API de servicios web orientados a accesibilidad

Sheila Plaza Estévez
Nerea Ramírez Lamela
Carmen Acosta Morales

Directores:
Raquel Hervás Ballesteros
Gonzalo Rubén Mendez Pozo

Trabajo de fin de grado del Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2015-2016

Documento maquetado con TEXIS v.1.0.

API de servicios web orientados a accesibilidad

Sheila Plaza Estévez
Nerea Ramírez Lamela
Carmen Acosta Morales

Directores:
Raquel Hervás Ballesteros
Gonzalo Rubén Mendez Pozo

Trabajo de fin de grado del Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2015-2016

Índice

Índice de figuras

Dedicatorias

Este proyecto queremos dedicarlo a todas esas persona que han estado con nosotras ayudándonos y apoyándonos de una manera o de otra. A nuestras familias por ser un apoyo incondicional y a nuestros tutores por la ayuda y paciencia que han demostrado durante todo el año.

Agradecimientos

Queremos agradecer en primer lugar el esfuerzo realizado por nuestros tutores Raquel Hervás y Gonzalo Méndez, por su gran apoyo, ayuda y paciencia durante todo el año, ya que nos han proporcionado gran ayuda para poder desarrollar este proyecto. También queríamos agradecer a los proyectos anteriores que de acceso a su trabajo, ya que con el proyecto del AraConversor hemos podido desarrollar nuestro servicio de pictogramas.

Resumen

Hoy en día vivimos en una sociedad en la que las tecnologías son accesibles para gran parte de la población y las cuales tienen como finalidad, entre otras cosas, facilitar la vida de las personas en la medida de lo posible.

Este proyecto busca hacer uso de esas tecnologías para facilitar la accesibilidad a textos para cualquiera que pueda tener dificultades con ellos, en mayor o menor medida. La finalidad de nuestro proyecto no es crear la aplicación para el uso directo de las personas sino desarrollar los servicios, y el acceso a los mismos, que faciliten la implementación de estas aplicaciones. Aún así en este proyecto se crea una aplicación de ejemplo para Android, en la cual se hace uso de los servicios como muestra de un posible uso de los mismos.

Para facilitar el acceso a estos servicios y la comprensión de los mismos, en este proyecto se crea también una API web en la cual quedan todos ellos explicados, como acceder a ellos, su descripción, ejemplos de llamadas y de resultados de las llamadas, en definitiva, todo lo que pueda ayudar a un desarrollador interesado en ellos a utilizarlos sin mayor problema.

Las aplicaciones posibles de estos servicios son de lo mas extensas. Así, por ejemplo, en un dispositivo móvil, si un usuario está leyendo un libro y no entiende una palabra o le resulta bastante complicada, tan solo tendría que sacar el móvil, introducir la palabra en una aplicación que llamara a uno de nuestros servicios y obtener el resultado requerido de esa palabra, de forma rápida y cómoda.

El proyecto queda dividido en tres partes: el desarrollo de los servicios de accesibilidad, la web de la API explicando cada uno de estos servicios y una aplicación Android de ejemplo de utilización.

Abstract

Nowadays we live in a society in which technologies are accessible for most of the people. These technologies have as a goal, mostly, helping people.

This project's goal is to use those technologies to make texts more accesible for whoever may have difficulties with them. This project goal is not develop an aplication which makes the accesibility to texts easier, but to create the services that can help to develop those kind of aplications. However, we have developed and example aplication.

To ease the access and undertanding of those services, this project creates an API that explains each one of them, as well as their description, examples, and everything a developer might need if he is interested in using them.

There are plenty of possible aplications of these services. For example, if a user has a mobile device, and he or she is reading a book and there is a word that is not easy to understand, he or she would just have to take the device, introduce the word in the aplication that is using our services, and will have the result in a easy and fast way.

This project is structured in three parts, the development of the accessibility services, the API explaining each one of these services, and an Android application as an example of how to use them.

Palabras Clave

- REST
- API
- Accesibilidad
- Servicio web
- XML
- JSON

Key workds

- REST
- API
- Accessibility
- Web services
- XML
- JSON

Capítulo 1

Introducción

1.1. Motivación

Desde hace ya tiempo el ser humano ha buscado medios de comunicación más allá de la palabra simplemente hablada, ya sea para comunicarse directamente con otras personas cercanas, para transmitir lo que piensa o para dar a conocer una verdad que cree que concierne a otros individuos. Hoy en día eso es cada vez más posible, ya que las comunicaciones alcanzan cada vez a más gente, siendo su impacto cada vez mayor. Sin embargo, no siempre un mensaje es accesible para todo el mundo, incluso cuando físicamente se puede acceder a él.

Hay muchas limitaciones que no conciernen al campo de las comunicaciones a la hora de que un mensaje pueda llegar a un público. La más obvia puede resultar el idioma en el que se transmite, y por eso es la más tratada y la primera que se lleva a cabo siendo, posiblemente, también la más sencilla de tratar.

Sin embargo, una a la que no se le presta tanta atención y que es igualmente limitante es la capacidad de comprensión de un idioma para las personas que lo comparten. Esto quiere decir que, dentro de una comunidad con un mismo lenguaje, hay personas que no son capaces de entender palabras complejas que a otras personas les pueden resultar comprensibles. Este escenario se puede dar en distintas situaciones: gente con problemas de aprendizaje con alguna discapacidad que limite el mismo.

Pero eso no acaba ahí. A menudo no se trata de una limitación en el aprendizaje de la persona en si misma sino un problema con el texto a leer, ya que puede tener una complejidad de términos totalmente específica del texto tratado, en el cual una persona ajena a esa serie de términos puede encontrarse perdida ante el texto. Esta situación se puede dar con asuntos legales, textos científicos, conceptos tecnológicos...

Siendo conscientes de esta necesidad, en la Facultad de Informática, a lo

largo de los años, se han ido realizando diferentes proyectos de gran utilidad social tales como simplificación de textos para facilitar su comprensión, traducción de textos a pictogramas para aquellas personas que no son capaces de leerlos, reconocimiento de la complejidad de las palabras dentro de un texto... Todos estos proyectos se han ido creando a lo largo de los años en distintos lenguajes, formatos y estilos.

Nuestro proyecto surge de la necesidad de darle una utilidad social a esos proyectos, haciéndolos más unificados y accesibles, mediante una API web y una aplicación para Android a modo de ejemplo de uso de los recursos de la API.

Finalmente cabe destacar que este proyecto se enmarca dentro de otro que recibe el nombre de: *IDiLyCo: inclusión digital, lenguaje natural y comunicación. TIN2015-66655-R (MINECO/FEDER).* , el cual está cofinanciado por el Ministerio de Economía y Competitividad y el Fondo Europeo de Desarrollo Regional (FEDER) dentro del Plan Nacional de I+D+i.

El objetivo del proyecto, es facilitar la comunicación y el acceso a la información digital a colectivos que, por su diversidad funcional pueden encontrar problemas a la hora de utilizar el lenguaje para comunicarse o acceder a dicha información.

IDiLyCo busca el desarrollo de soluciones tecnológicas que permitan de una manera sencilla y con gran capacidad de configuración, la inclusión digital de estas personas.

Ofrece a los diseñadores o programadores de nuevas herramientas que deseen permitir la inclusión digital de personas con discapacidad, integrar en sus implementaciones los servicios desarrollados en este proyecto, ahorrando así un enorme coste en investigación y desarrollo de estas tecnologías accesibles .

1.2. Objetivo

Nuestro objetivo, como acabamos de ver, es conseguir que los recursos de mejora de la comunicación estén unificados, completos y accesibles para todo aquel que quiera utilizarlos. Además aportaremos ejemplos de posibles utilizaciones de los mismos que puedan servir como ideas para potenciales interesados en estos recursos.

Para ello, en este proyecto nos encargaremos de revisar los proyectos anteriores, corrigiendo fallos que pudieran tener y mejorando funcionalidades. Así mismo nos encargaremos de unificar todos estos proyectos, poniéndolos todos en un mismo lenguaje, así como en un mismo servidor y guardando la consistencia entre los diferentes servicios, tanto en la forma de entrada como

en la de salida, teniendo dos formatos para la misma, XML y JSON. Además los pondremos todos en una API común para su mayor accesibilidad de cara a usuarios que puedan estar interesados en utilizarlos, así como ejemplos de utilización para que sea lo más claro posible.

1.3. Organización del proyecto

Podría decirse que el proyecto está dividido en tres partes principales: servicios REST, la API de los mismos y un ejemplo Android de utilización de los mismos.

- ***Servicios:*** En la parte de servicios nos encargaremos de revisar servicios anteriores, hacerlos consistentes unos con otros tanto en lenguaje en el que estén implementados como en formato, corregir fallos y hacer comprobaciones de errores de los mismos. Además ampliaremos con nuevos servicios para hacerlos lo más completos posible. Para ello tendremos que analizar algunas páginas de interés como: páginas web de sinónimos y antónimos, ficheros de palabras más usadas para deducir la complejidad de las mismas, webs de definiciones y de traducciones al inglés... Todo esto tendrá como resultado un servidor con unos servicios funcionales y fácilmente accesibles que facilitarán el cumplimiento de nuestros objetivos.

- ***API:***De cara a la parte de la API nos encargamos de que estos servicios resulten claramente representados y accesibles, aportando facilidad a la hora de ser utilizados, ya que la API proporcionará ejemplos de llamadas, así como ejemplos de resultados en ambos formatos, XML yJSON. La API dispondrá de un diseño sencillo, con una serie de descripciones clave para el entendimiento de la funcionalidad del servicio, así como los parámetros que necesita y lo que se espera obtener de él.

- ***Android:*** El ejemplo Andorid se encargará de mostrar, en una aplicación en Android, un ejemplo sencillo y conciso de como se les puede dar uso a nuestros servicios, a modo de posible idea para potenciales interesados. De este modo, al tener un ejemplo más real, más final y práctico, resulta mucho más tentador seguir tirando por ese campo, ves ya claramente con este ejemplo como se le puede sacar partido a los servicios, y como estos funcionan correctamente pudiendo probar las distintas funcionalidades en la aplicación.

Capítulo 2

Introduction

2.1. Motivation

For some time man has sought ways of communication beyond words simply spoken, either to communicate directly with others, to convey what they think or to publicize a truth that they believe that could be affecting to others. Today it is increasingly possible because communications reach more and more people each day, and its impact is growing. However, a message is not always accessible to everyone, even when they can physically access it.

There are many limitations that do not concern the field of communications when a message can reach an audience. The most obvious may be the language in which it is transmitted, and that is why it is the most treated and the first to be carried out, being possibly also the easiest to treat. However there is one which is not receiving enogh attention and it is equally limitating, the ability to understand a language for people who share it. This means that within a community with a common language, there are people who are not able to understand complex words that for other people may be understandable. This scenario can occur in different situations, people with learning disabilities, with a learning capability limitation. But that does not end there. Often it is not about a learning capability limitation of the person itself but a problem with the text to read, because it can have a complexity of terms entirely specific in which an outsider from that specific languaje could get lost reading it. This situation can occur with legal, scientific texts, technological concepts ...

Being aware of these needs, in the Faculty of Computer Science over the years, they have been doing different projects of great social utility such as simplifying texts to facilitate understanding, translating texts into pictograms for those who are not able to read, recognition of the complexity of words within a text... All these projects have been created over the years in different languages, formats and styles .

Our project arises from the need to give a social utility to these projects, making them more unified and reacheable by making a web API and an Android aplication as an example of how can you use the resources in the API.

2.2. Goals

Our goal, as we have seen in 2.1, is to get this communication improved services unified, complete and accessible to anyone who wants to use them. In addition we will provide examples of possible uses that can serve as ideas for potential developers interested in them.

Therefore, in this project we will be checking the previous projects, correcting mistakes that could have and improving functionality. Also we will unify all these projects, putting them all in the same language as well as on a single server and keeping consistency between the different services, both in the form of input and the output, having two formats for this one , XML and JSON. In addition we will put eachone of them in a common API for greater accessibility facing users who may be interested in using them, as well as examples of the use to make them as clear as possible.

It should be noted that this project is part of another that is called: *IDiLyCo: inclusión digital, lenguaje natural y comunicación. TIN2015-66655-R (MINECO/FEDER)*, which is funded by the Ministry of Economy and Competitiveness and the European Regional Development Fund (ERDF) within the National Plan of R + D + I.

The goal of the project is to ease the communication and the access to digital information to groups which, by their functional diversity may find problems when using language to communicate or access to such information.

