

---

# Mejora de la Comprensión Lectora mediante Analogías para la Inclusión

---



**Trabajo de Fin de Grado  
Curso 2018–2019**

**Autor**

Irene Martín Berlanga  
Pablo García Hernández

**Director**

Virginia Francisco Gilmartín  
Gonzalo Rubén Méndez Pozo

**Grado en Ingeniería de Software**

**Facultad de Informática**

**Universidad Complutense de Madrid**



# Mejora de la Comprensión Lectora mediante Analogías para la Inclusión

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

**Autor**  
**Irene Martín Berlanga**  
**Pablo García Hernández**

**Director**  
**Virginia Francisco Gilmartín**  
**Gonzalo Rubén Mendez Pozo**

*Dirigida por el Doctor*  
**Virginia Francisco Gilmartín**  
**Gonzalo Rubén Mendez Pozo**

**Grado en Ingeniería de Software**  
**Facultad de Informática**  
**Universidad Complutense de Madrid**

**18 de mayo de 2019**



# Autorización de difusión

Los abajo firmantes, matriculados en el Grado de Ingeniería de Software de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo de Fin de Grado: "Mejora de la Comprensión Lectora mediante Analogías para la Inclusión", realizado durante el curso académico 2018-2019 bajo la dirección de Virginia Francisco Gilmartín y Gonzalo Rubén Mendez Pozo en el Departamento de Ingeniería de Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Nombre Del Alumno

18 de mayo de 2019



# Dedicatoria

Texto de la dedicatoria...



# Agradecimientos

Texto de los agradecimientos



# Resumen

Resumen en español del trabajo

## Palabras clave

Máximo 10 palabras clave separadas por comas



# Abstract

Abstract in English.

## Keywords

10 keywords max., separated by commas.



# Índice

<b>1. Introduction</b>	<b>1</b>
<b>1. Introducción</b>	<b>3</b>
1.1. Motivación . . . . .	3
1.2. Objetivos . . . . .	4
1.3. Metodología de gestión del Proyecto . . . . .	5
1.4. Estructura de la memoria . . . . .	6
<b>2. Estado de la Cuestión</b>	<b>9</b>
2.1. Procesamiento del Lenguaje Natural . . . . .	10
2.1.1. ConceptNet . . . . .	12
2.1.2. Thesaurus . . . . .	16
2.1.3. Thesaurus Rex . . . . .	17
2.1.4. Metaphor Magnet . . . . .	20
2.1.5. WordNet . . . . .	22
2.2. Lectura Fácil . . . . .	24
2.3. Figuras retóricas . . . . .	26
2.4. Servicios Web . . . . .	27
2.4.1. Tipos de Servicios Web . . . . .	27
2.4.2. Arquitectura Servicios Web . . . . .	29
2.4.3. Ventajas de los Servicios Web . . . . .	29
2.4.4. Desventajas de los Servicios Web . . . . .	30
<b>3. Herramientas Utilizadas</b>	<b>31</b>
3.1. Django . . . . .	31
3.2. SpaCy . . . . .	32
<b>4. Aprende Fácil</b>	<b>35</b>
4.1. Diseño de la Aplicación . . . . .	35

4.1.1. Primera Iteración: Iteración Competitiva . . . . .	36
4.1.2. Segunda Iteración: Evaluación con Expertos . . . . .	39
4.2. Arquitectura . . . . .	41
4.3. Base de datos . . . . .	41
4.4. Back end . . . . .	41
4.4.1. Red Semántica Utilizada . . . . .	43
4.5. Servidor de Base de Datos . . . . .	44
4.6. Servicio Web para obtener sinónimos fáciles . . . . .	46
4.7. Servicio Web para obtener hipónimos fáciles . . . . .	47
4.8. Servicio Web para obtener hiperónimos fáciles . . . . .	48
4.9. Servicio Web para obtener una metáfora . . . . .	49
4.10. Servicio Web para obtener un símil . . . . .	49
4.11. Front end . . . . .	50
4.12. Evaluación . . . . .	50
<b>5. Trabajo Realizado</b>	<b>63</b>
5.1. Trabajo realizado por Irene . . . . .	63
5.2. Trabajo realizado por Pablo . . . . .	64
<b>6. Conclusiones y Trabajo Futuro</b>	<b>65</b>
<b>6. Conclusions and Future Work</b>	<b>67</b>
<b>A. Título</b>	<b>69</b>
<b>B. Título</b>	<b>71</b>
<b>Bibliografía</b>	<b>73</b>

# Índice de figuras

1.1.	Tablero de tareas al inicio del proyecto . . . . .	6
2.1.	Ejemplo de Red Semántica . . . . .	11
2.2.	Ejemplo de Red de Marco . . . . .	12
2.3.	Ejemplo de Red IS-A . . . . .	12
2.4.	Ejemplo de Grafo Conceptual . . . . .	13
2.5.	Resultados de ConcepNet para la palabra <i>chaqueta</i> . . . . .	13
2.6.	Resultados de ConcepNet para la palabra polisémica <i>book</i> . .	17
2.7.	Resultados búsqueda Thesaurus Rex con la palabra <i>house</i> . .	19
2.8.	Resultados búsqueda Metaphor Magnet con la palabra <i>house</i> .	21
2.9.	Resultados de la búsqueda en EuroWordNet para la palabra <i>casa</i> . . . . .	23
2.10.	Logo Lectura Fácil . . . . .	24
3.1.	Ejemplo de clasificación de palabras . . . . .	33
4.1.	Prototipo de Pablo mostrando resultado más común para la palabra vehículo . . . . .	40
4.2.	Prototipo de Pablo mostrando resultado más común para la palabra vehículo y un ejemplo . . . . .	41
4.3.	Prototipo de Pablo mostrando todos los resultados para la palabra vehículo . . . . .	42
4.4.	Prototipo de Pablo mostrando todos los resultados para la palabra vehículo junto con pictogramas . . . . .	43
4.5.	Prototipo de Irene mostrando resultado más común para la palabra vehículo . . . . .	44
4.6.	Prototipo de Irene mostrando resultado más común para la palabra vehículo y una definición . . . . .	45
4.7.	Prototipo de Irene mostrando todos los resultados para la palabra portero . . . . .	46

4.8. Prototipo de Irene mostrando todos los resultados para la palabra portero junto con pictogramas . . . . .	51
4.9. Prototipo mostrando resultado más común para la palabra portero . . . . .	52
4.10. Prototipo Versión 1 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo separados	53
4.11. Prototipo Versión 2 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo a simple vista . . . . .	54
4.12. Prototipo Versión 3 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo en un mismo botón . . . . .	55
4.13. Prototipo mostrando todos los resultados para la palabra portero . . . . .	56
4.14. Prototipo mostrando todos los resultados para la palabra portero incluyendo los pictogramas . . . . .	57
4.15. Tabla de resultados de las pruebas . . . . .	58
4.16. JSON devuelto al buscar los sinónimos fáciles de inmueble . .	58
4.17. JSON devuelto al buscar los hipónimos fáciles de inmueble .	59
4.18. JSON devuelto al buscar los hiperónimos fáciles de inmueble .	60
4.19. JSON devuelto al buscar las metáforas de la palabra inmueble .	61
4.20. JSON devuelto al buscar los símiles de la palabra inmueble .	62

# Índice de tablas



# Índice de Listados

2.1.	JSON devuelto por la API de ConceptNet para la palabra chaqueta . . . . .	15
2.2.	XML devuelto por Thesaurus para la palabra <i>peace</i> . . . . .	16
2.3.	JSON devuelto por Thesaurus para la palabra <i>peace</i> . . . . .	17
2.4.	XML devuelto por Thesaurus Rex para la palabra <i>house</i> . . .	18
2.5.	XML devuelto por Metaphor Magnet para la palabra <i>house</i> .	20
2.6.	Estructura de un mensaje SOAP . . . . .	28



# Chapter 1

## Introduction

Introduction to the subject area.



# Introducción

## 1.1. Motivación

Hoy en día, el español es la segunda lengua más hablada del mundo y actualmente más de 90.000 palabras forman el castellano. Se trata de una lengua con multitud de términos, y que dependiendo del contexto en el que se encuentren, pueden tener múltiples significados. Por ejemplo, la palabra gato puede hacer referencia a un animal o a una herramienta para elevar un coche. Si esto supone una complicación para cualquier persona, para ciertos colectivos afectados por algún trastorno cognitivo lo es aún mucho más, afectándoles en su vida cotidiana, profesional o personal. Por ejemplo, el simple hecho de leer un periódico es una acción bastante difícil para ellos ya que muchas palabras no saben lo que significan. Otro ejemplo podría ser leer un manual de instrucciones, donde se encuentran en la misma situación de no poder entender ciertos conceptos.

Una de las soluciones que se podrían pensar en un primer momento, es buscar su significado en un diccionario. Pero esto no les sirve, puesto que las definiciones que aparecen en muchos casos no utilizan términos o frases sencillas. Por ejemplo, la palabra computadora tiene la siguiente definición en el diccionario: *Máquina electrónica que, mediante determinados programas, permite almacenar y tratar información, y resolver problemas de diversa índole*<sup>1</sup>. Esta definición puede ser bastante complicada de entender por una persona con discapacidad cognitiva ya que utiliza términos más técnicos y es una frase bastante larga, por lo que una solución que se ha pensado para poder solventar dicho problema, es ofrecer una definición que compare el concepto en cuestión con otros conceptos más sencillos ya conocidos. Para hacer estas comparaciones usaremos las figuras retóricas, más concretamente haremos uso de las metáforas, analogías y símiles. Por ejemplo, para la

---

<sup>1</sup><https://dle.rae.es/?id=A4hIGQC>

palabra computadora se podrían obtener los siguientes resultados:

- *Una computadora es un ordenador.* (Metáfora)
- *Una computadora es como una máquina.* (Analogía)
- *Una computadora es fuerte como una piedra.* (Símil)

Creemos que el uso de las figuras retóricas facilitan el entendimiento del concepto. En estas definiciones se utilizarán frases cortas y conceptos sencillos. Además, añadiremos pictogramas a los resultados para llegar a un colectivo más amplio.

Por tanto, lo que proponemos es crear una aplicación que dada una palabra compleja devuelva los resultados mediante comparaciones con conceptos más sencillos y haciendo uso de las figuras retóricas.

## 1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado es crear una aplicación web basada en servicios web que dada una palabra compleja para el usuario devuelva una definición de dicha palabra comparándola con otras palabras más sencillas mediante símiles, analogías o metáforas. Para poder obtener estas definiciones, habrá que estudiar como obtener los términos relacionados de un concepto, así como distinguir de estos resultados cuales son palabras fáciles y difíciles. También habrá que saber que tipos de figuras retóricas existen y cuál utilizar según la relación entre el concepto inicial y los conceptos fáciles, de esta forma se podrá facilitar al usuario final un resultado claro y correcto.

La aplicación estará construida con servicios web que doten de funcionalidad a la aplicación y que sean reutilizables en otras aplicaciones, haciendo así que se puedan adaptar a las distintas necesidades de los usuarios finales. Los servicios web desarrollados estarán disponibles en una API pública para que todo el mundo pueda utilizarlos y puedan servir para que otros desarrolladores integren nuestros servicios en sus aplicaciones.

La aplicación se construirá de manera incremental, añadiendo valor al producto poco a poco. De este modo se podrán testear las distintas hipótesis de trabajo poco a poco y realizar modificaciones de una manera simple para así conseguir una aplicación que se adecúe a las necesidades de los usuarios.

El diseño de la interfaz estará centrado en el usuario, y para ello se debe obtener la mayor información posible sobre el usuario final. De esta forma se realiza un diseño basado en sus necesidades y se obtendrá una mayor satisfacción de este al utilizar la aplicación. Como dicho trabajo está enfocado para personas con discapacidad cognitiva, se debe realizar un diseño que se adapte aún más a sus necesidades y limitaciones.

Por último, no se deben olvidar los objetivos académicos de este trabajo, como pueden ser poner en práctica los conocimientos adquiridos durante el grado y ampliar nuestros conocimientos en distintas áreas.

Alcanzando los objetivos anteriormente descritos, se conseguirá obtener un producto de calidad, con una gran utilidad tanto social como académica, que pueda ayudar a muchos usuarios a aprender conceptos nuevos de nuestro idioma de una manera más sencilla.

### 1.3. Metodología de gestión del Proyecto

Desde el inicio del proyecto se ha buscado la eficiencia y la mejora continua, es por ello que se han tenido reuniones con los directores de este trabajo, cada dos o tres semanas. En estas reuniones se revisaba el trabajo realizado desde la última reunión, se buscaban soluciones a los problemas y dudas que pudieran surgir o plantearse y se fijaban las tareas a realizar hasta la próxima reunión. Por otro lado, ha habido comunicación constante con ambos directores vía email para consultar dudas y también ha habido tutorías para resolver los problemas que iban surgiendo.

En relación a la gestión de configuración se ha utilizado la plataforma de desarrollo online GitHub para llevar el control de versiones del proyecto.

Además, se ha hecho uso de un gestor de tareas que ha servido como radiador de información y así todos los integrantes del proyecto han podido conocer en cada momento el estado del proyecto y de cada una de sus tareas. Para ello, se ha elegido Trello, ya que dispone de una interfaz simple, amigable y que no lleva a confusión a la hora de crear nuevas tareas. Existen dos tipos de tareas en nuestro tablero: las relacionadas con código y las relacionadas con la memoria. Se ha realizado una distinción entre ambas, ya que la forma de cambiar su estado en el tablero varía significativamente. Para hacer esta distinción en las tareas se ha escrito la palabra CÓDIGO o la palabra MEMORIA según corresponda delante de la descripción de la misma. Nuestro tablero de tareas tiene tres columnas:

- TO DO: Tareas a realizar, desgranadas al mayor detalle posible e intentando que éstas sean lo más independientes las unas de las otras. De esta forma, se asegura que cada integrante del equipo trabaja en una tarea específica que no influye en el trabajo del otro compañero.
- En proceso: En el momento en el que un integrante del equipo se asigna una tarea, esta se pasa de la columna TO DO a la columna “En proceso”, lo que indica que se encuentra en proceso de realización y que ningún otro compañero puede ponerse a trabajar en ella.
- En revisión: En esta columna se encuentran las tareas terminadas pero no validadas. La validación depende del tipo de tarea: si la tarea es

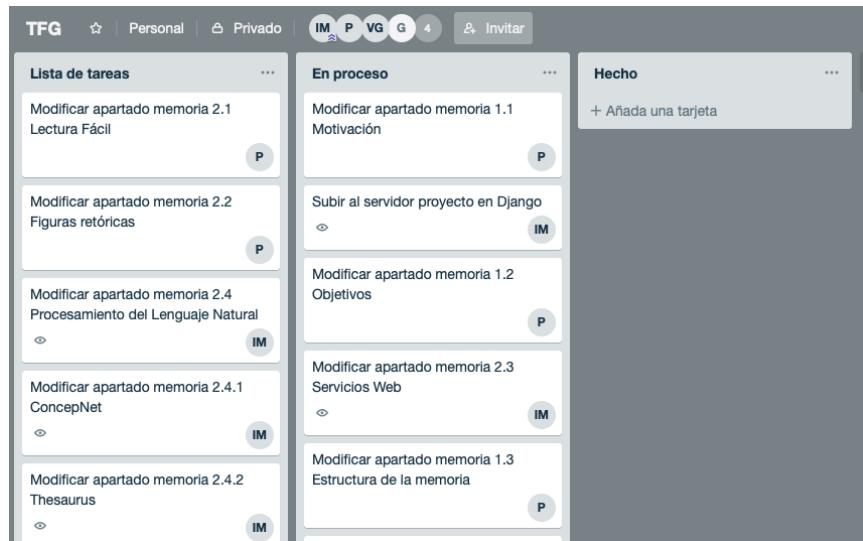


Figura 1.1: Tablero de tareas al inicio del proyecto

de tipo código, la validación la deben hacer los miembros del equipo realizando pruebas para comprobar que realmente cumple su objetivo y no provoca ningún error, y si es de tipo memoria la deben hacer los directores.

