Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior

Trabajo Fin de Grado

MEDSPELL: CORRECTOR ORTOGRÁFICO MÉDICO

Alumno:Ana Belén Parras Portillo

Tutor: Prof. D. Manuel Carlos Díaz Galiano
Tutora: Prof. D.ª Pilar López Úbeda
Dpto: Informática

Septiembre, 2020

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don MANUEL CARLOS DÍAZ GALIANO y Doña PILAR LÓPEZ ÚBEDA , tutores del Proyecto Fin de Carrera titulado: MEDSPELL: CORRECTOR ORTOGRÁFICO MÉDICO, que presenta ANA BELÉN PARRAS PORTILLO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, SEPTIEMBRE de 2020

El alumno:

PARRAS PORTILLO ANA BELEN - 26037975C
Firmado digitalmente por PARRAS PORTILLO ANA BELEN - 26037975C
Fecha: 2020.09.07 21:35:02 +02'00'

ANA BELÉN PARRAS PORTILLO

Los tutores:

Firmado digitalmente por DIAZ GALIANO MANUEL CARLOS - 26032999N
Fecha: 2020.09.08 09:33:23 +02'00'

Firmado por LOPEZ UBEDA PILAR - 15451728Y el día 08/09/2020 con un certificado emitido por AC FNMT Usuarios

MANUEL CARLOS DÍAZ GALIANO
PILAR LÓPEZ ÚBEDA

Índice

Índice de ilustraciones

Índice de tablas

Índice de fórmulas

Resumen

A lo largo de este trabajo se elabora un corrector ortográfico médico basado en el sistema sanitario español. Este corrector está contextualizado en la situación en la que el facultativo elabora un informe médico. Después de su elaboración, comprueba en el diccionario, creado con un corpus médico, los posibles errores ortográficos y devuelve las diferentes alternativas para cada palabra errónea detectada junto con el porcentaje de concordancia existente con la palabra original. Del mismo modo, y en el caso de informes extensos, se ha elaborado el corrector con la funcionalidad de que se pueda consultar la ortografía en dicho informe en cualquier momento, sin que esté finalizado.

Para su desarrollo, se han aplicado técnicas de PLN (Procesamiento del Lenguaje Natural) para procesar los documentos. Además, se ha realizado un estudio de los lenguajes de programación adecuados para PLN y distintos algoritmos relativos a la obtención de la similitud entre palabras. Del mismo modo, se ha procedido a la investigación en las tecnologías web. Por último, se han estudiado los frameworks que permiten crear APIs (Application Programming Interface) y aplicaciones web.

Palabras claves

Corrector médico automático; Errores ortográficos médicos; API; Python; Medidas de similitud; JSON; PLN; Flask; Metodología ágil;

Abstract

Throughout this work, a medical spelling checker based on the Spanish health system is developed. This corrector is contextualized in the situation in which the physician prepares a medical report. After it has been prepared, it checks the dictionary, created with a medical corpus, for possible spelling errors and returns the different alternatives for each erroneous word detected together with the percentage of agreement with the original word. In the same way, and in the case of extensive reports, the corrector has been created with the functionality that the spelling in the report can be consulted at any time, without it being finished.

For its development, NLP (Natural Language Processing) techniques have been applied to process the documents. In addition, a study has been made of the appropriate programming languages for NLP and different algorithms for obtaining word similarity. Similarly, research has been carried out into web technologies. Finally, the frameworks that allow the creation of APIs (Application Programming Interfaces) and web applications are studied.

Keywords

Automatic Medical Corrector; Medical spelling errors; API; Python; Similarity measures; JSON; PLN; Flask; Agile methodology;

1. INTRODUCCIÓN

El objetivo de este proyecto es crear un sistema de detección y corrección de errores ortográficos especializado en términos médicos, permitiendo verificar la ortografía de palabras sanitarias y farmacológicas dando sugerencias y porcentaje de concordancia con las que están escritas erróneamente. La implementación prototípica usa como fuente de información una colección de documentos técnicos facilitados por Dña. Pilar López Úbeda además de otros obtenidos por mí, como se explica en el punto 8.3.2.

El uso más habitual de los correctores ortográficos se asocia con los procesadores de texto, sin embargo, tienen diferentes marcos de actuación. Sistemas de reconocimiento automático de la voz -ya es posible hablar a los dispositivos móviles o los coches nuevos incorporan sistemas de comandos por voz, con lo que no es necesario despegar las manos del volante, o en ciertas profesiones, como en el dictado médico o en laboratorios donde la transcripción ahorra mucho tiempo- o sistemas de reconocimiento óptico de caracteres, en los que se hace necesario procesar la salida para verificar la corrección del texto analizado, son algunos ejemplos de esta utilidad.

Se ha llevado a cabo un estudio teórico de diferentes técnicas propuestas para la corrección ortográfica automática y de correctores médicos, con objeto de ofrecer una visión precisa sobre la complejidad del problema.

Por otra parte, sería un error obviar las tecnologías que Internet ofrece y no integrarlas en los desarrollos para obtener nuevas herramientas, más completas y eficaces siempre que sea posible, puesto que intervienen factores como la seguridad o la calidad de las comunicaciones que en muchas ocasiones actúan en detrimento del uso de estas tecnologías.

En esta investigación se ha elaborado una aplicación que se basa en la web que utiliza tecnologías de PLN con el fin de ayudar a los profesionales médicos a confeccionar informes, análisis de laboratorio, expedientes médicos y diagnósticos entre otros.

El esquema que se ha seguido en este proyecto se muestra en la ilustración 1.1. Se divide en dos partes bien diferenciadas. En primer lugar, una API que recibe un texto por parte del usuario final, realiza todo el PLN necesario para la corrección y finalmente devuelve un JSON con el resultado del análisis de dicho texto. En segundo lugar, una interfaz sencilla e intuitiva donde el usuario introduce un texto manualmente. Ésta llama a la API y muestra los errores detectados en el texto y las sugerencias de corrección obtenidas por dicha API.

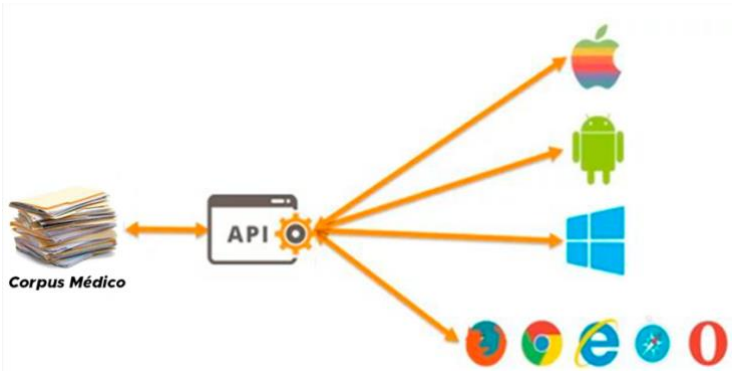

Ilustración 1.1 Esquema del proyecto

1.1. Agradecimientos

Agradezco a todo el personal docente de las diferentes asignaturas que conforman el Grado de Ingeniería Informática y de forma especial a Dª Pilar López Úbeda y D. Manuel Carlos Díaz Galiano como profesores tutores del Trabajo Final de Grado, medSpell: Corrector ortográfico médico, de la Escuela Politécnica Superior

perteneciente a la Universidad de Jaén, por darme la oportunidad de realizar este proyecto y proporcionarme recursos adecuados para el estudio.

1.2. Justificación

Hoy en día los profesionales médicos están obligados a almacenar electrónicamente toda la información relacionada con los pacientes (expedientes médicos, informes, etc). El ritmo de consultas y la falta de tiempo provocan que muchos de ellos no tengan especial cuidado a la hora de redactar dicha información.

Los correctores de ortografía existentes carecen de efectividad en el ámbito biomédico debido a que las características en corpus médico incluyen el uso de terminología demasiado técnica, acrónimo, abreviatura, etc.

Por ello y aprovechando el auge de las nuevas tecnologías, de una manera implícita, se hace necesario el uso de un recurso tecnológico para reducir el tiempo de respuesta de los facultativos.

Respecto a la pertinencia del proyecto elaborado, exponer que, en los últimos meses, es una realidad contrastada que la comunidad sanitaria y concretamente, los médicos de atención primaria, están reivindicando actuaciones por parte del organismo competente de, en nuestro caso, la Comunidad Autónoma de Andalucía en el sentido de dilatar el tiempo de atención al paciente. Se quejan los profesionales de este sector, entre otras cosas, de tener una elevada carga burocrática que les penaliza en la atención a sus pacientes.

En esta línea, cabe destacar que existen asociaciones, como ASD (Asociación de Salud Digital), que están promoviendo la digitalización e innovación en el ámbito de la Salud. Poniendo en contacto a profesionales con el fin de promover la inclusión dentro del ámbito sanitario de herramientas tales como la inteligencia artificial o el 5G de una manera normalizada.

1.3. Estructura del proyecto

El proyecto desarrollado se estructura según los bloques que siguen a continuación:

- **Bloque 1.** Es el bloque actual, correspondiente a la introducción. En él se expone la justificación del proyecto y los agradecimientos.
- **Bloque 2.** En este bloque se ha explicado la motivación.
- **Bloque 3.** Contiene los objetivos generales y específicos del proyecto.
- **Bloque 4.** Corresponde a la planificación del proyecto y los costes asociados que tendrá.
- **Bloque 5.** En este bloque se han explicado los elementos relativos a la ingeniería del software que se desarrollarán utilizando una metodología ágil.
- **Bloque 6.** Este bloque está dedicado a realizar un estudio sobre los distintos correctores ortográficos y correctores médicos. Además, se ha hecho un estudio sobre las métricas de similitud entre dos palabras y los algoritmos más adecuados para el desarrollo de este proyecto.
- **Bloque 7.** En este apartado se define el análisis de requisitos del proyecto, el estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos. Se detallan todos los requerimientos que debe cumplir el proyecto.
- **Bloque 8.** En este bloque se realiza un estudio de todas las tecnologías necesarias para la implementación del proyecto. Siendo necesario lenguajes de programación, librerías, frameworks para el desarrollo de APIs y aplicaciones web, etc.
- **Bloque 9.** En este apartado se recogen las conclusiones que se han obtenido tras la realización del trabajo fin de grado.
- **Bloque 10.** Se enumeran los posibles trabajos futuros y mejoras que se pueden aplicar al proyecto.
- **Bloque 11.** Incluye el manual de usuario donde se explica cómo debe interactuar con la aplicación web y todos los aspectos a tener en cuenta.
- **Anexo.** Finalmente, explica el manual de instalación del sistema, así como de todos sus componentes.
- **Bibliografía.** Bloque donde se han recogido las referencias del material consultado para la documentación de este proyecto.

2. MOTIVACIÓN

He tomado como punto de partida el impacto de las Tecnologías de la Información y la Comunicación como herramienta fundamental para el desarrollo de nuestra sociedad. El ámbito sanitario es uno de los pilares básicos de lo que hemos denominado tradicionalmente “la sociedad del bienestar”. Por tanto, todo lo que hagamos para avanzar en este marco de actuación, irá en beneficio de nuestra sociedad.

Es por esto que, como miembro activo de la comunidad científica, es un privilegio y una obligación contribuir en la medida de lo posible al desarrollo tecnológico de, en este caso, la comunidad sanitaria.

3. OBJETIVOS DEL PROYECTO

En este apartado se exponen los objetivos que se pretenden conseguir con el desarrollo de este proyecto, siendo el principal objetivo el desarrollo de un software capaz de detectar errores ortográficos y sugerir una lista de posibles soluciones.

3.1. Objetivos generales

- Estudio previo de las tecnologías existentes y necesarias.
- Estudio bibliográfico de correctores ortográficos y correctores médicos actuales.
- Estudio comparativo de los frameworks de desarrollo de aplicaciones web justificando la tecnología elegida.
- Análisis y diseño de una API.
- Análisis y diseño de una plataforma web.
- Generación de la documentación y pruebas del sistema completo.
- Elaboración de manuales de instalación y uso.
- Desarrollo de la memoria del trabajo realizado.

3.2. Objetivos específicos

- Crear un corpus de términos relacionados con la medicina a partir de una colección de documentos médicos contextualizado en el sistema de salud español.

- Realizar un estudio comparativo de los diferentes sistemas de corrección ortográfica existentes.
- Realizar un estudio comparativo de los diferentes correctores médicos existentes.
- Estudiar las distintas funciones de similitud que existen actualmente para la corrección ortográfica de un texto.
- Diseñar y desarrollar una API con las distintas funcionalidades que ofertará el prototipo.
- Diseñar y crear una plataforma web que muestre los errores detectados en un texto biomédico, utilizando la API anteriormente diseñada, incluyendo información adicional sobre posibles soluciones.
- Redactar una memoria que recoja todo el trabajo desarrollado, así como los manuales de instalación y de usuario.

4. PLANIFICACIÓN DEL PROYECTO

Para el desarrollo del proyecto se ajustan estimaciones razonables de recursos, costos, así como una planificación temporal. En este apartado se exponen las diferentes tareas que se van a realizar, el intervalo de tiempo en el que se van a desarrollar y su duración en días.

4.1. Definición de tareas

Las distintas tareas que conforman el desarrollo de este proyecto son:

- Estudio de correctores ortográficos y correctores médicos del lenguaje español.
- Estudio de las actuales funciones de similitud para determinar las sugerencias a una palabra errónea.
- Estudio de las tecnologías existentes para la implementación de una API y una aplicación web.
- Generación del corpus médico con toda la terminología técnica necesaria.
- Diseño e implementación de la API.
- Construcción de la aplicación web.
- Testeo para la experiencia de usuario, calidad, estabilidad y seguridad.

- Desarrollo de los manuales de usuario y de instalación.
- Generación de la memoria.

4.2. Estimación de tiempos

En la tabla 4.1, se especifica la planificación temporal estimada en el desarrollo de este proyecto. La duración estimada es de 113 días, desde la fecha de comienzo el 11 de mayo de 2020 hasta la fecha de fin el 01 de septiembre de 2020.

Tarea	Duración/días	Fecha inicio	Fecha fin
Inicio del proyecto	1	11-05-20	12-05-20
Estudio de diferentes tipos de correctores ortográficos	10	12-05-20	22-05-20
Estudio de diferentes tipos de correctores médicos	10	22-05-20	01-06-20
Estudio de las medidas de similitud	10	01-06-20	11-06-20
Estudio de las tecnologías existentes para la implementación de una API y una aplicación web	4	11-06-20	15-06-20
Construcción del corpus	7	15-06-20	22-06-20
Estudio de las estructuras y arquitecturas para almacenar y gestionar la información	10	22-06-20	02-07-20
Diseño e implementación de la API	28	02-07-20	30-07-20
Diseño e implementación de la aplicación web	16	30-07-20	15-08-20
Realización de pruebas de usuario	7	15-08-20	22-08-20
Desarrollo de los manuales de usuario y de instalación	10	22-08-20	01-09-20
Generación de la memoria	113	11-05-20	01-09-20

Tabla 4.1 Planificación temporal del proyecto

4.3. Diagrama de Gantt

Utilizando el siguiente diagrama de Gantt en la ilustración 4.1, se expone gráficamente el tiempo de dedicación previsto para las diferentes actividades que definen el proyecto.

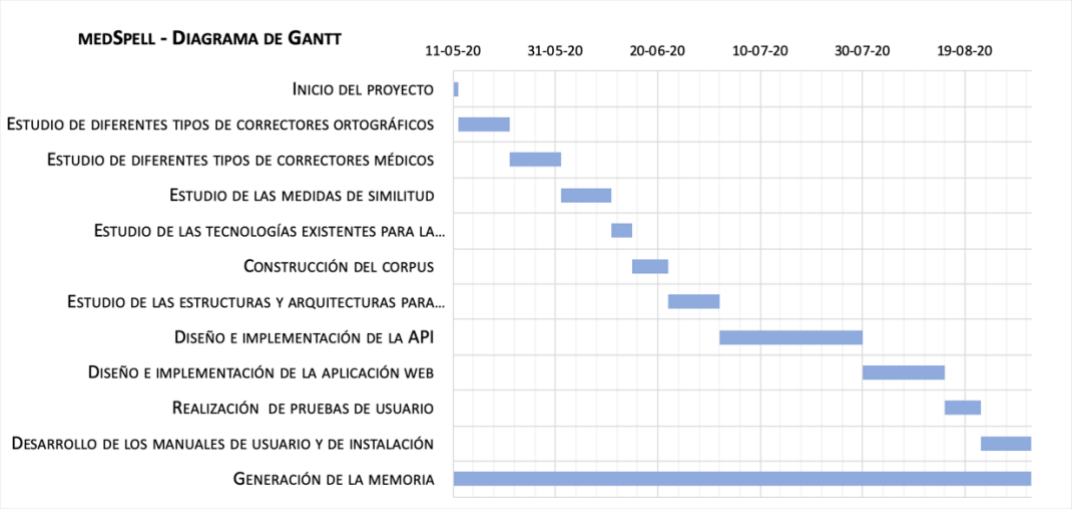

Ilustración 4.1 Diagrama de Gantt

4.4. Análisis de costes

La estimación de coste de este proyecto se realiza analizando los costes relativos a hardware, software y coste del personal.

