las mostradas por el prototipo, este no aprende
8	Servicio Web	Seleccionar un documento tipo JSON para la carga de la Base de Conocimiento	Completo	Ninguna

Nota: Esta tabla muestra las pruebas de funcionalidad ejecutadas al sistema.
Elaborado por: Bryan Licto y Miguel Saritama

4.4.1. Respuestas del servicio web

Permite verificar el funcionamiento adecuado del servicio web al recibir peticiones externas y al momento de entregar un resultado específico según el caso. Para esta prueba se creó un script que permite probar estas funcionalidades con distintos escenarios.

Escenario 1: obtener un JSON de los trámites de la agencia de una ciudad

Servicio web

```python
import json, requests
from requests.exceptions import HTTPError

cities = ['quito']
base_url = 'https://transitoec.herokuapp.com/servicio.php?ciudad='

for city in cities:
    try:
        response = requests.get(base_url + city)
        #Si la respuesta fue exitosa (200), no se ejecuta la excepcion
        response.raise_for_status()
    except HTTPError as http_err:
        print(f'Error HTTP : {http_err}')  # Python 3.6
    except Exception as err:
        print(f'Ocurrió un error inesperado: {err}')  # Python 3.6
    else:
        print('Tiempo de respuesta:',response.elapsed.total_seconds()*1000,'ms')
        print('Tamaño de recurso:',len(response.content),'bytes')
        print('El recurso para '+city+', se ha cargado corretamente')
        #Se convierte el json recibido a una lista
        my_dict = json.loads(response.content)
        print(my_dict)
```

Figura 32: Código ejecutado para la obtención de un JSON proporcionado por un servicio web
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 32 se muestra el código con el cual se obtiene la información acerca de los trámites de una agencia sobre una ciudad específica la cual está almacenada en un servidor web.

Ejecución del servicio web

```
Tiempo de respuesta: 563.082 ms
Tamaño de recurso: 123049 bytes
El recurso para quito, se ha cargado corretamente
[{'identificador': 'Menu01', 'servicio': 'matriculacion vehicular', 'informacion': ['Renovación de permis
  de circulación (Matriculación Anual del vehículo)', 'Transferencia de dominio o Cambio de Propietario de
 Vehículo', 'Duplicado de matrícula', 'Emisión del Certificado Único Vehicular (gravamen, matrícula, histo
 vehicular)', 'Cambio de servicio de particular a público o viceversa.', 'Casos especiales (Remates,
 Prescripción Adquisitiva de dominio).', 'Matriculación de vehículo con cambio de motor.', 'Solicitud y er
```

Figura 33: Tiempo de respuesta y tamaño de recurso obtenido mediante el servicio web
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 33 se puede apreciar que el tiempo de respuesta en obtener el recurso para una ciudad específica es de 596 ms que contiene un tamaño de 123kB.

Escenario 2: verificar la existencia de un recurso de una ciudad específica

El script de la figura 34 permite verificar si existe un recurso solicitado sobre una ciudad, en el caso de no existir encontrarlo mostrará un mensaje según el tipo de error generado.

Servicio web

```python
import json, requests
from requests.exceptions import HTTPError

cities = ['quito']
base_url = 'https://transitoec.herokuapp.com/servicio.php?ciudad='

for city in cities:
    try:
        response = requests.get(base_url + city)
        #Si la respuesta fue exitosa (200), no se ejecuta la excepcion
        response.raise_for_status()
    except HTTPError as http_err:
        print(f'Error HTTP : {http_err}')  # Python 3.6
    except Exception as err:
        print(f'Ocurrió un error inesperado: {err}')  # Python 3.6
    else:
        print('Tiempo de respuesta:',response.elapsed.total_seconds()*1000,'ms')
        print('Tamaño de recurso:',len(response.content),'bytes')
        print('El recurso para '+city+', se ha cargado corretamente')
        #Se convierte el json recibido a una lista
        my_dict = json.loads(response.content)
        print(my_dict)
```

