UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso para Alumnos con Práctica Profesional Supervisada

TÍTULO: Interfaz conversacional para la recolección de datos en la web semántica

AUTORES: Gaston Ezequiel Ciancio

DIRECTOR/A ACEDÉMICO: Alejandro Fernández

CODIRECTOR/A ACADÉMICO: Diego Torres

DIRECTOR/A PROFESIONAL: Sergio Firmenich

CARRERA: Licenciatura en Sistemas

Resumen

El presente trabajo de grado se ubica en el área del diseño de interfaces conversacionales. El objetivo de esta tesina es determinar el potencial de una interfaz conversacional para consultar una base de datos orientada a grafos, en particular Wikidata, haciendo un aporte a un ecosistema de servicios de consulta ya existente.

Palabras Clave

Chatbot, Web Semántica, SPARQL, Interfaz conversacional, Templet, QAWiki, Wikidata.

Conclusiones

Mediante el desarrollo de un Bot que permite responder preguntas en lenguaje natural, y construyendo sobre servicios existentes como Telegram y QAWiki (un catálogo de plantillas de preguntas a Wikidata en lenguaje natural), fue posible crear una interfaz conversacional para responder preguntas con información en Wikidata. Adicionalmente, la solución propuesta enriquece el catálogo de plantillas de preguntas de QAWiki.

Trabajos Realizados

Se realizó una exhaustiva investigación sobre interfaces conversacionales, así como también los trabajos relacionados a este tema y herramientas existentes. Se desarrolló un sistema que permite identificar preguntas en lenguaje natural en idioma inglés, establecer mapeos entre la pregunta y la correspondiente consulta SPARQL, y enriquecer las plantillas para preguntas cargadas QAWiki identificando preguntas con similitud semántica/sintáctica. El sistema también mantiene un contexto de la conversación.

Trabajos Futuros

Desarrollo de la presente aplicación en idioma español.

Desarrollo de plantillas para preguntas con más de una mención.

Desarrollo de plantillas para alias.

Fecha de la presentación: 03 2024

Índice

Exposición de lo realizado en la PPS

Se realizó una exhaustiva investigación del estado del arte en la actualidad relacionado al tema abordado en la PPS, para el cual se tomaron en cuenta artículos académicos y científicos, revistas académicas, libros, etc. Estas fuentes de información explican qué desafíos se presentan hoy en día, las características principales que ofrece cada una de las herramientas existentes que abordan cada uno, y qué diferencias y relación existe con otros artículos. Además, se aborda una variedad de temas relacionados con la generación y el desarrollo de chatbots para la exploración de datos.

Se exploró cómo los chatbots pueden ser generados automáticamente a partir del análisis de la estructura y el contenido de datos tabulares. Esto implica un enfoque de automatización para crear interfaces conversacionales que faciliten la exploración de datos. Se discutieron los paradigmas emergentes y patrones de diseño en el desarrollo de interfaces conversacionales, así como el contexto y las ventajas de los bots para usuarios y desarrolladores.

Se examinó cómo se pueden utilizar datos enlazados y bases de datos semánticas para mejorar la comprensión y precisión de las consultas de los usuarios en los chatbots. Esto implica la utilización de técnicas de aprendizaje automático para clasificar qué es lo que quiere el usuario y comprender el lenguaje natural del mismo.

Se investigó un marco de trabajo para el diseño de chatbots que faciliten la exploración de datos, incluyendo la caracterización de tablas de bases de datos relacionales en entidades conversacionales. Se analizó un paradigma interactivo para el desarrollo de chatbots para usuarios finales en la exploración de datos. Esto implica la creación rápida de prototipos de interfaces conversacionales utilizando bases de datos y la finalización automática de patrones de conversación. Durante el desarrollo de la PPS, se abordaron diversos aspectos para crear un sistema conversacional basado en Telegram que permita a los usuarios realizar preguntas en lenguaje natural y recibir respuestas relevantes de manera automatizada. Se realizó un análisis sobre qué tecnologías usar en base a los requerimientos y conocimientos previos en programación del estudiante a cargo de la PPS. A continuación, se presentan en detalle los principales puntos abordados y realizados:

En el desarrollo de esta investigación, se ha explorado el ecosistema de conocimiento colaborativo representado por Wikidata, una base de datos estructurada y enlazada creada por la fundación Wikimedia. Esta plataforma proporciona un repositorio centralizado de información que puede ser utilizada y compartida de manera global, facilitando la gestión de datos multilingües y eliminando la duplicación de elementos entre diferentes idiomas. Se destacó el papel del servicio de consultas de Wikidata, que permite a los usuarios realizar consultas complejas sobre los datos enlazados utilizando el lenguaje SPARQL, lo que abre oportunidades para la exploración avanzada de datos y la

extracción de conocimiento. Además, se exploraron iniciativas como QAWiki y Templet, que amplían el potencial de Wikidata para simplificar la interacción de los usuarios con los datos. Finalmente, se abordó el ascenso de las interfaces conversacionales como una herramienta prometedora para facilitar el acceso a la información y la interacción con sistemas complejos, ofreciendo beneficios como acceso rápido y preciso a la información, asistencia en tareas complejas, exploración de datos y automatización de tareas rutinarias.

Se realizó una comparación con trabajos previos relacionados para contextualizar y resaltar la singularidad del enfoque propuesto en esta tesina. Se resaltó cómo otros proyectos requieren que los usuarios adapten sus preguntas, mientras que el enfoque propuesto elimina esta restricción y permite consultas en lenguaje natural. También se compararon los proyectos señalando las diferencias en sus enfoques y tecnologías utilizadas en comparación con la propuesta de esta tesina. Se proporcionó una visión general de cómo el enfoque de la tesina se diferencia y destaca en comparación con trabajos relacionados, resaltando su singularidad y contribución al campo de las interfaces conversacionales.

Se detallaron los procesos en background que sustentan el funcionamiento del sistema desarrollado, al mismo tiempo que optimizan el rendimiento, eficiencia y disponibilidad del sistema. Se explicó la decisión de utilizar un almacenamiento temporal para almacenar preguntas y respuestas previamente obtenidas, destacando los beneficios que esto proporciona. Se describieron dos tareas fundamentales que el sistema lleva a cabo mientras los usuarios interactúan con él. En primer lugar, se explicó el proceso de borrado periódico del almacenamiento de respuestas. En segundo lugar, se abordó el proceso de actualización de preguntas.

Se abordaron los escenarios de uso del sistema desarrollado, centrándose en cómo el sistema responde a diferentes situaciones cuando los usuarios interactúan con él. Se detallaron los procesos que ocurren cuando un usuario envía un texto al sistema y cómo este lo analiza para determinar si constituye una pregunta que el sistema pueda procesar. Se destacó la importancia del almacenamiento temporal, donde se guardan preguntas y respuestas previamente procesadas con éxito. Asimismo, se explicó el uso de un contexto conversacional, fundamental para que el sistema comprenda adecuadamente las solicitudes y respuestas del usuario y proporcione una experiencia de conversación coherente y personalizada. Se presentaron varios escenarios posibles de interacción entre el usuario y el sistema, detallando las respuestas esperadas en cada caso. Estos escenarios abarcan desde situaciones donde el texto del usuario constituye una pregunta procesable que está o no almacenada en el sistema, hasta casos donde el texto no es una pregunta procesable o no existe un contexto conversacional. Se proporcionó una visión general del flujo de interacción y se describieron los procedimientos que se llevan a cabo para manejar cada situación posible. Además, se explicó cómo se gestionan los errores inesperados y cómo se recopila el feedback del

usuario para mejorar la experiencia de uso del sistema en futuras interacciones.

Se realizó una investigación para identificar las herramientas de diseño de interfaces conversacionales más adecuadas, considerando los requerimientos del sistema y la interacción con el mismo para procesar las preguntas de los usuarios. Se abordó la selección de la plataforma de base para el sistema desarrollado, con el objetivo de permitir a los usuarios realizar preguntas en lenguaje natural y recibir respuestas de manera efectiva. Finalmente, se explica por qué se eligió Telegram como la plataforma base para el desarrollo del proyecto. Se destacan las ventajas de Telegram, además, se mencionan las características específicas de Telegram que fueron fundamentales para la implementación del sistema.

Por otro lado, se presenta la arquitectura general de la solución desarrollada. Se describe cómo un usuario interactúa con el sistema a través de Telegram, y cómo el backend procesa la información recibida, se conecta con servicios externos y bases de datos, gestiona el guardado de respuestas y administra el contexto de la conversación. Se incluye el desarrollo de un diagrama de arquitectura que ilustra estas interacciones y se detallan las tecnologías utilizadas en el proyecto, incluyendo las bibliotecas y herramientas del lenguaje de programación utilizado. También, se mencionan herramientas complementarias para el desarrollo, empaquetado y gestión del código y la infraestructura. Se explica el propósito de cada tecnología y cómo contribuyó a la implementación exitosa del proyecto. Se detallan las implementamos técnicas de procesamiento de lenguaje natural hechas y cómo se realizaron análisis sintáctico y etiquetado gramatical, esenciales para comprender el significado de las preguntas de los usuarios. Se detalla la herramienta de gestión de errores y monitoreo utilizada para detectar desperfectos y garantizar el correcto funcionamiento del sistema.

Se aborda el funcionamiento detallado del sistema desarrollado, centrándose en las diversas actividades que realiza para procesar las consultas de los usuarios y proporcionar respuestas relevantes. Para esto, se presenta un cuadro con los nombres de las actividades, detallando con una descripción que es lo que realiza la actividad y que flujo debió tomar el sistema para poder realizar dicha actividad. Se proporciona una visión detallada del flujo de trabajo del sistema, desde la recepción de la consulta del usuario hasta la entrega de una respuesta relevante y la gestión de posibles errores.

Por otro lado, se presentó una demostración del funcionamiento del sistema, donde se describieron los flujos principales y casos de uso que tiene el sistema. Se explicaron los diferentes flujos que el sistema puede seguir en función de las interacciones del usuario, desde el envío de una pregunta válida hasta la finalización de la conversación. Se detallaron los diferentes casos de uso del sistema, como la respuesta a preguntas existentes en la base de datos, manejo de preguntas no válidas, finalización de la conversación, entre otros. Se

presentaron ejemplos concretos de interacción entre el usuario y el sistema, mostrando cómo el sistema responde en cada caso específico, desde que el usuario envía una pregunta válida, una pregunta relacionada o hasta un comando especial. Se explicó cómo el sistema guarda y utiliza un contexto de conversación para ofrecer respuestas más precisas y relevantes a las preguntas del usuario. En resumen, este capítulo proporciona una visión general del funcionamiento del sistema, destacando los flujos principales, casos de uso y ejemplos concretos de interacción entre el usuario y el sistema.

Finalmente, se detallan las conclusiones obtenidas a partir de la investigación realizada, así como también se plantean posibles áreas de trabajo futuro. En primer lugar, se destaca la exhaustiva investigación llevada a cabo sobre las herramientas y tecnologías disponibles para la recolección de datos en la web semántica, así como el proceso de diseño de interfaces conversacionales. Este análisis permitió identificar las mejores prácticas y enfoques para desarrollar una herramienta eficiente y accesible. En cuanto a los posibles trabajos futuros, se plantean diversas áreas de exploración y mejora.

Informe Tecnico

Capítulo 1: Introducción

1.1 Resumen

En la era de la información digital y la colaboración global, plataformas como Wikidata emergen como pilares fundamentales para la creación, gestión y distribución de conocimiento estructurado. Se han convertido en repositorios centralizados de datos con más de 90 millones de elementos, abordando inicialmente desafíos específicos en Wikipedia, como la duplicación de elementos multilingües. Su enfoque en elementos, y la utilización de datos enlazados mediante RDF, han permitido una gestión centralizada de información.

Sin embargo, la accesibilidad a los datos enlazados, aunque poderosa, presenta desafíos para el usuario común debido a la necesidad de lenguajes especializados como SPARQL. Para abordar esta brecha, han surgido sistemas como QAWiki y Templet, que se centran en la creación colaborativa de preguntas en lenguaje natural y consultas en SPARQL, facilitando el acceso a datos complejos de Wikidata de manera más intuitiva.

QAWiki, como repositorio de preguntas y consultas, busca mejorar la calidad de conjuntos de datos y proporcionar un recurso para entrenar sistemas de preguntas y respuestas. Templet, por otro lado, se construye sobre QAWiki, ofreciendo a los usuarios una interfaz amigable para responder preguntas complejas sobre Wikidata mediante plantillas en lenguaje natural y autocompletado.

Además, en el contexto de las interfaces conversacionales (por ejemplo, chatbots), se observa un cambio significativo hacia la eficiencia y la optimización en la interacción usuario-sistema. Estas interfaces permiten a los usuarios acceder rápidamente a información precisa, interactuar de manera intuitiva y recibir asistencia en tareas complejas. Su capacidad para explorar datos, acceder a bases de conocimientos y fomentar la colaboración y aprendizaje, así como la automatización de tareas rutinarias en el ámbito informático, las posiciona como herramientas clave en el panorama actual.

En este contexto, esta tesina explora la contribución fundamental de plataformas como QAWiki y Templet para facilitar el acceso a datos complejos. No obstante, nuestro enfoque central se orienta hacia la creación de un chatbot especializado en la recolección de datos en la web semántica. Este chatbot, basado en una sólida base de conocimiento y técnicas de procesamiento del lenguaje natural, busca optimizar la interacción entre usuarios y sistemas, permitiendo la recopilación eficiente de información estructurada en el vasto entorno de la web semántica.

1.2 Palabras claves

Chatbot, Web Semántica, SPARQL, Interfaz conversacional, Templet, QAWiki, Wikidata.

1.3 Objetivos

Mediante esta tesina, realizada en el marco de una práctica profesional supervisada, aspiramos a alcanzar los siguientes objetivos:

Objetivo principal

Determinar el potencial de una interfaz conversacional para consultar Wikidata, haciendo un aporte al ecosistema Wikidata+QAWiki+Templet. Para ello, se buscará alcanzar los siguientes objetivos específicos:

Objetivos específicos

- Identificar un diseño de interfaz conversacional efectivo para explotar los templates disponibles en QAWiki.
- Registrar aliases, es decir pregunta en lenguaje natural que son alternativas posibles a preguntas existentes QAWIki (pero no idénticas).
- Identificar preguntas en lenguaje natural que no se correspondan con templates en QAWiki, motivando su posterior implementación.
- Identificar una estrategia de diseño de interfaz conversacional que aproveche la naturaleza secuencial y esté anidada de la conversación.

El resto del documento se organiza de la siguiente manera:

Capítulo 2: Trabajo realizado de desarrollo. Describe la aplicación desarrollada, junto con sus decisiones y especificaciones técnicas y su validación mediante demostración.

Capítulo 3: Conclusiones y posibles trabajos futuros. Contiene la conclusión del trabajo en base al objetivo planteado, junto la presentación de posibles trabajos futuros.

Capítulo 4: Bibliografía. Contiene enlaces a documentos, libros y fuentes en las que este trabajo se basa.

Capítulo 5: Anexos.

1.4 Marco conceptual

Wikidata

Wikidata es una base de conocimientos documental editada en colaboración, creada en el 2012 por la Fundación Wikimedia, tiene el objetivo de proporcionar una fuente común de datos que puedan ser utilizados por cualquier otra persona, bajo una licencia de dominio público.