4.4.1. Hardware

Portátil Macbook 12 inch	1299 €

Debido a los seis años de vida útil que se estima para el ordenador y que tras este período el valor será 0 €, se estima un valor de 216,50 € al año o 18 € al mes. Supuesto que la duración de este proyecto es de aproximadamente 4 meses, su coste total sería de 72 €.

4.4.2. Software

Python: lenguaje de programación usado para la implementación de este proyecto.

Visual Studio Code: editor de programación multiplataforma.

Microsoft Office Word: herramienta de procesamiento de textos usada para la creación de documentación.

Microsoft Office Excel: herramienta de tratamiento de hojas de cálculo, utilizada para la creación de documentación y tablas.

Editores de texto: herramienta para crear y modificar archivos digitales compuestos solamente por texto plano.

Google drive: servicio gratuito para el almacenamiento de archivos online.

GitHub: herramienta para alojar proyectos utilizando el control de versiones Git.

Adobe Photoshop: herramienta de diseño para la creación de ilustraciones aparecidas en la memoria.

Visual Paradigm online: herramienta necesaria para la creación y diseño de diagramas de representación case para UML.

Free DNS Hosting: herramienta para obtener el dominio donde se ha almacenado el proyecto desarrollado.

Trello online: herramienta de gestión de proyecto usada para construir el tablero KanBan que caracterizan a las metodologías ágiles.

S. O. macOS Catalina	0 €
Python	0 €
Visual Studio Code	0 €
Microsoft Office	16,90 €/mes x 4 meses = 67,60 €
Editores de texto	0 €
Google drive	0 €
GitHub	0 €
Adobe Photoshop	30 €/mes x 4 meses = 120 €
Visual Paradigm online	0 €
Free DNS Hosting (servidor)	0 €
Trello online	0 €

4.4.3. Personal

Ya que en este proyecto sólo va a trabajar una persona, se considera que es un analista de sistemas.

El **Analista Programador** es la persona que realiza las funciones de un analista técnico y de un programador; es decir, parte de una información previa recibida del analista funcional, en función de la cual desarrolla las aplicaciones y organiza los datos, tal y como se define en [1].

Según la resolución del convenio estatal de empresas, en el BOE (Boletín Oficial del Estado) a 06 de marzo de 2018, Núm.57, Sec.III, Pág. 26981 y como refleja la tabla salarial del Anexo I para salarios a partir del 31-12-2019, en el área de actividad 3, grupo B, nivel I, en [2], la retribución establecida mensualmente para un analista de

sistemas es de 2.165,55 €. Al ser la duración estimada de este proyecto de 4 meses, el coste total en personal asciende a 8.662,20 €.

4.4.4. Coste total

Se obtiene el coste total sumando los costes hardware, software, de personal y el porcentaje de beneficio. El desglose de este cálculo se muestra en la siguiente tabla.

Coste hardware	72 €
Coste software	187,60 €
Coste de personal	8.662,20 €
Coste total	8.921,80 €
Beneficio estimado: 25 %	2.230,45 €
Coste total del proyecto	**11.152,25 €**

Tabla 4.2 Coste del proyecto MedSpell

5. METODOLOGÍA

Debido a la forma de trabajar de la metodología ágil, la cual queda reflejada en la ilustración 5.1, se ha optado por aplicar dicha metodología en el desarrollo software de este proyecto. Esta metodología se basa en las siguientes premisas y según los principios del manifiesto ágil publicado en [3]:

- Trabajo en base a iteraciones: se desarrolla un primer prototipo lo antes posible, y se va refinando añadiéndole nuevas funcionalidades y/o corrigiendo o modificando lo que vaya siendo necesario.
- Refactorización frecuente del código: refactorizar código es cambiar su estructura interna, pero sin cambiar su comportamiento externo. Así se optimiza su modularidad.
- Flexibilidad: esto implica estar abiertos a cambiar el código tantas veces como sea necesario, tomando las decisiones de diseño en el último momento que no suponga un riesgo (the last responsible moment).

Las metodologías ágiles mejoran la satisfacción del cliente dado que se involucra y compromete a lo largo de todo el proyecto. En cada etapa se informará al cliente de

los logros y progresos del mismo, con la intención de involucrarlo directamente para sumar su experiencia y conocimiento, y así, mejorar las características del producto final [4].

Además, una gestión ágil permite ahorrar tiempo y costes. El desarrollo ágil trabaja de un modo más eficiente y rápido, y con ello, se cumple de forma estricta el presupuesto y los plazos pactados dentro del proyecto. Gracias a la realización de entregas tempranas el cliente tiene rápido acceso a aquellas funcionalidades que aportan valor acelerando el retorno de la inversión.

Se trabaja con mayor velocidad y eficiencia debido a una de sus premisas, trabajar a través de entregas parciales del producto, de este modo, es posible entregar en el menor intervalo de tiempo posible una versión mucho más funcional del producto. Gracias a estas entregas parciales y a la implicación del cliente, será posible eliminar cualquier característica innecesaria del producto, consiguiendo así mejorar su calidad y obteniendo exactamente lo que el cliente busca y necesita [5].

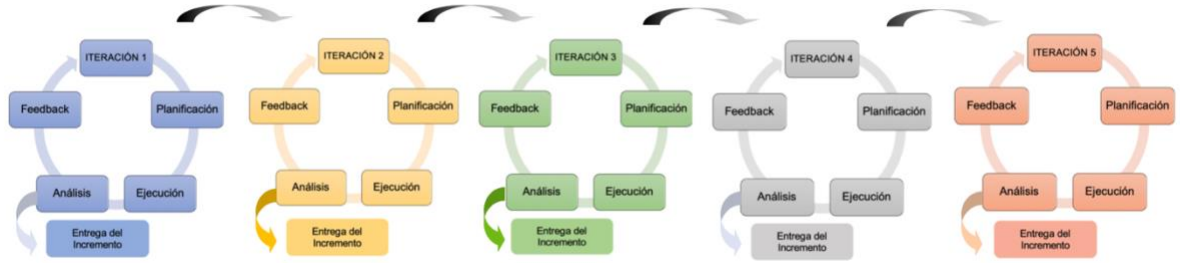

Ilustración 5.1 Ciclos Iterativos de la Metodología Ágil

5.1. Metodología ágil empleada

Siguiendo con lo expuesto en el anterior apartado, al aplicar lo que se denomina desarrollo ágil, existen diversas metodologías y filosofías que se relacionan con este paradigma tal y como refleja la ilustración 5.2 [6, pp. 30-39].

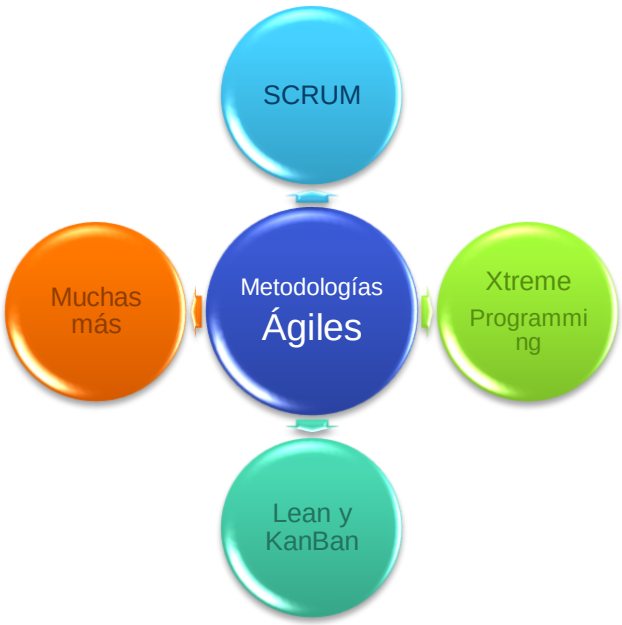

Ilustración 5.2 Metodologías ágiles

Para el escenario que plantea este proyecto se ha elegido LeanKanban, o "tarjetas visuales" puesto que es una técnica de gestión de las tareas muy visual, que permite ver a golpe de vista el estado del proyecto, así como pautar el desarrollo del mismo de manera efectiva. Además, esta metodología gestiona específicamente que se fabriquen sólo los productos necesarios en la cantidad y tiempo, basándose en el desarrollo incremental [7].

En la ilustración 5.3 se detallan los componentes de un tablero Kanban, y en la 5.4 se expone el que se ha seguido para la realización de este proyecto con la herramienta de gestor de tareas Trello.

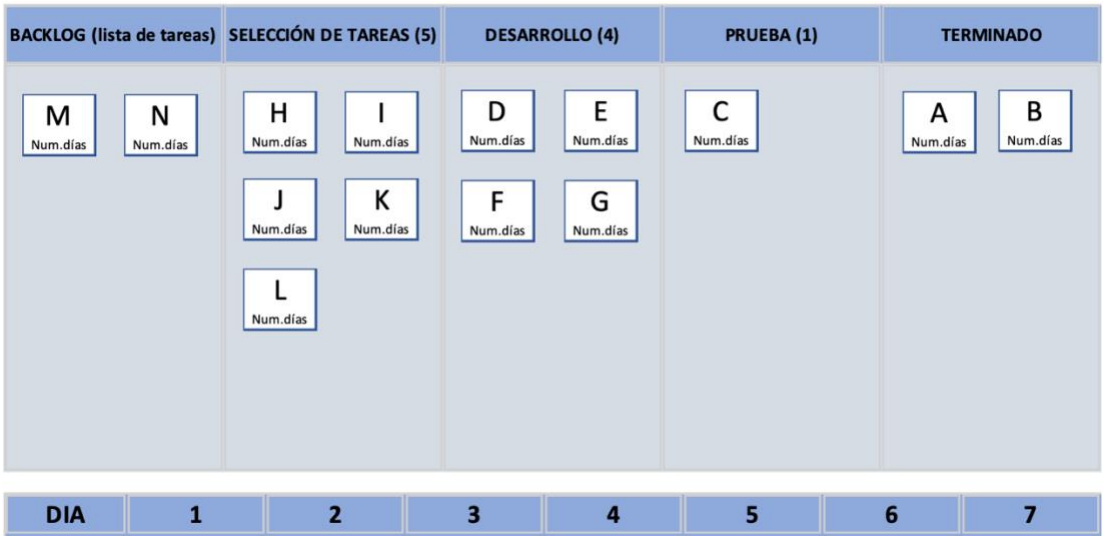

Ilustración 5.3 Tablero KanBan

1. Señales visuales: adhesivos, tickets, etc. donde los equipos escriben sus proyectos y elementos de trabajo.
2. Columnas: el tablero tendrá tantas columnas como estados por los que puede pasar la tarea. Cada columna representa una actividad específica que, en conjunto, conforman un flujo de trabajo. Las tarjetas van moviéndose por el flujo de trabajo hasta que éste termina.
3. Límites del trabajo en curso: en este método, el trabajo en curso debe estar limitado y, por tanto, existe un número máximo de tareas a realizar en cada fase. No se puede abrir una nueva tarea si finalizar otra, “Stop Starting, Start Finishing”, se prioriza el trabajo que está en curso.
4. Punto de compromiso: los equipos de KanBan suelen tener un backlog para su tablero. Es aquí donde los clientes y los compañeros de equipo plantean ideas para proyectos.
5. Punto de entrega: es el final del flujo de trabajo. Generalmente suele ser el momento en el que el producto o servicio está en manos del cliente.
6. Lead time o cycle time: el objetivo consiste en llevar las tarjetas desde el punto de compromiso hasta el punto de entrega cuanto antes. El tiempo transcurrido es lo que se conoce como plazo.

En resumen, se ha optado por utilizar la metodología KanBan por los siguientes motivos:

- Ayuda a registrar documentación clave de una tarea, permitiendo a los equipos tener una clara visión general de todos los elementos de trabajo y controlarlos a través de las diferentes etapas de su flujo de trabajo.
- Se basa en el principio de desarrollo incremental, visualizando las fases del ciclo de producción.
- Revisa los detalles de un vistazo y proporciona una transparencia general de la organización, por lo que es una metodología flexible.
- Minimiza la pérdida de tiempo ya que las iteraciones son cortas y la duración no es fija, agilizando así el flujo de trabajo.
- Organiza y gestiona el trabajo eficientemente.

- No requiere reuniones diarias con el cliente.

A continuación, se detalla el tablero KanBan usado en este proyecto.

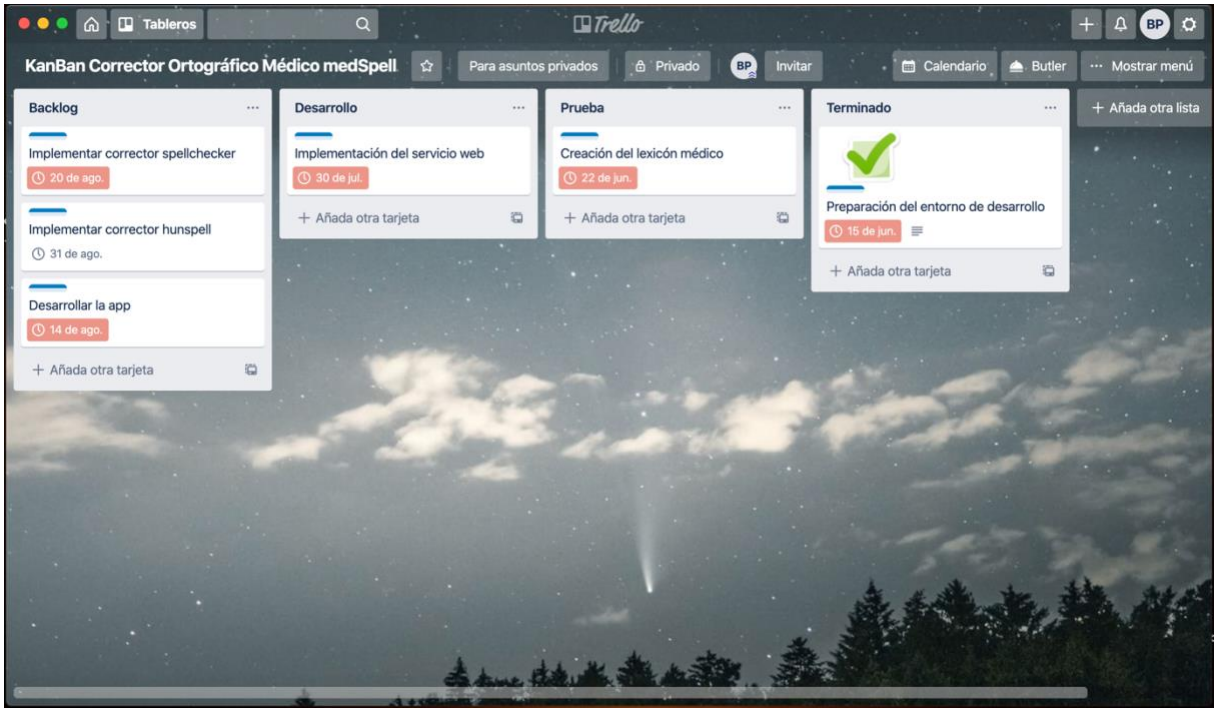

Ilustración 5.4 Tablero KanBan del Corrector Ortográfico Médico MedSpell

6. CONTEXTO Y ESTUDIO PRELIMINAR

El estudio se centra en dos cuestiones principales, la corrección ortográfica y los servicios web.

En los siguientes apartados se explicará, en primer lugar, la complejidad del análisis del lenguaje español con el objetivo de aclarar el enfoque necesario para la implementación de este proyecto. Dicho análisis se centrará en los distintos correctores ortográficos y médicos existentes en español -puesto que el sistema se va a desarrollar para este idioma-, y en los algoritmos de medidas de similitud para su posterior utilización en este proyecto. Y, en segundo lugar, las tecnologías existentes para el desarrollo de una API y de una aplicación web.

6.1. Correctores ortográficos

El enfoque, aparentemente universal, que tienen los correctores ortográficos como herramientas se centra en el uso de un diccionario. El método habitual comienza con la búsqueda de una palabra en el mismo y, si esta es encontrada, se prosigue con

la próxima; en caso contrario, se presenta al usuario una lista de palabras ordenadas de acuerdo a una determinada distancia entre la palabra analizada y las presentes en el diccionario, entre las cuales se espera esté la correcta. Sin embargo, este enfoque no ataca aquellas situaciones -enumeradas en [8]- en que el error se debe a ambigüedades propias del idioma, problemas de paronimia o confusiones entre palabras correctas o a la presencia de numerosas homofonías en la grafía (por ejemplo, halla, haya o allá, c-z, v-b, etc).