- **Hecho:** En esta columna se encuentran las tareas ya validadas por los directores o por los integrantes del equipo en función del tipo de tarea.

En la Figura 1.1 se muestra el estado del tablero al inicio del proyecto.

#### 1.4. Estructura de la memoria

En el **capítulo dos** se presenta el Estado de la Cuestión, en el que se explicará que es la Lectura Fácil y como se aplica y se introducirán los conceptos de Procesamiento del Lenguaje Natural (PLN) y algunas herramientas que sirven para PLN, además se hablará de figuras retóricas y servicios web, en especial qué son, su arquitectura y las ventajas y desventajas de su uso.

En el **capítulo tres** se explicarán las herramientas utilizadas para la creación de este trabajo, como pueden ser Django para el desarrollo de la aplicación y SpaCy para el etiquetado de palabras, donde se explicará que son ambas herramientas, para que se utilicen y sus características principales.

El **capítulo cuatro** se explicará todo lo relacionado con la implementación de la aplicación. Desde los primeros prototipos creados, las evaluaciones con los expertos, la arquitectura de la misma y la explicación tanto de los

servicios web implementados por los integrantes del trabajo para dotar de funcionalidad a la aplicación así como se ha conseguido la funcionalidad de la vista.

En el **capítulo cinco** se describe el trabajo realizado por cada uno de los autores de dicho trabajo..



# Capítulo 2

## Estado de la Cuestión

Como se ha comentado en el anterior capítulo, el objetivo principal de este trabajo es construir una aplicación que ayude a personas con discapacidad cognitiva a poder entender palabras más complejas mediante conceptos más sencillos. Para poder conseguir cumplir dicho objetivo hay que seguir ciertos pasos.

En primer lugar, se deben buscar los conceptos relacionados con la palabra introducida por el usuario. Para ello, se ha utilizado una rama de la Inteligencia Artificial llamada Procesamiento del Lenguaje Natural (PLN). En la sección 2.1 se explicará en qué consiste y las distintas herramientas disponibles para la búsqueda de conceptos relacionados con una palabra específica.

Una vez obtenidas las palabras relacionadas con el concepto introducido, se debe hacer una distinción de cuales de las palabras relacionadas pertenecen al grupo de palabras fáciles y cuales al grupo de palabras difíciles, es decir, distinguir entre aquellas palabras que se utilizan asiduamente y las que no. Esta distinción es necesaria ya que uno de nuestros objetivos es trabajar únicamente con las palabras fáciles para facilitar la compresión del significado de un concepto devolviendo un resultado lo más sencillo posible. Dado que este trabajo está enfocado para personas con discapacidad cognitiva, el concepto de lectura fácil nos ayuda a comprender como debemos trabajar con las palabras fáciles y por ello en la sección 2.2 se explicará que es la lectura fácil y algunas pautas que se pueden seguir para escribir correctamente un texto con esta adaptación, así como la definición de palabras fáciles y difíciles y los documentos que se utilizarán como referencia para las palabras fáciles.

Tras identificar las palabras fáciles relacionadas con el concepto introducido, se utilizarán figuras retóricas para comparar el concepto complejo con otros conceptos más sencillos que ayuden a entender el significado del concepto complejo introducido por el usuario. En la sección 2.3 se hablará de las figuras retóricas y se explicarán los tres tipos fundamentales que se van a utilizar para la realización de este trabajo.

Por último, vamos a hacer uso de servicios web para dotar a la aplicación de funcionalidad. Como se ha comentado en el anterior capítulo, uno de los objetivos tecnológicos de este trabajo es construir la aplicación con servicios web y en la sección 2.4 se explicará detalladamente que es un servicio web, los tipos que existen, sus características principales, su arquitectura y las ventajas de ser utilizados así como sus desventajas.

## 2.1. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es una rama de la Inteligencia Artificial que se encarga de la comunicación entre máquinas y personas mediante el uso del lenguaje natural (entendiendo como lenguaje natural el idioma usado con fines de comunicación por humanos, ya sea hablado o escrito, como pueden ser el español, el ruso o el inglés).

Una de las tareas principales en el Procesamiento del Lenguaje Natural (PLN) es interpretar un texto escrito en lenguaje natural y entender su significado, entendiendo como significado la relación entre una palabra o una frase con el mundo. Para realizar dicha acción no solo es necesario el conocimiento del propio lenguaje en que está escrito el texto sino que también es necesario un conocimiento del mundo. Por tanto, uno de los grandes retos del PLN es la representación del conocimiento. Se deben de buscar técnicas que permitan representar conceptos y relaciones semánticas entre ellos.

Una de las principales técnicas de representación de conocimiento son las redes semánticas, en ellas los conceptos que componen el mundo y sus relaciones se representan mediante un grafo. Las redes semánticas se utilizan para representar mapas conceptuales y mentales (Quillian, 1968). Los nodos están representados por el elemento lingüístico, y la relación entre los nodos sería la arista. Se puede ver un ejemplo en la Figura 2.1, donde el nodo *Oso* representa un concepto, en este caso un sustantivo que identifica a un tipo de animal, y otro nodo *Pelo* el cual también es un sustantivo. La relación entre ambos se ve representada por la arista con valor *tiene*, dando lugar a una característica de este animal: *Oso tiene pelo*.

Existen principalmente tres tipos de redes semánticas (Moreno Ortiz, 2000):

- Redes de Marcos: los enlaces de unión de los nodos son parte del propio nodo, es decir, se encuentran organizados jerárquicamente, según un número de criterios estrictos, como por ejemplo la similitud entre nodos. En la Figura 2.2, se muestra un ejemplo de Red de Marco donde por ejemplo el nodo ave tiene como características que vuela, que tiene plumas y que pone huevos, pero en cambio el nodo avestruz que es un tipo de ave, no puede volar. Por lo que los nodos de ave son las características principales de un ave, aunque no todas tienen por que

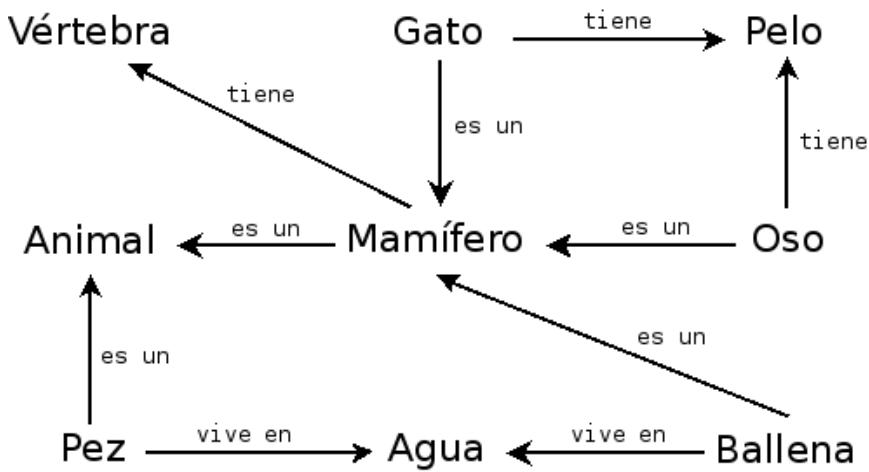


Figura 2.1: Ejemplo de Red Semántica

cumplirlo.

- Redes IS-A: los enlaces entre los nodos están etiquetados con una relación entre ambos. Es el tipo que habitualmente se utiliza junto con las Redes de Marcos. En la Figura 2.3 se muestra una red IS-A en la que se representa que: mujer y hombre son personas, y perro y gato son animales. Por último, tanto personas como animales son seres vivos y una de sus características en común es que tienen pelo.
- Grafos Conceptuales: existen dos tipos de nodos en estas redes: nodos de conceptos, los cuales representan una entidad, un estado o un proceso y los nodos de relaciones, que indican como se relacionan los nodos de concepto. En este tipo de red semántica no existen enlaces entre los nodos con una etiqueta, sino que son los propios nodos los que tienen el significado. Se puede ver un ejemplo en la Figura 2.4 en la cual la frase "*Man biting dog*" quedaría representada (Sowa, 1983). Los cuadrados implican el concepto y el círculo la relación entre ambos, por lo que en el caso de *man* y *bite*, la acción de morder la realiza *man* siendo éste el agente, y la relación entre *bite* y *dog* sería el objeto.

Para el trabajo que queremos realizar, existen varias redes semánticas que disponen de una representación del conocimiento que necesitaremos para relacionar el concepto difícil introducido por el usuario con otros conceptos. A continuación, describiremos las más representativas.

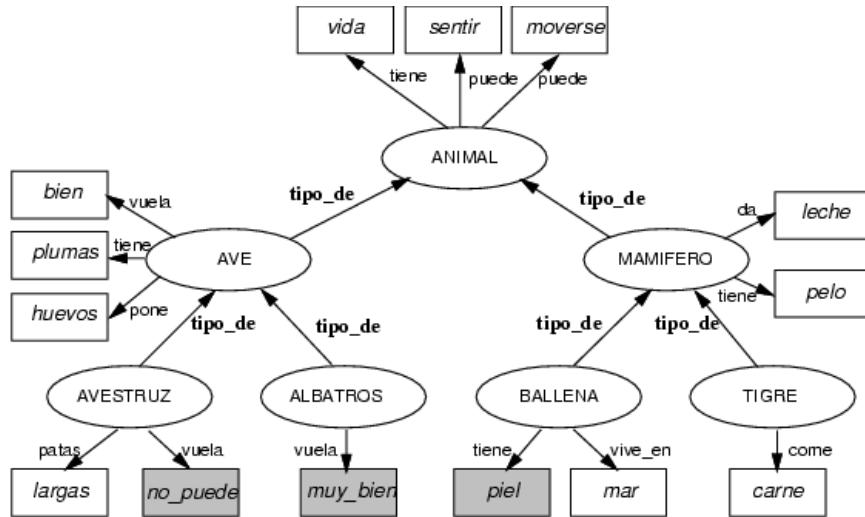


Figura 2.2: Ejemplo de Red de Marco

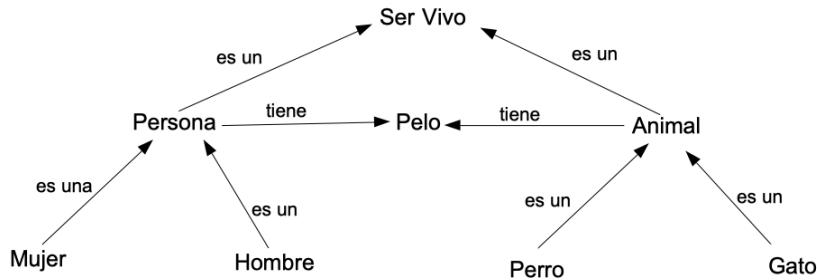


Figura 2.3: Ejemplo de Red IS-A

### 2.1.1. ConceptNet

ConceptNet es una red semántica creada por el MIT (*Massachusetts Institute of Technology*) en 1999, fue diseñada para ayudar a los ordenadores a entender el significado de las palabras. ConceptNet ofrece la posibilidad de obtener de una palabra un listado de sinónimos, términos relacionados, términos derivados, el contexto de la palabra, resultados etimológicamente relacionados, símbolos, etc.

ConceptNet dispone de una aplicación web<sup>1</sup> donde se pueden buscar palabras en distintos idiomas, como el español, el inglés o el chino. En la Figura 2.5 se puede ver la información que devuelve la aplicación ConceptNet para la palabra chaqueta. Como se puede ver en la Figura hay una columna con los

<sup>1</sup> <http://conceptnet.io/>



Figura 2.4: Ejemplo de Grafo Conceptual

Synonyms	Related terms	Derived terms	Word forms
<a href="#">jacket (n, artifact) →</a> <a href="#">jaqueta (n) →</a> <a href="#">jacket (n) →</a> <a href="#">jacket (n) →</a> <a href="#">americana (n) →</a> <a href="#">casaca (n) →</a> <a href="#">americana →</a> <a href="#">casaca →</a> <a href="#">jaqueta →</a> <a href="#">chumpa (n) →</a> <a href="#">jaqueta (n) →</a> <a href="#">blouson (n) →</a> <a href="#">veste (n) →</a> <a href="#">veston (n) →</a>	<a href="#">clothing →</a> <a href="#">jacket →</a> <a href="#">chaquetear →</a> <a href="#">branlette →</a> <a href="#">veste →</a> <a href="#">veston →</a> <a href="#">chaquetear (v) →</a> <a href="#">saco (n) →</a>	<a href="#">chaquetero →</a> <a href="#">chaquetilla →</a> <a href="#">chaquetita →</a>	<a href="#">chaquetas (n) →</a>

Figura 2.5: Resultados de ConcepNet para la palabra chaqueta

sinónimos del concepto en distintos idiomas (americana, *jacket* o *blouson*), otra columna con los términos relacionados (saco o chaquetear) también en distintos idiomas, otra columna que muestra los términos derivados del concepto (chaquetero, chaquetilla o chaquetita) y por último una columna que muestra otra forma de la palabra, en este caso el plural (chaquetas). Para algunos de los resultados de ConcepNet además aparece a la derecha de la palabra y entre paréntesis, una sola letra que indica si es un verbo, un sustantivo o un adjetivo.

ConcepNet no realiza ninguna agrupación de los resultados en función del significado, si no que muestra todo el listado de palabras y en algunas de ellas aparece el contexto al cual se refieren. Por ejemplo, para la palabra arco algunos de los resultados que devuelven son referentes al significado arco de arquitectura y otros son referentes al arco de geometría.

Por otro lado, ConcepNet dispone de un servicio web<sup>2</sup> que devuelve los resultados en formato JSON. Siguiendo con la palabra chaqueta, se pueden ver en el Listado 2.1 los resultados devueltos por el servicio web para chaqueta. El JSON devuelto en este caso consta de cuatro campos principales<sup>3</sup>:

<sup>2</sup><http://api.conceptnet.io>

<sup>3</sup><https://github.com/commonsense/conceptnet5/wiki/AP>

- @context: URL enlazada a un archivo de información del JSON para comprender la API. También puede contener comentarios que pueden ser útiles para el usuario.
- @id: concepto que se ha buscado y su idioma. En nuestro caso, aparece de la siguiente manera: */c/es/chaqueta*, donde *c* significa que es un concepto o término, *es* indica el lenguaje, en este caso, el español y por último *chaqueta* que es la palabra buscada.
- edges: representa una estructura de datos devueltos por ConceptNet compuesta por:
  - @id: describe el tipo de relación que existe entre la palabra introducida y la devuelta. En el Listado 2.1 se indica que la palabra *americana* es un sinónimo de *chaqueta*.
  - @type: define el tipo del id, es decir, si es una relación (edge) o un término (nodo).
  - dataset: URI que representa el conjunto de datos creado.
  - end: nodo destino, que a su vez se compone de:
    - @id: coincide con la palabra del id anterior.
    - @type: define el tipo de id, como se ha explicado anteriormente.
    - label: puede ser la misma palabra buscada o una frase más completa, donde adquiera significado la palabra obtenida.
    - language: lenguaje en el que está la palabra devuelta de la consulta.
    - term: enlace a una versión mas general del propio término. Normalmente, suele coincidir con la URI.
  - license: aporta información sobre como debe usarse la información proporcionada por conceptnet.
  - rel: describe la relación que hay entre la palabra origen y destino, dentro del cual hay tres campos: @id, @type y label, descritos anteriormente.
  - sources: indica porqué ConceptNet guarda esa información, este campo como los anteriores, es un objeto que tiene su propio id y un campo @type, A parte, hay un campo *contributor*, en el que aparece la fuente por la que se ha obtenido ese resultado y por último un campo *process* indicando si la palabra se ha añadido mediante un proceso automático.
  - start: describe el nodo origen, es decir, la palabra que hemos introducido en ConceptNet para que haga la consulta, este campo esta compuesto por elementos ya descritos como son: @id, @type, label, language y term.