Figura 34: Código ejecutado para verificar los recursos solicitados por el servicio web
Elaborado por: Bryan Licto y Miguel Saritama

Validación de recursos JSON por el servicio web

```
Error HTTP : 404 Client Error: Not Found for url: https://matrices.cauchosvikingo.com/public/transito/s
 .php?ciudad=loja

Process finished with exit code 0
```

Figura 35: Despliegue de mensaje de error al no encontrar un recurso JSON
Elaborado por: Bryan Licto y Miguel Saritama

En la figura 35 se muestra un mensaje de error de tipo HTTP al no encontrar el recurso solicitado sobre una ciudad.

4.5. Pruebas de portabilidad

Estas pruebas permiten verificar las funcionalidades del sistema al ser utilizados con distintos navegadores web.

Tabla 15 Pruebas de portabilidad de las funcionalidades del prototipo

Funcionalidades	Chrome	Firefox	Opera	Edge	Safari
Reconocimiento de voz	x			x	
Reproductor de texto	x	x	x	x	x
Acceder a la interfaz del chatbot	x	x	x	x	x

Nota: Esta tabla muestra la compactibilidad de los navegadores con las funcionalidades del sistema.
Elaborado por: Bryan Licto y Miguel Saritama

En la tabla 15, se describe el uso de las funcionalidades del sistema en distintos navegadores web y a partir de estas pruebas, se comprobó que todas a excepción de la primera funcionalidad, tienen compatibilidad con estos. En el caso del reconocimiento de voz para los navegadores Firefox, Opera y Safari, no existe soporte con el script de su ejecución.

4.6. Pruebas de eficiencia

Con el fin de verificar el rendimiento del sistema bajo ciertas condiciones, se estableció métricas con distintos escenarios para analizar los resultados en base al prototipo diseñado, mediante el uso del software de código abierto Apache Jmeter.

4.6.1. Pruebas de Rendimiento

Estas pruebas determinan el tiempo de respuesta del sistema en la que, se establecieron escenarios tomando como referencia en la que un usuario permanece en un sitio como máximo de 3 segundos si este no carga o recibe respuesta alguna.

Escenario 1

Este escenario se considera al realizar una búsqueda y no encontrar coincidencias, en el que se contempla distintos números de peticiones al sistema en un segundo.

Tabla 16 Pruebas de rendimiento al no encontrar coincidencias de búsqueda

Tipo	# Peticiones	Media(ms)	Min(ms)	Máx.(ms)	%Error	Kb/sec
Get	1	91	91	91	0.00	0,92
Get	50	1637	296	2236	0.00	10,82
Get	70	3023	489	4013	0.00	9,44
Get	100	4102	527	5341	0.00	10,49

Nota: Esta tabla muestra en tiempo de respuesta del sistema en milisegundos según el número de peticiones al no encontrar coincidencias en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

Como se puede observar en la tabla 16, al tener un máximo de 50 peticiones por segundo, el sistema tiene un tiempo de respuesta con un pico máximo de 2.2 segundos, pero al incrementar el número de peticiones y tomando como referencias a 70 por segundo, se aprecia que el tiempo de respuesta del prototipo se incrementa por encima de los 3s.

Escenario 2

Este escenario se considera al realizar una búsqueda y encontrar una sola coincidencia, en el que se contempla distintos números de peticiones al sistema en un segundo.

Tabla 17. Pruebas de rendimiento al encontrar una coincidencia de búsqueda

Tipo	# Peticiones	Media	Min	Máx.	%Error	Kb/sec
Get	1	55	55	55	0.0	28,39
Get	50	779	142	1009	0.0	46,89
Get	70	1668	384	2012	0.0	42,85
Get	100	2472	568	2768	0.0	46,56

Nota: Esta tabla muestra el tiempo de respuesta del sistema en milisegundos según el número de peticiones al encontrar coincidencias en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

Como se puede observar en la tabla 17, incluso al tener un máximo de 100 peticiones por segundo, el sistema tiene un tiempo de respuesta con un pico máximo de 2.7 segundos el cual, a diferencia del anterior escenario, no sobrepasa los 3 segundos.