Las reglas ortográficas apuntan en esta dirección, resolviendo tales ambigüedades mediante el uso de información lingüística de contexto.

Los algoritmos de corrección tienen como objetivos detectar y corregir, ya sea de forma automática o interactiva, los errores de redacción generados por los humanos; esto con el fin de aumentar la calidad de los textos. En la corrección interactiva el usuario selecciona una palabra de la lista de candidatas, mientras que en la selección automática es el corrector el que selecciona dicha palabra.

La clasificación de los diferentes tipos de errores que Daniel Naber [9] propone son:

- Errores ortográficos -palabras no reconocidas -. Partiendo de una lista de palabras conocidas, se comparan las palabras de un texto con éstas. Si no está en dicha lista se considera incorrecta. Posteriormente se sugerirán palabras similares como alternativas. Por ejemplo, para *coceh* se sugerirán, entre otras, *coche*.
- Errores gramaticales -real-word, palabras en el idioma, pero mal en el contexto- hacen que una oración no cumpla con la gramática del idioma. Supongamos el texto *La perro son mamíferos*. Todas las palabras son correctas desde un punto de vista ortográfico, sin embargo, existen varios errores gramaticales. Este error no puede ser encontrado por un corrector ortográfico, se debe de hacer un análisis morfológico.

Tal y como muestra la tabla 6.1, el primer paso para la detección de errores gramaticales consistirá en verificar la correspondencia de las etiquetas de género y número en palabras adyacentes dividiendo el texto en bigramas.

Seguidamente mediante la aplicación de reglas, generaremos todas las posibilidades de concordancia para los pares. Y finalmente seleccionaremos las

palabras con mayor ocurrencia y tomaremos los bigramas en los que aparecen para construir la frase correcta *El perro es mamífero*.

División en bigramas
La perro error de género
perro son error de número
son mamíferos correcto

Opciones bigrama 1	Opciones bigrama 2	Opciones bigrama 3
La perra	perros son	son mamíferos
El perro	perro es	es mamífero

Palabra	Ocurrencias
La	1
El	1
perra	1
perro	2
perros	1
son	1
es	2
mamíferos	1
mamíferos	1

Tabla 6.1 Pasos para la detección de errores gramaticales

- Errores de estilo -palabras no comunes y estructuras de oraciones complicadas-
- Errores semánticos. “Una oración que contiene información incorrecta que no es un estilo, error gramatical ni un error ortográfico. Por ejemplo, *MySQL es un excelente editor para programar.* Esta oración no es verdadera ni falsa, simplemente no tiene sentido ya que MySQL no es un editor, es una base de datos. Y esto no se puede saber sin un amplio conocimiento del mundo. El conocimiento del mundo también es una forma de contexto, pero está mucho más allá de lo que el software puede entender.”

6.2. Funciones de similitud sobre cadenas de texto

Al trabajar con sistemas de PLN es frecuente encontrar la necesidad de comparar diferentes palabras o frases entre sí o de buscar patrones de caracteres dentro de un texto. El proceso de corrección se inicia con la generación de sugerencias -se podrían obtener ninguna, una o varias sugerencias- generando una lista de posibles palabras correctas procediendo a su corrección seleccionando la más adecuada de forma interactiva o automática. En este caso, es de interés encontrar no solamente las coincidencias exactas entre dos cadenas de texto, sino también tener una medida de aproximación o similitud entre éstas cuando la coincidencia no es

perfecta, siendo de utilidad apoyarse en las medidas de similitud o distancia en cadenas de caracteres, así como para secuencias de palabras o tokens.

Debido a la diversidad de métricas existentes de este tipo, se han estudiado, tal y como se explica en [10] y [11], las más adecuadas para el caso específico que requiere el objetivo principal de este proyecto.

Una métrica es una función matemática que define una distancia entre cada par de elementos de un conjunto ("similitud inversa"). Cuando los elementos comparados son cadenas de texto entonces se dice que la métrica es de cadenas (string metric) [12].

Las propiedades que debe cumplir la función distancia, sean *x, y* y *z* elementos a comparar, son las siguientes:

- No negatividad. Según la fórmula 6.1, el valor de la distancia entre dos elementos cualquiera tiene que ser un número positivo o cero.

$$d(x,y) \geq 0$$

Fórmula 6.1 No negatividad

- Identidad de coincidencia. En caso de que la No negatividad sea cero, la Identidad de coincidencia implica que los elementos son idénticos entre sí y a su vez que la distancia de un elemento consigo mismo o su idéntico siempre será igual a cero, tal y como se expone en la fórmula 6.2.

$$d(x,y) = 0 \leftrightarrow x = y$$

Fórmula 6.2 Identidad de coincidencia

- Simetría. Como vemos en la fórmula 6.3, la distancia entre dos elementos es independiente del sentido o dirección en la que se mida, de *x* a *y* o a la inversa.

$$d(x,y) = d(y,x)$$

Fórmula 6.3 Simetría

- Desigualdad del triángulo. La fórmula 6.4 explica que la desigualdad del triángulo podemos asociarla con la idea de que la línea más corta entre dos puntos (x,y) es la línea recta entre ellas, es decir, si pasamos por un punto *z* en el camino, necesariamente la distancia recorrida se incrementa (si necesitamos desviarnos) o permanece igual (en caso de que no necesitemos desviarnos) pero nunca disminuye.

$$d(x,y) \leq d(x,z) + d(y,z)$$

Fórmula 6.4 Desigualdad del triángulo

Los términos distancia y similitud entre dos cadenas se relacionan como el inverso de la distancia normalizada, lo que indica que a mayor distancia menor es la similitud y viceversa. Dicho concepto queda explícito en la fórmula 6.5.

$$Sim(x,y) = 1 - dnorm(x,y)$$

Fórmula 6.5 Definición de similitud

6.2.1. Distancia de edición

Se definen las métricas en términos de su función de distancia; es decir, lo cerca o lejos se encuentran dos elementos entre sí. En el contexto de comparación de cadenas de caracteres se considera también el concepto distancia de edición.

La distancia de edición entre dos cadenas de caracteres *a* y *b* puede definirse como el número de operaciones básicas que hay que realizar sobre la cadena *a* para transformarla en *b,* es decir, compara todas las permutaciones. Las operaciones básicas son:

1. Inserciones. Consiste en insertar una letra en la cadena.
2. Eliminaciones. Consiste en eliminar una letra de la cadena.
3. Reemplazos/Sustituciones. Consiste en sustituir una letra por otra.
4. Trasposiciones. Consiste en el intercambio de dos caracteres adyacentes con palabras conocidas en una lista de frecuencia de palabras. Es probable que las palabras que se encuentran con más frecuencia en la lista de frecuencias sean los resultados correctos. La tabla 6.2 muestra, como ejemplo, el resultado de realizar dichas operaciones en secuencia a partir de la palabra *pato* para obtener la palabra *caos.*

Texto Origen	Operación	Texto Resultado
pato	Insertar s	pasto
pasto	Eliminar t	paso
paso	Sustituir p por c	caso
caso	Transponer s y o	caos

Tabla 6.2 Ejemplo de secuencia de operaciones de edición para transformar una palabra

6.2.2. Distancia normalizada

Dependiendo de la longitud de las cadenas de texto a comparar se puede encontrar entre ellas mayor o menor número de diferencias. Por lo tanto, no es lo mismo una distancia de edición d = 3 cuando se comparan cadenas pequeñas, por ejemplo, de 5 letras o menos, a cuando se comparan cadenas largas con más de 10 caracteres. Se aclara esta definición con el siguiente ejemplo:

- d(“pato”,”carro”) = 3; Sustituyendo p por c y t por r. Insertando la otra r.
- d(“El pato al patio”,”El pato va al patio”) = 3; Insertando los tres caracteres de la cadena “va_”.

En la primera comparación, una distancia de 3 representa un cambio proporcional importante en el texto, mientras que en la segunda comparación la frase original cambia poco en su totalidad.

Surge así la necesidad de una medida de distancia de edición normalizada, ya que la distancia de edición absoluta no es la mejor opción. La normalización de la distancia de edición tiene como objetivo escalar el valor a un rango común, [0, 1], con lo que los valores cercanos a cero corresponden a cadenas de texto semejantes (con d = 0 para cadenas idénticas) y los valores cercanos a 1 para cadenas de texto muy diferentes (con d = 1 diferencia total entre las cadenas comparadas).

Existen diferentes métodos de normalización, siendo el más simple dividir la distancia obtenida entre el tamaño en caracteres de la cadena más larga comparada. Siguiendo con el ejemplo, dnorm(“pato”,”carro”) = 3/5 = 0.6 y dnorm(“El pato al patio”, ”El pato va al patio”) = 3/19 = 0.16. Donde se aprecia la diferencia relativa entre las cadenas comparadas, 0.6 en la primera es notoriamente mayor a 0.16 de la segunda.

Siguiendo con el ejemplo, se ha calculado su similitud aplicando la fórmula 6.5:

- sim(“pato”,”carro”) = 1 – 0.6 = 0.4
- sim(“El pato al patio”,”El pato va al patio”) = 1 – 0.16 = 0.84

Con lo que la similitud en el segundo ejemplo es mucho mayor que en el primero.

6.2.3. Funciones de similitud basadas en caracteres

Éstas pueden dividirse en dos grupos, las medidas de similitud semánticas y las medidas de similitud léxicas [13].

6.2.3.1. Medidas de similitud semánticas

- Similitud de palabras basada en conocimiento, las cuales son métricas que se basan en encontrar la similitud entre palabras utilizando información derivada de redes semánticas. Wordnet es una gran base de datos léxica en inglés, convirtiéndose en la red semántica más popular [14].
- Similitud entre palabras considerando información semántica, cuyas métricas se calculan con la información obtenida a través de grandes volúmenes de información.

6.2.3.2. Medidas de similitud léxicas

En esta sección se cubren cinco de las más comunes.

6.2.3.2.1. Porcentaje de Subsecuencia común más larga (LCSR)

Propuesta por Melamed (1999), y según la fórmula 6.6, se computa dividiendo la longitud de la subsecuencia común más larga (LCS) por la longitud de la palabra más larga [15]:

$$\text{LCSR}(w_1,w_2) = \frac{|LCS(w1,w2)|}{\max(|w1|,|w2|)}$$

Fórmula 6.6 (LCSR) Porcentaje de Subsecuencia común más larga

Por ejemplo, LCSR(“ejemplo”,”ejamplo”) = $\frac{6}{7}$ (LCS = “e-j-m-p-l-o”)

6.2.3.2.2. Hamming

Tal y como recogen María Antonia Martí y Joaquim Llisterri en [16], es una de las métricas de distancia más simples. Este algoritmo está definido solamente para cadenas que tienen la misma longitud. Así la distancia d(h1,h2) es igual al número de posiciones en los cuales ambas cadenas son diferentes entre sí. Si el algoritmo se aplica a hileras de igual tamaño entonces se utilizan vectores con la misma dimensión.

En caso contrario, se rellena la hilera más pequeña hacia la derecha con espacios vacíos, tal y como se muestra en el ejemplo de la ilustración 6.1

G	R	A	N	D	E
G	R	A	N	O	S
✓	✓	✓	✓	✗	✗

HD(h1,h2) = 2

D	I	S	T	A	N	C	I	A	
D	I	S	T	R	O	S	I	O	N
✓	✓	✓	✓	✗	✗	✗	✓	✗	✗

HD(h1,h2) = 5

Ilustración 6.1 Ejemplo de cálculo de la distancia Hamming

6.2.3.2.3. Levenshtein

Esta distancia puede definirse, según [17], como el número mínimo de operaciones de edición requeridas para convertir una cadena en la otra, considerando como operaciones válidas el borrado, la inserción o la sustitución. Nótese que en caso de cambiar un carácter por el mismo en mayúscula es una sustitución, por lo que habitualmente se suelen convertir las cadenas a minúsculas antes de aplicar el algoritmo.

La forma de calcularla no es trivial como en el caso de la distancia de Hamming. Además, a diferencia de ésta última, la distancia de Levenshtein tradicional es capaz de comparar cadenas de diferente tamaño.

El algoritmo convencional para calcular esta métrica usa programación dinámica, aunque existen diversas adaptaciones y aproximaciones para reducir el costo computacional de su cálculo. En la implementación con programación dinámica, se utiliza una matriz en la que se van almacenando los resultados intermedios, producto de calcular la distancia de Levenshtein, entre los prefijos de las cadenas originales a comparar. El elemento de la esquina inferior derecha de la matriz contiene el valor de dicha distancia [18].

Como ejemplo de funcionamiento de este algoritmo, aplicado a las palabras *cebra y* caballo se ha elaborado la tabla 6.3, observando que su distancia de Levenshtein es 5. Además, en dicha tabla se observa que la distancia de Levenshtein entre dos palabras iguales es 0, como ocurre con las palabras *amigo* y *amigo*.

		a	m	i	g	o
	0	1	2	3	4	5
a	1	0	1	2	3	4
m	2	1	0	1	2	3
i	3	2	1	0	1	2
g	4	3	2	1	0	1
o	5	4	3	2	1	0

		c	a	b	a	l	l	o
	0	1	2	3	4	5	6	7
c	1	0	1	2	3	4	5	6
e	2	1	1	2	3	4	5	6
b	3	2	2	1	2	3	4	5
r	4	3	3	2	2	3	4	5
a	5	4	3	3	2	3	4	5

Tabla 6.3 Ejemplo de cálculo de la distancia Levenshtein

6.2.3.2.4. Jaro-Winkler

Según el artículo de Daniel Rodríguez [19] es una modificación de la distancia de Jaro propuesta en 1990 por Winkler en la que se le da un mayor peso a las cadenas que comparten prefijos. A diferencia de la distancia de Levenshtein el algoritmo de Jaro-Winkler ya ofrece directamente el grado de similitud, es decir, un valor entre 0 y 1, sin necesidad de normalizar posteriormente.

La similitud de Jaro es una métrica en la que la única operación de edición que utiliza es la trasposición de caracteres. Por lo que solamente es necesario contar el número de caracteres iguales entre dos cadenas y el número de transposiciones que son necesarias para llegar de una a otra. Así, matemáticamente, en la fórmula 6.7 se define la distancia de Jaro entre las cadenas de texto a y b obtenida del artículo de Yancey, William E. [20]

$$Ja,b = \begin{cases} 0 & si\ m = 0 \\ \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right) & si\ m \neq 0 \end{cases}$$

Fórmula 6.7 Similitud de Jaro

en donde |a| es la longitud de la cadena a, |b| es la longitud de la cadena b, m es el número de caracteres coincidentes en ambas cadenas y t es la mitad de número de transposiciones necesarios para convertir una cadena en otra. Para asumir que un carácter de la cadena a coincide con uno de la cadena b este ha de ser el mismo y no estar separado más de una distancia igual a la mitad de la cadena más larga menos uno. Esto es, no pueden estar separados más de la distancia.

Por otro lado, las trasposiciones son el número de caracteres que son iguales en ambas cadenas, pero se encuentran en diferente orden. Este valor dividido por dos es el valor que toma t en la expresión anterior.

Con la fórmula 6.8 se expone cómo Winkler propuso modificar la similitud de Jaro, ya que en la mayoría de los casos los prefijos tienen más valor que los sufijos de las cadenas para identificar texto que son similares. Así en la similitud de Jaro-Winkler se introduce una escala que otorga calificaciones más favorables a las cadenas que coinciden desde el principio. Siendo la longitud de prefijo establecida, la métrica se modifica mediante la expresión

$$JWa, b = Ja, b + lp(1 - Ja, b)$$

Fórmula 6.8 Similitud de Jaro-Winkler

donde l es la longitud del prefijo común al comienzo de las cadenas hasta un máximo de 4 y p es un factor de escala entre 0 y 0.25 (usualmente se utiliza p = 0.1) con el que se ajusta la puntuación hacia arriba por tener prefijos comunes.

Si las cadenas a comparar no concuerdan en el primer carácter, entonces la similitud de Jaro-Winkler es simplemente la similitud de Jaro, ya que al ser l=0 el segundo término de la ecuación se convierte en cero.

Con el siguiente ejemplo se ha chequeado el efecto que tiene la posición en la que se encuentran los caracteres diferentes en ambas cadenas.

Js('cargo','carga') = 0.87	JWs('cargo','carga') = 0.92
Js('curva','corva') = 0.87	JWs('curva','corva') = 0.88

Fórmula 6.9 Ejemplo distancia Jaro y Jaro-Winkler

Como se ve en la fórmula 6.9, la similitud de Jaro es la misma al comparar cadenas que difieren en un solo carácter. Si se usa la similitud de Jaro-Winkler el valor cambia dependiendo de la posición del carácter que no coincide, siendo menor cuando éste se encuentra cerca del principio de la cadena.