IDiLyCo seeks to develop technology solutions that allow an easy way of making a digital inclusion of these people.

It offers to the designers or programmers of new tools to integrate into their services implementations developed in this project, thus saving enormous costs in research and development of these technologies accessible.

2.3. Project management

Arguably, the project is divided into three main parts: REST services, the API and Android example of how to use the services.

- ***Services:*** In the part of services, we will take care of checking the previous services, making them consistent with each, both in language in

which they are implemented as in format. Furthermore we will expand the server with new services to make them as complete as possible. To do this we must parse some pages such as web of synonyms and antonyms, files of the words most used to deduce the complexity of a word, webs of definitions and translations into English... All this will have as a result a server with a functional and accessible services that will ease the achievement of our objectives.

- ***API:*** the API's part takes care that these services are clearly represented and accessible, providing ease when being used, the API will provide examples of calls, and examples of results in both formats, XML yJSON. API have a simple design with a number of key descriptions for understanding the functionality of the service and the parameters required and what is expected from them.

- ***Android:*** The Andorid example will show, in an application in Android, a simple and concise way of how our services can be used, as a possible idea to potential users. Thus, by having a more real and practical example, is much more tempting to keep evolving this field. With this example is clearly shown how the services can be used, and also you can test the different functionalities.

Capítulo 3

Estado de la Cuestión

3.1. API: Definiciones y tipos

Se puede entender API (*Application Programming Interface*) como las funcionalidades que aporta un cierto servicio software facilitando que pueda ser utilizado por otro software para mejorar sus resultados. Normalmente no es un resultado en si mismo, sino que sirve de enlace entre un software ya creado y otro al que este le puede resultar útil, lo que se conocería como una interacción "software-to-software".

En este contexto para realizar una API práctica esta tendrá que aportar un concepto claro de las funcionalidades que ofrece, quedando bien definidas, para que el que quiera usarlas posteriormente tenga claro cómo funcionan y cómo puede acceder a ellas. Los beneficios de esto son que, en vez de tener que partir de cero y crear un código necesario pero ya existente, las APIs proporcionan la posibilidad de obtener los mismos resultados sin tener que perder el tiempo en reimplementaciones.

Teniendo en cuenta lo anteriormente mencionado nos podemos encontrar diferentes tipos de APIs en función de a quien están distribuidas, (IBM (2016)):

- **Internas o privadas:** cuando son utilizadas solo dentro de una misma compañía o asociación, es decir, cuando solo los que generan el código de utilidad se pueden beneficiar del mismo. Esto, evidentemente, no quiere decir que lo tenga que hacer todo la misma persona, sino que no puede exportarse fuera de los limites de la compañía, mejorando así la productividad de la misma pero no la de otros.
- **Externas o públicas:** cuando se ponen al servicio de todo aquel que las quiera utilizar. Esto proporciona las mismas ventajas que una API privada en cuanto a mejora de la productividad y a no tener que re-programar código ya existente pero a un nivel global.

- **Partner:** esos tipos de APIs suponen un paso intermedio entre las privadas y las públicas, ya que, en vez de tratarse de una sola compañía, son utilizadas por varias compañías que colaboran entre ellas. De este modo si una de las compañías genera código que le pueda resultar útil a la otra esta segunda lo podrá utilizar sin necesidad de tener que empezarlo de cero, y viceversa. De este modo se mejora más la productividad sin necesidad de tener que aportar facilidades a la competencia.

Pero esta no es la única clasificación en la que se pueden englobar las APIS, ni en la que nos vamos a centrar más. A continuación enumeraremos la clasificación en función a qué están orientadas, dando una breve definición de cada una de ellas (Maddox (2016)):

- **APIs de servicios Web:** estas van a ser las que más relevancia tengan durante este proyecto ya que van a ser las que nosotras utilicemos. Esta proporciona acceso a su servicio mediante una dirección web e implica comunicación en una red. Normalmente utilizan servicios web estándar como REST, SOAP, XML-RPC y JSON-RPC, algunos de los cuales nos centraremos más adelante en este mismo capítulo.
- **APIs basadas en librerías:** este tipo de APIs tienen gran importancia cuando programamos en determinado lenguaje ya que, normalmente, hacemos uso de sus librerías constantemente, facilitando así la tarea del programador. Evidentemente las librerías no solo pueden ser proporcionadas por la plataforma software en la que se trabaja sino que pueden ser generadas por el programador para su propio beneficio y el de otros.
- **APIS de sistemas operativos:** este tipo de APIs te permiten saber como están estructuradas las funcionalidades del Sistema Operativo.

Estos son algunos de los ejemplos de tipos de API que puede haber. Hay muchos más posibles tipos, pero estos podrían considerarse los más importantes. A continuación nos centraremos en detallar la primera ya que es la que resulta de mayor importancia para nuestro proyecto puesto que será la que desarrollaremos.

3.1.1. APIs de Servicio Web

Como veíamos en la definición anterior, un servicio web es un tipo de API que proporciona acceso a un servicio mediante una dirección URL en una red. Es importante que la información proporcionada por una API de servicio web se presente en un formato que sea comprensible para otras aplicaciones, ya que si no se perdería toda la funcionalidad de la API en si misma. Para ello

habrá que tener estandarizado una serie de arquitecturas software válidas a utilizar en este determinado tipo de APIs.

Para entender lo anteriormente mencionado hay que tener claros una serie de conceptos que son fundamentales para la comprensión de las APIs de servicios Web. Estos conceptos quedan brevemente definidos a continuación, centrándonos especialmente en REST y SOAP en el apartado 3.2.

- **SOAP *(Simple Object Access Protocol)*:** es un protocolo estándar de comunicación entre distintos componentes software. Está basado en XML para la transmisión de sus mensajes, en los cuales quedan añadidos los parámetros de la petición. Aporta una gran ventaja y es que no requiere que los componentes software de los extremos de la comunicación corran en el mismo sistema operativo, ni siquiera que estén en el miso lenguaje de programación. Está normalmente asociado con HTTP como protocolo de transporte, aunque puede soportar otros. (Wikipedia (Marzo, 2016b)).

- **XML *(eXtensible Markup Language)*:** es un lenguaje de marcado, como su nombre indica, diseñado para describir los datos transportados. Este lenguaje es autodresciptivo y el concepto de extensible hace referencia a que puedes crearte tus propios elementos (*tags*) y bloques de elementos. (Wikipedia (Abril, 2016)).

- **XML-RPC *(Remote Procedure Calls)*:** como su nombre indica es un protocolo de llamada a procedimientos remotos. Utiliza HTTP como medio de transporte y XML como codificación y descripción de datos. Es un protocolo más antiguo que SOAP y resulta sencillo de utilizar y rápido en su ejecución. (Wikipedia (Mayo, 2016)).

- **JSON-RPC**: es similar a XML-RPC pero usa JSON en lugar de XML para transferencia de datos. (Wikipedia (Junio, 2016a)).

- **REST *(Representational State Transfer)*:** se encarga de representar la transferencia de datos. Usa llamadas HTTP siendo así un simple mecanismo de solicitud y respuesta. (Wikipedia (Marzo, 2016a)).

- **WSDL *(Web Services Description Language)*:** es un lenguaje de descripción de servicios web escrito en XML. Tanto mensajes como operaciones se describen de forma abstracta. Es recomendada por la W3C *World Wide Web Consortium*. (Wikipedia (Agosto, 2015)).

Así mismo un servicio Web deberá tener un protocolo de transporte estándar, por lo que casi siempre utiliza HTTP y, por supuesto, realiza las comunicaciones en una red, generalmente a través de la Web. Deberá, también, presentar una descripción clara y detallada de la funcionalidad y el

modo de empleo del servicio que ofrece, para facilitar así su comprensión y utilización. En la figura 3.1 se muestra un esquema explicativo de los servicios web.

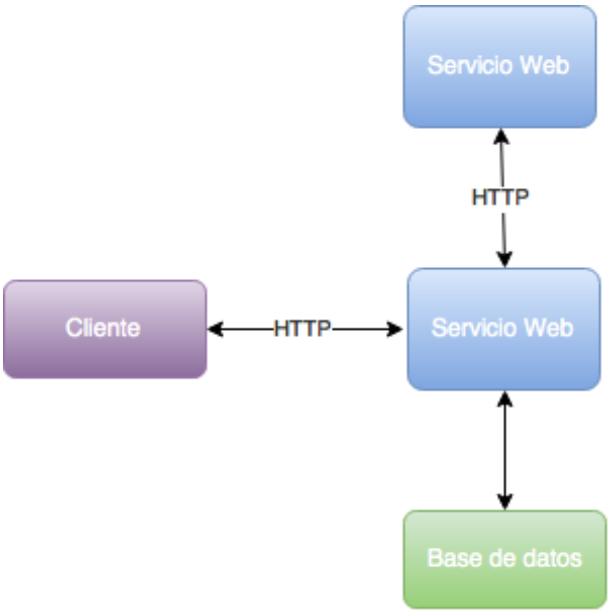


Figura 3.1: Esquema de Servicios Web

3.1.2. API: Ejemplos

En este apartado nos centraremos en las APIs más populares existentes en el mercado, describiéndolas con cierto detalle, aunque será en la sección 3.4 donde nos pararemos más detenidamente a analizar las que nos han servido más a nosotras y donde indicaremos los motivos que nos han llevado a descartar o asimilar en nuestra API el estilo y la funcionalidad de cada una de ellas.

- **Google Maps**: Es un servicio web ofrecido por Google con mapas desplazables e imágenes por satélite que permiten la localización geográfica de lugares, establecimientos, dispositivos, etc. En la página de Google Maps developers[1] tienes disponibles los diferentes tipos de API, así como las plataformas para las que está desarrollado. Así mismo te indica las limitaciones que presenta su uso en función de si lo realizas de manera estándar gratuita o premium de pago. Dispone de gran variedad de APIs de servicios web tales como conversión de direcciones en coordenadas, tiempo y distancia entre lugares, registro de zonas

[1] `https://developers.google.com/maps/documentation/javascript/?hl=es`

horarias, indicaciones entre direcciones... Además ofrece una serie de tutoriales para llevar a cabo distintas utilidades acordes con las funcionalidades ofrecidas por las APIs. Todo esto ha facilitado que estas APIs se puedan incrustar en distintos sitios web o dispositivos, pudiendo manipularlos para adaptarlos a las funcionalidades requeridas por tu servicio.

- **Amazon S3 (Simple Storage Service)**[2]: es un servicio web de almacenamiento en la nube por el que pagas por el espacio consumido y las peticiones realizadas. Resulta interesante que cuando intentas acceder a la API en español este te lo muestra en inglés de todas formas. Tiene el soporte sobre REST, teniendo algunos soportes limitados en SOAP también, pero no con las nuevas funcionalidades de Amazon S3. La API muestra una serie de descripciones de las diferentes operaciones que se pueden realizar sobre los distintos objetos. Dispone, al igual que Google, de numerosos ejemplos y descripciones de las distintas funcionalidades que se pueden realizar. También describe distintos códigos de error.

- **Facebook**[3]: es una red social, una de las más populares y extendidas en la actualidad, que permite compartir publicaciones, fotos, información, etc. Este sitio web tiene muchísimas posibilidades de negocio debido a que tiene un gran número de usuarios interaccionando con él. Esto hace que exista gran interés en desarrollo de funcionalidades que puedan, de algún modo, aprovechar este amplio publico. En este aspecto Facebook tiene una serie de servicios muy útiles para la captación de clientes así como para analizar el rendimiento de tu aplicación. Además dispone de herramientas de marketing que pueden facilitar la labor de extender el público interesado. Disponen de una documentación bien detallada de todos los servicios que ofrecen, así como la forma de acceder a ellos y ejemplos de utilización que resultan muy concisos.

