# 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 Mendez 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**


**7 de febrero 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

7 de febrero 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

# Índice de figuras

# Índice de tablas

Chapter 1

# Introduction

Introduction to the subject area.

Capítulo 1

# Introducción

Actualmente, existen diversos colectivos con un entendimiento limitado del castellano, lo que convierte a sus integrantes en personas vulnerables, ya que se las puede engañar fácilmente. Aunque existen multitud de herramientas online para facilitar la comprensión de textos en distintos idiomas, como pueden ser los servicios de traducción instantánea, en los que al escribir un texto se traduce automáticamente a cualquier lengua que el usuario desee, no existen herramientas que permitan definir palabras difíciles utilizando comparaciones o términos más sencillos para facilitar su comprensión. Por este motivo es importante ofrecer un servicio accesible a todo el mundo que pueda definir palabras de una manera clara y sencilla.

## 1.1. Motivación

En nuestra sociedad actual, existen ciertos colectivos como pueden ser las personas con algún tipo de trastorno cognitivo, inmigrantes, ancianos, analfabetos funcionales, niños, etc... que tienen dificultad para aprender conceptos complejos o no tan complejos. Existen multitud de palabras cuyo significado es bastante complicado de explicar de una manera sencilla, por lo que una solución para que cualquier persona lo pueda comprender es hacer uso de metáforas o analogías. Por ejemplo, para explicar que una persona dispone de un gran intelecto se dice que: “eres un cerebrito". Se hace uso de una metáfora para explicar un concepto díficil de entender a través de una explicación sencilla. De esta forma, se puede asimilar el concepto de una manera más rápida y sencilla.

La dificultad para entender conceptos complejos supone una serie de limitaciones en la vida cotidiana, en la forma de relacionarse con otros individuos, en la vida profesional e incluso la vida personal. Por ejemplo, suponemos que un inmigrante necesita realizar gestiones para legalizar su estancia o firmar

documentación. Si no tiene una buena comprensión del lenguaje no sabe las consecuencias de firmar los documentos porque no los entiende. Otro ejemplo sería cuando una persona firma un contrato, como por ejemplo una póliza de seguros o un contrato de trabajo, en el cual añaden ciertas cláusulas que son incomprensibles.

Para ayudar principalmente a estas personas a que puedan entender el significado de cualquier palabra, y de esta forma superar algunas de sus limitaciones, vamos a desarrollar una aplicación que permita definir palabras complejas mediante comparaciones con otras más fáciles ya conocidas por ellos. Por ejemplo, si queremos explicar una palabra compleja como puede ser piraña, podemos describirla utilizando conceptos más simples de la siguiente manera: *"Una piraña nada como un pez y es agresiva como un león"*. Mediante esta comparación, alguien que desconozca completamente el significado de *piraña*, puede hacerse una idea muy aproximada de lo que es.

## 1.2. Objetivos

El objetivo principal es crear una aplicación que dada una palabra compleja para el usuario obtenga una definición clara y sencilla mediante símiles, analogías o metáforas. Esto se puede ver claramente en el ejemplo de la piraña del apartado anterior, ya que, de esa manera cualquier persona que no sepa lo que es una piraña pueda hacerse una idea de cómo es dicho animal y asimilar el nuevo concepto. Además, se utilizarán técnicas centradas en el usuario para diseñar una interfaz lo más usable posible y que así el usuario tenga una experiencia de uso satisfactoria. Por último, el producto se diseñará para que se encuentre al alcance de todas aquellas personas que lo necesiten. Para que todo ello ocurra, los principales objetivos tecnológicos a alcanzar son:

- La aplicación estará construida con servicios web que la doten de funcionalidad para que finalmente se obtenga una aplicación con funcionalidad.
- Los servicios web desarrollados estarán disponibles en una API pública para que todo el mundo pueda utilizarlos.
- La aplicación se construirá de manera incremental, añadiéndole valor al producto poco a poco.
- La aplicación esté formada por distintos módulos independientes cada uno con una funcionalidad específica.
- Hacer un desarrollo que esté centrado en el usuario, que sea lo más amigable posible y que solucione problemas reales de una forma usable.

- Hacer un uso correcto de analogías y metáforas, para que de esta forma resulte más sencillo la comprensión del concepto.

Por último, no se deben de olvidar los objetivos académicos de este trabajo: poner en práctica los conocimientos adquiridos durante el Grado y ampliar nuestros conocimientos gracias a la utilización de herramientas, lenguajes y metodologías nuevas.

