UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

AUTOCOMPLETANDO PREGUNTAS SOBRE WIKIDATA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

FRANCISCA PAZ SUÁREZ SOTO

PROFESOR GUÍA:
AIDAN HOGAN

MIEMBROS DE LA COMISIÓN:
SANDRA DE LA FUENTE GONZÁLEZ
FELIPE BRAVO MÁRQUEZ

SANTIAGO DE CHILE
2023

Resumen

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 ordenadores. Esto permite que sean conectados y consultados datos de diferentes fuentes.

El problema de estos datos es que para acceder a ellos, se necesitan lenguajes especializados, como SPARQL. Generalmente, el usuario común no está familiarizado con este lenguaje, o 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 neuronal y traducen directamente el lenguaje natural a SPARQL. El problema de estas 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.

El sistema desarrollado en este trabajo, Templet, consiste en una plataforma web basada en plantillas y autocompletado, permitiendo consultar preguntas en lenguaje natural a partir de otras preexistentes, pudiendo hacer uso de todas las entidades disponibles en Wikidata (una de las bases de conocimiento de datos enlazados estructurados más importante). Las plantillas son obtenidas a partir de la colección de preguntas disponible en QAWiki, un repositorio editado en colaboración de preguntas en lenguaje natural y sus respectivas consultas en SPARQL, que actualmente almacena más de 370 preguntas.

El propósito de Templet es poder generar distintos pares de preguntas y consultas a partir de una plantilla. Además, tiene como finalidad permitir que personas no especializadas puedan navegar sobre los datos enlazados. Asimismo, tiene como propósito generar interés en QAWiki. De esta forma, se puede expandir su uso y contribución, logrando una colección más amplia y diversa de pares de preguntas y consultas, permitiendo el avance de los modelos del estado del arte que necesitan grandes datasets de preguntas y consultas.

El sistema se implantó en el entorno de producción y fue probado por múltiples usuarios reales, obteniendo observaciones positivas. Logró cumplir con los objetivos establecidos y el comportamiento esperado, aunque existe trabajo futuro que puede mejorar la usabilidad del sistema.

A mi madre, quien siempre estuvo allí.

Tabla de Contenido

1. Introducción **1**

1.1. Problema 3

1.2. Objetivos 4

1.3. Solución 4

1.4. Estructura 5

2. Estado del arte **6**

2.1. Wikidata 6

2.2. Lenguaje natural a SPARQL 9

2.2.1. Enfoques no neuronales 9

2.2.2. Enfoques neuronales 10

2.3. QAWiki 12

2.4. Contexto del presente trabajo 13

3. Diseño de la solución **15**

3.1. Requisitos del sistema 15

3.1.1. Requisitos funcionales 16

3.1.2. Requisitos no funcionales 16

3.2. Elección de las tecnologías de desarrollo 17

3.3. Arquitectura de la solución 18

3.4. Flujo de la aplicación 18

4. Implementación de la solución **21**

4.1. Backend 21

4.1.1. Generación de plantillas 21

4.1.2. Backend de plataforma web 25

4.2. Frontend 26

4.2.1. Interfaz 26

4.2.2. Sistema multilingüe 32

4.2.3. Estructura del código 34

5. Evaluación del sistema **35**

5.1. Rendimiento 35

5.2. Usabilidad 36

6. Conclusión **39**

6.1. Resumen 39

6.2. Objetivos 39

6.3. Fortalezas y limitaciones 41

6.4. Trabajo futuro 41

6.5. Conclusiones finales 42

Bibliografía **45**

Anexo **46**

Índice de Ilustraciones

2.1. Términos importantes utilizados en un elemento de Wikidata. Fuente: Artículo sobre Wikidata en Wikipedia. 7

2.2. Interfaz de Constructor de Consultas de Wikidata: consulta de elementos que tienen una ocupación pero no son humanos. 8

2.3. Servicio de Consultas de Wikidata: ejemplos de consultas según búsqueda *musician*. 9

2.4. ElNeuQA: Descripción general del enfoque del sistema para una pregunta de entrada de ejemplo [3]. 12

2.5. QAWiki: vista de pregunta *¿Quién fue el autor de One-Punch Man?* 13

2.6. QAWiki: extracto de algunas entidades y menciones de *¿Quién fue el autor de One-Punch Man?* 14

3.1. Mock-up inicial del sistema 15

3.2. Templet: arquitectura de la solución 19

3.3. Flujo de Templet 20

4.1. Templet: entrada de texto principal 26

4.2. Templet: plantilla de pregunta `Q137` de QAWiki 27

4.3. Templet: autocompletado en entradas de texto de plantilla `Q137` 28

4.4. Templet: selección de entidad en plantilla `Q137` 28

4.5. Templet: error en campo en plantilla `Q137` 29

4.6. Templet: carga de resultados en consulta generada a partir de plantilla `Q137` 29

4.7. Templet: despliegue de resultados en consulta generada a partir de plantilla `Q137` 30

4.8. Templet: despliegue de resultados crudos a partir de plantilla `Q65` 30

4.9. Templet: despliegue de resultados tipo tabla en consulta generada a partir de plantilla `Q297` 31

4.10. Templet: despliegue de resultados de pregunta contingente cuando no se tiene el resultado esperado, a partir de plantilla `Q65` 31

4.11. Templet: actualización exitosa de plantilla. 32

4.12. Templet: despliegue de resultados de pregunta a partir de plantilla `Q11` en un dispositivo móvil. 33

5.1. Resultados normalizados de SUS. 38

A. Estructura general del proyecto 47

Capítulo 1

Introducción

La cantidad de datos estructurados publicados en la web es cada vez mayor y continúa en crecimiento. Los datos enlazados corresponden a un método de publicación de datos estructurados que permiten ser interconectados y consultados desde distintas fuentes, de forma de tener mayor utilidad en consultas semánticas [15].

El interés sobre estos datos también se encuentra en aumento. Poderosos motores de búsqueda, tales como Google, emplean este tipo de datos con el fin de enriquecer sus resultados. Algunos usos incluyen *chatbots*, extracción de información, agentes personales, recomendaciones, búsqueda semántica, entre otros [3].

Para acceder a estos datos se utilizan lenguajes especializados, como SPARQL[1]. Generalmente, el usuario común no está familiarizado con este lenguaje, los lenguajes de la Web Semántica, la estructura de la nube de datos enlazados o los conjuntos de datos disponibles, produciéndose una brecha que impide la comunicación entre el usuario y estos datos. Debido a esto, se hacen necesarios paradigmas para permitir la interacción entre los datos y los usuarios de la web. Los sistemas de respuesta a preguntas en lenguaje natural tienen como objetivo eliminar esta limitación. Algunos de estos sistemas transforman expresiones en lenguaje natural a consultas estructuradas en lenguajes especializados [15].

En la actualidad existen diversas soluciones al problema de QA (*question answering* o respuesta a preguntas) en los datos enlazados. Las soluciones más recientes aplican un enfoque neuronal y aprovechan los avances en la traducción automática neuronal (NMT, por sus siglas en inglés), traduciendo el lenguaje natural a SPARQL. El problema de estas soluciones es el requisito de grandes datasets para entrenar los modelos, necesitando cerca del orden de millones de instancias. Actualmente, no se tiene una colección de datos de tal volumen, dificultando resolver por completo el problema del vocabulario desconocido, especialmente en preguntas complejas. En el trabajo de Diomedi y Hogan [3], se identifica que las dificultades de vocabulario se concentran en las entidades, por lo que se propone un sistema de vinculación de entidades (*entity linking* o EL) para extraer las entidades de la pregunta y ejecutar un relleno de ranuras (*slot filling* o SF) sobre el template de la consulta. Así, se reducen considerablemente los errores debido a entidades en comparación a las soluciones

[1] http://www.w3.org/TR/rdf-sparql-query/

que emplean solo NMT. Es importante notar que incluso con estas técnicas, consideradas dentro del estado del arte, los resultados no son excepcionales, alcanzando una precisión y recuperación de un 27 % con la mejor configuración.

La solución propuesta en este trabajo posee una arquitectura más simple basada en plantillas y autocompletado, permitiendo mayor control sobre el espacio acotado de preguntas a consultar. Además, el autocompletado de preguntas presenta una interfaz interactiva con el usuario, capaz de anteponerse a la consulta a generar. Los principales recursos a utilizar para desarrollar esta solución corresponden a Wikidata[16] y QAWiki[2].

Wikidata[3] es una base de conocimientos gratuita y abierta, fundada por Wikimedia, con el fin de seleccionar y representar la información de Wikipedia en todos los idiomas, en un formato interoperable y legible por máquina [16]. Actúa como almacenamiento central para los datos estructurados de sus proyectos hermanos, como lo es Wikipedia[4], Wikivoyage[5], Wiktionary[6], entre otros. Wikidata ofrece desde el 2015 un servicio de consultas[7], que permite a los usuarios realizar consultas sobre los millones de datos contenidos en el sitio. Este servicio utiliza SPARQL como lenguaje de consulta. Hoy en día, la plataforma posee alrededor de 97 millones de elementos y se han realizado más de 1600 millones de ediciones desde el lanzamiento del proyecto[8].

QAWiki es un repositorio editado en colaboración de preguntas en lenguaje natural y sus respectivas consultas en SPARQL, con el fin de respaldar las respuestas de preguntas sobre grafos de conocimiento como Wikidata. Actualmente almacena alrededor de 370 preguntas (disponibles en inglés y español), además de una descomposición detallada de las entidades contenidas en las preguntas. Por ejemplo, la pregunta *¿Quién interpretó el carácter Eleven en Stranger Things?*[9] (y su equivalente en inglés, *Who played Eleven in Stranger Things?*) tiene como *mención de entidad* los ítems `Q27955792`, correspondiente a *Eleven*, y `Q19798734`, equivalente a *Stranger Things*. QAWiki también posee un servicio de búsqueda[10], desde donde se pueden realizar consultas sobre el repositorio.

El sistema a desarrollar, llamado Templet (una forma alternativa de escribir *template*, plantilla en español) corresponde a una plataforma web de preguntas y respuestas, con una entrada principal de texto en lenguaje natural por parte del usuario, la cual es autocompletada para guiar y predeterminar la estructura de la consulta. Este autocompletado se alimenta de la colección de preguntas disponibles en QAWiki, previamente transformadas en plantillas (por ejemplo, la plantilla en español de la pregunta mencionada anteriormente es *¿Quién interpretó el carácter (...) en (...)?*). Al cliquear en una de estas plantillas, se despliega un nuevo formulario con un campo de entrada de texto para cada entidad de la plantilla. Estas entradas de texto consultan directamente a la API de Wikidata y también poseen autocompletado, permitiendo la selección de nuevas entidades a reemplazar en la pregunta.

[2]http://qawiki.dcc.uchile.cl/
[3]https://www.wikidata.org/
[4]https://es.wikipedia.org/
[5]https://www.wikivoyage.org/
[6]https://www.wiktionary.org/
[7]https://query.wikidata.org/
[8]How big is Wikidata?: https://www.wikidata.org/wiki/Wikidata:Statistics
[9]http://qawiki.dcc.uchile.cl/wiki/Item:Q11
[10]http://query.qawiki.dcc.uchile.cl/proxy/wdqs/bigdata/namespace/wdq/sparql

Finalmente, se mapean las entidades seleccionadas a la consulta SPARQL y se ejecuta en el Servicio de Consultas de Wikidata[11], obteniendo el resultado.

Templet permite que se generen nuevas instancias de preguntas y consultas a partir de un par inicial, del cual se crea una plantilla. De esta forma, se pueden extender conjuntos para el entrenamiento de redes neuronales. También hace que los datos enlazados sean más accesibles a usuarios no especializados, pudiendo consultar a partir de preguntas en lenguaje natural. Por último, la plataforma propuesta ofrece una demostración interactiva de QAWiki a los usuarios. De esta forma, se puede expandir su uso y contribución, logrando una colección más amplia y diversa de pares de preguntas y consultas, pudiendo ser utilizada, en un futuro, para el entrenamiento de redes neuronales profundas.

1.1. Problema

Existen múltiples soluciones al problema de QA sobre los datos enlazados. Debido a la complejidad de la tarea, es un problema difícil de resolver en su totalidad, debido al amplio espectro de preguntas que se pueden generar en lenguaje natural.

Las soluciones más recientes aplican un enfoque neuronal y aprovechan los avances en NMT, traduciendo el lenguaje natural a SPARQL. Empleando estas técnicas, consideradas dentro del estado del arte, los resultados alcanzan una precisión y recuperación de un 27 % con la mejor configuración [3]. Esto no necesariamente se debe a las arquitecturas neuronales, sino que la causa radica en que los modelos necesitan una gran cantidad de pares de preguntas en lenguaje natural con su respectiva consulta en SPARQL para lograr un entrenamiento acabado, y generar estas colecciones resulta muy costoso.

La motivación de QAWiki es mantener una plataforma abierta y colaborativa en donde los usuarios puedan contribuir y mejorar preguntas en lenguaje natural y consultas en SPARQL. De este modo, sistemas de QA basados en QAWiki pueden tomar estos ejemplares y mejorar su rendimiento.

El sistema propuesto ofrece una demostración de QAWiki a los usuarios de la web, siendo capaz de generar nuevas consultas a partir de las plantillas extraídas desde la Wiki. Además, facilita la ejecución de consultas sobre los datos enlazados, sin la necesidad de hacer uso de lenguajes especializados como SPARQL, haciendo más accesible la navegación sobre estos datos. También la creación de plantillas y reemplazo de entidades permite extender conjuntos de pares de preguntas y consultas para el entrenamiento de redes neuronales.

Se espera que con esta plataforma web se promueva el uso de QAWiki. De esta forma, los usuarios pueden contribuir con nuevas preguntas, enriqueciendo este repositorio y permitiendo mejorar el entrenamiento de redes neuronales profundas en QA.

[11] Servicio de consultas SPARQL de Wikidata

1.2. Objetivos

Objetivo General

El objetivo general corresponde a que un usuario común logre generar consultas SPARQL en Wikidata a través de preguntas en lenguaje natural, predefinidas a través de plantillas o *templates* que indican la estructura de una pregunta. Además, el sistema debe sugerir plantillas o entidades mediante autocompletado.

Objetivos Específicos

1. Generalizar plantillas a partir del repositorio de preguntas.
2. Sugerir plantillas según autocompletado.
3. Sugerir entidades para completar las plantillas, basándose en el autocompletado.
4. Considerar casos de preguntas dependientes y de desambiguación: algunas consultas pueden requerir de la respuesta de otra pregunta para ser respondidas.
5. Implementar un *front-end* capaz de funcionar con autocompletado de forma eficiente y amigable con el usuario.
6. Conectar el sistema desarrollado a esta interfaz y a los servicios proveídos por Wikidata y QAWiki.

1.3. Solución

La solución propuesta consiste en dos partes principales, siendo la primera la relacionada a la generación de plantillas, y la segunda, el sistema como tal.

La generación de plantillas es la primera tarea a desarrollar. En esta etapa, QAWiki provee las preguntas y consultas, además de las menciones y entidades contenidas en éstas. Luego, estos datos son procesados para obtener las plantillas, extrayendo las entidades tipo `wd:Q{n}` de la consulta en SPARQL y mapeando en la pregunta en lenguaje natural las menciones disponibles. Finalmente, se almacena el resultado en caché en un archivo JSON.

Esta tarea debe ser ejecutada periódicamente para actualizar posibles cambios desde QAWiki, por lo que se configura una tarea programada en el *back-end* mediante APScheduler [12], un programador de tareas y un sistema de cola de tareas para Python.

En cuanto a la arquitectura del sistema, se plantea un *back-end* en el framework Flask y un *front-end* simple en Bootstrap. Además, se hace necesario del uso de Javascript y jQuery para las llamadas asíncronas al *back-end* y las distintas funcionalidades del autocompletado.

[12] APScheduler GitHub

El *back-end* es el encargado de generar y enviar las plantillas almacenadas a *front-end*, además de gestionar el intercambio de información con las APIs y servicios de consulta de QAWiki y Wikidata.

Respecto al *front-end*, en primer lugar despliega una entrada principal de texto con autocompletado, que tiene como fuente las plantillas almacenadas. Cuando el usuario selecciona una de las preguntas sugeridas, carga un nuevo formulario con entradas de texto para las entidades a completar en la plantilla. Estas entradas también poseen autocompletado, pero a diferencia de la anterior, generan peticiones directamente a la API de Wikidata, permitiendo la sugerencia de nuevas entidades. Al completar de forma correcta este formulario, se activa el botón que envía la consulta al servicio de consultas de Wikidata y retorna los resultados.

