
Asistente web interactivo para la simplificación de textos a Lectura Fácil



Trabajo de Fin de Grado
Curso 2020–2021

Autores
Javier Sesé García
Estefanía Ortega Ávila

Directoras
Raquel Hervás Ballesteros
Susana Bautista Blasco

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

Asistente web interactivo para la simplificación de textos a Lectura Fácil

Trabajo de Fin de Grado en Ingeniería de Computadores
**Departamento de Ingeniería de Software e Inteligencia
Artificial**

Autores
Javier Sesé García
Estefanía Ortega Ávila

Directoras
Raquel Hervás Ballesteros
Susana Bautista Blasco

Convocatoria: *Junio 2021*

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

11 de junio de 2021

Agradecimientos

En primer lugar queremos agradecer a nuestras tutoras Raquel y Susana por su gran labor a la hora de ayudarnos a enfocar este TFG aportándonos ideas y formas de abordar las diferentes partes de la memoria.

A mi familia y, especialmente, a Eduardo que sin ellos no habría llegado a donde estoy y a los que estaré eternamente agradecidos.

Estefanía

A mi familia y a mi chica Sandra, por darme apoyo todos estos años en situaciones en las que necesitaba apoyo y ánimos.

Javier

Resumen

Debemos tener en cuenta que, en nuestra sociedad, hay cerca de un 30 % de personas con dificultades de comprensión lectora y de aprendizaje (dislexia, discapacidad intelectual, personas mayores, aquellas que estén iniciándose en el idioma...), convirtiéndose así la Lectura Fácil (LF) en un aliado esencial que permite la accesibilidad de este sector de la población a la información y a la cultura.

En muchas ocasiones aquella persona encargada de transformar textos a LF debe realizar manualmente la tarea de adaptación de textos, lo que supone un gran esfuerzo en detrimento de la eficiencia, que podría aumentar si se dispusiera de una aplicación o herramienta capaz de convertir este trabajo manual en otro más rápido y eficiente.

Surge así la necesidad de abordar el problema en este TFG llamado “Asistente web interactivo para la simplificación de textos”, diseñando e implementando un asistente web que tiene la finalidad de ayudar al usuario editor a adaptar textos a LF, ya sean descriptivos, narrativos, periodísticos, etc..., de una manera interactiva.

El usuario editor tendrá a su disposición diversas funcionalidades para hacer posible la adaptación, llevando a cabo simplificaciones léxicas en el texto mediante identificación de palabras que puedan conllevar más dificultad para el lector y reemplazándolas por otras más sencillas. Además, permitirá intercambios sintácticos en las frases, así como la supresión de palabras o adición de información en el texto para una mejor comprensión del mismo. Finalmente, el usuario editor obtendrá un borrador del texto final y podrá visualizar el resultado del texto una vez haya sido adaptado, pudiendo hacer sobre él modificaciones adicionales. En todo momento, el encargado de editar texto tendrá el control absoluto sobre la adaptación, pero apoyado por el asistente para realizar más rápidamente ciertas simplificaciones.

Con esta aplicación pretendemos favorecer la comprensión y el aprendizaje por parte del lector con dificultades lectoras acercándole a una igualdad social y cultural, ya que le permite captar de una manera más accesible las ideas que un texto quiera trasmitir.

Palabras clave

Procesamiento Lenguaje Natural, Lectura Fácil, simplificación de texto, API REST, NIL-WS-API.

Abstract

We must take into account that, in our society, there are about 30% of people with reading comprehension and learning difficulties (dyslexia, intellectual disability, elderly people, those who are starting to learn the language...), making Easy-to-Read (ER) an essential ally that allows the accessibility of this sector of the population to information and culture.

On many occasions, the person in charge of transforming texts into ER must manually perform the task of adapting texts, which is a greaworkt effort to the detriment of efficiency, which could be increased if an application or tool capable of converting this manual work into a faster and more efficient one was available.

Thus arises the need to address the problem in this work called “Interactive web assistant for text simplification”, designing and implementing a web assistant that aims to help the editor user to adapt texts to ER, whether descriptive, narrative, journalistic, etc..., in an interactive way.

The editor user will have at his disposal several functionalities to make the adaptation possible, carrying out lexical simplifications in the text by identifying words that may be more difficult for the reader and replacing them with simpler ones. In addition, it will allow syntactic exchanges in the sentences, as well as the deletion of words or addition of information in the text for a better understanding of the text. Finally, the editor user will obtain a draft of the final text and will be able to visualize the result of the text once it has been adapted, being able to make additional modifications. At any moment, the editor text will have full control over the adaptation, but supported by the application to make certain simplifications more quickly.

With this application we intend to promote understanding and learning by readers with reading difficulties, bringing them closer to a social and

cultural equality, since it allows them to grasp in a more accessible way the ideas that a text wants to convey.

Keywords

Natural Language Processing, Easy-to-Read, text simplification, REST API, NIL-WS-API.

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
1. Introduction	5
1.1. Motivation	5
1.2. Goals	6
1.3. Document structure	6
2. Estado del arte	9
2.1. Lectura Fácil	9
2.1.1. Un poco de historia	11
2.1.2. Destinatarios de la Lectura Fácil	11
2.1.3. ¿Cómo se identifican los textos de Lectura Fácil?	12
2.1.4. Pautas a seguir para la elaboración de Lectura Fácil	13
2.1.5. Niveles de adaptación a Lectura Fácil	14
2.1.6. Tareas para la simplificación de Texto	15

2.2.	Movimientos y asociaciones de Lectura Fácil	16
2.3.	Proyectos y materiales adaptados a la Lectura Fácil	18
2.3.1.	Guía en Lectura Fácil sobre los servicios del banco . .	18
2.3.2.	Guía de uso del Metro de Madrid en Lectura fácil . .	18
2.3.3.	Guía y plano accesible del Museo del Prado	18
2.3.4.	Noticias Fácil	18
2.3.5.	Proyecto de Lectura Fácil sobre COVID-19	19
2.4.	Programas y aplicaciones de Lectura Fácil	19
2.4.1.	DyTECTIVE para dislexia	19
2.4.2.	Léelo Fácil Educ. - Gallego	20
2.4.3.	Simplext	20
2.4.4.	“Frente al aislamiento. Nos conectamos”	21
2.4.5.	CAPITO	21
2.4.6.	SIMPATICO	22
2.4.7.	Proyecto FIRST	23
2.4.8.	Wheris	24
3.	Herramientas y tecnologías utilizadas	27
3.1.	Flask	27
3.2.	spaCy	28
3.3.	NIL-WS-API	30
3.4.	Postman	31
3.5.	PyCharm	32
4.	Asistente web interactivo para la simplificación de textos a Lectura Fácil	35

4.1.	Requisitos del asistente web interactivo para la simplificación de textos	35
4.2.	Introducción de texto y selección de frases	36
4.3.	Adaptaciones sobre frases	38
4.3.1.	Palabras complejas	40
4.3.2.	Intercambio de partes en el árbol de dependencias . .	41
4.3.3.	Sinónimos	42
4.3.4.	Eliminar partes en el árbol de dependencias	45
4.3.5.	Definiciones respecto a una palabra	45
4.3.6.	Resultado de la adaptación	47
5.	Implementación	51
5.1.	Arquitectura	51
5.1.1.	Ficheros del entorno	52
5.1.2.	Servicios web externos	53
5.1.3.	Librería spaCy	56
5.2.	Implementaciones del servidor	59
5.3.	Implementaciones de la aplicación web	60
5.4.	Extensibilidad del asistente web interactivo	71
6.	Conclusiones y Trabajo Futuro	75
6.1.	Conclusiones	75
6.2.	Trabajo futuro	76
6.	Conclusions and Future Work	79
6.1.	Conclusions	79
6.2.	Future work	80

7. Trabajo individual	81
7.1. Estefanía Ortega Ávila	81
7.2. Javier Sesé García	83
Bibliografía	85

Índice de figuras

2.1.	Texto original de Don Quijote de la Mancha	10
2.2.	Texto LF de Don Quijote de la Mancha	10
2.3.	Beneficiarios de la Lectura Fácil	12
2.4.	Logo de Lectura Fácil que cumplen las normas de la IFLA . .	13
2.5.	Logotipo europeo de Lectura Fácil	13
2.6.	Aplicación Dytective para dislexia	20
2.7.	Aplicación Léelo Fácil	21
2.8.	Simplext	22
2.9.	Aplicación “Frente al aislamiento. Nos conectamos”	23
2.10.	CAPITO	24
2.11.	SIMPATICO	24
2.12.	Open Book	25
2.13.	Wheris	26
3.1.	Árbol de dependencias.	29
3.2.	Algunos de los servicios que ofrece NIL-WS-API referidas a palabras.	31
3.3.	Petición hecha con Postman	32

3.4. IDE PyCharm	33
4.1. Interfaz inicial del asistente.	37
4.2. Panel lateral de introducción de texto.	38
4.3. Frases partiendo del resumen.	39
4.4. Frases partiendo del texto completo.	39
4.5. Árbol de dependencias.	40
4.6. Operaciones que podemos efectuar sobre los elementos del árbol de dependencias	41
4.7. Borrador del texto final.	42
4.8. Palabras complejas resaltadas en el árbol.	42
4.9. Elección de dos palabras para el intercambio.	43
4.10. Árbol de dependencias después del intercambio.	43
4.11. Resultado en el borrador del texto final tras el intercambio.	43
4.12. Lista de sinónimos de una palabra seleccionada	44
4.13. Interfaz de edición de sinónimos	44
4.14. Ventana de diálogo de reemplazo de sinónimo.	45
4.15. Resultado de reemplazo del sinónimo tanto en el árbol de dependencias como en el borrador	45
4.16. Árbol antes la eliminación de una palabra.	46
4.17. Árbol tras la eliminación de una palabra.	46
4.18. Borrador del texto final tras la eliminación de una palabra.	46
4.19. Listado de las definiciones	47
4.20. Glosario adjunto al borrador del texto final.	47
4.21. Botón Ver resultado junto al borrador del texto final.	48
4.22. Panel con el resultado final del texto adaptado	49

5.1.	Estructura del proyecto en Flask	52
5.2.	Servicio REST cliente/servidor	53
5.3.	Diagrama de la arquitectura REST del asistente web	55
5.4.	Petición para comprobar si una palabra es compleja	56
5.5.	Petición que devuelve una lista de definiciones	57
5.6.	Petición que devuelve una lista de sinónimos	58
5.7.	Diagrama de flujo de nuestro algoritmo BFS	66
5.8.	Creación y extracción de la primera sublistas del array “sentenceArray”	67
5.9.	Creación y extracción de la segunda sublistas del array “sentenceArray”	68
5.10.	Inserción de ambas sublistas en el array “sentenceArray”	68
5.11.	Resultado del array “sentenceArray” tras el intercambio.	69

Capítulo 1

Introducción

“La lectura no da al hombre sabiduría; le da conocimientos”
— William Somerset Maugham

1.1. Motivación

Hoy en día, el ser humano tiene multitud de formas de favorecer su aprendizaje. Una de ellas es la lectura. En el mundo en el que vivimos podemos ejercerla prácticamente a través de diversos medios, bien sea mediante libros, redes sociales, televisión o prensa. Es un derecho que cualquier persona debe tener a su alcance. De ese modo, un lector puede relacionar palabras, símbolos, imágenes o números dentro de su mente, y así aprender.

Se dan circunstancias que hacen que este proceso de lectura no sea tan trivial, sino que se necesita una “ayuda” para que sea más accesible comprender la información que se transmite. Es el caso de aquellas personas que presentan algún tipo de discapacidad cognitiva, edad avanzada o no está familiarizados con el lenguaje por cualquier motivo, provocando una barrera entre ellos y la lectura.

Superar esta barrera es el principal objetivo que persigue la Lectura Fácil (LF), favoreciendo la accesibilidad a estas personas por medio de textos adaptados de manera que una lectura que perciban como compleja de comprender se convierta en texto que transmita, de una manera más simple, la misma idea facilitando así la comunicación entre la lectura y el lector.

La adaptación manual de textos o documentos a Lectura Fácil es muy costosa y lenta. Teniendo en cuenta que la aparición de nueva información

(noticias, blogs, redes sociales...) está en constante crecimiento, se necesitaría mantener el mismo ritmo de adaptación a LF. Sin embargo, resulta difícil elaborar manualmente un texto que cumpla con las características de la LF.

Vivimos en un mundo rodeado de avances tecnológicos, teniendo a nuestra disposición un gran número de dispositivos o herramientas. Cada vez hay más iniciativas que hacen uso de ellas para poder superar esas barreras y hacer posible la lectura a todo el mundo, adaptándolas a formatos accesibles.

Así pues, surge la idea del desarrollo de una aplicación que aúna funcionalidades que cumplan con las características y acciones de la LF, proporcionando al usuario editor una ayuda adicional, minimizando su esfuerzo cuando tiene que adaptar un texto a LF.

1.2. Objetivos

La herramienta pone al servicio del editor del texto una serie de funcionalidades y acciones de forma visual e interactiva que ayuden a convertir textos originales a otros con un lenguaje más claro y conciso y de manera rápida, además de permitirle realizar ajustes manuales en todo momento.

Así, el objetivo principal es ayudar a personas facilitando la adaptación de textos a Lectura Fácil con el uso de una aplicación interactiva que les permite hacer transformaciones sintácticas y léxicas mediante el uso de técnicas para el procesamiento del lenguaje natural.

Con esta herramienta pretendemos que la labor del usuario editor sea menos tediosa y cada vez menos manual, permitiendo mayor fluidez en la adaptación de textos.

1.3. Estructura del documento

Hemos seguido una serie de pasos para el desarrollo de esta memoria, la cuál se divide en los siguientes capítulos:

- **Capítulo 1** (Introducción, motivación y estructura del documento): en este capítulo, traducido también al inglés, hacemos una pequeña introducción para poder entender el problema de las dificultades de lectura de la cuál surge este TFG, la motivación y la estructura con explicaciones acerca de lo que vamos a exponer en cada uno de los capítulos.

- **Capítulo 2** (Estado del arte): en este capítulo, explicamos que es la Lectura Fácil, cómo surge, a quién va dirigida, su identificación, pautas, niveles y tareas en una adaptación. También hablaremos de asociaciones, aplicaciones y materiales adaptados a Lectura Fácil.
- **Capítulo 3** (Herramientas y tecnologías utilizadas): en este capítulo hablaremos de las tecnologías que hemos usado para el desarrollo de la aplicación.
- **Capítulo 4** (Asistente web interactivo para la simplificación de textos a Lectura Fácil): en este capítulo se describirán los requisitos necesarios que debe cumplir la aplicación web. Además, haremos un recorrido por las diferentes interfaces de la aplicación y los detalles de su uso.
- **Capítulo 5** (Implementación): en este capítulo detallaremos la arquitectura en la que está basada nuestro asistente, así como se han implementado los diferentes servicios web externos y librerías tanto de la parte del servidor como de la aplicación.
- **Capítulo 6** (Conclusiones y trabajo futuro): en este capítulo, también en inglés, haremos unas valoraciones finales acerca del asistente y damos algunas ideas sobre ciertos desarrollos futuros que se podrían abarcar.
- **Capítulo 7** (Trabajo individual): en este capítulo mostramos las aportaciones realizadas por cada uno de los miembros durante el proyecto.

Chapter 1

Introduction

1.1. Motivation

Nowadays, human beings have a multitude of ways to promote learning. One of them is reading. In the world in which we live we can exercise it practically through various media, either through books, social networks, television or the press. It is a right that everyone should have within their reach. That way, a reader can relate words, symbols, images or numbers within his or her mind, and thus learn.

There are circumstances that make this reading process not so trivial, but that require “help” to make it more accessible to understand the information being conveyed. This is the case of those people who have some kind of cognitive disability, advanced age or are unfamiliar with the language for whatever reason, causing a barrier between them and written text.

Overcoming this barrier is the main objective pursued by Easy-to-Read (ER), favoring accessibility to these people by means of adapted texts so that a reading that they perceive as complex to understand becomes a text that transmits, in a simpler way, the same idea, thus facilitating communication between the text and the reader.

The manual adaptation of texts or documents to Easy-to-Read is very costly and time-consuming. Taking into account that the appearance of new information (news, blogs, social networks...) is constantly growing, it would be necessary to maintain the same pace of adaptation to ER. However, it is difficult to manually produce a text that meets the characteristics of ER.

We live in a world surrounded by technological advances, having at our disposal a large number of devices or tools. There are more and more initiatives that make use of them to overcome these barriers and make reading possible for everyone, adapting them to accessible formats.

Thus, comes the idea of developing an application that combines functionalities that meet the characteristics and actions of the ER, providing the editor user with additional help, minimizing his effort when he has to adapt a text to ER.

1.2. Goals

The tool provides to the editor user with a series of functionalities and actions in a visual and interactive way capable to convert original texts into others with a clearer and more concise language and in a quicker way, as well as allowing him to make manual adjustments at any time.

Thus, the main objective is to help people by facilitating the adaptation of texts to Easy-to-Read with the use of an interactive application that allows them to make syntactic and lexical transformations through the use of natural language processing techniques.

With this tool we intend to make the work of the editor user less tedious and less manual, allowing greater fluidity in the adaptation of texts.

1.3. Document structure

We have followed a series of steps for the development of this report, which is divided into the following chapters:

- **Chapter 1** (Introduction, motivation and structure of the document): in this chapter, also translated into English, we make a small introduction to understand the problem of reading difficulties from which this dissertation arises, the motivation and structure with explanations about what we are going to expose in each of the chapters.
- **Chapter 2** (State of the art): in this chapter, we explain what Easy-to-Read is, how it arises, to whom it is addressed, its identification, guidelines, levels and tasks in an adaptation. We will also talk about associations, applications and materials adapted to Easy Reading.

- **Chapter 3** (Tools and technologies used): in this chapter we will talk about the technologies we have used for the development of the application.
- **Chapter 4** (Interactive web assistant for the simplification of texts to Easy-to-Read): in this chapter we will describe the necessary requirements that the web application must fulfill. In addition, we will go through the different interfaces of the application and the details of its use.
- **Chapter 5** (Implementation): in this chapter we will detail the architecture on which our wizard is based, as well as how the different external web services and libraries of both the server side and the application have been implemented.
- **Chapter 6** (Conclusions and future work): in this chapter, also in English, we will make some final evaluations about the wizard and give some ideas about some future developments that could be covered.
- **Chapter 7** (Individual work): in this chapter we show the contributions made by each of the members during the project.

Capítulo 2

Estado del arte

“La posibilidad de leer aporta a las personas una enorme confianza, permitiéndoles expandir sus opiniones y ejercer un control sobre sus propias vidas. Las personas pueden mediante la lectura compartir experiencias, pensamientos y experiencias y crecer como seres humanos”

— Directrices de la IFLA

En este capítulo trataremos sobre qué es lo que trata la Lectura Fácil, a qué público va dirigida, así como los diferentes movimientos sociales y proyectos que se han derivado a causa de ella.

2.1. Lectura Fácil

Alrededor del 30 % de la población tiene problemas para la lectura y comprensión de textos. Este pequeño porcentaje de personas, por cualquier razón física, psíquica o social, tienen dificultades para utilizar la lectura como medio de comunicación, información, formación u ocio. Esto supone un gran y difícil esfuerzo para la comprensión de textos. La lectura es un derecho fundamental que tenemos todas las personas de buscar y tener acceso a la información, y eliminar estas barreras es el principal objetivo de la Lectura Fácil (LF).

La LF es una forma de adaptar la información para que sea más sencilla de leer y entender por personas con dificultades lectoras. Es un método de adaptación con un lenguaje sencillo y claro, simplificación de texto, imágenes descriptivas y dibujos. Estas adaptaciones son adecuadas para aquellas

personas con discapacidad intelectual, con dificultad para el lenguaje, con alguna enfermedad y/o trastorno mental, en proceso de aprendizaje, etc.

A modo de ejemplo, en la Figura 2.1 se ve un pequeño fragmento de la novela de “Don Quijote De La Mancha” de Miguel de Cervantes Saavedra. No obstante en la Figura 2.2 podemos ver la adaptación de Mercedes Belinchón y Alberto Anula a Lectura Fácil. En este texto, aparece en negrita la palabra Hidalgo con su significado. También se muestra una imagen relacionada con el texto para una mejor comprensión.