- **Twitter**[4]: es igualmente una red social que permite compartir breves contenidos de texto con tus amigos o con el resto de usuarios. Al igual que Facebook, dispone de grandes posibilidades porque maneja gran número de usuarios. Tiene un servicio que permite a empresas añadir publicidad en Twitter lo que proporciona un amplio abanico de probabilidades ya que, al tener tantos usuarios, el impacto de esta publicidad es muy alto. Su servicio de publicidad lo tienen dividido en 3 niveles: Developer, Basic y Standard, teniendo distintos niveles de exposición para cada uno. También tienen servicios para dar acceso a la cadena global de datos en los Tweets, lo cual es muy conveniente para hacer

[2] `https://aws.amazon.com/es/?nc2=h_lg`
[3] `https://developers.facebook.com/products/`
[4] `https://dev.twitter.com/overview/documentation`

estrategias de marketing. Como en las anteriores, Twitter dispone, además de lo ya mencionado, de un sistema de ejemplos de código con sus correspondientes resultados, facilitando así el manejo de las APIs

3.2. REST: Primeros conceptos

Como veíamos en 3.1.1 REST (***RE**presentational **S**tate **T**ransfer*) es una arquitectura software que se encarga de representar la transferencia de datos. Cabe destacar que se apoya en el protocolo HTTP para la transferencia de información entre máquinas. Podría decirse que tiene un estilo derivado de otras arquitecturas basadas en redes, lo cual lleva a considerársela más bien como un conjunto de arquitecturas. Es principalmente usada para la construcción de servicios web, los cuales, estando basados en REST, reciben el nombre de "servicios RESTful", en los cuales nos centraremos más adelante en esta sección.

REST se encarga, básicamente, de tratar objetos como recursos que pueden ser creados y destruidos, teniendo para ello cuatro métodos básicos: put, delete, get y post, los cuales se encargan respectivamente de crear, leer, actualizar y borrar y nos centraremos más en ellos en el apartado 3.2.2. Está basado en el principio cliente-servidor, teniendo que tener las peticiones del cliente su estado totalmente representado. Supone una separación entre cliente y servidor, teniendo que encargarse cada parte de sus componentes. Es así una arquitectura multiplataforma lo cual es uno de los motivos que la hacen ideal para ser el estilo de arquitectura software usado en WWW (*World Wide Web*).

En la figura 3.2 se muestra un esquema de la estructura básica de una arquitectura REST.

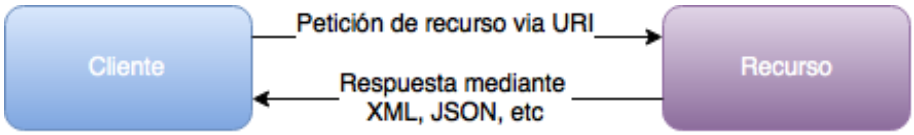


Figura 3.2: Estructura básica de arquitectura REST

Otro punto importante de la arquitectura REST es que es *stateless*, sin estado, lo cual no significa que no se sepa el estado del cliente si no que no es el servidor el que se encarga de almacenar este estado. Es responsabilidad del cliente pasar esta información, pudiendo así el servidor alojar distintos clientes en cualquier momento, sin estar vinculado a ninguno concretamente.

Todo lo anteriormente descrito nos lleva a ver que, dadas las numerosas ventajas que aporta REST, los servicios RESTful estén a la orden del día.

Un servicio RESTful es básicamente un servicio web que está basado en la arquitectura REST para su funcionamiento. Esta centrado en recursos y el acceso a los mismos. Es fundamental que sea sencillo y rápido, por ello tienen una identificación de los mencionados recursos mediante URIs (*Uniform Resource Identifiers*) para un direccionamiento global. Además disponen de uniformidad de interfaces, lo cual se traduce en las cuatro métodos anteriormente vistos de REST (put, delete, get y post).

3.2.1. Principios REST

Como hemos visto en la sección 3.2, REST es una arquitectura multiplataforma, sin estado, basada en HTTP y que resulta sencilla de manejar. En este apartado nos centraremos más en definir sus principios y dar una idea clara y concisa de la importancia que tienen cada uno de ellos.

Los principios básicos de las arquitecturas REST son (Burke (Noviembre, 2009) y Wikipedia (Junio, 2016b)):

- **Uniformidad de interfaces:** esta uniformidad se consigue a través de lo que HTTP es conocido como verbos, que son los métodos REST ya mencionados (get, post, delete, put). Existen otros dos menos utilizados que serían head y options, pero en esto nos centraremos más en la sección 3.2.2. Esto facilita la interacción estándar de un cliente con sus recursos y con cualquier servidor HTTP sin configuraciones especiales.

  En la figura 3.3 se muestra un esquema de la uniformidad.

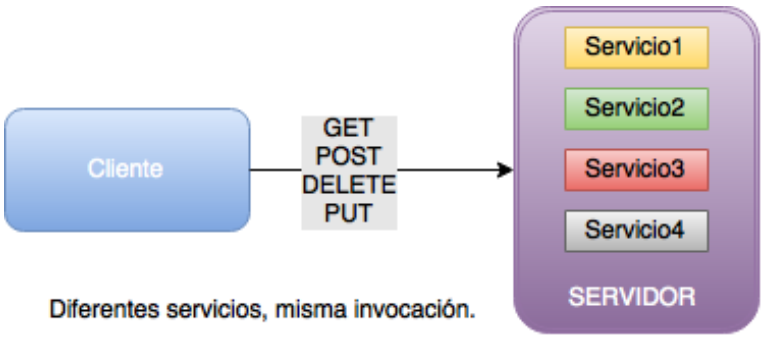


Figura 3.3: Estructura de Servicios Uniformes

- **Sin estado (stateless):** los servicios REST, como ya hemos visto antes, son servicios que no registran el estado del cliente en el servidor, sino que es el mismo cliente el que transporta esa información consigo mismo, y están representados con las URIs. Esto, junto con la separación de interfaces entre cliente y servidor, hace que los servidores sean más sencillos y escalables, lo cual permite que puedan manejar un aumento considerable en el número de recursos sin que se vea afectado

el rendimiento. Esto es de vital importancia en la Web debido a su constante crecimiento.

En la figura 3.4 se muestra un esquema del principio Stateless.

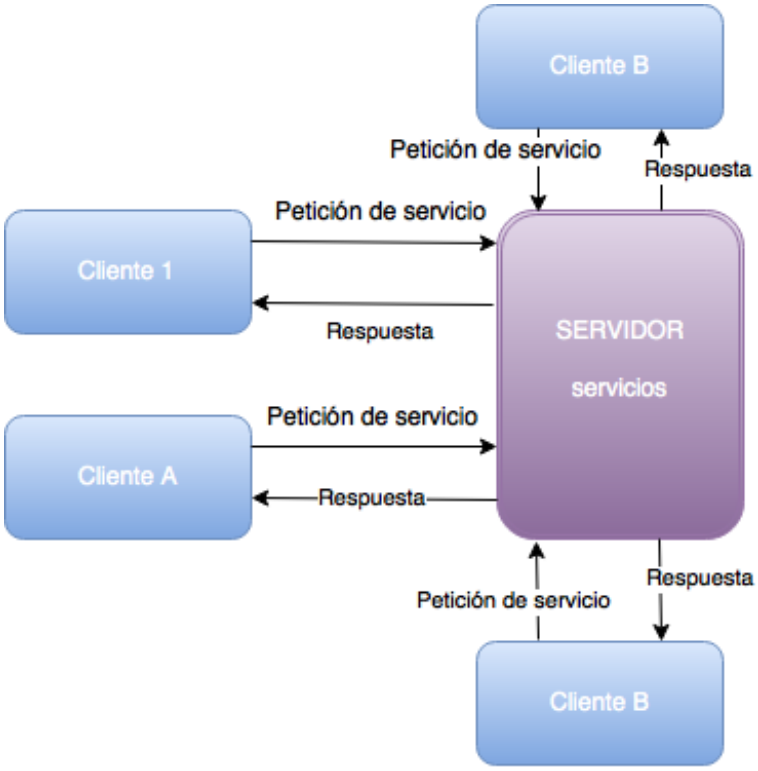


Figura 3.4: Estructura sin estado

- **Identificación de recursos:** este supone un punto muy importante para el correcto funcionamiento de los servicios REST ya que hace posible hacer referencia a los distintos recursos de forma única. Esto se consigue a través de las URIs, las cuales proporcionan capacidad de direccionamiento. Esto quiere decir que a través de un URI se proporciona un identificador único a un determinado recurso al cual se podrá acceder en cualquier momento a través de este.

En la figura 3.5 se puede apreciar la estructura general de un servicio REST.

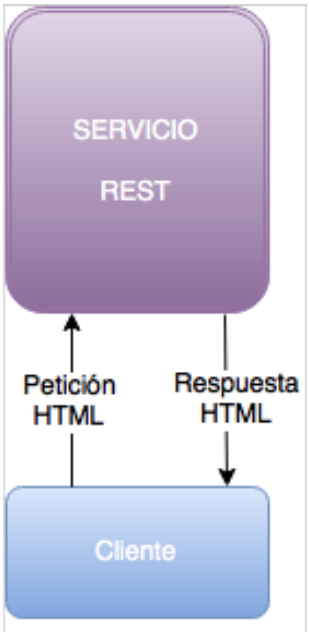


Figura 3.5: Estructura general de servicio REST

3.2.2. Métodos en REST

Una vez explicados los conceptos generales de REST y sus principios vamos a pasar a definir las cuatro métodos básicos en un sistema REST[5]. Estos coinciden con los verbos de HTTP y son: **POST, GET, PUT y DELETE** que hacen referencia a las operaciones conocidas como CRUD, respectivamente, create, read, update y delete. A continuación pasamos a hablar de ellos con más detalle.

- **POST:** es el método correspondiente a la operación `create`, y se encarga de crear un nuevo recurso. Estos nuevos recursos creados son subordinados, se crean a partir de un recurso padre. Si la creación tiene éxito se devolverá el estado 201 correspondiente a "*Created*" y en la cabecera de localización el link del nuevo recurso que contendrá la ID del mismo. Se pueden producir errores en la creación de recursos, puede ser que el recuso ya exista, en cuyo caso se devolvería el estado 409 correspondiente a "*Conflict*" o que no se encuentre el recurso en cuyo caso el estado es 404, "*Not Found*". El método POST no es idempoten-

[5]http://www.restapitutorial.com/lessons/httpmethods.html

te, lo cual quiere decir que si aplicas varias veces la misma operación al mismo elemento no vas a obtener siempre el mismo resultado.

- **GET:** es el método correspondiente a la operación `read`, y se encarga de leer un recurso existente. Cuando se lee un recurso este no se modifica por lo que GET es considerado un método seguro. Es, además, idempotente, ya que siempre que realices GET sobre un mismo recurso el resultado será el mismo. Si la lectura del recurso se produce sin problemas se devuelve un estado 200 correspondiente con "*Ok*" y una representación del mismo en XML o JSON. Si por el contrario se produce algún error este puede ser debido a recurso no encontrado "*Not Found*" devolviendo un estado 404 o a una incorrecta petición del recurso "*Bad Request*" con un estado 400. Es importante tener en cuenta que se devuelve el estado del recurso, y que no se modifica, o debe modificar el estado del servidor.

- **PUT:** es el método correspondiente a la operación `update`, y se encarga de actualizar un recurso. Puede igualmente ser usado para la creación de un recurso, llevando a la duda de cuando usar PUT y cuando POST. PUT se usa para crear recursos cuando tienes toda la información del recurso que te quieres crear, por lo que no tendrías que crearte el recurso partiendo de un recurso padre como ocurre en POST. Al tratarse de un método que modifica un recurso se considera como no seguro. Sin embargo, a diferencia de POST, PUT si es un método idempotente, ya que siempre que aplicas PUT de la misma forma sobre un mismo recurso obtienes los mismos resultados. Si la actualización del recurso se ha realizado con éxito este devolverá un estado 200 "*Ok*" o 204 si el recurso no tiene ningún contenido "No Content". Si por el contrario se produce un error devolverá 404.

- **DELETE:** es el método correspondiente a la operación `delete` y se encarga de borrar un determinado recurso. Se necesitará saber el identificador del recurso para poder realizar esta operación, ya que ha de ser realizada sobre el recurso concreto. Si la operación se realiza con éxito el recurso devolverá un estado 200 "*OK*" o 204 "*No Content*". La operación de DELETE es idempotente, ya que realizar la misma llamada de DELETE sobre el mismo recurso dará siempre el mismo resultado, la eliminación del mismo, que el recurso ya no exista. Existen objeciones al respecto de la idempotencia del método DELETE ya que, en ocasiones, si haces varias veces sobre el mismo recurso la operación de DELETE te puede devolver un estado de error 404 "*Not Found*", pero en cualquier caso el resultado sigue siendo que el recurso no exista. DELETE es a su vez un método no seguro, ya que modifica el valor de un recurso, concretamente eliminándolo.

Además de los métodos anteriormente mencionados, que resultan los más comúnmente utilizados cuando se trata con servicios REST, existen también métodos como: HEAD, OPTIONS y PATCH, siendo HEAD y OPTIONS métodos seguros e idempotentes solo HEAD. Debido a que son raramente utilizados no nos centraremos en dar más información sobre ellos.