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

## 1.3. 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 describe el trabajo realizado por cada uno de los autores.

Capítulo 2

# Estado de la Cuestión

Para que un contenido ilustrativo o en formato de texto sea sencillo de entender existe una adaptación llamada lectura fácil cuyo objetivo es facilitar la accesibilidad al mismo. En la sección 2.1 de este capítulo se explicará qué es la lectura fácil y algunas pautas que se pueden seguir para escribir correctamente un texto en lectura fácil. Por otro lado, para que las definiciones de las palabras díficiles sean fácilmente comprensibles para el usuario, utilizaremos figuras retóricas con las que compararemos conceptos complejos con otros más sencillos. De esta manera el usuario se podrá hacer una idea de sus características principales. En la sección 2.2 se hablará de las figuras retóricas y se explicará los tres tipos fundamentales que se van a utilizar para la realización de este trabajo. Pero para poder plasmar todo esto en el trabajo realizado y conseguir una funcionalidad, se deberan usar servicios web, los cuáles se han convertido en una tecnología cotidiana para la mayoría de los usuarios sin ser estos mismos conscientes de su uso. En la sección 2.3 se explicará detalladamente que es un servicio web, los tipos que existen, sus características principales, su aquitectura y las ventajas de ser utilizados así como sus deventajas. Por último, las comparaciones entre conceptos más complejos con conceptos más sencillos, se construirán a través de conceptos relacionados con la palabra que el usuario busque, es por ello que en la sección 2.4 se explicará en qué consiste en procesamiento del Lenguaje Natural, los tipos de redes semánticas que existen y las diversas aplicaciones que actúan como redes semánticas y que son capaces de procesar el Lenguaje Natural.

## 2.1. Lectura Fácil

Se llama lectura fácil 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,

Figura 2.1: Logo Lectura Fácil

es la adaptación de textos, ilustraciones y maquetaciones que permite una mejor lectura y comprensión. Este trabajo se va a centrar en la lectura fácil aplicada a textos.

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[1]. En la actualidad, podemos distinguir los documentos en lectura fácil gracias al logo de la Figura 2.1.

Los documentos escritos en Lectura Fácil (Nomura et al., 2010) son documentos de todo tipo que siguen las directrices internacionales de la IFLA[2] y de la Inclusion Europe[3] 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 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 carácteres y utilizar oraciones simples. Por ejemplo, la oración *Caperucita ha ido a casa de su abuela y ha desayu-*

[1] https://dilofacil.wordpress.com/2013/12/04/el-origen-de-la-lectura-facil/
[2] International Federation of Library Associations and Institutions
[3] Una asociación de personas con discapacidad intelectual y sus familias en Europa

*nado 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.

Debemos también hacer hincapié en la distinción entre palabras fáciles y complejas, puesto que son de gran importancia para la lectura fácil. Las palabras complejas son aquellas que no se utilizan a menudo en nuestra sociedad, como por ejemplo: melifluo o inefable. Es por ello que este tipo de palabras deben estar totalmente descartadas en la lectura fácil, y en su lugar debemos introducir palabras fáciles, que son aquellas que se utilizan asiduamente. La RAE (Real Academia Española) dispone de un documento con las mil palabras más usadas[4].

## 2.2. Figuras retóricas

Las figuras literarias (o retóricas) son 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. La metáfora, el símil y la analogía se basan en la comparación de dos conceptos: de 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*

[4] http://corpus.rae.es/lfrecuencias.html

es el tenor, *luceros* es el vehículo y el fundamento es la belleza de los ojos (Galiana y Casas, 1994).

En este trabajo se van a utilizar tres tipos de figuras retóricas (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 muy preciado se dice: "Mi tiempo vale 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".

## 2.3. 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[5]. Según el W3C (*World Wide Web Consortium*)[6],*"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 encontrarle. De esta forma tendremos la posibilidad de que una aplicación

[5] http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html
[6] https://www.w3.org/

localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

Por otro lado, los servicios web pueden definirse tanto a nivel conceptual como a nivel técnico. A nivel técnico podemos 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[7]. Los mensajes SOAP tienen el formato representado en el Listing 2.1:

Listing 2.1: 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>
```

  - <Envelope>: elemento raíz de cada mensaje SOAP y contiene dos elementos: <Header>que es opcional y <Body>que es obligatorio, ambos elementos los describiremos a continuación.
  - <Header>: es un elemento que se utiliza para indicar información acerca de los mensajes SOAP. Por ejemplo, como se puede ver en el Listing 2.1 dentro del campo Header estarían los campos de reservas y pasajeros.

[7] https://www.ibm.com/support/knowledgecenter/es

  - <Body>: elemento que contiene información dirigida al destinatario del mensaje. Haciendo referencia al Listing 2.1 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>: elemento en el que se notifican los errores.

- Servicios Web RESTful: RESTful es un protocolo que suele integrar mejor con HTTP que los servicios basado 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.

### 2.3.1. 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[8]:

- Service Discovery: responsable de centralizar los servicios web en un directorio común, de esta forma es más sencillo buscar y publicar.
- Service Description: como ya hemos comentado con anterioridad, los servicios web se pueden definir así mismos, por lo que una vez que los localicemos el Service Description nos dará la información para saber que operaciones soporta y como activarlo.
- Service Invocation: 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.
- Transport: todos los mensajes han de ser transmitidos de alguna forma entre el servidor y el cliente. El protocolo elegido para ello es HTTP *(HyperText Transfer Protocol)*.

[8] https://diego.com.es/introduccion-a-los-web-services

### 2.3.2. 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.
- 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.

### 2.3.3. Desventajas de los Servicios Web

El uso de servicios web también tiene algunas desventajas[9]:

- 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.

## 2.4. Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es una rama de la Inteligencia Artificial que se encarga de investigar la manera de comunicar

[9] http://fabioalfarocc.blogspot.com/2012/08/ventajas-y-desventajas-del-soap.html

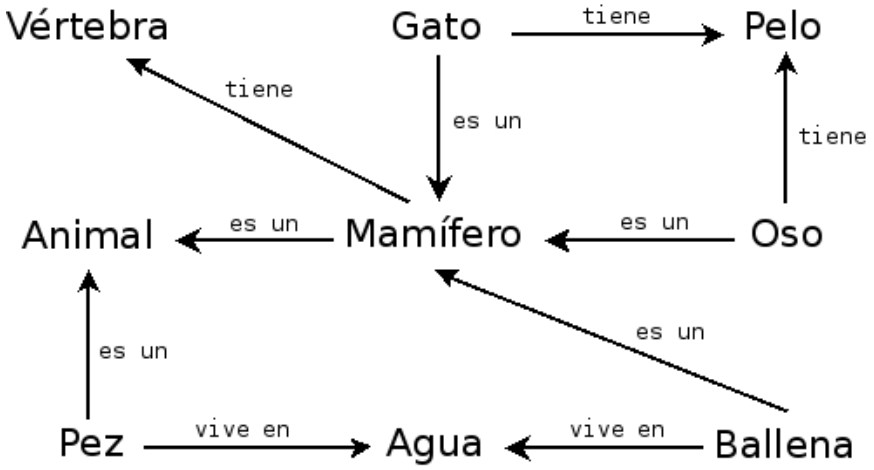


Figura 2.2: Ejemplo Red Semántica

máquinas con 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). El Procesamiento del Lenguaje Natural se ayuda de las redes semánticas, puesto que estas son una forma de representación del conocimiento en la que los conceptos que componen el mundo y sus relaciones se representan mediante un grafo. 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. Podemos ver un ejemplo en la Figura 2.2, donde el nodo *Oso* representa un concepto, en este caso un sustantivo que identifíca a un tipo de animal, y otro nodo *Pelo*. Su relación 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[10]:

- 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. Haciendo referencia a la Figura 2.2, tanto la palabra ballena, oso, y gato tienen en común que son mamíferos.

- 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. Un ejemplo de esto, haciendo referencia a la Figura 2.2 sería: la ballena vive en el agua.

[10] http://elies.rediris.es/elies9/4-3-2.htm


Figura 2.3: Ejemplo Grafo Conceptual

- Grafos Conceptuales: existen dos tipos de nodos: nodos de conceptos, los cuáles 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 2.3 (Sowa, 1983) en la cual la frase *"Man biting dog"* quedaría representada. 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 aplicaciones web que actúan como redes semánticas y son capaces de procesar el Lenguaje Natural. A continuación, hablaremos de algunas de ellas.

### 2.4.1. ConceptNet

Es una red semántica creada por el MIT *(Massachusetts Institute of Technology)* en 1999, diseñada para ayudar a los ordenadores a entender el significado de las palabras. Está disponible en múltiples idiomas, como el español, el inglés o el chino. ConceptNet dispone de una aplicación web[11], donde seleccionas el idioma deseado y añades la palabra a buscar. En la Figura 2.4, podemos ver que si añadimos la palabra chaqueta nos devuelve sus sinónimos, términos relacionados, símbolos, etc... Por otro lado, ConcepNet dispone de un servicio web[12] que devuelve los resultados en formato JSON. Siguiendo con el mismo ejemplo anterior, podemos ver en el Listing 2.2 los resultados en dicho formato para la palabra chaqueta. Este consta de cuatro campos principales[13]:

- @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

[11] http://conceptnet.io/
[12] http://api.conceptnet.io
[13] https://github.com/commonsense/conceptnet5/wiki/AP

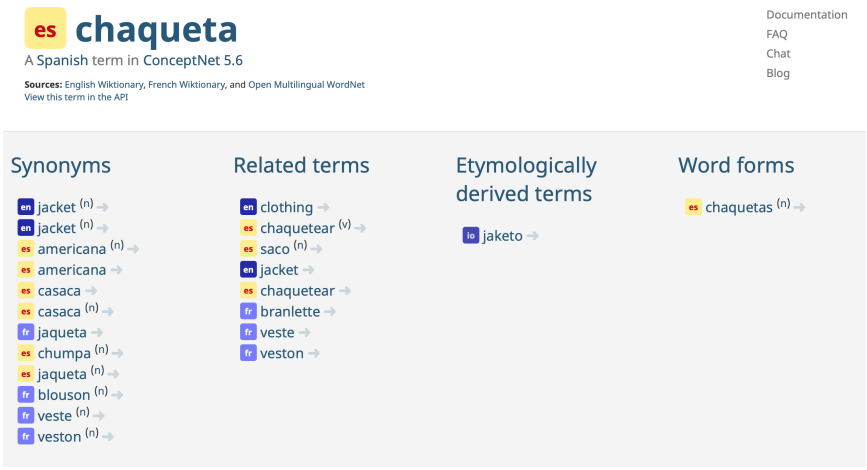


Figura 2.4: Resultados de ConcepNet para la palabra chaqueta

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.
    */a/[/r/Synonym/,/c/es/chaqueta/n/,/c/es/americana/]*, nos 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: 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 por qué 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: indica de que frase del lenguaje natural se han extraido los datos que estan guardados en conceptnet. En nuestro caso es 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.

- 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.

Listing 2.2: 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"
    }

}
```