- surfaceText: algunos datos de ConceptNet se extraen de texto en lenguaje natural. El valor de surface text muestra lo que era este texto, puede que este campo tenga valor nulo.
- weight: indica la fiabilidad de la información guardada en ConceptNet, siendo normal que su valor sea 1.0. Cuanto mayor sea este valor, más fiables serán los datos obtenidos.
- view: describe la longitud de la lista de paginación, es un objeto con un id propio, y además, aparecen los campos *firstPage* que tiene como valor un enlace a la primera pagina de los resultados obtenidos, y *nextPage* que tiene un enlace a la siguiente página de la lista.

Listado 2.1: JSON devuelto por la API de ConceptNet para la palabra chaqueta

```
{
  "@context": [
    "http://api.conceptnet.io/ld/conceptnet5.6/context.ld.json"
  ],
  "@id": "/c/es/chaqueta",
  "edges": [
    {
      "@id": "/a/[/r/Synonym/,/c/es/chaqueta/n,/c/es/americana/]",
      "@type": "Edge",
      "dataset": "/d/wiktionary/fr",
      "end": {
        "@id": "/c/es/americana",
        "@type": "Node",
        "label": "americana",
        "language": "es",
        "term": "/c/es/americana"
      },
      "license": "cc:by-sa/4.0",
      "rel": {
        "@id": "/r/Synonym",
        "@type": "Relation",
        "label": "Synonym"
      },
      "sources": [
        {
          "@id": "/and/[/s/process/wikiparsec/1/,/s/resource/wiktionary/fr/]",
          "@type": "Source",
          "contributor": "/s/resource/wiktionary/fr",
          "process": "/s/process/wikiparsec/1"
        }
      ],
      "start": {
        "@id": "/c/es/chaqueta/n",
        "@type": "Node",
        "label": "chaqueta",
        "language": "es",
        "sense_label": "n",
        "term": "/c/es/chaqueta"
      },
      "surfaceText": null,
      "weight": 1.0
    },
    ]
  ],
  "view": {
    "@id": "/c/es/chaqueta?offset=0&limit=20",
    "@type": "PartialCollectionView",
    "comment": "There are more results. Follow the 'nextPage' link for more.",
    "firstPage": "/c/es/chaqueta?offset=0&limit=20",
    "nextPage": "/c/es/chaqueta?offset=20&limit=20",
    "paginatedProperty": "edges"
  }
}
```

### 2.1.2. Thesaurus

Thesaurus<sup>4</sup> es una aplicación web que se autodefine como el principal diccionario de sinónimos de la web. Esta página ofrece la posibilidad de obtener los sinónimos de una palabra, pero solamente devuelve resultados en inglés. Aparte del listado de sinónimos, Thesaurus indica que tipo de palabra es y una definición de la misma así como un listado de antónimos y un listado de palabras relacionadas con dicho concepto.

Si la palabra introducida es polisémica, Thesaurus devuelve los resultados por grupos según su connotación. En la Figura 2.6, se puede ver los resultados devueltos por Thesaurus para la palabra *book*, donde este puede utilizarse como un sustantivo o como un verbo. Y dentro de si es un tipo u otro, los divide según el significado, ya sea para utilizarlo como un documento, un diario o como una reserva. Y cada uno se muestran en pestañas distintas para facilitar al usuario la distinción de conceptos.

Por otro lado, esta aplicación proporciona una API<sup>5</sup> tipo RESTful que obtiene los sinónimos de una palabra mediante una petición HTTP GET a la url <http://thesaurus.altervista.org/thesaurus/v1>. Este devuelve los resultados en formato XML o JSON. El contenido de la respuesta es una lista y cada elemento de esta lista contiene un par de elementos: categoría y sinónimos. Este último a su vez contiene una lista de sinónimos separados por el carácter |. Se puede ver en el Listado 2.2 el resultado devuelto por Thesaurus para la palabra *peace* en formato XML y en el Listado 2.3 para la misma palabra, *peace*, pero en formato JSON. Ambos son muy similares, por ejemplo en formato XML se puede ver que devuelve el tipo de categoría de las palabras, en este caso son sustantivos y a continuación aparecen los sinónimos. En caso de que alguna palabra sea un antónimo aparecerá entre paréntesis al lado de la misma, como ocurre con la palabra *war*. Por otro lado, el formato JSON devuelve dentro del campo *category* / *categoría* todos los sinónimos, y en caso de ser un antónimo aparecerá de la misma forma que en el formato XML.

Listado 2.2: XML devuelto por Thesaurus para la palabra *peace*

```
<response>
<list>
<category>(noun)</category>
<synonyms> order|war (antonym) </synonyms>
</list>
<list>
<category>(noun)</category>
<synonyms> harmony |concord |concordance </synonyms>
</list>
<list>
<category>(noun)</category>
<synonyms> public security |security </synonyms>
</list>
<list>
<category>(noun)</category>
```

<sup>4</sup><https://www.thesaurus.com/>

<sup>5</sup><http://thesaurus.altervista.org/>

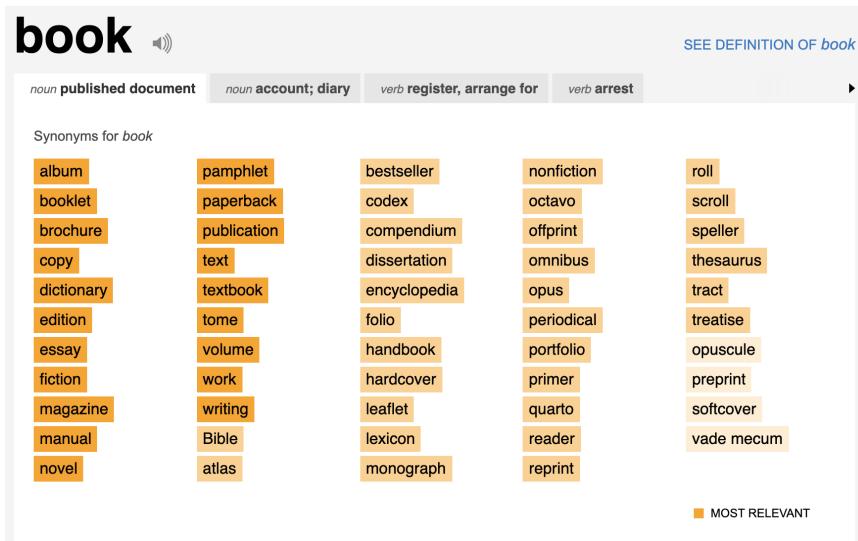


Figura 2.6: Resultados de ConcepNet para la palabra polisémica *book*

```
<synonyms> peace treaty | pacification | treaty | pact | accord </synonyms>
</list>
```

Listado 2.3: JSON devuelto por Thesaurus para la palabra *peace*

```
{
  "response": [
    {
      "list": [
        {
          "category": "(noun)", "synonyms": "order|war (antonym)" }
      ],
      "list": [
        {
          "category": "(noun)", "synonyms": "harmony|concord|concordance" }
      ],
      "lista": [
        {
          "categoria": "(noun)", "synonyms": "public security|security" }
      ],
      "lista": [
        {
          "categoria": "(noun)", "synonyms": "peace treaty | pacification | treaty | pact | accord" }
      ]
    }
  ]
}
```

### 2.1.3. Thesaurus Rex

Thesaurus Rex<sup>6</sup> es una red semántica que solo admite palabras en inglés y que permite obtener las palabras relacionadas con una palabra o las categorías que comparten dos palabras.

En Thesaurus Rex las palabras tienen categorías y modificadores. Las categorías son sustantivos que definen a la palabra introducida por el usuario,

<sup>6</sup><http://ngrams.ucd.ie/therex3/>

como se puede ver en la Figura 2.7, la palabra *house* tiene asociadas las categorías *structure*, *object*, *item* o *building*. Los modificadores son adjetivos que describen a la palabra, siguiendo con el mismo ejemplo, *house* tiene como modificadores *permanent*, *fixed* o *wooden*. Por último, dispone de un listado de las categorías más utilizadas por los hablantes de dicha lengua, como pueden ser *permanent-structure*, *inanimate-object*, *everyday-object*, etc...

En el caso de introducir dos palabras en la aplicación, por ejemplo *coffee* y *cola*, la aplicación devuelve las categorías que comparten dichos conceptos, en este caso serían *cold-beverage*, *dark-beverage* o *stimulating-beverage*. Si se introduce una palabra polisémica, como puede ser *book*, Thesaurus Rex no realiza ninguna distinción entre que resultados corresponden a los distintos significados del concepto buscado.

La aplicación también dispone de una API pública que devuelve los resultados en formato XML. En el Listado 2.4 se muestra el XML devuelto por la aplicación para la palabra *house*. El XML devuelto consta de tres grandes bloques: *Categories*, *Modifiers* y *CategoryHeads*. Todos los resultados tienen un peso (*weight*) asignado, esto significa que cuanto mayor sea el peso mayor es la similitud con el concepto dado. Los campos que se encuentran dentro del apartado *categories* son los resultados más utilizados en ese momento por los hablantes y que Thesaurus Rex ha encontrado, como por ejemplo *permanent-structure*. Los que se encuentran dentro de *modifiers*, como se ha comentado anteriormente son atributos del concepto a buscar, como por ejemplo *fixed*. Y por último, los que se encuentran en *categoryHeads* son las categorías más simples que se han encontrado para dicho concepto, como por ejemplo *structure*.

Thesaurus Rex utiliza el contenido de la web para generar sus resultados, con lo cual la información disponible no es fija, sino que varía según los datos de la web. La ventaja de utilizar esta herramienta es que se encuentra en continua actualización, pero el inconveniente es que en algunos casos la información puede resultar un poco extraña dado que se crea semiautomáticamente desde contenido de la web (Veale y Li, 2013). Por ejemplo, para la palabra *house*, uno de los resultados es *sphere*, que en principio no tiene ninguna relación con *house*.

Listado 2.4: XML devuelto por Thesaurus Rex para la palabra *house*

```
<MemberData>
<Categories kw="house">
<Category weight="91"> large:object </Category>
<Category weight="307"> inanimate:object </Category>
<Category weight="261"> everyday:object </Category>
<Category weight="154"> sensitive:area </Category>
<Category weight="318"> permanent:structure </Category>
<Category weight="194"> permanent:construction </Category>
<Category weight="148"> permanent:installation </Category>
<Category weight="98"> fixed:object </Category>
</Categories>

<Modifiers kw="house">
<Modifier weight="8"> recognizable </Modifier>
<Modifier weight="9"> relevant </Modifier>
```

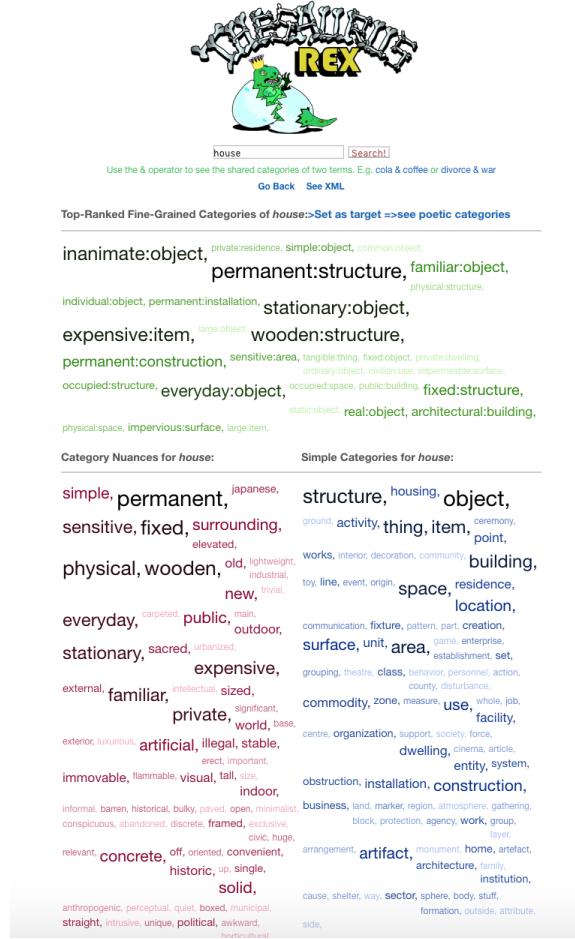


Figura 2.7: Resultados búsquedas Thesaurus Rex con la palabra *house*

```

<Modifier weight="863"> permanent </Modifier>
<Modifier weight="-15"> moderate </Modifier>
<Modifier weight="477"> fixed </Modifier>
<Modifier weight="5"> odd </Modifier>
<Modifier weight="7"> archaeological </Modifier>
<Modifier weight="5"> electrical </Modifier>
</Modifiers>

<CategoryHeads kw="house">
<CategoryHead weight="6"> protection </CategoryHead>
<CategoryHead weight="40"> obstruction </CategoryHead>
<CategoryHead weight="5"> whole </CategoryHead>
<CategoryHead weight="3320"> object </CategoryHead>
<CategoryHead weight="2340"> structure </CategoryHead>
<CategoryHead weight="98"> commodity </CategoryHead>
<CategoryHead weight="713"> thing </CategoryHead>
<CategoryHead weight="2"> theatre </CategoryHead>
</CategoryHeads>
</MemberData>

```

### 2.1.4. Metaphor Magnet

Metaphor Magnet es una aplicación web<sup>7</sup>, que crea metáforas a partir de una palabra y que solo está disponible para el inglés. Metaphor Magnet permite al usuario introducir palabras y, ayudándose de los n-gramas de Google (Veale y Li, 2012), busca e interpreta las distintas metáforas que existan sobre dicha palabra.

Un n-grama (Manning y Schütze, 1999) es una subsecuencia de n elementos consecutivos en una secuencia dada y a su vez estos pueden ser bigramas, trigramas, etc... Metaphor Magnet lo que hace es utilizando dichos n-gramas busca en la Web cuantas veces aparece el concepto introducido y cuantas veces aparece junto con otra palabra en un n-grama. De esta forma, se obtienen los resultados más utilizados por los hablantea. En la Figura 2.8 se puede ver el resultado obtenido en la aplicación para la palabra *house*. En este caso la aplicación devuelve métaforas propias del concepto, como *protecting:home*. En el caso de introducir una palabra polisémica, por ejemplo *book*, Metaphor Magnet no realiza ninguna agrupación según los distintos significados, si no que muestra todos los resultados juntos.

Metaphor Magnet también dispone de una API pública que devuelve los resultados en formato XML. En el Listado 2.5 se puede ver el resultado devuelto por la API para la palabra *house*. En el XML devuelto aparece la etiqueta *<Source Name>* seguido de la palabra a buscar, en este caso *house*. Otra etiqueta *<Score>* que muestra un número, cuanto mayor sea este número más relacionado será con el concepto introducido, por ejemplo "*tall:building*" tiene un *score* de 86 mientras que "*intuitive:natural*" tiene un *score* de 25, indicando así que es uno de los resultados menos relacionados con la palabra *house*. Por último, la etiqueta *<Text>* contiene la metáfora, por ejemplo "*protecting:home*".