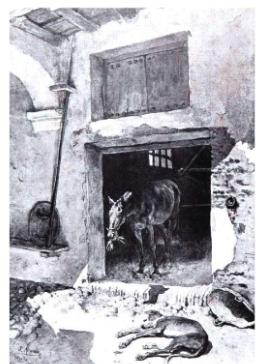
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocin flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lantejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas, con sus pantuflos de lo mismo, y los días de entresemana se honraba con su vellorí de lo más fino. Tenía en su casa una ama que pasaba de los cuarenta y una sobrina que no llegaba a los veinte, y un mozo de campo y plaza que así ensillaba el rocín como tomaba la podadera. Frisaba la edad de nuestro hidalgo con los cincuenta años. Era de complección recia, seco de carnes, enjuto de rostro, gran madrugador y amigo de la caza. Quieren decir que tenía el sobrenombre de «Quijada», o «Quesada», que en esto hay alguna diferencia en los autores que deste caso escriben, aunque por conjjeturas verisímiles se deja entender que se llamaba «Quijana». Pero esto importa poco a nuestro cuento: basta que en la narración dél no se salga un punto de la verdad.

Figura 2.1: Texto original de Don Quijote de la Mancha

En un pueblo de la Mancha,
de cuyo nombre no quiero acordarme,
vivió no hace mucho tiempo un **hidalgo**. ————— **Definición**

Nuestro hidalgo se llamaba Alonso Quijano.
Tenía muchos años y era muy delgado.
Don Alonso poseía un caballo flaco,
unas tierras y una casa muy grande.
El hidalgo vivía con su joven sobrina
y una criada.

Un hidalgo era
una persona
que había
heredado
tierras y vivía
sin tener que
trabajar.
Era un noble.



————— Ilustración

Figura 2.2: Texto LF de Don Quijote de la Mancha

2.1.1. Un poco de historia

El movimiento de la Lectura Fácil surgió en Suecia en 1968¹. En ese año se publicó el primer libro en Lectura Fácil y desde entonces hasta 1994 se crearon 330 obras, unas 15 y 20 nuevas cada año. Este movimiento se extendió a los países vecinos, Noruega y Finlandia.

En Noruega, por ejemplo, la iniciativa (proyecto) se denomina *Leser søker bok*² (Lector busca libro) que es una alianza de 20 organizaciones que incluyen editoriales y organizaciones de personas con discapacidad.

En 1988, en Bruselas, se crea la organización *Inclusion Europe*³, la alianza europea de organizaciones que trabajan por los derechos de las personas con discapacidad, en la que se agrupa a organizaciones y asociaciones de personas con discapacidad intelectual de 40 países europeos e Israel.

En 1998, se elabora la guía «*El camino más fácil: Directrices europeas para generar información de fácil lectura destinada a personas con discapacidad intelectual*»⁴ y se diseña un logotipo europeo de Lectura Fácil, para identificar todos los textos adaptados que siguen sus pautas.

En 2003, en España se crea la primera Asociación de Lectura Fácil en Barcelona⁵. Desde entonces, surgen diversas organizaciones e iniciativas a favor de la Lectura Fácil por toda España, donde hay más de 300 libros adaptados para aquellas personas con problemas de lectura.

2.1.2. Destinatarios de la Lectura Fácil

La Lectura Fácil se dirige a una serie de grupos con ciertas dificultades de compresión lectora. Algunos de ellos son los siguientes:

- Personas con dificultades en el aprendizaje (como la dislexia, etc.)
- Personas con poca cultura o escasa escolarización.
- Personas extranjeras o inmigrantes que no dominan bien la lengua española.
- Niños que necesitan un refuerzo en la lectura.

¹<https://www.lecturafacilextremadura.es/historia/>

²<https://lesersokerbok.no/english/>

³<http://www.inclusion-europe.eu/>

⁴<http://www.lecturafacil.net/media/resources/ILSMHcastell%C3%A0.pdf>

⁵<https://www.lecturafacil.net/es/>

- Personas sordas con dificultades en la comprensión.
- Personas mayores con trastornos mentales.
- Personas con hiperactividad y déficit de atención.
- Personas con discapacidad intelectual o del desarrollo (como autismo, afasia, etc.).

A modo representativo de todos los colectivos que necesitan de la LF se muestra la Figura 2.3. Los círculos representan a los grupos beneficiarios de LF, y el cuadrado la necesidad de la misma (Nomura et al., 2010).

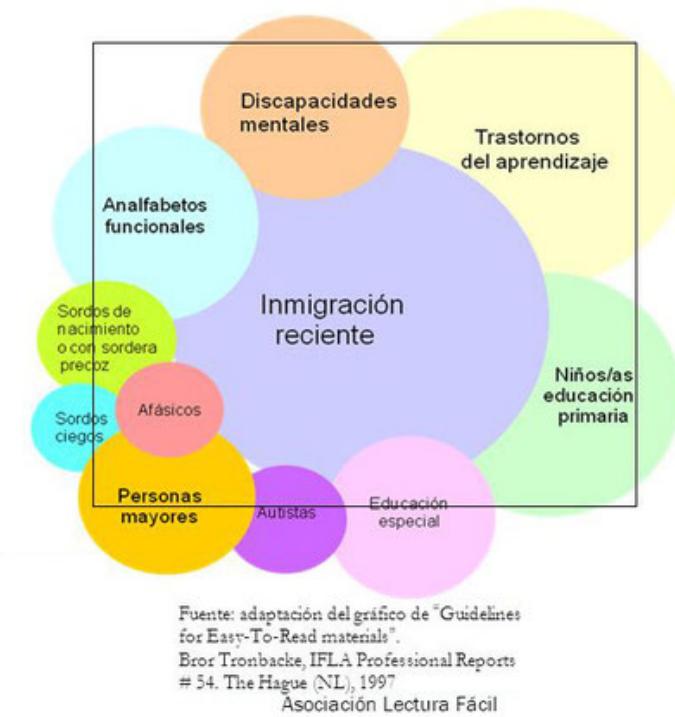


Figura 2.3: Beneficiarios de la Lectura Fácil

2.1.3. ¿Cómo se identifican los textos de Lectura Fácil?

Los textos adaptados a Lectura Fácil vienen identificados por dos tipos de logotipos. En la Figura 2.4 podemos ver el logo que la Asociación de Lectura Fácil otorga a los textos que se adaptan a las normas de la Federación Internacional de Asociaciones de Bibliotecarios y Bibliotecas (IFLA), del

inglés *International Federation of Library Associations and Institutions*⁶. Por otro lado, la Figura 2.5 muestra el logo utilizado por Inclusion Europe.



Figura 2.4: Logo de Lectura Fácil que cumplen las normas de la IFLA



Figura 2.5: Logotipo europeo de Lectura Fácil

2.1.4. Pautas a seguir para la elaboración de Lectura Fácil

Para hacer posibles las adaptaciones a LF de cualquier texto, se deben seguir una serie de directrices. Uno de los primeros documentos sobre como elaborar texto adaptado a Lectura Fácil fue publicado por la IFLA. Hay otro que fue elaborado por varias organizaciones de Inclusion Europe bajo el título “Información para todos” (Europe, 2019).

Algunas de las adaptaciones clasificadas por Plena Inclusión (García Muñoz, 2013) son:

- Ortografía:

⁶<https://www.ifla.org/ES>

- Evitar el uso de algunos signos de ortografía que dificulten la comprensión del texto.
- Evitar mayúsculas excepto cuando toca según las reglas ortográficas.
- Evitar signos de ortografía poco habituales (%,& , /, . .).
- Usar guión para los diálogos.
- Evitar los números romanos.
- Limitar el uso de la coma.

■ Gramática:

- Evitar estructuras complejas que puedan dificultar la comprensión.
- Usar voz activa, evitando siempre la voz pasiva y subjuntivo.
- No usar tiempos verbales complejos.
- Usar frases cortas. Escribir una idea por frase, es decir, separar cada idea por un punto o con una “y” en vez de una coma.
- Estructurar el texto de forma clara y coherente.

■ Léxico:

- Usar un lenguaje sencillo y directo.
- Evitar la jerga y los términos técnicos.
- Evitar abreviaturas.

■ Estilo:

- Usar la personalización, es decir, escribir de forma directa y personal.
- Usar una palabra por concepto.
- Incluir imágenes relacionadas con el texto.

2.1.5. Niveles de adaptación a Lectura Fácil

La Lectura Fácil no dispone de un estándar fijo, sino que se proponen distintos niveles ya que es imposible adaptar un texto de la misma manera para todas las personas con estas dificultades. Dicha adaptación se refiere tanto a texto como a imágenes, o cualquier otro elemento que se incorpore.

La IFLA establece tres niveles, semejantes tanto para obras originales en LF como para las adaptadas a LF:

- Primer nivel: es el más sencillo y simple con muchas imágenes y escaso texto, con una dificultad sintáctica baja.
- Segundo nivel: es intermedio, menos sencillo que el anterior con un vocabulario y expresiones que son conocidas por todos, fácil de seguir y comprender. En este nivel también se usan imágenes.
- Tercer nivel: es el más complejo, con textos más extensos, con palabras poco corrientes, con saltos en el tiempo y espacio. En este nivel hay pocas imágenes.

Esta clasificación se hace en base al usuario al que se dirija el texto adaptado.

2.1.6. Tareas para la simplificación de Texto

Muchas características de los textos se pueden modificar o transformar para hacerlo más legible y comprensible. El objetivo de las adaptaciones a Lectura Fácil es transformar oraciones complejas en otras más simples (Saggion, 2017).

Los métodos de simplificación deberían facilitar o agilizar la adaptación del texto disponible, haciendo posible el acceso de la información a personas con discapacidad cognitiva. Por lo general, los textos adaptados tendrían pérdida de información y un estilo más simple, aunque no necesariamente. Podemos valorar esto positivamente, siempre y cuando el resultado final pueda ser entendido por el lector objetivo.

Hay muchas características de los textos que pueden ser modificadas para hacerlo más legible y comprensible, incluyendo también el modo en el que se muestra. La simplificación del texto se suele basar en las siguientes cuatro tareas:

- Simplificación léxica: tiene como objetivo reemplazar las palabras difíciles por otras más fáciles (sinónimos) que se consideren mejores para comprender o leer, siempre y cuando el significado no quede alterado. Por ejemplo, “El automóvil es de color azul” por “El coche es de color azul”.
- Simplificación sintáctica: tiene como objetivo transformar frases u oraciones largas que contienen figuras sintácticas que son ilegible e incomprendibles, transformarlas en otras más simples y en forma activa (se debe evitar siempre que se pueda la forma pasiva). Por ejemplo, “China

se va de fiesta, que se esta recuperando del coronavirus” por “China se va de fiesta. China se esta recuperando del coronavirus”.

- Eliminar información: el objetivo es la reducción de frases u oraciones, manteniendo la información esencial, eliminando los detalles innecesarios que no añaden nada nuevo a la idea que se quiere transmitir. Por ejemplo, “Laura sacó a pasear a su perro, un San Bernardo, por el parque” por “Laura sacó a pasear a su perro por el parque”.
- Añadir información: el objetivo es aportar conocimiento extra que pueda ayudar al lector a comprender y aprender el significado de uno o varios términos que desconozca. Por ejemplo, “Mi vecino se compró un Ferrari” por “Mi vecino se compró un ferrari, un coche deportivo caro”.

Estas simplificaciones están relacionadas y en ocasiones se necesita la mezcla de ellas para mantener la coherencia y conseguir el texto final.

2.2. Movimientos y asociaciones de Lectura Fácil

A raíz de la importancia del movimiento Lectura Fácil, surgen en España una serie de movimientos con el fin de integrar y favorecer a las personas con dificultades en la comprensión lectora. Algunos de ellos son los siguientes:

- **Asociación de Lectura Fácil de Barcelona**⁷. Esta asociación fue creada en 2003, la primera en España del movimiento LF. Es una asociación sin ánimo de lucro que trabaja para hacer fácil el acceso la lectura, cultura e información a todas las personas, en especial a aquellas con dificultades en la lectura.
- **Asociación Lectura Fácil Extremadura**⁸. Es una asociación sin ánimo de lucro que trabaja a favor de la promoción, implantación y difusión de la LF. Buscan la obtención de la Igualdad para todos y cada uno de los ciudadanos de la Comunidad.
- **Fundación Ciudadanía (Extremadura)**⁹. Fundación sin ánimo de lucro, declarada de Utilidad Pública. Realiza aportaciones y estrategias en el ámbito de la Innovación Educativa y el Empleo, extendiendo su ámbito de actuación a todo el territorio español, poniendo énfasis en Extremadura y en su vocación europea y latinoamericana.

⁷<https://www.lecturafacil.net/es>

⁸<https://www.lecturafacilextremadura.es/>

⁹<https://www.fundacionciudadania.es/>

- **Dilee Lectura Fácil (Extremadura)**¹⁰. Es una empresa española, pionera en Extremadura, que pretende implantar y consolidar la Lectura Fácil en todos los ámbitos y sectores de la vida económica, socio-política, artística-cultural, educativa, etc.
- **Lectura Fácil Madrid**¹¹. Es una asociación creada en el año 2013 que tiene como finalidad lograr que todas las personas puedan participar de forma activa y responsable en la sociedad y hacer realidad la democracia lectora.
- **Cooperativa Altavoz (Madrid)**¹². Es una cooperativa formada por personas con discapacidad que trabaja para mejorar la autonomía de todas las personas, adaptando contenidos a lectura fácil.
- **Lectura Fácil Euskadi (Bilbao)**¹³. Es una asociación que pretende incentivar la creación, difusión y utilización de materiales en Lectura Fácil, a través de un programa de animación a la lectura dirigido a personas con dificultad lectora.
- **Lectura Fácil Castilla y León (Palencia)**¹⁴. Es una entidad sin ánimo de lucro que trabaja para difundir la Lectura Fácil como herramienta de conocimiento entre las personas con dificultades de comprensión lectora en Castilla y León.
- **Instituto de Lectura Fácil (Sevilla)**¹⁵. Es una organización social con el objetivo de reivindicar el derecho que tenemos a comprender la información que nos rodea, dar calidad de vida a la ciudadanía, en especial de los colectivos más vulnerables de la sociedad.
- **Plena Inclusión**¹⁶. Es una organización, formada por 17 federaciones autonómicas (Ceuta y Melilla también) y unas 900 asociaciones en toda España, que representa a las personas con discapacidad intelectual o del desarrollo. Defienden los derechos y fomentan la calidad de vida de personas con discapacidad intelectual o del desarrollo y su familia.
- **SOLCOM**¹⁷. Organización no gubernamental, independiente y orientada a dar asistencia legal para la solidaridad comunitaria de las personas con diversidad funcional y la inclusión social.

¹⁰<https://odsextremadura.es/dilee-lectura-facil-extremadura/>

¹¹<https://www.lecturafacilmadrid.com/>

¹²<http://altavozcooperativa.org/>

¹³<https://lecturafacileuskadi.net/>

¹⁴<http://www.lecturafacil.es/>

¹⁵<http://www.institutolecturafacil.org/>

¹⁶<https://www.plenainclusion.org/>

¹⁷<https://asociacionsolcom.org/>

2.3. Proyectos y materiales adaptados a la Lectura Fácil

Existen diversos proyectos que fomentan el uso de la Lectura Fácil. Así, en diversos ámbitos se han creado prospectos, manuales, guías, documentos y noticias para facilitar el acceso a la información a aquellos que lo necesiten.

2.3.1. Guía en Lectura Fácil sobre los servicios del banco

Plena inclusión Galicia ha publicado una guía en LF (Galicia, 2020), el 21 de julio de 2020, sobre cómo realizar operaciones básicas a través de los diferentes servicios que ofrece un banco, como crear una cuenta, controlar los movimientos, etc.

2.3.2. Guía de uso del Metro de Madrid en Lectura fácil

En esta guía adaptada a LF (Inclusión, 2019), tiene como objetivo contribuir a que las personas con discapacidad intelectual puedan moverse por la red del suburbano de forma autónoma. Fue publicada por Plena Inclusión en el año 2019 y ha sido revisada y actualizada en 2020.

2.3.3. Guía y plano accesible del Museo del Prado

Es la primera guía de Lectura Fácil (del Prado, 2020a) elaborada por el Museo del Prado con la colaboración de Plena Inclusión, ofreciendo una selección de 10 obras maestras ubicadas en el plano accesible (del Prado, 2020b) que le acompaña, para facilitar la visita de personas con discapacidad cognitiva. Ha recibido el premio CERMI.ES 2019 en la categoría de Accesibilidad Universal – Fundación Vodafone España.

2.3.4. Noticias Fácil

Noticias Fácil¹⁸ es un proyecto de Discapnet, creado en noviembre del 2013, en el marco del Plan Avanza del Ministerio de Industria, Comercio y Turismo de España. Es una plataforma interactiva de servicios y contenidos web adaptados a LF relacionados con la actualidad. Es una web de acceso libre, sin tener que ingresar ningún tipo de dato. Cuenta con 8 secciones:

¹⁸<http://www.noticiasfacil.es/>

¿Qué es Lectura Fácil?, noticias, biblioteca, agenda, encuestas y vocabulario. Este portal está pensado para que lo pueda leer todo el mundo, en un lenguaje sencillo y claro. Además, cualquier persona puede enviar noticias, algo de actualidad o de importancia en su vida.

2.3.5. Proyecto de Lectura Fácil sobre COVID-19

Un grupo de investigadoras de la Facultad de Psicología, junto con la Universidad Católica, con la colaboración de la Clínica Universidad de los Andes, ambas ubicadas en Santiago (Chile), y la casa de estudios, elaboran un documento sobre la COVID-19 en formato de LF (de los Andes, 2020), publicado el 4 de noviembre del 2020. El objetivo de este documento es que las personas con discapacidad cognitiva puedan comprender de manera más fácil la nueva enfermedad que hay en la actualidad.

2.4. Programas y aplicaciones de Lectura Fácil

En esta sección veremos algunas aplicaciones creadas por asociaciones, grupos y movimientos que trabajan la comprensión lectora para mejorar la lectura, practicar y aprender vocabulario.

2.4.1. Dytective para dislexia

Dytective para dixlexia (Rello, 2018) es una herramienta diseñada para niños que ayuda a mejorar las habilidades lectoescritoras con o sin dificultades de lectura y escritura, mientras se divierten jugando. Posee una serie de niveles personalizados para cada niño que hace que se superen día a día, realizando 4 retos semanales. Incluye también una prueba orientativa que detecta si tienes dificultades lecto-escritura. Es una aplicación gratuita disponible tanto para Android como para iOS. Para su descarga podemos visitar la página web <https://www.changedyslexia.org/>¹⁹ En la Figura 2.6 se muestran unos interfaces que se podrán encontrar en la aplicación.

¹⁹Última actualización el 24 de marzo del 2020.

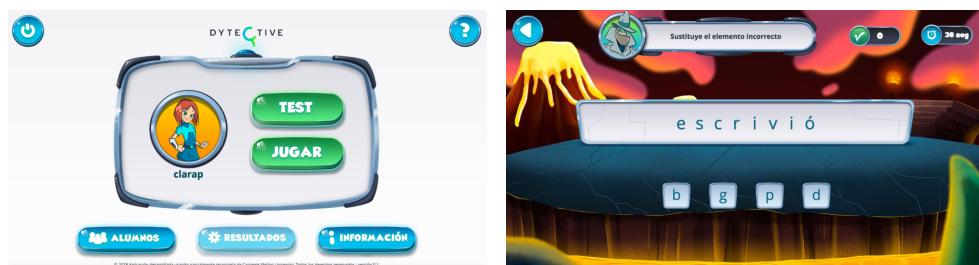


Figura 2.6: Aplicación Dytective para dislexia

2.4.2. Léelo Fácil Educ. - Gallego

Léelo Fácil Educ. - Gallego²⁰ es una aplicación que sirve para leer un libro adaptado a Lectura Fácil. Cuenta con dibujos, música y animaciones para un mejor entendimiento. Esta aplicación es un proyecto de FEAPS Confederación (ahora Plena Inclusión) publicada el 13 de abril del 2015. La aplicación tiene dos partes: obras de relevancia para consultar a modo educativo (Dos Leyendas de Bécquer) y obras para disfrutar como ocio (Novela de Jordi Sierra i Fabra). Actualmente se encuentra disponible un APK para su descarga. Este proyecto se ha quedado obsoleto. En la Figura 2.7 se puede ver un fragmento del libro “El rayo de luna” de Gustavo Adolfo Bécquer.