### 2.4.2. Thesaurus

Es una aplicación web[14] que se autodefine como principal diccionario de sinónimos de la web. Esta página ofrece la posibilidad de introducir una palabra para poder conocer sus sinónimos, pero solamente devuelve resultados en inglés. Aparte del listado de sinónimos, Thesaurus te dice que tipo de palabra es y una definición de la misma. Esta aplicación proporciona una API tipo RESTful[15] que obtiene los sinónimos de una palabra y 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 |. Podemos ver en el Listing 2.3 un ejemplo de como sería el resultado de una petición en formato XML y en el Listing 2.4 en formato JSON. Ambos son muy similares, por ejemplo en formato XML podemos ver que devuelve el tipo de categoría de las palabras, en este caso son sustantivos y a continuación aparcen los sinónimos. En caso de que alguna palabra sea un antónimo aparecera 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.

Listing 2.3: Ejemplo de salida de Thesaurus en formato XML


```
<response>
   <list>
      <category>(noun)</category>
      <synonyms> order | war (antonym) </synonyms>
   </list>
```


[14] https://www.thesaurus.com/
[15] http://thesaurus.altervista.org/

```
    <list>
        <category>(noun)</category>
        <synonyms> harmony | concord | concordance </synonyms>
    </list>
    <list>
        <category>(noun)</category>
        <synonyms> public security | security </synonyms>
    </list>
    <list>
        <category>(noun)</category>
        <synonyms> tratado de paz | pacificacion | acuerdo | pact | accord </
            synonyms>
    </list>
```

Listing 2.4: Ejemplo de salida de Thesaurus en formato JSON

```
{
    "response":
    [
        { "list":
            { "category": "(noun)", "sinonimos": "order | war (antonym)" }
        },
        { "list":
            { "category": "(noun)", "sinonimos": "armonia | concordia | concordancia
                " }
        },
        { "lista":
            { "categoria": "(sustantivo)", "sinonimos": "seguridad publica |
                seguridad" }
        },
        { "lista":
            { "categoria": "(sustantivo)", "sinonimos": "tratado de paz |
                pacificacion | tratado | pacto | acuerdo" }
        }

    ]
}
```