1.4. Estructura

La presente memoria se encuentra estructurada de la siguiente forma:

- Capítulo 2: se presentan las principales herramientas disponibles dentro del contexto del trabajo. Además, se describen las soluciones del estado del arte para la transformación de lenguaje natural a SPARQL.
- Capítulo 3: se describe el diseño de la solución propuesta, indicando los requisitos funcionales y no funcionales, la elección de las tecnologías de desarrollo, la arquitectura de la solución y el flujo de la aplicación.
- Capítulo 4: se describe la implementación del sistema según el diseño expuesto en el capítulo 3. Se explica en detalle el backend (el cual incluye la generación de plantillas), el frontend y la estructura del código.
- Capítulo 5: se evalúa el sistema a través de una encuesta y mediciones a los tiempos para la generación de plantillas y se muestran los resultados.
- Capítulo 6: contiene un resumen en retrospectiva del trabajo realizado, se discuten los objetivos logrados, las limitaciones del sistema y el trabajo futuro.

Capítulo 2

Estado del arte

En este capítulo se describe a Wikidata, uno de los recursos de mayor importancia dentro de este trabajo, y las herramientas que tiene disponibles para la navegación dentro de los datos enlazados. También se presenta brevemente el trabajo dentro del estado del arte en la traducción de lenguaje natural a SPARQL, separados por enfoque no neuronal y neuronal. Luego, se explica el propósito de QAWiki, otro de los recursos importantes para el desarrollo de este trabajo. Por último, se describe la relación del presente trabajo con los elementos presentados anteriormente.

2.1. Wikidata

Wikidata es una base de conocimiento editada en colaboración, creada en el 2012 por la Fundación Wikimedia [16]. Su propósito inicial era resolver algunos problemas concretos de Wikipedia, siendo el principal la duplicación de elementos multilingües, pues existía redundancia de artículos que referenciaban 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.

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 y una o más declaraciones. Estas declaraciones consisten de pares de *llave (propiedad)-valor*[1]. Por ejemplo, en la figura 2.1, para el elemento *Douglas Adams*, se tienen en las declaraciones los pares *alma mater-Saint John's College* y *alma mater-Brentwood School*, pues existen dos valores para la misma propiedad.

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

[1] https://en.wikipedia.org/wiki/Wikidata#Conceptos

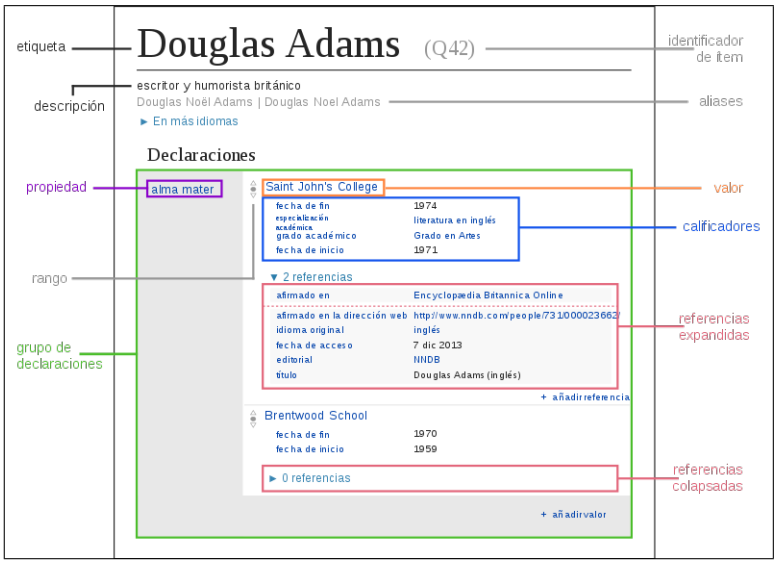

Figura 2.1: Términos importantes utilizados en un elemento de Wikidata. Fuente: Artículo sobre Wikidata en Wikipedia.

el lenguaje de consulta SPARQL[2]. Las consultas en este lenguaje se escriben contra los pares *llave-valor* descritos anteriormente, por lo que la base de datos del sistema es un conjunto de tripletas *sujeto-predicado-objeto*. Por ejemplo, de la figura 2.1, una tripleta sería *Douglas Adams-alma mater-Saint John's College*. Las consultas pueden ser simples siguiendo la estructura de una tripleta única, como la correspondiente a la pregunta *¿De qué religión es Joe Biden?*[3], que es `SELECT ?obj WHERE { wd:Q6279 wdt:P140 ?obj }`, en donde `wd:Q6279` identifica a *Joe Biden* y `wdt:P140` a *religión*. También pueden tener una mayor complejidad, por ejemplo, haciendo uso de agregaciones en un lapso de tiempo determinado, como en la pregunta *¿Cuál película de los últimos diez años ha ganado el mayor número de Premios Óscar?*[4], que se traduce en SPARQL a la consulta del código 1.

```
SELECT ?sbj WHERE {
  { SELECT ?sbj (MIN(?obj) AS ?obj1)
           WHERE {
             ?sbj wdt:P31/wdt:P279* wd:Q11424 .
             ?sbj wdt:P577 ?obj .
           }
   GROUP BY ?sbj }
  ?sbj wdt:P166 ?obj2 .
  ?obj2 wdt:P31/wdt:P279* wd:Q19020 .
  FILTER(YEAR(NOW())-YEAR(?obj1)-
         IF(MONTH(NOW())>MONTH(?obj1),
            0,IF(MONTH(NOW())=MONTH(?obj1)
                 && DAY(NOW())>=DAY(?obj1),0,1))< 10)
}
GROUP BY (?sbj)
```

[2]https://en.wikipedia.org/wiki/Wikidata#Query_service_and_other_improvements
[3]http://qawiki.dcc.uchile.cl/wiki/Item:Q325
[4]http://qawiki.dcc.uchile.cl/wiki/Item:Q187

```
16 ORDER BY DESC(COUNT(DISTINCT ?obj2))
17 LIMIT 1
```

Código 1: Consulta SPARQL de pregunta *¿Cuál película de los últimos diez años ha ganado el mayor número de Premios Óscar?*

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. Sin embargo, escribir una consulta en SPARQL no es trivial, generando una brecha entre el usuario común y los datos enlazados. Esta situación es reconocida por Wikidata, por lo que ha dispuesto de diversas herramientas que tienen como objetivo facilitar la navegación a los usuarios de la Web sobre los datos enlazados. Una de ellas es el Constructor de Consultas de Wikidata[5](Figura 2.2), un sistema multilingüe que aún se encuentra en desarrollo activo y que permite hacer consultas simples a través de una interfaz, requiriendo solo conocimientos básicos de cómo Wikidata modela los elementos que consulta.

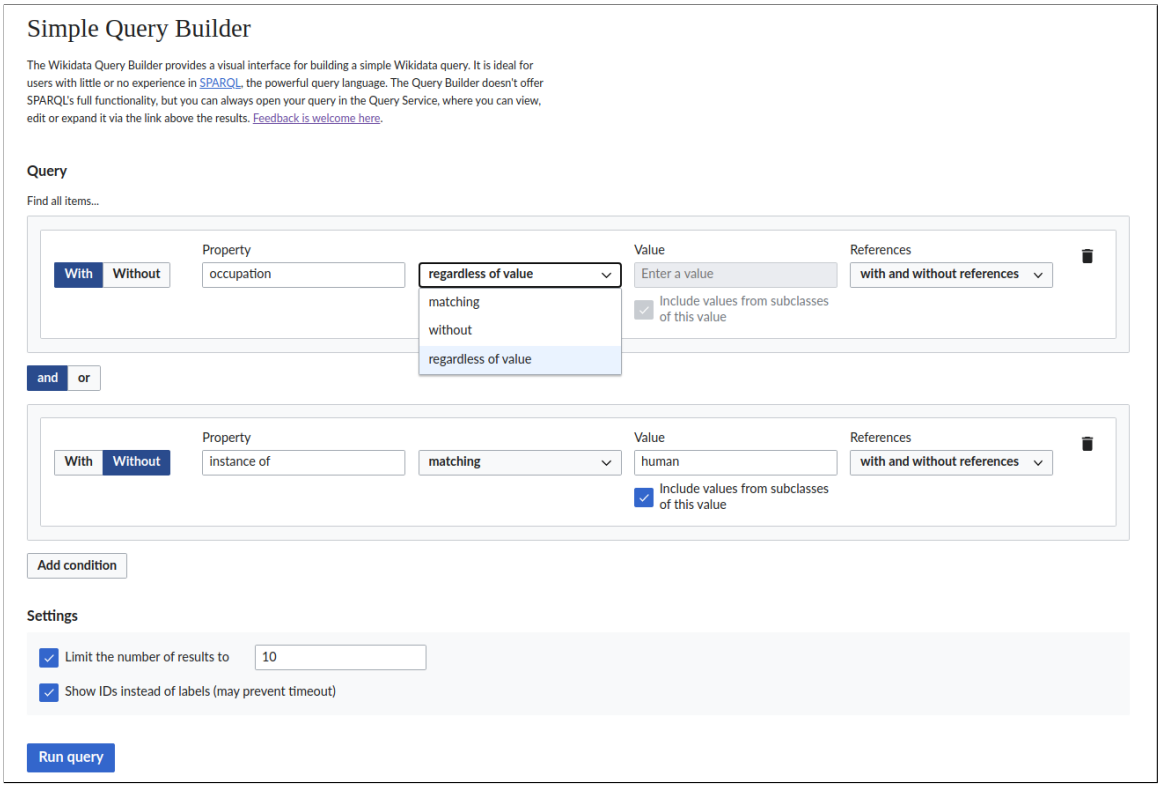

Figura 2.2: Interfaz de Constructor de Consultas de Wikidata: consulta de elementos que tienen una ocupación pero no son humanos.

Además, el Servicio de Consultas de Wikidata dispone de un panel de consultas de ejemplo (Figura 2.3). También los usuarios pueden solicitar consultas específicas a la comunidad[6].

[5]https://query.wikidata.org/querybuilder
[6]https://www.wikidata.org/wiki/Wikidata:Request$_{aq}uery$

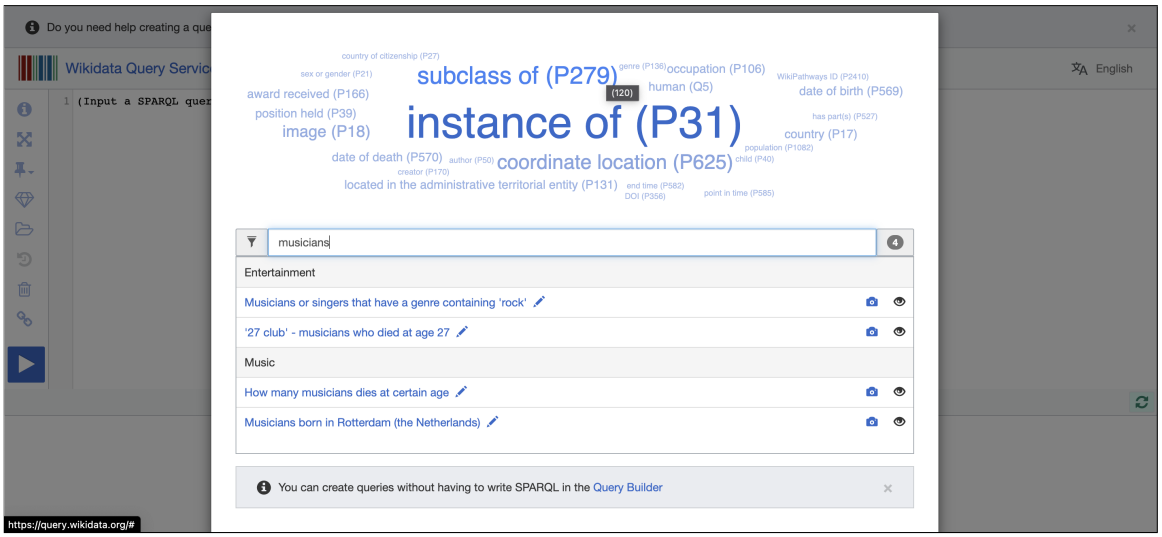

Figura 2.3: Servicio de Consultas de Wikidata: ejemplos de consultas según búsqueda *musician*.

2.2. Lenguaje natural a SPARQL

Las opciones anteriores acercan al usuario de Wikidata a la navegación sobre los datos enlazados. Sin embargo, aún necesitan de conocimiento técnico sobre cómo modelar los elementos o sobre cómo manejar el Servicio de Consultas. En contraste, la traducción de lenguaje natural a SPARQL resulta una aproximación más directa en términos de simplicidad para el usuario (por ejemplo, la consulta de la figura 2.2 se traduce en lenguaje natural a *¿Cuáles elementos tienen una ocupación pero no son humanos?*). En el presente, existen diversas soluciones al problema de QA en los datos enlazados. No obstante, se trata de una tarea compleja de abarcar en su totalidad, debido al amplio espectro de preguntas que se pueden generar en lenguaje natural. Los enfoques empleados en estas soluciones se dividen en dos tipos: no neuronales y neuronales.

2.2.1. Enfoques no neuronales

En algunos trabajos con enfoque no neuronal, se definen lenguajes de control mediante el uso de plantillas fijas. Otros utilizan la pregunta para guiar la exploración de un grafo, con el fin de encontrar nodos candidatos que respondan la consulta. En general, este tipo de enfoque posee una arquitectura más simple y su desempeño disminuye considerablemente al verse enfrentado a estructuras desconocidas de preguntas [3].

Enfoques basados en lenguajes naturales controlados: Este tipo de enfoque normalmente considera un subconjunto acotado bien definido de lenguaje natural, que permita ser interpretado sin ambigüedades por un sistema dado. Un ejemplo es Ginseng [1]. Este sistema permite realizar consultas en lenguaje cuasi natural a cualquier base de conocimiento OWL2[7]. Ginseng no usa un vocabulario predefinido y no intenta interpretar las consultas,

[7]https://www.w3.org/TR/owl-features/

sino que sólo utiliza el vocabulario definido por las ontologías cargadas al momento de la consulta, complementado con información de sinónimos. Por lo tanto, el vocabulario es cerrado y el usuario tiene que seguirlo. Esto limita la entrada del usuario, pero asegura que cada consulta conduce a un resultado coherente. Además, el vocabulario crece con cada ontología cargada adicionalmente.

Enfoques basados en gramáticas formales: Estos sistemas se encuentran generalmente basados en gramáticas lingüísticas. El beneficio principal es que puede tratar entradas de complejidad arbitraria si la estructura de ésta se encuentra cubierta por la gramática. Sin embargo, tiene como inconveniente su fragilidad, puesto que falla si la gramática no puede analizar una determinada pregunta. ORAKEL [2] y Pythia [14] son sistemas basados en ontologías. Esto significa que las ontologías juegan un papel central en la interpretación de las preguntas de los usuarios. Lo más importante de estos sistemas es que las preguntas de los usuarios se interpretan con respecto a una ontología subyacente al conjunto de datos que se consulta. Construyen representaciones cuyo vocabulario ya está alineado con el vocabulario de la ontología.

Enfoques basados en el mapeo de estructuras lingüísticas a estructuras semánticas compatibles con la ontología: Algunos sistemas como Aqualog [8] y PowerAqua [7] intentan hacer coincidir directamente estructuras lingüísticas con patrones semánticos triples. De esta forma, se calcula un mapeo biyectivo entre los elementos de la consulta y los recursos o predicados de la pregunta en lenguaje natural.

Enfoques basados en plantillas: Estos implementan un proceso de dos etapas. En primer lugar, se construye una plantilla en SPARQL a partir de un análisis de la estructura de la pregunta de entrada. Luego, se hacen coincidir los elementos de la pregunta en lenguaje natural con los elementos del conjunto de datos consultado. Estos enfoques son muy susceptibles a fallar debido a las posibles variaciones de la pregunta en lenguaje natural. TBSL [13] es un ejemplo de trabajo bajo este enfoque.

Enfoques de exploración de grafos: Estos enfoques implican el uso de la pregunta para guiar la exploración de un grafo. De esta se pueden extraer datos que sugieren nodos candidatos. Por ejemplo, en la pregunta *¿Quién es el director de Hogwarts?* se puede buscar primero nodos referentes a *Hogwarts* (`wd:Q174097` en Wikidata) y luego buscar la relación *director*, llegando a la respuesta buscada. Treo [4] y NLQ-KBQA [6] son ejemplos de estos enfoques [3]. El principal problema es la inviabilidad de búsquedas muy complejas, pues implican exploraciones muy profundas y poco eficientes.

2.2.2. Enfoques neuronales

Las soluciones más recientes aplican un enfoque neuronal y aprovechan los avances en la traducción automática neuronal (NMT, por sus siglas en inglés), traduciendo el lenguaje natural a SPARQL. A continuación, se describen algunos enfoques destacados en el área [3]:

En el trabajo de Ferreira Luz y Finger [10] se propone una representación vectorial para la pregunta en lenguaje natural y la consulta en SPARQL. Luego, se utiliza esta representación como entrada a una red neuronal con una arquitectura LSTM (Long Short-Term Memory)

con mecanismo de atención. El modelo codificador-decodificador LSTM es capaz de codificar lenguaje natural en inglés y decodificar consultas SPARQL. Respecto al lenguaje natural, se hace uso de Glove[11] para obtener la representación vectorial. En cuanto a las consultas SPARQL, se emplea una composición de métodos que generan la representación. Luego, se genera una tabla que es utilizada para emparejar palabras/entidades y obtener la traducción.