2.4.3. Simplext

Simplex (Saggion et al., 2011) es un proyecto del Ministerio de Industria, Energía y Turismo, con un presupuesto de más de 2,6 millones de euros. Este proyecto es un sistema automático para la transformación de cualquier tipo de textos a LF, reduciendo la complejidad léxica y sintáctica, permitiendo así una mejor comprensión de textos. Su objetivo es el desarrollo de una herramienta que sirva de apoyo para simplificar los textos. Esta herramienta detecta las palabras y oraciones complejas, convirtiéndolas en palabras más sencillas y oraciones más cortas. Es una herramienta sencilla, se copia el texto a adaptar y se simplifica automáticamente como se muestra en la Figura 2.8. En la 10^a edición de los premios BDigital a la Innovación Digital fue nominado y finalista. Para acceder a esta herramienta podemos visitar la página web de la demo de Simplext: <http://simplext.taln.upf.edu/>.

²⁰Instalación APK “Léelo Fácil” en <https://apkpure.com/es/l%C3%A9elo-f%C3%A1cil-educ-gallego/com.oneclick.ga.rayoluna>



Figura 2.7: Aplicación Léelo Fácil

2.4.4. “Frente al aislamiento. Nos conectamos”

“Frente al aislamiento, Nos conectamos”²¹ es una aplicación lanzada por Plena Inclusión en marzo del 2020 para dar respuesta a las necesidades de personas con discapacidad cognitiva durante la crisis del coronavirus. Se trata de una herramienta de información, participación y consulta.

Esta herramienta nos ofrece documentos, materiales, foros de consulta para preguntar dudas y agenda de actividades. Es una aplicación gratuita tanto para Android como para iOs. En la Figura 2.9 se puede ver a modo ilustrativo lo que nos ofrece esta aplicación.

2.4.5. CAPITO

CAPITO²² proviene del italiano cuyo significado es “lo entiendo”, creada por Atempo, empresa social que trabaja por la igualdad de las personas. Es una aplicación tanto para móvil, tablet o PC, disponible en lengua inglesa y alemana, que podemos ver en la Figura 2.10. Nos ofrecen principalmente

²¹ Descarga de la aplicación “Frente al aislamiento. Nos conectamos” en <https://my.yapp.us/ZNMC4A>

²² <https://www.capito.eu/en/>

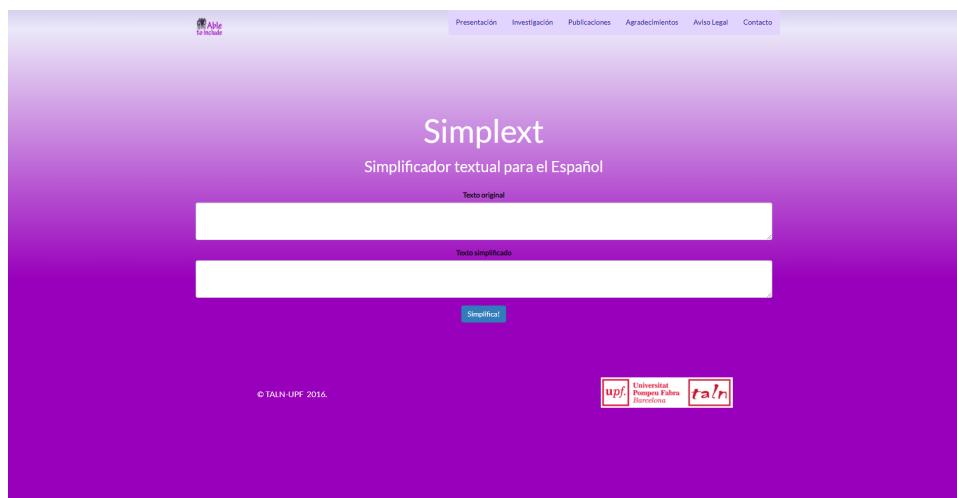


Figura 2.8: Simplext

traducciones en línea a un lenguaje fácil de entender. Dispone de 3 niveles de traducción: el A1 (breve y simple), el A2 (fácilmente comprensible) y el B1 (lenguaje coloquial), siendo el A2 y el B1 especialmente adecuados para personas con dificultades de aprendizaje y discapacidades. También podemos encontrar cursos online para el auto-aprendizaje de escritura a LF, talleres, etc. El idioma de enseñanza es el alemán.

2.4.6. SIMPATICO

SIMPATICO²³ es una plataforma cuyo objetivo es simplificar la comunicación entre los usuarios y las administraciones públicas mediante la tecnología, es decir, facilitando el acceso a diversos servicios relacionados con la administración de una manera digital más accesibles a aquellas personas con deterioros cognitivos.

Permite a los usuarios tener una interacción más fácil, adaptación del texto a su perfil, flujo de trabajo simplificado y personalizado y formularios web precargados con los datos personales del usuario.

Cuando algo no está claro para un determinado usuario, la simplificación del texto es a través de *Text Adaptation Engine*, sugiriendo cambios en el texto (léxico, sintáctico o semántico) y traducción, sinónimos y explicaciones.

Esta plataforma también pretende fomentar la participación del usuario en la administración pública, para que puedan publicar y resolver dudas sobre

²³<https://simpatico-project.com/>

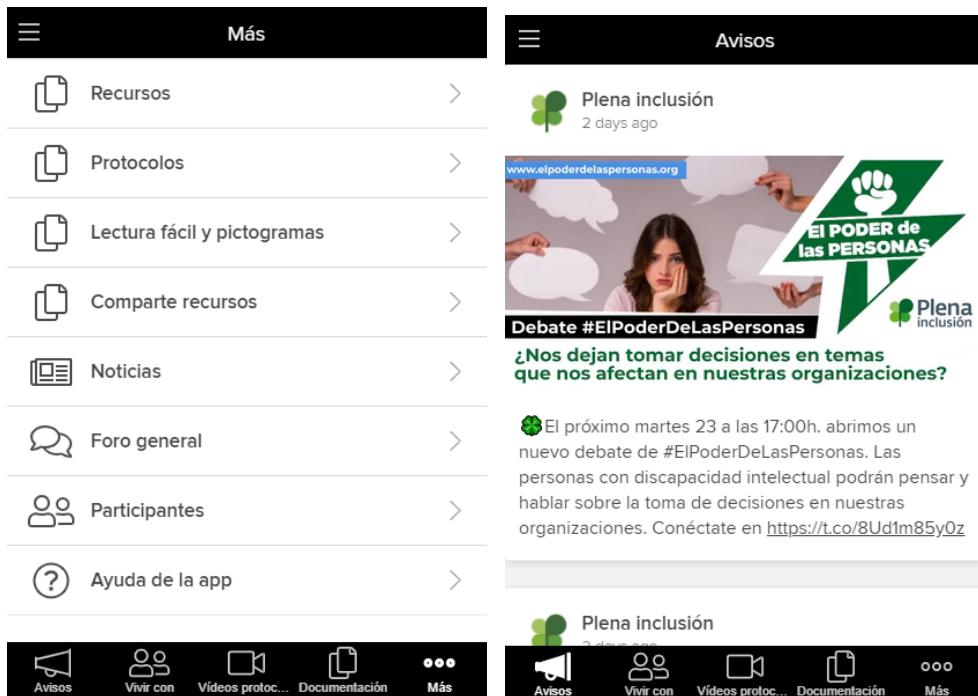


Figura 2.9: Aplicación “Frente al aislamiento. Nos conectamos”

trámites administrativos, entendiendo mejor de forma visual los procesos administrativos. Este proyecto se inició el 1 de marzo del 2016 y finalizó el 28 de febrero del 2019. En la Figura 2.11 podemos ver el sitio web del proyecto.

2.4.7. Proyecto FIRST

FIRST²⁴ es un proyecto europeo para el desarrollo de una herramienta multilingüe, llamada Open Book (Martín Valdivia et al., 2014), para la creación de contenidos accesibles para personas con autismo. Este proyecto empezó el 1 de octubre del 2009 y finalizó el 30 de septiembre del 2014. El proyecto tiene como objetivo el uso de las tecnologías del lenguaje, capaces de detectar y simplificar un contenido, para que pueda ser más fácil de comprender, así como el impacto que tendrá de mejora en la vida de esas personas.

A través de la herramienta online Open Book, que podemos ver en la Figura 2.12, se puede simplificar textos escritos en 3 idiomas (inglés, espa-

²⁴<http://www.openbooktool.net/>

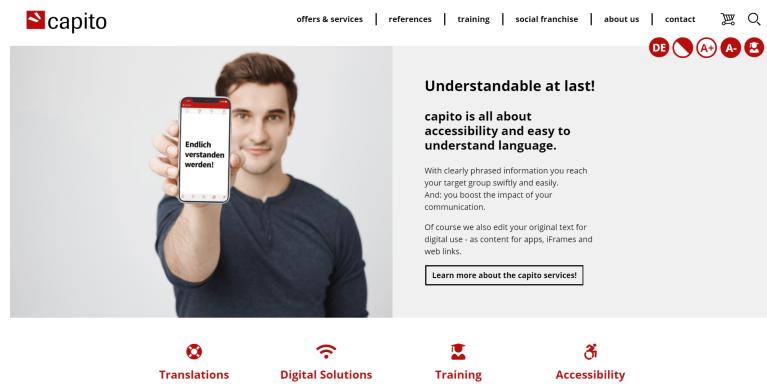


Figura 2.10: CAPITO



Figura 2.11: SIMPATICO

ñol y búlgaro), permitiendo una personalización y adaptación a cada tipo de usuario. La conversión se hace por la detección automática de carácter lingüístico en aquello que puede dificultar la comprensión, de tal forma que el resultado final no se vea alterado respecto al contenido original. Tiene dos perfiles: modo cuidador, que ofrece más posibilidades de revisión, edición y corrección automática de texto, y modo usuario final (persona con autismo).

2.4.8. Wheris

Wheris²⁵ es una aplicación gratuita para móvil disponible tanto para iOS y Android. Creada en 2016, y galardonada como mejor aplicación de nueva

²⁵Se puede descargar en http://www.tematicblog.com/WHERIS_Web/

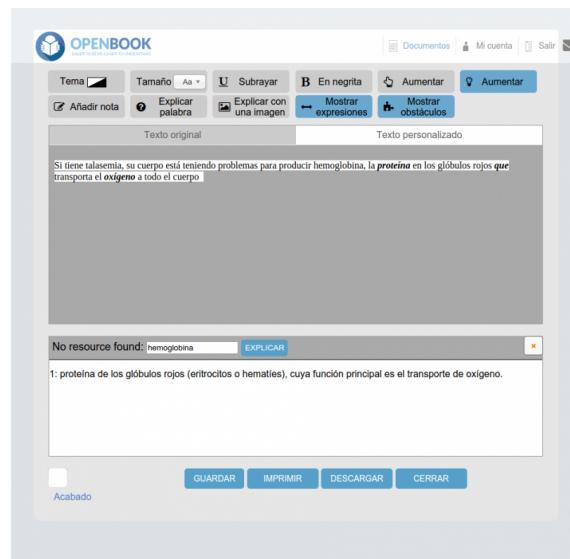


Figura 2.12: Open Book

creación con el premio “Best New App”. Esta aplicación detecta códigos invisibles en todos los impresos y en las reproducciones de audio y vídeo a tu alrededor. Permite a los usuarios con discapacidad visual o dificultades de compresión lectora acceso a información que, en su versión original, no es accesible para este colectivo. En la Figura 2.13 se puede ver como es la aplicación.



Figura 2.13: Wheris

Capítulo 3

Herramientas y tecnologías utilizadas

“Siempre llega una nueva herramienta. La tecnología es neutral, depende de cómo se use”

— Rick Smolan

En este capítulo hablaremos sobre las distintas herramientas y tecnologías utilizadas para el desarrollo del trabajo. También expondremos los motivos por los cuales hemos decidimos usar unas tecnologías frente a otras.

3.1. Flask

Flask¹ es un “micro” framework minimalista escrito en Python con la finalidad de facilitar el desarrollo de aplicaciones web y APIs. Desarrollado por Armin Ronacher a partir de 2010, actualmente es uno de los frameworks más populares para implementar en Python. Este framework² está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD (licencia utilizada para los sistemas operativos, y que tiene menos restricciones en comparación con otras licencias como la GPL).

El término “micro” hace referencia a que Flask trae únicamente por defecto las herramientas necesarias para crear una aplicación web básica. Si éstas no fueran suficientes hay un conjunto muy grande de extensiones (plu-

¹<https://openwebinars.net/blog/que-es-flask/>

²<https://leanmind.es/es/desarrollo/backend/python/>

gins) que se pueden instalar fácilmente proporcionando así libertad a los desarrolladores ya que contiene muy poco código predefinido.

Algunas de las características de Flask por las que decidimos desarrollar nuestra aplicación web con este framework son las siguientes:

- Agilidad, rapidez y facilidad en la instalación y configuración, que a diferencia de otros framework como Django, son más complejos y difíciles de usar.
- Compatible con Python.
- Incluye un servidor web de desarrollo. No es necesaria una infraestructura con un servidor web para probar las aplicaciones, sino que de forma sencilla se puede levantar un servidor web e ir viendo los resultados que se van obteniendo.
- Cuenta con un depurador. Si en algún momento obtenemos un error en el código que se está desarrollando, se puede depurar ese error y ver los valores de las variables.
- Buen manejo de rutas. Se controlan todas las peticiones que hacen los clientes y se tienen que determinar qué ruta está accediendo el cliente para ejecutar el código necesario.
- Cuenta con documentación extensa para el desarrollo de aplicaciones.

3.2. spaCy

El Procesamiento del Lenguaje Natural (PLN)³ es el campo de conocimiento de la Inteligencia Artificial y la lingüística que intenta replicar la facultad del lenguaje humano, es decir, comunica las máquinas con las personas mediante el uso de lenguas naturales, como el español, el inglés o el chino.

Existe una amplia variedad de herramientas informáticas para el PLN en diversos idiomas, como NLTK, spaCy, Scikit-learn, entre otras. En nuestro caso hemos decidido usar spaCy por su fácil uso, rapidez y precisión a la hora de realizar análisis sintácticos.

spaCy⁴ es una librería de código abierto que permite construir aplicaciones de PLN escrita en Python. Actualmente es considerada una de las

³<https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/>

⁴Para más información acceder a <https://spacy.io/>

mejores herramientas para el PLN.

Esta librería contiene los datos lingüísticos y los algoritmos que necesitará para procesar textos en lenguaje natural. Proporciona objetos que ayudan a representar elementos de texto, como oraciones y palabras. Estos objetos tienen una serie de atributos que representan las características lingüísticas. Además, nos ofrece visualizadores integrados para generar un gráfico de la estructura sintáctica de una oración. Puede ser usada para extraer información, para sistemas de comprensión del lenguaje natural o para el pre-procesado de texto para deep learning.

spaCy es capaz de realizar las siguientes tareas PLN:

- **Tokenization:** Divide una oración en tokens, donde cada token representa cada una de las palabras que componen dicha oración.
- **Part-of-speech (POS) Tagging:** asigna a cada token una etiqueta gramatical, designando su categoría gramatical (sustantivo o nombre, adjetivo, pronombre, verbo, etc.).
- **Dependency Parsing:** analiza una oración para establecer la dependencia gramatical de las palabras “principales” y otras que modifican las principales, describiendo la relación entre ellas. El resultado del análisis es la creación de un árbol de dependencias, así como el etiquetado de dependencia en cada palabra como muestra la Figura 3.1.

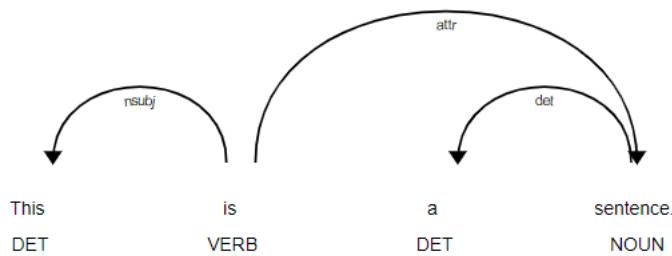


Figura 3.1: Árbol de dependencias.

- **Lemmatization:** permite obtener el lema correspondiente de una palabra. Por ejemplo, el lema de “dije” es “decir” y el de “ratas” es “rata”.
- **Sentence Boundary Detection:** detecta el límite de una oración.

- **Named Entity Recognition:** busca, ubica y clasifica palabras en el texto en categorías predefinidas como los nombres de personas, organizaciones, ubicaciones, fechas, etc.
- **Entity Linking:** desambiguado de entidades textuales a identificadores únicos en una base del conocimiento.
- **Similarity:** compara palabras, intervalos de texto y documentos para saber qué grado de similitud tienen entre ellos.
- **Text Classification:** asigna categorías o etiquetas a un documento completo, o a partes de un documento.
- **Rule-based Matching:** busca secuencias de tokens basada en sus textos y anotaciones lingüísticas, similar a expresiones regulares.

3.3. NIL-WS-API

NIL-WS-API⁵ es una API del grupo NIL (Interacción natural basada en el lenguaje) implementada en el proyecto IDiLyCo, según el estándar OpenAPI 3.0.1.

El grupo NIL⁶ surge en 2005 de un grupo de investigadores y profesionales cuyo objetivo principal es el desarrollo de tecnologías basadas en el lenguaje natural, para poder usarse en aplicaciones prácticas y ser aplicadas en la vida real.

Esta API ofrece una gran cantidad de servicios a nuestra disposición. Entre ellos podemos encontrar servicios para:

- Palabras: servicios que devuelven información relativa a una palabra.
- Textos: servicios que devuelven información relativa a un texto o a un grupo de palabras.
- Pictogramas: servicios que devuelven información relativa a un pictograma.
- Simplificación de texto: servicios para la simplificación léxica.
- Resúmenes: ofrece una simplificación sintáctica a partir del resumen de un texto.

⁵Para más información acceder a <https://holstein.fdi.ucm.es/nil-ws-api/>

⁶<http://nil.fdi.ucm.es/>

- Emociones: servicios para obtener las emociones de una palabra, una frase o un texto.

De todos estos servicios, para nuestro asistente hemos hecho uso de los que se refieren a las palabras (algunos de ellos se muestran en la Figura 3.2), que detallaremos en el capítulo 5, sección 5.1.2.

The screenshot shows a Postman collection named "Palabra". The title "Servicios de palabras" is displayed above the collection. A sub-section "Estos servicios devuelven información relativa a una palabra." is present. Below this, there are seven API endpoints listed in a table:

GET	Path	Description
/palabra/{palabra}/es_sencilla	Servicio para comprobar si una palabra es sencilla.	
/palabra/{palabra}/sinonimos	Dada una palabra en castellano, devuelve una lista de sinónimos.	
/palabra/{palabra}/antonimos	Dada una palabra en castellano, devuelve una lista de antónimos.	
/palabra/{palabra}/definiciones	Dada una palabra en castellano, devuelve una lista de definiciones.	
/palabra/{palabra}/traducciones	Dada una palabra en castellano, devuelve una lista de traducciones posibles al inglés.	
/palabra/{palabra}/conversion_a_facil	Dada una palabra en castellano, devuelve un sinónimo que sea considerado palabra fácil.	
/palabra/{palabra}/pictograma	Dada una palabra, devuelve el pictograma que le corresponde.	

Figura 3.2: Algunos de los servicios que ofrece NIL-WS-API referidas a palabras.

3.4. Postman

Postman⁷ surgió originariamente como una extensión para el navegador Google Chrome. Se trata de una herramienta dirigida a desarrolladores web que permite realizar peticiones HTTP a cualquier API. Es muy útil a la hora de programar y hacer pruebas, para comprobar el correcto funcionamiento de nuestros desarrollos.

Además de hacer peticiones a servicios, que es su objetivo principal, nos ofrece un conjunto de funcionalidades que nos ayudarán a organizar las peticiones en colecciones, hacer y automatizar pruebas, mantener equipos sincronizados y crear Mocks de APIs.

Es una herramienta gratuita en su versión básica pero además ofrece dos posibilidades de pago que mejoran las características.