En la imagen 3.6 podemos ver el diseño de una API REST.

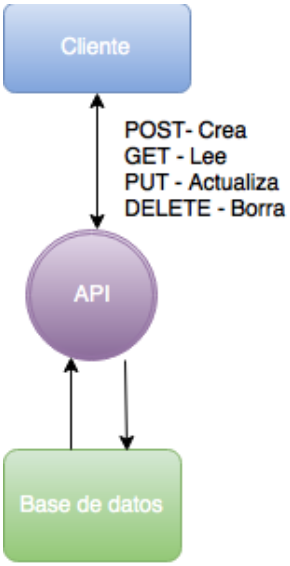


Figura 3.6: Diseño de API REST

En la imagen 3.7 podemos ver un ilustración de las diferencias entre SOAP y REST.

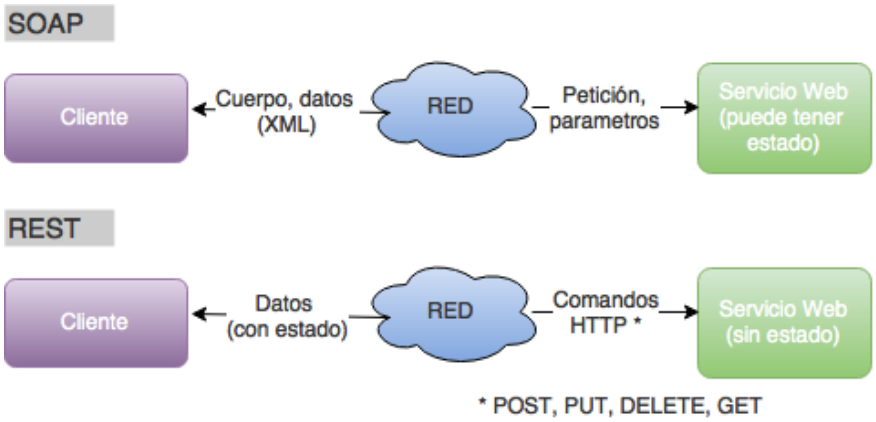


Figura 3.7: REST vs SOAP

3.3. Servicios de Accesibilidad

A continuación nos centraremos en el Estado de la Cuestión con respecto a los servicios de accesibilidad relacionados con los que vamos a realizar. Para ello haremos primero un repaso muy superficial de los servicios de accesibilidad que se encuentran hoy en día en el mercado, centrándonos posteriormente en aquellos orientados a facilitar palabras o textos.

Lo primero que hemos de tener en cuenta en este apartado es el concepto de accesibilidad. Para ello citamos las definiciones que nos aporta la *Real Academia Española* en cuanto a accesibilidad y accesible:

> **Accesibilidad:**
>
> - *1. f. Cualidad de accesible.*
>
> **Accesible:**
>
> - *1. adj. Que tiene acceso.*
> - *2. adj. De fácil acceso o trato.*
> - *3. adj. De fácil comprensión, inteligible.*
>
> Real Academia Española

Por otro lado tenemos la definición de la Wikipedia, más concisa y clara a la hora de entender el concepto de accesibilidad que nos atañe:

> *La accesibilidad o accesibilidad universal es el grado en el que todas las personas pueden utilizar un objeto, visitar un lugar o acceder a un servicio, independientemente de sus capacidades técnicas, cognitivas o físicas. Es indispensable e imprescindible, ya que se trata de una condición necesaria para la participación de todas las personas independientemente de las posibles limitaciones funcionales que puedan tener.*
>
> Wikipedia

Teniendo en cuenta esta última definición el concepto de accesibilidad que queremos tratar aquí queda mucho más claro, y es el objetivo de que todas las personas puedan utilizar un objeto, independientemente de sus capacidades. Para ello hay que tener en cuenta, a la hora de tratar dicho objeto, las limitaciones que puede presentar el acceso al mismo para personas con distintos tipos de limitaciones. Nos limitaremos a tratar las funcionalidades que se encargan de mejorar la accesibilidad en internet.

Hoy en día, dada la relevancia y el extendido uso de Internet y sus numerosas aplicaciones, la importancia de los servicios de accesibilidad es cada vez

mayor, permitiendo así que todas las personas, independientemente de sus capacidades, puedan acceder a los servicios que se ofrecen. En este ámbito te puedes encontrar servicios que amplían textos e imágenes para gente con discapacidad visual o que transmiten de forma auditiva los mismos, transcripciones de audios para gente con discapacidades auditivas, aplicaciones que permiten localizar las zonas habilitadas para gente con movilidad reducida, aplicaciones para comunicación mediante pictogramas, etc. A continuación pasaremos a ver algunos ejemplos de servicios de accesibilidad.

- **Google TalkBack:** es un servicio de accesibilidad que facilita a las personas ciegas o con visión limitada a interactuar con sus dispositivos. Esta incorpora respuestas útiles vía comentarios por voz o vibraciones del dispositivo, que le indican al usuario en que pantalla o proceso se encuentra en cada momento. Viene normalmente preinstalado en los dispositivos Android y se puede activar de manera sencilla. Este dispone de numerosos ajustes de voz, de vibración, de exploración táctil, etc. que le dan al usuario un feddback de las acciones que está realizando. (Google (Abril, 2016))

- **Google BrailleBack:** es un servicio de Google que facilita a personas invidentes la utilización de los dispositivos de braile. Está combinado con el ya visto TalkBack para ofrecer un servicio completo. Este conecta tu dispositivo vía Bluetooth con una pantalla braile, transmitiendo el contenido de tu dispositivo a la misma. También permite la introducción de texto a través de un teclado braile, también conectado a tu dispositivo. (Google (Diciembre, 2015))

- **VoiceOver:** es un servicio de accesibilidad similar al ***Google TalkBack*** pero para dispositivos de Apple. Te hace una descripción auditiva de lo que tienes en la pantalla en ese momento, pudiendo configurar la velocidad del habla, el tono, etc. Además controlas el dispositivo mediante una serie de gestos adaptados para personas con capacidad visual reducida o nula. Además incorpora un teclado braile configurable sin necesidad de teclados físicos. (Apple (2016))

- **Webvisual**[6]**:** es una web catalana enfocada a la transmisión de noticias de actualidad de forma accesible para sordos. Tiene apariencia similar a una web de noticias cualquiera pero está totalmente habilitada para gente con discapacidad auditiva. (FESOCA (2016)) .

- **Pictotraductor**[7]**:** es una página web que convierte frases a pictogramas facilitando así su comprensión a personas con dificultades para

[6] `http://www.webvisual.tv`
[7] `http://www.pictotraductor.com`

entender textos escritos. Es una herramienta totalmente accesible, además de tener un diseño muy sencillo facilitando así su utilización. En 3.8 mostramos un ejemplo de frase y resultado obtenido. (Promedia (2013)).

Figura 3.8: Ejemplo de resultado obtenido con pictotraductor

- **Simplext**[8]**:** es un sistema automático de simplificación de textos, que favorece el acceso a las tecnologías de personas con capacidades cognitivas limitadas o cualquier otra persona con dificultad para leer y escribir en el idioma, simplificando los contenidos textuales. Sus finalidades son usar un lenguaje simple y directo, sin sobrecarga de frases, evitar tecnicismos o palabras complejas y estructurando el texto de manera clara y concisa.
- **DysWebxia**[9]**:** es un proyecto enfocado a las personas con dislexia que facilita el acceso de las mismas a las tecnologías modernas. El objetivo es que la información textual en la web sea más accesible para las personas con dislexia, no solo destacando palabras clave en negrita, sino sustituyendo palabras por sinónimos más sencillos o esquematizando partes del texto, todo ello de forma automática.

Se pueden considerar servicios de accesibilidad, igualmente, a las funcionalidades de traducción de idiomas, como las traducciones automáticas de Google para páginas web, ya que facilitan el acceso a gente que desconoce el idioma origen de determinada página.

En el contexto de los servicios de accesibilidad, en años anteriores en la Facultad de Informática se han realizado distintos proyectos centrados en la

[8] `http://www.simplext.es`
[9] `http://www.text4all.net/dyswebxia.html`

comprensión y el facilitado de textos, siendo uno de ellos el ya mencionado Pictotraductor. A continuación pasaremos a hablar brevemente de cada uno de ellos y de sus funcionalidades principales.

- **XText:** en este proyecto desarrollaron un servicio web en el cual, dado un texto, se podía enriquecer el mismo con subrayado, definiciones de algunas palabras, sinónimos y antónimos, así como el análisis morfológico de algunas de ellas. Esto se volcó en una aplicación web multiplataforma que hacía uso de dichas utilidades. Esta consta de un editor de texto que, además de aportar funcionalidades comunes a los editores de texto, añadía las funcionalidades desarrolladas en su proyecto, tales como semántica y formato del mismo. Está dividida en dos pasos de elementos a seleccionar: tipos de elementos en los que centrarse (sustantivos, adjetivos, números, etc.) y las acciones que se pueden realizar (análisis morfológico, subrayar, traducción, etc.). En la imagen 3.9 se puede ver un ejemplo de análisis morfológico aplicado a sustantivos de una frase. A continuación citamos los elementos y acciones contenidos en el proyecto. (Linares y Flores (2013 -2014))
  - Tipo de palabra sobre la que se aplica la acción en la frase
    - Nombres
    - Verbos
    - Adjetivos
    - Pronombres
    - Determinantes
    - Adverbios
    - Preposiciones
    - Interjecciones
    - Palabras difíciles
    - Números
    - Fecha y hora
    - Conjunciones
  - Acciones sobre las palabras:
    - Subrayar
    - Análisis morfológico
    - Sinónimos
    - Antónimos
    - Definiciones
    - Traducción a inglés

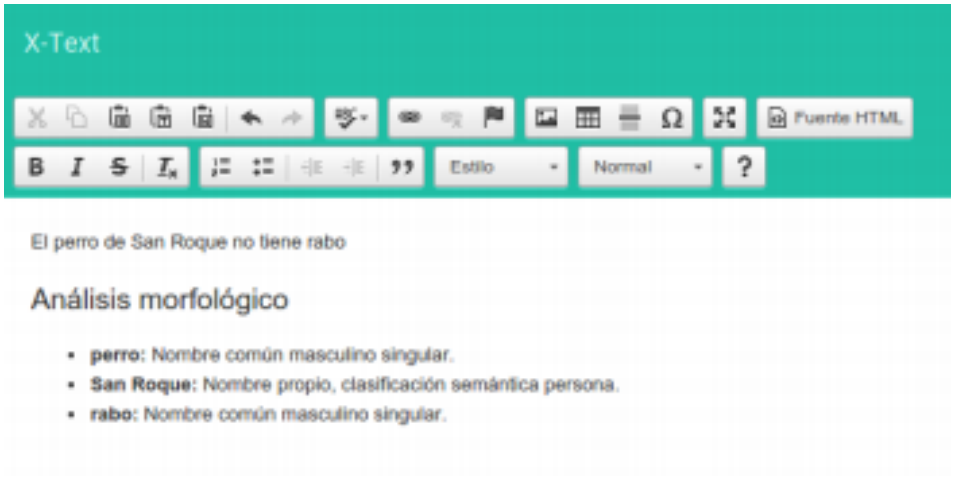


Figura 3.9: Ejemplo de resultado en aplicación web de XText.

- **Conversor de textos en pictogramas:** En este proyecto se proporciona una aplicación para Android llamada "AraTraductor" para facilitar la comunicación entre personas que no tienen dificultad para comunicarse de forma escrita, con otras que sí que tienen problemas de comunicación y no pueden comprender correctamente el lenguaje escrito. La aplicación se encargada de, dado un texto escrito, convertirlo a texto basado en pictogramas. En la imagen 3.10 podemos ver un ejemplo de frase y su correspondiente traducción a pictograma. (Alonso et al. (2013 -2014))

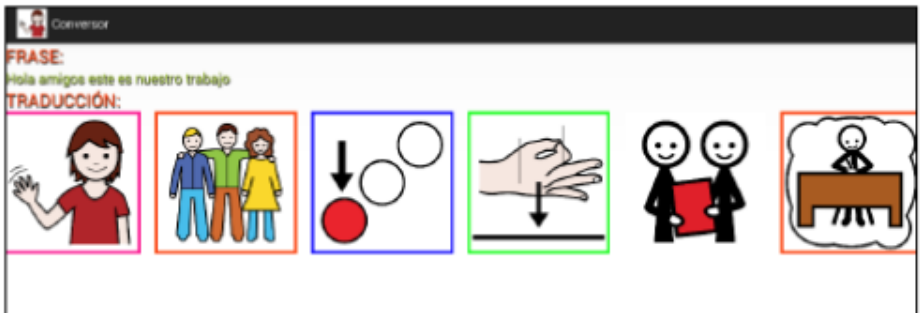


Figura 3.10: Ejemplo de resultado en AraTraductor.