Escenario 3

Este escenario se considera al realizar una búsqueda y encontrar una lista de coincidencias, en el que se contempla distintos números de peticiones al sistema en un segundo.

Tabla 18. Pruebas de rendimiento al encontrar una lista de coincidencias a una búsqueda

Tipo	# Peticiones	Media	Min	Máx.	%Error	Kb/sec
Get	1	141	141	141	0.0	10,35
Get	50	2729	681	3751	0.0	16,36
Get	70	4939	731	6503	0.0	14,03
Get	100	7013	609	9074	0.0	14,88

Nota: Esta tabla muestra el tiempo de respuesta del sistema en milisegundos según el número de peticiones al encontrar una lista de coincidencias en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

Como se puede observar en la tabla 18, al tener incluso solo 50 peticiones por segundo como máximo, el sistema genera un tiempo de respuesta con un pico máximo de 3.7 segundos, el de mayor tiempo promedio.

Análisis de los Resultados

Un usuario puede obtener una respuesta en base a los tres escenarios descritos anteriormente. Sabiendo que el sistema debería responder en menos de 3 segundos, se establece que el primer, segundo y tercer escenario con un número de peticiones de 50 se obtiene un tiempo de respuesta máximo de 2.2, 1.0 y 3.7 segundos respectivamente. Solamente en el tercer escenario se obtendría un tiempo mayor a los estimado, por lo

que, para mantener el tiempo de respuesta deseado, el número de peticiones debe ser como máximo de 50.

4.6.2. Pruebas de Estrés

Estas pruebas determinan cuando el sistema comienza a producir errores en las respuestas obtenidas a un determinado número de peticiones realizadas en un segundo.

Escenario 1

Este escenario se considera al realizar una búsqueda y no encontrar coincidencias, en el que se contempla distintos números de peticiones al sistema en un segundo hasta que produzca un error.

Tabla 19 Pruebas de estrés al no encontrar coincidencias de búsqueda

Tipo	# Peticiones	Media(ms)	Min(ms)	Máx.(ms)	%Error	Kb/sec
Get	1	91	91	91	0.00	0,92
Get	100	4102	527	5341	0.00	10,49
Get	250	11648	743	14565	0.00	10,78
Get	1000	51456	3250	60205	0.00	10,74
Get	1200	53918	877	72358	7.75	12.74

Nota: Esta tabla detalla cuando el sistema genera errores al realizar un determinado número de peticiones al no encontrar coincidencias en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

Como se puede observar en la tabla 19, al tener un máximo 1200 peticiones por segundo, el sistema ya comienza a generar un porcentaje de error del 7.75%.

Escenario 2

Este escenario se considera al realizar una búsqueda y encontrar una sola coincidencia, en el que se contempla distintos números de peticiones al sistema en un segundo hasta que produzca un error.

Tabla 20 Pruebas de estrés al encontrar una coincidencia de búsqueda

Tipo	# Peticiones	Media	Min	Máx.	%Error	Kb/sec
Get	1	55	55	55	0.0	28,39
Get	100	2472	568	2768	0.0	46,56
Get	250	6984	5304	7908	0.0	47,02
Get	500	14640	277	16808	0.0	45.3
Get	1000	8813	386	16977	47,6	119,89

Nota: Esta tabla detalla cuando el sistema genera errores al realizar un determinado número de peticiones al encontrar una coincidencia en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

En la tabla 20, el sistema a diferencia del anterior escenario, el sistema genera un error significativo del 47.6% al tener un máximo 1000 peticiones por segundo.