Esta base de conocimientos está enfocada en elementos. Cada uno de los elementos de Wikidata representa un tipo de tema, concepto u objeto y tiene asignado un identificador único y persistente, compuesto del prefijo Q, conocido como “QID”. De esta forma, es posible identificar elementos globalmente a través de los distintos idiomas disponibles. Un elemento posee obligatoriamente un QID asociado a una etiqueta y una descripción en uno o más lenguajes naturales. Además, opcionalmente, tiene varios alias. La información se agrega a los elementos mediante la creación de declaraciones, en forma de pares clave-valor, cada declaración consiste en una propiedad y un valor vinculado a la misma. Su principal ventaja es que ofrece datos enlazados, descritos mediante RDF [2], lo cual permite relacionarlos con otros conjuntos de datos de otros repositorios digitales.

Wikidata aprovecha las ventajas de la Web semántica para que su base sea de conocimiento universal. A la hora de editar Wikipedia o cualquier otra wiki de Wikimedia, el usuario puede cargar los datos dinámicamente desde Wikidata. De esta forma se pueden gestionar de manera centralizada estadísticas, fechas, ubicaciones, etc. Al cambiar el dato en Wikidata, se actualizará de manera automática en todos artículos en los que aparece y en todos los idiomas.

El propósito inicial de Wikidata era resolver algunos problemas concretos de Wikipedia, siendo el principal la duplicación de elementos multilingües, ya que existía redundancia de artículos que hacían referencia a un mismo elemento en idiomas diferentes y no se encontraban unificados. Actualmente es un depósito centralizado de datos estructurados, almacenando más de 90 millones de elementos.

En 2015, la Fundación Wikimedia anunció el lanzamiento del Servicio de Consultas de Wikidata, el cual permite a los usuarios navegar sobre los datos enlazados de Wikidata con el lenguaje de consulta SPARQL. Las consultas en este lenguaje se escriben contra los pares clave-valor, por lo que la base de datos del

sistema es un conjunto de tripletas sujeto-predicado-objeto. El gran conjunto de datos disponibles y la amplia gama de posibles consultas a ejecutar en el Servicio de Consultas de Wikidata ha permitido que se convierta en uno de los sitios de conocimiento abierto más destacados de la Web.

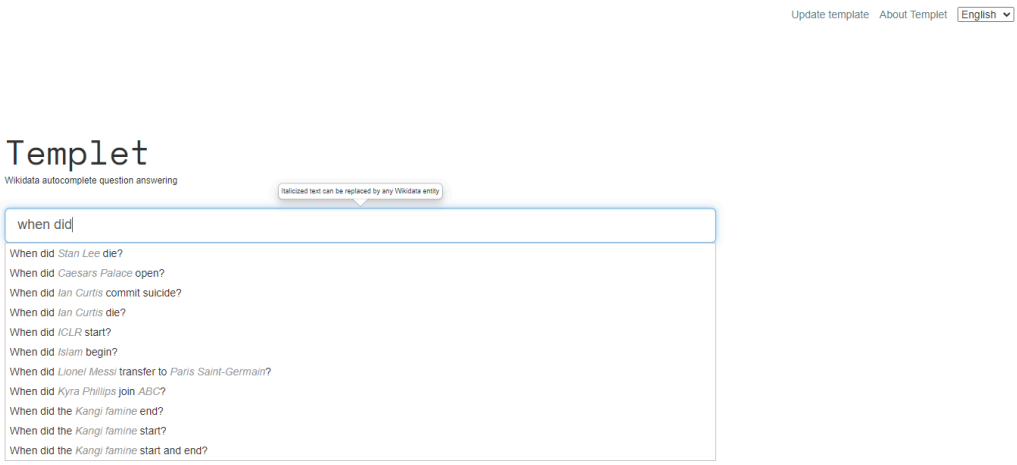
Los datos enlazados describen un método de publicación de datos estructurados. Son utilizados para compartir información de una manera que pueda ser leída automáticamente por dispositivos. Esto permite que los datos sean conectados y consultados de diferentes fuentes. El desafío de estos datos es que para acceder a ellos, se necesitan lenguajes especializados, como SPARQL, y generalmente el usuario común no está familiarizado con este lenguaje, tampoco con la semántica empleada en los datos enlazados, produciéndose una brecha que impide la comunicación entre el usuario y estos datos. Para mitigar este problema, se han creado sistemas de respuesta a preguntas en lenguaje natural, transformando expresiones en lenguaje natural a consultas estructuradas en lenguajes especializados. Las soluciones del estado del arte más recientes aplican un enfoque de redes neuronales artificiales y traducen directamente el lenguaje natural a SPARQL [8]. El desafío que enfrentan este tipo de soluciones es la necesidad de grandes datasets para poder entrenar los modelos, utilizando cerca del orden de millones de instancias. Actualmente, no se tiene una colección de datos de tal volumen, dificultando el avance de estas tecnologías.

QAWiki y Templet

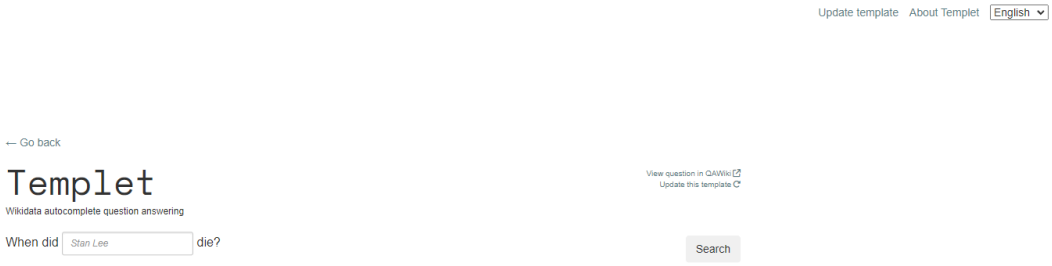
QAWiki [4] es un repositorio de preguntas en lenguaje natural con sus correspondientes consultas en SPARQL. Está basado en el software Wikibase. Fue creado con el fin de solucionar problemas respecto a la calidad del conjunto de datos LC-QuAD 2.0 [6], además de la necesidad de un conjunto de datos de entrenamiento y prueba que presente preguntas distintas a las plantillas comunes. QAWiki almacena un nuevo conjunto de datos independiente con más de 400 pares de preguntas y consultas que se pueden responder en Wikidata y que se extiende con el tiempo, además de una completa descripción de las menciones y entidades contenidas. Este conjunto de datos no sigue plantillas estándar y contiene consultas con una variedad de complejidades y características algebraicas, que van desde consultas simples hasta preguntas complejas. Actualmente, es una plataforma abierta y colaborativa con el propósito de mantener un conjunto de datos abierto para mejorar el rendimiento de sistemas de QA (Question-Answer).

Con el objetivo de mostrar tanto el potencial de Wikidata como el de QAWiki para usuarios finales, se desarrolló Templet. Templet [4] es un sistema en línea de respuestas a preguntas para Wikidata, se basa en el repositorio QAWiki y genera plantillas a partir de pares de preguntas y consultas en QAWiki al reemplazar entidades clave por identificadores. Utilizando el autocompletado, el usuario puede escribir una pregunta en lenguaje natural, seleccionar una plantilla y nuevamente, utilizando la autocompletación, elegir las entidades que desea insertar en los marcadores de posición de la plantilla, generando así una pregunta concreta, una consulta y resultados.

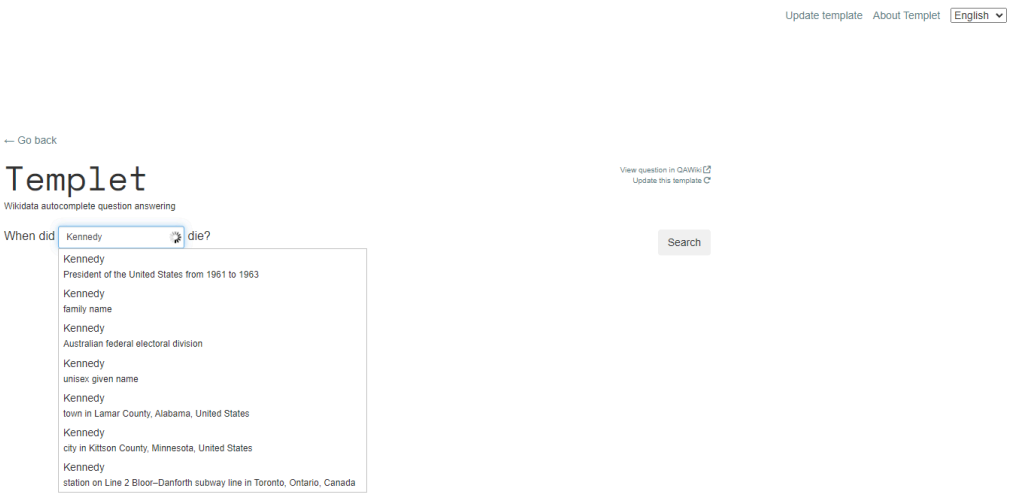
A continuación presentaremos un ejemplo para la búsqueda de una pregunta la cual su respuesta se encuentra en Wikidata, haciendo uso del sistema Templet



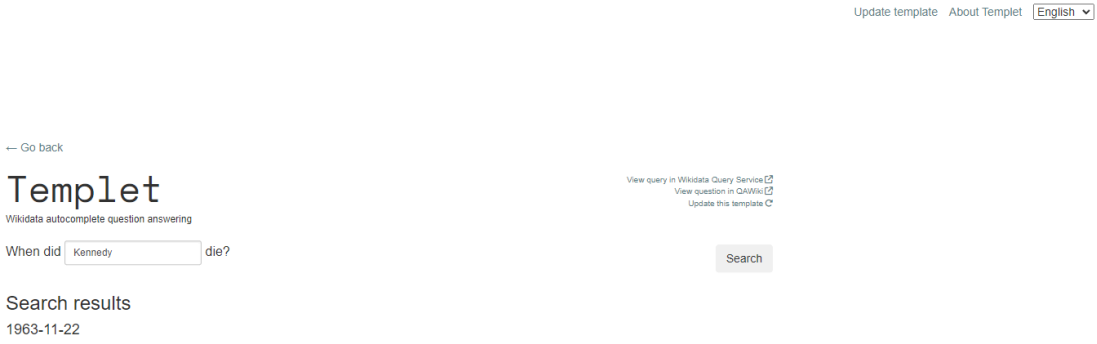
Como se puede apreciar la interfaz es sencilla, posee un input para ingresar la pregunta deseada, la cual a medida que el usuario escribe el sistema ofrece un autocompletado de posibles preguntas, las cuales marcan en un color gris la sección donde la pregunta es dinámica ya que ahí se inserta la entidad proveniente de Wikidata que se busque.



Eligiendo la primera pregunta que el autocompletado ofrece, el sistema muestra la plantilla correspondiente, la cual el usuario debe completar manualmente el input con la entidad que quiera.



Completando la plantilla, el sistema ofrece un autocompletado el cual ofrece entidades posibles de acuerdo al texto que ingrese el usuario. El usuario debe elegir cuál entidad es la que quiere utilizar en la pregunta y luego presionar el botón “Search” para realizar la búsqueda. De esta forma, Templet muestra el resultado obtenido desde Wikidata.



Los objetivos principales de Templet son:

i. Permitir a los usuarios responder preguntas potencialmente complejas sobre Wikidata utilizando plantillas en lenguaje natural y autocompletado.

ii. Fomentar que los usuarios creen colaborativamente nuevas plantillas a través de QAWiki, lo cual a su vez puede beneficiar no solo a Templet, sino también a otros sistemas de preguntas y respuestas.

En cuanto a las debilidades de Templet, una de ellas es que se tiene pasos extra entre la selección de plantilla y el reemplazo de entidades. Luego de pruebas con usuarios sin conocimiento previo en el sistema, varios usuarios mencionaron que sería ideal que al ingresar la pregunta en la primera entrada de texto, esto ya fuera suficiente información para obtener los resultados de la pregunta [7]. El problema yace en que para identificar las entidades dentro de la pregunta, es necesario el uso de inteligencia artificial. Por ejemplo, es fácil de lograr responder preguntas que tienen una única entidad a reemplazar en la plantilla, pero es muy complejo en aquellas que tienen más de una, puesto que el texto de la pregunta separa ambas entidades y se confunde con el contenido de la primera entidad. Otra debilidad es la presentación de la información, como se puede apreciar no ofrece más información que la respuesta literal de la pregunta cuando es posible expandir el conocimiento haciendo uso de las facilidades que otorga QAWiki. Además cabe aclarar que la única forma de que el usuario sepa si realizó una pregunta de manera incorrecta es buscando la respuesta y obtener un error, habiendo realizado todos los pasos previos, esto desencadena en una experiencia de usuario tediosa e insatisfactoria para el usuario.

Interfaces conversacionales

En los últimos tiempos han ganado relevancia las interfaces conversacionales. Estos son softwares que simulan una conversación con un humano. Se trata de agilizar el proceso para que el usuario no deba dar rodeos innecesarios o pasos extras. La idea es ser más eficientes a la hora de atender las necesidades del usuario

y ofrecer una interfaz más optimizada. Gracias a una base de conocimiento, la interfaz conversacional está entrenada para responder las preguntas frecuentes de los usuarios. De igual manera, es posible que en algún momento sea necesario comunicarse con un humano debido a que se sigue dependiendo del mismo para su correcto funcionamiento. Las interfaces conversacionales traen beneficios [9] [10] [11], tales como:

- Acceso a información rápida y precisa: Las interfaces conversacionales permiten a los usuarios acceder a información de manera rápida y precisa, ya que pueden formular preguntas en lenguaje natural y recibir respuestas detalladas y relevantes.
- Interacción intuitiva: Estas interfaces brindan una experiencia de usuario intuitiva, ya que imitan la forma en que las personas se comunican entre sí. Los usuarios pueden interactuar con sistemas complejos utilizando un lenguaje natural familiar sin la necesidad de aprender comandos o interfaces gráficas específicas.
- Asistencia en tareas complejas: Las interfaces conversacionales pueden proporcionar asistencia al guiar a los usuarios a través de procesos y proporcionar explicaciones detalladas.
- Exploración de datos: Las interfaces conversacionales pueden ayudar a los usuarios a explorar y analizar estos datos al permitirles hacer preguntas específicas o solicitar visualizaciones relevantes.
- Acceso a bases de conocimientos: Las interfaces conversacionales pueden conectarse a bases de conocimientos enlazados o sistemas de datos científicos, lo que permite a los usuarios acceder a información actualizada y confiable en tiempo real.
- Colaboración y aprendizaje: Las interfaces conversacionales pueden fomentar la colaboración entre científicos e informáticos al proporcionar una plataforma para compartir información, discutir ideas y resolver problemas de manera conjunta.
- Automatización de tareas rutinarias: En el ámbito informático, las interfaces conversacionales pueden ayudar a automatizar tareas rutinarias, como la gestión de sistemas, el monitoreo de servicios y la resolución de problemas técnicos.

1.5 Comparación con trabajos relacionados

Previamente se habló de Templet, este sistema está estrechamente relacionado ya que también utiliza QAWiki, y provee información de Wikidata. Este sistema necesita que la pregunta en lenguaje natural del usuario se adapte con las plantillas precargadas, teniendo que completar las secciones dinámicas de la plantilla para cualquier pregunta. El enfoque realizado en este trabajo no requiere este paso extra, solo es necesaria la consulta en lenguaje natural del usuario. Además, Templet no permite una serie de consultas relacionadas a la primera consulta que realizó, el

enfoque de esta tesina permitiría que los usuarios puedan realizar dicha secuencia de preguntas.