Listado 2.5: XML devuelto por Metaphor Magnet para la palabra *house*

```
<Metaphor>
<Source Name="house">
<Text> towering : mountain </Text>
<Score> 88 </Score>
</Source>
<Source Name="house">
<Text> protecting : home </Text>
<Score> 86 </Score>
</Source>
<Source Name="house">
<Text> tall : building </Text>
<Score> 86 </Score>
</Source>
<Source Name="house">
<Text> charming : castle </Text>
<Score> 85 </Score>
</Source>
<Source Name="house">
<Text> beautiful : tree </Text>
<Score> 84 </Score>
</Source>
<Source Name="house">
```

<sup>7</sup><http://ngrams.ucd.ie/metaphor-magnet-acl/>



**Source Metaphors: house**

---

killing:abattoir, pious:clergyman, **towering:mountain**, packed:olive, **protecting:home**, leading:general, **charming:castle**, organized:computer\_network, **tall:building**, sprawling:forest, charming:kitten, threatening:pathogen, **beautiful:tree**, inspiring:vision, loving:puppy, threatening:enemy, satisfying:settlement, **strong:elephant**, connected:cellphone, cramped:bathroom, threatening:competitor, **charming:mansion**, threatening:curse, crushing:seabuck, lending:vessel, loving:duty, entertaining:soap\_opera, accountable:government, **charming:movie\_star**, looming:crisis, threatening:cloud, **shining:star**, calm:retreat, **strong:rock**, pink:skin, dreary:failure, registered:vehicle, charming:playboy, entertaining:carnival, charming:swindler, charming:illusion, solid:concrete, powerful:sledghammer, inspired:idea, entertaining:magazine, respected:veteran, inspiring:gun, threatening:catastrophe, screaming:horn, **swinging:club**, appealing:privilege, attacking:desperado, **shining:mirror**, haunting:reminder, entertaining:movie, **loving:memory**, damaging:criticism, charming:lord, integrated:combination, inspiring:founder, protecting:key, **caring:woman**, charming:king, dreary:cave, entertaining:farce, **screaming:pig**, threatening:monster, crowded:market, threatening:giant, charming:scoundrel, commanding:controller, quaint:bistro, respected:expert, creeping:fungus, sparkling:sunbeam, attacking:party, healing:cure, threatening:thug, **invaded:ice**, shining:candle, beautiful:sunrise, **evading:light**, **invader**, attacking:belligerent, **protecting:fence**, threatening:danger, haunting:ballad, **charming:resort**, threatening:mob, powerful:supercomputer, crowded:mail, fruitful:crop, smiling:lurker, dignified:inhaler, dreary:tunnel, **trading:dealer**, wrecked:schooner, respected:yogi, teeming:ant hill, threatening:panther, organized:religion, entertaining:nightclub, **powerful:freight\_train**, inspiring:hymn, sweet:child, charming:gentleman, haunting:fear, **Strong:bull**, **shining:skyscraper**, respected:religious\_teacher, inspiring:symbol, charming:knaves, **crowded:hive**, surprising:coincidence, shining:dew, **threatening:weed**, teaching:educator, sweet:sweetheart, damaging:glitch, comforting:balm, appealing:banquet, majestic:cowl, exotic:aquarium, supporting:bone, charming:picnic, loving:husband, planning:developer, admired:sportsman, threatening:thunder, storm, reigning:bishop, loving:poet, comforting:pillow, integrated:mixture, unwanted:pestilence, **big:barn**, unwanted:waste, understanding:confidante, teaching:consultant, entertaining:melodrama, loving:hug, light:bee, supporting:pole, **strong:tank**, attacking:gator, **shining:light**, inspired:madman, raving:fanatic, charming:juliet, sweet:birthday, quaint:boutique, attacking:Mongol, dumb:gourd, striking:absurdity, inspiring:principle, loving:marchion, charming:jewel, **reinforced:door**, sparkling:liquid, celebrated:shrine, entertaining:holiday, tender:feet, **mignon**, pure:newborn, pioneering:revolutionary, dreary:depression, entertaining:game, teaching:coach, sparkling:writer, threatening:epidemic, charming:plaza, terrifying:challenge, charming:hunk, exuding:whit, exuding:dynam, attacking:teef, **inspiring:art**, handsome:cock, prestigious:penthouse, towering:rampart, respected:priest, sweet:rosebud, happy:squirrel, reigning:grandmaster, inspiring:idol, pointed:pen, charming:fairy\_tale, inspiring:inspiration, charming:luxury\_hotel, light:feather, respected:dignitary, ruling:court, glowing:thunderbolt, **packed:box**, charming:hearthrob, respected:thoroughbred, shining:brand, **hilkacountrywide**, beautiful:blimbo, **strong:stone**, intriguing:labyrinth, charming:gazebo, shining:monument, threatening:gun, inspiring:conqueror, fascinating:museum, **strong:horse**, powerful:computer, respected:wizard, entertaining:toy, towering:dragon, respected:profession, gloomy:pit, **loving:house**, **nuce**, fighting:fundamentalist, fast:fox, threatening:pitbull, fighting:defender, charming:actress, damaging:infestation, comforting:fleece, entertaining:story, smiling:kid, distinguished:minister, manufacturing:firm, inspiring:manifesto, loving:tribute, charming:babe, charming:picture, postcard

Figura 2.8: Resultados búsqueda Metaphor Magnet con la palabra *house*

```
<Text> charming:mansion </Text>
<Score> 83 </Score>
</Source>
<Source Name="house">
<Text> strong:rock </Text>
<Score> 80 </Score>
</Source>
<Source Name="house">
<Text> strong:elephant </Text>
<Score> 80 </Score>
</Source>
<Source Name="house">
<Text> powerful:locomotive </Text>
<Score> 57 </Score>
</Source>
<Source Name="house">
<Text> inspiring:manifesto </Text>
<Score> 41 </Score>
</Source>
<Source Name="house">
<Text> intuitive:natural </Text>
<Score> 25 </Score>
</Source>
</Metaphor>
```

### 2.1.5. WordNet

WordNet es un *corpus*<sup>8</sup> perteneciente a NLTK (*Natural Language Toolkit*)<sup>9</sup> que almacena sustantivos, verbos, adjetivos y adverbios ignorando preposiciones, determinantes y otras palabras funcionales. Además está disponible en varios idiomas como el español, el inglés o el francés. Los conceptos se agrupan en conjuntos de sinónimos cognitivos llamados *synsets*, es decir, se agrupan según su significado. Por ejemplo, la palabra *guardia* tiene el siguiente grupo de sinónimos cognitivos para un *synset*: “defensor, guardián, protector y tutor” y para otro *synset* el siguiente grupo de sinónimos cognitivos: “velador, sereno, guarda de seguridad y vigilante”. A su vez, cada *synset* contiene:

- Hiperónimos: Palabras cuyo significado está incluido en el de otras<sup>10</sup>. Por ejemplo, *mamífero* es hiperónimo de *gato* y de *perro* ya que los gatos y los perros pertenecen al conjunto de los mamíferos.
- Hipónimos: Palabras cuyo significado incluyen el de otra<sup>11</sup>. Por ejemplo, *gato* es hipónimo de *mamífero* ya que está incluido dentro del conjunto de los mamíferos.
- Holónimos: Palabras que representan el todo respecto a una parte. Por ejemplo, *coche* es el holónimo de *rueda*, *volante* y *acelerador* ya que forman parte de un todo, que es el coche.
- Antónimos: Palabras cuyo significado es totalmente opuesto al concepto inicial. Por ejemplo, el antónimo de *alegre* sería *triste*.
- Definición de la palabra: Oraciones cortas que explican el significado de la palabra. Esta información no aparece siempre.

Existen varias aplicaciones web que implementan dicho *corpus*, una de las más completas es EuroWordNet, ya que está disponible en varios idiomas y permite extraer sinónimos, antónimos, hiperónimos, hipónimos y holónimos. Además de algunas oraciones de ejemplo, y algunas definiciones de los *synsets* obtenidos. En la Figura 2.9 aparece la información devuelta por EuroWordNet para la palabra *casa*. Los datos obtenidos son:

- *Offset del synset*: es un código que lo identifica inequívocamente y que finaliza con una letra, la cual describe la categoría gramatical de los sinónimos devueltos. Por ejemplo, un *synset* de la palabra *casa* tiene asociado el *offset spa-30-02913152-n*, donde la “n” hace referencia a *noun*, esto quiere decir que es un sustantivo.

---

<sup>8</sup>Colección de documentos de texto

<sup>9</sup>Conjunto de bibliotecas y programas para el Procesamiento del Lenguaje Natural

<sup>10</sup><https://dle.rae.es/?id=KRW1qe2>

<sup>11</sup><https://dle.rae.es/?id=KU5UAn5>

The screenshot shows the EuroWordNet interface with the search term 'casa' entered. On the left, a tree diagram of semantic relations is visible, with 'near\_synonym' expanded to show 'edificio', 'inmueble', 'construcción', and 'edificación'. On the right, a search interface includes dropdowns for 'Look up' (set to 'Spanish\_3.0') and 'Gloss' (set to 'Spanish\_3.0'). Below these are checkboxes for 'Score', 'Rels', and 'Full', and dropdowns for 'English\_3.0', 'Catalan\_3.0', 'Basque\_3.0', 'Portuguese\_3.0', and 'Galician\_3.0'. The main results area displays two entries:

- spa-30-02913152-n** 298 **edificio\_1** una estructura que tiene un techo, paredes y se encuentra más o menos permanente en un solo lugar; *había un edificio de tres pisos en la esquina; se trataba de un edificio imponente*;
- spa-30-03544360-n** 68 **casa\_2** una vivienda que sirve de residencia para una o más familias; *el tiene una casa en Cape Cod;*

Figura 2.9: Resultados de la búsqueda en EuroWordNet para la palabra *casa*

- Sinónimos: listado con todos los sinónimos del *synset*. Por ejemplo en un mismo *synset* se obtiene edificio, inmueble, construcción y edificación.
- Definición: sirve para entender el concepto. Por ejemplo, en un mismo *synset* se obtiene la siguiente definición: “Una estructura que tiene un techo, paredes y se encuentra más o menos permanente en un solo lugar”.
- Ejemplo: En el caso de que la definición no sea suficientemente clara para el usuario, el ejemplo ayuda a comprender mejor el significado del concepto. Por ejemplo, para un *synset* se obtiene el siguiente ejemplo: “*Había un edificio de tres pisos en la esquina*”.

Si se introduce una palabra polisémica, EuroWordNet devuelve tantos *synsets* como significados tenga la palabra. Por ejemplo, para la palabra *portero* EuroWordNet devolverá en un *synset*: “conserje, guardabarrera y ostiario” haciendo referencia a portero como profesión y en otro *synset*: “arquero, guardameta, guardavalla y golero” que son los resultados para portero como jugador.



Figura 2.10: Logo Lectura Fácil

## 2.2. Lectura Fácil

Se llama lectura fácil<sup>12</sup> a aquellos contenidos que han sido resumidos y reescritos con lenguaje sencillo y claro, de forma que puedan ser entendidos por personas con discapacidad cognitiva o discapacidad intelectual. Es decir, es la adaptación de textos, ilustraciones y maquetaciones que permite una mejor lectura y comprensión.

La lectura fácil surgió en Suecia en el año 1968, donde se editó el primer libro en la Agencia de Educación en el marco de un proyecto experimental. A continuación, en 1976, se creó en el Ministerio de Justicia un grupo de trabajo para conseguir textos legales más claros. En 1984 nació el primer periódico en lectura fácil, titulado "8 páginas", que tres años más tarde, en 1987, se publicó de forma permanente en papel hasta que empezó a editarse en la web. En el año 2013, en México se produce la primera sentencia judicial en lectura fácil<sup>13</sup>. En la actualidad, podemos distinguir los documentos en lectura fácil gracias al logo de la Figura 2.10.

Los documentos escritos en Lectura Fácil (Nomura et al., 2010) son documentos de todo tipo que siguen las directrices internacionales de la IFLA<sup>14</sup> y de Inclusion Europe<sup>15</sup> en cuanto al contenido y la forma. Algunas pautas a seguir para escribir correctamente un texto en Lectura Fácil son (García Muñoz, 2012):

- Evitar mayúsculas fuera de la norma, es decir, escribir en mayúsculas sólo cuando lo dicten las reglas ortográficas, como por ejemplo, después de un punto o la primera letra de los nombres propios.
- Deben evitarse el punto y seguido, el punto y coma y los puntos

<sup>12</sup><https://www.discapnet.es/areas-tematicas/diseno-para-todos/accesibilidad-de-comunicacion/lectura-facil>

<sup>13</sup><https://dilofacil.wordpress.com/2013/12/04/el-origen-de-la-lectura-facil/>

<sup>14</sup>International Federation of Library Associations and Institutions

<sup>15</sup>Una asociación de personas con discapacidad intelectual y sus familias en Europa

suspensivos. El punto y aparte hará la función del punto y seguido.

- Evitar corchetes y signos ortográficos poco habituales, como por ejemplo: %, & y /.
- Evitar frases superiores a 60 caracteres y utilizar oraciones simples. Por ejemplo, la oración *Caperucita ha ido a casa de su abuela y ha desayunado con ella* es mejor dividirla en dos oraciones simples: *Caperucita ha ido a casa de su abuela* y *Caperucita ha desayunado con ella*.
- Evitar tiempos verbales como: futuro, subjuntivo, condicional y formas compuestas.
- Utilizar palabras cortas y de sílabas poco complejas. Por ejemplo: casa, gato, comer o mano.
- Evitar abreviaturas, acrónimos y siglas.
- Alinear el texto a la izquierda.
- Incluir imágenes y pictogramas a la izquierda y su texto vinculado a la derecha.
- Evitar la saturación de texto e imágenes.
- Utilizar uno o dos tipos de letra como mucho.
- Tamaño de letra entre 12 y 16 puntos.
- Si el documento está paginado, incluir la paginación claramente y reforzar el mensaje de que la información continúa en la página siguiente.

Se debe también hacer hincapié en la distinción entre palabras fáciles y complejas (García Muñoz, 2012), puesto que son de gran importancia para la lectura fácil. Las palabras complejas son aquellas que no se utilizan a menudo, como por ejemplo: melifluo o inefable. Las palabras complejas deben descartarse en la lectura fácil, y en su lugar se deben introducir palabras fáciles, que son aquellas que se utilizan asiduamente. La RAE (Real Academia Española) dispone de un corpus<sup>16</sup> donde se encuentran varios documentos en los que se incluyen las mil palabras más usadas, las cinco mil palabras más usadas y las diez mil palabras más usadas por los hablantes de lengua española. En estos listados se incluyen verbos, sustantivos, preposiciones, pronombres, adjetivos, etc. Para este trabajo, es fundamental la distinción entre palabras fáciles y palabras complejas, por ello los documentos facilitados por la RAE de las 1.000, 5.000 y 10.000 palabras más usadas se tendrán como

<sup>16</sup><http://corpus.rae.es/lfrecuencias.html>

referencia de palabras fáciles y por tanto se utilizarán para poder obtener los resultados de la aplicación. Aunque de todas las palabras de estos documentos, se trabajará únicamente con verbos, sustantivos y adjetivos.

### 2.3. Figuras retóricas

Las figuras literarias (o retóricas) se podrían definir como formas no convencionales de utilizar las palabras, de manera que, aunque se emplean con sus acepciones habituales, se acompañan de algunas particularidades fónicas, gramaticales o semánticas, que las alejan de ese uso habitual, por lo que terminan por resultar especialmente expresivas (Galiana y Casas, 1994). Según la RAE (Real Academia Española)<sup>17</sup>, “*la retórica es el arte de bien decir, de dar al lenguaje escrito o hablado eficacia bastante para deleitar, persuadir o conmover*”. Las tres principales figuras retóricas son la metáfora, el símil y la analogía. Estas figuras se basan en la comparación de dos conceptos (Galiana y Casas, 1994): el origen (o tenor), que es el término literal (al que la metáfora se refiere) y el de destino (o vehículo), que es el término figurado. La relación que hay entre el tenor y el vehículo se denomina fundamento. Por ejemplo, en la metáfora *Tus ojos son dos luceros*, *ojos* es el tenor, *luceros* es el vehículo y el fundamento es la belleza de los ojos.