Un problema común en sistemas con enfoque neuronal es el problema *fuera de vocabulario* (OOV por sus siglas en inglés). Este se manifiesta con la presencia de términos o palabras que no ocurren en el conjunto de entrenamiento, resultando en dificultades para manejar estas palabras, puesto que el sistema no sabe cómo asignarlas al vocabulario de salida. En este trabajo no se trata el problema OOV, puesto que es evitado durante la representación vectorial y se asume que cada palabra está asociada a un vector de características, lo cual se compara a tener un vocabulario controlado.

Soru et al. [12] proponen una Máquina Neuronal SPARQL: un modelo de aprendizaje de extremo a extremo que traduce una expresión en lenguaje natural a una secuencia de tokens que cumplen con la gramática de SPARQL. La entrada es un conjunto de plantillas, compuestas por pares de preguntas y consultas, en donde la entidad objetivo se encuentra reemplazada por un marcador. Luego, a partir de un grafo de conocimiento, se calculan entidades que pueden ser reemplazantes válidos en estos marcadores. Con este conjunto generado, se entrena un modelo de codificador-decodificador LSTM. Los resultados muestran que es capaz de generar composiciones de nuevos patrones que no ocurrieron en el conjunto de entrenamiento. Con esta solución se aborda el problema OOV, sin embargo, siguen siendo necesarias millones de instancias para lograr cubrir todas las entidades disponibles en Wikidata.

Yin et al. [18] estudia los resultados de modelos basados en CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks) y transformadores con el fin de comparar distintos modelos NMT en este contexto. Los resultados indican que el modelo que mejor se ajusta es el ConvS2S (Convolutional Sequence to Sequence). La arquitectura del modelo con mejor desempeño también se compone de un codificador-decodificador con mecanismo de atención, donde ambos consisten de bloques convolucionales apilados. Este modelo también enfrenta problemas con vocabularios más amplios. Esto se atribuye al conjunto de datos utilizado en el entrenamiento (DBNQA[8]), el cual es limitado. Se propone extender el conjunto combinando métodos de generación de LC-QuAD[9] con las plantillas de DBNQA.

Lukovnikov et al. [9] abordan el problema OOV entrenando una red neuronal para responder preguntas simples de forma integral, dejando todas las decisiones al modelo. Clasifica pares de sujeto-predicado para así recuperar datos importantes de una pregunta. La red incluye un codificador de preguntas a nivel de palabra y carácter anidado, a modo de poder manejar palabras raras y fuera del vocabulario. El problema de este enfoque es que está específicamente diseñado para tratar con preguntas simples de patrón triple (por ejemplo, *¿Dónde nació Thom Yorke?*). Además, este modelo no aprende representaciones de símbolos específicos de grafos de conocimiento.

Wang et al. [17] comparan en gran escala tres arquitecturas de redes neuronales distintas

[8]https://github.com/AKSW/DBNQA
[9]http://lc-quad.sda.tech/

(RNN, CNN y un modelo transformador). Además, combinan estos modelos con vinculación de entidades (EL por sus siglas en inglés) y vinculación de relaciones (RL o Relation Linking) para abordar el problema OOV y generar plantillas. No hay discusión acerca del emparejamiento de entidades con los marcadores en las plantillas.

En el trabajo de Diomedi y Hogan [3], se identifica que las dificultades de vocabulario se concentran en las entidades, por lo que se propone un sistema llamado *ElNeuQA* (Figura 2.4), que realiza vinculación de entidades para extraer las entidades de la pregunta y ejecutar un relleno de ranuras (*slot filling* o SF) sobre el template de la consulta. Así, se reducen considerablemente los errores debido a entidades en comparación a las soluciones que emplean solo NMT. Un detalle fundamental de este sistema es que se hace uso de un conjunto de datos independiente que contiene plantillas de distintas complejidades. Es importante notar que, incluso con las técnicas con enfoque neuronal, consideradas dentro del estado del arte, los resultados alcanzan una precisión y recuperación de un 27 % con la mejor configuración. Esto no necesariamente se debe a las arquitecturas neuronales, sino que la causa radica en que los modelos necesitan una gran cantidad de pares de preguntas en lenguaje natural con su respectiva consulta en SPARQL para lograr un entrenamiento acabado, y generar estas colecciones resulta muy costoso.

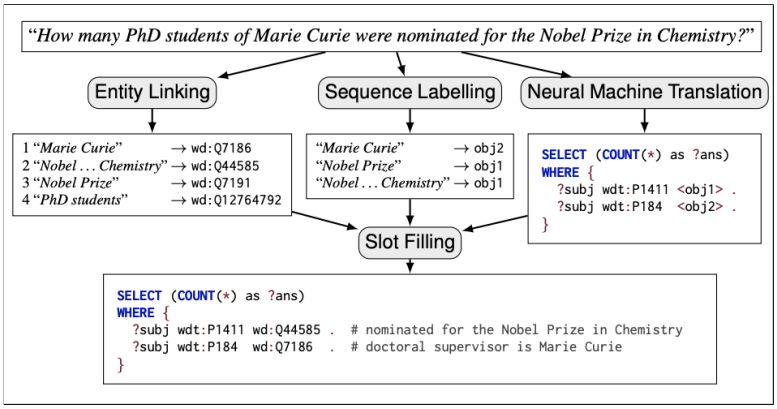

Figura 2.4: ElNeuQA: Descripción general del enfoque del sistema para una pregunta de entrada de ejemplo [3].

Conjuntos de datos como LC-QuAD son comúnmente utilizados para entrenar estos modelos, y están basados en pocas plantillas que generan varias instancias de un mismo tipo de pregunta, mediante reemplazo de constantes y parafraseo, obteniendo expresiones en lenguaje natural más diversas. Las redes neuronales logran buenos resultados sobre nuevas instancias creadas a partir de estas plantillas, sin embargo, no logran generalizar hacia nuevas estructuras de preguntas que no fueron vistas durante el entrenamiento.

2.3. QAWiki

QAWiki es un repositorio de preguntas en lenguaje natural y sus correspondientes consultas en SPARQL basado en el software Wikibase (Figura 2.5). Fue creado dentro del contexto del trabajo de Diomedi y Hogan [3], con el fin de solucionar problemas de calidad del conjunto

de datos LC-QuAD 2.0, 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 370 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 (Figura 2.6). Este conjunto de datos no sigue plantillas estándar y contiene consultas con una variedad de complejidades y características algebraicas, que van desde consultas simples como *¿Quién es el presidente de Polonia?*, hasta preguntas complejas como *¿Qué país no limita con Alemania tiene el alemán como idioma oficial?* [3].

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.

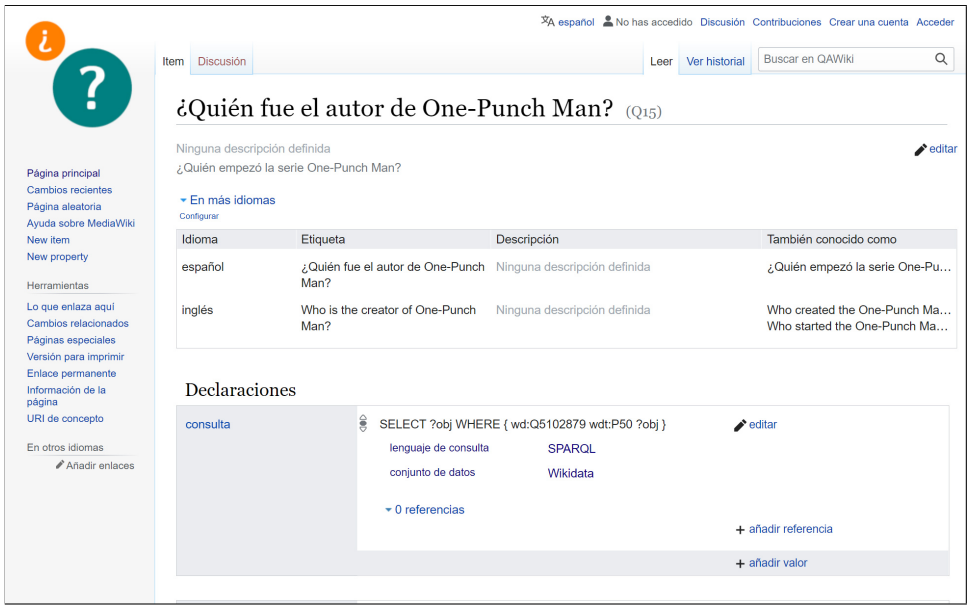

Figura 2.5: QAWiki: vista de pregunta *¿Quién fue el autor de One-Punch Man?*

2.4. Contexto del presente trabajo

La plataforma descrita en este trabajo, llamada Templet, es un sistema que entrega una demostración de QAWiki a los usuarios de la web. Teniendo en cuenta que la motivación principal de QAWiki es ofrecer una plataforma abierta y colaborativa con el fin de mantener un repositorio de pares de preguntas en lenguaje natural y consultas en SPARQL, es vital expandir su uso para que la comunidad añada, edite y solicite nuevos elementos. Así, es posible generar una amplia colección de pares que sistemas de QA pueden utilizar para mejorar su rendimiento.

Templet sugiere preguntas de QAWiki a partir de autocompletado en lenguaje natural, favoreciendo una navegación interactiva con el usuario. Extiende los pares de preguntas y consultas disponibles en QAWiki a plantillas, permitiendo que se generen nuevas preguntas y

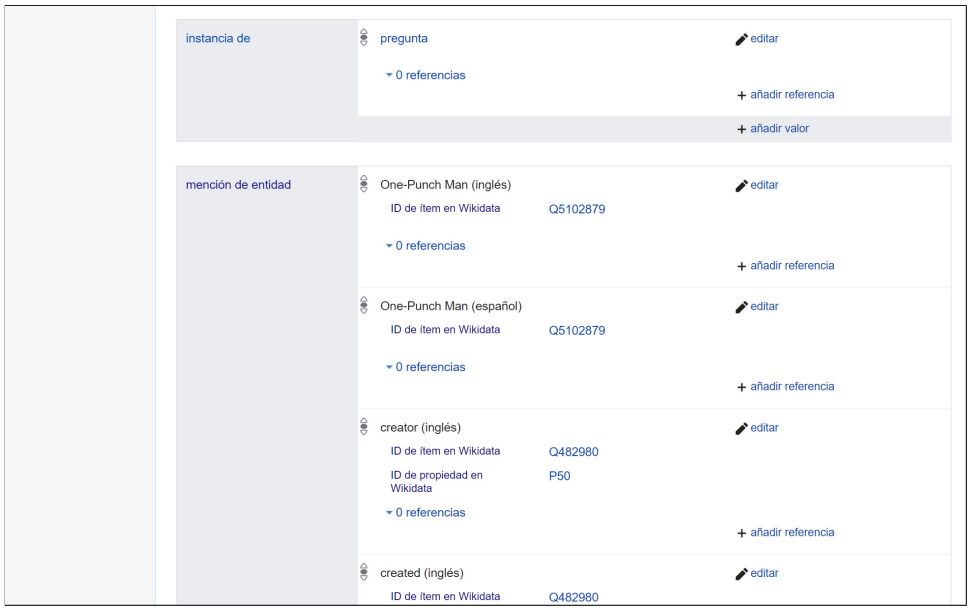

Figura 2.6: QAWiki: extracto de algunas entidades y menciones de *¿Quién fue el autor de One-Punch Man?*

consultas a partir de éstas. Por ejemplo, la pregunta *¿Cuál película de los últimos diez años ha ganado el mayor número de Premios Óscar?* se convierte en *¿Cuál (...) de los últimos diez años ha ganado el mayor número de (...)?*, en donde los espacios a completar son enriquecidos con un autocompletado conectado a la API de Wikidata. De esta forma, los usuarios pueden interactuar con los distintos elementos de Wikidata y generar nuevas consultas a partir del repositorio de QAWiki, sin necesidad de hacer uso de lenguajes especializados como SPARQL.

Capítulo 3

Diseño de la solución

En el presente capítulo se describen los requisitos funcionales y no funcionales del sistema, las tecnologías de desarrollo seleccionadas y la arquitectura de la solución.

3.1. Requisitos del sistema

El sistema descrito en el presente trabajo consiste en una aplicación web capaz de consumir datos desde QAWiki y Wikidata, con el fin de crear una plataforma, que a partir del autocompletado en lenguaje natural, sugiere preguntas de QAWiki extendidas a plantillas. Luego, ejecuta las consultas en SPARQL correspondientes a estas en el Servicio de Consultas de Wikidata y muestra los resultados obtenidos.

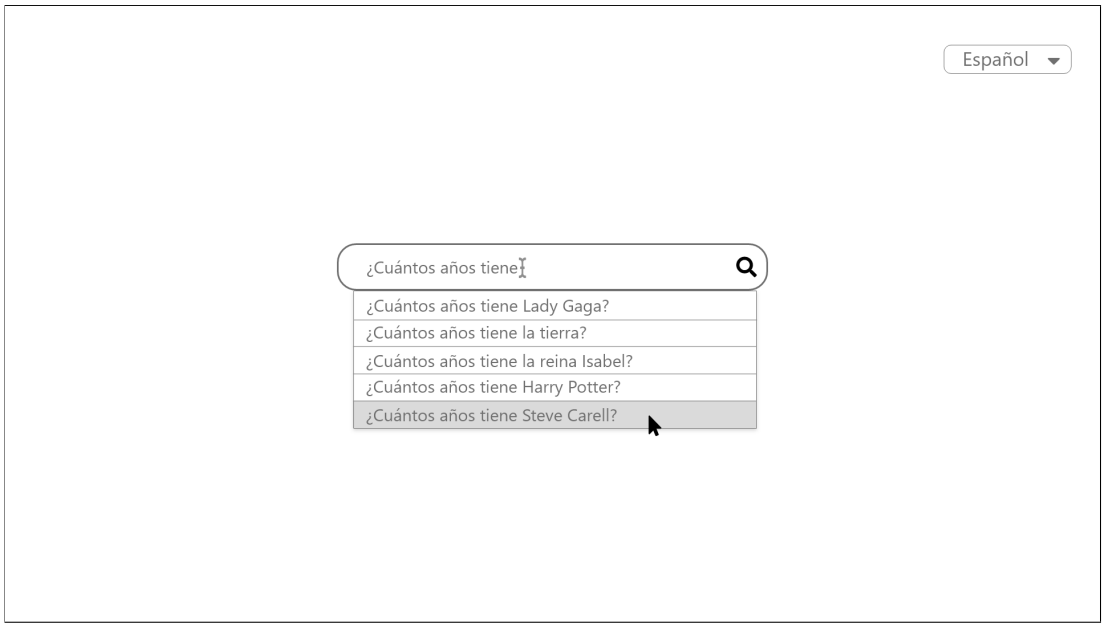

Figura 3.1: Mock-up inicial del sistema

La solución propuesta tiene dos aristas principales, siendo la primera la relacionada a la

generación de plantillas a consumir por el sistema, y la segunda, la aplicación web como tal.

La generación de plantillas a partir de las preguntas disponibles en QAWiki es un paso previo al despliegue de la aplicación web, y que debe seguir ejecutándose periódicamente para actualizar la colección de preguntas disponibles.

La aplicación web es una interfaz simple de una única vista, capaz de transicionar según las interacciones del usuario.

A continuación, se describen los requisitos necesarios para las funcionalidades mencionadas.

3.1.1. Requisitos funcionales

- El sistema debe tener una entrada principal de texto, en donde se buscan las plantillas de preguntas en el idioma seleccionado.
- Templet debe disponer de autocompletado en todas las entradas de texto, teniendo en la entrada principal las plantillas disponibles, y en las entradas de una pregunta en particular, las sugerencias de entidades desde Wikidata.
- Al momento de solicitar preguntas o resultados, todos los campos de entrada del sistema deben contener valores previamente sugeridos en la lista de selección del autocompletado. No deben ser aceptados elementos externos a estos resultados.
- Templet debe ofrecer un enlace al Servicio de Consultas de Wikidata con la consulta en SPARQL de la pregunta solicitada al momento de consultar.
- El sistema debe disponer, para cada pregunta, un enlace hacia la pregunta original en QAWiki.
- Templet debe desplegar los resultados de una consulta de forma atractiva para el usuario, incluyendo para cada entidad, una etiqueta, imagen, descripción y un enlace del elemento en Wikidata, sujetos a la disponibilidad de éstos.
- Templet, en su completitud, debe estar disponible en todos los idiomas de QAWiki, que al momento de la redacción del presente informe, son dos: inglés y español.
- Templet debe generar las plantillas de preguntas a partir de QAWiki. Estas deben actualizarse periódicamente cada cierto tiempo.

3.1.2. Requisitos no funcionales

- Templet debe responder a las entradas de texto del usuario en un tiempo suficientemente corto, tomando no más que un par de segundos en sugerir resultados. Esto incluye la entrada de texto principal, correspondiente a las plantillas, y a aquellas que sugieren entidades de Wikidata.