En nuestro caso, hemos usado esta herramienta para probar todos los

⁷<https://openwebinars.net/blog/que-es-postman/>

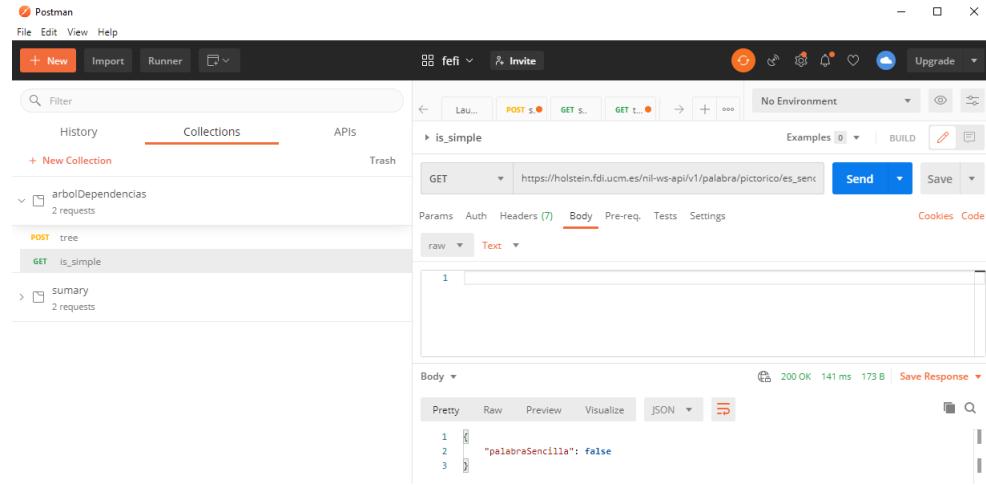


Figura 3.3: Petición hecha con Postman

endpoints y APIs, y comprobar que todos los datos recibidos eran los correctos o esperados.

3.5. PyCharm

PyCharm⁸, desarrollado por la empresa JetBrains, es un entorno de desarrollo (IDE) para desarrollar principalmente código en Python. Cuenta con un depurador y un intérprete que nos ayudarán a saber o conocer los posibles errores del código en tiempo real.

Además ofrece un soporte para HTML que incluye sintaxis y resaltado de errores, formateo de acuerdo con el estilo del código, validación de la estructura, finalización del código, etc.

Este IDE lo hemos usado en el desarrollo de la aplicación web (Figura 3.4)

⁸<https://blog.desdelinux.net/pycharm-un-entorno-de-desarrollo-para-python/>

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and a search bar. Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main area has tabs for 'src - views.py', 'views.py', 'funciones.js', 'vo.py', and 'home.html'. On the left is the 'Project' tool window showing a hierarchical view of the project structure under 'src': static (css, img, js), templates (home.html, views.py, vo.py), and External Libraries. The central code editor displays Python code for a route handler:

```
40 @app.route('/isSimple', methods=['POST'])
41
42 def simple():
43     data=request.get_json()
44     data=data['text']
45     nlp=spacy.load("es_core_news_sm")
46     textList=[]
47     for word in data:
48         textList.append(word['text'])
49     text=' '.join(textList)
50     doc=nlp(text)
51     position=0
52     simpleList=[]
53     simple=False
54     print('doc ', len(doc))
55     for token in doc:
56         pos=token.pos_
57         if (pos != "ADP" and pos != "DET"
58             and pos != "CONJ" and pos != "NUM")
```

The bottom status bar shows 'Problems: Current File 121 Project Errors' with two error messages: 'Type: In word 'NILWS':30' and 'Type: In word 'sencilla':31'.

Figura 3.4: IDE PyCharm

Asistente web interactivo para la simplificación de textos a Lectura Fácil

“La tecnología hace posible lo que antes era imposible. El diseño hace que sea real”
— Michael Gagliano

En este capítulo se describirán los requisitos necesarios que debe cumplir nuestro asistente para un correcto funcionamiento. Además mostraremos un recorrido por las diferentes vistas que encontrará el usuario editor durante el uso del mismo.

4.1. Requisitos del asistente web interactivo para la simplificación de textos

Para ayudar a los distintos destinatarios descritos en el capítulo 2, sección 2.1.2, uno de los primeros pasos ha sido valorar qué funcionalidades era necesario tener definidas antes del desarrollo del asistente. Éste deberá servir de apoyo al editor al realizar una serie de operaciones sobre un texto, artículo, relato, etc. para facilitar la adaptación del mismo a Lectura Fácil.

Hay que tener en cuenta que la tarea de adaptación de textos puede ser costosa y tediosa en muchas ocasiones. Para evitarlo, gracias a este asistente, el usuario tendrá a su alcance funcionalidades que tienen como objetivo una

simplificación y mejor comprensión del texto, teniendo, además, la posibilidad de realizar modificaciones manuales.

Tal y como se hizo referencia en el capítulo 2 (ver sección 2.1.6), es imprescindible poder hacer transformaciones y modificaciones en nuestro texto para poder hacerlo más accesible a un determinado público. Así pues, el asistente permite realizar una serie de operaciones que facilitará la adaptación de cualquier texto a LF permitiendo al usuario en todo momento realizar los ajustes manuales necesarios. Estas operaciones son las siguientes:

- Hacer un resumen del texto introducido, favoreciendo una simplificación y eliminación de información redundante o poco relevante.
- Detección de palabras que puedan ser más complejas de cara al lector.
- Facilitar el reemplazo de palabras por sinónimos que puedan ser más sencillos y accesibles al lector derivando en una simplificación léxica.
- Posibilidad de eliminar palabras, suprimiendo información no esencial que al lector no le aporte valor en la lectura.
- Permitir al editor añadir definiciones de términos que pueden ser añadidas como parte del texto final, aportando al lector información adicional sobre aquello que puede no serle familiar.
- Intercambio sintáctico de una o varias palabras de orden de manera que el editor pueda colocarlas en favor de una mejor comprensión por parte del lector.

Una vez definidas las funcionalidades necesarias que incluye nuestro asistente, veremos en las siguientes secciones el flujo de trabajo que realizará el editor, donde se describirán las funcionalidades ya mencionadas anteriormente.

4.2. Introducción de texto y selección de frases

El usuario editor debe de navegar a la dirección. <https://holstein.fdi.ucm.es/tfg/2021/simpli/> para comenzar a adaptar su texto. La primera pantalla que visualizará el usuario editor será como la que muestra la Figura 4.1. En la parte superior de la pantalla encontrará una barra de navegación mientras que en la parte central de la misma se muestra el título del asistente, una descripción y un botón (**Introducir texto**).



Figura 4.1: Interfaz inicial del asistente.

Al pulsar en dicho botón se mostrará un panel lateral con las siguientes opciones:

- **Texto original:** un panel donde se introducirá el texto que queremos adaptar (ver Figura 4.2).
- **Botón Resumen:** si pulsamos esta opción se mostrará una vista con una serie de frases resultado del resumen del texto original previamente introducido (véase Figura 4.3). Este botón permanecerá inactivo hasta que se introduzca texto.
- **Botón Texto completo:** si, por el contrario, pulsamos esta opción se mostrará el texto completo dividido en frases, también selecciona-

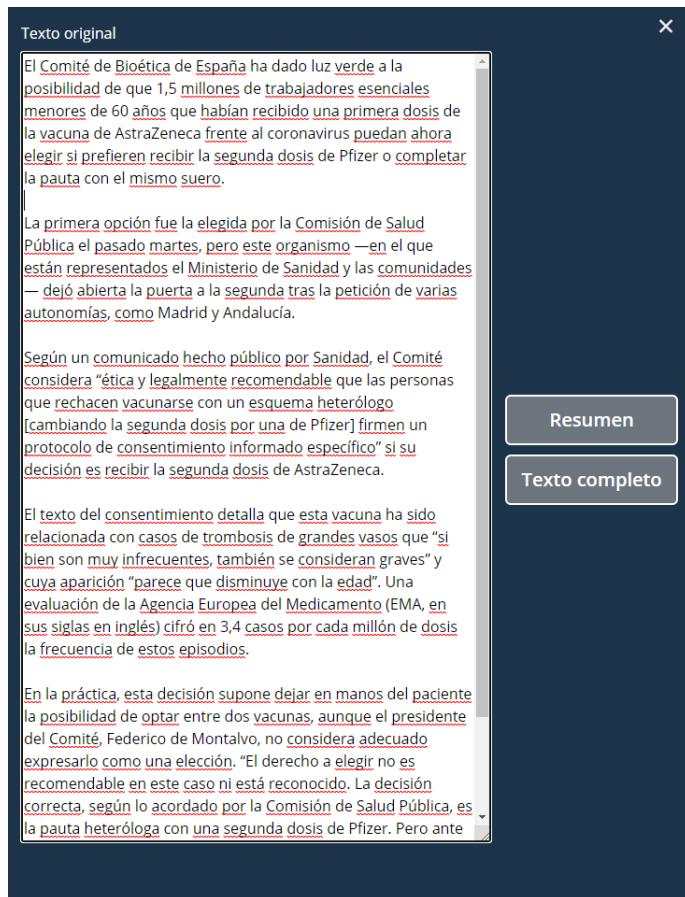


Figura 4.2: Panel lateral de introducción de texto.

bles (Figura 4.4). Al igual que el anterior botón también permanecerá inactivo hasta que el usuario introduzca texto.

Recordemos que en la Figura 4.1 sólo contábamos con la pestaña de “Inicio” en la barra de navegación, la cual cambia añadiendo una nueva pestaña “**Texto original**”. Esta pestaña nos permitirá consultar siempre el texto original que hemos introducido y cambiar en cualquier punto del flujo la opción elegida detallada anteriormente (resumen o texto completo).

4.3. Adaptaciones sobre frases

Al clicar sobre una frase como las que se muestran en las Figuras 4.3 y 4.4, se cambiará a una vista donde aparecerá un árbol de dependencias

Selecciona una frase para adaptar

Puedes borrar las frases que no quieras en tu texto final pulsando sobre (X)

Frases del texto original:

El Comité de Bioética de España ha dado luz verde a la posibilidad de que 1,5 millones de trabajadores esenciales menores de 60 años que habían recibido una primera dosis de la vacuna de AstraZeneca frente al coronavirus puedan ahora elegir si prefieren recibir la segunda dosis de Pfizer o completar la pauta con el mismo suero.

La primera opción fue la elegida por la Comisión de Salud Pública el pasado martes, pero este organismo —en el que están representados el Ministerio de Sanidad y las comunidades— dejó abierta la puerta a la segunda tras la petición de varias autonomías, como Madrid y Andalucía.

Pero ante el riesgo de que una persona deje sin completar la vacunación, por lo que esto supone para la colectividad y la propia persona, si manifiesta su rechazo a esta pauta, entonces recomendamos que se le pueda ofrecer la segunda dosis de AstraZeneca”, sigue De Montalvo.

Figura 4.3: Frases partiendo del resumen.

Selecciona una frase para adaptar

Puedes borrar las frases que no quieras en tu texto final pulsando sobre (X)

Frases del texto original:

El Comité de Bioética de España ha dado luz verde a la posibilidad de que 1,5 millones de trabajadores esenciales menores de 60 años que habían recibido una primera dosis de la vacuna de AstraZeneca frente al coronavirus puedan ahora elegir si prefieren recibir la segunda dosis de Pfizer o completar la pauta con el mismo suero.

La primera opción fue la elegida por la Comisión de Salud Pública el pasado martes, pero este organismo —en el que están representados el Ministerio de Sanidad y las comunidades— dejó abierta la puerta a la segunda tras la petición de varias autonomías, como Madrid y Andalucía.

Según un comunicado hecho público por Sanidad, el Comité considera “ética y legalmente recomendable que las personas que rechacen vacunarse con un esquema heterólogo [cambiando la segunda dosis por una de Pfizer] firmen un protocolo de consentimiento informado específico” si su decisión es recibir la segunda dosis de AstraZeneca.

El texto del consentimiento detalla que esta vacuna ha sido relacionada con casos de trombosis de grandes vasos que “si bien son muy infrecuentes, también se consideran graves” y cuya aparición “parece que disminuye con la edad”.

Una evaluación de la Agencia Europea del Medicamento (EMA, en sus siglas en inglés) cifró en 3,4 casos por cada millón de dosis la frecuencia de estos episodios.

En la práctica, esta decisión supone dejar en manos del paciente la posibilidad de optar entre dos vacunas, aunque el presidente del Comité, Federico de Montalvo, no considera adecuado expresarlo como una elección.

“El derecho a elegir no es recomendable en este caso ni está reconocido.”

La decisión correcta, según lo acordado por la Comisión de Salud Pública, es la pauta heteróloga con una segunda dosis de Pfizer.

Figura 4.4: Frases partiendo del texto completo.

(véase Figura 4.5), que describe la estructura sintáctica de dicha frase, y que le servirá al usuario editor a la hora de adaptarla de una manera interactiva.

En dicho árbol encontramos las unidades léxicas separadas de la frase, las cuales podremos pulsar y aplicar sobre ellas una serie de operaciones que se encuentran como botones en la parte inferior izquierda. Junto a estos, aparece una breve explicación de la funcionalidad (ver operaciones que podemos efectuar en la Figura 4.6) que detallaremos en el capítulo 5, sección 5.3.

En la parte superior de la pantalla el editor podrá visualizar en todo momento la frase previamente elegida y un enlace (**Volver a las frases**) para volver al listado de frases, en el caso de que se quiera elegir otra o retomar alguna que ya haya sido modificada.

En la parte inferior derecha de la vista, tenemos el borrador del texto final (Ver Figura 4.7) donde se puede visualizar la transformación de la frase seleccionada (en negrita) habiendo realizado, o no, las diferentes operaciones.

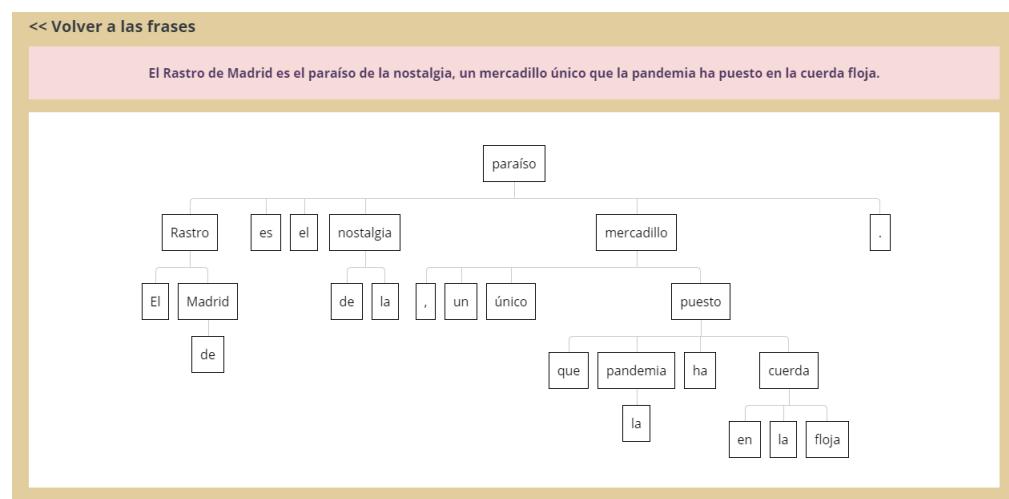


Figura 4.5: Árbol de dependencias.

A continuación, describiremos en qué consisten, en qué situaciones y cómo el usuario debe utilizar cada una de las operaciones que se pueden efectuar.

4.3.1. Palabras complejas

Esta funcionalidad es de utilidad cuando el usuario editor desee conocer aquellos términos de la frase que pueden ser susceptibles de ser reemplazados por sinónimos que sean más asequibles de comprender para el lector o expli-

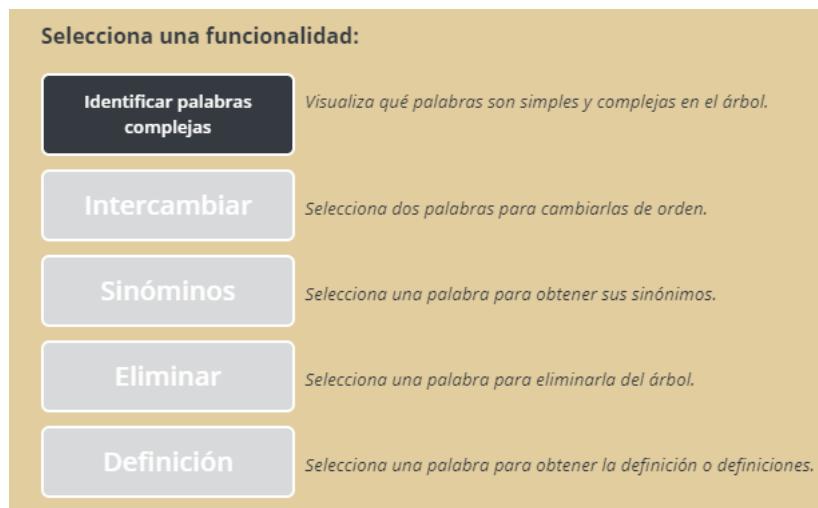


Figura 4.6: Operaciones que podemos efectuar sobre los elementos del árbol de dependencias

cados en más detalle. El elemento visual que se encargará de esta función es el botón **Identificar palabras complejas**.

Una vez hagamos clic en este botón nuestro árbol de dependencias mostrará las palabras complejas. Como podemos ver en la Figura 4.8, aparece una leyenda en la parte superior izquierda del árbol, indicando el color en el que se colorean. Una vez pulsado, el texto del botón cambiará a **Desactivar palabras complejas** para poder volver a la vista original del árbol de dependencias.

4.3.2. Intercambio de partes en el árbol de dependencias

Puede ocurrir que el usuario editor desee modificar el orden sintáctico de la frase. Por ejemplo, se puede dar la situación de que la segunda parte de una frase sea más importante y por ende se quiera que el lector ponga más atención a esa parte poniéndola al principio.

Para que esta funcionalidad se active es necesario tener seleccionadas dos palabras del árbol. Ambas palabras cambiarán de orden incluyendo también aquellas que dependan de las mismas. Se puede ver un ejemplo del intercambio de dos palabras en el árbol de dependencias en las figuras 4.9 y 4.10). El texto resultante quedaría como se muestra en la Figura 4.11. En cualquier momento, el usuario puede editar el texto para solucionar la concordancia minúsculas/mayúsculas en el caso de que sea necesario.

Borrador del texto final

Copiar texto al portapeles

El Rastro de Madrid es el paraíso de la nostalgia , un mercadillo único que la pandemia ha puesto en la cuerda floja . La última apuesta para revitalizarlo es recuperar la Feria de desembalajes en la plaza del General Vara de Rey, una tradición que tuvo su origen en los años 70, y que ahora se asienta el primer y el tercer sábado del mes para dinamizar las ventas y que no se encasillen en los clásicos domingos. Estamos limitados en espacio con control del número de personas que entran, pero cuando pase esta catástrofe se animará más", dice optimista Manolo González, presidente de la Asociación de comerciantes del Rastro, que junto con el Ayuntamiento de Madrid han lanzado esta castiza iniciativa: los sábados del Rastro. No solo incluye más de 38 expositores de antigüedades y colecciónismo en la Feria de desembalajes, engloba también un circuito gastronómico con una selección de locales y gran variedad de actividades para las familias en la Glorieta de Puerta de Toledo, que arrancarán el próximo sábado.

Ver resultado

Figura 4.7: Borrador del texto final.

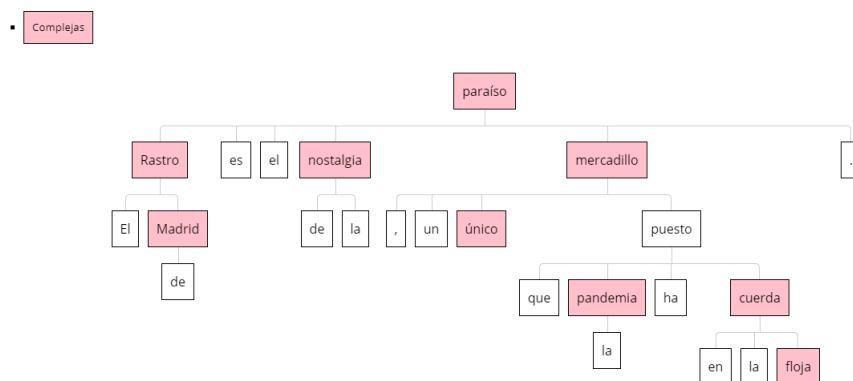


Figura 4.8: Palabras complejas resaltadas en el árbol.