Escenario 3

Este escenario se considera al realizar una búsqueda y encontrar una lista de coincidencias, en el que se contempla distintos números de peticiones al sistema en un segundo hasta que produzca un error.

Tabla 21. Pruebas de estrés al encontrar una lista de coincidencias a una búsqueda

Tipo	# Peticiones	Media	Min	Máx.	%Error	Kb/sec
Get	1	141	141	141	0.0	10,35
Get	50	2729	681	3751	0.0	16,36
Get	70	4939	731	6503	0.0	14,03
Get	100	7013	609	9074	0.0	14,88
Get	250	19786	3196	23057	0.0	15,39
Get	1000	78244	2000	90943	2,1	16,27

Nota: Esta tabla detalla cuando el sistema genera errores al realizar un determinado número de peticiones al no encontrar una lista de coincidencias en una búsqueda.
Elaborado por: Bryan Licto y Miguel Saritama

Como se aprecia en la tabla 21, al igual que en el segundo escenario, el sistema genera un porcentaje de error del 2.1% desde las 1000 peticiones.

Análisis de los Resultados

Con los resultados obtenidos se puede deducir que el sistema a partir de las 1000 solicitudes comienza a generar errores en las solicitudes realizadas. Sin embargo, cabe recalcar que, aunque el sistema no generé errores, el tiempo de respuesta se incrementa de forma significativa. Tomando como ejemplo al Escenario 3 con 1000 peticiones vemos que el tiempo de respuesta es de 90 segundos, un valor muy alto para la espera de una respuesta.

CONCLUSIONES

- Chatterbot, al ser una librería de código abierto, permitió la modificación y creación de clases y funciones en su estructura interna, esto sirvió para mejorar el proceso de búsqueda, filtración y selección de respuestas de mayor confiabilidad a las peticiones de usuario.
- Para el procesamiento de las entradas de los usuarios, el algoritmo de Levenshtein Distance, estableció un mayor porcentaje de similitud entre las entradas tipo texto y la información de la base de conocimiento, además no necesito recursos adicionales del kit de herramientas de lenguaje natural (NLTK) para su ejecución.
- Se estableció un aprendizaje supervisado basado en niveles de confianza, debido a que Chatterbot solamente realiza un aprendizaje de tipo secuencial, es decir, no consideró la relación entre una pregunta y una posible respuesta, lo que resultó ineficiente al momento de aprender y automatizar las respuestas.
- Se creó una base datos NoSQL, la cual contiene una colección que almacena las preguntas y respuestas para establecer la conversación entre el chatbot y el usuario, además, de otras colecciones con la información de los trámites de las agencias de tránsito separadas por ciudad las cuales se relacionaron mediante un identificador para el procesamiento de la información.
- La integración de las APIS para el reconocimiento y síntesis de voz mejoró la experiencia del usuario al interactuar con el chatbot, esto se vio reflejado en las pruebas de usabilidad y funcionalidad realizadas.
- El sistema necesitó un entrenamiento inicial para la interacción con el usuario, para ello se desarrolló un servicio web que consume la información de las

agencias de tránsito en formato JSON y la almacena en la base de conocimientos.

- Se demostró que el prototipo tiene una precisión mayor al 80%, al momento de proporcionar una respuesta confiable a una petición realizada por el usuario, esto se vio reflejado en las pruebas de confiabilidad realizadas.

RECOMENDACIONES

- Se podría crear un interfaz de usuario para acceder a las colecciones que contienen la información de cada agencia con el propósito de realizar modificaciones o inserciones de información.
- Para depurar las entradas tipo texto enviadas por el usuario, es recomendable crear funciones e integrarlas como preprocesadores de Chatterbot.
- Para un tiempo de respuesta óptimo frente a un escenario con un mayor número de peticiones, se debería contar con un equipo de hardware con características más robustas.