- **Editor predictivo de mensajes en pictogramas:** La funcionalidad es similar a la de Pictotraductor. El objetivo es, dado un texto, convertirlo a pictogramas. En este caso, además, se distinguen los pictogramas en colores según la categoría a la que pertenezcan en grupos de palabras (verbos, sustantivos, expresiones, nombres propios, etc.).

Así mismo este proyecto tiene implementada una aplicación Android para poner en uso los servicios implementados. Además usan un algoritmo predictivo para la composición de frases de acuerdo al número de veces que ha sido seleccionada una imagen así como por qué tipo de imagen ha sido sucedida. Además dispone de la posibilidad de guardar y compartir frases generadas. En la imagen 3.11 podemos ver un resultado obtenido marcando en rojo además los iconos de compartir y guardar. (Martín y Calleja (2013 -2014)).

Figura 3.11: Resultado del editor predictivo de mensajes en pictogramas.

- **Apoyo a la simplificación de textos sobre navegadores web (proyecto navega fácil):** en este proyecto se busca facilitar la comprensión de los textos en la web. Se trata de tener opciones de buscar definiciones de palabras, sinónimos y antónimos, etc. pudiendo acceder a otros servicios de búsqueda a su vez. Es un proyecto desarrollado para facilitar la navegación en Internet a aquellos usuarios con diferentes niveles de discapacidad que puedan tener dificultades a la hora de leer o interpretar textos. Fue creado para simplificar a dichos usuarios la navegación en Internet, ofreciéndoles de una manera sencilla un conjunto de herramientas que les ayudará a comprender los textos cuando estén navegando. Este proyecto fue dividido en dos partes: servicios web y una página web encargada de aplicar dichos servicios. Los servicios que llevan a cabo son:

  - Lematización: encargado de realizar la lematización de una palabra dada, lo cual quiere decir que dada una palabra cualquiera se devolvería el lema de la misma. Se entiende como lema la forma de representar una palabra y sus derivadas, en otras palabras, la palabras que encontrarías en el diccionario al hacer una búsqueda.

Por ejemplo: si tenemos las palabras corríamos, corren, corría y correrán, todas estas palabras tienen como lema correr, que es lo que devolvería este servicio. No es solo aplicable a verbos, "casa" supondría el lema de la palabra "casas".

- Definición: encargado de devolver la definición de una palabra dada según el diccionario de la Real Academia Española (RAE).
- Sinónimos y antónimos: encargado de devolver los sinónimos y antónimos de una palabra dada según la web de ElPaís: `www.elpais.com`
- Sinónimos (origen latinoamericano): encargado de devolver los sinónimos de una palabra dada según la web `www.sinonimos.org`.
- Difíciles: dado un texto te devuelve las palabras en él que resultan difíciles, usando para ello las 10.000 palabras más usadas del castellano según la RAE, considerando estas fáciles y el resto como difíciles.

La página web recibe el nombre de NavegaFacil y hacen una puesta en práctica de los servicios arriba mencionados (que se puede encontrar en: `http://hypatia.fdi.ucm.es`) aplicando estos servicios a páginas web por las que quiera navegar el usuario, pudiendo aplicarle los servicios mencionados facilitando así la comprensión de la misma. En la imagen 3.12 podemos ver un pequeño ejemplo de su funcionalidad. (González y Ortiz (2013 -2014))

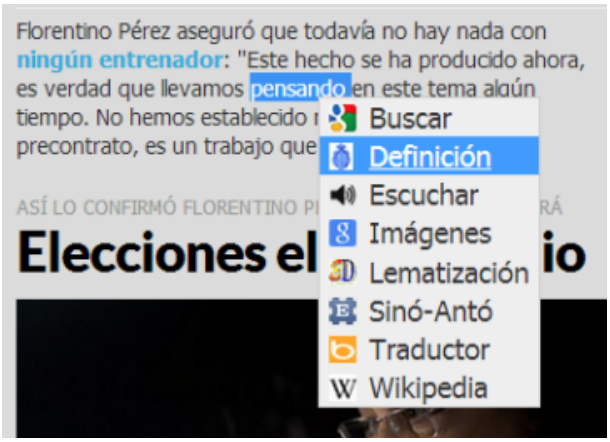


Figura 3.12: Ejemplo de funcionalidad de la web NavegaFacil.

3.4. Análisis de APIs disponibles

Para llevar a cabo nuestro proyecto hemos buscado APIs ya existentes para tomar ideas de las mismas, tanto en diseño y organización como en contenidos. Para ello hemos analizado las diferentes estructuras de las principales APIs del momento, captando lo que más nos gusta de cada una y descartando elementos que no nos terminaban de convencer.

Las APIs en las que más nos centramos a la hora de buscar fueron las de Google Maps, Facebook y Twitter. A continuación pasaremos a ver las distintas apariencias de cada una de ellas y detallar sus pros y sus contras:

- **Google Maps**: la API de Google Maps[10] presenta la opción de elegir la plataforma que deseas de entrada (Android, IOS o Web). Sin embargo la página principal de Google Maps es mucho más sencilla ya que solo te da a elegir entre las plataformas y una opción de ver el directorio de productos completo, sin ninguna imagen publicitaria que distraiga o lleve a confusión. Además los ejemplos, al igual que en Facebook, también están claramente explicados, siendo muy sencillo seguir el comportamiento y la funcionalidad del código, incluso en algunos ejemplos muestran un video con el resultado obtenido al realizar determinada operación. Por ello nos hemos basado en esta API para la página de inicio de la nuestra, sin hacer distinción entre plataformas.

  Podemos observar en 3.13 como la página principal es sencilla y concisa, sin elementos que distraigan de lo que se anda buscando.

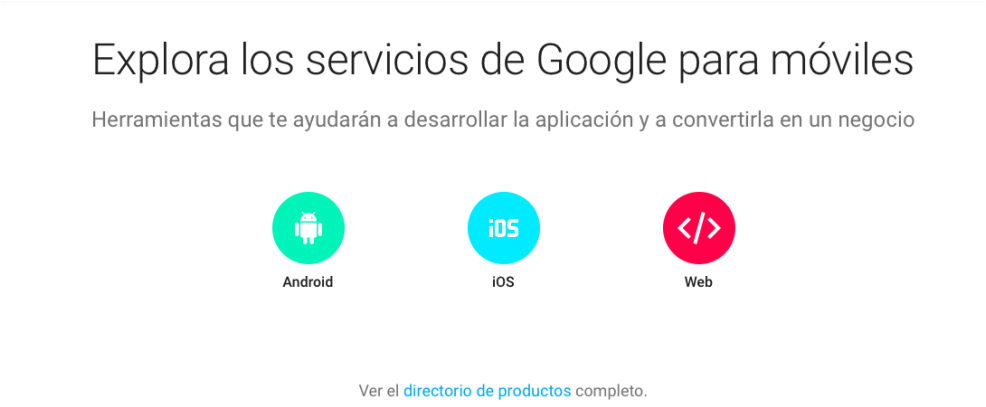


Figura 3.13: Google Inicio

[10] https://developers.google.com/maps/documentation/javascript/?hl=es

En la imagen 3.14 podemos ver un ejemplo bien detallado de la funcionalidad de login con su imagen correspondiente

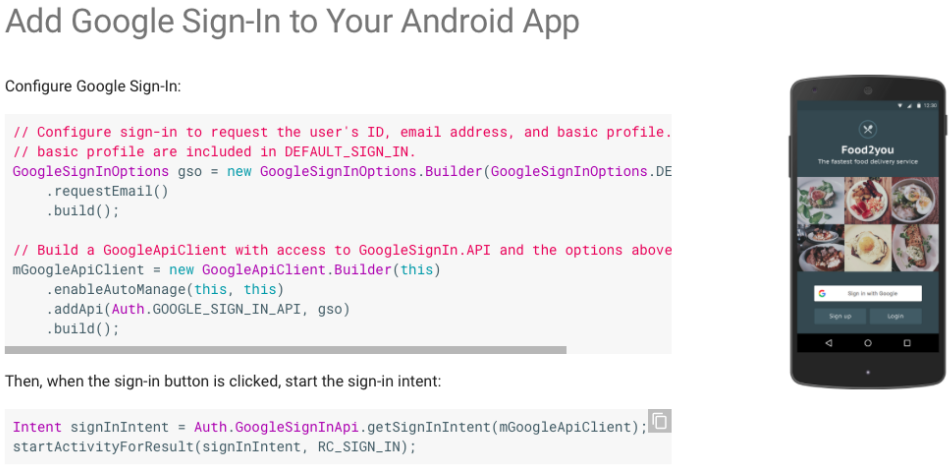


Figura 3.14: Google Ejemplo

- **Facebook**: La API de Facebook[11], aunque muy intuitiva una vez que estás dentro, tiene una presentación inicial que puede ocasionar confusión de entrada ya que dispone de una serie de imágenes que te llevan más a pensar en una campaña publicitaria. Una vez que te sitúas en lo que andas buscando resulta bastante sencilla e intuitiva, con ejemplos bastante claros y una organización concisa. Además divide claramente las funcionalidades que desarrollan en tres grandes campos: IOS, Android y Web, lo cual resulta muy útil, ya que normalmente se buscará el código para una plataforma en concreto, no importando tanto el resto de ellas. El mayor problema con la API de Facebook es que dispone de muchos niveles de profundidad, llegando a ser complicado acceder a lo que estás buscando si no tienes claro donde está, lo cual consideramos que resulta engorroso y que puede llevar a los usuarios a buscar en otras APIs en las que resulte más fácil acceder a lo que buscan concretamente. Sin embargo en la imagen 3.15 podemos ver como presentan claridad en los ejemplos facilitando así el uso del código proporcionado.

  En la imagen 3.16 podemos ver como de entrada parece más una página de publicidad de Facebook y puede distraer nuestra atención de lo que se busca realmente.

[11] https://developers.facebook.com/products/

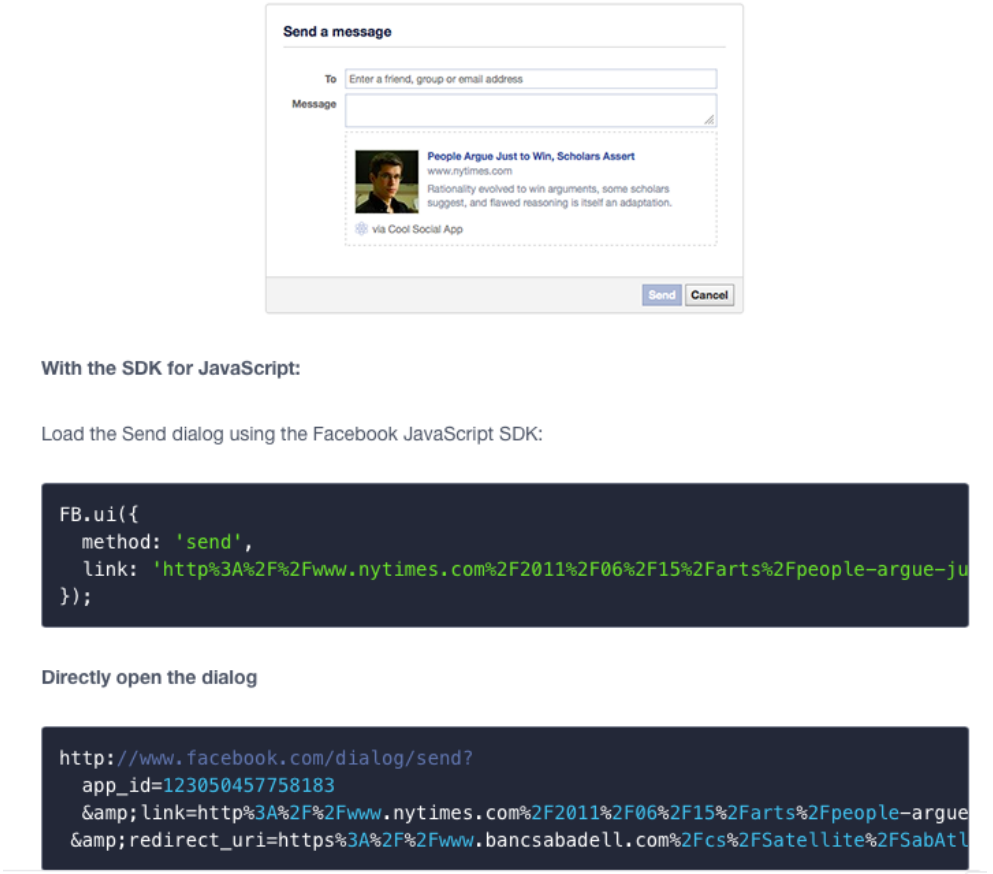


Figura 3.15: Facebook Ejemplo

- **Twitter**: Con la web de Twitter [12] pasa un poco lo mismo que con la de Facebook, el inicio resulta mas engorroso, mostrando más publicidad sobre el tema y centrándose menos en ir más directos. En ella muestran información poco relevante, como anuncios de conferencias, que hacen que al usuario le sea mas dificil concentrarse en el propósito principal de la API. Además ponen los enlaces a los servicios con el link en vez de poner un nombre más autoexplicativo, lo cual hace que la búsqueda resulte más complicada. Por otro lado a la hora de poner los ejemplos estos resultan muy claros y de fácil manejo, y además consta de una estructura lateral en la cual puedes ver en que elemento te encuentras y moverte por otros de forma sencilla lo cual facilita moverse por las distintas funcionalidades. Esta funcionalidad nos ha resultado muy interesante y útil por lo que hemos incorporado la idea en nuestra propia API.