En este trabajo se van a utilizar los tres tipos de figuras retóricas mencionados anteriormente (Calleja, 2017):

- Metáfora: Utiliza el desplazamiento de características similares entre dos conceptos con fines estéticos o retóricos. Por ejemplo, cuando el tiempo de una persona es muypreciado se dice: “Mi tiempo es oro”.
- Símil: Realiza una comparación entre dos términos usando conectores (por ejemplo, *como*, *cual*, *que*, o verbos). Por ejemplo, cuando nos referimos a una persona que es muy corpulenta, se dice: “Es como un oso”, ya que los osos son muy grandes.
- Analogía: Es la comparación entre varios conceptos, indicando las características que permiten dicha relación. En la retórica, una analogía es una comparación textual que resalta alguna de las similitudes semánticas entre los conceptos protagonistas de dicha comparación. Por ejemplo: “Sus ojos son azules como el mar”.

En nuestro trabajo, las figuras retóricas van a permitir comparar el concepto complejo introducido por el usuario con otros conceptos más sencillos.

---

<sup>17</sup> <https://dle.rae.es/?id=WISC3uX>

## 2.4. Servicios Web

Para definir el concepto de servicio web de la forma más simple posible, se podría decir que es una tecnología que utiliza un conjunto de protocolos para intercambiar datos entre aplicaciones, sin importar el lenguaje de programación en el cual estén programadas o ejecutadas en cualquier tipo de plataforma<sup>18</sup>. Según el W3C (*World Wide Web Consortium*)<sup>19</sup>, “*un servicio web es un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable*”.

Las principales características de un servicio web son (Torres, 2017):

- Es accesible a través de la Web. Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda ser accesible por cualquier cliente que quiera utilizar el servicio.
- Contiene una descripción de sí mismo. De esta forma, una aplicación web podrá saber cual es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
- Debe ser localizado. Debe tener algún mecanismo que permita encontrarlo. De esta forma tendremos la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente.

### 2.4.1. Tipos de Servicios Web

Los servicios web pueden definirse tanto a nivel conceptual como a nivel técnico. A nivel técnico se pueden diferenciar dos tipos de servicios web (Torres, 2017):

- Servicios web SOAP (*Simple Object Access Protocol*): SOAP es un protocolo basado en XML para el intercambio de información entre ordenadores. Normalmente utilizaremos SOAP para conectarnos a un servicio e invocar métodos remotos<sup>20</sup>. Los mensajes SOAP tienen el formato representado en el Listado 2.6, donde podemos ver un ejemplo para reservar un vuelo y está formado por los siguientes campos:
  - <Envelope>: elemento raíz de cada mensaje SOAP. Contiene dos elementos:

---

<sup>18</sup><http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html>

<sup>19</sup><https://www.w3.org/>

<sup>20</sup><https://www.ibm.com/support/knowledgecenter/es>

- <Header>: es un elemento opcional que se utiliza para indicar información acerca de los mensajes SOAP. En el ejemplo del Listado 2.6 dentro del campo Header estarían los campos de reservas y pasajeros.
- <Body>: es un elemento obligatorio que contiene información dirigida al destinatario del mensaje. En el ejemplo del Listado 2.6 se puede ver los campos asociados a un itinerario, teniendo este el lugar de partida, de llegada, la fecha de llegada y la preferencia de asiento.
- <Fault>: es un elemento opcional para notificar errores. En el Listado 2.6 podemos ver que no se encuentra presente, pero en caso de ser utilizado deberá aparecer dentro del elemento *Body* y no puede aparecer más de una vez.

Listado 2.6: Estructura de un mensaje SOAP

```
<?xml version = '1.0' Encoding='UTF-8' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/
      reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role
      /next">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:
        reference>
      <m:dateAndTime>2007-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/
      next">
      <n:name>Fred Bloggs</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary xmlns:p="http://travelcompany.example.org/reservation
      /travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2007-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2007-12-20</p:departureDate>
        <p:departureTime>mid-morning</p:departureTime>
        <p:seatPreference></p:seatPreference>
      </p:return>
    </p:itinerary>
  </env:Body>
</env:Envelope>
```

- Servicios Web RESTful: RESTful es un protocolo que suele integrar mejor con HTTP que los servicios basados en SOAP, ya que no requieren mensajes XML. Cada petición del cliente debe contener toda la información necesaria para entender la petición, y no puede aprovecharse de ningún contexto almacenado en el servidor.

### 2.4.2. Arquitectura Servicios Web

Hay que distinguir tres partes fundamentales en los servicios web (Torres, 2017):

- El proveedor: es la aplicación que implementa el servicio y lo hace accesible desde Internet.
- El solicitante: cualquier cliente que necesite utilizar el servicio web.
- El publicador: se refiere al repositorio centralizado en el que se encuentra la información de la funcionalidad disponible y como se utiliza.

Por otro lado, los servicios web se componen de varias capas<sup>21</sup>:

- Descubrimiento del Servicio: responsable de centralizar los servicios web en un directorio común, de esta forma es más sencillo buscar y publicar.
- Descripción del Servicio: como ya hemos comentado con anterioridad, los servicios web se pueden definir a sí mismos, por lo que una vez que los localicemos el Service Description nos dará la información para saber qué operaciones soporta y cómo activarlo.
- Invocación del Servicio: invocar a un Servicio Web implica pasar mensajes entre el cliente y el servidor. Por ejemplo, si utilizamos SOAP (Simple Object Access Protocol), el Service Invocation especifica cómo deberíamos formatear los mensajes request para el servidor, y cómo el servidor debería formatear sus mensajes de respuesta.
- Transporte: todos los mensajes han de ser transmitidos de alguna forma entre el servidor y el cliente. El protocolo elegido para ello es HTTP.

### 2.4.3. Ventajas de los Servicios Web

Las principales ventajas del uso de los servicios web son las siguientes (Vega Lebrún, 2005):

- Permiten la integración “justo-a-tiempo”: esto significa que los solicitantes, los proveedores y los agentes actúan en conjunto para crear sistemas que son auto-configurables, adaptativos y robustos.
- Reducen la complejidad por medio del encapsulamiento: un solicitante de servicio no sabe cómo fue implementado el servicio por parte del proveedor, y éste, a su vez, no sabe cómo utiliza el cliente el servicio. Estos detalles se encapsulan en los solicitantes y proveedores. El encapsulamiento es crucial para reducir la complejidad.

---

<sup>21</sup><https://diego.com.es/introduccion-a-los-web-services>

- Promueven la interoperabilidad: la interacción entre un proveedor y un solicitante de servicio está diseñada para que sea completamente independiente de la plataforma y el lenguaje.
- Abren la puerta a nuevas oportunidades de negocio: los servicios web facilitan la interacción con socios de negocios, al poder compartir servicios internos con un alto grado de integración.
- Disminuyen el tiempo de desarrollo de las aplicaciones: gracias a la filosofía de orientación a objetos que utilizan, el desarrollo se convierte más bien en una labor de composición.
- Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

#### 2.4.4. Desventajas de los Servicios Web

El uso de servicios web también tiene algunas desventajas<sup>22</sup>:

- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear.
- Existe poca información de servicios web para algunos lenguajes de programación.
- Dependen de la disponibilidad de servidores y comunicaciones.

---

<sup>22</sup><http://fabioalfarocc.blogspot.com/2012/08/ventajas-y-desventajas-del-soap.html>

# Capítulo 3

## Herramientas Utilizadas

En este capítulo se van a explicar las herramientas utilizadas para el desarrollo de este trabajo. En el apartado 3.1 se explicará Django que es el *framework* utilizado para el desarrollo de la aplicación y en el apartado 3.2 se explicará SpaCy, que es la herramienta que se ha utilizado para la clasificación semántica de las palabras fáciles. Las herramientas y recursos que se han usado para la gestión del proyecto se explicarán en un capítulo aparte.

### 3.1. Django

Para construir un servicio web se necesita una manera de gestionar los elementos propios de estos servicios, como pueden ser el procesamiento de formularios y el mapeo de urls, para satisfacer estas necesidades, se requieren *frameworks* web que son estructuras que contienen los componentes necesarios para el desarrollo de aplicaciones web<sup>1</sup>. Django es un *framework* de alto nivel que permite el desarrollo rápido de sitios web seguros y mantenibles y se basa en el patrón MVC<sup>2</sup>. Fue desarrollado entre los años 2003 y 2005 por un grupo de programadores que se encargaban de crear y mantener sitios web de periódicos. Es gratuito y de código abierto y dispone de una gran documentación actualizada así como muchas opciones de soporte gratuito y de pago.

Algunas de las razones por las que se ha elegido este *framework* han sido las siguientes<sup>3</sup>:

- Seguridad: Implementa por defecto algunas medidas de seguridad para evitar SQL Injection(adición de consultas SQL malignas que puedan

<sup>1</sup><https://tutorial.djangogirls.org/es/django/>

<sup>2</sup><https://docs.djangoproject.com/en/2.0/>

<sup>3</sup><https://openwebinars.net/blog/que-es-django-y-por-que-usarlo/>

alterar la base de datos) o Clickjacking(Que el usuario haga click en un enlace oculto, para obtener información del mismo sin su permiso o incluso tomar el control de su ordenador) por JavaScript.

- Escalabilidad: Se puede pasar de una aplicación sencilla a otra más compleja rápidamente, ya que es muy fácil añadir nuevos módulos al *framework*.
- Fácil acceso a bases de datos: Mediante ORM(Object Relational Mapper), que es una biblioteca para el acceso de datos, generando clases a partir de las tablas de la base de datos para poder realizar las consultas. Django tiene su propio ORM, con el que se pueden hacer consultas de manera muy intuitiva.
- Además, es muy popular por lo que para resolver cualquier problema que surja, como se ha explicado anteriormente, hay mucha documentación disponible y muchos hilos en foros de programación en donde encontrar posibles soluciones.

### 3.2. SpaCy

Cuando se obtuvieron las palabras fáciles de la RAE solo se necesitaban adverbios, nombres, verbos y adjetivos, por lo que se requería un análisis semántico de cada una de ellas para obtener solo las palabras que pertenecían a uno de estos grupos. Para ello se utilizó SpaCy. SpaCy<sup>4</sup> es una biblioteca de código abierto para el Procesamiento del Lenguaje Natural en Python. Soporta más de 34 idiomas, entre ellos el español.

El analizador semántico de SpaCy, según un estudio realizado en 2015 por la universidad Emory<sup>5</sup>, es el más rápido y según Medium Corporation tiene un índice de acierto mucho mayor que el analizador sintáctico de NLTK<sup>6</sup>. Para utilizar en analizador, hay que importar la biblioteca de idioma correspondiente (en este caso español: “es\_core\_news\_sm”) y pasar como parámetro las palabras que se deseen clasificar. El resultado para cada palabra introducida en el analizador, serán una serie de etiquetas, de todas ellas, la utilizada es la etiqueta “pos”, que contiene el grupo semántico de la palabra analizada, como se puede ver en la Figura 3.1, se ha introducido la frase: “el coche es rojo” y se han obtenido los siguientes resultados:

- el: ha obtenido la etiqueta DET, haciendo referencia a que es un determinante.
- coche: ha obtenido la etiqueta NOUN, que significa nombre en inglés.

---

<sup>4</sup><https://spacy.io/>

<sup>5</sup><https://www.aclweb.org/anthology/P15-1038>

<sup>6</sup><https://medium.com/@pemagrg/private-nltk-vs-spacy-3926b3674ee4>

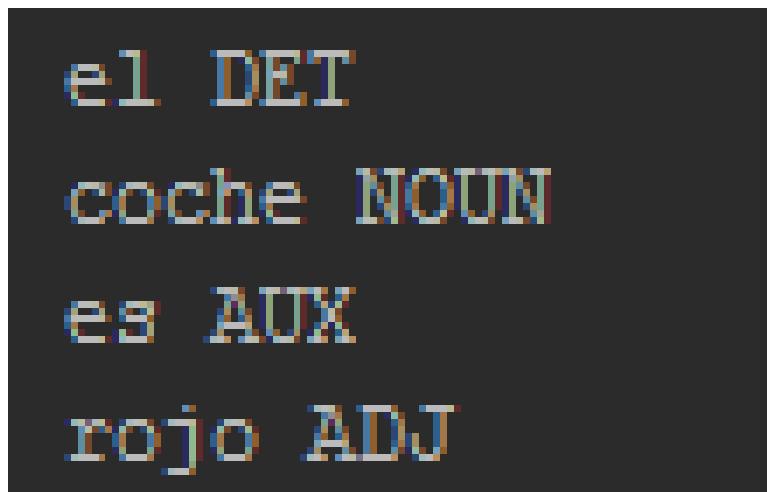


Figura 3.1: Ejemplo de clasificación de palabras

- es: ha obtenido la etiqueta AUX, haciendo referencia a que es un verbo auxiliar.
- rojo: ha obtenido la etiqueta ADJ, haciendo referencia a que es un adjetivo.

Se ha elegido ya que ha sido el que mejor resultado ha dado. Inicialmente se utilizó el POS-tagger (clasificador sintáctico de palabras) de NLTK, pero su índice de acierto no era muy bueno, por lo que se descartó su uso. A continuación se probó con SpaCy y ,además de ser más rápido, su índice de acierto ha sido prácticamente del 100 % ,por lo que fue lo finalmente utilizado.



# Aprende Fácil

## 4.1. Diseño de la Aplicación

Como ya se ha explicado en capítulos anteriores, esta aplicación está creada por y para personas que tienen alguna discapacidad cognitiva, por lo que el diseño de la interfaz debe ser sencillo y su uso no debe llevar a confusión ni debe hacer sentir al usuario frustrado al usarla. En definitiva, debe cumplir con las Ocho Reglas de Oro del diseño de interfaces, pero simplificándola aún más de lo que podría ser un diseño para otro tipo de aplicación destinada para otro tipo de usuario. Las Ocho Reglas de Oro del diseño de interfaces consisten en que el diseño tenga (Shneiderman y Plaisant, 2006):

- **Consistencia:** La funcionalidad debe ser similar a otras aplicaciones que el usuario está acostumbrado a utilizar. En cuanto a la interfaz debe tener los mismos colores, iconos, formas, botones, mensajes de aviso... Por ejemplo, si el usuario está acostumbrado a que el botón de eliminar o cancelar sea rojo, no debemos añadirle uno de color verde.
- **Usabilidad Universal:** Debemos tener en cuenta las necesidades de los distintos tipos de usuario, como por ejemplo, atajos de teclado para un usuario experto o filtros de color para usuarios con deficiencias visuales.
- **Retroalimentación activa:** Por cada acción debe existir una retroalimentación legible y razonable por parte de la aplicación. Por ejemplo, si el usuario quiere guardar los datos obtenidos de la búsqueda, la aplicación debe informarle de si han sido guardados o no.
- **Diálogos para conducir la finalización:** El usuario debe saber en qué paso se encuentra en cada momento. Por ejemplo, en un proceso de compra que conlleva varios pasos hasta la finalización de la misma, se le debe informar donde se encuentra y cuánto le queda para terminar.