LISTA DE REFERENCIAS

Artículos Académicos

Benjamin Roseth, Angela Reyes, Carlos Santiso. (2018 Junio). *Wait No More: Citizens, Red Tape, and Digital Government.* Washington, D.C. 20577: Banco Interamericano de Desarrollo.

Bibliografía

Aguilar, L. J. (09 de abril de 2016). *Big Data, Análisis de grandes volúmenes de datos en organizaciones.* Mexico: Ink. Obtenido de http://www.cantabriatic.com/introduccion-a-mongodb/#:~:text=MongoDB%20y%20Colecciones,que%20las%20colecciones%20almacenan%20documentos.

Calero, C., Piattini, M., & Moraga, Á. (2010). *Calidad del producto y proceso software.* Madrid: RA-MA

Casas Roma, J., & Conesa Caralt, J. (2014). *Diseño conceptual de bases de datos en UML*. Barcelona: UOC.ca

Elliott, E. (05 de Octubre de 2014). Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. United States: O'Really Media Inc. Obtenido de https://codingornot.com/mvc-modelo-vista-controlador-que-es-y-para-que-sirve

Indurkhya, N., Damerau, F., Graepel, T., & Herbrich, R. (2010). *Handbook of Natural Language Processing.* Florida: CRC Press.

Libby, A. (10 de 05 de 2020). *Introducing the HTML5 Web Speech API: Your Practical Introduction to Adding Browser-Based Speech Capabilities to your Websites and Online Applications.* Apress. Obtenido de https://developer.mozilla.org/es/docs/Web/API/Web_Speech_API/Uso_de_la_Web_Speech_API

Matchware. (10 de 07 de 2020). *A Guide to the Project Management Body of Knowledge (5thed).* Obtenido de Matchware Software for Creative Minds: https://www.matchware.com/product-breakdown-structure

Mathivet, V. (Febrero de 2018). Inteligencia Artificial para desarrolladores. Conceptos e implementación en C#. Barcelona: ENI. Obtenido de https://www.researchgate.net/publication/322963551_bases_motor_de_inferencia

Ortega, J. M. (2018). *Hacking ético con herramientas Python.* Madrid: RA-MA.

Perkins, J. (2014). *Procesamiento de texto Python 3 con el libro de cocina NLTK 3.*

Pressman, R. (2010). *Ingenería del software Un Enfoque práctico.* México: ISBN.

Recios, M. L. (28 de Octubre de 2015). UF1889 - Desarrollo de componente software en sistemas ERP-CRM. España: Elearning S.L. Obtenido de https://openwebinars.net/blog/que-es-mongodb/

Rim Jallouli, O. R. (03 de Mayo de 2017). *Digital Economy. Emerging Technologies and Business Innovation: Second International Conference, ICDEc 2017, Sidi Bou Said, Tunisia, May 4–6, 2017, Proceedings.* Springer. Obtenido de o es una medida clásica de similitud entre dos conjuntos que introdujo Paul Jaccard en 1901. Dados dos conjuntos, A y B, la similitud de Jaccard se define como el tamaño del intersección del conjunto A y el conjunto B (es decir, el número de elementos com

Rothman, D. (12 de Julio de 2018). *Artificial Intelligence By Example: Develop machine intelligence from scratch using real artificial intelligence use cases.* Birmingham: Packt Publishing Ltd. Obtenido de https://www.ibm.com/cloud/watson-assistant/

Tudor, J. (2019). *Python para principiantes: Aprenda Python en 5 días con orientación paso a paso y ejercicios prácticos.* Babelcube Inc.

Vijay Nath, J. K. (02 de Octubre de 2019). *Proceedings of the Third International Conference on Microelectronics, Computing and Communication Systems: MCCS 2018.* Springer. Obtenido de https://dzone.com/articles/the-levenshtein-algorithm-1#:~:text=The%20Levenshtein%20distance%20is%20a,one%20word%20into%20the%20other.