Actualmente existen proyectos en desarrollo que se presentaron como alternativas, entre ellas se encuentra un software que propone un nuevo enfoque [1] en el que los bots se deriven automáticamente a partir de un análisis de la descripción y el contenido de los datos tabulares. Se infieren los tipos de las columnas a partir de los valores de las mismas y, junto con sus nombres, se utilizan para generar preguntas que los usuarios podrían hacer sobre el conjunto de datos. El proceso no requiere ninguna entrada manual obligatoria y puede escalar para cubrir un gran número de conjuntos de datos, al mismo tiempo que ofrece una interfaz web opcional para enriquecer la descripción de los datos con el fin de mejorar la generación del bot si así lo desea . Mediante este enfoque, con el acceso de un archivo de datos, se entrena el chatbot correspondiente, en cambio, mi enfoque plantea usar la web semántica como acceso a los datos.

Por otro lado, existe otro proyecto que propone un modelo [5] de trabajo de un chatbot basado en una ontología que maneja consultas de usuarios para un sitio web de comercio. Su principal objetivo es brindar al usuario un control total sobre los resultados de búsqueda en el sitio web. Este chatbot ayuda al usuario a mapear las relaciones entre las diversas entidades requeridas por el usuario, brindando así información detallada y precisa. Este enfoque está relacionado al utilizar una ontología para resolver las consultas de los usuarios en lenguaje natural, mi enfoque utiliza la ontología de QAWiki para obtener los datos solicitados.

CHATIDEA [3] utiliza un enfoque de generación automática de chatbots a partir de un volcado de una base de datos relacional y un archivo de descripción. Al combinarlos, CHATIDEA genera automáticamente el sistema de diálogo del chatbot, siguiendo patrones conversacionales que guían progresivamente a los usuarios a explorar los datos. El punto particular que tiene CHATIDEA es que hace uso y manipulación de base de datos relacionales para la creación de una interfaz conversacional. Mi proyecto abarcaría el uso de la web semántica y los datos ya existentes en Wikidata, enfoque que CHATIDEA no tiene en cuenta.

Capítulo 2: Trabajo realizado de desarrollo

2.1 Introducción

Una de las principales motivaciones de este trabajo es la falta de herramientas dedicadas a la recolección de datos en la web semántica usando lenguaje natural. La ventaja de trabajar usando interfaces conversacionales es que podemos utilizar su accesibilidad, el usuario puede realizar preguntas y obtener resultados sin necesidad de conocimientos especializados en consultas de bases de datos o lenguajes de programación. El objetivo de este trabajo es entonces proveer una herramienta que facilite a los usuarios la consulta de información provista en la web semántica, enfocado particularmente en Wikidata. También debe poder acoplarse con la API de Telegram, ya que utilizaremos los servicios que este provee. Dado que es necesario

realizar procesamientos específicos con la pregunta en lenguaje natural de los usuarios, se realizó un trabajo de investigación e implementación sobre posibles herramientas que nos permitan extraer entidades en la pregunta del usuario, además de poder compararla semánticamente y sintácticamente con una serie de preguntas ya existentes en QAWiki.

2.2 Retos propuestos

El presente trabajo contiene una serie de retos propuestos a cumplir para que el desarrollo sea completo y exitoso. Estos retos son:

- Crear una interfaz conversacional, mediante el desarrollo del software.
- Identificar sí los datos que el usuario ingresó corresponden a una pregunta.
- Conectarse y consumir el ecosistema de Wikidata, QAWiki y Template realizando las peticiones correspondientes.
- Crear un algoritmo para la identificación de preguntas similares, ya cargadas en QAWiki, a la pregunta que el usuario realizó en lenguaje natural.
- Notificar a los desarrolladores de QAWiki la falta de consultas SPARQL para las preguntas en lenguaje natural solicitadas, así como también cualquier falla o evento inesperado que ocurra en el sistema.
- Registrar la pregunta inicial que el usuario realizó como un alias a la pregunta ya cargada en QAWiki, que el usuario eligió como alternativa válida a su pregunta.
- Presentar la información al usuario final de forma legible en el marco de la interfaz convencional.

2.3 Resultados esperados

A raíz de los retos propuestos, se esperan una serie de resultados como producto final del desarrollo. Estos resultados son:

- Una interfaz conversacional que consuma los templates de consultas en lenguaje natural disponibles en QAWiki.
- Un algoritmo para procesar y presentar la información solicitada y obtenida de QAWiki en un formato legible para el usuario.
- Una estrategia de alias para incrementar el ecosistema de Wikidata + QAWiki + Templet a las preguntas ya existentes.
- Una estrategia de soporte para los responsables del ecosistema Wikidata + QAWiki + Templet.
- Una estrategia para obtener preguntas relacionadas a las preguntas que los usuarios realizan.
- Un flujo de conversación utilizando preguntas de posible interés respecto a las pregunta que el usuario realizó.

2.4 Escenarios de uso

A lo largo del desarrollo, se analizaron los posibles casos de uso cuando un usuario utiliza el sistema creado y envía un determinado texto. Se debe tener en cuenta que el sistema realizado hace uso de un almacenamiento temporal, llamado Caché, en donde se almacenan las preguntas y respuestas que previamente el sistema procesó con éxito, esto se detalla con más detalle en la sección siguiente. Además, el sistema desarrollado hace uso de un contexto conversacional, el cual mantiene la información relevante sobre la interacción actual entre el usuario y el sistema. Este contexto es fundamental para que el sistema pueda comprender las solicitudes y respuestas del usuario de manera adecuada y proporcionar una experiencia de conversación coherente y personalizada.

El Anexo 2 ofrece un pantallazo general del flujo de la interacción. A continuación se detalla cada escenario de uso y la respuesta esperada del sistema:

Escenario 1 - El texto ingresado se cataloga como pregunta válida.

El usuario envía un texto mediante la interfaz de Telegram y el sistema lo recibe como un mensaje nuevo. Luego, el texto es analizado y se determina que es una pregunta válida para intentar encontrar una respuesta. A raíz de esto pueden suceder dos escenarios posibles, dependiendo de las preguntas que se encuentren almacenadas en el sistema:

Escenario 1.1 - El texto ingresado se cataloga como pregunta válida y está almacenada en el sistema.

La pregunta del usuario fue exactamente igual a una pregunta realizada anteriormente, en un periodo de tiempo menor a 1 hora. Por este motivo, el sistema recupera la pregunta y su respuesta, establece la pregunta como parte del contexto y retorna la respuesta al usuario.

Escenario 1.2 - El texto ingresado se cataloga como pregunta válida y no está almacenada en el sistema.

La pregunta del usuario no fue exactamente igual a una pregunta realizada anteriormente en un periodo de tiempo menor a 1 hora. Por este motivo, el sistema realiza la búsqueda de la pregunta en el sistema de QAWiki y la correspondiente obtención de preguntas similares a la que el usuario realizó. Esto ocasiona 2 posibles escenarios de uso:

Escenario 1.2.1 - El texto ingresado se cataloga como pregunta válida, no está almacenada en el sistema y está en el sistema QAWiki.

Se encontró exitosamente la pregunta cuando se busco en QAWiki, esto permite a nuestro sistema obtener el identificador de la pregunta, para posteriormente buscar en QAWiki el detalle de la misma y así obtener su correspondiente consulta SPARQL. Utilizando dicha consulta puede ocurrir dos escenarios de uso:

Escenario 1.2.1.1 - El texto ingresado se cataloga como pregunta válida, no está almacenada en el sistema, está en el sistema QAWiki y existe consulta SPARQL.

Se pudo obtener la consulta SPARQL desde QAWiki. El sistema procede a realizar la búsqueda en Wikidata haciendo uso de dicha consulta y se parsea la respuesta para presentarla al usuario. Si dentro de la búsqueda o el parseo de la respuesta ocurre un error se notifica al usuario encargado del mantenimiento del sistema y se otorga un feedback al usuario sobre lo sucedido. En caso de que la búsqueda y el parseo de la respuesta sea exitoso, el sistema procede a almacenar temporalmente la pregunta y su respuesta, y, en el contexto de la conversación, la pregunta del usuario realizada, para luego finalmente presentar la respuesta al usuario.

Escenario 1.2.1.2 - El texto ingresado se cataloga como pregunta válida, no está almacenada en el sistema, está en el sistema QAWiki y no existe consulta SPARQL.

No se pudo obtener la correspondiente consulta SPARQL de la pregunta del usuario. Dado este escenario, se notifica al usuario mantenimiento del sistema sobre el problema y se otorga al usuario un feedback relacionado al error obtenido.

Escenario 1.2.2 - El texto ingresado se cataloga como pregunta válida, no está almacenada en el sistema y no está en el sistema QAWiki.

No se encontró exitosamente la pregunta cuando se busco en QAWiki, a raíz de esto el sistema utiliza las preguntas similares obtenidas de la búsqueda fallida en QAWiki para obtener una pregunta que posiblemente sea útil para el usuario. Una vez obtenida la pregunta similar más relevante, el sistema obtiene su consulta SPARQL y la modifica para obtener la correspondiente consulta SPARQL a la pregunta del usuario. Luego el sistema usa la consulta SPARQL modificada en Wikidata y se obtiene el resultado, posteriormente el sistema procesa esta respuesta y se la presenta al usuario junto con dos opciones personalizadas para recibir un feedback sobre la utilidad de la respuesta otorgada. Estas opciones son “the answer was helpful” y “this did not help me”. Esto habilita los siguientes escenarios de uso:

Escenario 1.2.2.1 - La respuesta fue útil.

El usuario otorgó un feedback positivo seleccionando la opción “the answer was helpful”, por lo que el sistema procede a almacenar temporalmente la pregunta con su respectiva respuesta, además de agregar la pregunta como alias o como template nuevo en el sistema, dependiendo el tipo de pregunta que se realizó. Además, se guarda en el contexto de la conversación la pregunta que realizó el usuario. Finalmente el sistema le responde al usuario un mensaje personalizado indicando el caso exitoso.

Escenario 1.2.2.2 - La respuesta no fue útil.

El usuario otorgó un feedback negativo seleccionando la opción “this did not help me”, por lo que el sistema no realiza ningún procedimiento con la pregunta y su respectiva respuesta otorgada, y le responde al usuario un mensaje personalizado.

En caso de que haya habido un error inesperado en los escenarios de uso, se notifica al usuario mantenimiento del sistema y se retorna al usuario un feedback de qué ocurrió dicho error.

Escenario 2 - El texto ingresado no se cataloga como pregunta válida.

El usuario envía un mensaje de texto mediante la interfaz de Telegram y el sistema lo recibe como un mensaje nuevo. Luego, el texto es analizado y se determina que no es una pregunta válida. A raíz de esto pueden suceder dos escenarios posibles, dependiendo del contexto actual en el que usuario envió dicho texto:

Escenario 2.1 - El texto ingresado no se cataloga como pregunta válida y no existe pregunta en contexto.

El usuario no había realizado una pregunta anteriormente que haya tenido una respuesta exitosa, por lo que, en el contexto de la conversación, no existe una pregunta previamente guardada. Debido a esto, el sistema le responde al usuario la necesidad obligatoria de que el texto empiece con una determinada sintaxis para ser catalogada como una pregunta.

Escenario 2.2 - El texto ingresado no se cataloga como pregunta válida y existe una pregunta en contexto.

El usuario había realizado una pregunta con respuesta exitosa previamente, la cual el sistema guardó en el contexto de la conversación. Para este caso, se utiliza esta pregunta contextual para adaptarla al texto del usuario.

El sistema responde con un error personalizado en caso de que no se haya podido obtener información con la pregunta contextual adaptada, además de notificar al usuario mantenimiento del sistema el problema ocurrido.

El sistema responde con éxito si se encontraron resultados, retornando la respuesta a la pregunta contextual adaptada, además de presentarle al usuario dos opciones personalizadas para recibir un feedback sobre la utilidad de la respuesta otorgada. Estas opciones son “the answer was helpful” y “this did not help me”, y habilitan el Escenario 1.2.2.1 y Escenario 1.2.2.2

Escenario 3 - El texto ingresado es un comando

Este escenario de uso es particular ya que el usuario desea realizar alguna acción en particular relacionado al sistema. A continuación se detallaran los comandos que el usuario puede realizar desde la interfaz de Telegram y la respuesta del sistema:

- /start: el sistema retorna “Type a question to start”.
- /help: el sistema retorna 'You have to send question in English starting with "what","which", "where", "when", "how", "is", "did", "do", "in", "who", "on" ,"kim", "from", "has", "was" or "are”’
- /templates_update: el sistema actualiza el listado de plantillas sobre las preguntas cargadas en el sistema de QAWiki, plantillas que se generan automáticamente cada una hora en el sistema. Es una forma de forzar

la actualización manualmente. Una vez finalizada la actualización se informa al usuario del escenario exitoso.

- /answers_reset: el sistema borra las preguntas cacheadas, este proceso se realiza cada una hora de forma automática. Es una forma de forzar la limpieza de la caché manualmente.

2.5 Procesos en Background

Antes de comenzar a explicar esta sección, es importante explicar que para el correcto funcionamiento del sistema se tomó la decisión de utilizar una caché para almacenar las preguntas y respuestas realizadas previamente en el sistema. Se trata de una estrategia para almacenar datos previamente obtenidos.

Esta decisión se tomó por varias razones importantes que benefician tanto a los usuarios como al sistema en sí [12] [13] [14]. Al almacenar respuestas en una caché, el rendimiento del sistema mejora significativamente, esto se debe a que evitamos tener que recalcular la misma respuesta repetidamente. En lugar de ejecutar procesos costosos, como consultas a bases de datos, simplemente recuperamos la respuesta almacenada en la caché, lo que acelera la entrega de esa información. Además, al reducir la necesidad de repetir estos procesos costosos, también reducimos la carga en los recursos del sistema, esto significa que el sistema puede manejar más solicitudes simultáneas sin degradación del rendimiento. Además, al no depender de recursos externos, aumentamos la disponibilidad del sistema, siempre y cuando la información necesaria esté en la caché, el sistema puede continuar funcionando incluso si esos recursos externos están temporalmente no disponibles. Sin embargo, es importante gestionar la caché adecuadamente para garantizar la consistencia de los datos. Esto se logra mediante estrategias como la expiración de la caché, que nos permiten controlar cuándo y cómo se actualizan los datos almacenados.

El proyecto desarrollado realiza dos tareas fundamentales mientras los usuarios están interactuando con el mismo:

- Borrado de caché de respuestas: Cada una hora el sistema borra su caché interna, la cual posee las preguntas con respuestas válidas hechas previamente. La caché permite obtener la respuesta de una pregunta que se hizo exactamente igual sin la necesidad de realizar la comunicación con aplicaciones terceras, obteniendo una respuesta mas rapida. El motivo por el cual se decide realizar este borrado con un periodo de una hora es porque la información que hay en la web puede cambiar en cualquier momento, otorgando resultados diferentes para la misma pregunta. Con esta tarea nos cercioramos que la respuesta del sistema sea siempre actualizada, siempre y cuando la respuesta no haya cambiado en un periodo menor a una hora. De esta forma se encuentra un equilibrio entre tener eficiencia en el sistema mediante el uso de una caché, y obtener información actualizada.