### 2.4.3. Thesaurus Rex

Thesaurus Rex[16] es una red semántica. La aplicación permite introducir una palabra y devuelve palabras relacionados, o bien puedes introducir un par de palabras (separadas por el operador & ) y devolverá las categorías que comparten ambos términos. Thesaurus Rex solo admite palabras en inglés. En la Figura 2.5 podemos ver los resultados para la palabra *house*. La aplicación devuelve estos resultados en formato XML, y este como se puede ver en el Listing 2.5 los divide en distintos campos: *Categories*, *Modifiers* y *CategoryHeads*. Como se puede ver todos los resultados tienen un peso asignado, esto significa que cuanto mayor sea el peso mayor es la similitud con el concepto dado, y en la página aparecerá dicha palabra en un tamaño superior al resto. Los campos que se encuentran dentro del apartado *categories* son los resultados de mayor grano fino que Thesaurus Rex ha encontrado, los que se encuentran dentro de *modifiers* son atributos del concepto a buscar y por último los que se encuentran en *categoryHeads* son las categorías más simples que se han para dicho concepto. Thesaurus Rex utiliza la Web Semántica para generar sus resultados, con lo cual la información disponible no es fija, sino que varía según los datos actuales de la web. La ventaja de

[16] http://ngrams.ucd.ie/therex3/

utilizar esta herramienta es que se encuentra en continua actualización, pero el incoveniente es que en algunos casos la información puede resultar un poco extraña dado que se creea semiautomáticamente desde contenido de la web (Calleja, 2017).

Listing 2.5: Ejemplo formatos XML Thesaurus Rex

```
<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>
      <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>
```

### 2.4.4. Metaphor Magnet

Aplicación web[17], que dada una palabra crea metáforas. El objetivo (Veale y Li, 2012) de dicha aplicación es encontrar metáforas comunes en los n-gramas de Google. Se llama n-grama[18] a una subsecuencia de n elementos consecutivos en una secuencia dada, es decir, contar la cantidad de veces que aparece una palabra, la cantidad de veces que aparecen dos palabras juntas, tres palabras, etc... Google utiliza estos mapeos para interpretar metáforas. Esta aplicación está limitada, ya que como podemos observar en la Figura 2.6 introduciendo la palabra *house* solo está disponible para el inglés. Esta consulta devuelve un fichero XML como el expuesto en el Listing 2.6, que puede ser utilizado para otras aplicaciones de Procesamiento de Lenguaje Natural.

Listing 2.6: Ejemplo formatos XML Metaphor Magnet

[17] http://ngrams.ucd.ie/metaphor-magnet-acl/
[18] https://en.wikipedia.org/wiki/N-gram

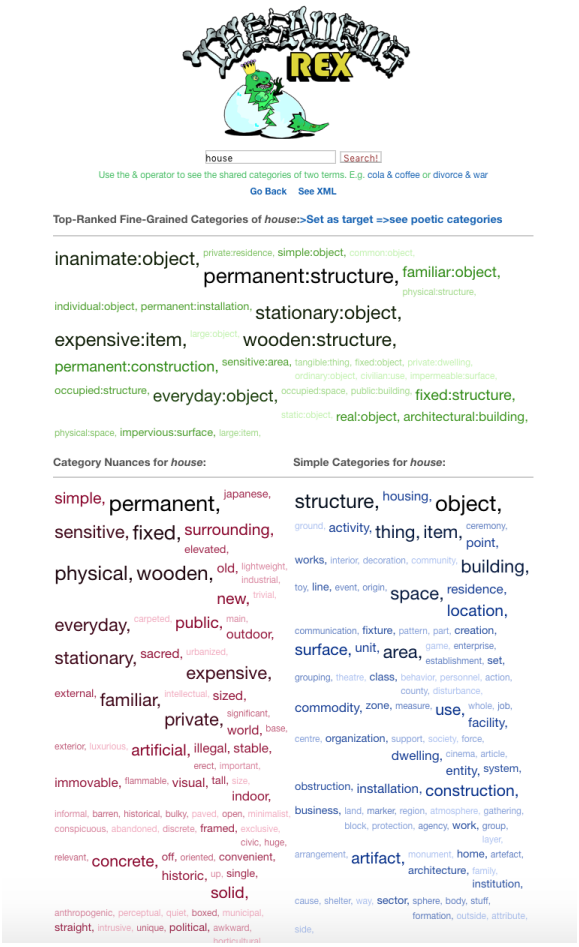


Figura 2.5: Resultados búsqueda Thesaurus Rex con la palabra *house*

```
<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">
        <Text> charming:mansion </Text>
        <Score> 83 </Score>
```