- Templet debe retornar los resultados desde el Servicio de Consultas de Wikidata en un tiempo similar al que el servicio propio los entrega. Es decir, el postprocesamiento de Templet debe tomar unos pocos segundos en desplegar la información al usuario.
- El sistema debe ser intuitivo y fácil de usar para los usuarios de la web.
- La extensión del sistema a otros idiomas debe ser una operación simple para los mantenedores, por lo que no deben haber valores escritos en código, sino que debe existir un archivo configurable del cual se extraen.

3.2. Elección de las tecnologías de desarrollo

Luego de un estudio de las distintas tecnologías disponibles para la creación de Templet, se determinó que el sistema se desarrollaría utilizando las siguientes herramientas:

- El backend del sistema se desarrolló en Python, utilizando el framework Flask. La principal razón de esta elección es la experiencia previa en este lenguaje que facilita el desarrollo. El framework se caracteriza por su minimalismo, pues se pueden desplegar aplicaciones web rápidamente y con un mínimo número de líneas de código. Esto resulta conveniente dadas las características de Templet, ya que no posee un modelo de datos y necesita de scripts auxiliares para la creación de plantillas a partir de las preguntas de QAWiki. Además, existe una biblioteca de Python llamada APScheduler, que permite programar la actualización periódica de las plantillas en pocos pasos y sin necesidad de hacer modificaciones en la arquitectura del sistema.
- El frontend de Templet se compone de una única vista minimalista, por lo que no se hizo necesario utilizar frameworks más completos (como React o Angular, por ejemplo), sino que se decidió construirlo en HTML, CSS y Javascript nativo. También se utilizó la librería jQuery para las transiciones según las interacciones y distintos eventos de la plataforma, y el framework Bootstrap para los estilos del sistema y para facilitar la adaptabilidad entre distintos dispositivos.
- Debido a que la aplicación se debió implantar en el servidor, para el entorno de producción se utilizó Gunicorn. De esta forma, Templet puede comunicarse con varios servidores web, reaccionar a múltiples peticiones a la vez y mantener varios procesos de la aplicación web en ejecución.
- Para facilitar la implantación de la aplicación en los servidores del DCC, esta se encuentra empaquetada en un contenedor Docker.
- Para el control de versiones, se utilizó Git.

3.3. Arquitectura de la solución

La arquitectura de Templet (Figura 3.2) está construida según los requisitos y herramientas descritas anteriormente. A grandes rasgos, el sistema se encuentra dentro de un contenedor Docker, en donde el backend tiene dos funciones:

- Gestionar las peticiones desde el frontend, generando las pertinentes consultas a las APIs de Wikidata, limpiando estos resultados, para finalmente retornarlos al frontend.
- Manejar la lógica para actualizar las plantillas, generadas a partir de las preguntas disponibles en QAWiki, haciendo consultas a sus APIs. Estas plantillas son cacheadas en un archivo JSON, que luego es consumido por el frontend.

Respecto al frontend, es el encargado de alimentar la entrada principal a partir del JSON de plantillas, de autocompletar las entidades y de enviar las nuevas consultas SPARQL (creadas a partir del input del usuario) hacia el backend. También debe mostrar los resultados obtenidos, o los errores, dependiendo del caso.

3.4. Flujo de la aplicación

La aplicación tiene un único flujo (Figura 3.3), el cual comienza con el input del usuario sugiriendo preguntas en el idioma seleccionado. A continuación, si el usuario clickea una de las sugerencias, se muestra el formulario correspondiente a la pregunta, en donde debe completar todos los campos con elementos sugeridos desde Wikidata. Al completarlos correctamente, se puede realizar la nueva consulta SPARQL en el Servicio de Consultas de Wikidata. Luego, al cliquear el botón *Buscar*, se envía la petición al backend y se verifica si la pregunta tiene asociada una *pregunta contingente*.

Una pregunta contingente es una pregunta que también pertenece al universo de preguntas de QAWiki (por lo que en nuestro sistema, también tiene una plantilla asociada), que tiene un valor esperado que determina si la pregunta principal tiene sentido. Por ejemplo, la pregunta *¿Cuántos años tenía Justin Bieber cuándo murió?*[1] tiene asociada la pregunta contingente *¿Está muerto Justin Bieber?*[2] con *verdadero* como valor esperado. Esto quiere decir que si el resultado de la pregunta contingente es verdadero, la pregunta principal tiene validez.

Si dentro del flujo la pregunta seleccionada tiene una pregunta contingente y esta tiene el valor esperado, se retorna el resultado de la pregunta principal, es decir, la cliqueada al principio. Este resultado puede contener entidades de Wikidata, estar vacío (pues la pregunta/consulta no aplica a ninguna entidad) o contener algún mensaje de error desde el servidor o Wikidata. Si la pregunta contingente no tiene el resultado esperado, se retorna esta pregunta y su respuesta, que es booleana.

[1] http://qawiki.dcc.uchile.cl/wiki/Item:Q146
[2] http://qawiki.dcc.uchile.cl/wiki/Item:Q145

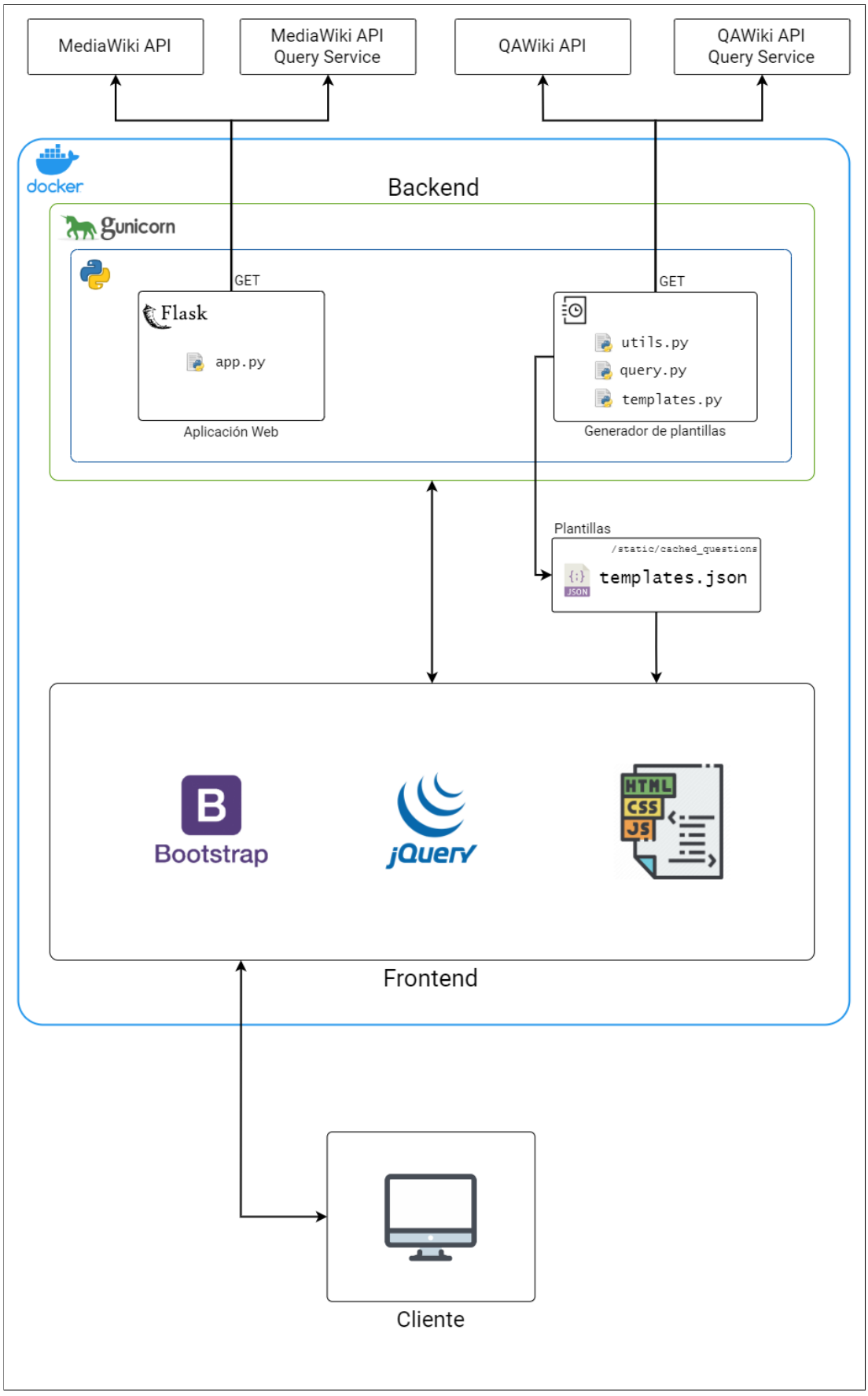

Figura 3.2: Templet: arquitectura de la solución

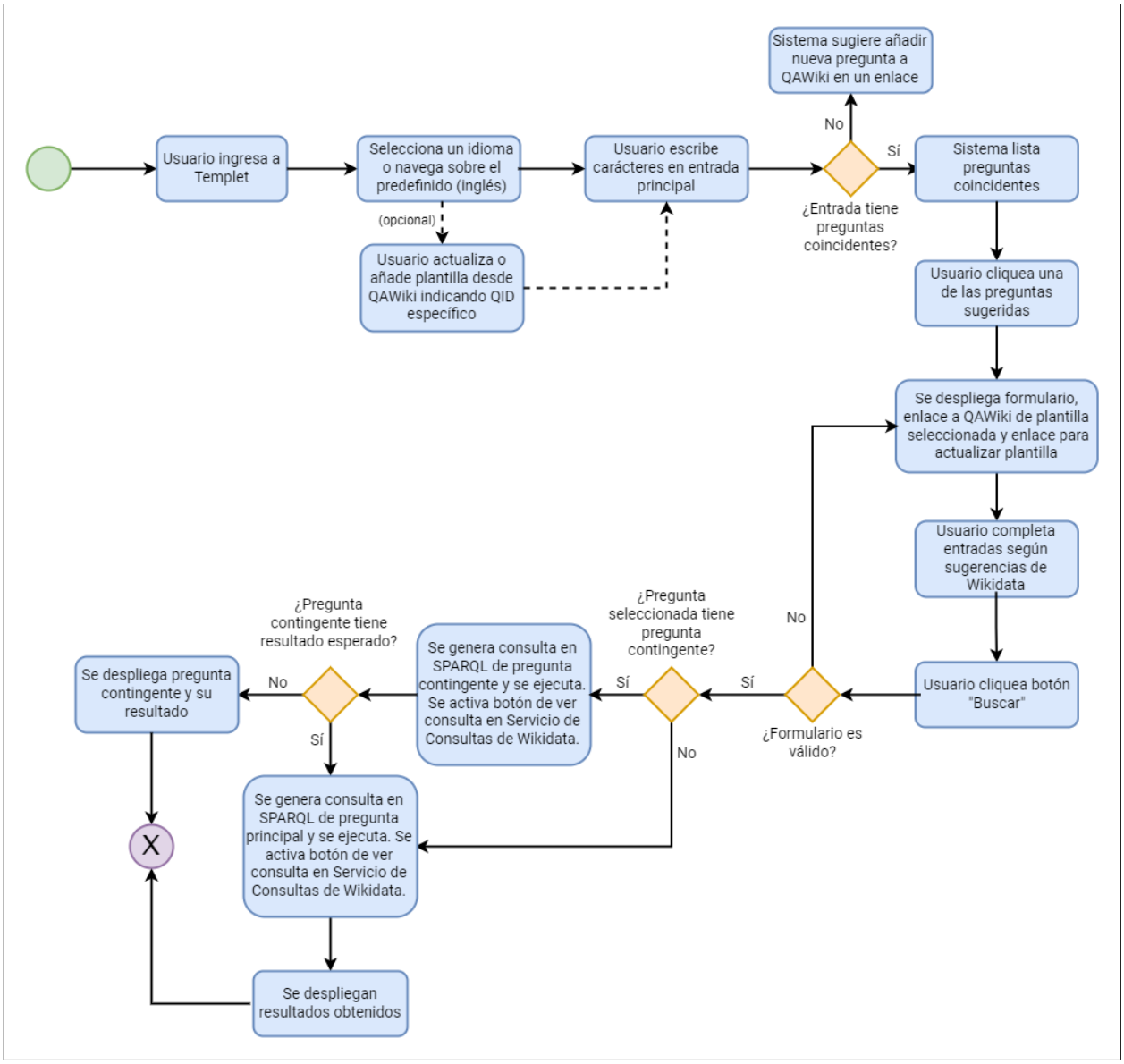

Figura 3.3: Flujo de Templet

Capítulo 4

Implementación de la solución

En esta sección se describen los aspectos de la implementación del sistema según el diseño descrito en el capítulo anterior.

4.1. Backend

El backend del sistema se encuentra construido en Python, con el framework Flask, y se descompone en dos áreas: la encargada de generar las plantillas de preguntas según las preguntas de QAWiki, y el backend de la plataforma web como tal.

4.1.1. Generación de plantillas

Parte fundamental del sistema es la generación de plantillas a partir de las preguntas disponibles en QAWiki. La solución desarrollada para este propósito consiste en identificar los ítems objetivos de la pregunta según las ID de estos en la consulta, y mapear estos espacios destinados a una entidad, en la consulta. Por ejemplo, en la pregunta *¿Cuánto mide Lana del Rey?* se tiene la correspondiente consulta en SPARQL:

```
SELECT ?obj WHERE {
wd:Q37150 p:P2048/psn:P2048/wikibase:quantityAmount ?obj
}
```

A través de la API de QAWiki, se puede obtener esta consulta y la ID del ítem objetivo, el cual corresponde a `wd:Q37150`, referente a *Lana del Rey*. Con lo anterior, se puede generar la plantilla de la forma *¿Cuánto mide [entidad]?* y la consulta análoga. La descripción detallada de este procedimiento (que llamaremos Mapeo de Preguntas a Consultas o MPC) se encuentra a continuación:

- Se extraen las entidades tipo `wd:Q{n}` de la consulta SPARQL correspondiente a una pregunta.
- Para cada una de estas entidades obtenidas, se deben filtrar las menciones que cumplan lo siguiente:
  - La entidad correspondiente a la mención tiene el mismo QID que la entidad objetivo.
  - Comparte el lenguaje de pregunta.
  - Están contenidas en la pregunta en lenguaje natural.

Mediante este proceso, lo común es que quede una mención por entidad, logrando un mapeo *mención* $\iff$ `wd:Q{n}`. La notación a utilizar para emparejar menciones y entidades es la siguiente: `$mention_n` $\iff$ `$entity_n`. Por ejemplo, en la pregunta *¿Cuántos Premios Turing han ganado personas de Austria?*[1] se tiene la plantilla en lenguaje natural `¿Cuántos $mention_1 han ganado personas de $mention_0?` y la consulta SPARQL:

```
SELECT ( COUNT ( ?sbj ) AS ?value ) WHERE { ?sbj wdt:P166 $entity_1 . ?sbj wdt:P27 $entity_0 . }
```

Para extraer las plantillas de todas las preguntas disponibles en QAWiki, se desarrollaron múltiples scripts en Python. De la Figura A, estos corresponden a:

- `query.py`, que se encarga de gestionar las peticiones a las APIs de QAWiki (y MediaWiki);
- `templates.py`, que contiene el código que explícitamente genera las plantillas;
- `utils.py`, que contiene algunas funciones auxiliares para leer y guardar archivos JSON, y la función que envuelve todas las funciones necesarias para generar las plantillas y exportarlas a un archivo JSON.

Respecto al procedimiento para crear las plantillas que realizan estos scripts, en primer lugar extraen los QIDs de las preguntas disponibles en QAWiki, ejecutando la siguiente consulta en el Servicio de Consultas de QAWiki:

```
SELECT * WHERE { ?q wdt:P1 wd:Q1 }
```

Luego, se itera sobre estos QIDs para comenzar a armar una lista de diccionarios, en donde cada uno identifica a una pregunta. Para cada pregunta, se tiene el siguiente flujo:

1. Se crea un nuevo diccionario

[1] http://qawiki.dcc.uchile.cl/wiki/Item:Q71

2. Se crea la llave `id` en el diccionario, la cual contiene el QID de la pregunta.

3. Se verifica si tiene una pregunta contingente asociada

   - Si tiene una pregunta contingente, se crea una nueva llave en el diccionario, llamada `contingent_question`, que contiene un diccionario anidado con datos de la pregunta contingente, incluyendo las llaves `id`, correspondiente al QID, `query_raw`, la consulta en SPARQL cruda (original sin modificaciones) y `expected_value`, el valor esperado.
   - Si no tiene una pregunta contingente, se deja en nulo el valor de la llave `contingent_question`.

4. Se comienza a iterar sobre los idiomas disponibles en QAWiki, que al momento de la redacción de este informe, son inglés y español. Dentro de estos pasos, `{lang}` se refiere al idioma en el que se encuentra la iteración.