4.3.3. Sinónimos

En ocasiones el usuario se verá en la necesidad de realizar una simplificación léxica sustituyendo una palabra por un sinónimo más conocido. Podrá

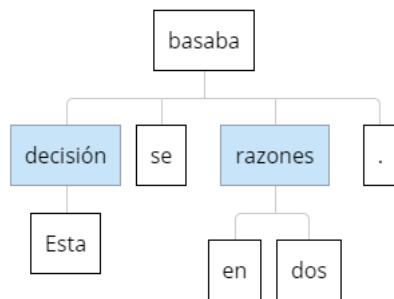


Figura 4.9: Elección de dos palabras para el intercambio.

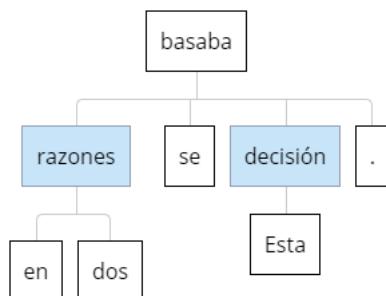


Figura 4.10: Árbol de dependencias después del intercambio.

a dejar de utilizarlo definitivamente en menores de 60 años y reservar este suero para la franja de edad de 60 a 69 años. **en dos razones se basaba Esta decisión**. La primera es que la mayoría de los casos de trombo detectados lo habían sido en personas más jóvenes, aunque este es un ~~asunto~~ según un estudio. La segunda es que la ~~ba~~ más frecuencia de los

Figura 4.11: Resultado en el borrador del texto final tras el intercambio.

hacer uso de ella pulsando sobre el botón (**Sinónimos**), siempre y cuando se haya seleccionado una palabra del árbol previamente. Una vez pulsado, se muestran, si existen, todos los sinónimos de la palabra elegida. En caso de que la palabra no tenga sinónimos aparecerá un texto informando de que carece de ellos. Si los tiene, se mostrará un listado de estos indicando cuáles son sencillos y cuáles no (ver opción sinónimos en la Figura 4.12). En esta figura mostramos, por ejemplo, los sinónimos complejos y sencillos de la palabra “dosis”.

Llegados a este punto, se podrá seleccionar uno de ellos o bien haciendo doble clic o bien editándolo si fuese necesario para un significado coherente

*Selecciona una palabra y pulsa enter para
continuar (o doble click) o pulsa el botón de editar
para cambiar la palabra*

Sinónimos de 'dosis':

Sinónimos complejos

porcion	<input checked="" type="checkbox"/>
dosificacion	<input checked="" type="checkbox"/>
cuota	<input checked="" type="checkbox"/>

Sinónimos sencillos

parte	<input checked="" type="checkbox"/>
cantidad	<input checked="" type="checkbox"/>

Figura 4.12: Lista de sinónimos de una palabra seleccionada

con la frase (concordancia en género, número, persona y conjugación), y posteriormente pulsar “Enter” para el cambio (véase la Figura 4.13). Aquí se edita el sinónimo sencillo “parte” por “partes”, ya que por el contexto de la frase es conveniente adaptarlo a plural.

*Selecciona una palabra y pulsa enter para
continuar (o doble click) o pulsa el botón de editar
para cambiar la palabra*

Sinónimos de 'dosis':

Sinónimos complejos

porcion	<input checked="" type="checkbox"/>
dosificacion	<input checked="" type="checkbox"/>
cuota	<input checked="" type="checkbox"/>

Sinónimos sencillos

partes	<input checked="" type="checkbox"/>
cantidad	<input checked="" type="checkbox"/>

Figura 4.13: Interfaz de edición de sinónimos

Al hacer doble clic o “Enter”, si lo hemos editado, se mostrará una ventana de diálogo (Aceptar y Cancelar) para asegurar si es realmente la acción que queremos realizar (ver Figura 4.14). En el caso de que se elija “Aceptar”, tanto en el árbol de dependencias como en el borrador del texto final se reemplazará la palabra (Figura 4.15). En caso contrario, no habrá modificación alguna.

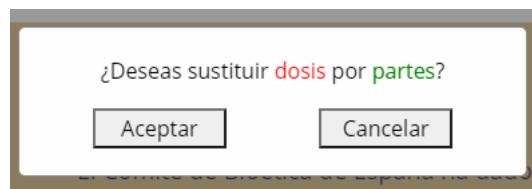


Figura 4.14: Ventana de diálogo de reemplazo de sinónimo.

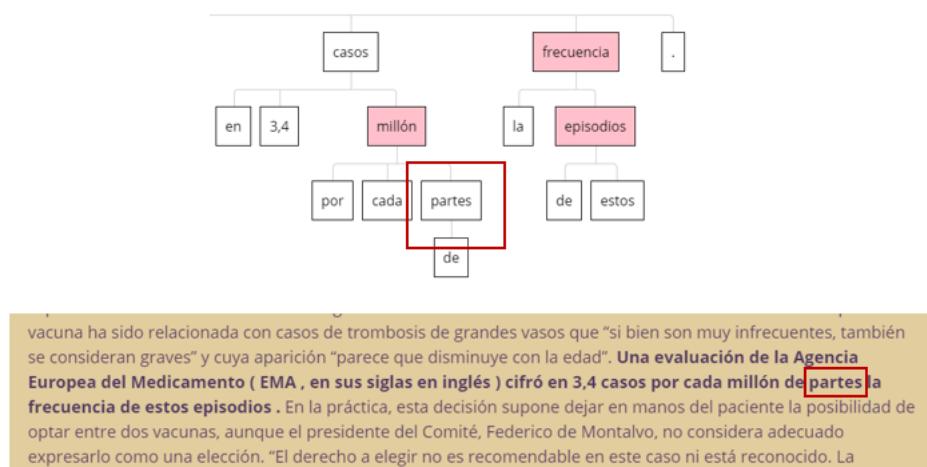


Figura 4.15: Resultado de reemplazo del sinónimo tanto en el árbol de dependencias como en el borrador

4.3.4. Eliminar partes en el árbol de dependencias

El usuario editor puede encontrarse con frases que no requieren de ciertas palabras para que se capte la esencia de las mismas por parte del lector. Es por ello por lo que tiene la opción de eliminar términos a su disposición. Es necesario seleccionar una palabra del árbol para que esta funcionalidad pueda llevarse a cabo y se suprime tanto a la palabra como aquellas que dependan de la misma. En las Figuras 4.16 y 4.17 vemos la palabra seleccionada, en este caso "acordado", que queremos eliminar, y el resultado de su eliminación en el árbol y en el texto del borrador final. El resultado de la frase modificada lo podemos ver en la Figura 4.18.

4.3.5. Definiciones respecto a una palabra

Gracias a esta funcionalidad, el encargado de adaptar texto podrá nutrir el texto final con definiciones de uno o varios términos que considere que

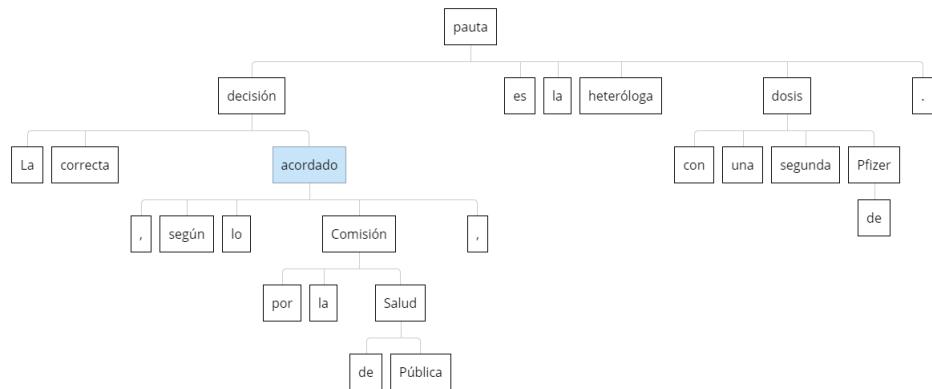


Figura 4.16: Árbol antes la eliminación de una palabra.

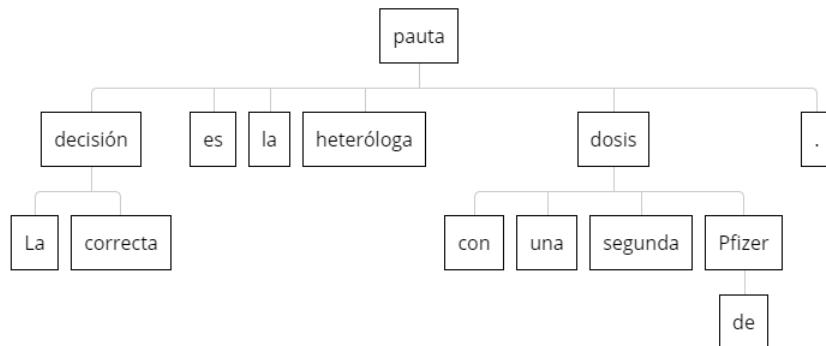


Figura 4.17: Árbol tras la eliminación de una palabra.

expresarlo como una elección. “El derecho a elegir no es recomendable en este caso ni está reconocido. **La decisión correcta es la pauta heteróloga con una segunda dosis de Pfizer**. Pero ante el riesgo de que una persona deje sin completar la vacunación, por lo que esto supone para la colectividad y la propia persona, si manifiesta su rechazo a esta pauta,

Figura 4.18: Borrador del texto final tras la eliminación de una palabra.

puedan precisar de alguna aclaración.

Seleccionando una palabra en el árbol, el botón (**Definición**) se activará. Al pulsar sobre este se mostrará un listado con todas las acepciones, si las tiene, de la palabra seleccionada. Vemos, por ejemplo, las de la palabra “correcta” en la Figura 4.19. Haciendo clic en una de ellas, ésta se adjuntará a modo de glosario como parte del texto final, sirviendo de apoyo a la comprensión del mismo (ver Figura 4.20). En caso de que no posea definiciones (por ejemplo, nombres propios) aparecerá un texto informando que carece

de ellas.

Selecciona una o varias definiciones para adjuntarla al final del texto como glosario

Definición de 'correcta':

adj. Que esta libre de errores o defectos, conforme a las reglas.

[Persona] educada, atenta, cortes.

tr. Rectificar, enmendar los errores o defectos de alguien o algo: Tambien prnl.

Advertir, amonestar, reprender.

Figura 4.19: Listado de las definiciones

caso ni está reconocido. **La decisión correcta es la pauta heteróloga con una segunda dosis de Pfizer**. Pero ante el riesgo de que una persona deje sin completar la vacunación, por lo que esto supone para la colectividad y la propia persona, si manifiesta su rechazo a esta pauta, entonces recomendamos que se le pueda ofrecer la segunda dosis de AstraZeneca", sigue De Montalvo. "Se ha delegado la responsabilidad de la recomendación a una comisión ética que no ha estado a la altura a la hora de desmarcarse de una decisión eminentemente política y considerar realmente la evidencia científica disponible, y las implicaciones éticas de aceptar una combinación de vacunas no avalada por ninguna entidad regulatoria", afirma Bassat.

Glosario:

correcta : adj. Que esta libre de errores o defectos, conforme a las reglas.

Figura 4.20: Glosario adjunto al borrador del texto final.

4.3.6. Resultado de la adaptación

Cuando hayamos considerado que el texto esté adaptado a nuestras necesidades, podemos visualizarlo haciendo clic en el botón (**Ver resultado**), el cuál lo encontramos en la parte inferior derecha del borrador del texto final (ver Figura 4.21). Al pulsarlo, se desplegará un panel editable en la parte derecha con el mismo texto que obtuvimos en el borrador (este panel lo podemos ver en la Figura 4.22). En este momento, el usuario editor puede terminar de modificar el contenido del texto, como por ejemplo, incluir la acepción del glosario dentro de la frase como aclaración.

Como hemos visto en la Figura 4.21, también contamos con un botón **Copiar al portapapeles** que nos da la opción de copiar el texto del borrador para poder introducirlo en cualquier herramienta externa más fácilmente.

Borrador del texto final

 Copiar texto al portapeles

Según un comunicado hecho público por Sanidad, el Comité considera "ética y legalmente recomendable que las personas que rechacen vacunarse con un esquema heterólogo [cambiando la segunda dosis por una de Pfizer] firmen un protocolo de consentimiento informado específico" si su decisión es recibir la segunda dosis de AstraZeneca. El texto del consentimiento detalla que esta vacuna ha sido relacionada con casos de trombosis de grandes vasos que "si bien son muy infrecuentes, también se consideran graves" y cuya aparición "parece que disminuye con la edad". Una evaluación de la Agencia Europea del Medicamento (EMA, en sus siglas en inglés) cifró en 3,4 casos por cada millón de dosis la frecuencia de estos episodios. En la práctica, esta decisión supone dejar en manos del paciente la posibilidad de optar entre dos vacunas, aunque el presidente del Comité, Federico de Montalvo, no considera adecuado expresarlo como una elección. "El derecho a elegir no es recomendable en este caso ni está reconocido. **La sentencia correcta es la pauta heteróloga con una segunda dosis de Pfizer**. Pero ante el riesgo de que una persona deje sin completar la vacunación, por lo que esto supone para la colectividad y la propia persona, si manifiesta su rechazo a esta pauta, entonces recomendamos que se le pueda ofrecer la segunda dosis de AstraZeneca", sigue De Montalvo.

Glosario:

correcta : adj. Que esta libre de errores o defectos, conforme a las reglas.

Ver resultado

Figura 4.21: Botón Ver resultado junto al borrador del texto final.

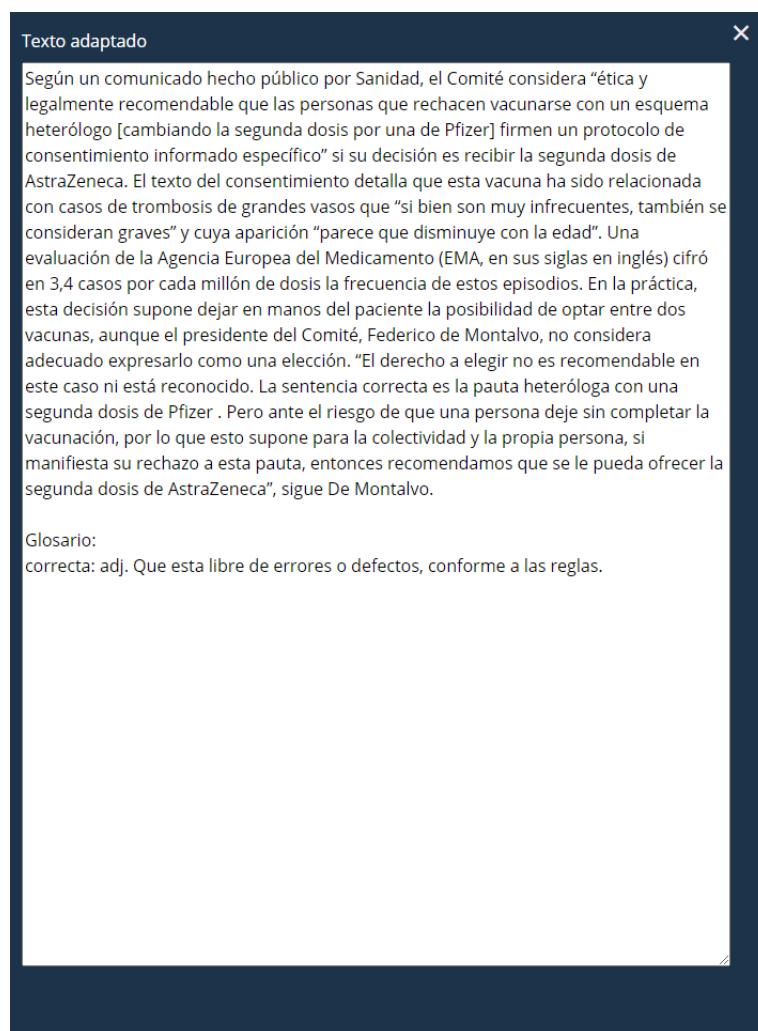


Figura 4.22: Panel con el resultado final del texto adaptado

Capítulo 5

Implementación

“Puede que tengas grandes ideas en la cabeza, pero lo que importa es la acción. Una idea, si no se lleva a cabo, no producirá ninguna manifestación, ni resultados ni recompensas”
— Miguel Ruiz

En este capítulo hacemos una descripción detallada sobre la arquitectura de nuestro asistente. Hablaremos también sobre la implementación de las distintas funcionalidades que se ha llevado a cabo tanto en la parte del servidor como en la aplicación en sí.

5.1. Arquitectura

La estructura de nuestro proyecto está soportada sobre un entorno Flask, teniendo el nivel de directorios que se muestra en la Figura 5.1. En las posteriores subsecciones describimos la parte de implementación correspondiente al servidor, explicando tanto los servicios externos utilizados como la librería spaCy, fundamentales para llevar a cabo las funcionalidades que requiere el asistente.

A continuación, se explican la función de cada uno de los ficheros que componen la aplicación web.

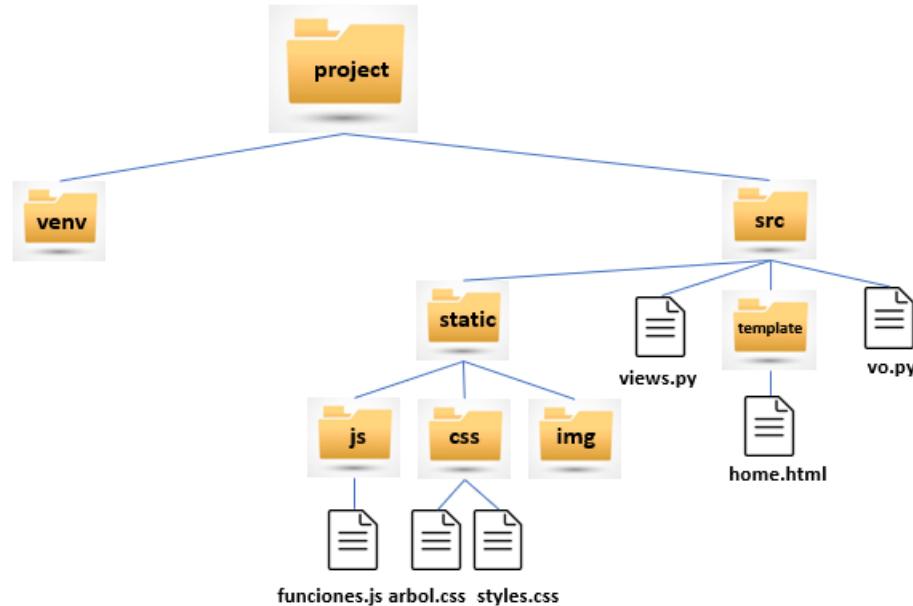


Figura 5.1: Estructura del proyecto en Flask

5.1.1. Ficheros del entorno

Los ficheros principales son:

- **views.py**: es el encargado de la lógica de todos los endpoints de la aplicación, así como el renderizado de la plantilla (home.html). Cada una de esas rutas tendrá una funcionalidad distinta, las cuales serán llamadas por una función del archivo de funciones.js dependiendo de la acción que se elija.
- **vo.py**: se encarga de la organización de clases que puedan ser necesarias para la aplicación. En este caso la utilizaremos para definir la clase de cada nodo del árbol de dependencias.
- **home.html**: es la plantilla donde se mostrará el contenido del asistente y con el que el usuario interactuará. En esencia, lo que percibe el usuario por medio de la vista.
- **funciones.js**: la misión de este fichero es comunicar la aplicación web con los elementos del DOM de la misma, haciendo posible la modificación del HTML dinámicamente. Las modificaciones que efectuamos en este archivo es añadir nuevas etiquetas, modificando o eliminando

otras, cambiar sus atributos, añadiendo clases, cambiar el contenido de texto, etc. También es el encargado de la comunicación con la vista (`views.py`) para la petición y respuesta de servicios web.