Figura 2.6: Resultados búsqueda Metaphor Magnet con la palabra *house*

```
    </Source>
    <Source Name="house">
        <Text> strong:rock </Text>
        <Score> 80 </Score>
    </Source>
    <Source Name="house">
        <Text> strong:elephant </Text>
        <Score> 80 </Score>
    </Source>
</Metaphor>
```

### 2.4.5. WordNet

Es un *corpus*[19] perteneciente a NLTK *(Natural Languaje Toolkit)*[20] que almacena distintos tipos de palabras como sustantivos, verbos, adjetivos y adverbios ignorando preposiciones, determinantes y otras palabras funciona-

[19] Colección de documentos de texto
[20] Conjunto de bibliotecas y programas para el Procesamiento del Lenguaje Natural

les en varios idiomas como el español, el inglés o el francés. Los conceptos se agrupan en conjuntos de sinónimos cognitivos llamados *synsets*. Cada *synset* contiene una serie de sinónimos llamados *lemmas*, hiperónimos que son palabras cuyo significado está incluido en el de otras[21], por ejemplo: mamífero es hiperónimo de gato y de perro ya que los gatos y los perros pertenecen al conjuntos de los mamíferos, hipónimos que son el antónimo del hiperónimo, es decir, son palabras cuyo significado incluye el de otra[22], por ejemplo: gato es hipónimo de mamífero ya que está incluido dentro del conjunto de los mamíferos. Por último, holónimos que son 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. Por otro lado, también contiene una breve definición y en muchas ocasiones, oraciones cortas que explican su significado. Además, por cada uno de los *lemmas* que contenga el *synset*, se podrán consultar sus antónimos. En caso de que la palabra tenga distintas acepciones, aparecerá un *synset* por cada una[23].

Listing 2.7: Ejemplo de uso de WordNet

```
import nltk
from nltk.corpus import wordnet as wn
lista = wn.synsets('coche', lang='spa')
print(lista[0].lemma_names('spa'))
print(lista[0].definition())
print(lista[0].examples())
print(lista[0].hypernyms())
print(lista[0].hyponyms())
print(lista[0].member_holonyms())
print(lista[0].lemmas()[0].antonyms().name())
```

Se puede ver en el Listing 2.7 la forma de obtener la lista de *synsets* de la palabra coche. Primero se debe importar el paquete nltk y el *corpus* WordNet. A continuación, se invoca al método *synsets* de WordNet, introduciendo como parámetros la palabra de la que se quiere obtener los *synsets* y el lenguaje en el que está dicha palabra (si es una palabra en inglés se puede dejar el segundo parámetro en blanco). Después, del primer *synset* de la lista, se muestran por pantalla cada uno de sus sinónimos (en español), su definición, una oración de ejemplo, sus hiperónimos, hipónimos y holónimos. Por último, del primer *lemma*, se obtienen sus antónimos.

Listing 2.8: Ejemplo de relaciones entre *synsets*

```
import nltk
from nltk.corpus import wordnet as wn
synset1 = wn.synset('dog.n.01')
synset2 = wn.synset('cat.n.01')
print(synset1.path_similarity(synset2))
```

En el Listing 2.8 se ilustra como se puede obtener el grado de similitud que tienen dos *synset*. Las primeras instrucciones son similares al Listing

[21] https://dle.rae.es/?id=KRW1qe2
[22] https://dle.rae.es/?id=KU5UAn5
[23] http://www.nltk.org/howto/wordnet.html

2.7, con la diferencia, de que en este caso a la hora de invocar al método *synset*, se le pasa por parámetro el identificador de un *synset* específico. Dicho identificador, se puede obtener invocando al método *synsets* explicado anteriormente. En el caso de *path similarity* dicho grado es un número entre 0 y 1, que cuanto más grande sea, mayor similitud existe entre ambos. Si no existiera ninguna similitud, aparecerá la palabra *none*. Para obtener este índice, se navega entre los hiperónimos e hipónimos de uno de los *synset* y se intenta llegar al otro. Cuanto menor sea el número de saltos necesarios para alcanzarlo, mayor será el grado.