- Actualización de preguntas: El proyecto realizado necesita estrictamente el uso de las preguntas que se encuentran en el sistema QAWiki ya que se utilizan para obtener la respuesta en la web semántica. Haciendo uso de los beneficios

de la caché, se tomó la decisión de almacenar dichas preguntas en la caché y, así, lograr una mayor eficiencia del sistema. Esta solución también implica almacenar los alias de las preguntas, logrando una mayor cobertura con las preguntas que el usuario pudiera realizar. Además, el almacenado de dichas preguntas incluye la correspondiente consulta SPARQL y su plantilla.

Hoy en día, QAWiki está en constante crecimiento dado que sus desarrolladores agregan preguntas nuevas con frecuencia, debido a esto, nuestro sistema necesita tener actualizaciones constantes de las preguntas que posee, manteniendo el catálogo de preguntas de QAWiki siempre actualizado. Para ello se decidió realizar una tarea que, cada una hora, realice el procesamiento necesario para obtener todas las preguntas de QAWiki y actualizar la caché del proyecto.

2.6 Selección de plataforma de base

Se realizó una investigación teniendo en cuenta las herramientas de diseño de interfaces conversacionales más utilizadas, en busca de alguna que nos permita diseñar una sin muchas complicaciones. Una de las principales condiciones que tenía que tener la herramienta es contar con la posibilidad de guardar un contexto de la conversación, ya que esto es primordial a la hora de lograr una conversación entre el usuario y el sistema. Asimismo la herramienta debía proveernos una forma de interactuar con el sistema desarrollado para procesar la pregunta en lenguaje natural del usuario y retornar la respuesta adecuada al mismo.

En esta sección se explicarán en detalle la herramienta elegida, Telegram, con el objetivo de utilizar dicha plataforma donde los usuarios puedan realizar preguntas en lenguaje natural y obtener respuestas con una experiencia de usuario satisfactoria.

Telegram

Telegram es una plataforma de mensajería de origen ruso, desarrollada por Nikolái y Pável Dúrov. La aplicación está enfocada en la mensajería instantánea, el envío de varios archivos y la comunicación en masa. El servicio ofrece funcionalidades enfocadas en realizar charlas entre usuarios. Salvo excepciones, los mensajes son almacenados, o archivados en caso de que se desee ocultarlos en la nube, posee opción de reenvío, borrado temporizado, acciones de búsqueda, opciones clickeables personalizables, etc.

Para la automatización de tareas masivas se desarrollan bots, que realizan actividades y servicios extras como pagos, juegos, moderación de grupos o asignación de otras tareas bajo inteligencia artificial. Los bots se aplican también en el ámbito empresarial y social.

Inicialmente el servicio fue empleado para teléfonos móviles Android e iOS y el año siguiente para múltiples plataformas: macOS, Windows, GNU/Linux, Firefox OS, navegadores web, y otras. Las aplicaciones usan la interfaz de acceso gratuito, permitiendo a los desarrolladores crear clientes externos. Hay clientes con licencia libre (GNU GPL versión 2 y 3) pero el servidor es software privativo. La aplicación soporta varios idiomas. La adaptación a los cambios de diseño y la facilidad de uso influyó en que hoy en día sea una de las aplicaciones de mensajería más utilizadas.

En particular, Telegram posee dos características destacables:

- Bots y Desarrollo de API: Los bots están optimizados en realizar servicios de mayor carga, diferenciándose de los usuarios corrientes. Tienen capacidad de obedecer comandos avanzados de texto solamente, administrar canales y grupos, compartir contenido, elaborar votaciones en tiempo real y ejecutar juegos y pagos. Algunos de ellos son bots integrados para compartir contenido sin salir de la aplicación. Mediante interacciones como comandos o archivos multimedias, los bots se aplican en diversas actividades. Pueden ser programados para enviar mensajes automáticos, proporcionar información, realizar tareas específicas, interactuar con usuarios, etc. Los usuarios pueden agregar bots a sus chats y grupos para mejorar la funcionalidad y recibir servicios automatizados.

  Para crear un bot en Telegram, los usuarios interactúan con el "BotFather", que es el bot oficial de Telegram para la creación y gestión de bots. BotFather guía a los usuarios a través del proceso de creación del bot, asigna un token de acceso único y proporciona comandos para personalizar el bot.

  Además, Telegram ofrece una API abierta que permite a los desarrolladores crear aplicaciones personalizadas y servicios que se integran con la plataforma. Esta API proporciona acceso a diversas funciones de Telegram, como el envío de mensajes, la gestión de grupos, la creación de bots, etc. Los desarrolladores pueden utilizar la API para crear experiencias de usuario particulares y agregar funcionalidades específicas a sus aplicaciones. Esta funcionalidad, junto con la implementación de bot, es crucial en el desarrollo del proyecto hecho, ya que representa la base del sistema.

- Multiplataforma: Puedes utilizar Telegram en una variedad de dispositivos, incluyendo teléfonos móviles (iOS y Android), tabletas y computadoras. Además, la aplicación se puede utilizar en navegadores web sin necesidad de instalar software adicional. Esto permite que el sistema creado pueda utilizarse en más de una plataforma.

2.7 Arquitectura de la solución

Esta sección pretende documentar las decisiones de diseño y arquitectura de la solución utilizadas en este trabajo, justificando y detallando los caminos elegidos hasta llegar al armado de la solución completa. Además, mencionar varias de las técnicas y conceptos que fueron necesarios para la construcción de la herramienta.

Como se describió en capítulos anteriores, utilizaremos Telegram como herramienta de prototipado, ya que presenta las siguientes ventajas:

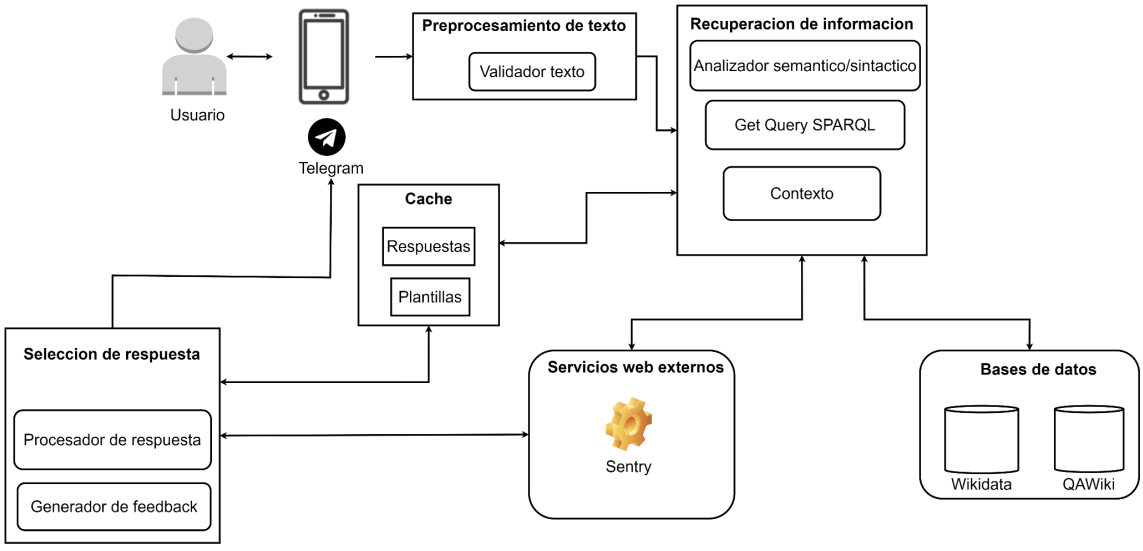
- Automatización de tareas: Los bots de Telegram pueden realizar diversas tareas de forma automatizada, como enviar mensajes programados, realizar búsquedas en la web, enviar recordatorios, entre otras funciones. Esto es particularmente útil ya que dentro del desarrollo se realizó automatización de tareas.

- Interacción con usuarios: Los bots permiten la interacción directa con usuarios a través de la plataforma de Telegram. Esto permite que no sea necesario el desarrollo de la interfaz gráfica.
- Notificaciones y alertas: Los bots pueden enviar notificaciones y alertas a los usuarios, manteniéndolos informados sobre eventos particulares. Esto se planteó dentro del desarrollo, permitiendo notificar al usuario en caso de que hubiera un evento inesperado.
- Integración con servicios externos: Los bots pueden integrarse con servicios externos, como bases de datos, API web, etc. Esto permite una amplia gama de funcionalidades personalizadas, ventaja que se utilizó integrando el sistema realizado.
- Desarrollo ágil: La plataforma de desarrollo de bots de Telegram es sencilla y está bien documentada, lo que facilita la creación y mejora continua de bots. La sencillez para desarrollarlo fue un punto a favor a tomar en cuenta.
- Escalabilidad: Los bots pueden manejar interacciones con una gran cantidad de usuarios simultáneamente, lo que permite la escalabilidad en caso de que la popularidad del bot aumente. La posibilidad de administrar la concurrencia de manera transparente al programador facilita el desarrollo.

El proyecto desarrollado está compuesto por una aplicación backend, que contiene la lógica de almacenamiento y manipulación de la información. A continuación detallaremos cada una de las partes que componen la herramienta, se mencionara tecnologías utilizadas y consideraciones técnicas. Finalizamos esta sección con una demostración del flujo completo necesario para reproducir los escenarios planteados en la sección “Escenarios de uso”.

Diagrama de arquitectura

A continuación se mostrará el diagrama de arquitectura del proyecto realizado, mostrando como un usuario interactúa, a través de Telegram, con el sistema backend realizado, el cual valida y procesa la información recibida, se conecta con servicios externos y bases de datos, a su vez que gestiona la cache de mismo y administra el contexto de la conversación.



Tecnologías utilizadas

El sistema de backend fue realizado utilizando el lenguaje de programación Python, ya que este presenta librerías de utilidad para las funciones que se requirieron, además de presentar una sintaxis fácil y cómoda para el desarrollador experimentado. A continuación se detallaran las librerías y herramientas utilizadas.

- ### Telegram-bot

Es una biblioteca de Python que proporciona una interfaz para interactuar con la API de Telegram. Esta biblioteca facilita la creación de bots de Telegram en Python, permitiendo a los desarrolladores realizar diversas operaciones como enviar mensajes, recibir actualizaciones, manejar comandos, gestionar teclados personalizados, etc. Para usar la API de Telegram, se necesita un token de bot. Para obtenerlo se debe interactuar con un bot de telegram llamado "BotFather", como se mencionó anteriormente, y seguir las instrucciones. La biblioteca utiliza un objeto llamado Updater para recibir actualizaciones de Telegram y distribuirlas a los controladores correspondientes. Este objeto se encarga de gestionar la conexión con el servidor de Telegram y proporciona una interfaz para registrar manejadores de eventos. Se debe registrar manejadores de eventos para responder a comandos, mensajes, etc, facilitando el envío de mensajes de texto, imágenes y documentos. También proporciona un objeto Context que se pasa a los manejadores y permite el intercambio de datos entre ellos, además de permitir mantener un contexto en la conversación.

- ### APScheduler

APScheduler es una biblioteca de Python que proporciona un programador de tareas configurable que se puede utilizar para automatizar la ejecución de tareas en momentos específicos o en intervalos regulares. Es una librería flexible y puede utilizar varios backends de almacenamiento para almacenar la información de las tareas programadas, algunos ejemplos de backends son SQLite, SQLAlchemy, MongoDB y Redis. Proporciona diferentes tipos de desencadenadores para ejecutar tareas en momentos específicos, a intervalos regulares, o de acuerdo con reglas personalizadas. Es posible especificar detalles como el tiempo de inicio, la hora de finalización, la zona horaria, entre otros, además de incluir un sistema de manejo de excepciones para controlar errores durante la ejecución de las tareas programadas.

- ### Sentry

Sentry es una plataforma de monitoreo de errores y seguimiento de problemas en aplicaciones, proporciona integración con la plataforma Sentry para facilitar el registro y el seguimiento de errores en aplicaciones. Es necesario configurar el proyecto con el DSN (Data Source Name) proporcionado por Sentry, este DSN es un identificador único para el proyecto en Sentry. La librería automáticamente captura excepciones no manejadas en la aplicación y las envía a Sentry, se puede enriquecer los informes de errores agregando contexto adicional, esto podría incluir información del usuario, datos de solicitud, etc. Incluso se puede personalizar la forma en que Sentry recopila y maneja los errores mediante la configuración avanzada, esto podría incluir la configuración de filtros, ajustes del nivel de registro, etc. Además de errores, se puede enviar eventos personalizados a Sentry para realizar un seguimiento de eventos específicos en la aplicación.

- **Nltk**

NLTK (Natural Language Toolkit) es una librería en Python diseñada para trabajar con procesamiento de lenguaje natural. Proporciona herramientas y recursos para trabajar con texto y lenguaje, facilitando tareas como tokenización, análisis sintáctico, etiquetado gramatical, extracción de información, etc. NLTK ofrece funciones para dividir el texto en palabras o frases, llamada tokenización, que divide un texto en unidades más pequeñas, como palabras o frases. NLTK proporciona una lista de palabras comunes conocidas como "stopwords" que son generalmente ignoradas durante el análisis de texto, ya que no aportan mucha información significativa. Se puede utilizar la lista de stopwords de NLTK para filtrar estas palabras.

Esta librería utiliza pos_tag y ne_chunk, que son funciones proporcionadas por NLTK utilizadas en el procesamiento de lenguaje natural. La función pos_tag se utiliza para asignar etiquetas gramaticales a cada palabra en un texto, estas etiquetas indican la categoría gramatical a la que pertenece cada palabra, como sustantivos, verbos, adjetivos, etc. Esta información es útil para entender la estructura sintáctica y semántica de un texto. La función ne_chunk se utiliza para el etiquetado de entidades nombradas (NER), que implica identificar y clasificar entidades como nombres de personas, organizaciones, lugares, fechas, etc. Esta función agrupa las palabras relacionadas en fragmentos que representan entidades nombradas. Estas dos funciones son esenciales en el análisis de texto y proporcionan información valiosa para tareas avanzadas en procesamiento de lenguaje natural.

- **Spacy**

Spacy es una biblioteca de procesamiento de lenguaje natural (NLP) para Python que proporciona herramientas eficientes y rápidas para realizar tareas relacionadas con el procesamiento de texto. Spacy realiza la tokenización de manera eficiente, dividiendo el texto en unidades más pequeñas llamadas "tokens". Estos tokens pueden ser palabras, signos de puntuación u otros elementos dependiendo del contexto. Asigna etiquetas gramaticales a cada token, indicando su parte (sustantivo, verbo, adjetivo, etc), esto es útil para comprender la estructura gramatical de una oración. Proporciona lemas para cada palabra, lo que facilita la normalización y el análisis del texto, esto corresponde al proceso de reducir las palabras a sus formas base. Además, Spacy es capaz de identificar y clasificar entidades en el texto, como nombres de personas, organizaciones, lugares, fechas, entre otros, esto es esencial para tareas de extracción de información. Analiza las dependencias sintácticas entre las palabras en una oración, mostrando las relaciones gramaticales entre ellas, esto es útil para comprender la estructura y el significado de una oración.