   4.1. Se crea dentro del diccionario una nueva llave llamada `question_{lang}`, que identifica la pregunta en lenguaje natural.

   4.2. Haciendo uso del método MPC descrito anteriormente, se agregan:

   - `matches_{lang}`, que identifican las menciones contenidas en la pregunta en lenguaje natural;
   - `question_template_{lang}`, que son las plantillas de la pregunta en lenguaje natural;
   - `query_template_{lang}`, que corresponden a las plantillas de la consulta SPARQL.

   4.3. Para facilitar el despliegue en el frontend, se añade la llave `visible_question_{lang}`, que reemplaza el mapeo de menciones por *(...)* y evita que el usuario se confunda al ver las menciones.

   4.4. Si la pregunta tenía una pregunta contingente, se completa la pregunta en lenguaje natural de ésta en el diccionario anidado contenido en `contingent_question`, en la llave `question_raw_{lang}`. Finalmente, utilizando el método MPC, se emparejan las menciones de la pregunta principal a la pregunta contingente, pues se refieren a las mismas menciones/entidades. Esto se almacena en el diccionario anidado en las llaves `question_template_{lang}` y `query_template_{lang}`, de la misma forma que en la pregunta principal.

En el código 2 se tiene la plantilla para la pregunta *¿Cuántos años tiene Justin Bieber?*[2], obtenida a partir del método descrito. Allí se puede observar la pregunta contingente *¿Está vivo Justin Bieber?*[3].

```
1 {
2    "id": "Q143",
3    "contingent_question": {
4      "id": "Q144",
```

[2]http://qawiki.dcc.uchile.cl/wiki/Item:Q143
[3]http://qawiki.dcc.uchile.cl/wiki/Item:Q144

```
      "query_raw": "ASK WHERE { FILTER NOT EXISTS { wd:Q34086 wdt:P570 ?obj2 } }",
      "expected_value": 1,
      "question_raw_en": "Is Justin Bieber alive?",
      "question_template_en": "Is $mention_0 alive?",
      "query_template_en": "ASK WHERE { FILTER NOT EXISTS { $entity_0 wdt:P570 ?obj2 } }",
      "question_raw_es": "¿Está vivo Justin Bieber?",
      "question_template_es": "¿Está vivo $mention_0?",
      "query_template_es": "ASK WHERE { FILTER NOT EXISTS { $entity_0 wdt:P570 ?obj2 } }"
    },
    "question_en": "What age is Justin Bieber?",
    "matches_en": [
      {
        "mention": "Justin Bieber",
        "entity": "Q34086"
      }
    ],
    "question_template_en": "What age is $mention_0?",
    "query_template_en": "SELECT (YEAR(NOW())-YEAR(?obj) - IF(MONTH(NOW())>MONTH(?obj),0,IF(MONTH(NOW())=MONTH(?obj) && DAY(NOW())>=DAY(?obj),0,1)) AS ?age) WHERE { $entity_0 wdt:P569 ?obj }",
    "visible_question_en": "What age is (...)?",
    "question_es": "¿Cuántos años tiene Justin Bieber?",
    "matches_es": [
      {
        "mention": "Justin Bieber",
        "entity": "Q34086"
      }
    ],
    "question_template_es": "¿Cuántos años tiene $mention_0?",
    "query_template_es": "SELECT (YEAR(NOW())-YEAR(?obj) - IF(MONTH(NOW())>MONTH(?obj),0,IF(MONTH(NOW())=MONTH(?obj) && DAY(NOW())>=DAY(?obj),0,1)) AS ?age) WHERE { $entity_0 wdt:P569 ?obj }",
    "visible_question_es": "¿Cuántos años tiene (...)?"
  }
```

Código 2: Plantilla de pregunta *¿Cuántos años tiene Justin Bieber?*

En un principio, se esperaba generar las plantillas en tiempo real. Es decir, mientras el usuario tipeaba se iban procesando las preguntas de QAWiki y generando las plantillas. La latencia empleando este método era notoria y dificultaba la navegación, perdiendo la fluidez del autocompletado. Se optó por almacenar en caché las preguntas de QAWiki, en un archivo JSON, de forma que el frontend pudiera consumirlo fácilmente.

Este archivo debe ser actualizado periódicamente, por lo que se utilizó la librería APScheduler para ejecutar esta tarea. En particular, se usó `BackgroundScheduler`, un programador de tareas proporcionado por la librería que se ejecuta en segundo plano como un hilo separado. Esto permite que la tarea coexista con otras aplicaciones y evita que el hilo principal se bloquee.

El periodo de actualización de las plantillas es una variable de entorno configurable, e indica la cantidad de minutos. Por el momento, se fijó el valor en 60 minutos, pero para poder ver los cambios en QAWiki reflejados más rápidamente en Templet, se agregó una función para actualizar la plantilla de una pregunta particular de QAWiki a petición del usuario, que se describe en la Subsección 4.1.2 y la Sección 4.2.

4.1.2. Backend de plataforma web

El backend tiene definidas 5 rutas en el script principal de la aplicación web (`app.py`, en el directorio base *templet*, Figura A)

- La ruta base (`/`) es la página principal del sitio, y renderiza el *template* `index.html` (Figura A, en `templet/static/index.html`), que corresponde a la única vista del sistema.
- `/wikidata_search` admite peticiones `POST` y se encarga de recibir las entradas de texto de los formularios creados a partir de las plantillas. Esta ruta está conectada a la API de MediaWiki, que con una simple entrada de texto y un idioma determinado, sugiere entidades coincidentes de Wikidata. Gracias a esta ruta se alimenta el autocompletado de entidades en los formularios de las plantillas.
- `/wikibase_results` admite peticiones `GET` y se encarga de recibir consultas en SPARQL, ejecutarlas en la API del Servicio de Consultas de MediaWiki, y retornar los resultados (o errores, de ser el caso). Desde esta ruta se obtienen los resultados de las consultas generadas a partir del formulario que completa el usuario.
- `/question_template/<question_id>` admite peticiones `POST` y tiene la función de identificar si un determinado QID (correspondiente al parámetro `question_id`) existe en las plantillas disponibles. De esta forma, se puede formar una URL en el frontend que identifica el formulario para cada pregunta, y de solicitarse una plantilla o URL que no existe, se retorna un error 404.
- `/update_template/<question_id>` permite peticiones `GET` y se encarga de actualizar una plantilla de una pregunta con un *QID* en particular (`question_id`). De esta forma, si un usuario añade una nueva pregunta o hace cambios en una pregunta existente en QAWiki, puede visualizarlos inmediatamente y no necesita esperar el próximo escaneo general de preguntas.
- La ruta `/setcookie` permite peticiones `POST` y se utiliza para asignar el valor de la *cookie* que determina el idioma de la página web. Así, el sistema recuerda la elección del usuario.

4.2. Frontend

El frontend de Templet se caracteriza por ser minimalista, pues no necesita de demasiadas funcionalidades en la interfaz. Por esta razón, está construido en HTML, CSS y Javascript nativo, usando también el framework Bootstrap y la librería jQuery.

4.2.1. Interfaz

Templet tiene una vista única, que interactúa con el usuario y transiciona con ayuda de Javascript y jQuery. Al ingresar, se observa el selector de idioma y la entrada principal de texto (Figura 4.1), la cual tiene autocompletado y se encuentra conectada directamente a las plantillas cacheadas en `templates.json`. Esta entrada de texto es procesada para ser insensible a tildes y al símbolo de interrogación de apertura (`¿`). Además, a través de un archivo de configuración (`inputTokens.json`), se pueden crear equivalencias entre palabras, como *qué* con *cuáles*, permitiendo sugerir preguntas de estructura similar.

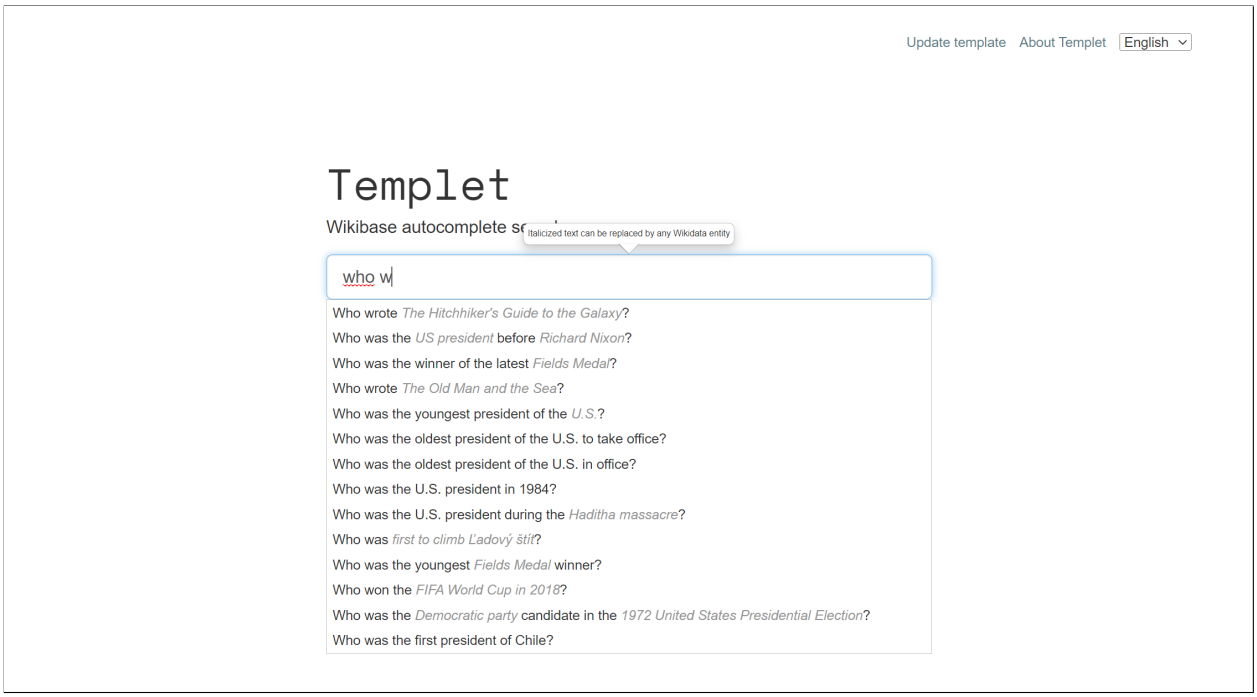

Figura 4.1: Templet: entrada de texto principal

Al cliquear en una de las plantillas, se muestra el formulario. Se incluye un enlace para volver a la página principal, el enlace de la pregunta original en QAWiki y un enlace para actualizar la plantilla seleccionada. En la Figura 4.2 se tiene la plantilla de la pregunta *¿Quién fue el presidente de los EEUU antes que Richard Nixon?*[4]. Como se puede observar, en las entradas de texto se mantuvo un *placeholder* con la mención original. Esto con el fin de guiar la consulta del usuario, ya que hay algunas plantillas que pueden resultar poco intuitivas (por ejemplo, la pregunta original *¿Cuáles autos gasóleos fabrica Honda?*[5] se transforma en

[4]http://qawiki.dcc.uchile.cl/wiki/Item:Q137
[5]http://qawiki.dcc.uchile.cl/wiki/Item:Q188

¿Cuáles (...) (...) fabrica (...)?).

Figura 4.2: Templet: plantilla de pregunta `Q137` de QAWiki

Cada una de las entradas de texto de la plantilla está conectada a la API de MediaWiki, a través del *endpoint* `/wikidata_search`, descrito anteriormente en la subsección Backend de plataforma web. Esta API retorna las etiquetas y descripciones de las entidades sugeridas en el idioma seleccionado, según la disponibilidad de estos atributos. En la Figura 4.3 se muestran estas sugerencias.

Para poder enviar la pregunta y transformarla a una consulta SPARQL, todos los campos deben ser completados cliqueando una de las entidades sugeridas, como en la Figura 4.4.

De haber errores en las entradas de texto, se genera un *popover* sobre los campos que tienen problemas (Figura 4.5).

Si los campos son válidos, se genera la consulta en SPARQL, se muestra un enlace hacia el Servicio de Consultas de Wikidata con la consulta adosada y se comienzan a cargar los resultados (Figura 4.6).

A continuación, se despliega el resultado de la pregunta ingresada, mostrando la etiqueta, imagen y descripción de la entidad en el idioma seleccionado (Figura 4.7). Para obtener estos datos, se genera una consulta en SPARQL que retorna esta información (Código 4), redimensionando la imagen original a una más pequeña, para evitar la lentitud al cargar. Esta consulta se ejecuta en el *endpoint* `/wikibase_results` y puede traer estos datos para una o múltiples entidades. Así, no se satura la API del Servicio de Consultas de MediaWiki.

Es importante notar que solo los resultados tipo URI, que contienen un enlace a una entidad, pueden ser desplegados de forma enriquecida con imágenes y texto de apoyo. Hay otros resultados literales que son valores crudos, como en la Figura 4.12.

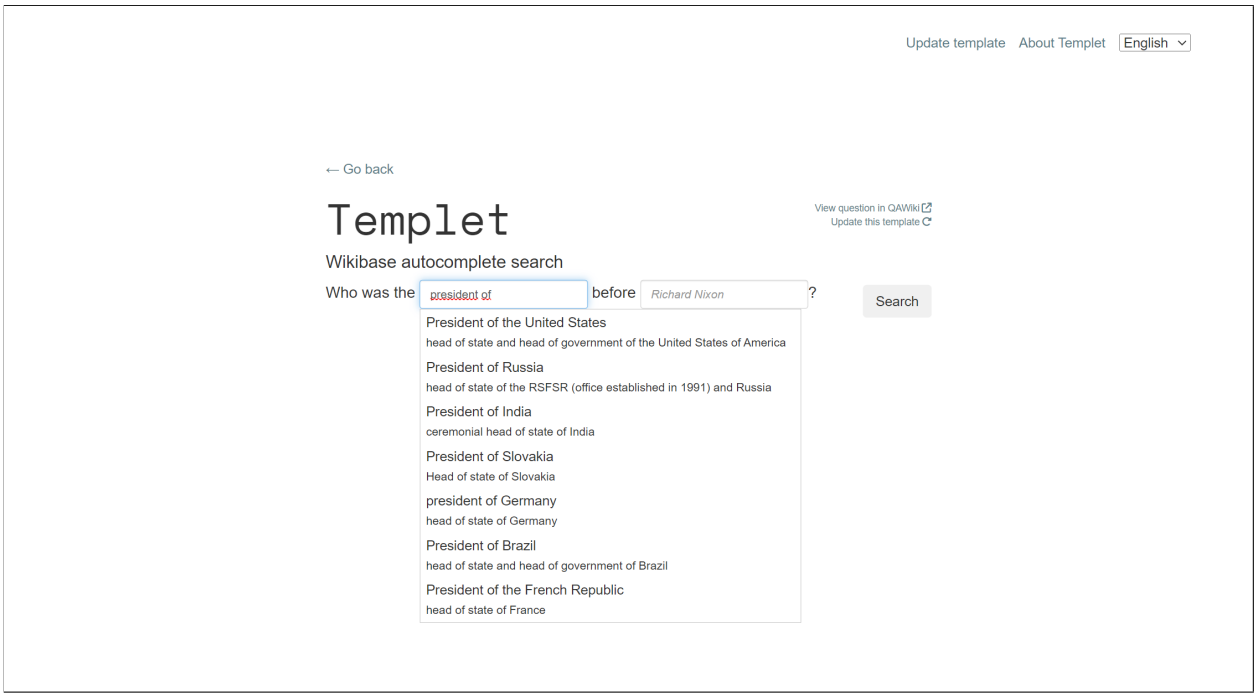

Figura 4.3: Templet: autocompletado en entradas de texto de plantilla `Q137`

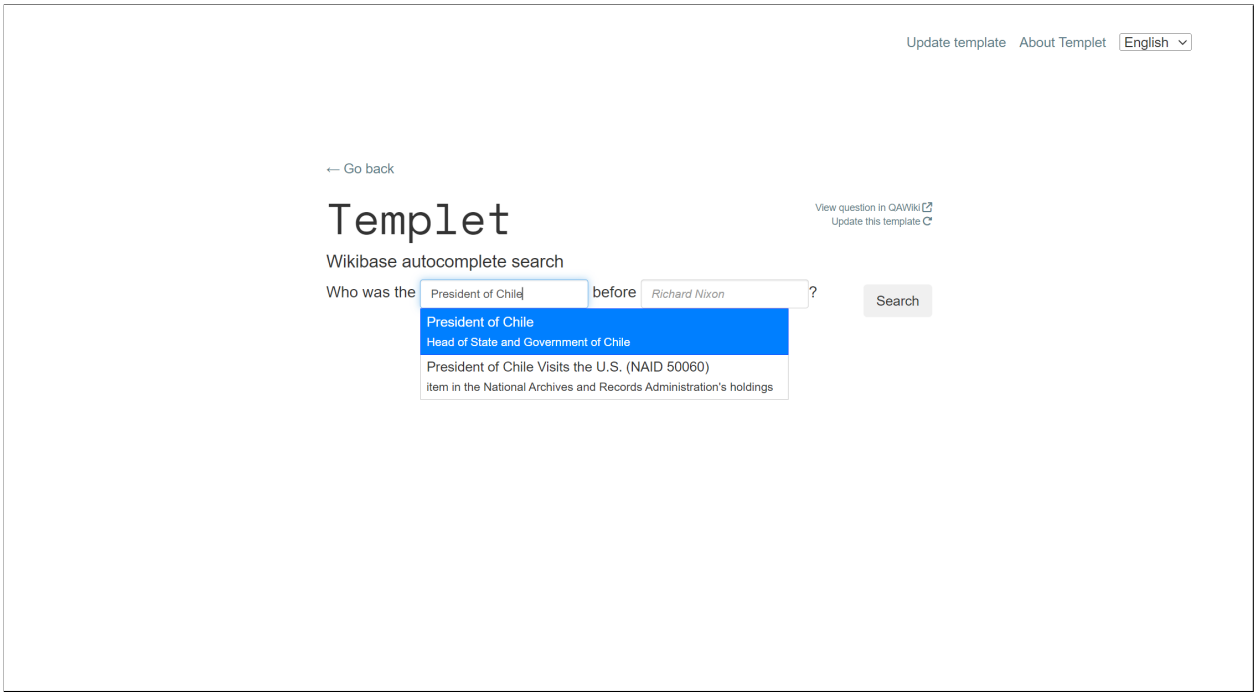

Figura 4.4: Templet: selección de entidad en plantilla `Q137`

También el sistema soporta resultados tipo tablas, como en la Figura 4.9.