[12] https://dev.twitter.com/overview/documentation

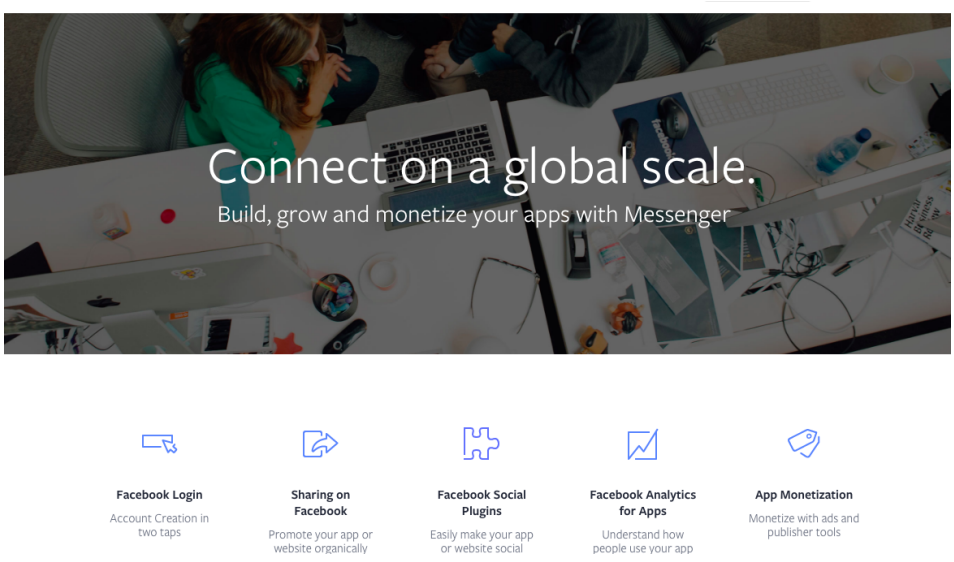


Figura 3.16: Facebook Inicio

En 3.17 observamos el inicio sobrecargado explicado anteriormente.

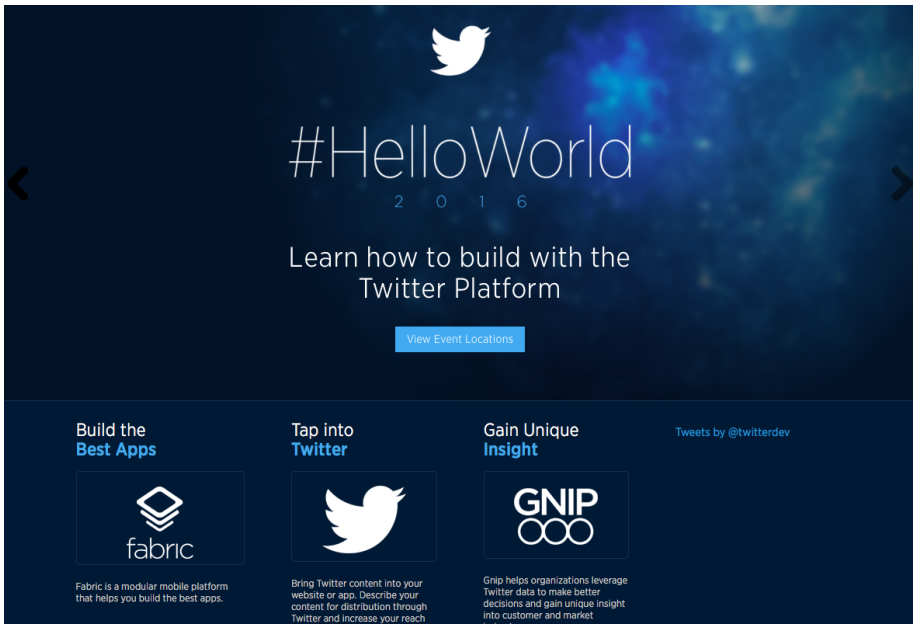


Figura 3.17: Twitter inicio.

En la imagen 3.18 podemos ver como son explicados los pasos a seguir con claridad y precisión.

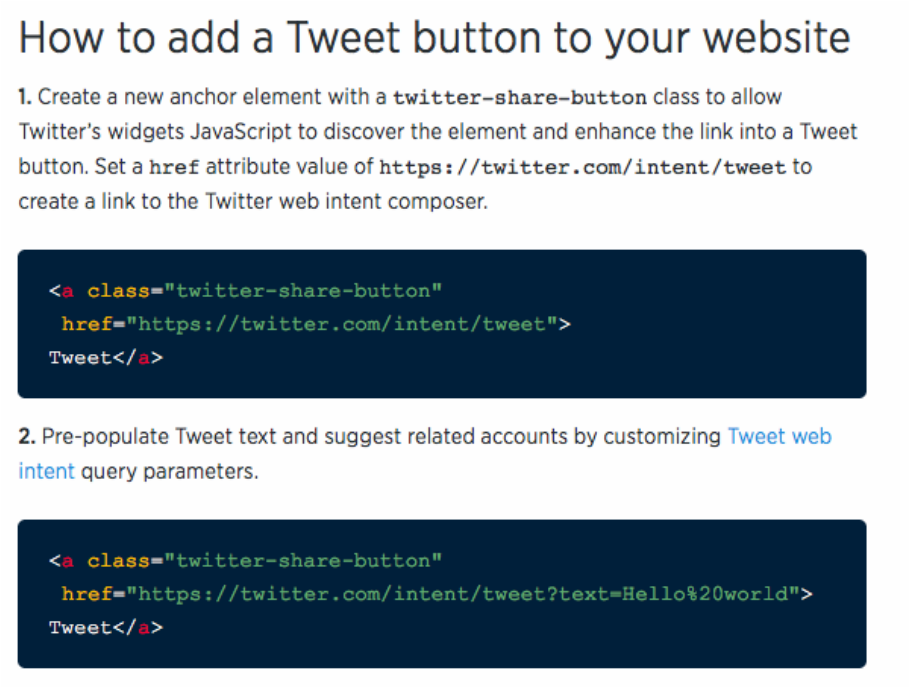


Figura 3.18: Twitter Ejemplo.

3.5. Conclusiones y elementos a utilizar

De toda la información recopilada a través del análisis realizado en este capítulo, se han sacado en claro varias ideas para poder utilizar en el desarrollo del proyecto.

En primer lugar se ha visto que para realizar una buena API, esta tiene que aportar un concepto claro de las funcionalidades que ofrece, para que el usuario que decida utilizarlas tenga un esquema claro de su funcionamiento y de cómo se puede acceder a ellas.

Nuestra API va a ser de tipo de servicios Web, proporcionando acceso a su servicio mediante una dirección web en una red, utilizando el estándar REST. Este tipo de estándar, permite la transferencia de datos a través de peticiones HTTP, haciendo que la comunicación sea transparente y natural, permitiendo interactuar con otros sistemas de una forma sencilla. El diseño transparente lo consigue gracias a la utilización de métodos siendo los principales y más utilizados GET,POST, DELETE y PUSH.

Por último, se han analizado distintos ejemplos de APIs, donde se ha conseguido obtener una visión de algunas de las cualidades que debería tener la API que vamos a generar. La página principal tiene que ser sencilla, concisa y no tener elementos adicionales con poca relevancia que pueda dis-

traer al usuario, para que le resulte fácil acceder a la información que busca concretamente.

Otra idea, es hacer que los enlaces a los servicios tengan un nombre auto-explicativo para que resulte intuitivo y, de esta forma, facilite el movimiento por las distintas funcionalidades que ofrece la API.

Capítulo 4

Planteamiento del proyecto

Como hemos explicado en el capítulo 1, este proyecto tenía la finalidad inicial de agrupar los servicios de accesibilidad generados durante años anteriores en la Facultad de Informática, unificándolos y facilitando que pudieran ser usados por otros desarrolladores, así como ampliándolos con nuevos servicios en el caso de disponer del tiempo necesario.

Se presentan finalmente ocho tipos de servicios, una aplicación web API y una aplicación Android. Los servicios desarrollados son los siguientes:

- **Palabra sencilla:** devuelve *true* en caso de que una palabra se considere sencilla y *false* en caso contrario. El criterio utilizado para considerar a una palabra sencilla o dificil viene dado por un archivo de la RAE . Consideramos una palabra sencilla si se encuentra dentro de este archivo. Se darán más detalles cuando pasemos a hablar sobre este servicio más adelante. Los resultados de este servicios están disponibles en JSON y XML.

- **Sinónimos**: devuelve todos los sinónimos de una palabra introducida en la URL mediante un parámetro. El formato del resultado puede ser XML o JSON.

- **Convertir a pictograma:** este servicio se encarga de, dada una palabra o frase, devolver el pictograma o la secuencia de pictogramas que las representan. Este tiene dos formatos distintos de URL en función de si se va a meter una sola palabra o una frase.

- **Antónimos:** devuelve todos los antónimos de una palabra introducida en la URL mediante un parámetro. El formato del resultado puede ser XML o JSON.

- **Definiciones:** se encarga de devolver todas las definiciones de las palabra buscada mostrando solo las acepciones, sin mostrar información

adicional como coloquialismos, frase hechas con esa palabra o ejemplos del uso de esas palabras con la acepción mostrada en ese momento. El formato del resultado puede ser XML o JSON.

- **Convertir a palabra sencilla:** dada una palabra introducida como parámetro en la URL de este servicio, si aplicándosele el servicio "Palabra sencilla" este devuelve *false* se procede a buscar un sinónimo de la palabra que sea sencillo, el cual será el resultado devuelto. En el caso de que no se encuentre sinónimo sencillo o que la palabra inicial sea sencilla, se devolverla la palabra introducida en la URL.
- **Traducir a inglés:** este servicio devuelve las traducciones a inglés de una palabra introducida en la URL.
- **Lema:** este devuelve el origen y la categoría gramatical de la palabra buscada.

Estos servicios han sido desarrollados en Java, como servicios REST. Se decidió una implementación REST ya que estos servicios están basados en estándares como HTML, URL, XML, JSON, etc., siendo estos los que se querían utilizar en nuestros servicios.

Para el desarrollo de los servicios en Java se ha hecho uso de la librería JSoup para analizar las páginas web usadas en nuestro proyecto como herramientas. Algunos ejemplos de las mencionadas páginas y recursos usados son:

- Para la definición, sinónimos y antónimos: WordReference[1]
- Para traducción: Span¡shD!ct[2]
- Para palabras fáciles: 1000_formas.TXT[3], archivo de texto aportado por la RAE de las 1000 palabras más utilizadas en la lengua española. Se verá más sobre este archivo más adelante.

El análisis de las páginas web se hace sobre su código HTML. Para ello se debe estudiar la pagina, analizando como está desarrollada y como están etiquetados los resultados que devuelve para posteriormente coger los que nos interesen de la misma.

Una vez creados los servicios, para facilitar el acceso a ellos se crea la web de la API donde se aporta toda la información sobre cada uno de los servicios, tanto descripciones como modos de uso y ejemplos. Gracias a esta

[1] http://www.wordreference.com/es/
[2] `http://www.spanishdict.com/traductor/`
[3] http://corpus.rae.es/frec/1000_formas.TXT

API estos servicios pueden ser utilizados tanto por un desarrollador para la implementación de una aplicación que los utilice como herramientas, como por un usuario que tan solo quiera hacer alguna consulta mediante la web.