- Prevención de errores: La interfaz debe ayudar al usuario a no cometer errores serios, y en caso de cometerlos se le debe dar una solución lo más clara y sencilla posible. Por ejemplo, deshabilitando opciones o indicando en un formulario el campo en el cual se ha producido el error sin perder la información ya escrita.
- Deshacer acciones fácilmente: Se debe dar al usuario la capacidad de poder deshacer o revertir acciones de una manera sencilla.
- Sensación de control: Hay que dar al usuario la sensación de que tiene en todo momento el control de la aplicación, añadiendo contenidos fáciles de encontrar y de esta forma no causarle ansiedad o frustración por utilizar nuestra aplicación.
- Reducir la carga de memoria a corto plazo: La interfaz debe ser lo más sencilla posible y con una jerarquía de información evidente, es decir, hay que minimizar la cantidad de elementos a memorizar por el usuario.

Teniendo en cuenta estas ocho reglas, la creación del diseño de la Interfaz se ha realizado en dos iteraciones distintas. Una primera iteración competitiva entre los integrantes del grupo y una segunda iteración con expertos del Colegio Estudio3 Afanias.

#### 4.1.1. Primera Iteración: Iteración Competitiva

En esta primera iteración cada integrante del grupo ha realizado su propio diseño de la aplicación. En la realización de estos diseños, los integrantes no podían hablar entre ellos ni comentar las diferentes ideas que tenían para la implementación. De esta forma se consigue que los diseños sean totalmente dispares y que las ideas de uno no provoquen la modificación del diseño del otro y que surjan ideas distintas. Se realizaron cuatro prototipos distintos ya que no teníamos claro si los usuarios finales iban a preferir que se mostrara un solo significado o todos, si mostrar la definición y el ejemplo les iba a ayudar o no y si añadir los pictogramas les ayudaría o no. Los prototipos fueron los siguientes:

- Prototipo 1: Se muestra un único significado, siendo este el más común e incluyendo las comparaciones.
- Prototipo 2: Se muestra el significado más común, junto con una definición tradicional del concepto e incluyendo las comparaciones.
- Prototipo 3: Se muestran todos los significados de la palabra buscada e incluyendo las comparaciones.

- Prototipo 4: Se muestran todos los significados de la palabra y se añaden pictogramas<sup>1</sup> a las palabras, haciendo así que la comprensión del concepto sea mucho más sencilla. Tanto en el prototipo diseñado por Pablo como en el de Irene se han utilizado los pictogramas de ARASAAC<sup>2</sup>.

En las Figuras 4.1, 4.2, 4.3 y 4.4 se muestran los prototipos creados por Pablo y en las Figuras 4.5, 4.6, 4.7 y 4.8 los prototipos creados por Irene.

Una vez que los prototipos estaban terminados, los juntamos para hacer una puesta en común y analizar los prototipos creados. Por lo general, los prototipos de los dos integrantes del grupo eran bastante similares, las principales diferencias eran:

- Ambos prototipos integran los mismos elementos: un campo de texto para introducir la palabra, un botón de búsqueda y los resultados se muestran en forma de lista, agrupando los distintos resultados en rectángulos, que a partir de ahora denominaremos fichas. Pablo ha optado por formas rectangulares, tanto para las fichas como para el campo de texto, mientras que Irene utiliza formas redondeadas en todos los elementos. Por otro lado, Pablo implementó un diseño orientado a niños por lo que para el color utilizó azules muy suaves y fondos juveniles con lápices de colores, gomas de borrar, etc. Irene utilizó un color mostaza, siendo un diseño más simple pero intentando abarcar a un usuario de cualquier edad. Por último, los resultados del prototipo de Irene, se muestran en color mostaza indicando de esta forma que se pueden pinchar en ellos, y se realizará la búsqueda del concepto pulsado.
- El orden a la hora de mostrar los resultados es distinto. En los prototipos de Pablo se muestran primero las metáforas (*Un vehículo es una máquina* o *Un vehículo es un transporte*), después los símiles (*Un vehículo es como un taxi*) y finalmente las analogías (*Un vehículo es rápido como un caballo*). En los prototipos de Irene se muestra únicamente una analogía, una metáfora y un símil para cada significado de la palabra buscada.
- En el prototipo 3 de Pablo incluye justo debajo de los resultados mostrados un ejemplo siempre visible (“Él necesita un coche para ir a trabajar”) para facilitar al usuario la comprensión del término buscado.
- En el prototipo 3 de Irene incluye justo debajo de los resultados mostrados una definición dentro de un botón (“Vehículo motorizado de cuatro

<sup>1</sup>Se entiende como pictograma un dibujo, imagen o figura que representa el significado de una palabra.

<sup>2</sup><http://www.arasaac.org/>

ruedas por lo general impulsado por un motor de combustión interna” dando a elegir al usuario la decisión de poder verla o no.

- Pablo decidió añadir el título “Un X es ...” antes de la lista de los resultados, englobando de esta forma los conceptos que tienen el mismo significado. Si existen varias acepciones del concepto, añade el título “O también puede ser...” en los siguientes, dejando claramente divididos los distintos significados que pudieran existir para el concepto buscado. En cambio Irene, añade un único título al principio (“Resultados para la palabra X”) donde queda claro que los resultados que se obtienen son para dicho concepto y además añade delante de cada metáfora: “Un X es...” y delante de cada símil y analogía: “Un X es como...”.
- Para la numeración de los resultados obtenidos, Pablo incluye dentro de cada ficha un listado numérico e Irene únicamente añade un número a la ficha.
- En el prototipo donde se incluyen también los pictogramas, Pablo añade un pictograma que hace referencia al concepto buscado según el contexto en que se utilice. Por ejemplo, un vehículo puede hacer referencia a un coche o puede ser un medio para llegar o lograr un fin, por lo que el añade el pictograma en función de su significado, e Irene además de incluir el mismo pictograma que Pablo, añade pictogramas a las palabras usadas en las comparaciones. Por ejemplo, en la frase “Un portero es un vigilante” Irene añadió el pictograma de vigilante.

Una vez analizados los prototipos de ambos integrantes nos reunimos con los directores del trabajo y se decidió crear un prototipo con las siguientes características:

- Para el diseño de la interfaz se eligió el de Irene por tener una interfaz más atractiva.
- Se eligió la palabra Portero para utilizarla como ejemplo en todos los prototipos.
- El orden de los resultados se modificó, haciendo que primero aparezcan las metáforas (“Un portero es un vigilante”), después los símiles (“Un portero es como un guardia”) y por último las analogías (“Un portero es tan ágil como un pájaro”).
- Se eliminó la palabra *tan* en la plantilla usada para las analogías.
- Se creó un nombre para la aplicación que estuviese en español: “Aprende Fácil”.

- En vez de mostrar un único símil, una metáfora y una analogía. Se deben mostrar todos los resultados obtenidos para dicho concepto.
- Se decidió que si la palabra solo tenía un significado se eliminaba el número de la ficha.
- Se añadió un pictograma a la característica que relaciona el concepto con el resultado en las analogías. Por ejemplo, en la frase *Un portero es tan fuerte como un gorila* se añadió el pictograma correspondiente a fuerte.
- Dejar el ejemplo del prototipo de Pablo junto con la definición del prototipo de Irene.
- Crear un nuevo prototipo donde la definición y el ejemplo se muestren a primera vista, y el usuario no tenga que estar pinchando en ningún botón para que los expertos nos indiquen cual es la mejor opción.
- Crear otro prototipo donde la definición y el ejemplo estén separados. Así el usuario puede elegir lo que quiere ver, si solo una cosa o ambas.

Con todas estas decisiones, se crearon los prototipos que se muestran en las Figuras 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 y que fueron los que se usaron para en la siguiente iteración donde nos reunimos con los expertos para que nos dieran su opinión sobre la aplicación, y nos ayudarán a decidir sobre los siguientes aspectos:

- ¿Se debe mostrar el significado más común o todos los significados de la palabra buscada?
- ¿Se debe añadir la definición tradicional y el ejemplo junto con las figuras retóricas o no? Y en caso de añadirlos, ¿Se deben mostrar juntos en un mismo botón, en distintos botones o sin botón y que aparezcan debajo de los resultados?
- ¿Son útiles los pictogramas? En caso de serlo, ¿solamente el pictograma de la palabra buscada o también añadimos los pictogramas de las palabras que se usan en las figuras retóricas?
- ¿La forma de mostrar los resultados es correcta, o es mejor otra disposición?

#### 4.1.2. Segunda Iteración: Evaluación con Expertos

El día 26 de Marzo de 2019 a las 09:00h nos reunimos con la directora y los profesores del colegio Estudio3 Afanias<sup>3</sup> situado en la Comunidad de

---

<sup>3</sup><https://afanias.org/que-hacemos/educacion/colegio-estudio-3/>

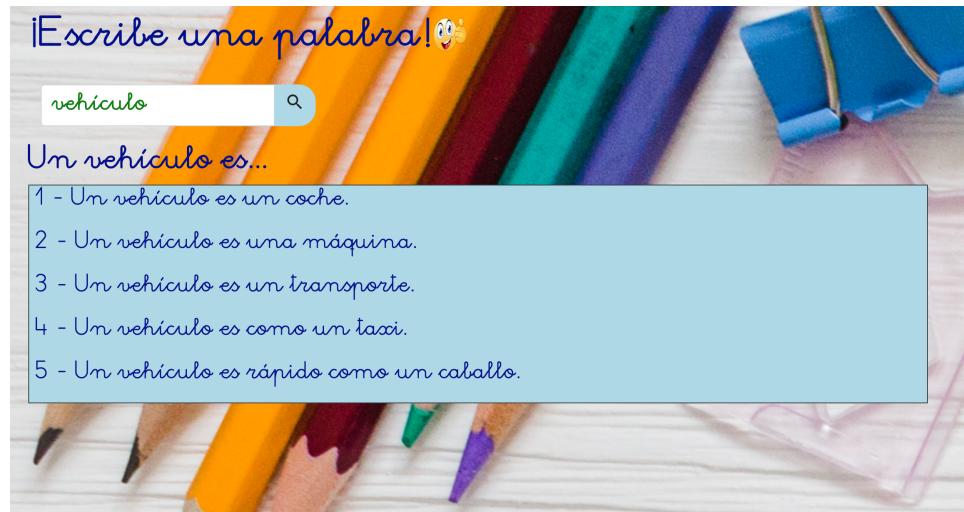


Figura 4.1: Prototipo de Pablo mostrando resultado más común para la palabra vehículo

Madrid. Este colegio atiende a niños y jóvenes con discapacidad intelectual entre los 3 y 21 años de edad.

Lo primero que hicimos fue presentarles la aplicación y mostrarles los prototipos creados al final de la iteración anterior (Figuras 4.9, 4.10, 4.11, 4.12, 4.13, 4.14). Una vez expuestos los distintos prototipos, nos dieron la enhorabuena y nos hicieron participes de la gran ayuda que supondría esta herramienta. A continuación nos dieron su opinión sobre distintos aspectos que se podrían mejorar:

- Modificar el color amarillo-mostaza de los textos por un color más oscuro que contraste más con el fondo blanco. El color actual es morado oscuro para la barra que contiene el título de la aplicación y un violeta para resaltar los resultados obtenidos, así como para el botón de Definición y Ejemplo.
- El tipo de letra debe ser Arial o Script, ya que son las letras con las que los alumnos están familiarizados y las que mejor entienden.
- Añadir un reproductor que lea la frase, para hacer la aplicación accesible a aquellas personas que no puedan leer.
- Incluir la opción de poder ver un video para complementar la información devuelta por la aplicación.
- Los pictogramas deben situarse debajo de la palabra y no al lado, ya que los expertos comentaron que esto puede llevar a confusión a los alumnos pensando que sería otra palabra más para leer.

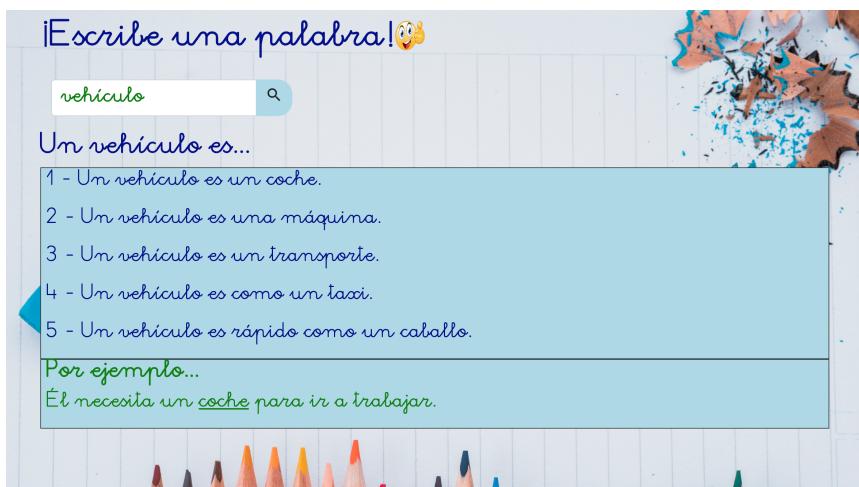


Figura 4.2: Prototipo de Pablo mostrando resultado más común para la palabra vehículo y un ejemplo

- Introducción de distintas personalizaciones:
  - Búsqueda en tres niveles: sencillo, medio y amplio. El nivel sencillo sería realizando la búsqueda de las palabras fáciles en las 1.000 palabras más usadas de la RAE, el medio en las 5.000 palabras más usadas de la RAE y el amplio en las 10.000 palabras más usadas de la RAE.
  - Añadir una opción que permita poner todos los textos en mayúsculas, haciendo la aplicación más accesible para aquellos alumnos que no entienden los textos en minúsculas.
  - Dejar que el usuario configure si quiere que aparezca la definición y el ejemplo o no.
  - Dejar que el usuario configure si aparecen los pictogramas o no.

Teniendo en cuenta las observaciones de los expertos se creó el diseño final de la aplicación.

## 4.2. Arquitectura

## 4.3. Base de datos

## 4.4. Back end

Como se ha comentado en anteriores capítulos existen varias aplicaciones web que son redes semánticas y facilitan sinónimos, términos relacionados,

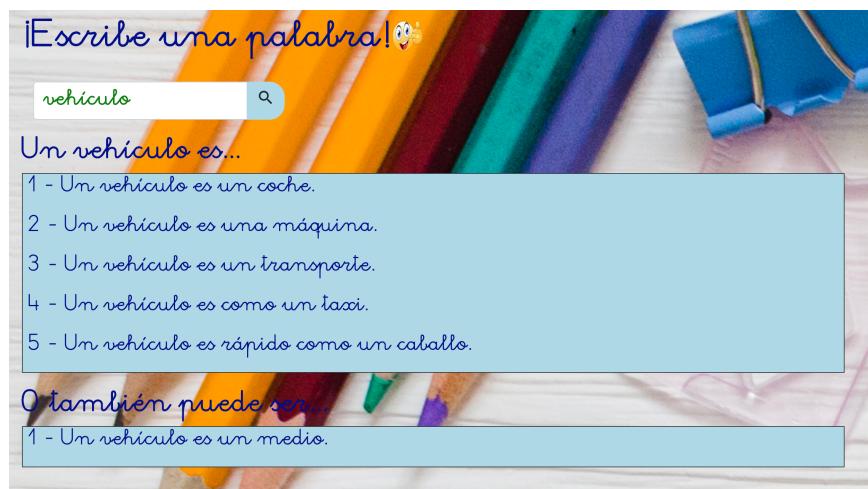


Figura 4.3: Prototipo de Pablo mostrando todos los resultados para la palabra vehículo

metáforas, hiperónimos, etc... de un concepto dado por el usuario. Para este trabajo, algunas de ellas son de gran utilidad ya que de esta forma podemos obtener las palabras fáciles para un concepto más complicado, pero hay que saber exactamente cuál es la que mejor conviene y la que mejores resultados ofrece. Hay que corroborar que los conceptos devueltos son correctos y que disponen de un significado claro y parecido respecto al concepto buscado. Para ello se han implementado dos servicios web, uno utilizando la aplicación de ConceptNet y el otro utilizando WordNet, en la sección 5.1 se explicará el diseño de la prueba que se ha realizado para ver que red semántica se utilizará finalmente en el proyecto, en la sección 5.2 se analizarán los resultados cuantitativos de la prueba realizada, en el punto 5.3, se analizarán los resultados obtenidos a nivel cualitativo y en el 5.4 se describirán las conclusiones finales de la prueba. Por otra parte, en los apartados posteriores, se detallarán los distintos servicios web utilizados para este proyecto, en la sección 5.5 se hablará de la base de datos utilizada y de su estructura, en los apartados 5.6, 5.7 y 5.8 se describirán los servicios web que se utilizan para la obtención de sinónimos, hipónimos e hiperónimos fáciles respectivamente. Por último, se hablará en los puntos 5.9 y 5.10 sobre los servicios web utilizados para crear las metáforas y los símiles que finalmente leerá el usuario.