## 2.5. Conclusiones

Para la obtención de las palabras fáciles que se utilizarán para definir los conceptos difíciles, se decidió consultar las 1000 formas más utilizadas en castellano según la RAE, sin embargo, muchas de estas formas no son útiles ya que son conjunciones o preposiciones y lo que se necesitaba eran sustantivos, verbos, adverbios y adjetivos para ello era necesario clasificar las palabras según su categoría gramatical para solo tener en cuenta las que pertenezcan a estas cuatro. El primer recurso que se utilizó para la clasificación fue Stanford CoreNLP, que es una herramienta implementada en Java por la universidad de Stanford que permite el análisis de texto (Martín Guerrero, 2018), sin embargo, la clasificación que realizaba no era la correcta por lo que se descartó su uso y se investigaron otras opciones. Posteriormente, se utilizó SpaCy que es una librería para el procesamiento de texto escrita en Python[24], se obervó que la clasificación de las palabras era satisfactoria por lo que fue lo que finalmente se eligió. Una vez que se clasificaron las palabras obtenidas de la RAE, se empezó a trabajar con Conceptnet ya que es una red semántica muy completa en castellano. Sin embargo, cuando se implementó el primer prototipo y se querían obtener los términos relacionados de un concepto, se observó que dichos términos relacionados no tenían nada que ver con la palabra inicialmente buscada, por lo que se descartó su uso. Se investigaron otras herramientas y finalmente, se decidió trabajar con WordNet, en su versión en español, ya que contiene una biblioteca muy amplia de palabras así como ejemplos, definiciones, sinónimos, hiperónimos y antónimos de las mismas, además los términos relacionados obtenidos son mucho más coherentes que los que se podían lograr con Conceptnet.

[24] https://explosion.ai/blog/introducing-spacy

Capítulo 3

# Herramientas Utilizadas

Como base para poder trabajar en el proyecto de la manera más ordenada y limpia posible, ambos compañeros hemos utilizado ciertas tecnologías que nos facilitan dicho trabajo y nos ayudan a saber en qué momento exacto se encuentra cada uno en la realización del mismo. Por ello, en este capítulo se va a explicar todas las herramientas utilizadas con este fin, desde las más básicas que se explicarán en el apartado 3.1, como en el apartado 3.2 se explicarán las herramientas utilizadas para la gestión de tareas, ya que ambos debemos dividirnos el trabajo en tareas, siendo éstas lo más simples e independientes unas de las otras para que no nos inmiscuiyamos en sendos trabajos. Y en el capítulo 3.3 explicaremos los pasos iniciales para instalar Django correctamente.

## 3.1. Herramientas básicas

- Repositorio: como herramientas básicas para poder comunicarnos tanto entre nosotros como con los directores, e ir teniendo guardadas todas las versiones de nuestro código así como de la memoria, hacemos uso de un repositorio común en *GitHub*. La dirección del mismo sería https://github.com/NILGroup/TFG-1819-Analogias, donde se pueden ver todos los cambios realizados desde el principio del proyecto.
- Pruebas Unitarias:
- Pruebas de Integración:

## 3.2. Herramientas para la Gestión de Tareas

Nuestro equipo no se rige por una metodología en especial, ya que al ser únicamente dos personas no podemos utilizar Scrum o cualquier otra

metodología ágil. Pero si que hemos adoptado ciertas características de ese tipo de metodologías para realizar nuestro trabajo. Hemos hecho uso de un gestor de tareas con el fin de ayudarnos a saber cuál es la próxima tarea por hacer, cuáles están en proceso y cuáles se encuentran ya finalizadas. Existen varios gestores de tareas, pero nos hemos decantado por Trello, ya que dispone de una interfaz simple, amigable y que no lleva a confusión a la hora de crear nuevas tareas o moverse por el tablero. Se han añadido tres columnas:

- Lista de tareas: en dónde se ven representadas todas las tareas a realizar, desgranadas al mayor detalle posible e intentando que éstas sean lo más independientes las unas de las otras. De esta forma, nos aseguramos que cada integrante del equipo trabaja en una tarea específica que no influye en el trabajo del compañero.

- En proceso: en el momento en el que un integrante del grupo se asigna una tarea, ésta se traslada a la siguiente columna. Lo que indica que se encuentra en proceso de realización y que ningún otro compañero puede realizarla.

- Hecho: última columna de nuestro tablero. Cuando una tarea se encuentra en dicha columna implica que la tarea ha sido terminada y validada, esta validación la deben dar los directores y nunca los propios integrantes del grupo.

Para que se pueda entender con mayor claridad, podemos ver la Figura 3.1 dónde en cada columna se ve claramente la descripción de la tarea y quién la tiene asignada.

## 3.3. Django

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 (Eugercios Suárez et al., 2018). 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. Gracias a utilizar Django se puede crear un software[1]:

- Completo: provee de casi todo lo que los desarrolladores esperan utilizar, sigue principios de diseño consistentes y dispone de una amplia y actualizada documentación.