- **arbol.css**: archivo encargado de generar los estilos para que nuestro árbol de dependencias tenga la apariencia de un árbol genealógico.
- **style.css**: fichero encargado de dar estilos (fuentes, disposición, tamaños, colores, etc.) a todo lo que concierne a nuestra aplicación web excepto el árbol de dependencias que tiene sus estilos especiales en “`arbol.css`”.

Todos estos ficheros se encuentran alojados en el repositorio <https://github.com/NILGroup/TFG-2021-AsistenteSimplificacion> junto con el resto de directorios del proyecto.

5.1.2. Servicios web externos

Para hacer uso de las principales funcionalidades descritas en el capítulo 4, contamos con una arquitectura de servicios web REST basada en endpoints (*URIs*), intercambiando mensajes entre cliente y servidor (Figura 5.2).

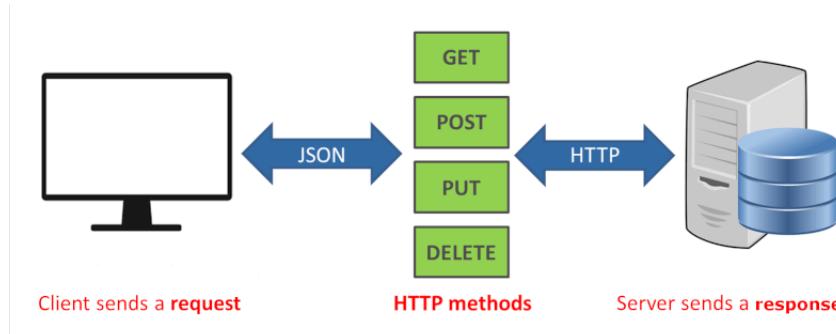


Figura 5.2: Servicio REST cliente/servidor

Un servicio web REST es una interfaz para conectar varios servicios web basados en el protocolo HTTP que define una gran cantidad de métodos. A continuación, describimos los cuatro más básicos:

- **GET**: se utiliza para acceder a los distintos recursos. Si requiere del envío de un parámetro al servidor (*URI param*), éste se pasa como un elemento en la URI (del inglés, *Uniform Resource Identifier*).

- **POST**: se usa para realizar acciones de creación de nuevos recursos. Si se requiere el envío de información al servidor, esta se pasa dentro del cuerpo de la petición HTTP (*body param*).
- **PUT**: se utiliza para la modificación de los recursos existentes. Puede enviar parámetros tanto en la URI como en el cuerpo de la petición HTTP.
- **DELETE**: se utiliza para la eliminar los recursos existentes, siendo la operación análoga al POST. El parámetro será informado a través de la URI.

Estos métodos pueden ser usados en distintas situaciones devolviendo los datos en distintos formatos como XML y JSON. En nuestro caso, hemos usado el formato JSON.

Nuestra arquitectura REST tiene el aspecto como muestra la Figura 5.3. A continuación hablaremos de los diferentes servicios web externos que ofrece la NIL-WS-API y de la librería spaCy que integramos en los distintos endpoints.

Hacemos uso de los siguientes servicios REST que nos ofrece la API del grupo NIL¹ descrita en el capítulo 3, sección 3.3:

- **Servicio para comprobar si una palabra es sencilla.**

```
GET https://holstein.fdi.ucm.es/nil-ws-api/
palabra/{palabra}/es\_sencilla
```

Este recurso devuelve un objeto en formato JSON que contiene el campo “palabraSencilla” de tipo booleano, que en el caso que sea False la palabra introducida a través de la URI es compleja (ver Figura 5.4).

- **Servicio para obtener definiciones de una palabra.**

```
GET https://holstein.fdi.ucm.es/nil-ws-api/
palabra/{palabra}/definiciones
```

En este caso, el servicio devuelve un objeto JSON que contiene el campo “definiciones” de tipo arrayList en el que en cada posición hay otro objeto “definicion” cuyo valor es de tipo string con la acepción correspondiente (ver Figura 5.5).

- **Servicio para obtener sinónimos de una palabra.**

¹Para más información acceder a <https://holstein.fdi.ucm.es/nil-ws-api/>

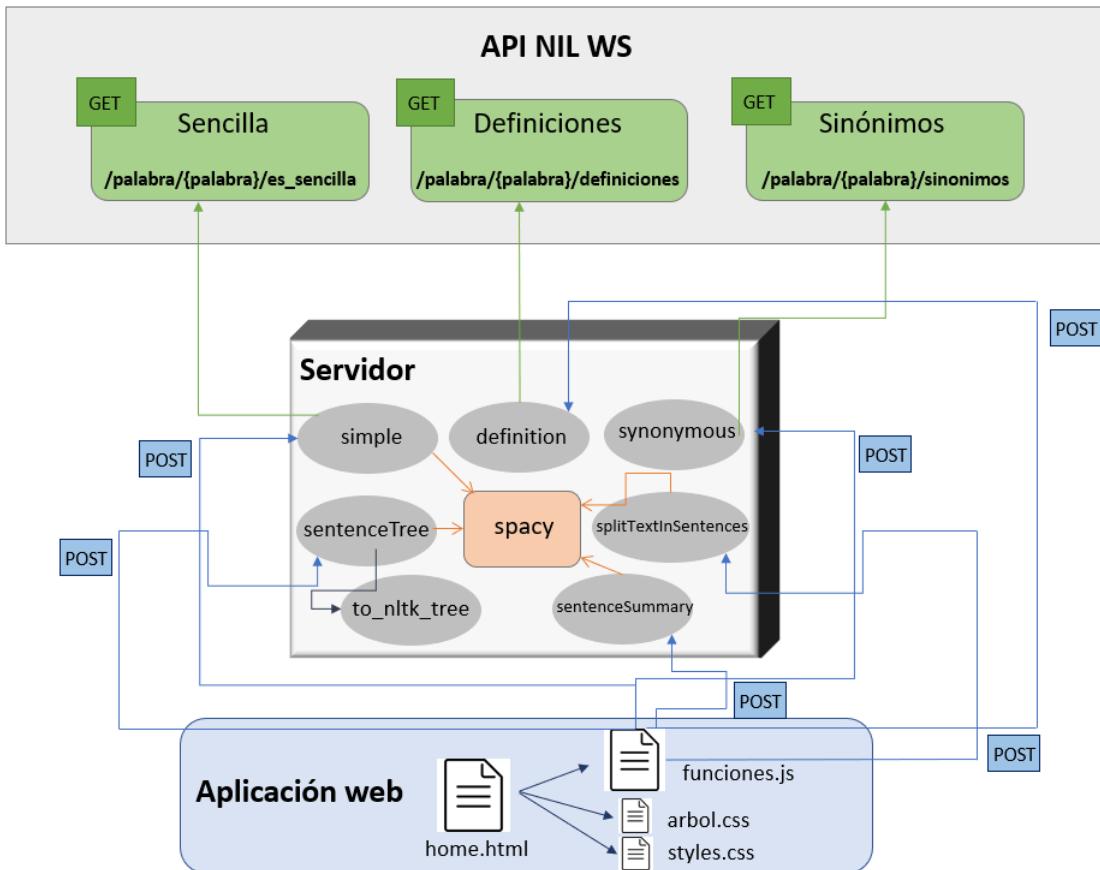


Figura 5.3: Diagrama de la arquitectura REST del asistente web

```
GET https://holstein.fdi.ucm.es/nil-ws-api/
palabra/{palabra}/sinonimos
```

El servicio devuelve un objeto JSON que contiene el campo “sinonimos” de tipo arrayList en el que en cada posición hay otro objeto “sinonimo” cuyo valor es de tipo string con el sinónimo correspondiente (ver Figura 5.6).

Cabe destacar que esta API no hace uso de las reglas de acentuación, de manera que debemos de insertar las palabras sin tildes (obsérvese la Figura 5.6, por ejemplo).

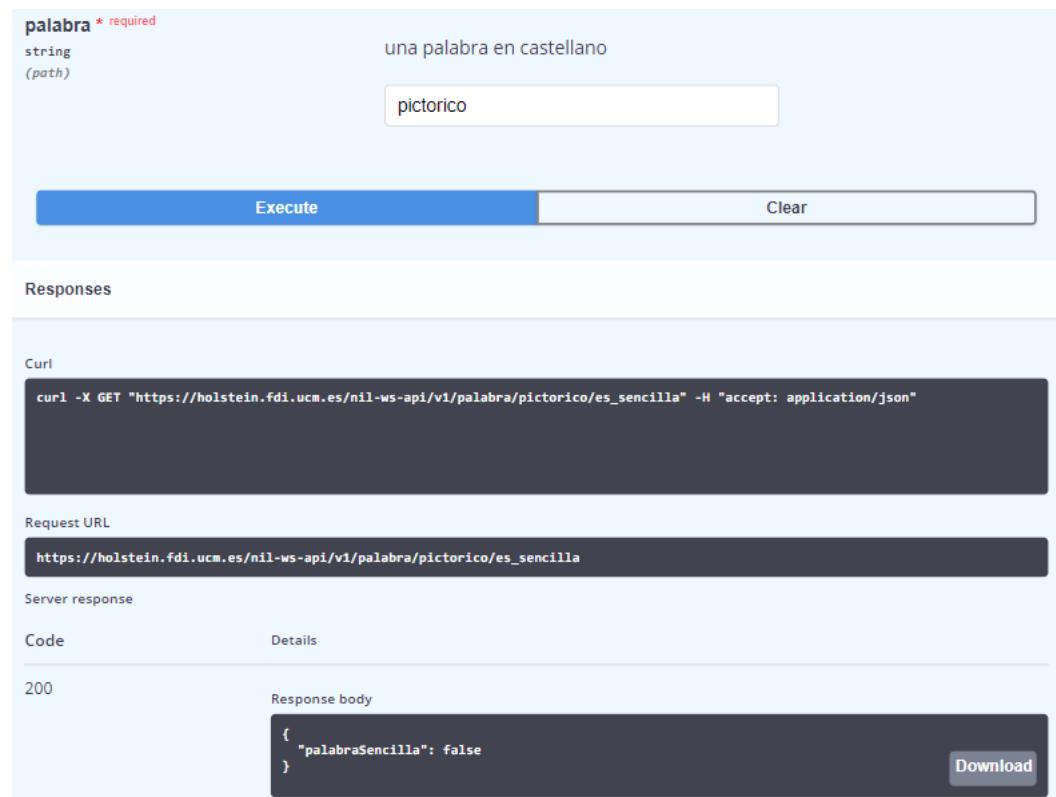


Figura 5.4: Petición para comprobar si una palabra es compleja

5.1.3. Librería spaCy

Para el procesamiento del lenguaje natural (PLN) nos hemos servido de la librería spaCy, en concreto para obtener el etiquetado gramatical y las dependencias entre las palabras de una frase (explicado en el capítulo 3, sección 3.2).

El uso que damos a esta librería es muy similar en todos los endpoints utilizados en la aplicación, con la siguiente parte en común en cada función:

- En primer lugar cargamos el modelo para procesar texto en español con el que vamos a trabajar. El código que se encarga de esto es el siguiente:

```
1 nlp = spacy.load("es_core_news_sm")
```

- Como segundo paso inicializamos el objeto spaCy a partir del lenguaje previamente elegido (español en nuestro caso) y del texto recibido (reci-

The screenshot shows a web-based API testing interface. At the top, there is a form field labeled 'palabra * required' with the placeholder 'una palabra en castellano' and a '(path)' note. A text input field contains the word 'Zanahoria'. Below the form are two buttons: 'Execute' and 'Clear'. Underneath the form, the word 'Responses' is displayed. The first section, 'Curl', shows the command: curl -X GET "https://holstein.fdi.ucm.es/nlp-api/v1/palabra/Zanahoria/definiciones" -H "accept: application/json". The next section, 'Request URL', shows the URL: https://holstein.fdi.ucm.es/nlp-api/v1/palabra/Zanahoria/definiciones. The final section, 'Server response', shows the status code 200 and the response body as a JSON object:

```
{
  "definiciones": [
    {
      "definicion": "f. Planta herbacea umbelífera con flores blancas y purpuras en el centro, de fruto seco y comprimido, y raíz gruesa de color naranja que se utiliza como alimento."
    },
    {
      "definicion": "Raíz de esta planta."
    }
  ]
}
```

A 'Download' button is located at the bottom right of the response area.

Figura 5.5: Petición que devuelve una lista de definiciones

bido previamente en formato JSON, obteniendo el valor que deseamos, en este caso el texto).

```
1 doc = nlp(text)
```

A continuación explicamos cómo y para qué hemos usado la librería spaCy en los diferentes endpoints:

- **/isSimple**: este endpoint lo usamos para comprobar si las palabras de la frase son sencillas o complejas. A partir del objeto en formato spaCy obtenemos las etiquetas que marcan las categorías gramaticales del texto. Esto ha servido a la hora de filtrar las palabras para distinguir su gramática y poder excluir algunas, dado que no interesa saber si un signo de puntuación o un número se considera palabra sen-

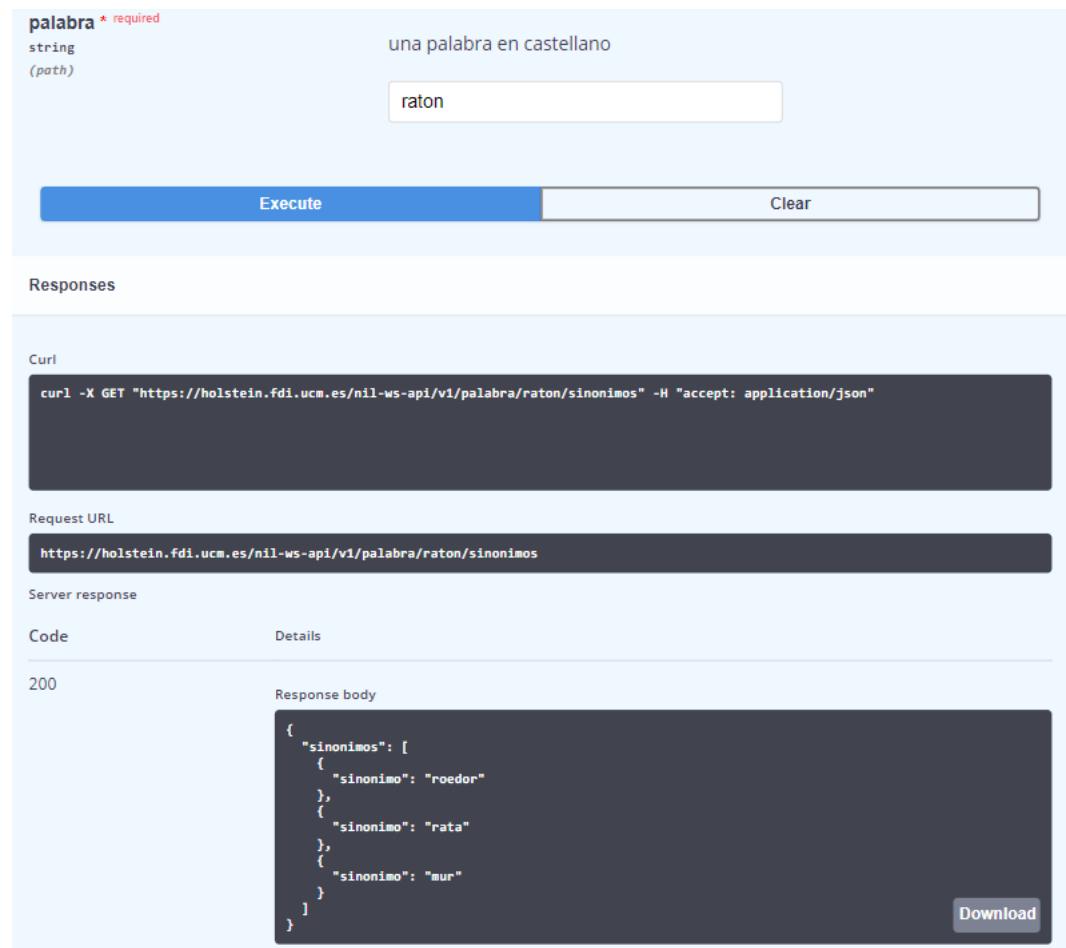


Figura 5.6: Petición que devuelve una lista de sinónimos

cilla o no, reduciendo la cantidad de llamadas a NIL-WS-API (/nil-ws-api/palabra/palabra/es_sencilla), mejorando el tiempo de respuesta de este endpoint.

- **/summary**: para el uso de la funcionalidad “resumen”, partiendo del objeto spaCy, conseguimos todas las palabras del texto, asignándoles una frecuencia de aparición. A continuación, obtenemos el texto dividido en frases mediante “Sentence Boundary Detection”, que spaCy genera cuando creamos el objeto. A cada frase le asignamos una “fuerza” en función de cuantas palabras frecuentes (calculadas previamente) posea. De todas ellas, hacemos una selección del 40 % de las que tengan un valor mayor de “fuerza”.
- **/sentences/tree**: este endpoint lo hemos usado para la construcción del árbol de dependencias. A partir del objeto spaCy obtenemos la

palabra considerada raíz de la frase recibida. Esta será la referente de las que dependen las demás, generando así su árbol de dependencias con la función `to_nltk_tree` (para más aclaración ver sección 5.2).

- **/sentences**: para obtener el texto dividido en frases, hacemos uso de este endpoint. Al generar el objeto spaCy reconoce automáticamente, a través de expresiones regulares, las frases que componen el texto.

5.2. Implementaciones del servidor

En esta subsección explicamos las principales funcionalidades implementadas en el servidor, los endpoints y las funciones auxiliares, con la finalidad de reducir la carga del navegador que ejecute la aplicación.

Parte de estas funcionalidades, implementadas en Python, han requerido el uso de servicios externos (explicados en la sección 5.1.2) y la librería spaCy (explicada en la sección 5.1.3). Las explicamos a continuación:

- **isSimple**: es una función auxiliar que servirá de soporte a los endpoint “/isSimple” y “/synonymous”, que realiza una petición GET al servicio NIL-WS-API (/nil-ws-api/palabra/<palabra>/es_sencilla), devolviendo un booleano con valor “TRUE” en caso de que sea simple. En caso contrario, devolverá “FALSE”.
- **simple**: función que se ejecutará desde la ruta “/isSimple”, recibiendo un objeto JSON con una frase y recorriendo los tokens (palabras) que nos ha proporcionado spaCy usando la función auxiliar descrita en el primer punto. Así se genera una respuesta en formato JSON con un identificador de todas las palabras y un booleano que indican si son sencillas o no.
- **definition**: función que se realizará tras la llamada a la ruta “/definition”, incluyendo un objeto JSON que contiene la palabra objeto de saber sus acepciones, y haciendo uso del servicio (/nil-ws-api/palabra/<palabra>/definiciones) que nos proporciona NIL-WS-API.
- **summary**: funcionalidad que devuelve el resumen del texto, ya explicado en la sección 5.1.3. Para llamar a esta función, se usa la ruta “/summary”.
- **to_nltk_tree**: A partir de un nodo (token) y la altura, inicialmente cero (nodo raíz), generamos de forma recursiva el árbol de dependencias con un recorrido en pre-orden, devolviéndolo completo.

Cada nodo es de la clase personalizada **SentenceTree** localizada en el archivo `vo.py`, la cual tiene como atributos el texto del nodo, los hijos, la altura del nodo y un identificador único.

- **sentenceTree**: función encargada de generar el árbol de dependencias una vez se haya llamado a “sentences/tree”, conteniendo este la función “`to_nltk_tree`”. Se retorna un objeto JSON con la información del árbol de dependencias y un array que contiene cada palabra del árbol con su identificador para facilitar el procesado del mismo en la aplicación web.
- **splitTextInSentences**: función que genera la división del texto en frases. Estás se guardan en un array que se devuelve como un objeto JSON. Se accede a través de la ruta “/sentences”.
- **synonymous**: función que se ejecuta con la llamada “`synonymous`” en la que se recibe, en formato JSON, la palabra. Eliminamos los signos de puntuación de la misma para poder hacer una llamada al servicio NIL-WS-API (`/nil-ws-api/palabra/<palabra>/sinonimos`) que devolverá la lista de sus sinónimos, si los tiene. Luego dividimos esa lista, llamando a la función “`isSimple`” por cada uno de ellos para saber si el sinónimo es sencillo o no, guardándolo en dos arrays distintos, que se devuelven como un JSON.