Figura 4.4: Prototipo de Pablo mostrando todos los resultados para la palabra vehículo junto con pictogramas

#### 4.4.1. Red Semántica Utilizada

##### 4.4.1.1. Diseño de la evaluación

Para decidir que red semántica se iba a usar se decidió hacer una prueba con palabras que fuesen lo más heterogéneas posible, para ello se eligieron una serie de artículos periodísticos de distintos temas: medioambiental, tecnológico, deportivo y político. Estos artículos se filtraron para utilizar únicamente los verbos, artículos, sustantivos y adverbios. Para cada una de estas palabras se obtuvieron en WordNet y ConceptNet sus sinónimos y términos relacionados (en WordNet se entiende como término relacionado a los hiperónimos e hipónimos) y se analizaron, tanto la cantidad de términos relacionados y sinónimos que generaban cada una de las redes semánticas que coincidían con alguna de las palabras fáciles de la RAE (prueba cuantitativa) probando en cada una de las tres listas(1.000 palabras fáciles, 5.000 palabras fáciles y 10.000 palabras fáciles), como la calidad de los mismos, es decir, si las palabras que generaban tenían alguna relación aceptable con la palabra origen(prueba cualitativa).

##### 4.4.1.2. Resultados cuantitativos

Como se puede observar en la Tabla 4.15 en términos generales, WordNet ofrece mejores resultados que ConceptNet ya que el porcentaje de palabras sin términos relacionados, sinónimos o ambas, es mayor en esta última, siendo especialmente notorio en el caso de los sinónimos con más de 60 puntos de diferencia en todos los casos en favor de WordNet. Los únicos casos en los que ConceptNet supera a WordNet son con las listas de palabras fáciles de 5.000 y 10.000 palabras, ConceptNet deja menos palabras sin coincidencia



Figura 4.5: Prototipo de Irene mostrando resultado más común para la palabra vehículo

que WordNet, sin embargo, encuentra mas términos relacionados y muchos más sinónimos que ConceptNet.

#### 4.4.1.3. Análisis cualitativo

Pendiente de terminar...

#### 4.4.1.4. Conclusiones

Pendiente de terminar...

### 4.5. Servidor de Base de Datos

Para este proyecto, se ha utilizado una base de datos para la persistencia de los datos. El sistema encargado de la gestión de la base de datos corresponde con MariaDB y esta constituida por varias tablas:

- Se ha hecho uso de MCR (*Multilingual Central Repository*), utilizando

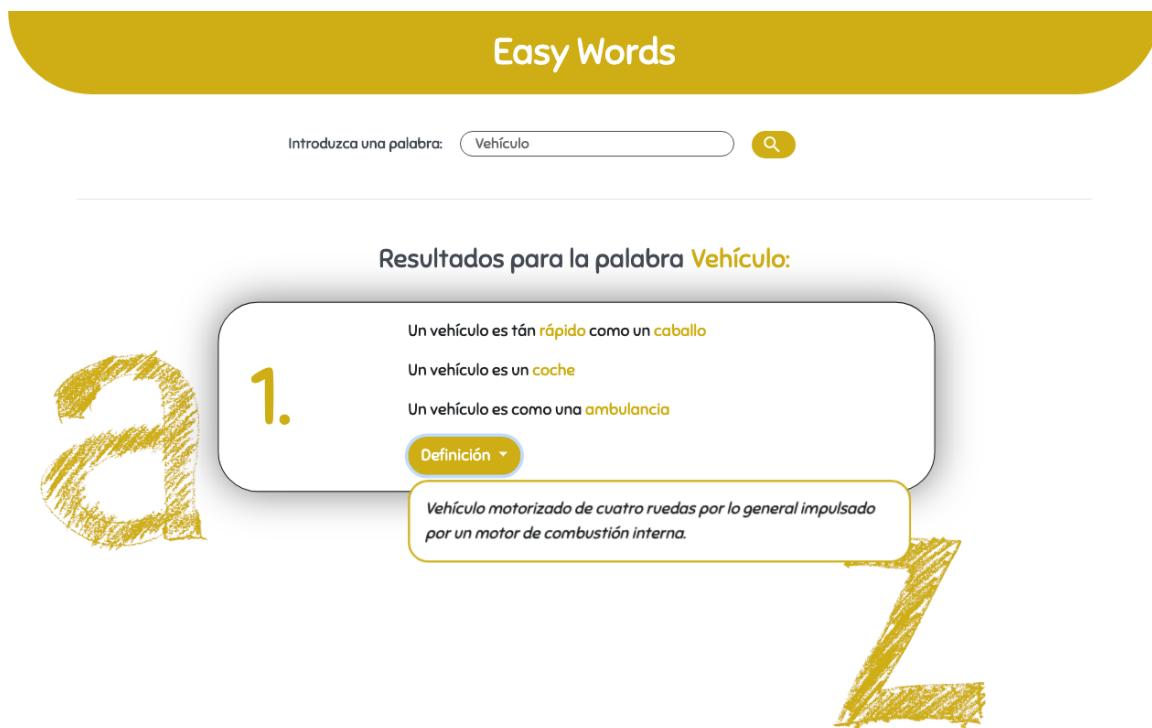


Figura 4.6: Prototipo de Irene mostrando resultado más común para la palabra vehículo y una definición

la versión 3.0. MCR es una base de datos de código abierto que integra distintas versiones de WordNet para seis lenguajes diferentes: Inglés, Español, Catalán, Vasco, Gallego y Portugués. Se ha utilizado principalmente la tabla llamada **wei\_spa-30\_variant** y **wei\_spa-30\_relation**. De la tabla *Variant* las columnas con las que hemos trabajado han sido:

- *Word*: Contiene la palabra. De esta forma se pueden realizar búsquedas cuando el usuario introduce el concepto para de esta forma saber si dicha palabra se encuentra en la base de datos.
- *Offset*: Es el identificador de la palabra, aunque una misma palabra puede tener distintos *offsets*. Con dicha columna se pueden obtener los sinónimos, así como el identificador para posteriormente obtener los hipónimos e hiperónimos.
- Tres tablas llamadas **1000\_palabras\_faciles**, **5000\_palabras\_faciles** y **10000\_palabras\_faciles** que guardan tanto las 1.000, 5.000 y 10.000 palabras más usadas de la RAE. Las tres se componen únicamente de una columna llamada *word* y que almacena la palabra.

The image shows a user interface for a service called "Easy Words". At the top, there is a yellow header bar with the title "Easy Words". Below it, a search bar contains the word "Portero". A magnifying glass icon is used to submit the search. The main content area displays results for the word "Portero". It includes two numbered sections: section 1 with three analogies involving strength and vigilance, and section 2 with three analogies involving agility and sports. Large, stylized letters "a" and "z" are positioned on either side of the results.

Introduzca una palabra: Portero

Resultados para la palabra Portero:

**1.**

- Un portero es tan fuerte como un gorila
- Un portero es un vigilante
- Un portero es como un guardia

**2.**

- Un portero es tan ágil como un pájaro
- Un portero es un jugador
- Un portero es como un futbolista

Figura 4.7: Prototipo de Irene mostrando todos los resultados para la palabra portero

- Una tabla llamada **datos\_picto**, que está formada por las siguientes columnas:
  - Palabra: Contiene la palabra en cuestión.
  - Offset31: Es el identificador del *synset* en la versión 3.1 de WordNet.
  - Offset30: Es el identificador del *synset* en la versión 3.0 de WordNet.
  - id\_picto: Es el identificador del pictograma.
- Una última tabla llamada **pictogramas** que almacena los pictogramas de ARASAAC y que contiene dos columnas:
  - id\_picto: Es el identificador del pictograma.
  - imagen: Contiene el pictograma.

#### 4.6. Servicio Web para obtener sinónimos fáciles

La implementación de dicho servicio web se basa en que introduciendo una palabra y un nivel de búsqueda, este devuelve todos los sinónimos fáciles

correspondientes en formato JSON. Para ello se realiza la siguiente petición GET:

`http://127.0.0.1:8000/easySynonym/json/word=palabra&level=nivel`

Donde *palabra* es el concepto a buscar y *nivel* es el grado de búsqueda, el cual puede tomar los siguientes valores:

- Nivel 1 (Nivel sencillo): Se realizará la búsqueda en las 1.000 palabras más usadas de la RAE.
- Nivel 2 (Nivel medio): Se realizará la búsqueda en las 5.000 palabras más usadas de la RAE.
- Nivel 3 (Nivel avanzado): Se realizará la búsqueda en las 10.000 palabras más usadas de la RAE.

Una vez introducida la palabra y el nivel, se realiza una consulta a la base de datos de MCR 3.0 a través de una *queryset* para obtener todos los *offsets* cuya palabra sea igual que la introducida. Por cada *offset* volveremos a buscar en la tabla *Variant* de la base de datos de MCR 3.0 para obtener las palabras que comparten dicho identificador y posteriormente, mediante un cursor se realizará una búsqueda en una de las tres tablas de la RAE (en función del nivel introducido) y buscará si alguna de estas palabras se encuentra en dicha tabla. Si el resultado es positivo se añadirá al JSON.

Por ejemplo, si se realizará una búsqueda para la palabra inmueble con un nivel de búsqueda 2, la petición GET sería de la siguiente manera:

`http://127.0.0.1:8000/easySynonym/json/word=inmueble&level=2`

Y en la Figura 4.16 se puede ver el resultado obtenido en formato JSON. En primer lugar aparece la palabra buscada, en este caso inmueble y a continuación un objeto que incluye tanto el *offset* como la lista de sinónimos fáciles .

## 4.7. Servicio Web para obtener hipónimos fáciles

La implementación de dicho servicio web se basa en que introduciendo una palabra y un nivel de búsqueda, este devuelve todos los hipónimos fáciles correspondientes en formato JSON. Para ello se realiza la siguiente petición GET:

`http://127.0.0.1:8000/easyHyponym/json/word=palabra&level=nivel`

Donde *palabra* es el concepto a buscar y *nivel* es el grado de búsqueda, igual que en el caso explicado anteriormente para buscar sinónimos fáciles.

Una vez introducida la palabra y el nivel, se realiza una consulta a la base de datos de MCR 3.0 a través de una *queryset* para obtener todos los *offsets* cuya palabra sea igual que la introducida. Por cada *offset* volveremos a buscar en la tabla *Relation* de la base de datos de MCR 3.0 para obtener los

offsets que se encuentran en la columna *targetSynset*. Después, con cada *offset* obtenido de esta *queryset*, se realizará la búsqueda en la tabla *Variant* para obtener las palabras cuyo identificador sea igual que el offset de *targetSynset*. Mediante un cursor se realizará una búsqueda en una de las tres tablas de la RAE (en función del nivel introducido) y buscará si alguna de estas palabras se encuentra en dicha tabla. Si el resultado es positivo se añadirá al JSON.

Por ejemplo, si se realizará una búsqueda para la palabra inmueble con un nivel de búsqueda 2, la petición GET sería de la siguiente manera:

```
http://127.0.0.1:8000/easyHyponym/json/word=inmueble&level=2
```

Y en la Figura 4.17 se puede ver el resultado obtenido en formato JSON. En primer lugar aparece la palabra buscada, en este caso inmueble y a continuación un objeto que incluye el *offset* de hipónimo fácil , la lista de hipónimos fáciles así como el *offsetFather*, este identificador corresponde al *offset* de uno de los *synsets* de inmueble. De esta forma, si una palabra buscada dispone de varios *synsets*, se podrá mostrar sus correspondientes sinónimos e hipónimos.

## 4.8. Servicio Web para obtener hiperónimos fáciles

La implementación de dicho servicio web se basa en que introduciendo una palabra y un nivel de búsqueda, este devuelve todos los hiperónimos fáciles correspondientes en formato JSON. Para ello se realiza la siguiente petición GET:

```
http://127.0.0.1:8000/easyHyperonym/json/word=palabra&level=nivel
```

Donde *palabra* es el concepto a buscar y *nivel* es el grado de búsqueda, igual que en los casos explicados anteriormente para buscar sinónimos e hipónimos fáciles.

Una vez introducida la palabra y el nivel, se realiza una consulta a la base de datos de MCR 3.0 a través de una *queryset* para obtener todos los *offsets* cuya palabra sea igual que la introducida. Por cada *offset* volveremos a buscar en la tabla *Relation* de la base de datos de MCR 3.0 para obtener los offsets que se encuentran esta vez y con diferencia de la búsqueda de hipónimos fáciles, en la columna *sourceSynset*. Después, con cada *offset* obtenido de esta *queryset*, se realizará la búsqueda en la tabla *Variant* para obtener las palabras cuyo identificador sea igual que el offset de *sourceSynset*. Mediante un cursor se realizará una búsqueda en una de las tres tablas de la RAE (en función del nivel introducido) y buscará si alguna de estas palabras se encuentra en dicha tabla. Si el resultado es positivo se añadirá al JSON.

Por ejemplo, si se realizará una búsqueda para la palabra inmueble con un nivel de búsqueda 2, la petición GET sería de la siguiente manera:

```
http://127.0.0.1:8000/easyHyperonym/json/word=inmueble&level=2
```

En la Figura 4.18 se puede ver el resultado obtenido en formato JSON.

En primer lugar aparece la palabra buscada, en este caso inmueble y a continuación un objeto que incluye el *offset* de hiperónimo fácil, la lista de hiperónimos fáciles así como el *offsetFather*, ya explicado en el apartado anterior.

## 4.9. Servicio Web para obtener una metáfora

Este servicio web dado un offset y una profundidad obtiene las metáforas de un concepto, es decir, se obtienen tanto los sinónimos como los hiperónimos fáciles formando la metáfora y este servicio devuelve dichos resultados en formato JSON. Para ello se realiza la siguiente petición GET:

`http://127.0.0.1:8000/metaphor/json/word=palabra&level=nivel`

Donde *palabra* es el concepto a buscar y *nivel* es el grado de búsqueda, igual que en los casos explicados anteriormente para buscar sinónimos, hipónimos e hiperónimos fáciles.

La implementación de dicho servicio es idéntico al ya explicado en el servicio web para obtener sinónimos fáciles e hiperónimos fáciles, con la única diferencia que para poder formar la metáfora se llama a otro servicio implementado, el cual utiliza SpaCy para definir si el concepto es masculino o femenino y si está en singular o plural.

Por ejemplo, si se realizará una búsqueda para la palabra inmueble con un nivel de búsqueda 2, la petición GET sería de la siguiente manera:

`http://127.0.0.1:8000/metaphor/json/word=inmueble&level=2`

En la Figura 4.19 se puede ver el resultado obtenido en formato JSON. En primer lugar aparece la palabra buscada, en este caso inmueble y a continuación un objeto que incluye el *offset* del sinónimo o hiperónimo fácil, el *offsetFather* ya explicado en apartados anteriores y la lista de metáforas (es una casa, es una construcción, son unos edificios, es un edificio).