Documentos

Ministerio de Telecomunicaciones y de la Sociedad de la Información. (2018). Obtenido de Plan Nacional de Gobierno Electrónico 2018 - 2021: https://www.gobiernoelectronico.gob.ec/wp-content/uploads/2018/09/PNGE_2018_2021sv2.pdf

PRESIDENCIA DE LA REPUBLICA DEL ECUADOR. (11 de octubre de 2010). Obtenido de CODIGO ORGANICO ORGANIZACION TERRITORIAL AUTONOMIA DESCENTRALIZACION: http://www.oas.org/juridico/pdfs/mesicic4_ecu_org.pdf

Sitios Web

Amazon. (12 de Julio de 2020). *Aws.amazon.com.* Obtenido de https://aws.amazon.com/lex/?nc1=h_ls

Apache.org. (7 de Julio de 2020). *Apache JMeter*. Obtenido de https://jmeter.apache.org/

CONADIS. (04 de julio de 2019). *Estadísticas de Discapacidad*. Obtenido de Consejo Nacional para la Igualdad de Discapacidades: https://www.consejodiscapacidades.gob.ec/estadisticas-de-discapacidad/

Facebook. (12 de Julio de 2020). *Wit.ai.* Obtenido de https://wit.ai/docs

Google. (12 de Julio de 2020). *Cloud.google.* Obtenido de https://cloud.google.com/dialogflow?hl=es

Gunther Cox Revision. (2019). *Chatterbot*. Obtenido de https://chatterbot.readthedocs.io/en/stable/

Ionos. (25 de Septiembre de 2019). Obtenido de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/software-development-kit/

JetBrains. (2020). Obtenido de https://www.jetbrains.com/es-es/pycharm/features/

Json.org. (10 de Julio de 2020). *json.org*. Obtenido de https://www.json.org/json-es.html

Medina, A. (16 de Febrero de 2018). *En Pichincha hay 2 autos por cada 10 personas desde el 2016.* Obtenido de EL COMERCIO: https://www.elcomercio.com/datos/pichincha-tasa-autos-personas-vehiculos.html

Microsoft. (12 de Julio de 2020). *botframework.com.* Obtenido de https://dev.botframework.com/

MonkeyLearn. (02 de Enero de 2020). Obtenido de https://monkeylearn.com/sentiment-analysis/

Mozilla, D. (11 de Julio de 2020). *Developer.mozilla.org*. Obtenido de https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

Palletsprojects. (9 de Julio de 2020). *The Pallets Projects*. Obtenido de https://palletsprojects.com/p/flask/

PHP. (10 de Julio de 2020). *PHP*. Obtenido de https://www.php.net/manual/es/intro-whatis.php

Proyecto NLTK. (13 de abril de 2020). *Documentación de NLTK 3.5*. Obtenido de https://www.nltk.org/

Redacción Política. (22 de Agosto de 2014). *25 de 221 municipios han asumido las competencias de tránsito (Infografía)*. Obtenido de El Telégrafo: https://www.eltelegrafo.com.ec/noticias/politica/3/25-de-221-municipios-han-asumido-las-competencias-de-transito-infografia

RedHat. (08 de Julio de 2020). *Red Hat.* Obtenido de https://www.redhat.com/es/topics/api

Reyna, A. (s.f.). *La inteligencia artificial acelera la evolución de los 'chatbots'*. Obtenido de BBVA: https://www.bbva.com/es/inteligencia-artificial-acelera-evolucion-chatbots/

Stack, R. (12 de Julio de 2020). *Rasa.com*. Obtenido de https://rasa.com/docs/getting-started/

W3C. (8 de Julio de 2020). *W3C*. Obtenido de https://www.w3.org/TR/ws-arch/#whatis

ANEXOS

Anexo 1.

Ver documento Plan de Pruebas.docx

Anexo 2.

Ver documento Encuestas.xlsx

Anexo 3.

Ver documento EncuestasOnline.docx