Si la plantilla seleccionada tiene una pregunta contingente asociada, y el resultado de ésta no es el esperado, se despliegan los resultados como en la Figura 4.10. Por el contrario, si se tiene el resultado esperado, se muestra de forma normal el resultado de la pregunta principal.

Figura 4.5: Templet: error en campo en plantilla `Q137`

Figura 4.6: Templet: carga de resultados en consulta generada a partir de plantilla `Q137`

Además, el sistema posee un pequeño panel (Figura 4.11) en donde el usuario puede ingresar el QID de una pregunta en QAWiki para agregar una pregunta recientemente añadida (y que no ha sido escaneada automáticamente) o actualizar una pregunta existente en el catálogo de plantillas cacheadas. Si la pregunta existe, aparece un mensaje indicando que la tarea se ejecutó correctamente y se redirecciona a la pregunta. Si no existe, se despliega un

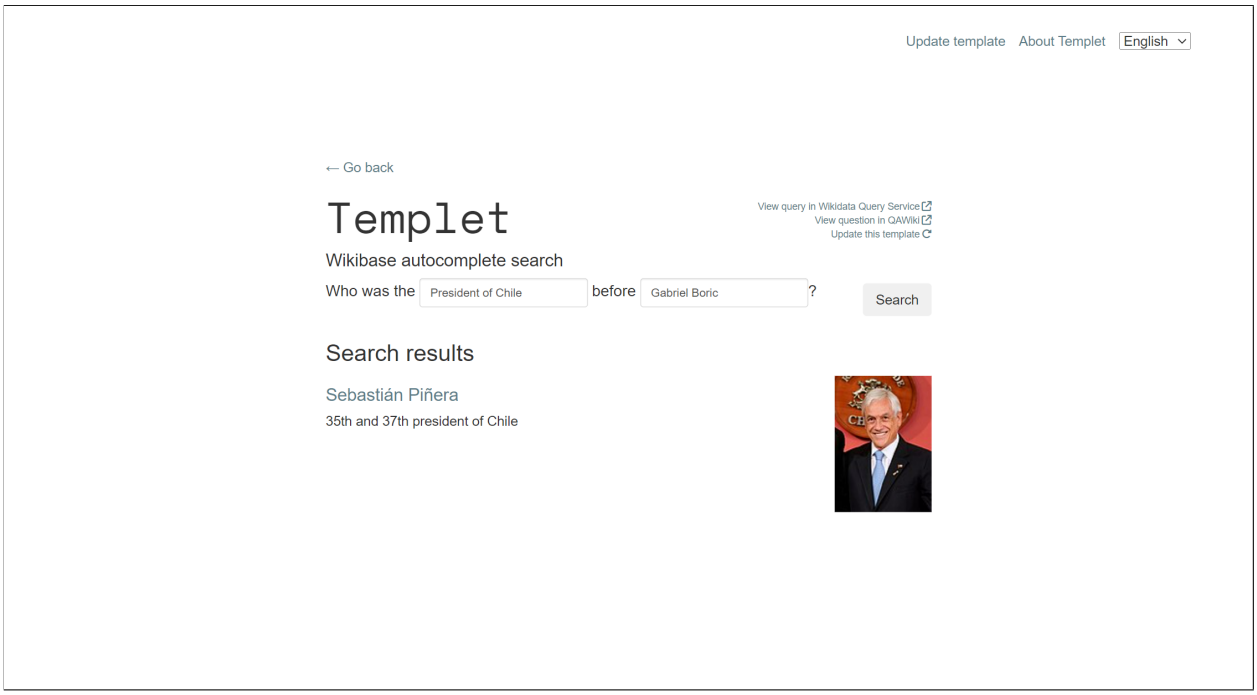

Figura 4.7: Templet: despliegue de resultados en consulta generada a partir de plantilla `Q137`

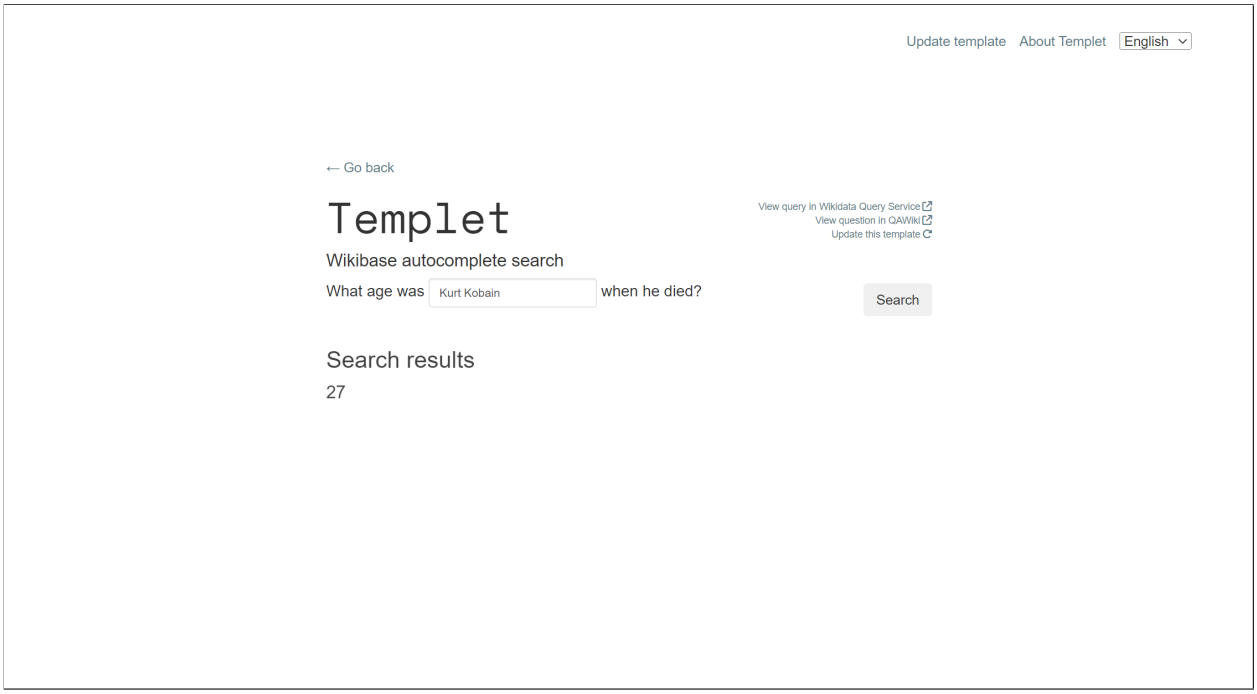

Figura 4.8: Templet: despliegue de resultados crudos a partir de plantilla `Q65`

mensaje de error y se envía al usuario a la página principal.

Finalmente, se decidió hacer la interfaz adaptativa a distintos dispositivos, por lo que se visualiza de forma amigable desde dispositivos móviles, como se puede ver en la Figura 4.12

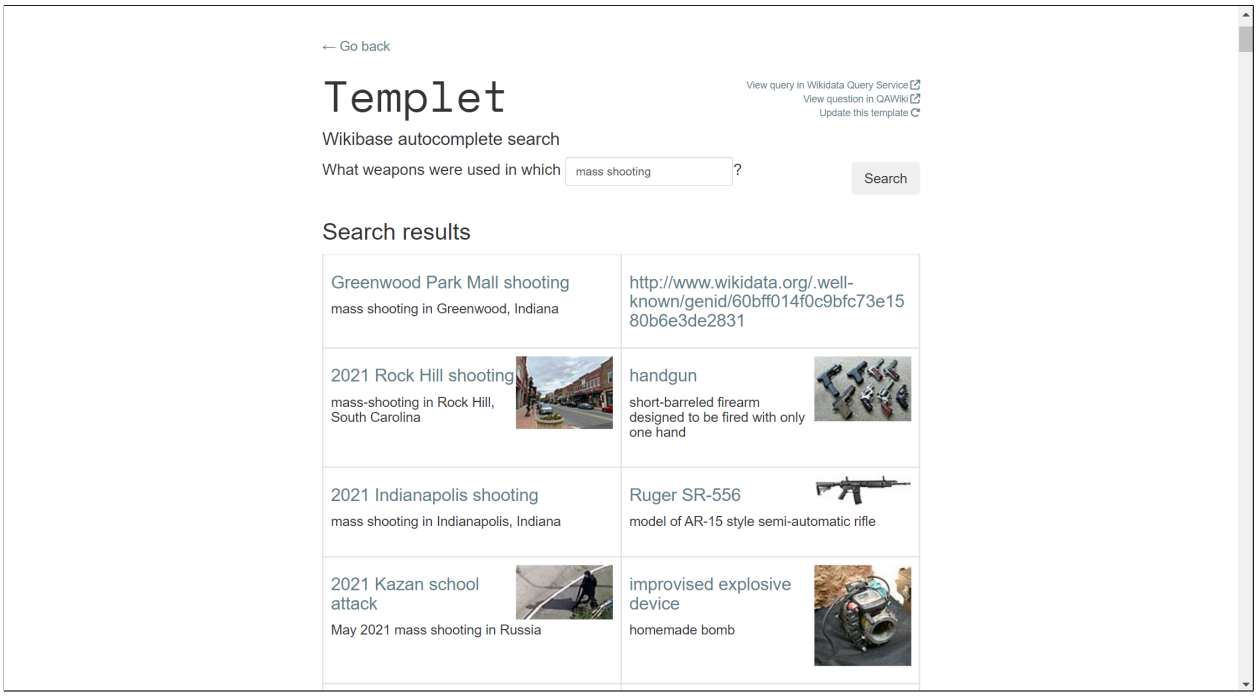

Figura 4.9: Templet: despliegue de resultados tipo tabla en consulta generada a partir de plantilla `Q297`

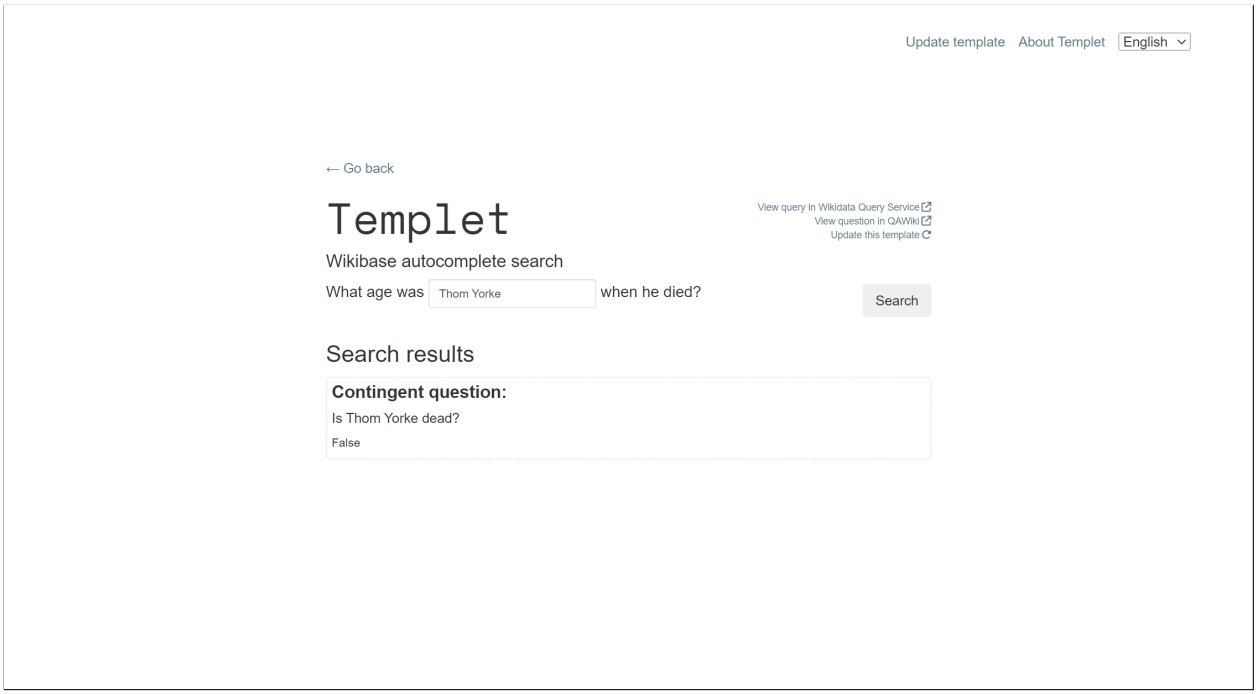

Figura 4.10: Templet: despliegue de resultados de pregunta contingente cuando no se tiene el resultado esperado, a partir de plantilla `Q65`

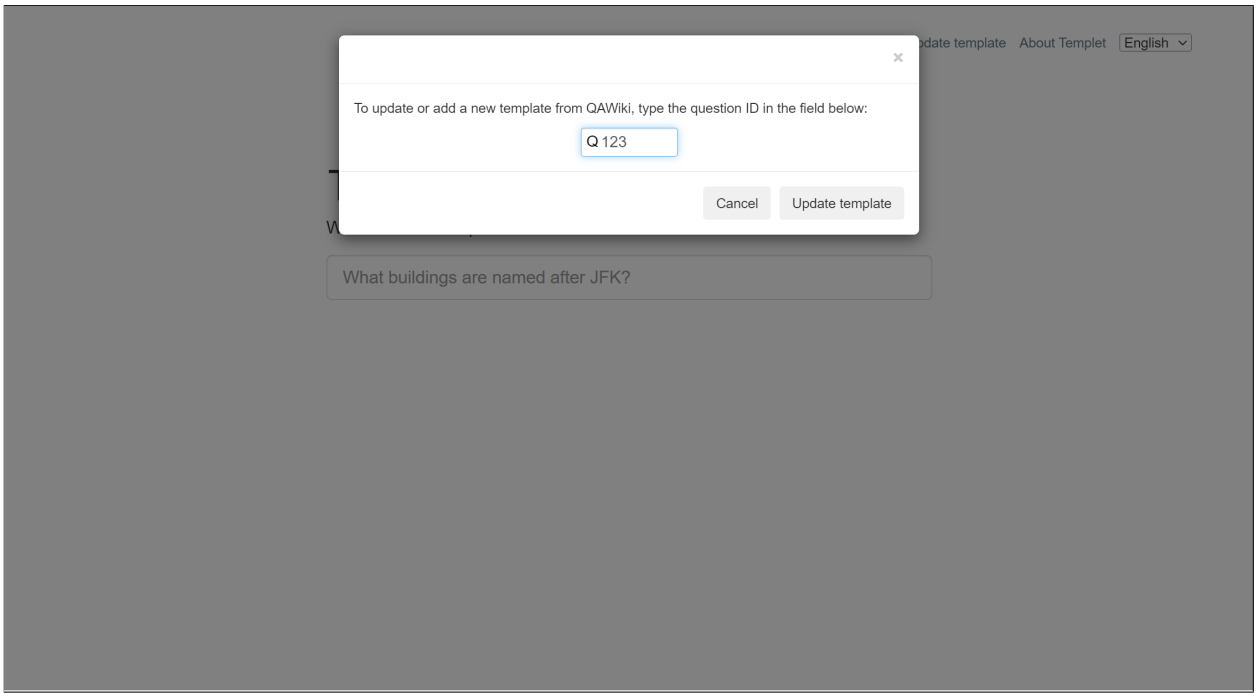

Figura 4.11: Templet: actualización exitosa de plantilla.