6.2.3.2.5. N-Gramas

Siguiendo a los autores Iván Amón, Francisco Moreno y Jaime Echeverri [21], un q-gram, también llamado n-gram, es una subcadena de longitud q. Esta función de similitud se basa en que, cuando dos cadenas son muy similares tienen muchos q-grams en común. Es frecuente usar uni-grams (q=1), bi-grams o di-grams (q=2) y tri-grams (q=3). Es posible agregar q-1 ocurrencias de un carácter especial (no definido en el alfabeto Σ original) al principio y final de ambas cadenas. Esto llevará a un puntaje de similitud mayor entre cadenas que compartan algún prefijo o sufijo, aunque presenten diferencias en el medio. Un modelo alternativo se conoce bajo el nombre de k-skip-q-grams que omiten k caracteres adyacentes. Por ejemplo, la cadena “Peter” contiene los bi-grams “Pe”, “et”, “te”, “er” y los 1-skip- 2-grams “Pt”, “ee”, “tr”.

Mediante el uso de funciones hash adecuadas, la similitud de q-grams puede ser calculada eficientemente.

6.2.4. Funciones de Similitud basadas en tokens

A diferencia de las funciones de similitud basadas en caracteres expuestas en el apartado anterior, que calculan la similitud a partir de la proximidad de los caracteres de las palabras, las funciones de similitud basadas en tokens consideran cada cadena como un conjunto de subcadenas delimitadas por caracteres especiales como son comas, puntos y espacios en blanco; es decir, consideran como un conjunto de tokens y calculan la similitud entre cada pareja de ellos mediante alguna función basada en caracteres.

En este apartado se han expuesto dos de las funciones basadas en tokens mas generales.

6.2.4.1. Similitud de Monge-Elkan

En [22] se desarrolla la propuesta de Monge y Elkan de un método simple pero efectivo para medir la similitud entre dos cadenas de texto que contiene varios tokens. Dados dos textos A, B, con |A| y |B| siendo su número respectivo de tokens, y una medida de similitud entre tokens externa la medida Monge-Elkan se calcula según la fórmula 6.10 como

$$Sim_{MongeElkan}(A,B) = \frac{1}{|A|} \sum_{i=1}^{|A|} \max \{sim'(ai, bj)\}_{j=1}^{|B|}$$

Fórmula 6.10 Similitud de Monge-Elkan

Informalmente, esta medida es el promedio de los valores de similitud entre los pares de tokens más similares en ambas cadenas.

6.2.4.2. Similitud coseno TF-IDF

Dadas dos cadenas A y B, sean α_1, α_2 …α_k y β_1, β_2 … β_L sus tokens respectivamente, que pueden verse como dos vectores V_A y V_B de K y L componentes. Cohen, William W. [23] propone una función que mide la similitud entre A y B como el coseno del ángulo que forman sus respectivos vectores.

La similitud coseno TF-IDF no es eficaz bajo la presencia de variaciones a nivel de caracteres, como errores ortográficos o variaciones en el orden de los tokens. Por ejemplo, las cadenas “Pablo Gutiérrez Pamos” y “Pamos Pablo Javier” tendrían similitud cero. En [24] se explica cómo se propone una variante llamada SoftTF-IDF para solucionar este problema, que tiene en cuenta parejas de tokens (α_i, β_j) cuya similitud es mayor que cierto umbral (mediante alguna función de similitud basada en caracteres).

6.3. Servicios web

Un corrector ortográfico podría proveerse como un servicio. La principal ventaja de esta solución es que se puede integrar la corrección ortográfica con cualquier aplicación, o servicio, de forma simple. En este apartado se expone el estudio realizado de las tecnologías existentes para la implementación de una API y de una aplicación web según los requisitos de este proyecto.

Los servicios web exponen APIs que pueden ser accedidas dentro de una red, principalmente Internet, y son ejecutados en el sistema que los aloja. Permiten la intercomunicación entre sistemas de cualquier plataforma y se utilizan en una gran variedad de escenarios de integración.

Un servicio web [25] es una tecnología que utiliza un conjunto de protocolos y estándares, independientes de la plataforma y del lenguaje de programación utilizado para su implementación que sirven para intercambiar datos entre aplicaciones, y que

podemos invocar desde Internet. Es una máquina que atiende las peticiones de los clientes y les envía los recursos solicitados.

6.3.1. Arquitectura orientada a servicios

Internet y la SOA (Service Oriented Architecture o Arquitectura Orientada a Servicios) que se apoya en la orientación a servicios han favorecido los servicios especializados en la solución de problemas específicos y ofrecen nuevas tecnologías para obtener nuevas herramientas más completas y eficaces.

De lo expuesto en [26], SOA es una arquitectura de software cuya funcionalidad aplicativa se brinda a través de componentes denominados servicios, que presentan interfaces estándar bien definidas.

La implantación de una solución basada en SOA no exige implantar servicios web. No obstante, los servicios web son la forma más habitual de implementar SOA.

6.3.2. Estándares empleados

En este proceso intervienen varias tecnologías que posibilitan el intercambio de información:

- **Web Services Protocol Stack:** conjunto de servicios y protocolos de los servicios web.
- **XML** (Extensible Markup Language): formato estándar para los datos que se vayan a intercambiar.
- **SOAP** (Simple Object Access Protocol) o XML-RCP (XML Remote Procedure Call): protocolos sobre los que se establece el intercambio.
- Otros protocolos: los datos en XML también pueden enviarse de una aplicación a otra mediante protocolos normales como **HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP),** o **Simple Mail Transfer Protocol (SMTP)**.
- **WSDL** (Web Services Description Language): es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios web, formato de los mensajes, requisitos del protocolo, forma de comunicación, etc.

- **UDDI** (Universal Description, Discovery and Integration): protocol para publicar la información de los servicios web. Permite comprobar qué servicios web están disponibles.
- **WS-Security** (Web Service Security): protocolo de seguridad aceptado como estándar por **OASIS** (Organization for the Advancement of Structured Information Standards). Garantiza la autenticación de los actors y la confidencialidad de los mensajes enviados.
- **REST** (Representational State Transfer): arquitectura que, haciendo uso del protocolo HTTP, proporciona una API que utiliza cada uno de sus métodos (GET, PUT, POST, DELETE, etc) para poder realizar diferentes operaciones entre la aplicación que ofrece el servicio web y el cliente -recuperar, crear, actualizar y eliminar los recursos-
- **GraphQL,** arquitectura alternativa a REST.

6.3.3. Arquitectura REST

El término REST se originó en el año 2000, en la tesis doctoral de Roy Fielding [27] -uno de los principales autores de la especificación del protocolo HTTP- sobre “Estilos Arquitecturales y el Diseño de Arquitecturas de Software basadas de Red”, quien define REST como un estilo de arquitectura para sistemas hipermedia distribuidos. Si bien el término REST se refería originalmente a un conjunto de principios de arquitectura, actualmente se usa en sentido más amplio para describir cualquier interfaz entre sistemas que utilice el protocolo HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y **JSON**.

La arquitectura REST es una arquitectura cliente-servidor [28] en la que el cliente envía una solicitud al servidor y luego el servidor procesa la solicitud y devuelve las respuestas. Estas solicitudes y respuestas se construyen alrededor de la transferencia de representaciones de recursos. Un recurso es algo que es identificado por URI (Identificador de Recursos Uniforme). La representación de un recurso es típicamente un documento que captura el estado actual o previsto de un recurso.

La siguiente ilustración 6.2 muestra el funcionamiento de un servicio RESTful.

Ilustración 6.2 Funcionamiento de un servicio RESTful

El cliente hace una petición, mediante una URI, al servidor para visualizar (GET) un recurso. El servidor recupera el recurso correspondiente y lo devuelve en el formato indicado (text/xml/json/…) al cliente.

La aplicación web que sigue la arquitectura REST se llama servicio web RESTful.

6.4. Conclusiones

Tras el estudio y dada la extensión de los tipos de errores posibles en el lenguaje español para tener en cuenta a la hora de desarrollar un corrector ortográfico, este proyecto se centra únicamente en el problema de la corrección ortográfica en un sentido estricto, es decir, errores non-word (palabras no reconocidas) o errores de confusión entre letras. El algoritmo de medida de similitud léxica Jaro-Winkler es el que ofrece una concordancia más exacta en el propósito de este proyecto, del corrector ortográfico medSpell.

Los principios de diseño de REST [29, p. 668], direccionabilidad (recursos marcados con URI), apátridas (o carente de nacionalidad legal) e interfaz uniforme y estándar para acceder al resto de los recursos (métodos HTTP) hacen que la aplicación REST sea sencilla y ligera. Además, la diversidad de formatos de datos diferentes que permite, han sido las características por las que se ha optado implementarla en este proyecto.

7. ANÁLISIS DEL PROYECTO

El modelado de este proyecto se ha basado en la metodología UML (Lenguaje de Modelado Unificado) que es una especificación de notación orientada a objetos y compuesto de diferentes diagramas, los cuales representan las diferentes etapas del desarrollo del proyecto [30].

La finalidad es obtener una aplicación capaz de editar un texto de manera cómoda para detectar las palabras que contengan errores ortográficos y aportar al usuario una lista de palabras sugeridas, así como obtener una estadística del porcentaje de concordancia de la palabra sugerida con la palabra errónea original.

7.1. Casos de uso

El caso de uso indica las funcionalidades del sistema desde el punto de vista de la interacción con el usuario, las cuales, normalmente, se corresponden con las funciones declaradas en la especificación de requisitos expuestos en los siguientes apartados.

En este proyecto, la aplicación consta de un solo actor y se trata del usuario de la aplicación. Los usuarios accederán al corrector médico a través de una aplicación web que hará uso del servicio de corrección. El usuario recibirá los resultados (informe de palabras sugeridas y estadísticas sobre el porcentaje de concordancia así como los algoritmos usados para obtener dicho resultado) como respuesta a la petición. El diagrama de casos de uso se muestra en la ilustración 7.1.

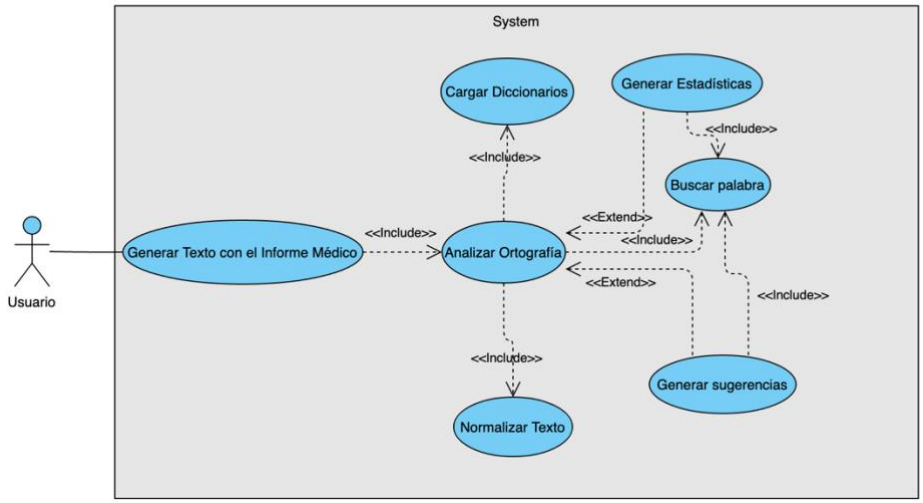

Ilustración 7.1 Diagrama de Casos de Uso

7.2. Diagrama de secuencias

Con la ilustración 7.2 se expone la interacción entre los objetos de este sistema reflejada en el siguiente diagrama de secuencias, el cual modela el caso de uso con los detalles de la implementación del escenario de este proyecto.

Tras escribir el usuario un texto y pulsar el botón analizar, a través de la aplicación web, ésta hace uso del servicio web RESTFul y lo procesa limpiando el HTML (HyperText Markup Language) y eliminando los caracteres raros. A continuación, la API usa la clase MedSpell_Check (contenida en el módulo medspell_analyzer.py) para normalizar el texto pasándolo a minúsculas, eliminando signos de puntuación, descartando palabras muy cortas -sólo analiza palabras de más de 2 caracteres a nivel de tokens-, procesando sólo palabras con signos alfabéticos -isalpha- y obteniendo la lista de tokens con spaCy. A partir de esta lista de tokens y tras una decisión en tiempo de diseño, se ha optado por obtener las palabras erróneas sólo con hunspell (se deshecha hacerlo también con Spellchecker), buscando en su diccionario (al diccionario original se añaden los términos médicos y farmacológicos que ofrecen el corpus médico creado). El siguiente paso es obtener las palabras sugeridas a dichas palabras erróneas extraídas según el corrector ortográfico hunspell. Además, se obtienen las palabras sugeridas (de las palabras incorrectas según hunspell) por el corrector ortográfico Spellchecker (a su diccionario original se le añaden los términos del corpus médico). Seguidamente, la clase MedSpell_Check

calcula el porcentaje de similitud -con la de las palabras sugeridas por hunspell y de las palabras sugeridas por Spellchecker- aplicando la función de similitud léxica basada en caracteres Jaro-Winkler. Finalmente, la API devuelve el JSON resultante del análisis a la aplicación web, la cual presenta los resultados al usuario.

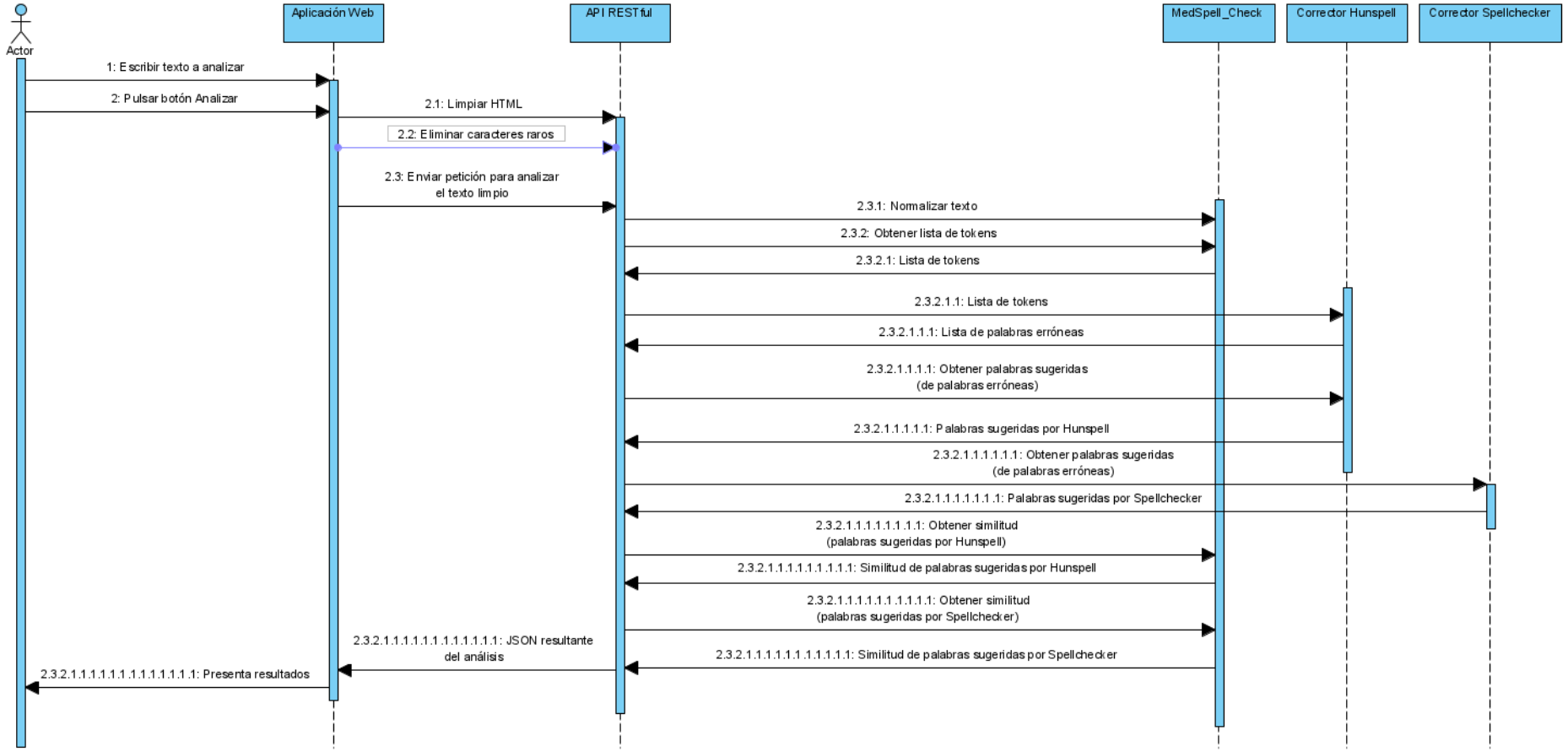

Ilustración 7.2 Diagrama de secuencias

7.3. Análisis de requisitos

Los requisitos del sistema proporcionan una descripción más detallada de los servicios exactos que se proporcionarán y sus restricciones. A continuación, se describen los requisitos, tanto funcionales como no funcionales, de este sistema. Cada descripción de requerimiento tendrá la siguiente forma:

IDENTIFICADOR DE REQUISITO. Texto descriptivo. **(calificación)**	
IDENTIFICADOR DE REQUISITO	Identificador del requisito. Estará formado por las iniciales de la categoría a la que pertenece: - **RF:** Requisito Funcional - **RNF:** Requisito No Funcional - **RFU**: Requisito Funcional de Usuario - **RFO:** Requisito Funcional Operacional - **RFI**: Requisito Funcional de Interfaz - **RFR:** Requisito Funcional de Recursos - **RFP:** Requisito Funcional de Pestaciones Seguidas de un guión y un número natural. Ejemplo **RF-1**
Texto descriptivo	Párrafo que describe el requisito de forma concisa
(calificación)	Determinan la necesidad del requisito. Existen tres calificaciones posibles: - **Esencial:** debe estar presente obligatoriamente. - **Deseable:** no es obligatorio pero supone mejoras al producto - **Opcional:** define alternativas que podrían definir productos diferentes.