5.3. Implementaciones de la aplicación web

En esta sección se explica en detalle como han sido desarrolladas las funcionalidades (capítulo 4 sección 4.1) desde un punto de vista de la interfaz.

El archivo “`home.html`” inserta el de “`funciones.js`”, desarrollado en JavaScript, que será el encargado de llamar a los diferentes endpoints del servidor (archivo “`views.py`”). Éste es el que nos proporcionará una respuesta que hará que modifique nuestra vista a través del DOM de una manera dinámica. A continuación, describimos los casos más importantes en los que se produciría este flujo:

- Botón **Resumen**: se realiza una llamada Fetch al endpoint “/summary”. Los datos a enviar en la petición POST tienen la siguiente estructura JSON:

```
{"text": "<texto completo>"};
```

El campo “`text`” incluye el texto completo introducido previamente. La respuesta recibida será un objeto JSON, similar al siguiente:

```
1  {
2      "summary": [
3          "El Rastro de Madrid es el paraíso de la
4              nostalgia, un mercadillo único que la
5                  pandemia ha puesto en la cuerda floja.", ,
6          "La última apuesta para revitalizarlo es
7              recuperar la Feria de desembalajes en la
8                  plaza del General Vara de Rey, una
9                      tradición que tuvo su origen en los años
10                         70, y que ahora se asienta el primer y
11                             el tercer sábado del mes para dinamizar
12                                 las ventas y que no se encasillen en los
13                                     clásicos domingos."
14     ]
15 }
```

El campo “summary” es un array en el que cada posición es una frase que constituye el resumen. Este array se recorre incrustándose cada frase en el código HTML a modo de lista (una debajo de otra).

- Botón **Texto completo**: hacemos uso del endpoint “/sentences”, devolviendo el texto completo en frases. La respuesta que nos da esta petición es la siguiente:

```
1  {
2      "sentences": [
3          "Muebles antiguos, cuadros, lámparas de vidrio
4              , molinillos de café, porcelanas, vasijas,
5                  bustos y todo tipo de cachivaches.", ,
6          "El Rastro de Madrid es el paraíso de la
7              nostalgia, un mercadillo único que la
8                  pandemia ha puesto en la cuerda floja.", ,
9          "Los comercios que viven de este enclave
10             singular han pasado un año difícil, con
11                 bajos ingresos y muchas restricciones.", ,
12          "Pero poco a poco, el Rastro vuelve a su
13              ebullición y reivindica la calle como un
14                  lugar de encuentro e intercambio.", ,
15          "La última apuesta para revitalizarlo es
16              recuperar la Feria de desembalajes en la
17                  plaza del General Vara de Rey, una tradici
18                      ón que tuvo su origen en los años 70, y que
19                          ahora se asienta el primer y el tercer s
20                              ábado del mes para dinamizar las ventas y
```

```

        que no se encasillen en los clásicos
        domingos."
8     ]
9 }
```

Al igual que en el caso anterior, haremos un recorrido en cada frase del array para mostrarlas en la interfaz de frases seleccionables en modo lista.

- **Creación y construcción del árbol de dependencias:** para esta parte hacemos una llamada Fetch al endpoint “/sentences/tree”. El cuerpo de la petición POST incluirá el siguiente JSON:

```
{"sentence": "<frase>}";
```

Lo que nos devuelve esta llamada son dos objetos JSON análogos, pero se van a usar para distintas finalidades. El primero, contendrá el campo “sentencesIds” siendo éste un array que incluirá un identificador único (campo “id”) y la palabra (campo “text”), que nos servirá para facilitar transformaciones futuras. Ese identificador es el mismo que se obtiene en la estructura del árbol, evitando así los problemas que puedan surgir con palabras duplicadas. Volcaremos el contenido en otro array de objetos al que llamaremos en referencias futuras “sentenceArray”, que usaremos como soporte para cambiar dinámicamente el borrador del texto final, e iremos modificándolo en cada funcionalidad que requiera una transformación en el árbol de dependencias. Por ejemplo, para la frase “La sonrisa es la mejor respuesta para una mirada.”, el JSON que recibimos tendría el siguiente aspecto:

```

1   "sentenceIds": [
2   {
3       "id": 0,
4       "text": "La"
5   },
6   {
7       "id": 1,
8       "text": "sonrisa"
9   },
10  {
11      "id": 2,
12      "text": "es"
13  },
14  {
15      "id": 3,
16      "text": "la"
```

```

17 },
18 {
19     "id": 4,
20     "text": "mejor"
21 },
22 {
23     "id": 5,
24     "text": "respuesta"
25 },
26 {
27     "id": 6,
28     "text": "para"
29 },
30 {
31     "id": 7,
32     "text": "una"
33 },
34 {
35     "id": 8,
36     "text": "mirada"
37 },
38 {
39     "id": 9,
40     "text": "."
41 }
42 ]

```

El segundo JSON que recibimos contiene la estructura del árbol, con los siguiente datos:

- **children**: array de nodos hijos.
- **height**: nivel del nodo con respecto a la raíz.
- **id**: identificador único de cada nodo.
- **text**: palabra propia del nodo.

Para la frase utilizada en el ejemplo anterior, el JSON recibido es:

```

1   "tree": {
2       "children": [
3           {
4               "children": [
5                   {
6                       "children": [],
7                       "height": 2,

```

```
8         "id": 0,
9         "text": "La"
10        }
11        ],
12        "height": 1,
13        "id": 1,
14        "text": "sonrisa"
15    },
16    {
17        "children": [],
18        "height": 1,
19        "id": 2,
20        "text": "es"
21    },
22    {
23        "children": [],
24        "height": 1,
25        "id": 3,
26        "text": "la"
27    },
28    {
29        "children": [],
30        "height": 1,
31        "id": 4,
32        "text": "mejor"
33    },
34    {
35        "children": [
36            {
37                "children": [],
38                "height": 2,
39                "id": 6,
40                "text": "para"
41            },
42            {
43                "children": [],
44                "height": 2,
45                "id": 7,
46                "text": "una"
47            }
48        ],
49        "height": 1,
50        "id": 8,
51        "text": "mirada"
```

```

52      },
53      {
54         "children": [],
55         "height": 1,
56         "id": 9,
57         "text": "."
58     }
59     ],
60     "height": 0,
61     "id": 5,
62     "text": "respuesta"
63 }
```

Una vez tenemos la información para la construcción del árbol, se ha hecho uso de un algoritmo de búsqueda en profundidad (BFS)², recorriendo todos los nodos (en nuestro caso palabras de la frase). El funcionamiento de este algoritmo consiste en ir expandiendo recursivamente cada uno de sus nodos desde la raíz hasta el nodo hoja, es decir, obedeciendo a la estructura del segundo JSON vamos procesando los arrays children de cada nodo hasta que este array sea vacío, insertando esos nodos en unas listas y sublistas en el HTML de manera recurrente. El flujo que se ha seguido para la implementación de este algoritmo se muestra en la Figura 5.7.

- Botón **Elección de las palabras**: cuando se hace clic en una palabra en el árbol de dependencias, se recoge su id, añadiéndole en el atributo classList la clase “active”, que nos permite manipular el DOM de esa palabra.
- Botón **Activar palabras complejas**: se realiza una petición POST al endpoint “/isSimple”, con un objeto JSON, que contiene los siguientes datos:
 - **id**: mismo identificador que se utiliza en la creación del árbol descrito anteriormente.
 - **simple**: booleano con valor “False” si la palabra es compleja.

Se recorre el objeto modificando los atributos en el DOM, en el caso de que la palabra (designada por su id) sea compleja, añadiendo una nueva clase para darle así un estilo destacado.

- Botón **Intercambio**: cuando se escogen dos palabras del árbol de dependencias, obtenemos sus ids gracias a la clase “active”. Después se

²Para más información https://es.wikipedia.org/wiki/B%C3%A1squeda_en_profundidad

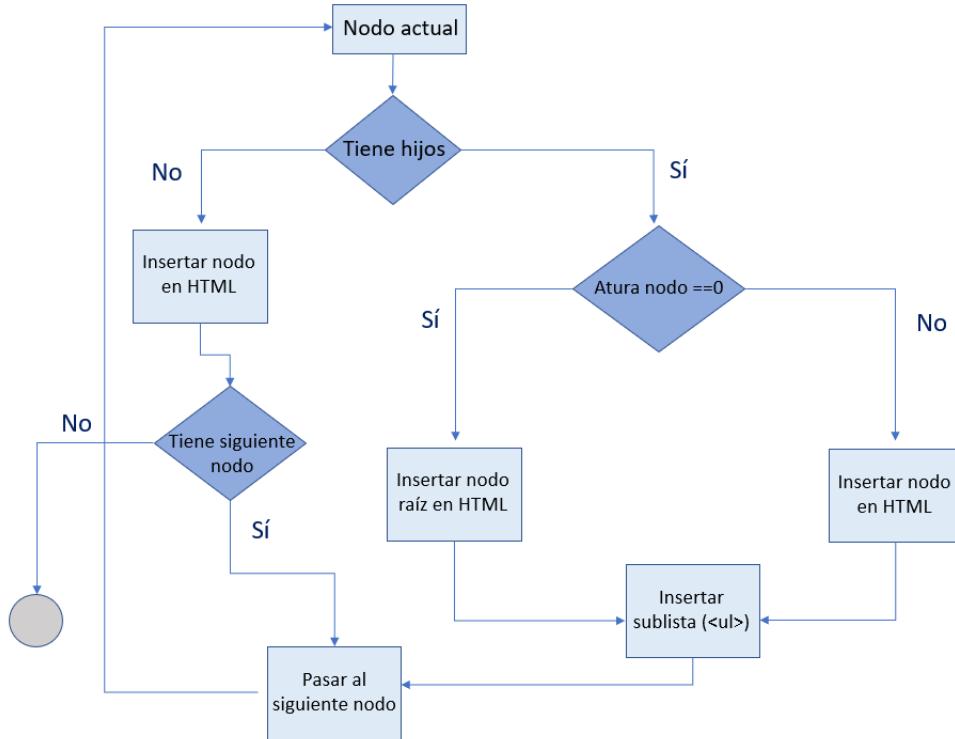


Figura 5.7: Diagrama de flujo de nuestro algoritmo BFS

reemplaza en el DOM el id del primer término por el segundo y viceversa. Durante el desarrollo de esta parte hemos tratado la funcionalidad de dos maneras diferentes:

- Si las dos palabras son padre e hijo, se intercambia sólo sus ids y la palabra en el DOM, excluyendo sus dependencias. Para que se vea el resultado de este intercambio en el borrador final, se intercambia el contenido del “sentenceArray” de ambas palabras e ids.
- Si el caso anterior no se da, se intercambian ambos nodos incluyendo los nodos dependientes, intercambiéndolos en el DOM. En este caso, para escribir en el borrador la frase resultante, se guarda en un array auxiliar los ids involucrados, es decir, aquellos que formen parte de las palabras seleccionadas en el intercambio incluyendo estas y sus dependencias, ordenándolos de menor a mayor, para saber la posición inicial de dichos nodos. A continuación, se forma una sublista con los nodos de la primera palabra seleccionada y se extrae del array “sentenceArray” que se muestra a modo de ejemplo en la Figura 5.8. Una vez eliminada esta sublista, se

evalúa la nueva posición en la que se encuentra la segunda sublista y se vuelve a eliminar de “sentenceArray” como se puede ver en la Figura 5.9. Por último se inserta la segunda sublista en la posición de la primera y viceversa (Figura 5.10), teniendo así nuestra frase con este intercambio en el array “sentenceArray” como observamos en la Figura 5.11.

Paso 1. Creación primera sublista

```
Array sentenceArray
▶ 0: {id: 0, text: "La"}
▶ 1: {id: 1, text: "desinfección"}
▶ 2: {id: 2, text: "de"}
▶ 3: {id: 3, text: "manos"}
▶ 4: {id: 4, text: "también"}
▶ 5: {id: 5, text: "será"}
▶ 6: {id: 6, text: "crucial"}
▶ 7: {id: 7, text: "en"}
▶ 8: {id: 8, text: "una"}
▶ 9: {id: 9, text: "jornada"}
⋮
▶ 21: {id: 21, text: "en"}
▶ 22: {id: 22, text: "las"}
▶ 23: {id: 23, text: "9.000"}
▶ 24: {id: 24, text: "urnas"}
▶ 25: {id: 25, text: "habilitadas"}
▶ 26: {id: 26, text: "."}
```

Primera sublista

```
▶ 0: {id: 0, text: "La"}
▶ 1: {id: 1, text: "desinfección"}
▶ 2: {id: 2, text: "de"}
▶ 3: {id: 3, text: "manos"}
```

A red bracket on the left side of the original sentenceArray list groups items 0 through 3. A red arrow points from this bracket to the 'Primera sublista' below it. Another red bracket on the right side of the original list groups items 4 through 26. A red arrow points from this bracket to the 'Extraída' label below it.

Paso 2. Extracción primera sublista

```
Array sentenceArray
▶ 0: {id: 4, text: "también"}
▶ 1: {id: 5, text: "será"}
▶ 2: {id: 6, text: "crucial"}
▶ 3: {id: 7, text: "en"}
▶ 4: {id: 8, text: "una"}
▶ 5: {id: 9, text: "jornada"}
⋮
▶ 17: {id: 21, text: "en"}
▶ 18: {id: 22, text: "las"}
▶ 19: {id: 23, text: "9.000"}
▶ 20: {id: 24, text: "urnas"}
▶ 21: {id: 25, text: "habilitadas"}
▶ 22: {id: 26, text: ".")}
```

Extraída

A red bracket on the left side of the modified sentenceArray list groups items 0 through 3. A red arrow points from this bracket to the 'Extraída' label below it. Another red bracket on the right side of the list groups items 4 through 22. A red arrow points from this bracket to the 'Primera sublista' label above it.

Figura 5.8: Creación y extracción de la primera sublista del array “sentenceArray”.

- Botón **Sinónimos**: antes de la llamada al endpoint correspondiente (“/synonymous”), la palabra seleccionada será convertida a un objeto JSON de la siguiente manera:

```
objeto = {"synonymous": "Mirada"}
jsonObj = JSON.stringify(objeto);
```

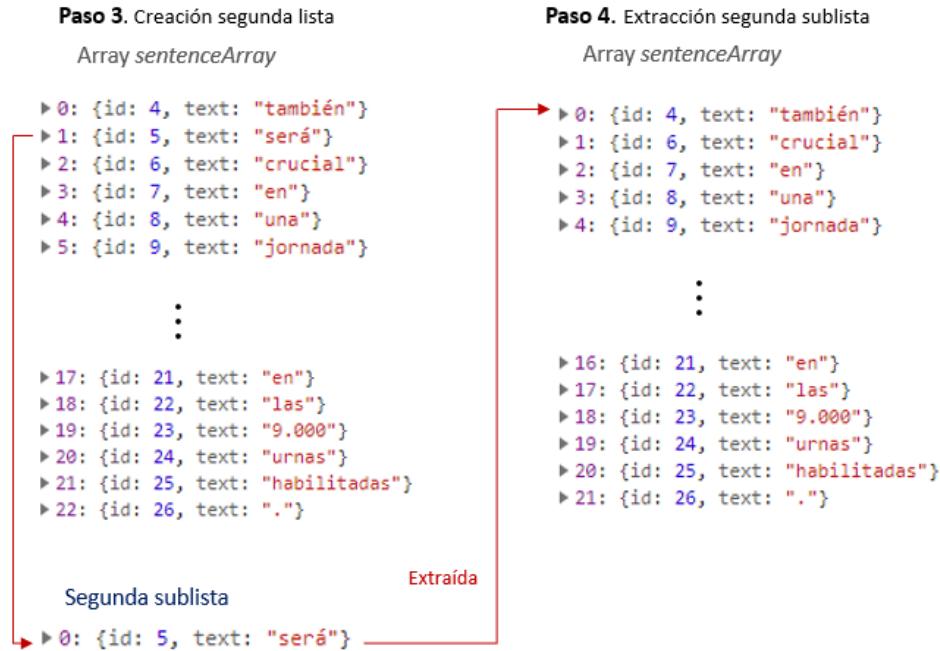


Figura 5.9: Creación y extracción de la segunda sublistas del array “sentenceArray”.

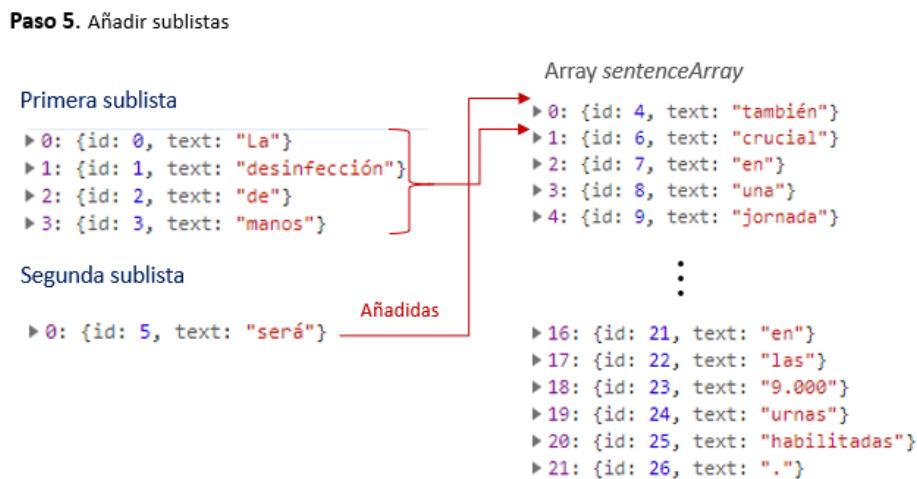


Figura 5.10: Inserción de ambas sublistas en el array “sentenceArray”.

Tras la llamada, se recibirá una respuesta con dos objetos JSON, uno con un listado de sinónimos sencillos, si los tiene, y otro con los com-

Paso 6. Resultado

```
Array sentenceArray
▶ 0: {id: 5, text: "será"}
▶ 1: {id: 4, text: "también"}
▶ 2: {id: 0, text: "La"}
▶ 3: {id: 1, text: "desinfección"}
▶ 4: {id: 2, text: "de"}
▶ 5: {id: 3, text: "manos"}
▶ 6: {id: 6, text: "crucial"}
▶ 7: {id: 7, text: "en"}
▶ 8: {id: 8, text: "una"}
▶ 9: {id: 9, text: "jornada"}

⋮

▶ 21: {id: 21, text: "en"}
▶ 22: {id: 22, text: "las"}
▶ 23: {id: 23, text: "9.000"}
▶ 24: {id: 24, text: "urnas"}
▶ 25: {id: 25, text: "habilitadas"}
▶ 26: {id: 26, text: "."}
```

Figura 5.11: Resultado del array “sentenceArray” tras el intercambio.

plejos. Si la palabra no tuviese sinónimos, una de las listas o ambas serán vacías:

```
1  {
2      "simpleSynonymous": [],
3      "synonymous": [
4          "ojeada",
5          "vistazo",
6          "observacion",
7          "inspeccion",
8          "examen"
9      ]
10 }
```

Para introducir los sinónimos en HTML se recorren ambos objetos, siempre y cuando tengan algún sinónimo, y se van incrustando en HTML con la etiqueta de lista (no enumerada). En este caso, observamos en el código de ejemplo anterior que la palabra “Mirada” no nos devuelve sinónimos sencillos (campo “simpleSynonymous”), mostrando una lista vacía. En cambio, sí que podremos observar una lista de

sinónimos complejos (campo “synonymous”).

En este endpoint encontramos un inconveniente. En ocasiones, los sinónimos recibidos no concuerden correctamente en número, persona o conjugación con el contexto de la frase. Para solucionarlo, hemos optado por que el usuario pueda ser capaz de editarlo in situ. Así, en la lista de los sinónimos cada uno tendrá un input que se mostrará con el botón que aparece a su derecha para editarlo. Se recogerá el valor del input modificándolo en el DOM.