4.2.2. Sistema multilingüe

Para lograr una plataforma web extensible en idiomas, todos los textos de etiquetas y mensajes se obtienen a partir de *archivos de configuración*, que son archivos JSON con una estructura predefinida. Se trata de una lista de objetos, en donde la llave es la `id` del elemento HTML del que se desea cambiar el idioma. Cada una de estas llaves contiene un objeto, el cual debe tener como llaves las abreviaturas de los idiomas que se desean configurar. Si el elemento HTML es componente principal de la vista, es decir, no se genera dinámicamente desde Javascript, también debe incluir la clase `lang`, pues esto permite cambiar los valores del texto al cambiar de opción en el selector de idioma, sin necesidad de recargar la página. En el Código 3 se tiene un extracto de `lang.json`, que determina las etiquetas de los elementos principales de la vista.

Además, como se mencionó en la subsección Backend de plataforma web, se tiene una *cookie* para guardar el último idioma seleccionado.

```
1  [ (...)
2     "wikidata-query": {
3         "en": "Wikidata query",
4         "es": "Consulta en Wikidata"
5     },
6     "cont-question-title": {
7         "en": "Contingent question:",
8         "es": "Pregunta contingente:"
9     },
10    "qawiki-link": {
```

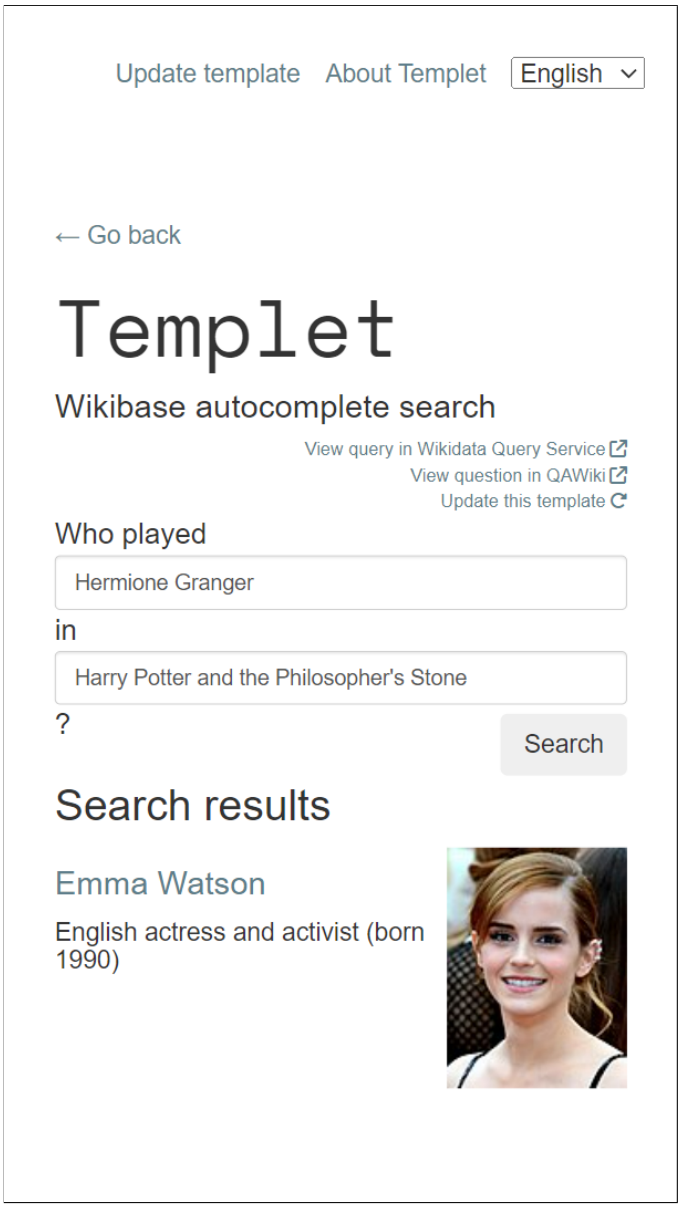

Figura 4.12: Templet: despliegue de resultados de pregunta a partir de plantilla `Q11` en un dispositivo móvil.

```
        "en": "View question in QAWiki",
        "es": "Ver pregunta en QAWiki"
    },
    "wikidata-query-link": {
        "en": "View query in Wikidata Query Service",
        "es": "Ver consulta en Servicio de Consultas de Wikidata"
    }
]
```

Código 3: Extracto de archivo de configuración de etiquetas de Templet

4.2.3. Estructura del código

Como se observa en la Figura A, se tiene un único *template* HTML llamado `index.html` y una única hoja de estilos llamada `index.css`. Además, se distribuyeron las funciones de Javascript en ocho archivos, cada uno con un propósito en particular:

- `autocomplete.js` contiene las funciones que pueblan con autocompletado el campo principal y los campos de entrada de las plantillas.
- `resultsParser.js` parsea los datos en formato JSON retornados por el backend a elementos visualmente amigables con el usuario. Maneja distintos casos: resultados tipo entidad/URI, literales (texto crudo) y booleanos.
- `index.js` carga asíncronamente los archivos JSON de las plantillas y los mensajes de error. Además controla el flujo necesario para desplegar los resultados de una consulta.
- `navigation.js` se encarga del despliegue de los distintos elementos según las interacciones del usuario.
- `lang.js` maneja los idiomas de las etiquetas de los elementos HTML. Actualiza estos valores sin recargar la página cuando el usuario selecciona un idioma distinto y envía una petición al backend para asignar el nuevo valor a la *cookie* que controla el idioma.
- `templates.js` genera el formulario correspondiente a la plantilla seleccionada por el usuario y la consulta SPARQL a partir de las entradas de este formulario.
- `validation.js` se encarga de las validaciones de los campos generados a partir de las plantillas, para evitar enviar consultas erróneas en la API del Servicio de Consultas de MediaWiki.
- `variables.js` contiene las variables globales, como por ejemplo, las URL de QAWiki y Wikidata.
- `js.cookie.js` es un script de Javascript Cookie[6], una API de JavaScript simple y liviana para manejar cookies.

De acuerdo a los detalles descritos en esta sección, se tiene un sistema multilingüe completamente funcional implantado en un servidor. Actualiza las plantillas de forma autónoma y es fácilmente configurable a través de archivos JSON.

[6]https://github.com/js-cookie/js-cookie

Capítulo 5

Evaluación del sistema

La evaluación del sistema tuvo dos etapas, una relacionada a mediciones indicadoras del rendimiento, y otra dedicada a la usabilidad. Estas evaluaciones buscan resolver las siguientes interrogantes: ¿cómo es el rendimiento del sistema en términos del autocompletado? y ¿cómo es la usabilidad del sistema?.

5.1. Rendimiento

La evaluación del rendimiento de Templet se llevó a cabo haciendo múltiples mediciones a tres aspectos del sistema: el tiempo de respuesta del autocompletado principal, el tiempo de respuesta del autocompletado de entidades y el tiempo de demora en el proceso de generación de plantillas.

En cuanto a el tiempo de respuesta del autocompletado principal, se tomó una muestra de 75 mediciones locales, resultando un promedio de 328 milisegundos, con una máxima de 773 milisegundos. El tiempo de respuesta del autocompletado de la entrada principal puede variar entre dispositivos dependiendo de sus recursos, pues esta tarea se hace completamente en el frontend, ya que las plantillas se *cachearon* en un archivo JSON que es consumido directamente por este. Sin embargo, los resultados obtenidos son positivos, y en la navegación, el autocompletado del campo principal es instantáneo.

El tiempo de respuesta del autocompletado de las entidades se calculó gracias a los datos recopilados por el *logger* en el sistema implantado. Estos datos consisten en registros de la navegación de múltiples usuarios reales que utilizaron Templet mientras se recolectaban respuestas para la encuesta. Para este tiempo de respuesta se recopilaron 2.328 datos, resultando un promedio de 568 milisegundos y una máxima de 2,46 segundos. Estos valores permiten que la navegación de los usuarios sea fluida y las respuestas se retornen sin demoras perceptibles. Además, es importante notar que este tiempo depende completamente de lo que demore el servidor de Wikidata en retornar una respuesta.

Para la medición del tiempo de generación de plantillas también se hizo uso de los datos recopilados por el *logger*. Se obtuvo una muestra de 158 actualizaciones totales de plantillas,

lo que corresponde a aproximadamente 7 días, puesto que está programada la ejecución de la actualización de plantillas cada 1 hora. Al momento de esta evaluación, en QAWiki se almacenaban alrededor de 370 preguntas, y el sistema tomó, en promedio, 126 segundos en escanear y procesar todas las preguntas para generar las plantillas. Este tiempo resulta aceptable, además de que el procesamiento de plantillas no interfirió con la usabilidad de la plataforma web.

5.2. Usabilidad

La evaluación de la usabilidad del sistema se hizo a partir de la experiencia de los usuarios. Se hizo una encuesta en Google Forms, compuesta de tres secciones, descritas a continuación:

- Sección 1: Navegación guiada por Templet

  En esta sección se les pidió a los usuarios realizar tres pequeñas tareas dentro de la plataforma, anotando las respuestas y observaciones encontradas. Las tareas eran las siguientes:

  - Buscar cuándo murió Amy Winehouse.
  - Buscar quién interpretó al personaje de Hermione Granger en Harry Potter y la piedra filosofal.
  - Buscar qué colecciones fuera de Chile exponen moái.

  Todas estas tareas referencian preguntas que se encuentran disponibles en el sistema.

- Sección 2: Encuesta SUS

  En esta sección se solicitó navegar sin guía dentro de Templet. Luego, para medir la usabilidad se pidió completar un Sistema de Escalas de Usabilidad (SUS por sus siglas en inglés). Consiste en diez enunciados que pueden ser puntuados de 1 a 5, en donde 1 es *Totalmente en desacuerdo* y 5 es *Totalmente de acuerdo*. Permite evaluar la eficacia (pueden los usuarios lograr con éxito sus objetivos), eficiencia (cuánto esfuerzo y recursos se gastan para lograr esos objetivos) y satisfacción (fue satisfactoria la experiencia).

- Sección 3: Comentarios generales sobre Templet

  En esta última sección opcional, se pidió escribir todos los comentarios que hayan aparecido durante la navegación dentro del sistema.

Esta encuesta se hizo en inglés para la comunidad de Wikidata, y en español para los alumnos del ramo *La Web de Datos*, la comunidad del foro del Departamento de Computación, y profesores del Departamento de Computación. En inglés se obtuvieron 4 respuestas y en español 32.

Respecto a la primera sección, fue útil para detectar y solucionar problemas que los usuarios tenían al buscar preguntas en el sistema. El principal y más comentado fue que

intuitivamente escribían la pregunta completa, es decir, incluyendo la mención (por ejemplo, en la primera pregunta se escribía directamente *cuándo murió Amy Winehouse*). Esto provocaba que no retornara plantillas, pues no existían para tales entidades, por lo que se añadió texto de ayuda y se determinó que de no haber plantillas coincidentes con el texto de entrada, se vayan restando palabras de esta entrada hasta coincidir con alguna plantilla.

En cuanto a la segunda sección, se inglés se obtuvo un puntaje SUS de 61,88. En español se logró un puntaje SUS de 76,72. El promedio entre ambos puntajes (ponderando la cantidad de participantes) es de 75,07. En la Figura 5.1 se tiene un diagrama de caja de las 36 respuestas donde los marcadores $\times$ indican el valor promedio. Las etiquetas del eje X tienen los siguientes significados:

(a) Creo que me gustaría usar este sistema con frecuencia.

(b) Encontré el sistema innecesariamente complejo.

(c) Me pareció que el sistema era fácil de usar.

(d) Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema.

(e) Encontré que las diversas funciones en este sistema estaban bien integradas.

(f) Pensé que había demasiada inconsistencia en este sistema.

(g) Me imagino que la mayoría de la gente aprendería a usar este sistema muy rápido.

(h) Encontré el sistema muy engorroso de usar.

(i) Me sentí muy confiado usando el sistema.

(j) Necesitaba aprender muchas cosas antes de poder utilizar este sistema.

Los resultados de los enunciados de connotación negativa (b, d, f, h y j) se normalizaron, por lo que el valor 1 indica un resultado negativo y 5 positivo, tal como en los otros enunciados (a, c, e, g e i).

Sauro[1], a partir de estudios realizados en más de 500 sistemas, concluye que el puntaje promedio es 68. Un resultado menor a esta cifra indica que hay varios aspectos a corregir. Entonces, aplicando esta información a los resultados del sistema, tenemos que en la comunidad Wikidata, el sistema no tuvo un buen puntaje, aunque hubo pocas respuestas. Sin embargo, en la encuesta en español y en el promedio general, tuvo una buena recepción.

Es importante notar que aquellos usuarios que dejaron los puntajes más bajos, tuvieron problemas con un *bug* que no fue detectado y solucionado hasta casi la finalización del periodo de la encuesta. Este *bug* desencadenaba múltiples peticiones a la API de Wikidata en la búsqueda de entidades, bloqueando la recepción de los resultados. Esto hacía parecer que nada sucedía cuando el usuario escribía en los campos para buscar las entidades, por lo que el sistema no funcionaba correctamente y no lograba el objetivo de hacer una consulta a través del lenguaje natural. Se ha resuelto este bug ahora.

[1]Measuring Usability with the System Usability Scale (SUS)

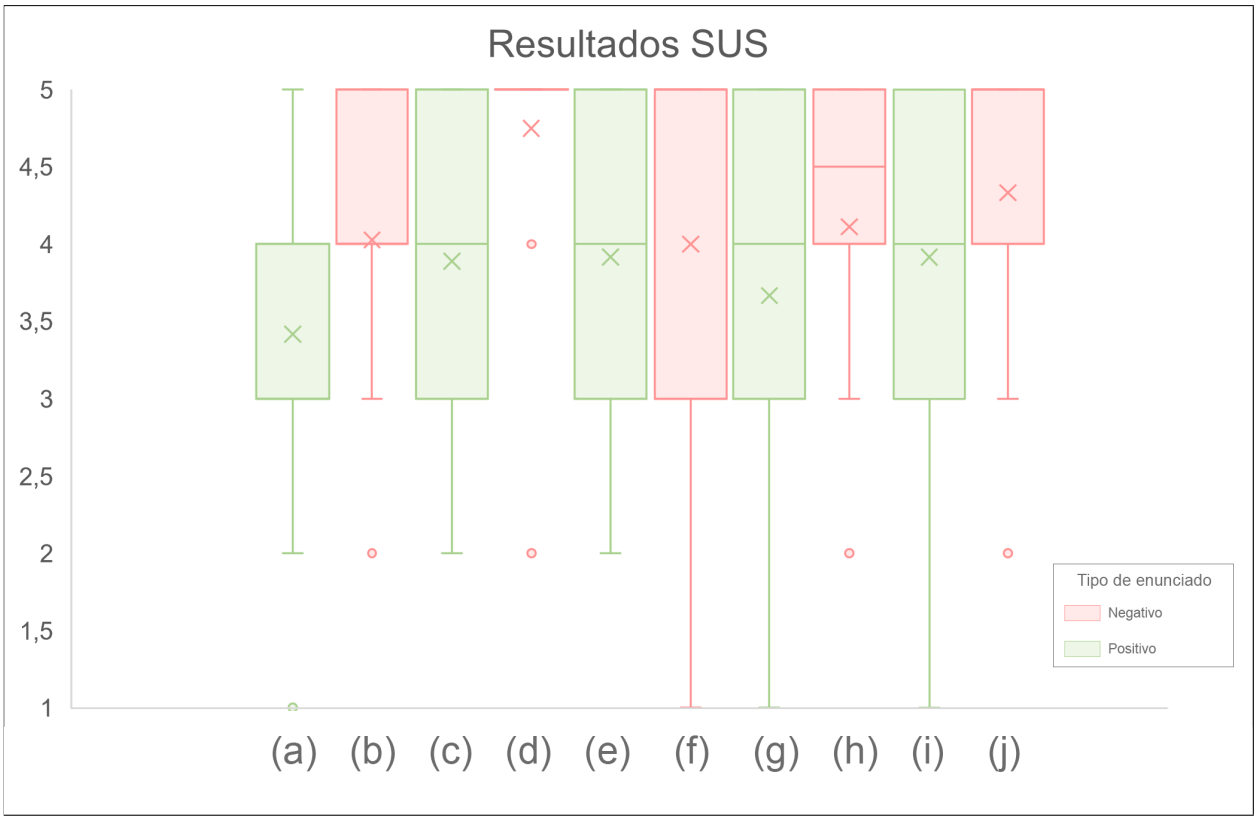

Figura 5.1: Resultados normalizados de SUS.