## 4.10. Servicio Web para obtener un símil

Este servicio web dado un offset y una profundidad obtiene los símiles de un concepto, es decir, se obtienen los hipónimos fáciles formando así el símil y devolviendo los resultados en formato JSON. Para ello se realiza la siguiente petición GET:

`http://127.0.0.1:8000/simil/json/word=palabra&level=nivel`

Donde *palabra* es el concepto a buscar y *nivel* es el grado de búsqueda, igual que en los casos explicados anteriormente. Para la implementación de dicho servicio, se utiliza exactamente el mismo código explicado en el apartado de obtención de hipónimos fáciles pero incluyendo la llamada al servicio de SpaCy para volver a definir si el concepto es masculino o femenino y si es singular o plural.

Por ejemplo, si se realizará una búsqueda para la palabra inmueble con un nivel de búsqueda 2, la petición GET sería de la siguiente manera:

`http://127.0.0.1:8000/simil/json/word=inmueble&level=2`

En la Figura 4.20 se puede ver el resultado obtenido en formato JSON. En primer lugar aparece la palabra buscada, en este caso inmueble y a continuación un objeto que incluye el *offset* del hipónimo fácil, el *offsetFather* ya explicado en apartados anteriores y la lista de símiles (es como una biblioteca es como un colegio, es como un restaurante).

#### 4.11. Front end

#### 4.12. Evaluación

## Easy Words

Introduzca una palabra: Portero  

---

**Resultados para la palabra Portero:**

**1.** 

Un portero es tán **fuerte** como un **gorila** 

---

Un portero es un **vigilante** 

---

Un portero es como un **guardia** 

---

**2.** 

Un portero es tán **ágil** como un **pajaro** 

---

Un portero es un **jugador** 

---

Un portero es como un **futbolista** 

---

Figura 4.8: Prototipo de Irene mostrando todos los resultados para la palabra portero junto con pictogramas

Aprende Fácil

Introduzca una palabra: Portero

Resultados para la palabra Portero:

Un portero es un **vigilante**  
Un portero es como un **guardia**  
Un portero es:  
**fuerte como un gorila**  
**grande como un oso**

a

Z

Figura 4.9: Prototipo mostrando resultado más común para la palabra portero

Aprende Fácil

Introduzca una palabra: Portero

Resultados para la palabra Portero:

Un portero es un **vigilante**  
Un portero es como un **guardia**  
Un portero es:  
**fuerte como un gorila**  
**grande como un oso**

Definición ▾ Ejemplo ▾

**a**

**Z**

Figura 4.10: Prototipo Versión 1 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo separados

## Aprende Fácil

Introduzca una palabra: Portero  

Resultados para la palabra Portero:

Un portero es un **vigilante**  
Un portero es como un **guardia**  
Un portero es:  
**fuerte como un gorila**  
**grande como un oso**

**Definición:** Un portero es alguien que protege una entrada.  
**Ejemplo:** Mi tío trabaja de portero en una discoteca.

Figura 4.11: Prototipo Versión 2 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo a simple vista

Aprende Fácil

Introduzca una palabra: Portero

Resultados para la palabra Portero:

Un portero es un **vigilante**  
Un portero es como un **guardia**  
Un portero es:  
    **fuerte como un gorila**  
    **grande como un oso**

Definición y Ejemplo ▾

Definición: *Un portero es alguien que protege una entrada.*  
Ejemplo: *Mi tío trabaja de portero en una discoteca.*

The interface features a large yellow header bar with the text "Aprende Fácil". Below it is a search bar with the placeholder "Introduzca una palabra:" and the word "Portero" typed in. To the right of the search bar is a magnifying glass icon. The main content area displays search results for the word "Portero". On the left, there is a large, stylized letter "a" and on the right, a large, stylized letter "Z", both drawn with a yellow pencil-like texture. The search results include common analogies ("vigilante", "guardia") and descriptive qualities ("fuerte como un gorila", "grande como un oso"). A button labeled "Definición y Ejemplo" is present, which, when clicked, reveals a detailed definition and an example.

Figura 4.12: Prototipo Versión 3 mostrando resultado más común para la palabra portero junto con una definición y un ejemplo en un mismo botón

## Aprende Fácil

Introduzca una palabra:  

Resultados para la palabra Portero:

1.

Un portero es un **vigilante**  
Un portero es como un **guardia**  
Un portero es:  
**fuerte como un gorila**  
**grande como un oso**

2.

Un portero es un **jugador**  
Un portero es como un **futbolista**  
Un portero es:  
**rápido como un caballo**  
**delgado como un flamenco**



Figura 4.13: Prototipo mostrando todos los resultados para la palabra portero

## Aprende Fácil

Introduzca una palabra:



### Resultados para la palabra Portero:

1.



Un portero es un **vigilante**



Un portero es como un **guardia**



Un portero es:

**fuerte**



como un **gorila**



**grande**



como un **oso**



2.



Un portero es un **jugador**



Un portero es como un **futbolista**



Un portero es:

**rápido**



como un **caballo**



**delgado**



como un **flamenco**



Figura 4.14: Prototipo mostrando todos los resultados para la palabra portero incluyendo los pictogramas

	Lista 1.000 palabras			Lista 5.000 palabras			Lista 10.000 palabras		
	% nada	% sin term sins	% sin sins	% nada	% sin term sins	% sin sins	% nada	% sin term sins	% sin sins
Conceptnet	<b>68,2</b>	<b>72,1</b>	<b>87,6</b>	49,6	<b>53,7</b>	<b>79,7</b>	43,4	<b>47,6</b>	<b>77,6</b>
Wordnet	62,6	47,6	26,5	<b>56,5</b>	33,0	12,7	<b>55,0</b>	30,5	9,2

Figura 4.15: Tabla de resultados de las pruebas

```
GET http://127.0.0.1:8000/easySynonym/json/word=inmueble&level=2

Response headers 5 Request headers 0

date: Sun, 05 May 2019 19:47:33 GMT
server: WSGIServer/0.2 CPython/3.7.0
content-type: application/json
x-frame-options: SAMEORIGIN
content-length: 119

[{"word": "inmueble", "offset": "spa-30-02913152-n", "synonyms": ["casa", "construcción", "edificio", "edificios"]}]
```

Figura 4.16: JSON devuelto al buscar los sinónimos fáciles de inmuble

```
GET http://127.0.0.1:8000/easyHyponym/json/word=inmueble&level=2

Response headers 5
Request headers 0

date: Sun, 05 May 2019 20:03:33 GMT
server: WSGIServer/0.2 CPython/3.7.0
content-type: application/json
x-frame-options: SAMEORIGIN
content-length: 318

[Array[2]
-0: {
    "word": "inmueble"
},
-1: {
    "offsetFather": "spa-30-02913152-n",
    "offset": "spa-30-80000135-n",
    "hyponyms": [Array[19]
        0: "arquitectura",
        1: "centro",
        2: "club",
        3: "residencia",
        4: "hotel",
        5: "casa",
        6: "casa",
        7: "biblioteca",
        8: "ministerio",
        9: "dependencia",
        10: "templo",
        11: "comedor",
        12: "restaurante",
        13: "pista",
        14: "colegio",
        15: "escuela",
        16: "torre",
        17: "templo",
        18: "teatro"
    ],
}
]
```

Figura 4.17: JSON devuelto al buscar los hipónimos fáciles de inmueble

```
GET http://127.0.0.1:8000/easyHyperonym/json/word=inmueble&level=2

Response headers 5
Request headers 0

date: Sun, 05 May 2019 20:12:35 GMT
server: WSGIServer/0.2 CPython/3.7.0
content-type: application/json
x-frame-options: SAMEORIGIN
content-length: 139

[  < > ]
[Array[2]
 -0: {
    "word": "inmueble"
 },
 -1: {
    "offsetFather": "spa-30-02913152-n",
    "offset": "spa-30-04341686-n",
    "hyperonyms": [Array[2]
      0: "construcción",
      1: "estructura"
    ],
 }
]

```

Figura 4.18: JSON devuelto al buscar los hiperónimos fáciles de inmueble

```
GET http://127.0.0.1:8000/metaphor/json/word=inmueble&level=2

Response headers 5 Request headers 0

date: Sun, 05 May 2019 20:23:39 GMT
server: WSGIServer/0.2 CPython/3.7.0
content-type: application/json
x-frame-options: SAMEORIGIN
content-length: 297

[Array[3]
-0: {
  "word": "inmueble"
},
-1: {
  "offsetFather": "",
  "offset": "spa-30-02913152-n",
  "metaphor": [Array[4]
    0: "es una casa",
    1: "es una construcción",
    2: "es un edificio",
    3: "son unos edificios"
  ],
},
-2: {
  "offsetFather": "spa-30-02913152-n",
  "offset": "spa-30-04341686-n",
  "metaphor": [Array[2]
    0: "es una construcción",
    1: "es una estructura"
  ],
}
] ,
```

Figura 4.19: JSON devuelto al buscar las metáforas de la palabra inmueble

GET http://127.0.0.1:8000/simil/json/word=inmueble&level=2

Response headers 5 Request headers (

---

```
date: Sun, 05 May 2019 20:47:31 GMT
server: WSGIServer/0.2 CPython/3.7.0
content-type: application/json
x-frame-options: SAMEORIGIN
content-length: 535
```

□ ↻ <> ⌂

```
[Array[2]
-0: {
    "word": "inmueble"
},
-1: {
    "offsetFather": "spa-30-02913152-n",
    "offset": "spa-30-80000135-n",
    "simil": [Array[19]
        0: "es como una arquitectura",
        1: "es como un centro",
        2: "es como un club",
        3: "es como una residencia",
        4: "es como un hotel",
        5: "es como una casa",
        6: "es como una casa",
        7: "es como una biblioteca",
        8: "es como un ministerio",
        9: "es como una dependencia",
        10: "es como un templo",
        11: "es como una comedor",
        12: "es como una restaurante",
        13: "es como una pista",
        14: "es como un colegio",
        15: "es como una escuela",
        16: "es como una torre",
        17: "es como un templo",
        18: "es como un teatro"
    ],
}
]
```

Figura 4.20: JSON devuelto al buscar los símiles de la palabra inmueble

# Capítulo 5

## Trabajo Realizado

En este capítulo vamos a describir que trabajo hemos hecho cada uno

### 5.1. Trabajo realizado por Irene

Primero investigué las bibliotecas que utilizaremos para el procesado de las palabras, al principio encontramos una biblioteca para el procesado de texto en Python, que es la nltk pero vimos que las etiquetas que ponía a las palabras no eran del todo correctas por lo que buscamos otra biblioteca y encontramos Spacy, con esta ya pudimos etiquetar bien todas las palabras diseñando un programa inicialmente en el Jupyter. A continuación, investigué que tecnologías utilizar para la realización del prototipo tecnológico, encontramos como entorno de desarrollo Pycharm y como framework Django. Una vez seleccionadas las tecnologías, investigamos como se utilizaban y nos pusimos a trabajar en el prototipo tecnológico.

Yo me encargué de conectar las vistas html con la lógica en Python, a continuación vimos como se implementaba un formulario y como se hacia una redirección a vista. Cuando supimos como se hacia todo esto, integraron el código desarrollado en Jupyter en nuestro servicio web finalizando el prototipo tecnológico.

En cuanto a la memoria me la dividí a partes iguales con Pablo, intentando que los dos toquemos todo, por lo que ambos redactamos tanto una parte de la introducción (en la que redacté la motivación) como el estado de la cuestión (yo hice el apartado de lectura fácil y Procesamiento del Lenguaje Natural).

La investigación de como funcionaba Conceptnet y su API la hicimos conjuntamente.

## 5.2. Trabajo realizado por Pablo

Al igual que mi compañera, lo primero que hicimos fue investigar como podíamos etiquetar las palabras, encontramos la librería nltk de Python para hacerlo, pero tras un primer intento nos dimos cuenta de que muchas palabras no estaban etiquetadas como deberían por lo que decidimos buscar alternativas, indagando un poco encontramos Spacy, la probamos y obtuvimos unos resultados mucho mejores que con nltk por lo que decidimos utilizar esta última (todo esto lo hicimos desde el Jupyter).

Cuando terminamos de etiquetar las palabras nos pusimos a investigar herramientas para el desarrollo del prototipo tecnológico y nos decantamos por utilizar Django como framework integrado en Pycharm, que es el entorno de desarrollo.

A continuación empezamos el desarrollo del prototipo tecnológico primero investigando como se utilizaban estas tecnologías(implementar formularios, hacer las redirecciones a vista...). Para finalizar migramos lo hecho desde Jupyter a nuestro servicio web.

Irene y yo nos dividimos la redacción de la memoria de tal manera que los dos hicimos tanto la parte de la introducción como del estado de la cuestión, de la introducción a mí me tocó la parte de los objetivos y del estado de la cuestión la parte de figuras retóricas y servicios web.

La investigación de como funcionaba Conceptnet y su API la hicimos de manera conjunta.

Capítulo **6**

## Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.



# Chapter 6

## Conclusions and Future Work

Conclusions and future lines of work.



Apéndice **A**

## Título

Contenido del apéndice



Apéndice **B**

## Título



# Bibliografía

*Y así, del mucho leer y del poco dormir,  
se le secó el celebro de manera que vino  
a perder el juicio.*

Miguel de Cervantes Saavedra

CALLEJA, P. G. *Generación de recursos lingüísticos mediante la extracción de relaciones entre conceptos*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2017.

GALIANA, A. y CASAS, J. *Introducción al análisis retórico: tropos, figuras y sintaxis del estilo*. Universidad de Santiago de Compostela, 1994.

GARCÍA MUÑOZ, O. *Lectura Fácil: Métodos de redacción y evaluación*. Real Patronato sobre Discapacidad, Ministerio de Sanidad, Servicios Sociales e Igualdad, 2012.

MANNING, C. D. y SCHÜTZE, H. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. ISBN 0-262-13360-1.

MORENO ORTIZ, A. *Diseño e implementación de un lexicón computacional para lexicografía y traducción automática*. Facultad de Filosofía y Letras., 2000.

NOMURA, M., SKAT NIELSEN, G. y TRONBACKE, B. *Directrices para materiales de lectura fácil*. Federación Internacional de Asociaciones e Instituciones Bibliotecarias, 2010.

QUILLIAN, M. R. *Semantic Memory*. Cambridge, 1968.

SHNEIDERMAN, B. y PLAISANT, C. *Diseño de interfaces de usuario : estrategias para una interacción persona-computadora efectiva*. Pearson Education, 2006.

- SOWA, J. *Conceptual structures: Information processing in mind and machine.* Addison-Wesley Longman Publishing Co., Inc., 1983. ISBN 0-201-14472-7.
- TORRES, J. *Servicios Web.* Universidad Complutense de Madrid, 2017.
- VEALE, T. y LI, G. *Exploding the Creativity Myth: The Computational Foundations of Linguistic Creativity.* Bllomsbury Academic, 2012.
- VEALE, T. y LI, G. *Creating Similarity: Lateral Thinking for Vertical Similarity Judgment.* In Proceedings of ACL 2013, the 51st Annual Meeting of the Association for Computational Linguistics, 2013.
- VEGA LEBRÚN, C. A. *Integración de herramientas de tecnologías de información como soporte en la administración del conocimiento.* Trabajo doctorado, Universidad Popular Autónoma del Estado de Puebla, 2005.

*-¿Qué te parece desto, Sancho? – Dijo Don Quijote –  
Bien podrán los encantadores quitarme la ventura,  
pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero  
Don Quijote de la Mancha  
Miguel de Cervantes*

*-Buena está – dijo Sancho –; fírmela vuestra merced.  
–No es menester firmarla – dijo Don Quijote–,  
sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero  
Don Quijote de la Mancha  
Miguel de Cervantes*