Spacy proporciona vectores de palabras pre-entrenados que representan la semántica de las palabras en un espacio vectorial. Estos vectores pueden ser utilizados para medir similitudes semánticas entre palabras. Ofrece modelos pre entrenados para varios idiomas que se pueden utilizar directamente, estos modelos son entrenados en grandes cantidades de datos y son capaces de realizar diversas tareas de procesamiento del lenguaje natural.

- **Fuzzywuzzy**

La biblioteca fuzzywuzzy en Python es una herramienta útil para comparar cadenas de texto y calcular la similitud entre ellas mediante algoritmos de coincidencia difusa. Esta biblioteca facilita la comparación de cadenas incluso cuando hay

pequeñas diferencias o errores tipográficos. Utiliza varios algoritmos de coincidencia difusa, como el algoritmo de distancia de Levenshtein (también conocido como distancia de edición) y otros métodos similares. La distancia de Levenshtein mide la cantidad mínima de ediciones necesarias para convertir una cadena en otra (inserciones, eliminaciones o sustituciones de caracteres). La biblioteca proporciona una puntuación de similitud, que indica cuán similares son dos cadenas., cuanto mayor sea la puntuación, mayor será la similitud entre las cadenas. Esta librería ofrece varias funciones comunes para la comparación de cadenas, estas funciones permiten realizar comparaciones basadas en la longitud de las cadenas, en el orden de los tokens, o en un conjunto de tokens único.

Fuzzywuzzy puede ser eficaz, pero el rendimiento puede ser un problema si se utiliza para comparar grandes conjuntos de datos, ya que la complejidad computacional de algunos algoritmos de coincidencia difusa puede ser alta. La biblioteca puede necesitar ajustes o combinarse con otras técnicas de procesamiento de texto según el caso de uso específico.

- ### Docker

Docker es una plataforma de software que facilita la creación, implementación y administración de aplicaciones en entornos virtualizados llamados contenedores. Estos contenedores permiten empaquetar una aplicación y todas sus dependencias en una unidad única y portátil. Docker simplifica el desarrollo y la implementación de aplicaciones al proporcionar una forma eficiente y consistente de empaquetar, distribuir y ejecutar software en entornos contenerizados. Esto mejora la portabilidad, la escalabilidad y la eficiencia del desarrollo y operaciones de software.

- ### Git

Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en el código fuente durante el desarrollo de software. Git permite realizar un seguimiento de los cambios en el código fuente a lo largo del tiempo, esto significa que puedes revertir a versiones antiguas, comparar cambios, y trabajar en múltiples ramas de desarrollo de manera simultánea. A diferencia de otros sistemas de control de versiones centralizados, Git es distribuido, cada desarrollador tiene una copia completa del repositorio, incluyendo el historial completo de cambios, permitiendo trabajar de manera independiente y fusionar los cambios más tarde. Git permite crear ramas para trabajar en paralelo sin afectar el código en otras ramas, útil para desarrollar nuevas características, arreglar errores o realizar experimentos sin tocar el código principal. Git además te permite especificar archivos y carpetas que deben ser ignorados y no incluidos en el control de versiones, así evitar subir archivos temporales, archivos generados por compilación, o archivos sensibles. Existen servicios, como GitHub, GitLab y Bitbucket, que ofrecen almacenamiento remoto de repositorios Git y herramientas adicionales para la colaboración en equipo, seguimiento de problemas, integración continua, etc.

2.8 Descripción de actividades del sistema

El sistema desarrollado realiza diversas actividades, como se puede apreciar en el diagrama de flujo, a continuación se detallaran cada una de dichas actividades

para lograr un mayor entendimiento de cómo funciona el sistema para cada caso de uso posible. Antes de comenzar es necesario aclarar que cuando se habla de “respuesta personalizada” se hace mención a las opciones clickeables que el sistema otorga para que el usuario las utilice como feedback al sistema, tales como “this did not help me”, “the answer was helpful”, “goodbye”, etc.

A continuación, se realizó una tabla donde cada fila describe una actividad en particular además de especificar, en caso de ser necesario, la actividad previa realizada, el flujo al cual pertenece y que flujos genera.

Actividad	Descripción	Previa actividad	Proviene de flujo	Crea flujos
Se analiza si es una respuesta personalizada	Cuando el usuario envía un texto al sistema puede ser un texto que escribió, o una opción que el sistema previamente le otorgó y el usuario eligió. Para ello se analiza si el texto enviado pertenece a una de las opciones personalizadas (strings) que el sistema ofrece.			La respuesta no es personalizada. La respuesta es personalizada.
Se analiza si es una pregunta válida	Una vez que se identifica que el texto que envió el usuario no es una respuesta personalizada, es decir, es un texto que escribió el usuario por su cuenta, es necesario saber si el texto que el usuario envió es una pregunta válida. Cuando se habla de pregunta válida hace referencia a que el usuario realizó una pregunta sintácticamente correcta. Para ello, al ser un sistema puramente en idioma inglés, se verifica que el texto comience con las palabras "what", "which", "where", "when", "how", "is", "did", "do", "in", "who", "on", "from", "has", "was" y "are".	Se analiza si es una respuesta personalizada	La respuesta no es personalizada	La pregunta es válida. La pregunta no es válida
Se analiza si la pregunta está cacheada	El sistema posee una caché interna, la cual consiste de un archivo con extensión .json como se mencionó anteriormente. En esta caché almacenamos las preguntas con su respectiva información relevante para el sistema y el usuario, esto con el fin de otorgar una respuesta al usuario más rápida y que el sistema sea más eficiente. Lo que se realiza es una búsqueda en esta caché por la pregunta en busca de una coincidencia exacta en cuanto a la sintaxis de la pregunta.	Se analiza si es una pregunta válida	La pregunta es válida	La pregunta está cacheada La pregunta no está cacheada

Se solicita todas las preguntas y alias en QAWiki	En este punto del flujo se determinó que el usuario envía una pregunta válida que no se encuentra cacheada en el sistema, por lo que se procede a buscar en el sistema de QAWiki la pregunta exacta que hizo el usuario. Para ello se tiene en cuenta las preguntas cargadas en el sistema y también los posibles alias que cada pregunta posee ya que corresponden al mismo concepto. La búsqueda se realiza usando la API que provee QAWiki, es decir, es necesaria la conexión con un servidor de otro sistema. Esto puede causar una bifurcación ya que puede fallar dicha conexión.	Se analiza si la pregunta está cacheada	La pregunta no está cacheada	La respuesta del servidor fue exitosa. La respuesta del servidor no fue exitosa
Se busca la pregunta literal en la respuesta	Dado que el sistema respondió exitosamente, se procede a buscar en la respuesta de QAWiki si existe una coincidencia exacta con la pregunta que el usuario realizó. Esta búsqueda deriva en una bifurcación del flujo ya que la pregunta del usuario puede estar o no dentro de la respuesta de QAWiki	Se solicita todas las preguntas y alias en QAWiki	La respuesta del servidor fue exitosa	Se encontró la pregunta en QAWiki. No se encontró la pregunta en Qawiki
Se obtiene QID y se busca entidad en QAWiki	Al encontrarse la pregunta en el sistema de QAWiki, éste en su respuesta nos otorga el QID para poder buscar en particular la entidad (pregunta en QAWiki) y así obtener su correspondiente consulta en lenguaje SPARQL, query necesaria para obtener la respuesta desde Wikidata. Realizamos la búsqueda en particular de la entidad con el identificador QID en QAWiki. La búsqueda se realiza usando la API que provee QAWiki, es decir, es necesaria la conexión con un servidor de otro sistema. Esto puede causar una bifurcación ya que puede fallar dicha conexión	Se busca la pregunta literal en la respuesta	Se encontró la pregunta en QAWiki	La respuesta del servidor de QAWiki fue exitosa La respuesta del servidor de QAWiki no fue exitosa.
Se obtiene query SPARQL y preguntas relacionadas desde QAWiki	El servidor de QAWiki respondió exitosamente, por lo que se procede a buscar dentro de la información obtenida la correspondiente consulta SPARQL de la pregunta, además de posibles preguntas	Se obtiene QID y se busca entidad en QAWiki	La respuesta del servidor de QAWiki fue exitosa	Se obtuvo la consulta SPARQL en la respuesta. No se obtuvo la consulta

	relacionadas que QAWiki puede tener cargadas para la pregunta. La respuesta debería tener la query SPARQL ya que es el objetivo del sistema QAWiki, pero como es un sistema que se carga la información manualmente esto puede causar que haya una correcta carga de información, creando la posibilidad de que no se encuentre la correspondiente consulta SPARQL en la respuesta del servidor.			SPARQL en la respuesta.
Se obtiene resultado con consulta en Wikidata	Con la correspondiente query SPARQL de la pregunta del usuario, se procede a utilizar el servicio de Wikidata para realizar la consulta por la respuesta para el usuario. La búsqueda se realiza usando la API que provee Wikidata, es decir, es necesaria la conexión con un servidor de otro sistema. Esto puede causar una bifurcación ya que puede fallar dicha conexión.	Se obtiene query SPARQL y preguntas relacionadas desde QAWiki	Se obtuvo la consulta SPARQL en la respuesta	La respuesta del servidor de Wikidata fue exitosa. La respuesta del servidor de Wikidata no fue exitosa
Se parsea respuesta de Wikidata	La respuesta de Wikidata posee información con diferentes tipos de datos, por lo que es necesario procesar la respuesta adecuadamente. Dentro de los posibles tipos de datos se encuentra URI a otras entidades, es decir, referencias a otras entidades relevantes para la respuesta. Esto puede generar una bifurcación en el flujo ya que si se encuentran URI hay que procesarlas también.	Se obtiene resultado con consulta en Wikidata	La respuesta del servidor de Wikidata fue exitosa	La respuesta posee uris. La respuesta no posee uris
Se arma query SPARQL con las uri	La respuesta de Wikidata posee información de referencias a otras entidades, las cuales deben ser procesadas, para ello se arma la query SPARQL correspondiente y se repite recursivamente la obtención del resultado con la query en Wikidata y su respectivo parseo. Esta recursión termina hasta que no haya URIs relevantes en la respuesta de Wikidata.	Se parsea respuesta de Wikidata	La respuesta posee uris	
Se analiza si la pregunta ya existe	Una vez procesada la respuesta de Wikidata, ya obtuvimos toda la información relevante para la respuesta al usuario. El siguiente paso es cachear la respuesta, y	Se parsea respuesta de Wikidata	La respuesta no posee uris	La pregunta no existe en caché.

en la caché	para ello se consulta en la caché si ya existe la pregunta, esto se debe a que puede existir otro usuario que haya realizado la misma pregunta, es decir, existe una concurrencia y se debe asegurar que la caché no contenga preguntas duplicadas. La búsqueda puede causar una bifurcación ya que puede encontrarse o no encontrarse la pregunta en la caché.			La pregunta existe en caché, o no existe pero ya se guardo
Se guarda respuesta en caché	Se procede a guardar la información de la pregunta y de su correspondiente respuesta en la caché.	Se analiza si la pregunta ya existe en caché	La pregunta no existe en caché	
Se guarda respuesta en contexto	El sistema maneja un contexto de la conversación debido al uso de la librería de Telegram. Esto permite al usuario poder realizar más preguntas relacionadas a la hecha sin necesidad de mantener una sintaxis particular para consultar. Para ello guardamos la información en un objeto particular de la librería, que luego se utilizará en otros flujos del sistema.	Se guarda respuesta en caché. Se analiza si la pregunta ya existe en caché	La pregunta existe en caché, o no existe pero ya se guardo	
Se analiza si en la respuesta existen preguntas relevantes	Como se mencionó previamente, la respuesta que nos otorga QAWiki puede contener preguntas relevantes a la que realizó el usuario. Para una mayor usabilidad del sistema se decidió incluir estas posibles preguntas útiles para el usuario como parte de la respuesta, para ello se analiza si existen, ya que dependerá de que en QAWiki estén cargadas las preguntas manualmente. Esto genera una bifurcación en el flujo ya que de existir se deben cargar	Se guarda respuesta en contexto	La pregunta existe en caché, o no existe pero ya se guardo	Existen preguntas relevantes No existen preguntas relevantes, o existe y ya se agregaron como opciones
Se agrega opciones de preguntas a respuesta	Dentro del armado de la respuesta hacia la interfaz de Telegram, para que el usuario pueda ver los resultados, se utiliza un objeto particular de la librería de telegram para agregar las preguntas relevantes como opciones clickeables. Además se incluye una opción clickeable para rechazar dichas opciones.	Se analiza si en la respuesta existen preguntas relevantes	Existen preguntas relevantes	

Se retorna respuesta al usuario	Se retorna la respuesta procesada de Wikidata obtenida con la query SPARQL obtenida de QAWiki. El usuario puede ver la información y las posibles preguntas relevantes.	Se agrega opciones de preguntas a respuesta. Se analiza si en la respuesta existen preguntas relevantes. Se guarda pregunta en contexto	No existen preguntas relevantes, o existe y ya se agregaron como opciones	
Se busca preguntas similares	En caso de que no se haya encontrado la pregunta exacta en QAWiki, se procede a buscar de entre todas las preguntas de QAWiki la pregunta que posea una mayor similitud. La similitud tiene un orden de prioridades, contemplamos primero las preguntas que tengan una mayor similitud semántica, seguido de las que tengan una mayor similitud sintáctica. La búsqueda puede causar una bifurcación ya que puede encontrarse o no encontrarse preguntas similares en QAWiki.	Se busca la pregunta literal en la respuesta	No se encontró la pregunta en QAWiki	Existen preguntas similares. No existen preguntas similares
Se busca entidad principal de la pregunta del usuario	Debido a que se encontró preguntas similares, se puede iniciar el procesamiento de la pregunta que realizó el usuario, ya que esto es esencial y, de lo contrario, esto sería un procesamiento sin utilidad. El objetivo de esta actividad es extraer la entidad de la pregunta del usuario para luego insertarla en la consulta SPARQL de la pregunta similar de QAWiki. Para ello se utilizan algoritmos para la identificación de entidades principales en frases, utilizando las librerías de Spacy y NLTK, descritas anteriormente. La búsqueda puede causar una bifurcación ya que puede encontrarse o no encontrarse la entidad principal.	Se busca pregunta similares	Existen preguntas similares	Se obtiene la entidad principal. No se obtiene la entidad principal
Se busca en Wikidata entidades que contengan	Una vez obtenida la entidad principal de la pregunta, se procede a buscar en wikidata todas las entidades que contengan a la entidad principal de la pregunta del usuario. Por ejemplo, para la pregunta “¿Cuál es la	Se busca entidad principal de la pregunta del usuario	Se obtiene la entidad principal	La respuesta del servidor de Wikidata fue exitosa La respuesta