[1] https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introducción

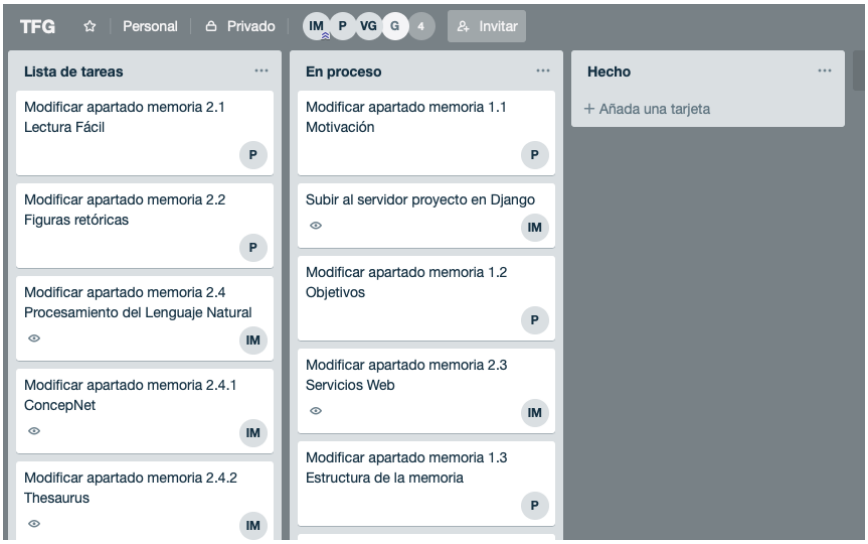


Figura 3.1: Ejemplo Gestor de Tareas en Trello

- Versátil: puede funcionar con cualquier *framework* en el lado del cliente, y puede devolver contenido en casi cualquier formato incluyendo HTML, RSS feeds, JSON y XML. Internamente, ofrece opciones para casi cualquier funcionalidad, como por ejemplo: distintos motores de base de datos , motores de plantillas, etc...

- Seguro: Django proporciona una manera segura de administrar cuentas de usuario y contraseñas, por lo que las *cookies* solo contienen una clave y los datos se almacenan en la base de datos, o se almacenan directamente las contraseñas en un hash de contraseñas.

- Escalable: usa un componente basado en la arquitectura *shared-nothing*, es decir, cada parte de la arquitectura es independiente de las otras, y por lo tanto puede ser reemplazado o cambiado si es necesario.

- Mantenible: el código de Django está escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable, como por ejemplo el patrón MVC *(Model View Controller)*, el cuál agrupa código relacionado en módulos. Por otro lado, utiliza el principio No te repitas *Don't Repeat Yourself* (DRY) para que no exista una duplicación innecesaria, reduciendo la cantidad de código.

- Portable: Django está escrito en Python, el cual se ejecuta en muchas plataformas. Lo que significa que no está sujeto a ninguna plataforma en particular, y puede ejecutar sus aplicaciones en muchas distribuciones de Linux, Windows y Mac OS X.

Capítulo 4

# Trabajo Realizado

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

## 4.1. 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 ntlk 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, integramos 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.

## 4.2. Trabajo realizado por Pablo

Al igual que mi compañera, lo primero que hicimos fue investigar como podíamos etiquetar las palabras, encontramos la librería ntlk 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 ntlk 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.

El diseño de la Interfaz debe cumplir las Ocho Reglas de Oro del diseño de interfaces:

- 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 que 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.

Capítulo 5

# Conclusiones y Trabajo Futuro

Conclusiones del trabajo y líneas de trabajo futuro.

Chapter 5

# 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 linguisticos mediante la extracción de relaciones entre conceptos*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2017.

Eugercios Suárez, G., Gutiérrez Merino, P. y Kaloyanova Popova, E. *Análisis emocional para la inclusión digital*. Trabajo de fin de grado, Universidad Complutense de Madrid, 2018.

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.

Martín Guerrero, A. *PICTAR: una herramienta de elaboración de contenido para personas con TEA basada en la traducción de texto a pictogramas*. Trabajo de fin de máster, Universidad Complutense de Madrid, 2018.

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.

Sowa, J. Conceptual structures: Information processing in mind and machine. vol. 1983(1983), 1983.

Torres, J. *Definición SOAP*. Universidad Complutense de Madrid, 2017.

Veale, T. y Li, G. *Exploding the Creativity Myth: The Computational Foundations of Linguistic Creativity*. Bllomsbury Academic, 2012.

Vega Lebrún, C. A. *Integración de herramientas de tecnologías de información como soporte en la adm 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*