- Botón **Eliminar**: esta funcionalidad se ha realizado tomando de nuevo el id asociado a la palabra seleccionada de nuestro árbol mediante la clase “active”. Para eliminar una palabra o subárbol, es decir, palabra propia seleccionada y sus dependientes en el borrador final, necesitaremos la ayuda de un array auxiliar, donde almacenamos todos los ids que dependen de la palabra, incluida ésta, ordenándolos de forma creciente. Como hemos visto en la construcción del árbol, tenemos a nuestra disposición el “sentenceArray” con el id y la palabra, asociados entre sí, que vamos a usar para extraer el contenido del array auxiliar en el “sentenceArray” para escribir el resultado en el borrador. Para hacer esta eliminación en el árbol se hace uso del `ChildNode.remove()`, que proporciona JavaScript para eliminar el objeto del DOM al que pertenece.
- Botón **Definición**: se realiza una llamada Fetch al endpoint “/definition” con el siguiente JSON:

```
{"word": "<palabra>"}
```

El resultado de la petición consistirá en un objeto JSON con un listado de las acepciones, si las tiene, y en caso contrario devolverá una lista vacía. El siguiente código muestra el resultado una vez realizada la petición POST con la palabra “secador”:

```

1  [
2      "definiciones": [
3          {
4              "definicion": "adj. Que seca."
5          },
6          {
7              "definicion": "m. y f. Aparato o
8                  maquina que sirve para secar."
9          }
10     ]

```

Estas definiciones se insertan en el código HTML, escribiéndolas a modo de lista, incluyendo un enlace para seleccionarlas y poder adjuntarlas al borrador.

5.4. Extensibilidad del asistente web interactivo

Una vez explicada la implementación tanto la parte relacionada con el servidor como de la aplicación en sí, describiremos a continuación qué pasos debe de seguir un desarrollador que retome este asistente y desee añadir una nueva funcionalidad que requiera del uso de los servicios que ofrece NIL-WS-API o la librería spaCy.

Para ello, habría que realizar modificaciones en los siguientes ficheros:

- **views.py**: en el caso de que sea necesario añadir una nueva funcionalidad llamando a un servicio de NIL-WS-API mediante un nuevo endpoint. He aquí un ejemplo:

```
1  @app.route('/definition', methods=['POST'])
2  def definition():
3      word=request.get_json()
4      word=word['word']
5      response = requests.get(NILWS_URL + word +
6          DEFINITION_URL)
6      {...} #Tratamiento de la respuesta.
7      data = response.json()
8
9      return jsonify(definiciones=data[',
10                      definiciones'])
```

Aspectos a tener en cuenta para llamar a un servicio NIL-WS-API:

- En la línea 1 se escribirá la instrucción “@app.route(’<ruta>’, methods=[‘POST’])” con el endpoint correspondiente. Esta instrucción se utilizará para llamar a la ruta desde funciones.js.
- En la línea 2 se define el nombre de la función.
- En la línea 3 convierte el objeto JSON de la petición en datos de Python para utilizarlos a la hora de enviar la petición en la línea 5.
- En la línea 5 se realiza la llamada al servicio NIL-WS-API guardando su respuesta.

- En las líneas sucesivas se produce el tratamiento de la respuesta a esa petición.

A la hora de utilizar la librería spaCy, cabe destacar que:

- Las cuatro primeras líneas tienen el mismo comportamiento que las de una petición de un servicio de la API.
- En la línea 5 cargamos un modelo para procesar texto en español.
- En la línea 6 se ejecuta una llamada a nlp para producir un documento analizado.
- En las líneas posteriores se procederá a la manipulación de este documento.

```

1  @app.route('/sentences/tree', methods=['POST',
2      ])
3  def sentenceTree():
4      data = request.get_json()
5      sentence = data["sentence"]
6      nlp = spacy.load("es_core_news_sm")
7      doc = nlp(sentence)
8      {...} #Tratamiento del documento.
9      jsonTree = ''
10     sentenceIds = []
11     for sent in doc.sents:
12         tree = to_nltk_tree(sent.root, 0)
13         for word in sent:
14             sentenceIds.append({'text': word.text, 'id': word.i})
15
16     jsonTree = jsonify(tree=tree.serialize(),
17                         sentenceIds=sentenceIds)
18     print(tree)
19
20     print(tree.serialize())
21     return jsonTree

```

- **funciones.js**: en este fichero se añade una nueva función que incluya una llamada Fetch mandando los diferentes campos para el uso de una ruta determinada que se llamará en views.py.

```

1  function definition() {
2      ...
3
4      var j = {"word": palabra}

```

```
5     var jsonObj = JSON.stringify(j);
6     fetch("/definition", {
7         method: "POST",
8         headers: {
9             'Content-Type': 'application/json',
10            },
11            body: jsonObj
12        }).then(response => response.json()
13        ).then(data => {
14            var elements = data.definiciones;
15
16            {...} //Operaciones a realizar tras la
17            //respuesta.
18        }
19    );
20 }
```

- **home.html**: añadir un nuevo componente a la interfaz que se encargue de llamar a una función del archivo “funciones.js”.

```
1 <button title="Consulta la/s definición/es de
2     una palabra del árbol seleccionada"
3     id="bDefinicion" class="botonesFuncionales"
4     disabled
5     onclick="definition();">Definición
6 </button>
```

- **styles.css**: añadir nuevas reglas para dar estilo al nuevo componente de la vista objeto de la nueva funcionalidad (colores, tamaños, etc.)

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este apartado queremos poner en valor a todas aquellas personas encargadas de facilitar la compresión y el aprendizaje de un determinado texto a esa parte de la sociedad que, desgraciadamente, no tienen la suficiente capacidad psíquica y/o física para comprender lo que el texto quiere transmitir.

Por nuestra parte, hemos querido contribuir con nuestro granito de arena para que este trabajo se haga lo menos costoso posible para el editor, ya que es importante que cualquier persona, a pesar de sus discapacidades, tenga acceso a la cultura y a la información de una manera u otra.

Previa a la implementación de este asistente, tuvimos que realizar una labor de investigación para ver cuál era el proceso que seguían estos editores para convertir textos a Lectura Fácil, concluyendo que, a día de hoy, necesitaban todavía de procesos manuales para llevar a cabo estas adaptaciones.

De esta manera, este asistente se implementó teniendo presente el objetivo de convertir frases complejas a otras más simples mediante simplificaciones léxicas y sintácticas para potenciar la comprensión lectora, proporcionando al editor varias funcionalidades para tal fin. Con la ayuda del árbol de dependencias, el usuario podrá llevar a cabo estas simplificaciones de una manera interactiva, simplemente seleccionando palabras y acción, pudiendo visualizar en todo momento el resultado provisional de estas adaptaciones.

Hicimos uso de servicios externos para llevar a cabo las distintas funcionalidades: NIL-WS-API, así como spaCy para el procesamiento del lenguaje

je natural y desarrollar análisis sintácticos. Facilitamos la extensión de las funcionalidades gracias a NIL-WS-API, que ofrece diversos servicios que se podrían añadir en futuras mejoras del asistente.

Con respecto al diseño, pretendemos que la interfaz implementada sea lo más sencilla e intuitiva posible para mejorar la experiencia del editor.

6.2. Trabajo futuro

Este asistente presenta mejoras técnicas que pueden ser implementadas en próximos desarrollos. Explicamos algunas de ellas desde el punto de vista de la funcionalidad y la portabilidad:

Referente a la funcionalidad, sería interesante abordar los siguientes puntos:

- Al reemplazar un sinónimo, que pueda ser éste conjugado y concuerde con el resto de la oración automáticamente sin necesidad de que el editor lo adapte a las necesidades del contexto de la frase.
- Implementar diferentes niveles de complejidad de la adaptación ya que no todos los lectores tendrán las mismas necesidades.
- Importación de imágenes o pictogramas para explicar algún concepto que pueda ser algo más complejo o que el lector desconozca, y que de esta manera resulte más claro el texto.
- Trabajar en los servicios que ofrece NIL-WS-API para que haga uso de las reglas de acentuación del castellano, puesto que todas las palabras vienen dadas sin signos de puntuación.
- Extender estos servicios a otros idiomas de manera que el asistente sea capaz de reconocer diferentes tipos de lenguajes.
- Almacenar datos de las adaptaciones realizadas por el usuario para que el sistema aprenda y haga sugerencias.
- Evaluación con un usuario editor para darnos feedback y conocer su punto de vista.

Con respecto a la portabilidad de la aplicación web, proponemos la siguiente mejora:

- Sería interesante desarrollar esta aplicación para dispositivos Android e iOS para una mayor comodidad del editor, pudiendo, por ejemplo, realizar estas adaptaciones desde una tablet, aprovechando el uso táctil a la hora de elegir palabras o funcionalidades.

Chapter 6

Conclusions and Future Work

6.1. Conclusions

In this section we want to highlight all those people in charge of facilitating the understanding and learning of a certain text to that part of society that, unfortunately, do not have the psychic and/or physical capacity to understand what the text wants to transmit.

For our part, we wanted to do our bit to make this work as inexpensive as possible for the user editor, since it is important that anyone, despite their disabilities, have access to culture and information in one way or another.

Prior to the implementation of this assistant, we had to carry out a research work to see what was the process followed by these editors to convert texts to Easy-to-Read, concluding that, to this day, they still needed manual processes to carry out these adaptations.

Thus, this assistant was implemented with the objective of converting complex sentences to simpler ones by means of lexical and syntactic simplifications to enhance reading comprehension, providing the editor with several functionalities for this purpose. With the help of the dependency tree, the user will be able to carry out these simplifications in an interactive way, simply by selecting words and action, being able to visualize at any time the provisional result of these adaptations.

We made use of external services to carry out the different functionalities: NIL-WS-API, as well as spaCy for natural language processing and parsing. We facilitate the extension of the functionalities thanks to NIL-WS-API,

which offers several services that could be added in future enhancements of the wizard.

Regarding the design, we intend to make the implemented interface as simple and intuitive as possible to improve the editor experience.

6.2. Future work

This assistant presents technical improvements that can be implemented in future developments. We explain some of them from the point of view of functionality and portability:

Referring to functionality, it would be interesting to address the following points:

- When replacing a synonym, that it can be conjugated and match the rest of the sentence automatically without the need for the editor to adapt it to the needs of the context of the sentence.
- Implement different levels of complexity of adaptation since not all readers will have the same needs.
- Importing images or pictograms to explain a concept that may be more complex or unknown to the reader, thus making the text clearer.
- Work on the services offered by NIL-WS-API to make use of the Spanish accentuation rules, since all words are given without punctuation marks.
- Extend these services to other languages so that the assistant is able to recognize different types of languages.
- Store data of the adaptations made by the user so that the system can learn and make suggestions.
- Evaluation with an editor user to give us feedback and know his point of view.

Regarding the portability of the web application, we propose the following improvement:

- It would be interesting to develop this application for Android and iOS devices for greater convenience of the editor, being able, for example, to make these adaptations from a tablet, taking advantage of the touch use when choosing words or functionalities.

Trabajo individual

7.1. Estefanía Ortega Ávila

En primer lugar, mi labor en este TFG se centró en una investigación de las tareas y los pasos a seguir que se llevan a cabo para adaptar un texto a Lectura Fácil. Para ello, consulté diversas webs y vídeos que me proporcionaran dicha información. De esta manera, entendí a qué público iba dirigida este tipo de lectura y qué técnicas eran las más utilizadas.

Posteriormente, tanto mi compañero como yo, identificamos diferentes herramientas que podrían ser útiles para transformar textos, probando algunas de ellas, haciéndonos una idea de qué medios disponen los editores para realizar su labor.

Una vez realizada la investigación, mi compañero y yo comenzamos a estudiar las tecnologías que iban a hacer posible la implementación de nuestro asistente web. Optamos por hacer uso de servicios API REST, así como de la librería spaCy para el PLN.

Por otro lado, y esta es la que yo abordé, tenemos la implementación de la aplicación, es decir, la interfaz visual con el que el usuario editor interactuará y hará uso de las diferentes funcionalidades.

Para el desarrollo de esta implementación, utilicé JavaScript para que los resultados de la llamada al servidor mediante una promesa Fetch a un endpoint me devolviera un objeto JSON que, al operar sobre él, plasme cierta información en la interfaz; usando HTML, para el desarrollo de la interfaz visual, y CSS para dar estilo a ésta (fuentes, tamaños, colores...).

La parte donde tuve más dificultad fue la de dibujar el árbol de dependencias con el aspecto de árbol genealógico y el poder procesar el objeto JSON que recibo del servidor para representar cada nodo con sus correspondientes dependencias. La solución por la que opté fue la de implementar un algoritmo BFS para poder recorrer el objeto, obteniendo la estructura de árbol deseada.

Otra parte de especial dificultad fue la relacionada con la funcionalidad “Intercambiar” en el caso de que se tuviera que modificar dos palabras independientes entre sí y, a su vez, las dependientes de éstas, reflejándose estos cambios en el borrador del texto final. Para solventarlo, hice uso de arrays auxiliares que me permitieron obtener el resultado que quería en el borrador para poder cambiar el texto dinámicamente.

Realicé algunas pruebas en Postman de los diferentes servicios de NIL-WS-API para validar que el comportamiento de la interfaz se correspondía con lo que la API devolvía en su respuesta, por ejemplo, en el caso de comprobar que si una palabra se muestra de color rojo en el árbol, la API nos indique que dicha palabra es compleja. Esto me permitió, por primera vez, hacer uso de la herramienta Postman, la cual es bastante apropiada para el testeo de APIs.

Para el diseño de la interfaz del asistente, tomé en cuenta las recomendaciones de mis tutoras en las reuniones periódicas, sobre todo en lo que concierne a nombres de botones, elementos de la interfaz visibles o no visibles al pinchar en una determinada funcionalidad, etc.

Además de la implementación, me encargué de algunas de las partes que consta la memoria. En el capítulo 1, redacté los puntos relacionados con la motivación y los objetivos que persigue este asistente. El capítulo 2, los puntos 2.3 y 2.4 relacionados con proyectos, programas y aplicaciones en LF. Con respecto al capítulo 4, fui la persona encargada de desarrollarlo en su totalidad hablando sobre la aplicación y sus diferentes funcionalidades y vistas, mientras que del capítulo 5 fui la autora del punto 5.1 y 5.3, relacionados con la arquitectura y con la implementación del asistente, respectivamente. Finalmente, el capítulo 6, lo elaboré junto con la ayuda de mi compañero. Por último, traduje al inglés la parte de la introducción y la parte de las conclusiones con la ayuda de mi compañero.

Por supuesto, se fue realizando modificaciones en aquellas partes que a nuestras tutoras les parecía que debíamos mejorar.

7.2. Javier Sesé García

Mi contribución al proyecto comenzó investigando a quién iba dirigida la Lectura Fácil y las formas que se utilizaban para adaptar textos.

Posteriormente, nos centramos en el diseño de la arquitectura de la aplicación, la cual es una arquitectura REST basada en endpoints. Elegimos este diseño ya que nos pareció que es fácilmente ampliable en el caso de que se quisiera realizar un servicio nuevo.

Implementé esta arquitectura con Flask, del cual he adquirido los conocimientos para ser capaz de desplegar el servidor en él. He elegido Flask ya que, aunque tenía más experiencia con Django, me ha parecido un entorno ligero y fácil de usar y una oportunidad de ampliar mis conocimientos.

En lo que concierne a las herramientas, investigué y aprendí a usar la librería spaCy que otorga la principal funcionalidad de análisis gramatical del asistente. Para ello tuve que familiarizarme con todas las funcionalidades de spaCy que hemos usado, investigando en profundidad lo que podían hacer y de qué manera, así como hacer un repaso de la gramática, para entender completamente el funcionamiento y uso que le podemos dar a la funcionalidad del etiquetado gramatical de spaCy.

Además, implementé las llamadas a los diferentes servicios de NIL-WS-API desde la parte del servidor, escrito con lenguaje Python.

Al igual que mi compañera, me serví de Postman para verificar los datos que recibía el servidor, y gracias a ello encontré una carencia de la API, y es que no acepta caracteres especiales, ya que estos se pasan como argumento de la URI, por lo que decidí, como posible solución, eliminar estos caracteres de las palabras.

Por el contrario, esta solución presentaba otro problema en sí, y es que sin esos caracteres especiales se da ambigüedad en los resultados de los servicios, como por ejemplo el de sinónimos, que la propia API ya considera retornando todos los posibles valores, por lo que decidimos hacer lo mismo y dejar a elección del usuario el valor con el que quedarse.

Desconocía como se debían de hacer las llamadas a los endpoints del servidor, lo que me supuso un esfuerzo hasta que aprendí a realizar las promesas mediante la operación Fetch. Adicionalmente, implementé parte de la funcionalidad de las propias promesas en JavaScript junto con mi compañera.

También investigué como hacer la conjugación de palabras a partir de su lema, tiempo verbal y número gramatical, lo que me llevó a la conclusión de

que no sería capaz de hacerlo solamente con spaCy, por lo que busqué otras formas de realizarlo en Python.

Llegué a dar con varias formas de realizar esta conjugación, pero ninguna adaptada al texto en castellano a pesar de que encontré una posibilidad de montar otro servidor en Java para ello, pero valoré que no tenía el tiempo suficiente para aprender a usarlo lo cual no nos servía para esta aplicación ya que estaba disponible en inglés.

Por otro lado, estudié la opción de hacer nuestro propio conjugador, pero de nuevo valoré que no teníamos el tiempo necesario suficiente para ello. Por todo ello, decidimos dejar la posibilidad de la conjugación automática de palabras como posible trabajo futuro.

Una vez terminada la implementación me encargué de alojar el servidor en el contenedor que nos han otorgado (<https://holstein.fdi.ucm.es/tfg/2021/simpli/>). Para ello, tuve que aprender cómo usarlo, así como de saber lanzar el servidor de Flask en un docker, lo que ha obligado a realizar una modificación en las rutas de los archivos .js y .css locales.

En cuanto a la memoria además del resumen, redacté, en lo que concierne al capítulo 1, la sección de “Estructura del documento”. En el capítulo 2 me encargué de los puntos 2.1 y 2.2. El capítulo 3 relacionados con las herramientas lo elaboré al completo. El capítulo 5 escribí las tres primeras secciones ligadas a la parte del servidor de la aplicación y la librería spaCy, salvo el punto 5.1.1. Junto con la ayuda de mi compañera, redactamos el capítulo 6 para explicar las conclusiones y posibles mejoras. Finalmente traduje la parte del resumen y palabras claves al inglés, así como las conclusiones y trabajo futuro con mi compañera.

Bibliografía

*Los libros son compañeros, maestros,
magos y banqueros de los tesoros de la
mente.*

Bárbara Wertheim Tuckman

DE LOS ANDES, C. U. *¿Qué es el coronavirus?*, tipo @ONLINE. 2020.

EUROPE, I. Información para todos. 2019.

GALICIA, P. I. Los servicios del banco, tipo @online. 2020.

GARCÍA MUÑOZ, S. *Lectura Fácil - Métodos de redacción y evaluación.* 2013.

INCLUSIÓN, P. Guía de uso de metro de madrid, tipo @online. 2019.

MARTÍN VALDIVIA, M. T., MARTÍNEZ CÁMARA, E., BARBU, E., UREÑA LÓPEZ, L. A., MOREDA, P. y LLORET, E. Proyecto FIRST (Flexible Interactive Reading Support Tool): desarrollo de una herramienta para ayudar a personas con autismo mediante la simplificación de textos. 2014.

NOMURA, M., TRONBACKE, B., NIELSEN, G., OF LIBRARY ASSOCIATIONS, I. F. y OF LIBRARIES SERVING DISADVANTAGED PERSONS, I. S. *Guidelines for Easy-to-read Materials*. IFLA professional reports. IFLA Headquarters, 2010. ISBN 9789077897423.

DEL PRADO, M. N. Guía accesible. 10 obras maestras, tipo @online. 2020a.

DEL PRADO, M. N. Plano accesible. 10 obras maestras, tipo @online. 2020b.

RELLO, L. *Superar la dislexia: Una experiencia personal a través de la investigación*. Educación. Grupo Planeta, 2018. ISBN 9788449335150.

SAGGION, H. *Automatic Text Simplification*. Morgan and Claypool Publishers, 2017.

SAGGION, H., GÓMEZ MARTÍNEZ, E., ETAYO GIL, E., ANULA REBOLLO, A. y BOURG, L. Text simplification in Simplext: making texts more accessible. 2011.