Nuestro proyecto además de los servicios ya descritos consta de una aplicación Android a modo de ejemplo. Esta aplicación está formada por una pantalla principal donde el usuario puede introducir la palabra a consultar y seleccionar el servicio que le quiera aplicar, estando estos representados en forma de botones en la aplicación.

Tras la introducción de la palabra y la selección del servicio la aplicación pasará a la pantalla secundaria en la cual se mostrará el resultado de aplicar el servicio seleccionado a la palabra introducida, con un formato sencillo indicando con una pequeña frase que servicio está devolviendo seguido del resultado.

En los próximos capítulos nos centraremos en hablar sobre los servicios desarrollados, la página web de representación de la API y una aplicación en android creada con el fin de mostrar un pequeño ejemplo del uso que se le puede dar a los servicios generados.

En la imagen 4.1 se puede ver un diagrama sencillo del funcionamiento del proyecto.

Figura 4.1: Diagrama explicativo del proyecto.

Capítulo 5

Servicios

5.1. Aspectos comunes de los servicios utilizados

En este apartado se tratan los aspectos que tienen todos los servicios en común para no hacer repetición de información en cada servicio.

Los servicios de este proyecto se han desarrollado en eclipse, introduciéndolos todos dentro de un mismo paquete con cada servicio implementado en una clase diferente. Cada uno utiliza sus propios métodos, es decir, no se apoya en los de otras clases y, si en algún servicio necesita información externa, se obtiene haciendo llamadas a los servicios almacenados en el servidor "*sesat.fdi.ucm.es*". En la imagen 5.1 se puede ver la estructuración del paquete con todos los servicios.

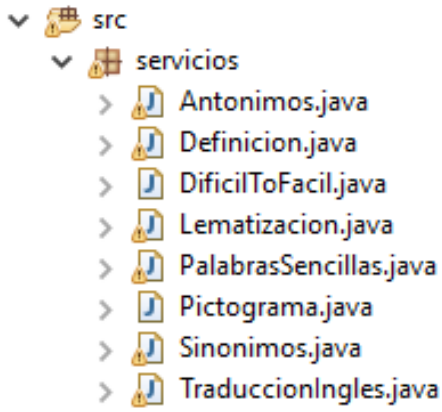


Figura 5.1: Paquete con los servicios en Eclipse.

Para la generación de los servicios se crea un archivo xml (web.xml) en el cual queda especificado el formato concreto que debe tener la URL del servidor para acceder a los servicios ofrecidos. En la figura 5.2 se puede observar el formato del archivo anteriormente mencionado en el cual `servlet-name`

representa el nombre asignado al servidor y `servlet-class` el de la clase. En la variable `init-param` se introduce un parámetro llamado `param-value` al que se le pasa una palabra, por la cual va a empezar la URL, seguida del nombre del servidor y el puerto dónde se encuentra el servidor.

```
<servlet>
<servlet-name>Jersey REST Service</servlet-name>
<servlet-class>org.glassfish.jersey.servlet.ServletContainer
</servlet-class>
<init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>servicios</param-value>
</init-param>

 <load-on-startup>1</load-on-startup>

 </servlet>
```

Figura 5.2: Archivo web.xml

Además de lo ya mencionado anteriormente, se ha de añadir a la URL el parámetro */rest/** el cual completaría toda la parte que tienen en común las URL de todos nuestros servicios, quedando de la siguiente forma:

sesat.fdi.ucm.es:8080/servicios/rest/

y cuya implementación queda representada en la figura 5.3. Esta URL será completada en los siguientes apartados según el servicio que quiera ser llamado, guardando consistencia unos servicios con otros. Para guardar esta consistencia hacemos que las URL de todos los servicios respondan a la siguiente estructura:

sesat.fdi.ucm.es:8080/servicios/rest/nombreServicio/XMLJSON/palabra
en la cual:

- **nombreServicio:** hace referencia al nombre del servicio asignado para la URL, siendo este una versión reducida del nombre del mismo. Se verá con más detalles cuando se explique cada servicio.
- **XMLJSON:** en esta parte se pondrá XML o JSON para especificar el formato de salida en el que queremos el servicio.
- **palabra:** palabra introducida por el usuario a la que se le quiere aplicar el servicio

Este final de ruta queda representado en cada servicio por el parámetro **Path(NombreServicio)**, declarándose esto en la cabecera de la clase de

```
<servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

Figura 5.3: Archivo web.xml para URL completa

cada uno de los servicios, indicando así en cada uno de ellos la ruta concreta que este posee.

Además se utilizarán una serie de parámetros importantes para el desarrollo de los servicios que quedan explicados a continuación:

- **@GET:** se tiene que definir en todos los métodos en lo que se vaya a hacer una petición *get*, es decir, en todos aquellos en los que se quiera realizar una petición de información.
- **@Produces(NombreEjemplo):** sirve para indicar el formato en el que el servicio REST nos devuelve el resultado.

En este proyecto los formatos que se devuelven son XML y JSON, siendo éste último añadido en versiones más avanzadas del proyecto, habiendo empezado inicialmente devolviendo solo el XML. Esto queda realizado en el método que se encarga de devolver el XML dentro de las distintas clases. En el caso del método encargado de devolver el JSON no lo necesitamos, ya que este lo devuelve en forma de *String*.

Es en los métodos anteriormente mencionados donde se acaba de completar la ruta de cada servicio, quedando esta de la siguiente forma:

- *sesat.fdi.ucm.es:8080/servicios/rest/NombreServicio/xml/"palabra"*
- *sesat.fdi.ucm.es:8080/servicios/rest/NombreServicio/JSON/"palabra"*

Siendo "palabra" la palabra a la que el usuario le quiera aplicar el servicio NombreServicio.

Para la representación de la salida en XML se utiliza la instrucción `@Produces(MediaType.TEXT_XML)` en la cabecera del método y se añade `<?xml version=\"1.0\"?>` antes de las etiquetas XML que se quieran devolver, tratando el resto como un string normal. Esto permite que el texto devuelto sea reconocido como formato XML y pueda ser interpretado por el navegador sin problema.

En cuanto a JSON existen dos opciones diferentes para su producción, `@produces(MediaType.APPLICATION_JSON)` y `@produces(MediaType.APPLICATION_JSON_TYPE)`, siendo el resultado de la

primera la conversión de un *String* en formato JSON pero devolviendo el resultado como un archivo JSON en lugar de un texto en ese formato. Así, si se ejecuta por ejemplo:

http://localhost:8080/servicios/rest/palabras/json/hola

se genera el archivo de extensión .json, saliendo un mensaje en el que se da la opción a guardar dicho archivo. En el caso de `@produces(MediaType.APPLICATION_JSON_TYPE)` se producía un error que no llegamos a resolver.

Finalmente el formato JSON se genera mediante la construcción de un JSON y devolviendo este convertido a string. Para ello utilizamos la librería `org.json.JSONObject` en nuestro proyecto.

En el caso del servicio de pictogramas la URL resulta diferente puesto que no interesa si es XML o JSON ya que lo que devolvemos es una imagen, por lo que quedaría eliminada la notación de `@Produces`, y en vez de poner XML o JSON pondríamos "pict", pero esto ya quedará explicado más adelante cuando se hable del servicio pictogramas en el apartado 5.4

`@PathParam()` es un elemento que sirve para enviar a través de la URL un parámetro especificado por el usuario, para que este pueda ser utilizado en el método que llama a esa URL. Para ello lo que se hace es pasar una palabra por la URL recogiendo esta con `@PathParam()` mediante una variable por valor, la cual luego será utilizada en el método como se ve en la figura 5.4.

```
public String formatoXML(@PathParam("palabra") String palabra)
```

Figura 5.4: Uso de @PathParam

En los distintos servicios que se verán en las siguientes secciones se irán utilizando los elementos que se mencionan a continuación, de los cuales se verán ejemplos de uso cuando se expliquen los distintos servicios:

- `@Path(''/palabra'')`
- `@PathParams(''/palabra'')`
- `@Produces(''application/json'')` y `@Produces(''application/xml'')`
- `@FormParam(''/palabra'')`
- `@PathParam(''/palabra'')`
- `@GET(''/palabra'')`
- `@Consumes(MediaType.APPLICATION_JSON)`

- `@Consumes(MediaType.APPLICATION_XML)`

Siendo *palabra* la palabra introducida por el usuario.

Fue desarrollada una versión de los servicios en la cual se llevaba a cabo un control de errores que aportara un *feedback* en caso de que se produjera algún error del tipo "palabra introducida incorrecta". Esto suponía una ventaja de cara a informar al usuario, en caso de que introdujera mal una palabra, porque el servicio no le devolvía el resultado que estaba esperando.

Esta versión fue finalmente suprimida ya que se llegó a la conclusión de que los servicios no iban a suponer una interacción directa con un usuario sino una herramienta para otros desarrolladores, los cuales serían los encargados de decidir que hacer con los mensajes vacíos o en caso de que un usuario cometiera un error al introducir la palabra, por lo que finalmente se optó por devolver un mensaje vacío en caso de palabra incorrecta.

Para devolver los resultados en XML se tiene, en los distintos servicios, el método `convertiraXML`. Este método devuelve la salida en formato XML, y dependiendo de la salida del servicio, puede producirse algún cambio en su implementación, pero la estructura y la lógica empleada en todos los servicios para este método es la misma. El parámetro de entrada "lista", contiene una lista de *String* de tipo `ArrayList`, con todos los resultados que se quieren mostrar.

En el método `convertirJson` ocurre lo mismo que con el `convertirXml` explicado arriba, todos los servicios siguen la misma lógica y estructura de implementación, sufriendo pequeños cambios. Este método se encarga de, dada una lista de String de tipo `ArrayList` que contiene los resultados a mostrar,construir el JSON correspondiente.

El método `reemplazarCaracteresRaros` se encarga de cambiar las palabras con tilde, así como caracteres no estándar, por su equivalente estándar. Es decir, si se tiene la palabra "cacería", tras pasar por el método `reemplazarCaracteresRaros` se tendría "caceria". La implementación de este método ha sido fundamental para tratar el formato JSON ya que no se ha conseguido devolver en este formato las palabras con tilde u otros símbolos no estándar.

Para devolver una cadena vacía en XML basta con devolver en el string correspondiente la cadena `<servicio></servicio>` lo cual devolverá el mensaje XML `<servicio/>`, siendo este una representación XML de mensaje vacío. En JSON no hay un resultado de "vacío" tan estandarizado por lo que nos planteamos las siguientes opciones como representación de mensajes vacíos: `{}, {servicio: null}, {servicio, {}} o {servicio,[]}`. Finalmente nos decantamos por `{servicio, {}}` al resultar la más similar a la forma devuelta por XML.

Para realizar los servicios de sinónimos, antónimos y definición este proyecto se ha apoyado en uno ya desarrollado años anteriores **"Apoyo a la simplificación de textos sobre navegadores web"** González y Ortiz (2013 -2014), el cual ha quedado explicado en el punto 3.3.

Este proyecto estaba implementado en PHP y basado en servicios SOAP por lo que no se podía reutilizar directamente, ya que este tiene desarrollados sus servicios como REST programados en Java. se realizó, en cualquier caso, un estudio del código, reutilizando ideas de implementación como consultas a una página web para obtener los datos necesarios para los servicios.

5.2. Servicio palabra sencilla

Servicio encargado de especificar si una palabra es sencilla o no. Introducida la palabra el servicio devolverá *true* si es fácil y *false* en caso contrario.

Para desarrollar este servicio lo primero que hay que tener en cuenta es qué se considera una palabra sencilla y qué no. Hemos decidido usar como criterio para saber si una palabra es sencilla o no el número de veces que esta es utilizada. Así si una palabra es utilizada muchas veces es más probable que sea palabra sencilla que otra que es usada menos frecuentemente.

Dicho esto, se considera para este proyecto palabra fácil toda aquella que se encuentre entre las 1000 más utilizadas del castellano, para lo cual se accede a un archivo proporcionado por la Real Academia Española (RAE), 1000_formas.TXT [1] en el cual aparece una lista de las 1000 palabras más utilizadas en la lengua española. Además, se muestra el orden en el que aparecen las mismas (de la más usada a la menos usada), y sus frecuencias absolutas y normalizadas.

Además del archivo mencionado, existen otros dos ficheros cuya única diferencia es el número de palabras que contienen (5000 y 10000 palabras). Se optó por el de 1000 ya que se podía observar que excedidas las 1000 palabras aparecían algunas que, desde nuestro punto de vista, no resultaban sencillas, como nación, terreno, rostro, etc.

En la imagen 5.5 se pueden ver algunas de las primeras palabras excedentes al archivo de las 1000, así como la estructura anteriormente mencionada.

Se utiliza el archivo de la Real Academia Española debido a que este se irá actualizando a lo largo del tiempo, mientras que si se usara un fichero propio subido al servidor "*http://sesat.fdi.ucm.es:8080*", lo mas probable es que no se volviera a modificar y quedándose finalmente obsoleto.

[1] `http://corpus.rae.es/frec/1000_formas.TXT`