Tabla 7.1 Cómo definir requisitos software

7.3.1. Requisitos funcionales

Especifican lo que debe hacer o los servicios que debe proporcionar el sistema, es decir, deben describir cómo responderá el sistema ante distintas entradas, y su comportamiento frente a situaciones particulares.

CÓDIGO	DESCRIPCIÓN	EJEMPLOS
RF-1	El corrector será capaz de detectar la omisión de una letra en una palabra. **(esencial)**	1. Debe tomar **ibuprofno.** (ibuprofeno) 2. El **asceso** ha sido debido a… (absceso)
RF-2	El corrector será capaz de detectar la adición de letras en una palabra. **(esencial)**	3. Tomar **paraccetamol** cada 4 h. (paracetamol)

CÓDIGO	DESCRIPCIÓN	EJEMPLOS
RF-3	El corrector será capaz de detectar la alteración del orden de las letras, en una palabra. **(esencial)**	4. **Analegsia** general en intervenciones dolorosas. (Analgesia)
RF-4	El corrector será capaz de detectar el cambio de una letra por otra. **(esencial)**	5. Tomar **lorazapam** 1 mg. (lorazepam)
RF-5	Independientemente del número de veces que una misma palabra se escriba erróneamente, el corrector será capaz de detectarlas. **(esencial)**	6. El paciente presenta un **neumotra** dcho. El **neumotrax** ha sido causado por un traumatismo. (neumotórax)
RF-6	El corrector será capaz de detectar la repetición de una letra, en una palabra, causada por la pulsación mantenida de una letra. **(esencial)**	7. **Urbaaason** comp. 4 mg. (Urbason)
RF-7	El corrector será capaz de detectar la falta de espacio entre palabras. **(esencial)**	8. Debe **tomarvalium** comp. 5 mg. (tomarvalium)
RF-8	El corrector será capaz de detectar errores tipográficos en una palabra por introducir una letra equivocada por proximidad en el teclado. **(esencial)**	9. Debe tomar **bvqalium** comp. 5 mg. (valium)
RFU-1	El usuario introducirá de forma manual el texto que se analizará. **(esencial)**	
RFU-2	El usuario podrá ampliar o reducir el texto a analizar en cualquier momento. **(esencial)**	
RFU-3	El usuario podrá visualizar las palabras detectadas como erróneas en pantalla. **(esencial)**	
RFU-4	El usuario podrá visualizar una lista con las palabras sugeridas a la palabra detectada como errónea. **(esencial)**	
RFU-5	El usuario podrá visualizar las estadísticas/porcentaje de concordancia de todas las palabras sugeridas a la palabra	

	detectada como errónea. **(esencial)**	
RFU-6	El usuario podrá reiniciar el corrector cuando desee. **(esencial)**	
RFO-1	El corrector únicamente tendrá en cuenta aspectos morfológicos para determinar la similitud entre palabras, descartando los semánticos y fonéticos. **(esencial)**	
RFI-1	El corrector recibirá el texto a analizar manualmente por el usuario. **(esencial)**	
RFI-2	El corrector genera como salida el texto con las palabras detectadas como erróneas marcadas y se muestra en pantalla. **(esencial)**	
RFR-1	El corrector podrá ser usado bajo cualquier sistema operativo. **(esencial)**	
RFP-1	En la salida por pantalla del corrector, se marcará con color azul los errores tipográficos y ortográficos. **(esencial)**	
RFP-2	En la salida por pantalla del corrector, al posicionarse sobre la palabra detectada como errónea, se despliega una lista con todas las palabras sugeridas. **(esencial)**	
RFP-3	En la salida por pantalla del corrector, al clicar sobre la palabra detectada como errónea, se visualiza una ventana con las estadísticas de cada una de las palabras sugeridas, porcentaje de concordancia y algoritmo usado. **(esencial)**	

7.3.2. Requisitos no funcionales

Los requerimientos no funcionales representan características generales y restricciones de los servicios del sistema o funciones que ofrece.

CÓDIGO	DESCRIPCIÓN	EJEMPLOS
RNF-1	El sistema debe ser capaz de procesar de forma ágil, reduciendo al mínimo el tiempo de espera del usuario. **(esencial)**	
RNF-2	El sistema debe ser capaz de atender a la vez a varios usuarios con sesiones concurrentes. **(esencial)**	
RNF-3	El tiempo de aprendizaje de la aplicación por un usuario deberá ser mínimo. **(esencial)**	
RNF-4	La aplicación web debe poseer un diseño “Responsive” a fin de garantizar la adecuada visualización en múltiples computadores personales, dispositivos tableta y teléfonos inteligentes. **(esencial)**	
RNF-5	El sistema debe poseer interfaces gráficas bien formadas. **(esencial)**	
RNF-6	Debe de disponer de memoria y espacio en disco para almacenar el corpus médico. **(esencial)**	

7.4. Prototipo

De los dos tipos de prototipos existentes, software y papel, en este apartado se diseña la página web y su navegación con un prototipo de papel. Se realiza con el objetivo de comunicar ideas generales y ofrecer una visión de la funcionalidad del

producto y de la aplicación web. Así, se ha conseguido hacer participar al usuario en el diseño y se ha podido evaluar en las primeras fases.

Para poder simular las diferentes iteraciones que se van a realizar con el sistema, se ha realizado una hoja para cada uno de los diferentes escenarios que se van a tener como resultado de las diferentes iteraciones que se pueden realizar. Para simular la aplicación, se han apilado dichas hojas, tal y como lo detallan las ilustraciones 7.3, 7.4, 7.5 y 7.6:

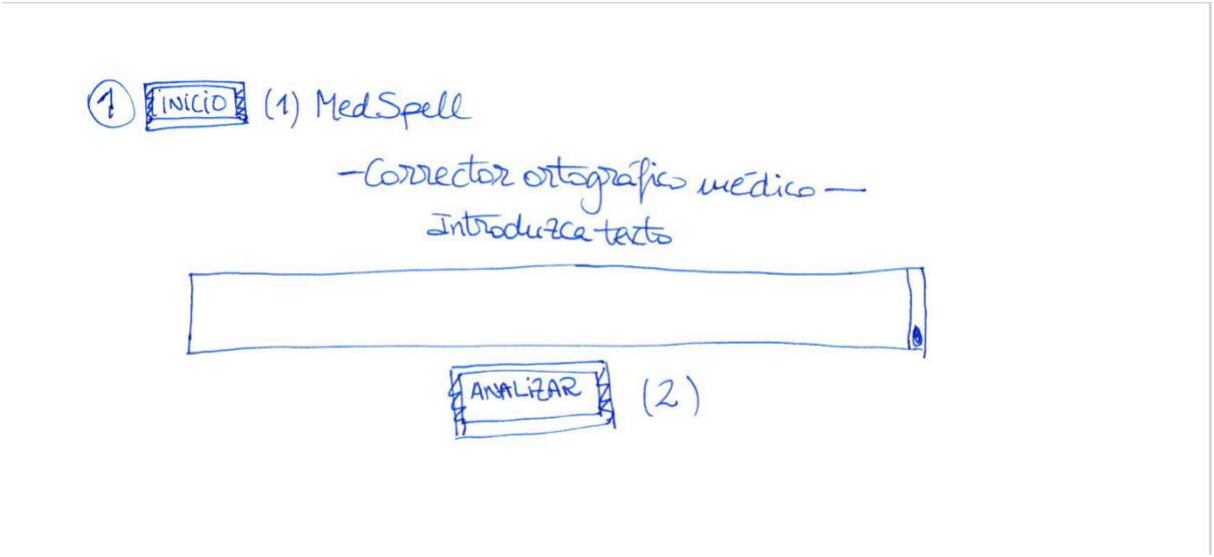

Ilustración 7.3 Prototipo MedSpell parte I

La página inicial -ilustración 7.3- se compone con el nombre del proyecto, descripción del servicio que se realiza, además de dos botones, *Inicio*, para volver a la página inicial con el cuadro de texto limpio, y *Analizar,* para procesar el texto y detectar los errores. Al pulsarlo obtenemos el resultado como muestra la ilustración 7.4

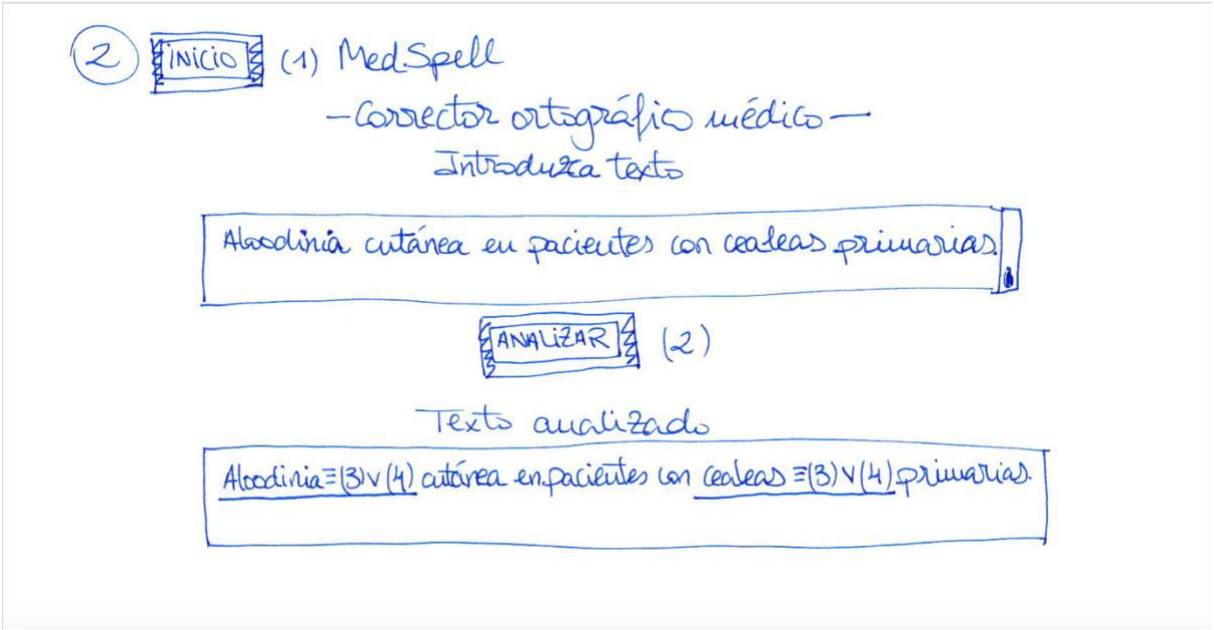

Ilustración 7.4 Prototipo MedSpell parte II

El texto analizado se visualiza en otro cuadro de texto con las palabras incorrectas marcadas. Éstas ofrecen dos tipos de información, si mantiene el cursor del ratón sobre ella, se obtiene una lista con todas las palabras sugeridas, como se ve en la ilustración 7.5. O si se clica sobre ella, se despliega una tabla informando de la estadística de dicha palabra. En ella se refleja el corrector que la ha sugerido, la palabra que ha sugerido dicho corrector y el porcentaje de concordancia con la palabra errónea original. Esta información aparecerá ordenada de mayor a menor similitud. Toda esta última funcionalidad se expone en la ilustración 7.6

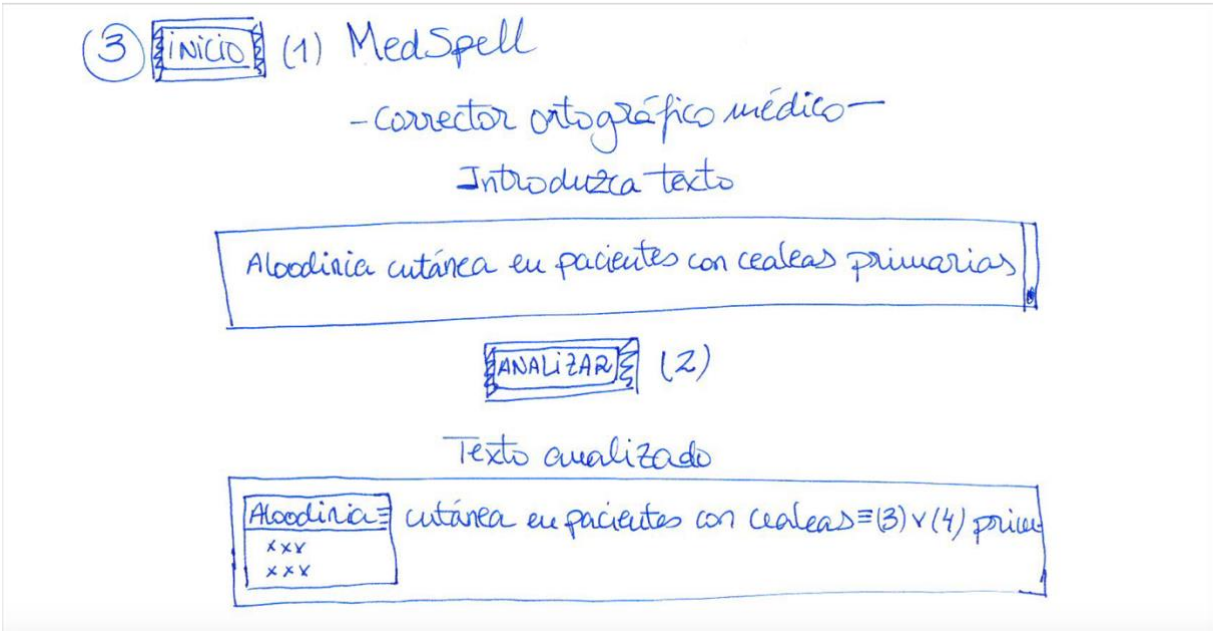

Ilustración 7.5 Prototipo MedSpell parte III

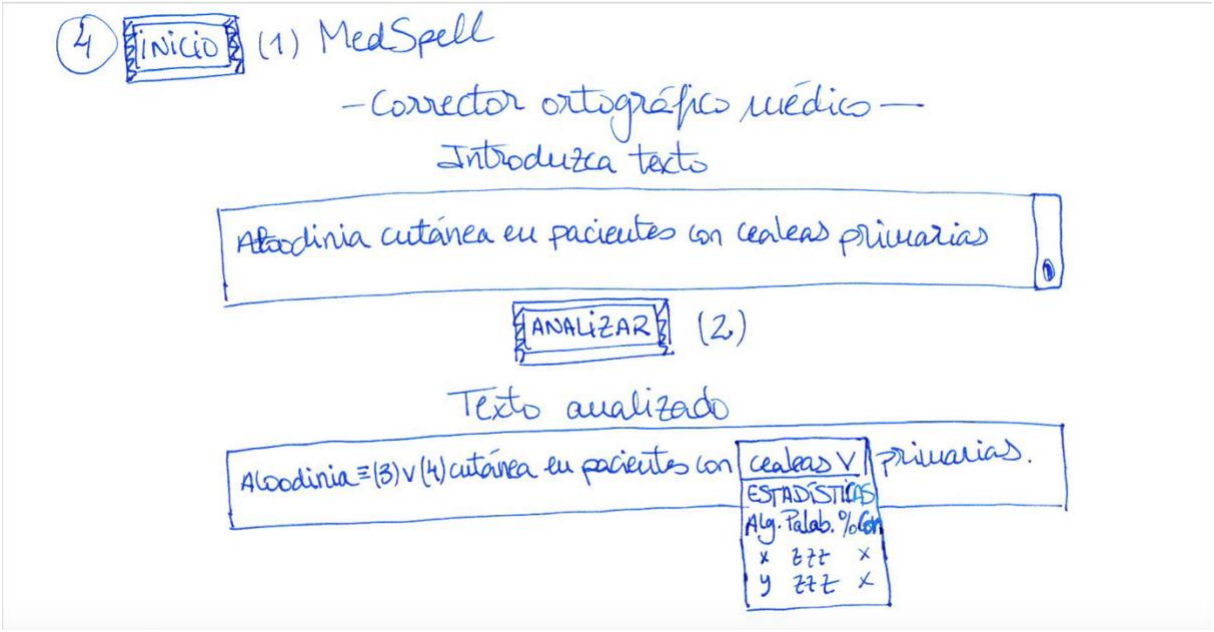

Ilustración 7.6 Prototipo MedSpell parte IV

8. DISEÑO E IMPLEMENTACIÓN

En este apartado se expone de manera detallada el desarrollo e implementación de todas las partes del proyecto y se describen todas las tecnologías que se han utilizado.