main entity	vida útil de una naranja?”, su entidad principal va a ser “naranja”, la cual se buscará en Wikidata y traerá entidades que hacen referencia a un color, una fruta, un nombre, etc. La búsqueda se realiza usando la API que provee Wikidata, es decir, es necesaria la conexión con un servidor de otro sistema. Esto puede causar una bifurcación ya que puede fallar dicha conexión			del servidor de Wikidata no fue exitosa.
Filtramos entidades y obtenemos la más similar	De entre todas las entidades que Wikidata retorno, el sistema debe quedarse con la pregunta que sea más similar a la hecha por el usuario, esto se realiza contando la cantidad de coincidencias de los tipos de datos de cada entidad de wikidata, quedando la pregunta que mayor coincidencias tenga. Siguiendo con el ejemplo anterior, nos quedamos con la entidad llamada “naranja” de Wikidata, que hace referencia a una fruta	Se busca en Wikidata entidades que contengan main entity	La respuesta del servidor de Wikidata fue exitosa	
Reemplazamos entidad más similar en pregunta similar	Obtenida la entidad de Wikidata, se debe obtener el identificador correcto para reemplazar en la pregunta similar de QAWiki. Para ello se identifica la entidad principal de la pregunta de QAWiki, esto es fácil gracias a la información que el sistema de QAWiki provee en su respuesta, la cual ya otorga cual es la entidad y su respectiva parte de la query SPARQL. Identificada la parte de la query SPARQL, se reemplaza el QID de la pregunta similar de QAWiki por el QID de la entidad de la pregunta del usuario. De esta forma, se obtuvo una nueva query SPARQL, equivalente a la pregunta que el usuario realizó y no se encontraba textualmente en QAWiki. Esto incrementa el dominio de las preguntas que el usuario puede realizar y que el sistema puede responder.	Filtramos entidades y obtenemos la más similar	La respuesta del servidor de Wikidata fue exitosa	
Se busca en Wikidata resultado usando pregunta	Una vez obtenida la query SPARQL correspondiente a la pregunta del usuario, se procede a buscar en Wikidata la respuesta correspondiente. En este punto del flujo se repite el mismo procedimiento	Reemplazamos entidad más similar en pregunta similar	La respuesta del servidor de Wikidata fue exitosa	

similar	para el procesamiento de la información para el caso de que la pregunta estaba en el sistema de QAWiki, es decir, se parsea la respuesta de Wikidata, en caso de que haya uris se arma la query SPARQL y se obtiene resultados de las mismas en Wikidata. También se contempla el error de conexión con el servicio externo de Wikidata, notificando al usuario y enviando reportes con Sentry			
Se agrega respuestas personalizadas sobre si fue util la respuesta	Luego de obtener la respuesta y procesarla, el sistema agrega respuestas personalizadas para recibir un feedback de parte del usuario y saber si la información que se otorgó fue de utilidad. Para ello se utiliza la librería de Telegram que aporta opciones clickeables, aparte de presentar la información de la respuesta de Wikidata. Luego, el flujo continúa igual que sí la pregunta del usuario se encontraba en QAWiki, es decir, se guarda la respuesta en el contexto de la conversación y se retorna la respuesta final al usuario, terminando el flujo correspondiente.		La respuesta del servidor de Wikidata fue exitosa	
Se guarda pregunta en contexto	En caso de que la pregunta que realizó el usuario está en la caché, este flujo es más corto y sencillo, ya que solo se procede a guardar la pregunta en el contexto y luego se retorna la respuesta de la caché que se encuentra con la pregunta que previamente se encontró. En este punto, termina uno de los flujos del sistema.	Se analiza si la pregunta está cacheada	La pregunta del usuario está cacheada	
Se analiza si existe un contexto	Dado que el texto que envió el usuario no corresponde a una respuesta personalizada del sistema y tampoco es una pregunta válida, se procede a analizar si ya existe una pregunta en el contexto de la conversación, es decir, si el usuario ya había realizado una pregunta con un resultado exitoso. Esto puede causar una bifurcación ya que puede existir una pregunta previa guardada en el contexto de la conversación, o puede que no exista	Se analiza si es una pregunta válida	La pregunta no es válida	Existe pregunta previa guardada en el contexto. No existe pregunta previa guardada en el contexto

Se obtiene la pregunta del contexto	Al existir una pregunta en el contexto, la obtenemos y la procesamos. En este punto del flujo se realiza el mismo procedimiento descrito anteriormente para cuando la pregunta no se encuentra en QAWiki, es decir, se busca la entidad principal de la pregunta del usuario y la entidad de la pregunta en wikidata, se filtra las entidades de wikidata por la que tenga mayor similitud con la entidad de la pregunta de contexto, se reemplaza el QID de la entidad de la pregunta del contexto por el QID de la entidad de la pregunta del usuario, se obtiene la query SPARQL equivalente a la pregunta en lenguaje natural del usuario, se busca la respuesta en wikidata, se procesa la respuesta y las posibles uris de wikidata, se agrega respuestas personalizadas para dar feedback al sistema y se retorna la respuesta al usuario. También se contempla los errores de conexión con el servidor de Wikidata y las fallas de identificación de las entidades principales en las preguntas, notificando al usuario y enviando reportes a Sentry.	Se analiza si existe un contexto	Existe pregunta previa guardada en el contexto	
Se retorna al usuario error pregunta no válida	En caso de que el texto enviado por el usuario no corresponde a una respuesta personalizada del sistema, no es una pregunta válida y tampoco existe una pregunta previa guardada en el contexto de la conversación entonces se procede a notificar al usuario que la pregunta no es válida y cuál es la sintaxis es requerida para poder utilizar el sistema.	Se analiza si existe un contexto	No existe pregunta previa guardada en el contexto	
Se analiza si confirmo respuesta similar útil	Las respuestas personalizadas que el sistema provee incluyen un feedback para determinar si el procesamiento que se realizó fue útil para el usuario final. Se analiza si ésta respuesta personalizada confirma que fue útil la respuesta del sistema a una pregunta similar. Esto puede causar una bifurcación ya que puede que la respuesta haya sido útil o no.	Se analiza si es una respuesta personalizada	Es una respuesta personalizada	Se confirmó respuesta útil. La respuesta del usuario confirma que no fue útil la respuesta del sistema
Se analiza si la	Es necesario verificar que la pregunta que realizó el usuario corresponde a una	Se analiza si confirmo	Se confirmo	La pregunta del usuario es

pregunta del usuario fue un alias o una pregunta nueva	pregunta nueva, o si corresponde a una pregunta que se puede considerar como un alias a otra pregunta que ya existe, ya que de esta forma evitamos repetir información en un registro interno de preguntas que posee el sistema. Para ello se verifica sí se cambió la estructura de la query SPARQL perteneciente a la pregunta del contexto o a la pregunta similar de QAWiki, para consultar por otra entidad diferente. En caso de haberse cambiado, significa que la pregunta realizada por el usuario es diferente y debe agregarse como pregunta nueva en el registro. Caso contrario, se trata de un alias, es decir, la pregunta en lenguaje natural es diferente pero la query SPARQL es la misma a la pregunta que se utilizó como referencia. Esto puede causar una bifurcación ya que puede que la respuesta haya sido un alias o una pregunta nueva.	respuesta similar útil	respuesta útil	un alias a otra pregunta. La pregunta del usuario es una pregunta nueva
Agregar pregunta como alias	Al determinar que la pregunta del usuario corresponde a un alias de otra pregunta existente, se procede a agregar, dentro del registro de preguntas, a la misma dentro de un arreglo de preguntas de alias que posee la pregunta existente	Se analiza si la pregunta del usuario fue un alias o una pregunta nueva	La pregunta del usuario es un alias a otra pregunta	
Agregar pregunta como nueva plantilla	Al determinar que la pregunta del usuario corresponde a una pregunta nueva, se procede agregar en el registro de preguntas a la misma con la estructura adecuada para su correspondiente uso en futuras preguntas por parte del usuario.	Se analiza si la pregunta del usuario fue un alias o una pregunta nueva	La pregunta del usuario es una pregunta nueva	
Retornar feedback exitoso a usuario	Una vez que se determinó que la respuesta del sistema no fue de utilidad para el usuario, o la respuesta lo fue y ya se agregó como alias o como pregunta nueva en el registro de preguntas del sistema, se procede a retornar un feedback al usuario y terminar el flujo del sistema.	Se analiza si confirmo respuesta similar útil Agregar pregunta como alias Agregar pregunta como nueva plantilla	La respuesta del usuario confirma que no fue útil la respuesta del sistema La pregunta del usuario es un alias	

			a otra pregunta. La pregunta del usuario es una pregunta nueva	
Notificar vía Sentry	En caso de un fallo en la conexión con servidores externos, o en caso de que QAWiki no provea la query SPARQL para la pregunta en su sistema, el flujo de nuestro sistema se ve inevitablemente interrumpido. Para estas situaciones mantenemos un control de lo que sucedió notificando a los usuarios encargados del mantenimiento del proyecto. Para ello integramos el servicio de Sentry, el cual permite enviar reportes detallados a una cuenta particular del mismo. De esta forma, el usuario de mantenimiento puede tener un contexto del error ocurrido, y tomar una acción determinada para el caso.	Se solicita todas las preguntas y alias en QAWiki. Se buscan preguntas similares. Se busca main entity de la pregunta del usuario. Se busca en Wikidata entidades que contengan main entity. Se busca en Wikidata resultado usando pregunta similar. Se obtiene resultado con consulta en Wikidata. Se obtiene QID y se busca entidad en QAWiki Se busca en Wikidata resultado usando pregunta de contexto	Respuesta de servidor de QAWiki no exitosa. Respuesta de servidor de Wikidata no exitosa. No se obtuvo consulta SPARQL.	
Se retorna al usuario	El usuario consumidor de la aplicación debe tener un feedback en caso de que haya	Notificar vía Sentry	Respuesta de servidor	

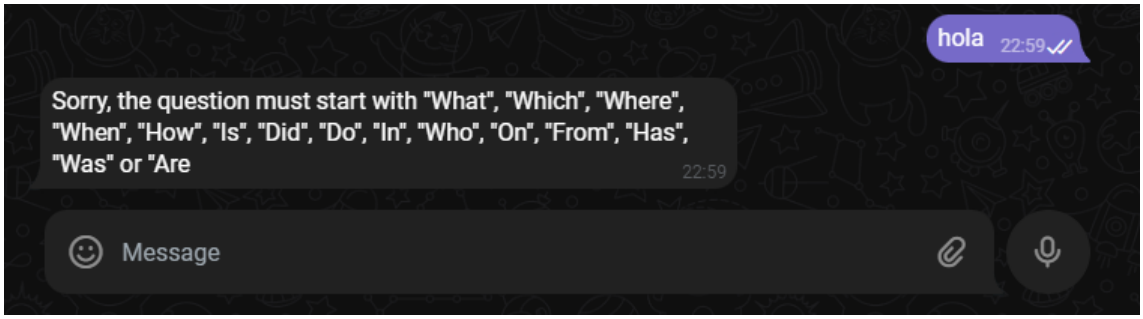
error no existe información	ocurrido un error, para ello se arma la respuesta y se notifica un error sin detalle particular, ya que para el usuario sin conocimiento técnicos no le es útil saber cuál fue el error exacto.		de QAWiki no exitosa. Respuesta de servidor de Wikidata no exitosa. No se obtuvo consulta SPARQL.	

2.9 Demostración del funcionamiento del sistema

Explicado los flujos que tiene el sistema y los casos de uso, a continuación se procede a mostrar ejemplos de respuestas del sistema.

Ejemplo 1: El texto enviado por el usuario no es una pregunta válida

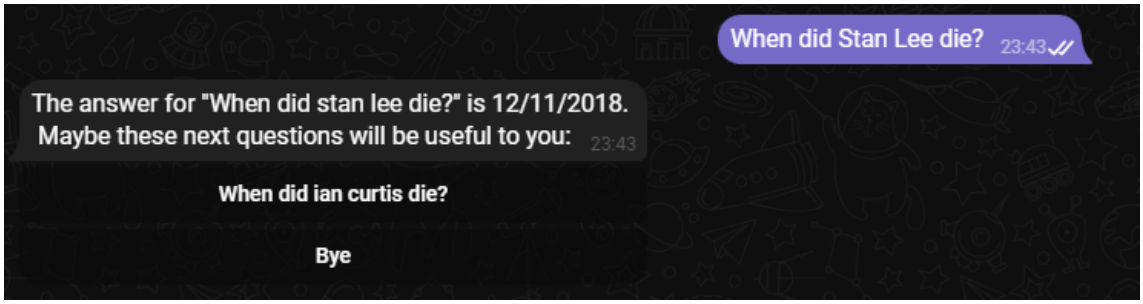
En este caso, como el usuario no envió lo que para el sistema se considera una pregunta válida y no existe una respuesta a una pregunta previa realizada con éxito, el sistema retorna un feedback notificando como el usuario debe iniciar el texto que envía para considerarse una pregunta.



Ejemplo 2: El texto enviado por el usuario es una pregunta existente en QAWiki

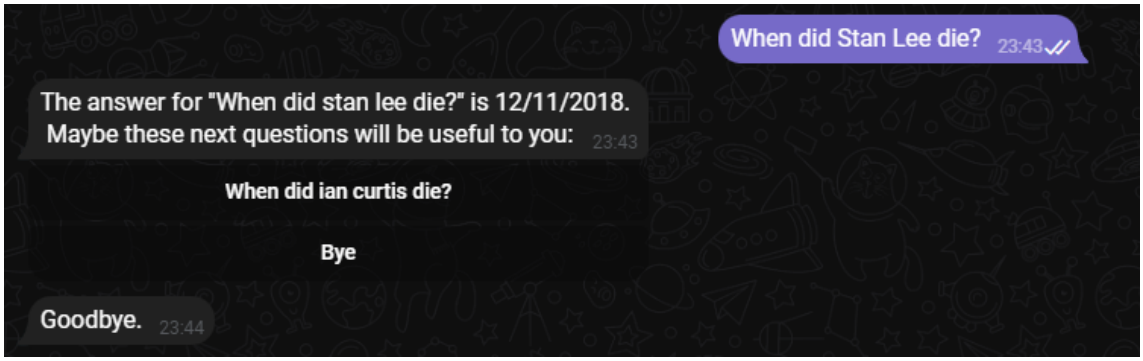
En este caso el usuario realiza una pregunta que se encuentra dentro del catálogo de preguntas que tiene el sistema de QAWiki, en este caso se busca y se obtiene la correspondiente consulta SPARQL, para luego obtener la información correcta desde Wikidata.

Como funcionalidad extra, se agregan preguntas relacionadas a la pregunta del usuario, obtenidas desde QAWiki. En este caso la única pregunta relacionada en QAWiki es “When did ian curtis die?”. Además, se agrega la opción de terminar con la conversación, para ello es necesario seleccionar la opción “Bye”.



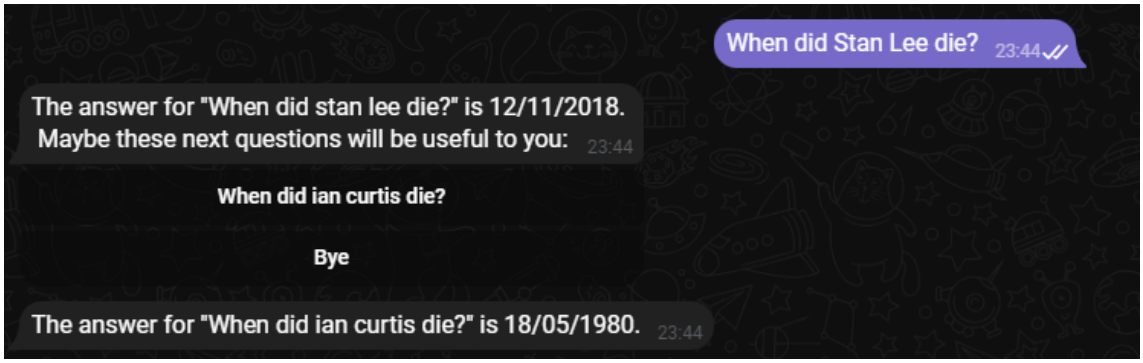
Ejemplo 3: El usuario termina la conversación:

Para este caso el usuario seleccionó la opción “Bye”, por lo que el sistema retorna un mensaje de despedida como feedback al usuario.