En la última sección, en general los comentarios son positivos, destacando el detalle mencionado en los resultados de la sección 1, que trataba sobre que escribían directamente la pregunta completa, con las menciones incluidas, y lo ideal que resultaría que este fuera el funcionamiento por defecto de Templet. Esto fue ampliamente discutido durante el desarrollo para encontrar una solución, pero para lograr identificar automáticamente las menciones desde una pregunta completa (por ejemplo, si se tiene la entrada *quién interpretó al personaje de Hermione Granger en Harry Potter y la piedra filosofal*, se desea separar *Hermione Granger* y *Harry Potter y la piedra filosofal*), se hace necesario el uso de aprendizaje de máquina, lo que se escapa del alcance de este trabajo.

Capítulo 6

Conclusión

6.1. Resumen

El sistema desarrollado en esta memoria consistió en una aplicación web basada en la búsqueda por autocompletado. Este sistema, a partir de un constante escaneo a las preguntas almacenadas en QAWiki, disponibiliza plantillas creadas a partir de esta colección de preguntas, las cuales contienen ranuras o *slots* para ser reemplazadas por nuevas entidades de Wikidata. De esta forma, permite consultar nuevas preguntas en lenguaje natural sobre los datos enlazados, mapeando las entradas del usuario en la plantilla de la pregunta y consulta SPARQL.

Templet se encuentra en producción en un servidor del DCC, en la URL *templet.dcc.uchile.cl*. Se logró realizar diversas pruebas al sistema dentro de este entorno final para su mejoramiento. También se implementaron varias sugerencias de los usuarios que evaluaron Templet, las que fueron muy importantes para el mejoramiento del diseño y flujo del sistema.

El código se ha dejado público en el repositorio **github.com/franpss/qa-autocomplete**. De esta forma, otros usuarios pueden contribuir a la mejora del sistema.

6.2. Objetivos

Para recapitular, los objetivos planteados para este sistema fueron los siguientes:

Objetivo General

El objetivo general corresponde a que un usuario común logre generar consultas SPARQL en Wikidata a través de preguntas en lenguaje natural, predefinidas a través de plantillas o *templates* que indican la estructura de una pregunta. Además, el sistema debe sugerir plantillas o entidades mediante autocompletado.

Objetivos Específicos

1. Generalizar plantillas a partir del repositorio de preguntas.
2. Sugerir plantillas según autocompletado.
3. Sugerir entidades para completar las plantillas, basándose en el autocompletado.
4. Considerar casos de preguntas dependientes y de desambiguación: algunas consultas pueden requerir de la respuesta de otra pregunta para ser respondidas.
5. Implementar un *front-end* capaz de funcionar con autocompletado de forma eficiente y amigable con el usuario.
6. Conectar el sistema desarrollado a esta interfaz y a los servicios proveídos por Wikidata y QAWiki.

El objetivo general se logró, pues los usuarios pueden completar todas las tareas descritas en este. Los alcances logrados en los objetivos específicos se muestran a continuación:

En cuanto al primer objetivo, Templet es capaz de crear plantillas de forma autónoma y programada a partir del repositorio de preguntas QAWiki. Además, permite que el usuario modifique cuando necesite la biblioteca de plantillas disponibles mediante la web, ingresando el QID de la plantilla que desea añadir o actualizar. De esta forma, no necesita esperar el próximo escaneo para que se añada o modifique una pregunta.

Respecto al segundo objetivo, Templet sugiere plantillas a través del autocompletado de forma rápida y efectiva. Para que esta acción fuera eficiente, las plantillas fueron *cacheadas* en un archivo JSON que es consumido directamente por el frontend. De esta forma, no hay problemas en los tiempos de respuesta del sistema.

En cuanto al tercer objetivo, el autocompletado de entidades también funciona de la forma esperada. El tiempo de demora depende directamente del tiempo de respuesta de Wikidata, tomando no más de unos milisegundos.

El cuarto objetivo también fue logrado, pues el sistema considera preguntas contingentes que pueden afectar el resultado de la pregunta principal y despliega correctamente sus resultados. En cuanto a las preguntas de desambiguación, no fue necesario tomar acciones especiales para estas preguntas, ya que en Templet solo se indexan las preguntas no ambiguas (por ejemplo, *¿Cuándo se incorporó Kyra Phillips a ABC?*), y las ambiguas (por ejemplo, *¿Es Nueva York más grande que Pekín?*, pues no especifica si es más grande en términos de área o población) son directamente ignoradas.

El quinto objetivo también se logró, ya que Templet posee un frontend minimalista eficiente y amigable, aunque permite mejoras en el futuro.

Por último, el sexto objetivo también fue logrado, puesto que se desarrolló un backend en Flask que se comunica con los servicios de Wikidata y QAWiki, entregando los resultados limpios y ordenados al frontend.

Con lo anterior, se puede observar que todos los objetivos fueron completados, resultando un sistema funcional, capaz de realizar las tareas para las que fue diseñado.

6.3. Fortalezas y limitaciones

Entre las fortalezas de Templet se tiene la eficiencia del sistema, ya que se trata de una aplicación liviana que no hace uso de demasiados recursos. Además, la plataforma es dinámica y simple para facilitar la navegación de los usuarios. También la extensión de la aplicación no es una tarea compleja, ya que el código se encuentra documentado y separado por el tipo de función que desempeña. En caso de que en el futuro se encuentren disponibles más idiomas en QAWiki, añadir estos idiomas al sistema es una tarea muy simple, puesto que solo se necesita que los idiomas sean agregados a las variables de entorno y a los archivos de configuración (`messages.json` y `lang.json`). La implantación del sistema también es instantánea y sencilla, gracias a que la aplicación se encuentra dentro de un contenedor Docker. Por último, el código se encuentra de forma pública. De esta forma, se facilita la contribución y mejoramiento del sistema.

En cuanto a las debilidades de Templet, se tiene el *paso extra* entre la selección de plantilla y el reemplazo de entidades. Varios usuarios mencionaron en sus comentarios que sería ideal que al ingresar la pregunta en la primera entrada de texto, ya fuera suficiente información para obtener los resultados de la pregunta. El problema yace en que para identificar las entidades dentro de la pregunta, parece ser necesario el uso de inteligencia artificial, lo que escapa al alcance de esta memoria. También se podría usar heurísticas que funcionarían bien en varios casos, pero no se logró definir una función que funcione en todos los casos. Por ejemplo, este reemplazo automático es fácil de lograr en 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 que separa ambas entidades se confunde con el contenido de la primera entidad.

6.4. Trabajo futuro

El usar técnicas y modelos de procesamiento del lenguaje natural (NLP, por sus siglas en inglés) mejoraría enormemente la experiencia del usuario en Templet, pues se solucionaría el problema descrito en las debilidades. También se podría utilizar para crear nuevas plantillas a partir de las existentes, creando mezclas y generalizándolas, enriqueciendo la colección de preguntas disponibles.

La forma en la que se despliegan los resultados para una consulta, también podría ser mejorada. Por ejemplo, actualmente el sistema no enriquece resultados de lugares con coordenadas, sino que son desplegados como texto plano e idealmente podrían mostrarse dentro de un mapa interactivo.

6.5. Conclusiones finales

Como conclusiones finales, se lograron los objetivos y alcances propuestos. Además, la forma en que Templet genera las plantillas y reemplaza las entidades con otras se puede utilizar para extender un conjunto de pares de preguntas y consultas, con el fin de entrenar modelos neuronales.

Luego, si se compara Templet con sistemas neuronales, los usuarios de Templet y QAWiki tienen control sobre la consulta que corresponde a una pregunta. El sistema tiene mayor transparencia con respecto a la interpretación de la pregunta, y permite que los usuarios visualicen por qué se generaron los resultados obtenidos.

Haciendo uso de QAWiki y Templet, comunidades con un interés en común pueden hacer consultas particularmente útiles. Al crear una pregunta que utilicen frecuentemente, pueden consultarla fácilmente a partir del lenguaje natural con distintas entidades útiles.

Finalmente, Templet permite generar interés en QAWiki, que era uno de los propósitos principales del sistema. La combinación de QAWiki y Templet crea una herramienta que puede ser utilizada para múltiples objetivos, y por usuarios que no necesariamente son expertos en lenguajes especializados.

Bibliografía

[1] Abraham Bernstein, Esther Kaufmann, and Christian Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. *15th Workshop on Information Technology and Systems, WITS 2005*, 01 2005.

[2] Philipp Cimiano, Peter Haase, Jörg Heizmann, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases – the case of the orakel system. *Data Knowledge Engineering*, 65(2):325–354, 2008. Including Special Section: 3rd XML Schema and Data Management Workshop (XSDM 2006) – Five selected and extended papers.

[3] Daniel Diomedi and Aidan Hogan. Question answering over knowledge graphs with neural machine translation and entity linking. *CoRR*, abs/2107.02865:1, 2021.

[4] André Freitas, João Gabriel Oliveira, Seán O'Riain, Edward Curry, and João Carlos Pereira da Silva. Treo: Best-effort natural language queries over linked data. In Rafael Muñoz, Andrés Montoyo, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems - 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings*, volume 6716 of *Lecture Notes in Computer Science*, pages 286–289. Springer, 2011.

[5] 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.

[6] Haemin Jung and Wooju Kim. Automated conversion from natural language query to sparql query. *Journal of Intelligent Information Systems*, 55, 12 2020.

[7] Vanessa López, Miriam Fernández, Enrico Motta, and Nico Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2012.

[8] Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. Aqualog: An ontology-driven question answering system for organizational semantic intranets. *SSRN Electronic Journal*, 01 2007.

[9] Denis Lukovnikov, Asja Fischer, Jens Lehmann, and Sören Auer. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, page 1211–1220, Republic and Canton of Geneva, CHE, 2017. International World Wide Web Conferences Steering Committee.

[10] Fabiano Ferreira Luz and Marcelo Finger. Semantic parsing natural language into SPARQL: improving target language representation with neural attention. *CoRR*, abs/1803.04329, 2018.

[11] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[12] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, Andre Valdestilhas, Diego Esteves, and Ciro Baron Neto. SPARQL as a foreign language. In Javier D. Fernández and Sebastian Hellmann, editors, *Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems - SEMANTiCS2017 co-located with the 13th International Conference on Semantic Systems (SEMANTiCS 2017), Amsterdam, The Netherlands, September 11-14, 2017*, volume 2044 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[13] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web*, 04 2012.

[14] Christina Unger and Philipp Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In Rafael Muntilde;oz, Andreacute;s Montoyo, and Elisabeth Meacute;tais, editors, *Natural Language Processing and Information Systems - 16th International Conference on Applications of Natural Language to Information Systems, NLDB 2011, Alicante, Spain, June 28-30, 2011. Proceedings*, volume 6716 of *Lecture Notes in Computer Science*, pages 153–160. Springer, 2011.

[15] Christina Unger, André Freitas, and Philipp Cimiano. An introduction to question answering over linked data. In Manolis Koubarakis, Giorgos B. Stamou, Giorgos Stoilos, Ian Horrocks, Phokion G. Kolaitis, Georg Lausen, and Gerhard Weikum, editors, *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, volume 8714 of *Lecture Notes in Computer Science*. Springer, 2014.

[16] Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

[17] Shujun Wang, Jie Jiao, Yuhan Li, Xiaowang Zhang, and Zhiyong Feng 0002. Answering questions over rdf by neural machine translating. In Kerry L. Taylor, Rafael Gonçalves, Freddy Lécué, and Jun Yan, editors, *Proceedings of the ISWC 2020 Demos and Industry*

Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1-6, 2020 (UTC), volume 2721 of *CEUR Workshop Proceedings*, pages 189–194. CEUR-WS.org, 2020.

[18] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to SPARQL. *Future Gener. Comput. Syst.*, 117:510–519, 2021.

Anexo

```
1
2  SELECT ?sbj ?label ?desc ?thumb
3  WHERE {
4    VALUES ?sbj { {entitiesIds} }
5    OPTIONAL { ?sbj rdfs:label ?labelLang . FILTER(lang(?labelLang)="{lang}") }
6    OPTIONAL { ?sbj rdfs:label ?labelEn . FILTER(lang(?labelEn)="en") }
7    OPTIONAL { ?sbj wdt:P18 ?image . }
8    OPTIONAL { ?sbj schema:description ?desc . FILTER(lang(?desc)="{lang}") }
9    BIND(REPLACE(wikibase:decodeUri(STR(?image)),
        "http://commons.wikimedia.org/wiki/Special:FilePath/", "") as
        ?fileName) .
10   BIND(REPLACE(?fileName, " ", "_") as ?safeFileName)
11   BIND(MD5(?safeFileName) as ?fileNameMD5) .
12   BIND(CONCAT("https://upload.wikimedia.org/wikipedia/commons/thumb/",
13               SUBSTR(?fileNameMD5, 1, 1), "/",
14               SUBSTR(?fileNameMD5, 1, 2), "/",
15               ?safeFileName, "/100px-", ?safeFileName,
                    IF(regex(?safeFileName, ".svg"), ".png", "")) as ?thumb)
16   BIND(COALESCE(?labelLang,?labelEn) AS ?label)
17 }
```

Código 4: Consulta SPARQL para obtener etiqueta, descripción e imagen redimensionada para una o más entidades en un idioma, en donde `{entitiesIds}` son los QIDs de las entidades a buscar separadas por comas, y `{lang}` el idioma seleccionado.

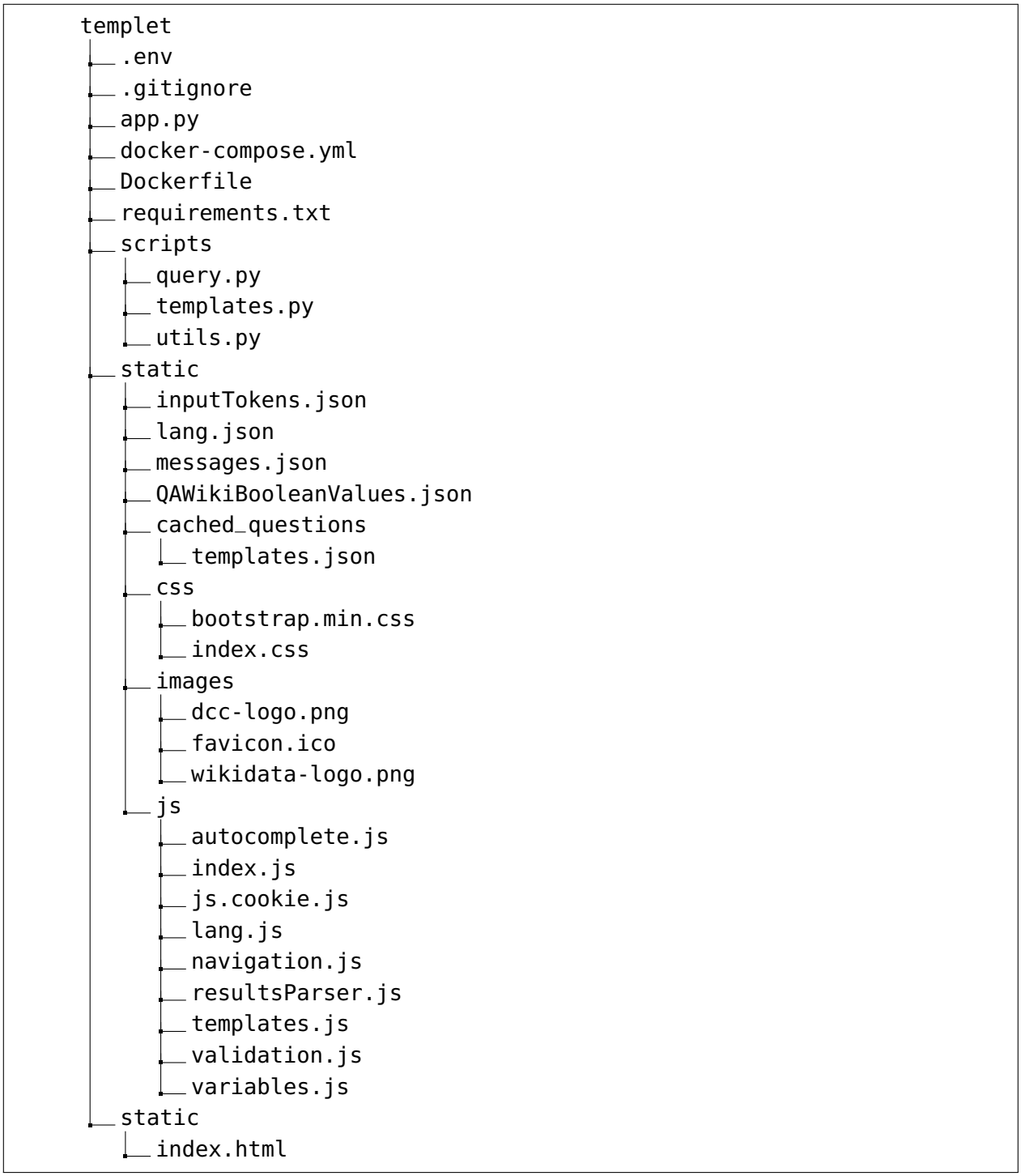

Figura A: Estructura general del proyecto