8.1. Tecnologías y librerías

En este apartado se describen todas las tecnologías usadas para el desarrollo del proyecto.

8.1.1. Entorno de desarrollo

Se ha usado Visual Studio Code, versión 1.48.2 como editor de programación, al ser multiplataforma y con unas características adecuadas para el desarrollo del proyecto.

8.1.2. Python

Python ha sido el lenguaje de programación utilizado para el desarrollo del proyecto, concretamente Python 3.

Su sintaxis es extremadamente limpia, lo que lo convierte en una alternativa natural. Se ha considerado que mezclar técnicas *ágiles* y Python pueden derivar en

una potente herramienta de programación tal y como se explica en [31]. Se puede definir como lenguaje interpretado, multiparadigma, de alto nivel, de tipo dinámico y especialmente, con fuerte orientación a objetos. Por ello, con la intención de obtener un código bien organizado con vista a posibles modificaciones futuras y comprensión, se ha creado un fichero .py por cada clase necesaria.

Se ha escogido Python frente a otras opciones por los siguientes motivos:

- Primero, Python es uno de los lenguajes de programación en los que las limitaciones del ordenador no han restringido la naturaleza de éste (nacido cuando los ordenadores eran ya tan potentes como para no tener que estar continuamente pensando en la memoria y la velocidad de ejecución). Además, de ser portable, las propiedades de este lenguaje son simples y fáciles de aprender, su naturaleza como lenguaje de propósito general hace que disponga de una gran cantidad de librerías y herramientas para programar.
- Segundo, se trata de una de las herramientas más potentes debido a la gran extensión temática de su biblioteca, protocolos de Internet, multimedia, GUI, Internacionalización, etc. Siendo lo más relevante para la implementación de este proyecto el módulo para Manipulación de texto [32].
- Tercero, permite la creación de entornos virtuales. Un entorno virtual es un ámbito de trabajo aislado al sistema, permitiendo así tener un entorno de trabajo personalizado en un proyecto concreto y pudiendo instalar las librerías o versiones de las mismas sin que afecte al entorno del sistema.
- Y cuarto, es el lenguaje Orientado a Objetos, que el equipo docente ha aconsejado en el desarrollo de asignaturas tales como Recuperación de Información y Procesamiento del Lenguaje Natural, ambas íntimamente relacionadas con el procesamiento de texto, tema central en la implementación de este proyecto.

Para la implementación del proyecto se ha creado un entorno virtual donde se han instalado todas las librerías necesarias para el correcto funcionamiento del sistema. Con ello, al instalar el servidor se instala también dicho entorno virtual facilitando la portabilidad de la aplicación.

8.1.3. Librerías

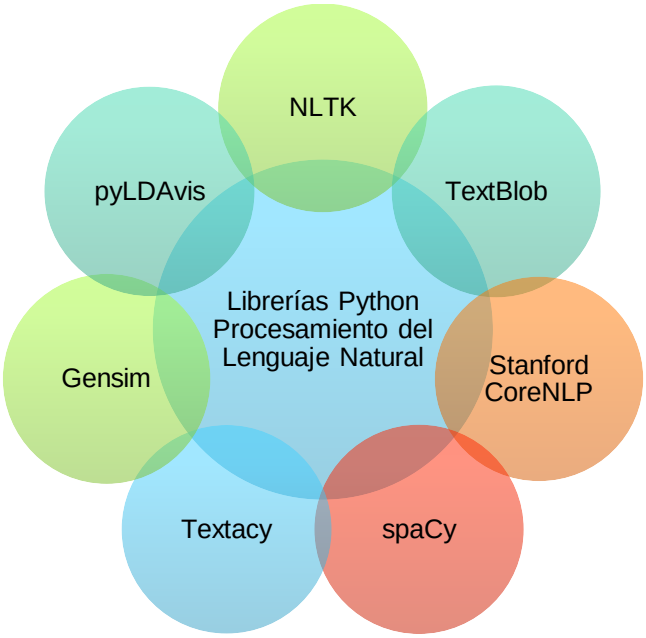

Ilustración 8.1 Librerías Python para PLN

Entre las actuales librerías que proporciona Python para abordar los problemas de PLN, como pueden verse en la ilustración 8.1, se han usado spaCy y NLTK para el desarrollo del proyecto.

Se ha optado por spaCy por ser la librería de procesamiento natural más rápida que existe. Está diseñada para trabajar con los caracteres sin procesar, generalmente utilizando el conocimiento lingüístico para agregar información útil. spaCy también es muy útil para preparar texto para otras tareas de aprendizaje automático. Además, está diseñada específicamente con el objetivo de ser una biblioteca útil para implementar sistemas listos para producción, en productos reales.

Sus principales características son:

- Tokenización no destructiva
- Entidad denominada reconocimiento
- Soporte para más de 59 idiomas
- 46 modelos estadísticos para 16 idiomas
- Vectores de palabras preentrenados
- Velocidad
- Integración fácil de aprendizaje profundo
- Análisis de dependencia etiquetado

- Segmentación de oraciones basadas en sintaxis
- Codifica todas las cadenas en valores hash, reduciendo el uso de memoria y mejorando la eficiencia.
- Serialización binaria eficiente.
- Empaquetado y despliegue de modelos fáciles.
- Precisión robusta, rigurosamente evaluada.

Para el objetivo de este proyecto, spaCy aporta velocidad y todo el preprocesamiento de un texto, normalización, asigna vectores de token específicos de contexto, etiquetas POS (Part-of-speech), análisis de dependencia y entidades con nombre de forma intrínseca.

spaCy se ha utilizado en la API que se ha desarrollado, para realizar el procesamiento del texto a corregir, obteniendo los tokens que serán evaluados por los correctores ortográficos para determinar si son palabras mal escritas o no.

Dado que spaCy realiza un modelado de texto y un exhaustivo procesamiento de lenguaje a un texto, se ha usado también la librería NLTK para ampliar el vocabulario de los correctores ortográficos utilizados, con los términos médicos recogidos en el Corpus_médico creado para tal efecto. Han sido procesados con NLTK ya que de estos ficheros lo único que interesa son los tokens, y es mucho más rápido obtenerlos con NLTK, en lugar de realizar todo el procesamiento del lenguaje natural que realiza spaCy, para quedarse solo con los tokens.

En el apartado 8.3.1. Diccionario, se explica con más detalle este proceso.

8.1.4. Tecnologías web

A continuación, se exponen todas las tecnologías web que se han utilizado para el desarrollo del proyecto.

8.1.4.1. Flask

Flask es un microframework escrito en Python para facilitar el desarrollo de Aplicaciones Web. Está basado en el motor de templates Jinja2.

De las opciones estudiadas para crear páginas web, se ha elegido el microframework Flask [33] para realizar este proyecto. principalmente por los siguientes motivos:

- Permite crear de una manera muy sencilla aplicaciones web con Python, no tiene una curva tan elevada de aprendizaje como Django.
- Es compatible con Python 3 y con WSGI -Web Server Gateway Interface-, protocolo que utilizan los servidores web para servir páginas web escritas en Python.
- Soporta de manera nativa el uso de cookies seguras.
- Sirve para construir servicios web -como APIs REST- o aplicaciones de contenido estático (el código HTML no cambia una vez colocado en el servidor), que es el que se ha implementado en este proyecto.
- Flask es Open Source, tiene buena documentación, código GitHub y está amparado bajo una licencia de software para los sistemas BSD (Berkeley Software Distribution), un tipo del sistema operativo Unix-like.

Para añadir nuevas funcionalidades a la aplicación web, se han usado las extensiones (plugins) que en los siguientes puntos se detallan.

8.1.4.2. Flask RESTful

Flask-RESTful es la extensión para Flask que se ha empleado para construir la API REST de este proyecto.

8.1.4.3. Flask-WTF

Sirve para generar formularios de HTML con clases y objetos de forma sencilla y segura. Como se ha explicado anteriormente, Flask está basado en el protocolo WSGI teniendo integradas opciones de seguridad para cookies y prevención de ataques CSRF (Cross Site Request Forgery), mediante una clave criptográfica que se puede definir para firmar tokens. La clave para cifrar los tokens se toma del parámetro de configuración *“SECRET_KEY”.*

Con ello se consigue encapsular cualquier módulo en clases. Así, cuando se ha necesitado crear un elemento para dicho formulario, únicamente se ha creado el objeto del tipo adecuado. La plantilla correspondiente ha usado este objeto para crear

los formularios obteniendo un código más legible y ahorrando escribir demasiado código HTML.

8.1.4.4. Flask-Markdown

"Small extension to make using markdown easy". Extensión necesaria para que la aplicación web visualice contenido HTML.

8.1.4.5. Requests

Para tratar las peticiones web HTTP. Requests quita las complicaciones de trabajar HTTP/1.1 en Python -haciendo que la integración con servicios web sea transparente-. No hay necesidad de agregar queries manualmente a las URLs, o convertir la información a formularios para hacer una petición POST. La reutilización de keep-alive y conexión HTTP se hace automáticamente, todo gracias a urllib3, el cual está integrado en Requests.

8.1.4.6. Bootstrap

Es una biblioteca multiplataforma o conjunto de herramientas de código abierto usada para el diseño de la aplicación web. Se han creado las hojas de estilo ocupándose solo del desarrollo front-end permitiendo convertir los datos en una interfaz gráfica, mediante el uso de HTML, CSS (ajustándose a teléfonos, tabletas y escritorios) y JavaScript para que el usuario pueda ver e interactuar con estos datos.

8.1.5. Tipos de Correctores

En este apartado se explican, de los diferentes correctores Python, los dos utilizados en este proyecto por sus características adecuadas para ello.

A pesar de que la mayoría de los correctores ortográficos hacen uso de diccionarios, éstos pueden discrepar en forma, ya que no es necesario el almacenamiento de todo el vocabulario de un idioma con todas sus palabras completas. Es la separación entre la raíz de una palabra y sus posibles prefijos y sufijos, proceso denominado lematización o stemming, lo que provoca dicha diferencia entre distintos correctores ortográficos. Este sistema surge como respuesta a la

necesidad de reducir el espacio de memoria y es el que usa el corrector ortográfico Hunspell que es uno de los que se utilizan en el desarrollo de este proyecto.

8.1.5.1. Corrector ortográfico Spell-checker

Soporta Python 2.7 y 3, siendo este último la versión preferida [34]. pyspellchecker proporciona una biblioteca para determinar si una palabra está mal escrita y cuál es la probable ortografía correcta basada en la frecuencia de las palabras.

Como se explica en [35], se basa en el trabajo de Peter Norvig [36] y utiliza el algoritmo *Levenshtein Distance* (o *Edit Distance)* para encontrar permutaciones dentro de una distancia con la palabra original. Permite la configuración de esta distancia, teniendo valor predeterminado 2, y usando un valor de distancia 1 para palabras más largas, considerando que las cadenas son idénticas si su distancia es 0.

8.1.5.2. Corrector ortográfico Hunspell

Hunspell, basado en Myspell, es el corrector ortográfico utilizado por herramientas como *LibreOffice, OpenOffice.org, Mozilla Firefox 3 y Thunderbird, Google Chrome,* y también es utilizado por paquetes de software propietarios, como *macOs, InDesign, memoQ, Opera y SDL Trados.*

Hunspell utiliza diccionarios para proporcionar la verificación ortográfica y análisis morfológico en el que se distinguen dos archivos de texto diferentes. Uno en el que se establecen los códigos y los caracteres a introducir o eliminar, con extensión .aff. Y otro con con extensión .dic, que es la lista de palabras base junto con un grupo de letras que se refieren a los afijos que se encuentran en el archivo .aff. Esto ahorra espacio porque en lugar de tener que incluir todas las formas de una palabra, -como salto, saltos, saltó-, el archivo .dic incluirá la palabra una vez y las referencias a los afijos en el archivo .aff permiten la construcción de toda las otras formas, tal y como se explica en [37].

Ilustración 8.2 Imagen de archivo .aff y .dic

Sus principales funcionalidades son:

- Soporte extendido para las peculiaridades del lenguaje, codificación de caracteres Unicode, composición y morfología compleja.
- Sugerencias mejoradas usando similitud con n-gramas (n palabras consecutivas en el corpus).
- Análisis morfológico, derivación y generación.
- Posee interfaces para distintos lenguajes de programación (incluyendo Python).

8.2. Modelo de datos

Para el desarrollo del proyecto MedSpell no se ha utilizado ninguna BBDD (base de datos). Se han usado los modelos de lenguaje que usan las librerías spaCy y NLTK además de un corpus médico.

8.3. Implementación

En este apartado se analiza de manera detallada la implementación de cada una de las partes primordiales que componen el proyecto.

8.3.1. Preparación del entorno de desarrollo

Los pasos seguidos para el desarrollo del proyecto se detallan en este apartado.

8.3.1.1. Instalación de Anaconda Individual Edition 2020.02

Se ha descargado el instalador gráfico de la distribución, desde la página oficial de Anaconda en [38].

Existen instaladores para los sistemas Windows, MacOS y Linux. MacOS en mi caso: https://repo.anaconda.com/archive/Anaconda3-2020.02-MacOSX-x86_64.pkg

Se ha iniciado el instalador y seguido sus instrucciones hasta completar el proceso.

8.3.1.2. Instalación de Visual Studio Code

Visual Studio Code es un editor de código libre y abierto, diseñado por MicroSoft y optimizado para crear y depurar aplicaciones web modernas y aplicaciones en la nube.

Se ha descargado el instalador de la última versión disponible en su página oficial, en [39].

Una vez instalado el editor de código, se han instalado varias extensiones que facilitarán la labor a la hora del desarrollo de la aplicación:

8.3.1.2.1. Python para Visual Studio Code

Una extensión con amplio soporte para el lenguaje Python (para todas las versiones soportadas activamente: 2.7, >= 3.5), que incluye características como IntelliSense, linting, depuración, navegación de código, formato de código, soporte para notebook Jupyter, refactorización, explorador de variables, explorador de pruebas, fragmentos y mucho más.

8.3.1.2.2. flask-snippets

Colecciones de fragmentos de código. Inicialmente portado desde PyCharm, TextMate, SublimeText y otros editores / IDEs.

8.3.1.2.3. Jinja2 Snippet Kit

Esta extensión agrupa todos los fragmentos de jinja más utilizados, listos para su utilización en plantillas HTML. También incluye utilidades para tareas como generar números, unir listas y cosas por el estilo.

8.3.1.3. Entorno virtual Python

Se ha hecho uso de un entorno virtual Python para mantener en un espacio separado el proyecto con sus dependencias y módulos. Este entorno es específico para el proyecto y no interferirá, ni se verá afectado, con las dependencias de otros proyectos.

8.3.1.3.1. Creación del entorno virtual

Se ha creado el entorno virtual devmedspell con el siguiente comando *conda create -n devmedspell python=3.7*

8.3.1.3.2. Activación del entorno virtual

Antes de empezar a trabajar en el proyecto, siempre se debe de activar el entorno virtual con el comando *conda activate devmedspell*

8.3.2. Diccionario

Uno de los retos que debe afrontarse a la hora de trabajar en el desarrollo de un corrector ortográfico médico español es la construcción de un lexicón que contenga todas las formas correctas de las palabras que existen en el español además de todas las palabras técnicas médicas, farmacológicas, abreviaturas, etc.

El lexicón desarrollado para este proyecto se compone de un compendio de archivos de textos adquiridos de las siguientes fuentes:

- El diccionario incluido en los modelos de lenguaje que usan las librerías spaCy y NLTK.
- El diccionario del que hace uso Spell-checker, el corrector ortográfico usado.
- El diccionario que incorpora Hunspell, el corrector ortográfico usado, hunspell-dicts.
- La colección de medicamentos (vademécum) existentes en [40] y en [41].
- El vocabulario normalizado SNOMED CT, seleccionado para la Historia Clínica Digital del Sistema Nacional de Salud (HCDSNS) [42].