Ejemplo 4: El usuario elige pregunta relacionada

Para este caso el usuario seleccione una de las opciones que contienen una pregunta relacionada, por lo que el sistema retorna la respuesta correspondiente a dicha pregunta relacionada. En caso de que exista otra pregunta relacionada también va a presentar las debidas opciones en la respuesta. De esta manera, el flujo de conversación se realiza con mayor eficiencia.

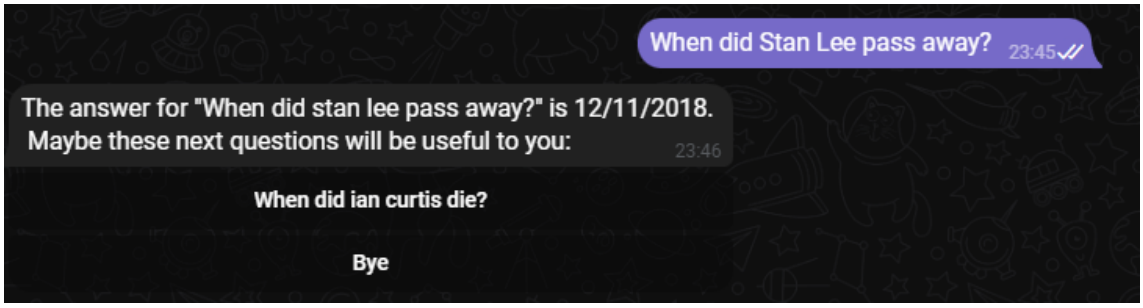


Ejemplo 5: El texto enviado por el usuario es un alias a una pregunta existente en QAWiki

Para este caso la pregunta no está cargada como tal en QAWiki, sino que está creada como un alias de una pregunta diferente, a un nivel más técnico, están

cargadas dentro un arreglo de otra pregunta. Esto se realizó así para evitar duplicar información, ya que los resultados serían los mismos al ser preguntas sinónimas.

Para este caso la respuesta va ser la misma que la pregunta a la que pertenece. Usamos un alias a la pregunta “When did Stan Lee die?”

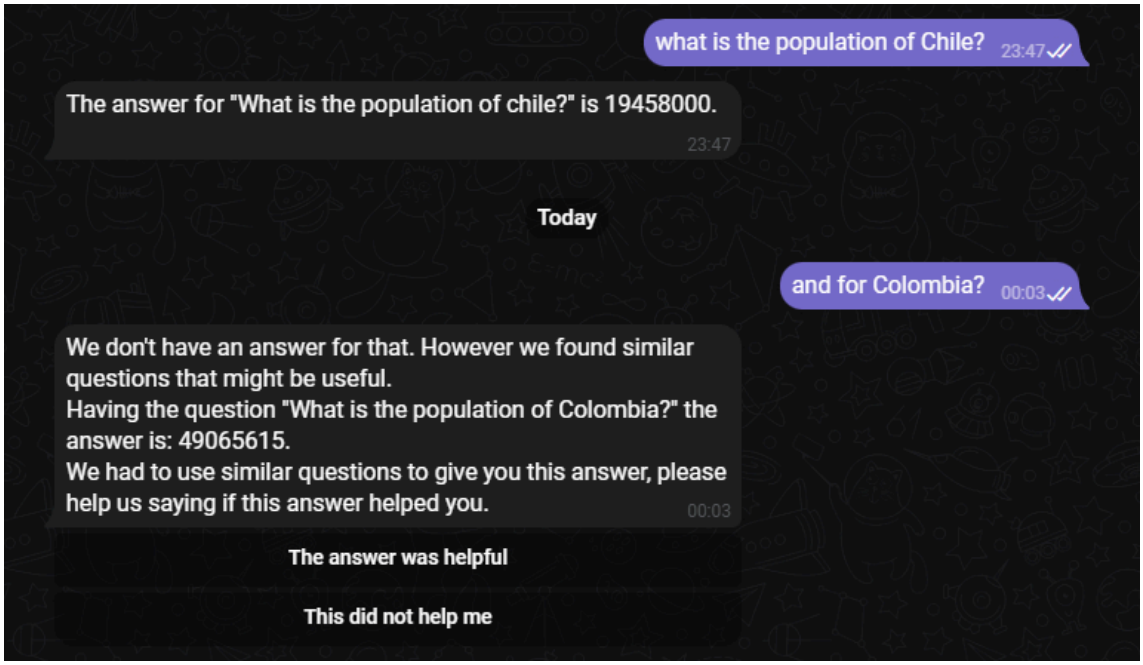


Ejemplo 6: El texto enviado por el usuario es una pregunta válida no existente en QAWiki y existe un contexto

Para este caso, el usuario ya realizó una pregunta previamente, y el sistema respondió exitosamente. Dada esta situación, el sistema guarda un contexto de la conversación, permitiendo al usuario realizar preguntas relacionadas a la pregunta anterior sin necesidad de realizar una sintaxis particular.

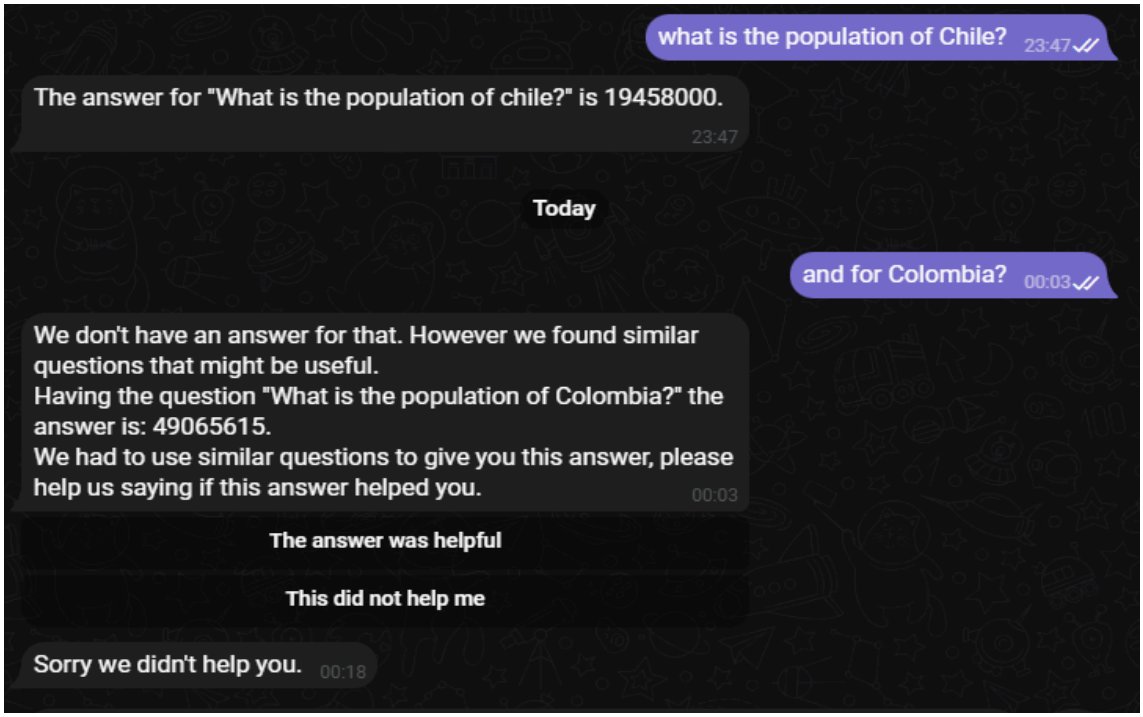
Como ejemplo se puede ver que el usuario preguntó “¿Cuál es la población de Chile?”, y luego preguntó “Y para Colombia?”. El sistema retorna una respuesta pero también notificando que esta pregunta puede ser de utilidad o no, debido a que se realizó un procesamiento interno para determinar la mejor respuesta.

Luego de que el sistema retorna la respuesta a la pregunta, el usuario puede elegir entre las opciones “The answer was helpful” o “This did not help me”.



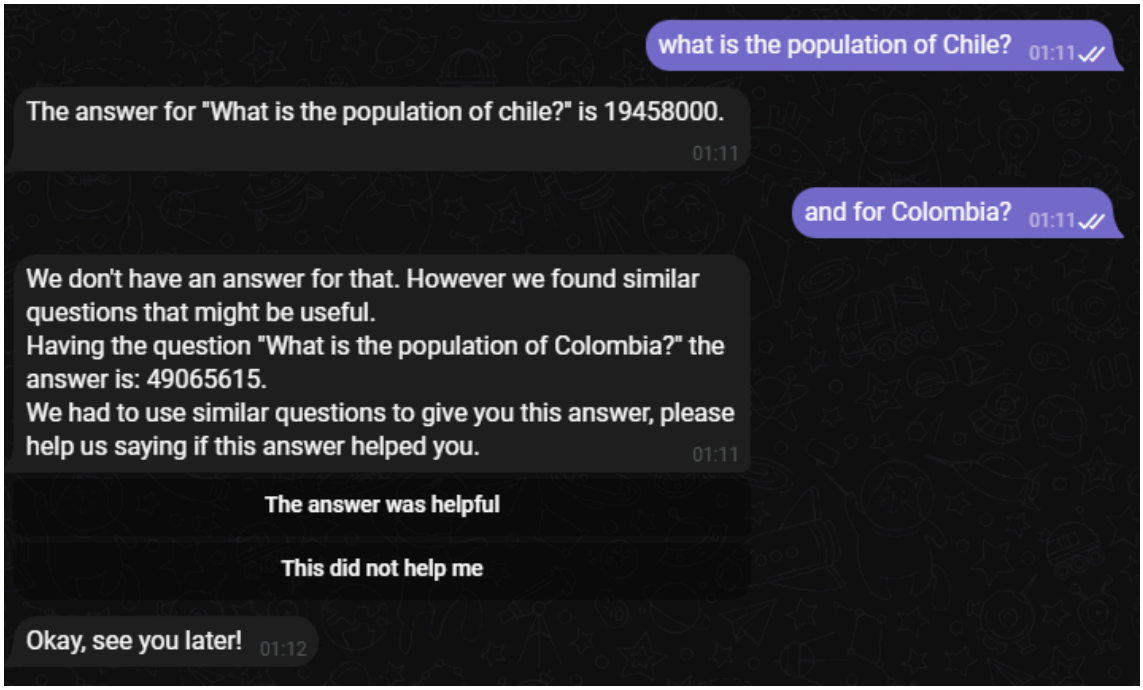
Ejemplo 7: El usuario elige que la respuesta no fue útil

Para este caso, el usuario eligió “This did not help me”, esta opción simplemente descarta la información que se obtuvo y se notifica al usuario el feedback correspondiente.



Ejemplo 8: El usuario elige que la respuesta fue útil

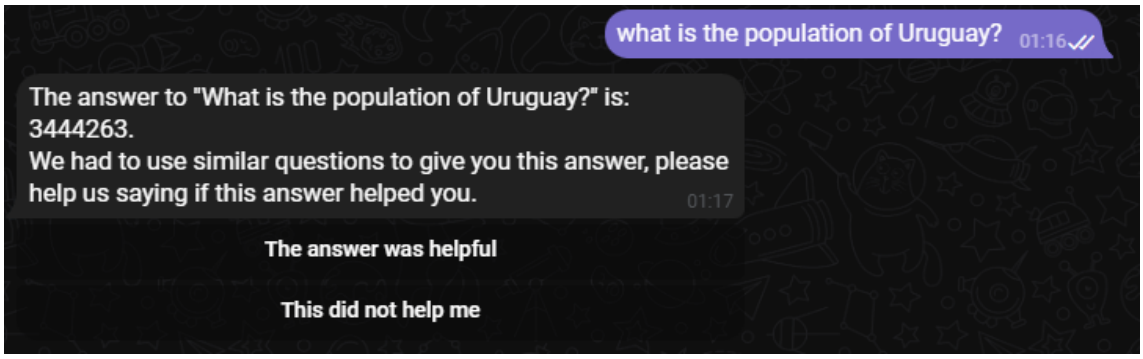
Para este caso, el usuario eligió “The answer was helpful”, esta opción permite al sistema cachear la respuesta, guardarla en el contexto, crear la plantilla o el alias, y notificar al usuario el feedback correspondiente.



Ejemplo 9: El usuario envía una pregunta que no está en QAWiki pero hay relacion con otra pregunta

En este caso en particular, el usuario realiza una pregunta que no se encuentra como tal en QAWiki, pero dicho sistema posee una pregunta que, sintácticamente, son iguales y se corresponde a otro sujeto. Para ello el proyecto también detecta estos casos y retorna la correcta respuesta a la pregunta del usuario, reemplazando el sujeto en la pregunta de QAWiki por el sujeto de la pregunta del usuario. De esta forma se puede agrandar el dominio de preguntas y responder preguntas que no están cargadas en QAWiki.

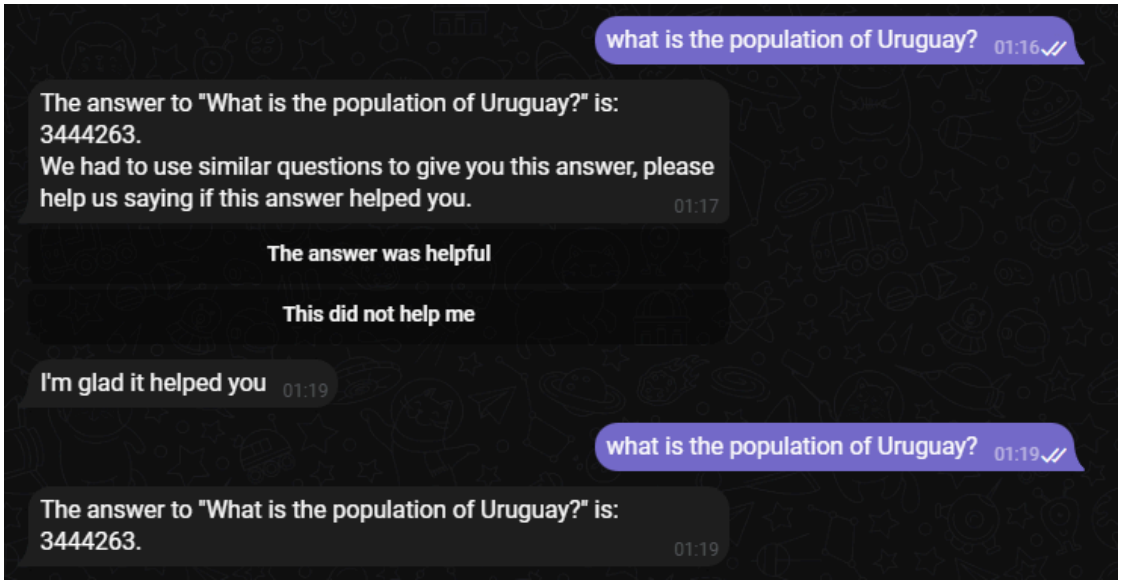
La idea es la misma que el anterior caso, salvo que para este caso poseemos información relevante en QAWiki para obtener una respuesta.