- El manual de codificación de diagnósticos CIE-10 obtenido de [43] ofrecido por el Sistema Nacional de Salud del Gobierno de España.
- Vocabulario del UMLS (Sistema Unificado de Lenguaje Médico) proporcionado por mi tutora. Es un compendio de muchos vocabularios biomédicos diseñado y mantenido por la Biblioteca Nacional de Medicina de EEUU.

Se ha tenido especial cuidado en que los textos de entrada estén correctamente escritos, ya que, si alguno de ellos incluyera errores ortográficos, estos se traspasarían al listado de palabras correctas, lo que iría en desmedro del funcionamiento posterior del corrector ortográfico.

8.3.3. Servicio web API (webapi)

El módulo **__init__.py** es el encargado de la creación del servicio web. Este quedará a la escucha de peticiones en el puerto 8080, en el punto de servicio */medspell-check/<texto_a_analizar>*

En el proceso de inicio, el servidor:

- Realiza la carga del modelo del lenguaje español de la librería spaCy (es_core_news_sm).
- Crea instancias de los correctores ortográficos spellchecker y hunspell.
- Añade a ambos correctores el corpus de términos médicos creado, como se ha definido en el punto 8.3.2.

El módulo **medspell_analycer.py** define la clase *MedSpell_Check*, que implementa el método get, encargado de:

- Recibir la petición de servicio.
- Procesar el texto a analizar con spaCy.
- Devolver al peticionario del servicio el resultado del proceso en un documento JSON, haciendo uso de la función medspell_to_JSON que brinda el módulo medspell_lib.py.

El módulo **medspell_lib.py** proporciona el grueso de la lógica aplicada en el análisis del texto. El procesamiento es realizado en la función *medspell_to_JSON*, en la que:

- Se normaliza el texto, pasándolo a minúsculas, eliminando símbolos de puntuación y descartando palabras muy cortas y formas no convencionales (números y palabras compuestas por caracteres alfanuméricos y otros símbolos).
- A partir del texto normalizado, se obtiene una lista con las palabras mal escritas, detectadas por el corrector ortográfico hunspell.
- Para cada palabra mal escrita, contenida en la lista obtenida en el punto anterior, se obtienen las palabras sugeridas por los correctores spellchecker y hunspell, así como el porcentaje de concordancia de cada una de ellas, con la palabra original, utilizando el algoritmo jaro_winkler de Levenshtein.
- Con toda esta información (palabras mal escritas, palabras sugeridas y porcentaje de similitud) se compone el objeto JSON que será devuelto al consumidor de la API. Este JSON tiene la siguiente estructura:


```
{
    "diccionarios": ["Vademecum", "Diagnosticos", "CIE10", "Monografias ATP", "SNOMED", "UMLS"],
    "palabras": [{
        "original": "ibrofeno",
        "sugeridas": [{
            "spellchecker": [{
                "palabra": "ibuprofeno",
                "jaro-winkler-similitud": 0.9466666666666665
            }],
            "hunspell": [{
                "palabra": "ibuprofeno",
                "jaro-winkler-similitud": 0.9466666666666665
            }, {
                "palabra": "apirofeno",
                "jaro-winkler-similitud": 0.8842592592592592
            }, {
                "palabra": "nitrofenol",
                "jaro-winkler-similitud": 0.8583333333333334
            }, {
                "palabra": "indoprofeno",
                "jaro-winkler-similitud": 0.7248376623376623
            }]
        }]
    }]
}
```

- **diccionarios** contiene una lista con los archivos del corpus de terminología médica que ha sido añadido a los correctores ortográficos.
- **palabras,** es la lista de palabras erróneas detectadas en el texto procesado, junto con las palabras sugeridas por cada uno de los correctores ortográficos utilizados y el porcentaje de concordancia de cada sugerida con la errónea.

8.3.4. Aplicación web (webapp)

El módulo **__init__.py** es el encargado de la creación de la aplicación web, que quedará publicada en el puerto 9090, y estará disponible a través de la URL http://localhost:9090

La aplicación web presentará una interface simple, compuesta por:

- Un ***cuadro de*** **texto**, para permitir al usuario la introducción del texto a procesar, y
- un **botón Analizar**, que realizará la petición a la API desarrollada, http://localhost:9090/medspell-check/<texto>, y presentará los resultados recibidos al usuario.

El botón *Analizar:*

- Realiza una limpieza del texto antes de pasarlo a la API, eliminando las etiquetas HTML que el usuario haya podido introducir, así como caracteres especiales que pudieran interferir en el proceso de análisis por parte de la API, tales como / # y el carácter Unicode 'NO-BREAK SPACE' (U+00A0).
- Invoca a la API enviándole el texto limpio.
- Procesa el documento JSON recibido como respuesta a la petición y, compone el código HTML responsable de presentar de forma amigable al usuario, el texto con el resultado del análisis ortográfico obtenido.
- Presenta al usuario el texto procesado, remarcando en él las palabras mal escritas. Manteniendo el cursor del ratón sobre cada palabra remarcada, la interface presentará una lista con las palabras sugeridas que la API ha encontrado para dicha palabra. Si hacemos click sobre la palabra remarcada la aplicación nos presentará una ventana con las estadísticas sobre dicha palabra, mostrando el porcentaje de concordancia de cada palabra sugerida, así como el algoritmo que la ha sugerido.

8.4. Pruebas y validación

El proyecto pasa por este estadio con el objetivo de analizar la funcionalidad, por una parte, de los elementos de la aplicación web y, por otra parte, probar la eficacia y eficiencia de los sistemas de corrección implementados.

Para la validación del correcto funcionamiento de la WebApp se hacen:

- Pruebas de Contenido. Realizando pruebas sintácticas (ortografía y gramática del contenido), semánticas (exactitud de la información presentada), comprobando que la información es precisa, concisa y exacta, que el contenido no infrinja derechos de autor o marcas registras.
- Pruebas de Interfaz del Usuario según explica Hassan Montero en [44]. Se ha probado ejecutando la aplicación, haciendo pruebas de HTML dinámico ejecutando la aplicación, pruebas dentro de una diversidad de ambientes para asegurar su compatibilidad, pruebas para la validación de la usabilidad midiendo cuanto de fácil es entender la aplicación con la participación de varios usuarios inexpertos. Se ha prestado especial atención en los casos en los que el sistema debe mostrar mensajes de error o alerta y queden resueltos. En cuanto al diseño de la interfaz, se ha optado por un diseño minimalista (menos, es más) tal y como propone Jakob Nielsen con las 10 heurísticas/principios básicos de usabilidad en [45].
- Pruebas de semántica de la interfaz con el objetivo de evaluar cuán bien el diseño se ocupa de los usuarios ofreciendo una dirección clara y manteniendo consistencia de lenguaje y enfoque.
- Pruebas de compatibilidad debido a que el usuario puede tener diferentes ambientes (sistemas operativos, navegador, plataformas de hardware).
- Pruebas de navegación. Se han realizado con el objetivo de garantizar que todos los mecanismos que permiten al usuario de la WebApp viajar a través de ella sean funcionales. Comprobando para ello los vínculos de navegación.

Para probar la eficiencia y eficacia del corrector se han introducido fragmentos de textos con erratas de distintos sitios web, enfocados a la medicina. Dichos textos proceden de distintas revistas españolas de medicina, [46] y [47].

Mediante el análisis de las correcciones que ha realizado el sistema, se confirma que el corrector hunspell es más efectivo y conciso que el corrector spellchecker.

9. CONCLUSIÓN

La elección de este proyecto surgió tras cursar las asignaturas de Sistemas de Recuperación de la Información, y en especial Procesamiento del Lenguaje Natural. Ambas tienen como base el análisis y procesamiento de textos.

La idea que me propusieron los tutores del proyecto me pareció interesante, puesto que es una temática que está en auge debido a la introducción de las TICs (Tecnologías de la Información y la Comunicación) en los diferentes ámbitos sociales y concretamente, en el ámbito sanitario.

En el transcurso del desarrollo del proyecto y a medida que avanzaba en la investigación, he asumido que se trata de un campo muy amplio y bastante complejo. Se evidencia, por tanto, que en este campo hay mucho en lo que investigar y avanzar.

En este proyecto, se matiza la importancia de la corrección ortográfica, las arquitecturas orientadas a servicios y la combinación de ambas. También se ha puesto de relieve la necesidad que hay en la actualidad de este tipo de soluciones, no solo en el ámbito de aplicaciones ofimáticas, sino también como soporte a procesos de conversión de voz a texto, reconocimiento óptico de caracteres, reconocimiento del habla, etc.

El trabajo de investigación, documentación y preparación del diccionario ha sido arduo. Me gustaría puntualizar que en lo referente a la creación de interfaces, desarrollo e implementación del código he contado con el apoyo de los conocimientos adquiridos durante la realización del Grado.

Finalmente, reflejar que he asumido que la planificación y gestión de proyectos no es algo trivial. Es necesario contar con experiencia. Esta afirmación la fundamento en que, por ejemplo, al estimar el número de horas necesarias para realizar algunas tareas he cometido algún error en la temporización de las mismas -asignar una duración superior o inferior a la que realmente requería la tarea-.

10. FUTURAS MEJORAS

Se proponen los siguientes trabajos futuros con el fin de mejorar la funcionalidad de la aplicación desarrollada:

- Serán necesarias actualizaciones periódicas del vocabulario debido al ritmo vertiginoso de los nuevos avances científicos, lo que implica la evidente actualización del lenguaje científico. Por ello, se podría añadir la funcionalidad de que el usuario amplíe algún diccionario en concreto o incluso pueda añadir nuevos.
- Se puede ofrecer al usuario la posibilidad de elegir la fuente de información en la cual buscar.
- Ampliar la funcionalidad de la web para dar la posibilidad al usuario de seleccionar alguna de las opciones sugeridas para corregir el texto.
- Ofrecer al usuario la posibilidad de elegir el color de la fuente de las palabras incorrectas detectadas.

11. MANUAL DE USUARIO

El manual de usuario se desarrolla con el objetivo de facilitar la tarea de conocimiento, uso y aprendizaje del sistema desarrollado. Contiene información acerca de todas las operaciones básicas que el sistema ofrece, así como capturas de pantallas útiles para el seguimiento de la explicación. El lenguaje utilizado es el más adecuado al perfil del usuario.

La aplicación web estará disponible a través de la URL http://localhost:9090 y presentará una interface simple:

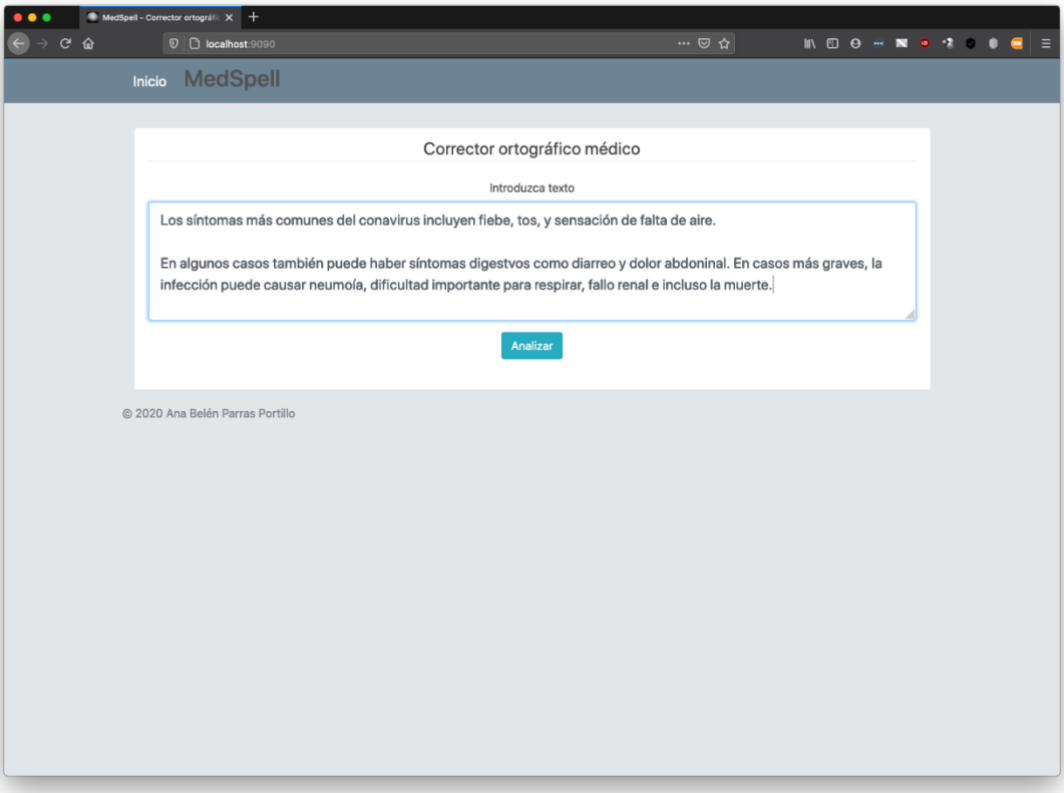

Ilustración 11.1 Interfaz de la aplicación web desarrollada

La interface está compuesta por:

- **Cuadro de texto**, permitiendo al usuario, en este caso al facultativo, la introducción del texto a procesar (informe médico, análisis de laboratorio, etc).
- **Botón Analizar,** que realizará la petición a la API desarrollada, http://localhost:9090/medspell-check/<texto>, para que realice el análisis del texto y presentará los resultados recibidos al usuario.

El botón Analizar enviará el texto a la API y los resultados de dicho análisis serán mostrados, resaltando las palabras detectadas como incorrectas:

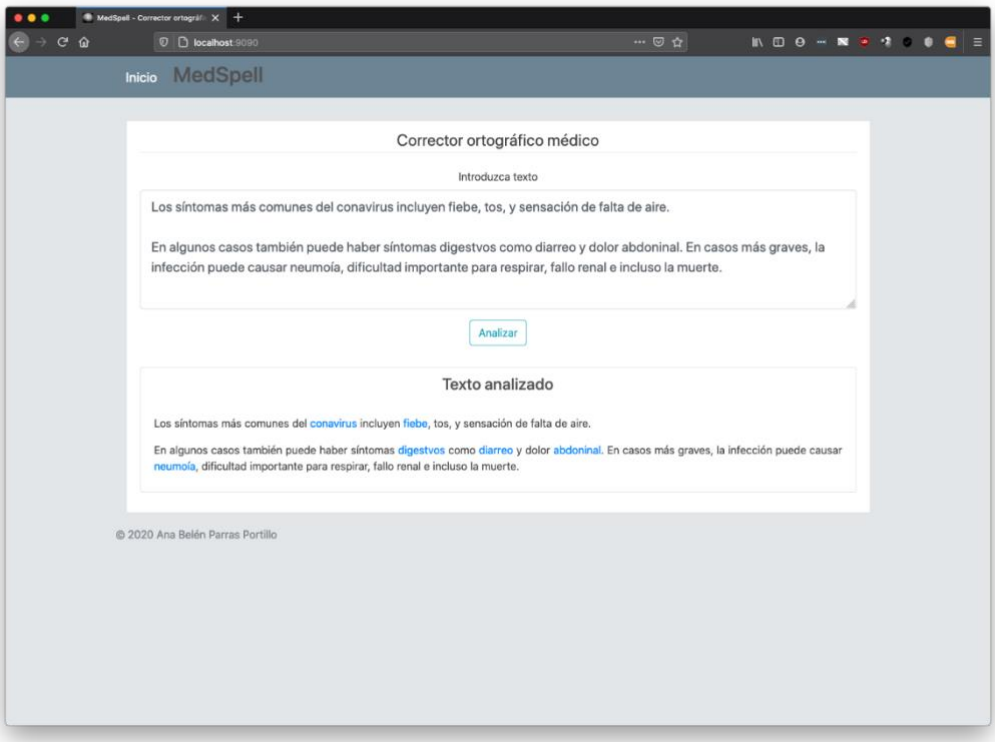

Ilustración 11.2 Visualización del análisis y resultado

Manteniendo el icono del ratón sobre alguna de las palabras incorrectas resaltadas, se mostrará una lista con las palabras sugeridas por el corrector médico:

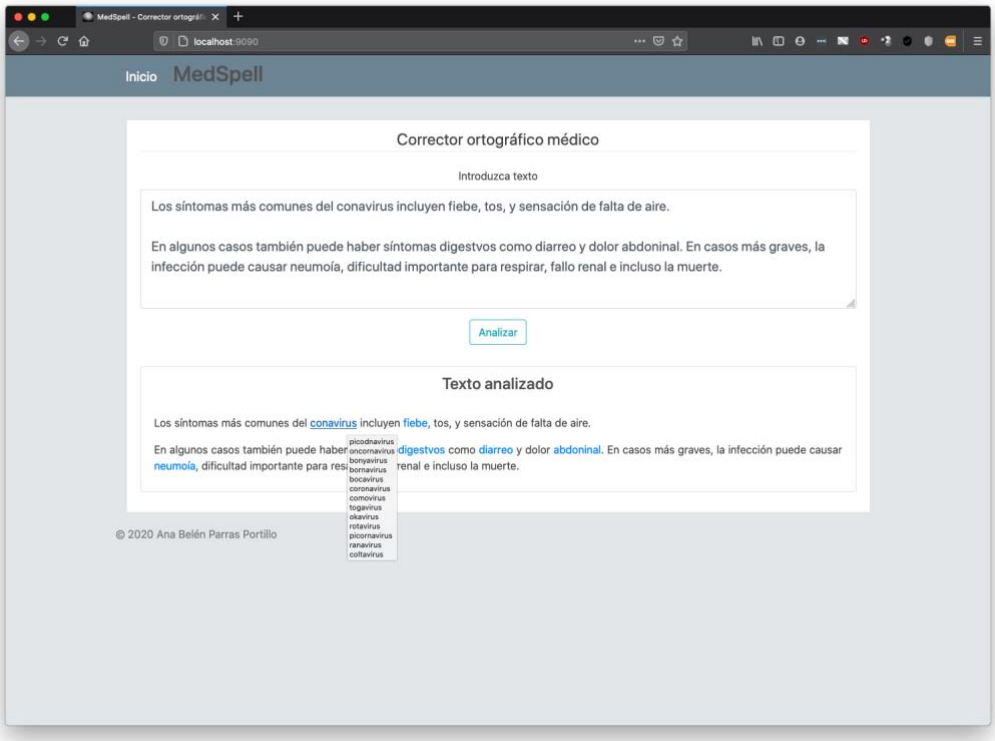

Ilustración 11.3 Visualización de la lista de palabras sugeridas por el corrector MedSpell

Y haciendo click sobre dichas palabras, obtendremos una ventana que mostrará las estadísticas de las palabras sugeridas:

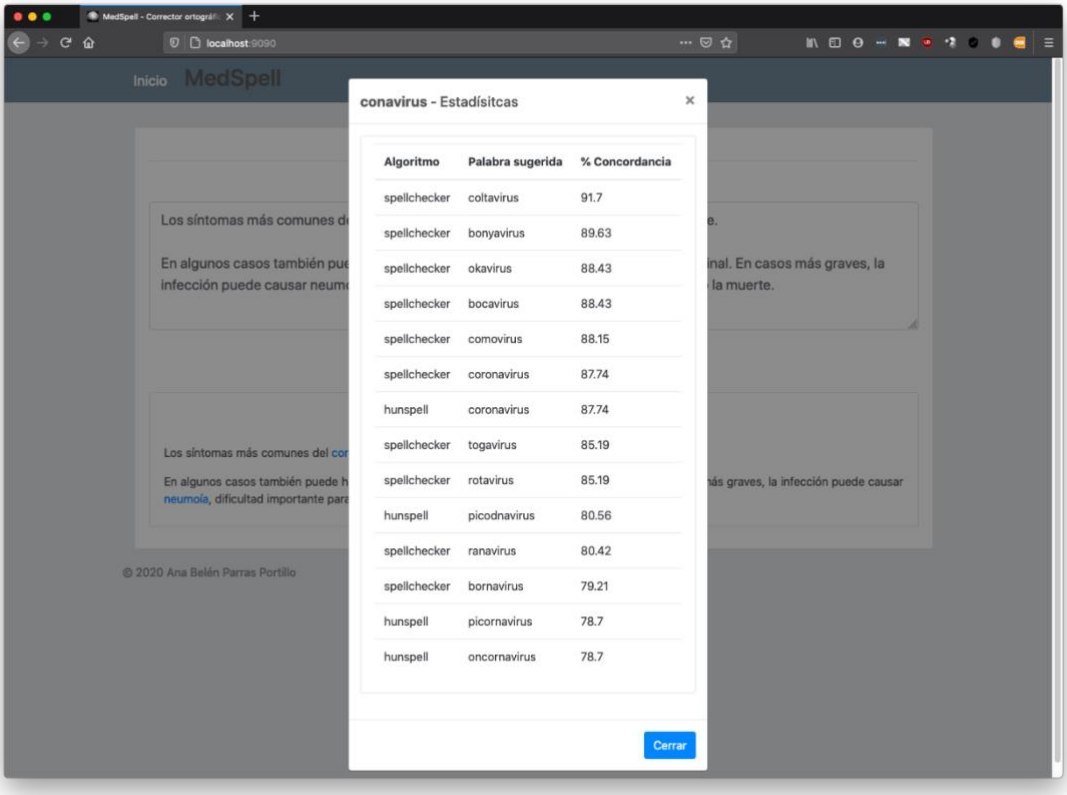

Ilustración 11.4 Visualización de las estadísticas de las palabras sugeridas por MedSpell

ANEXO I: MANUAL DE INSTALACIÓN Y EJECUCIÓN DE LA APLICACIÓN

Como se explicó en 8.2. y en 8.3.2., para el desarrollo del proyecto MedSpell no se ha utilizado ninguna BBDD, se han usado los modelos de lenguaje que usan las librerías spaCy y NLTK además de un corpus médico.

De igual forma, también se explicó en 8.3.1.3. que se ha utilizado un entorno virtual Python, devmedspell, debido a sus ventajas, por lo que se aconseja crear uno en el momento de instalar esta aplicación.

Anexo I.1 Arquitectura del corrector ortográfico médico

El sistema final ha sido concebido como un sistema distribuido, con dos servicios diferenciados:

Un **Servicio Web RESTful**, que atenderá peticiones para procesamiento de texto.

Y una **aplicación Web**, que consumirá la funcionalidad publicada por dicho servicio web, encargada de mostrar los resultados del procesamiento del texto.

Estructura de la carpeta del proyecto:

El contenido de la carpeta raíz del proyecto es el siguiente:

- **webapi**, carpeta con el código fuente del servicio web y los diccionarios con la terminología médica añadida a los correctores ortográficos utilizados.
- **webapp,** carpeta con el código fuente de la aplicación web.

Dentro de cada una de las carpetas encontraremos los siguientes ficheros entre otros:

- **README.TXT,** archivo de texto con instrucciones par la instalación del proyecto y ejecución de ambos servidores, el del servicio web y el de la aplicación web.

- **requirements.txt,** archivo que especifica los paquetes externos que el proyecto requiere.

Anexo I.2 Instalación

1. Creación de un entorno virtual Python

Situándose en la carpeta del proyecto, en la carpeta que ha sido creada tras descomprimir el archivo medspell.zip

Desde un terminal se ejecuta el comando: `$ python3 -m venv medspell-venv`
Si se está trabajando con Windows, se hará desde el símbolo del sistema:

```
> python -m venv medspell-venv
```

2. Activación del entorno virtual

Antes de empezar a trabajar con el proyecto es necesario activar el entorno virtual. Para ello se ejecuta desde el terminal: `$ . medspell-venv/bin/activate`

En Windows: `> medspell-venv\Scripts\activate`

3. Instalación de los módulos Python requeridos

Una vez activado el entorno virtual, ejecutar el comando: `pip install -r requirements.txt`

4. Descargar modelos y corpus, usando el gestor de descargas de NLTK

Iniciar el intérprete de Python: `$ Python`
y ejecutar los siguientes comandos:

```
>>> import nltk
>>> nltk.download('punkt')
>>> nltk.download('stopwords')
>>> nltk.download('treebank')
>>> nltk.download('wordnet')
>>> nltk.download('maxent_treebank_pos_tagger')
```

```
>>> nltk.download('maxent_ne_chunker')
>>> nltk.download('words')
```

Anexo I.3 Ejecución

1. Servidor Servicio Web RESTful – webapi

Desde un terminal activar el entorno virtual Python medspell-venv (ver el punto 2 Activación del entorno virtual en el apartado de Instalación)

Una vez activado dicho entorno virtual, acceder a al carpeta webapi e iniciar el servidor:

```
$ cd webapi
$ python __init__.py
```

El servidor quedará a la escucha en http://localhost:8080

2. Servidor Aplicación Web – webapp

Desde otro terminal activar el entorno virtual Python medspell-venv (ver el punto 2 Activación del entorno virtual en el apartado de Instalación)

Una vez activado el entorno virtual, acceder a la carpetea webapp e iniciar el servidor:

```
$ cd webapp
$ python __init__.py
```

El servidor quedará a la escucha en http://localhost:9090

Bibliografía

[1] M. C. P. P. L. José H. Canós, «Métodologías Ágiles en el Desarrollo de Software,» 13 03 2012. [En línea]. Available: http://roa.ult.edu.cu/handle/123456789/476.

[2] «Manifesto for Agile Software Development,» 2001. [En línea]. Available: http://agilemanifesto.org.

[3] A. Navarro, J. D. F. y J. M. , «Revisión de metodologías ágiles para el desarrollo de software,» *PROSPECTIVA*, pp. 30-39, 2013.

[4] L. D. -. 2. -. books.google.com, «Frameworks Python,» 2007. [En línea]. Available: https://scholar.google.es/scholar?hl=es&as_sdt=0%2C5&q=frameworks+python&oq=.

[5] github.com/dandelea, «spell-checker-python#description,» 2020. [En línea]. Available: https://github.com/dandelea/spell-checker-python#description.

[6] T. T. Z. -. M. LAN, «Volume 1: Proceedings of the Main Conferenceand the Shared Task,,» p. 126, 2013.

[7] G. C. Pastor, «Investigar con corpus en traducción: los retos de un nuevo paradigma,» de *Investigar con corpus en traducción: los retos de un nuevo paradigma*, Peter Lang, 2008, p. 177.

[8] V. R. Villán, «iebschool,» 15 03 2019. [En línea]. Available: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/.

[9] L. Gilibets, «IEBSchool- Qué es Kanban y cómo utilizarlo en el desarrollo de proyectos,» 31 07 2013. [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/.

[10] «The Python Tutorial ; Python 3.8.4 documentation,» 25 Junio 2020. [En línea]. Available: https://docs.python.org/3/tutorial/.

[11] jarboleya, «velneo,» velneo, 19 08 2019. [En línea]. Available: https://velneo.es/guia-indispensable-buen-analista-programador/. [Último acceso: 21 08 2020].

[12] Ministerio de Empleo y Seguridad Social, «BOE.es,» 06 03 2018. [En línea]. Available: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf. [Último acceso: 22 08 2020].

[13] Naber, 28 08 2003. [En línea]. Available: http://www.danielnaber.de/languagetool/download/style_and_grammar_checker.pdf. [Último acceso: 15 05 2020].

[14] Revista signos, «SciELO,» 2016. [En línea]. Available: https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0718-09342016000100005&lang=es. [Último acceso: 12 05 2020].

[15] D. C. Sobrino, «SOLDAI,» 24 Diciembre 2019. [En línea]. Available: https://medium.com/soldai/m%C3%A9tricas-de-similitud-para-cadenas-de-texto-parte-i-introducci%C3%B3n-154e4b724a27. [Último acceso: 01 06 2020].

[16] M. Á. Á. Carmona, *Detección de similitud en textos cortos considerando trasplate, orden y relación semántica de palabras*, Tonantzintla, Puebla, 2014.

[17] «Wiki,» [En línea]. Available: https://es.qwe.wiki/wiki/String_metric. [Último acceso: 08 06 2020].

[18] J. L. B. María Antonia Martí Antonín, «Tratamiento del lenguaje natural: tecnología de la lengua oral y escrita,» de *Tratamiento del lenguaje natural: tecnología de la lengua oral y escrita*, p. 158.

[19] W. H. G. -. A. A. Fahmy, «A Survey of Text Similarity Approaches,» *International Journal of Computer Applications (0975 –8887),* April 2013.

[20] Wikipedia. [En línea]. Available: https://en.wikipedia.org/wiki/Levenshtein_distance#cite_note-1. [Último acceso: 11 06 2020].

[21] G. Navarro, «A Guided Tour to Approximate String Matching,» pp. 15-18, 2001.

[22] D. Rodriguez, «analyticslane.com,» 2020. [En línea]. Available: https://www.analyticslane.com/2020/06/24/la-similitud-de-jaro-winkler/. [Último acceso: 11 06 2020].

[23] W. E. Yancey, «Evaluating String Comparator Performance for Record Linkage,» *Research Report Series,* pp. 3-12, 2005.

[24] I. Amón, F. M. y J. E. , «Algoritmo fonético para detección de cadenas de texto duplicadas en el idioma español,» *Revista Ingenierías Universidad de Medellín,* pp. 130-131, 2012.

[25] S. J. C. B. A. G. y F. G. , «Generalized Mongue-Elkan Method for Approximate Text String Comparison,» Verlag Berlin Heidelberg, A. Gelbukh (Ed.), 2009, pp. 560-562.

[26] W. W. Cohen, «Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity,» pp. 202-210, 1998.

[27] W. W. Cohen, P. R. y S. E. Fienberg, «A Comparison of String Distance Metrics for Name-Matching Tasks,» *American Association for Artificialo Intelligence(www.aaai.org),* 2003.

[28] S. O. A. (SOA), «fing.edu.uy,» [En línea]. Available: https://www.fing.edu.uy/inco/cursos/tsi/TSI4/2006/trabajos/SOA_paper.pdf. [Último acceso: 13 06 2020].

[29] Roy Fielding, «wikipedia.org,» [En línea]. Available: https://es.wikipedia.org/wiki/Roy_Fielding. [Último acceso: 14 06 2020].

[30] Roy Thomas Fielding, «ics.uci.edu,» 2000. [En línea]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Último acceso: 15 06 2020].

[31] Servicio_web, «wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Servicio_web. [Último acceso: 12 06 2020].

[32] «International Journal of Scientific and Research Publications,» vol. 3, nº 5, pp. 1-758, 2013.

[33] A. C. 05 12 2017. [En línea]. Available: https://d1wqtxts1xzle7.cloudfront.net/55717772/UML__ejemplo_sencillo_sobre_Modelado_de_un_Proyecto.pdf?1517820407=&response-content-disposition=inline%3B+filename%3DUML_ejemplo_sencillo_sobre_Modelado_de_u.pdf&Expires=1598354306&Signature=JwdQO4m01uDC6Peg. [Último acceso: 12 08 2020].

[34] «flask.com,» [En línea]. Available: https://flask.palletsprojects.com/en/1.1.x/quickstart/. [Último acceso: 02 07 2020].

[35] «yspellchecker,» Tyler Barrus, 2018. [En línea]. Available: https://pyspellchecker.readthedocs.io/en/latest/code.html. [Último acceso: 13 05 2020].

[36] «norvig.com,» Febrero 2007. [En línea]. Available: https://norvig.com/spell-correct.html. [Último acceso: 13 05 2020].

[37] «anaconda.com,» Individual edition, [En línea]. Available: https://www.anaconda.com/products/individual. [Último acceso: 04 07 2020].

[38] «code.visualstudio.com,» [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 04 07 2020].

[39] «wiki,» [En línea]. Available: https://es.qwe.wiki/wiki/MySpell. [Último acceso: 17 05 2020].

[40] «vademecum.es,» Vidal Vademecum Spain, 03 Diciembre 2010. [En línea]. Available: https://www.vademecum.es/medicamentos-d_1. [Último acceso: 17 06 2020].

[41] Access Medicina TM - McGraw-Hill Medical, [En línea]. Available: https://accessmedicina.mhmedical.com/book.aspx?bookid=1552&ChapterID=90376353. [Último acceso: 17 06 2020].

[42] «mscbs.gob.es,» Ministerio de Sanidad, Consumo y Bienestar Social, 19 06 2020. [En línea]. Available: https://www.mscbs.gob.es/profesionales/hcdsns/areaRecursosSem/snomed-ct/snomedHCD.htm.

[43] S. P. ALONSO, R. N. CEBRIÁN y M. D. P. SANMILLÁN., «mscbs.gob.es,» Subdirección General de Información Sanitaria e Innovación. , [En línea]. Available: https://www.mscbs.gob.es/estadEstudios/estadisticas/normalizacion/CIE10/UT_MANUAL_DIAG_2016_prov1.pdf. [Último acceso: 19 06 2020].

[44] Y. M. F. F. Hassan Montero, «Guía de Evaluación Heurística de sitios web,» 2003. [En línea]. Available: http://www.nosolousabilidad.com/articulos/introduccion_usabilidad.htm. [Último acceso: 16 08 2020].

[45] J. Nielsen, «Ten Usability Heuristics,» 1994. [En línea]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/. [Último acceso: 17 08 2020].

[46] «fesemi.org,» [En línea]. Available: https://www.fesemi.org/publicaciones/revista-clinica/noticias. [Último acceso: 17 08 2020].

[47] «portalesmedicos.com,» [En línea]. Available: https://www.portalesmedicos.com/. [Último acceso: 20 08 2020].