Ejemplo 10: La pregunta que realiza el usuario se encuentra cacheada

Cuando un usuario realiza la misma pregunta que previamente realizó, con un tiempo menor a una hora ya que la caché se limpia automáticamente, la información que se retorna proviene de la caché, lo que aumenta la eficiencia del sistema al otorgar un resultado más rápido y sin realizar ningún procesamiento extra.

Siguiendo el ejemplo anterior, se prosiguió a aceptar el resultado que el sistema otorgó, esto generó que se cacheara la pregunta con su respuesta, permitiendo consultar nuevamente y obtener el mismo resultado.



Ejemplo 11: El texto que envía el usuario es un comando

Este caso es especial, ya que el usuario no realizó una pregunta, sino más bien realizó un comando especial que el sistema permite. Por ejemplo, el usuario puede solicitar borrar la caché. El sistema otorga un feedback para notificar al usuario del resultado al ejecutar el comando que solicitó.

Capítulo 3: Conclusiones y posibles trabajos futuros

Se realizó una exhaustiva investigación sobre los progresos actuales en materia de herramientas y tecnologías disponibles para la recolección de datos en la web semántica, como también se estudió el proceso de diseño de interfaces conversacionales, en busca de generar una herramienta que permita dicha recolección, beneficiando a los usuarios que quieran obtener información de la web semántica utilizando el lenguaje natural, sin previos conocimientos técnicos. Se creó una herramienta que permite la obtención de datos en la web semántica haciendo uso del ecosistema Wikidata+QAWiki+Templet, que a su vez permitimos la expansión del dominio del mismo mediante técnicas de análisis semántico y sintáctico. Se espera que el flujo de ejecución de esta herramienta pueda ser útil para los usuarios, y también sea fácil de integrar, ya que solo es necesario obtener las credenciales del bot de telegram y del DNS de Sentry para hacer uso de los mismos. La herramienta desarrollada en este trabajo presenta una interfaz muy simple e intuitiva, haciendo uso

de la misma que proporciona Telegram, por lo que se espera que la curva de aprendizaje no sea alta en absoluto.

3.1 Posibles Trabajos Futuros

Una de las cuestiones a destacar sobre el enfoque tomado en este trabajo es la idea de permitir el desarrollo de esta herramienta utilizando las facilidades que otorga Telegram para la obtención de una interfaz agradable al usuario. Sin embargo, un camino alternativo podría ser el desarrollo de una aplicación web o una aplicación nativa de dispositivos móviles, para Android o iOS, que permita la creación de una interfaz personalizada. El foco de este trabajo fue la recolección de información en la web semántica, por lo que el diseño de una interfaz pasó a estar en un segundo plano. También ésta herramienta utiliza únicamente el lenguaje Inglés, pero podría tomarse un enfoque para la utilización del idioma español, ya que hoy en día el ecosistema de Wikidata+QAWiki+Templet también permite las búsquedas de preguntas en dicho idioma, lo cual implicaría cambios relacionados al análisis sintáctico y semántico de los textos que el usuario envía al sistema. Además, este trabajo está orientado a un contexto de preguntas existentes en el ecosistema mencionado, pero podría ser extendido para abarcar preguntas en lenguaje natural que puedan ser respondidas integrando otras herramientas más al mismo, teniendo en cuenta los costos económicos que esto implicaría si se utilizara herramientas pagas. Por último, otra cuestión a destacar fue el uso estricto de Wikidata para obtener la información a las preguntas que el usuario realiza en lenguaje natural, sin embargo, un camino extra podría ser la integración de alguna otra plataforma que también ofrezca las caracteristicas que Wikidata tiene para una correcta integración, como es el caso de DBpedia.

Por último, en el culmen de este arduo pero gratificante viaje académico, deseo expresar mi profundo agradecimiento a la Universidad Pública que ha sido la fuente de mi sendero educativo. Este recorrido no solo ha sido una exploración intelectual, sino también un compromiso con los valores fundamentales de la educación accesible y equitativa que encarna nuestra institución. Agradezco sinceramente a la Universidad por proporcionar un ambiente propicio para el aprendizaje, donde la excelencia académica y la diversidad de pensamiento se entrelazan para fomentar un crecimiento integral. Las experiencias compartidas con profesores excepcionales, compañeros inspiradores y programas académicos enriquecedores han dejado recuerdos imborrables en mi desarrollo personal y profesional. No puedo pasar por alto el apoyo incondicional de mi familia, cuyo sacrificio y dedicación han sido esenciales en mi camino educativo. Este logro no es solo mío, sino de todos aquellos que han contribuido a mi formación. Agradezco a mi familia por ser mi red de seguridad emocional, a mi pareja y mis amigos, en especial a mi amigo Santiago, por el estímulo continuo, y a la Universidad Pública por proporcionarme las herramientas necesarias para mi camino hacia el futuro. Que este agradecimiento sirva como un testimonio de mi profunda gratitud hacia aquellos que han sido pilares fundamentales en mi travesía académica.

Capítulo 4: Bibliografía

[1] M. Gomez, J. Cabot, and R. Clarisó, “Towards the Automatic Generation of Conversational Interfaces to Facilitate the Exploration of Tabular Data,” CoRR, vol. abs/2305.11326, 2023, doi: 10.48550/arXiv.2305.11326.

[2] Daniel Hernández, Aidan Hogan, and Markus Krötzsch. Reifying RDF: what works well with wikidata? In Thorsten Liebig and Achille Fokoue, editors, Proceedings of the 11th International Workshop on Scalable Semantic Web Knowledge Base Systems co-located with 14th International Semantic Web Conference (ISWC 2015), Bethlehem, PA, USA, October 11, 2015, volume 1457 of CEUR Workshop Proceedings, page 32. CEUR-WS.org, 2015. https://iccl.inf.tu-dresden.de/web/Inproceedings3037/en

[3] L. Piro, G. Desolda, M. Matera, R. Lanzilotti, S. Mosca, and E. Pucci, “An Interactive Paradigm for the End-User Development of Chatbots for Data Exploration,” in Human-Computer Interaction – INTERACT 2021, C. Ardito, R. Lanzilotti, A. Malizia, H. Petrie, A. Piccinno, G. Desolda, and K. Inkpen, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 177–186. doi: 10.1007/978-3-030-85610-6_11.

[4] Francisca Suárez and Aidan Hogan. 2023. Templet: A Collaborative System for Knowledge Graph Question Answering over Wikidata. In Companion Proceedings of the ACM Web Conference 2023 (WWW ’23 Companion), April 30-May 4, 2023, Austin, TX, USA. ACM, New York, NY, USA, 4 pages. https: //doi.org/10.1145/3543873.3587335

[5] A. Vegesna, P. Jain, and D. Porwal, “Ontology based Chatbot (For E-commerce Website),” IJCA, vol. 179, no. 14, pp. 51–55, Jan. 2018, doi: 10.5120/ijca2018916215.

[6] The Semantic Web – ISWC 2019: 18th International Semantic Web Conference, Auckland, New Zealand, October 26–30, 2019, Proceedings, Part IIOct 2019 Pages 69–78 https://doi.org/10.1007/978-3-030-30796-7_5

[7] Suárez Soto, F. 2023 “Autocompletando preguntas sobre Wikipedia”. https://repositorio.uchile.cl/handle/2250/19305

[8] Ait-Mlouk and L. Jiang, “KBot: A Knowledge Graph Based ChatBot for Natural Language Understanding Over Linked Data,” IEEE Access, vol. 8, pp. 149220–149230, 2020, doi: 10.1109/ACCESS.2020.3016142.

[9] Kerry, A., Ellis, R., Bull, S. (2009). Conversational Agents in E-Learning. In: Allen, T., Ellis, R., Petridis, M. (eds) Applications and Innovations in Intelligent Systems XVI. SGAI 2008. Springer, London. https://doi.org/10.1007/978-1-84882-215-3_13

[10] Poivet R, Lopez Malet M, Pelachaud C and Auvray M (2023) The influence of conversational agents’ role and communication style on user experience. Front. Psychol. 14:1266186. doi: https://doi.org/10.3389/fpsyg.2023.1266186

[11] McTear MF, Callejas Z, Griol D (2016) Speech Input and Output. In: McTear MF, Callejas Z, Griol D (eds) The Conversational Interface Talking to Smart Devices. Springer, pp 75–92. https://dl.acm.org/doi/10.5555/2965050

[12] Jia Wang, “A survey of web caching schemes for the Internet” https://dl.acm.org/doi/10.1145/505696.505701

[13] V. Martina, M. Garetto and E. Leonardi, "A unified approach to the performance analysis of caching systems," IEEE INFOCOM 2014 - IEEE Conference on Computer

Communications, Toronto, ON, Canada, 2014, pp. 2040-2048, doi: 10.1109/INFOCOM.2014.6848145.

[14] Zulfa, M.I., Hartanto, R. and Permanasari, A.E. (2020), "Caching strategy for Web application – a systematic literature review", International Journal of Web Information Systems, Vol. 16 No. 5, pp. 545-569. https://doi.org/10.1108/IJWIS-06-2020-0032

Además de las referencias incluidas en el apartado previo, se toma como literatura de base para este trabajo la que se detalla a continuación.

McBride, B. (2004). The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. In: Staab, S., Studer, R. (eds) Handbook on Ontologies. International Handbooks on Information Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24750-0_3

SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information RetrievalJune 2018 Pages 1371–1374 https://doi.org/10.1145/3209978.3210183

Department of Computer Science, Universidad de Chile, Santiago de Chile, Chile Aidan Hogan, (2020). The web of Data (Book). https://aidanhogan.com/webofdatabook/

DIS '17: Proceedings of the 2017 Conference on Designing Interactive SystemsJune 2017 Pages 555–565 https://doi.org/10.1145/3064663.3064672

Capítulo 5: Anexos

Anexo 1: Repositorios de código creados

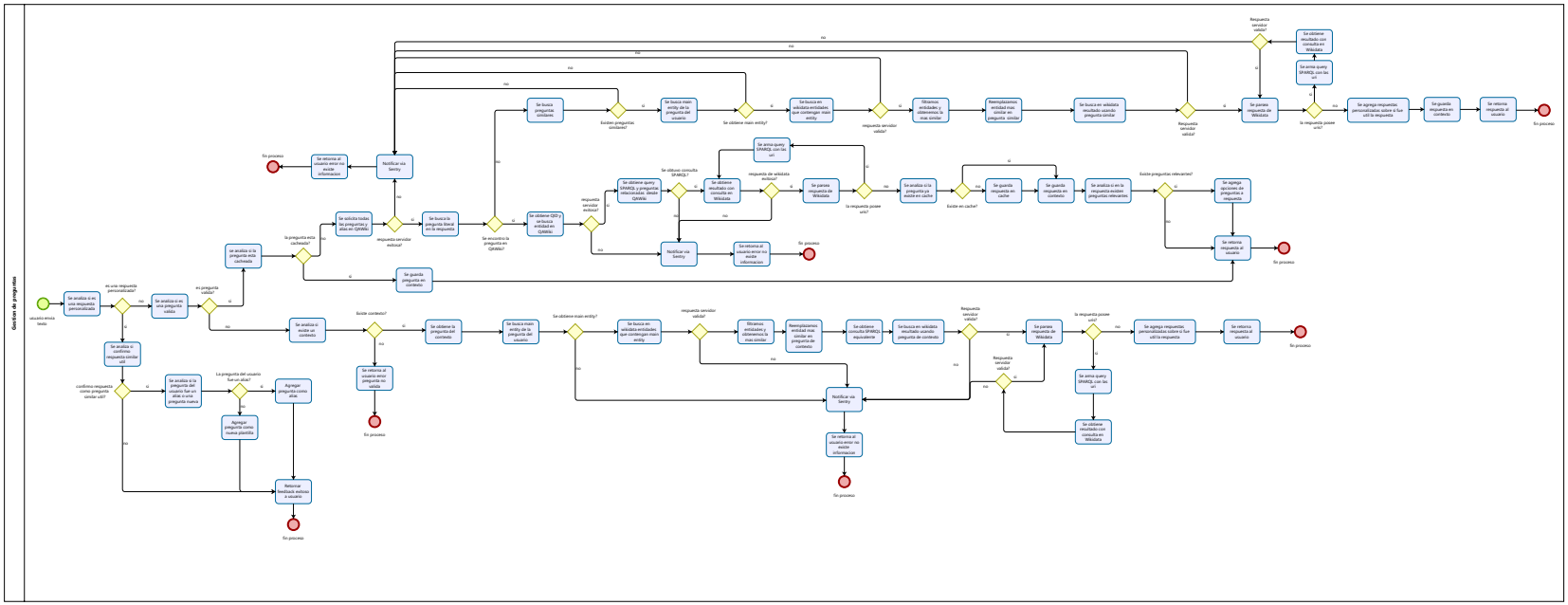
Para realizar la programación de la herramienta presentada en este trabajo se utilizó la herramienta de versionado de código Git y se utilizó Github como plataforma para albergar el código fuente. A continuación se presenta el enlace al repositorio.

- Aplicación backend: chatbot telegram

Anexo 2: Diagrama de flujo

A continuación se presenta el diagrama en detalle:

1 CHATBOT


Gestion de preguntas
usuario envia texto
Se analiza si es una respuesta personalizada
es una respuesta personalizada?
Se analiza si es una pregunta valida
es pregunta valida?
Se analiza si confirma respuesta similar uril
confirmo respuesta como pregunta similar uril?
Se analiza si la pregunta del usuario fue un alias
La pregunta del usuario fue un alias?
Se analiza si la pregunta esta cacheada
la pregunta esta cacheada?
Se solicita todas las preguntas y alias en QdrANS
Se busca la pregunta literal en la respuesta
requuesta servidor exitosa?
Se encontro la pregunta en QdrANS?
Se obtiene QID y se busca entidad en QdrANS
Se obtiene query SPARQL y pregunta relacionadas desde QdrANS
Se obtuvo consulta SPARQL?
Se arma query SPARQL con las uril
Se obtiene resultado con consulta en Wikidata
requuesta de wikidata exitosa?
Se parsea respuesta de Wikidata
la respuesta posee uris?
Se analiza si la pregunta ya existe en cache
Existe en cache?
Se guarda respuesta en cache
Se guarda respuesta en contexto
Se analiza si en la respuesta existen preguntas relevantes
Existe preguntas relevantes?
Se agrega opciones de preguntas a responder
Se retorna respuesta al usuario
fin proceso
Se guarda pregunta en contexto
Se retorna al usuario error no existe informacion
Notificar via Sentry
Se retorna al usuario error no valido
Existe contexto?
Se obtiene la pregunta del contexto
Se busca pregunta similares
Existen preguntas similares?
Se busca main entity de la pregunta del usuario
Se obtiene main entity?
Se busca en wikidata entidades que contengan main entity
requuesta servidor valida?
Obtenemos entidades y, o mas similar
Reemplazamos entidad mas similar en pregunta de contexto
Se busca en wikidata resultado usando pregunta similar
Se busca en wikidata resultado usando pregunta de contexto
Se obtiene query SPARQL equivalente
Agregar pregunta como alias
Agregar pregunta como nueva plantilla
Retornar respuesta exitosa al usuario
Se retorna al usuario error no valido
Se retorna al usuario error no existe informacion
Se arma query SPARQL con las uril
Se obtiene resultado con consulta en Wikidata
Se agrega respuestas personalizadas sobre si fue util la respuesta
Se guarda respuesta en contexto
Se retorna respuesta al usuario
Modeler