```
1029. novela      13,361          87.57
1030. alma        13,357          87.55
1031. doble       13,353          87.52
1032. anteriores          13,350          87.50
1033. obtener     13,347          87.48
1034. dije        13,346          87.48
1035. selección           13,345          87.47
1036. podido      13,335          87.40
1037. nación      13,332          87.38
1038. ido         13,331          87.38
1039. crear       13,289          87.10
1040. motivo      13,285          87.08
1041. tercer      13,284          87.07
1042. detrás      13,283          87.06
1043. significa           13,261          86.92
1044. empleo      13,252          86.86
1045. escrito     13,247          86.83
1046. departamento        13,232          86.73
1047. contacto            13,227          86.70
1048. casas       13,203          86.54
1049. red         13,203          86.54
1050. rostro      13,200          86.52
1051. oportunidad         13,103          85.88
1052. procesos            13,098          85.85
1053. terreno     13,097          85.84
1054. daba        13,096          85.84
1055. calles      13,077          85.71
1056. lista       13,060          85.60
1057. nacionales          13,060          85.60
1058. funciones           13,057          85.58
```

Figura 5.5: Fragmento del archivo 5000_formas.TXT de la RAE.

A la hora de analizar el archivo "1000_Formas.TXT" no se tienen en cuenta mayúsculas y minúsculas, dando como resultado que la búsqueda de la palabra "Casa" sería lo mismo que la de "casa".

Se consideraron dos opciones a la hora de mostrar el resultado en este servicio:

- "Palabra sencilla" o "Palabra difícil": resultaba mucho más claro ya que te da la información que le preguntas al servicio.
- "*True*" o "*False*": resulta más fácil de tratar por otro desarrollador.

Finalmente se optó por devolver *true* o *false* ya que el objetivo de los servicios es que puedan ser utilizados por otros desarrolladores y de esta forma se facilita el análisis, y por tanto el uso de los mismos.

En la figura 5.6 se puede ver un diagrama explicativo de como funciona

este servicio. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

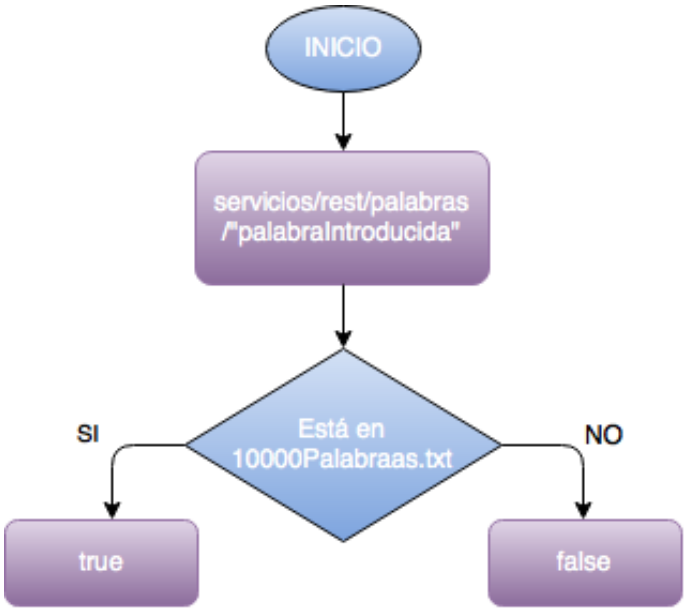


Figura 5.6: Diagrama explicativo del servicio palabras sencillas

5.3. Servicio sinónimos

Servicio que muestra los sinónimos de la palabra introducida, obteniendo la información de la página de WordReference[2] gracias a una consulta previa. Si la palabra no tiene sinónimos o no existe este servicio devolverá un resultado vacío. Como hemos visto en el apartado 5.1 el que devuelva un resultado vacío en lugar de un mensaje del estilo de "Sinónimo no encontrado" se debe a que resulta más inmediato de tratar para terceros si no devuelve nada, dejando ya al criterio del que utilice el servicio que hacer con el resultado final cuando este te devuelva vacío.

Una vez que se ha obtenido la información de la consulta a WordReference, se analiza el resultado del html devuelto y se obtienen los diferentes sinónimos proporcionados por esta y guardándose estos en un `ArrayList` para luego mostrarlo, dependiendo de la opción seleccionada, en XML o JSON.

La estructura general del resultado, independientemente de si es en XML o en JSON, viene dada por el siguiente esquema:

[2] `http://www.wordreference.com/sinonimos/`

- Una etiqueta “sinonimos” seguido de la etiqueta “sinonimo” (una por cada entrada/acepción que aparezca de esta palabra).

A continuación podemos ver la invocación del servicio de sinónimos con la palabra “buscar”, tanto con XML como con JSON.

La llamada con XML sería:

http://sesat.fdi.ucm.es:8080/servicios/rest/sinonimos/xml/confundir
cuyo resultado se puede ver en la imagen 5.7

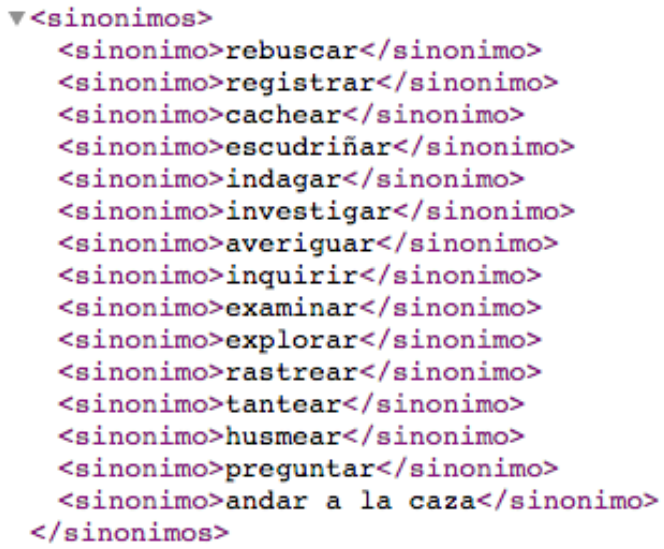


Figura 5.7: Ejemplo de resultado de sinónimos en XML

Mientras que la llamada con JSON quedaría:

http://sesat.fdi.ucm.es:8080/servicios/rest/sinonimos/json/confundir
cuyo resultado se puede ver en la imagen 5.8

{"sinonimos":[{"sinonimo":"rebuscar"},{"sinonimo":"registrar"},
{"sinonimo":"cachear"},{"sinonimo":"escudrinar"},{"sinonimo":"indagar"},
{"sinonimo":"investigar"},{"sinonimo":"averiguar"},{"sinonimo":"inquirir"},
{"sinonimo":"examinar"},{"sinonimo":"explorar"},{"sinonimo":"rastrear"},
{"sinonimo":"tantear"},{"sinonimo":"husmear"},{"sinonimo":"preguntar"},
{"sinonimo":"andaralacaza"}]}

Figura 5.8: Ejemplo de resultado de sinónimos en JSON

Cabe destacar que uno de los motivos que nos llevo a utilizar la página de WordReferences como la que nos "proporciona" los sinónimos es que, cuando buscas una palabra de la cual ellos no tienen ninguna acepción, es decir, que no se encuentra en su sistema, buscan su lema y vuelven a realizar la consulta con este, siendo el resultado de la nueva, en el caso de que lo hubiera, lo que te muestran como resultado de la búsqueda final. Además, al ser WordReference una web bastante accedida y fiable, nos aseguramos de que nuestro contenido esté actualizado y accesible.

Por ejemplo, si se busca "compraría" no encuentra directamente sinónimos de la palabra "compraría" sino que la lematiza para obtener el lema de la palabra, en este caso "comprar" y nos devuelve sinónimos de "comprar". En la imagen 5.9 se puede ver el resultado de este ejemplo.

Figura 5.9: Ejemplo de consulta "compraría" en WordReferences.

Para terminar de aclarar este servicio en la figura 5.10 se muestra un diagrama explicativo de como funciona. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

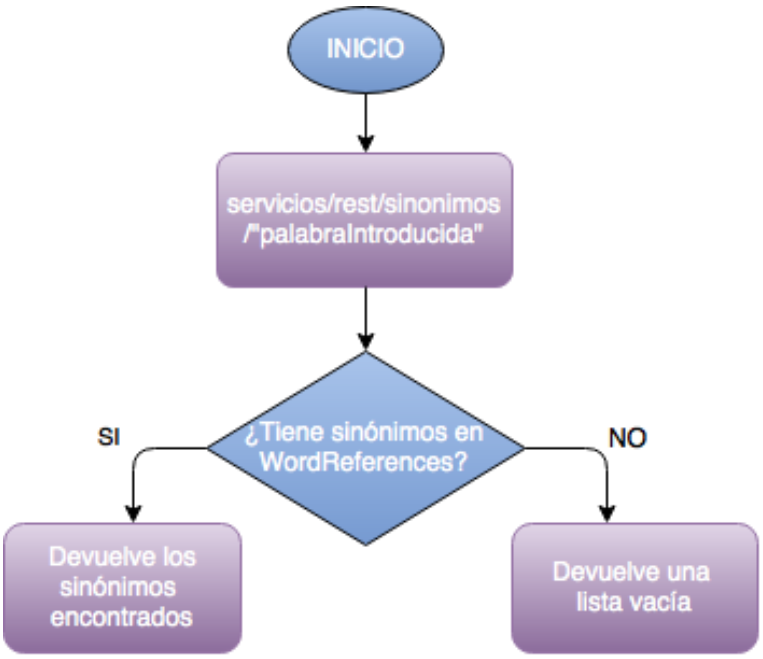


Figura 5.10: Diagrama explicativo del servicio sinonimos.

5.4. Servicio convertir a pictograma

Servicio encargado de devolver un pictograma correspondiente a la palabra o frase pasadas por parámetro. Este tiene dos variantes de las cuales se hablará a continuación y a las cuales se accede con las siguientes URL:

- *http://sesat.fdi.ucm.es:8080/servicios/rest/pictograma/frase/ "fraseBuscada"*
- *http://sesat.fdi.ucm.es:8080/servicios/rest/pictograma/palabra/ "palabraBuscada"*

En las imágenes 5.11 y 5.12 se muestran dos ejemplos de resultados obtenidos al darle el valor de "ropa de verano" a "fraseBuscada" y de "jugar" a "palabraBuscada", respectivamente.

Este servicio está basado en el proyecto "Conversor de texto a pictogramas" (Alonso et al. (2013 -2014)). En este se realiza una llamada al servicio "hypatia" de la forma:

hypatia.fdi.ucm.es:5223/PictoTraductorAplicationServer/api/TraduceFrase/"palabra"

En esta consulta se devuelve un JSON con los objetos encontrados en la BBDD que hagan referencia a la palabra o frase representadas por "palabra".

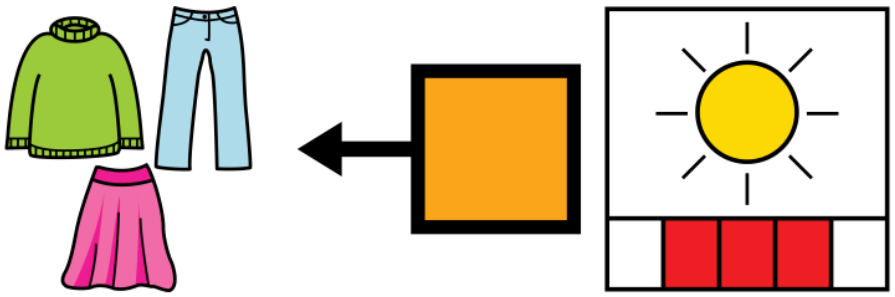


Figura 5.11: Ejemplo de consulta a “frase” con la frase “ropa de verano”.

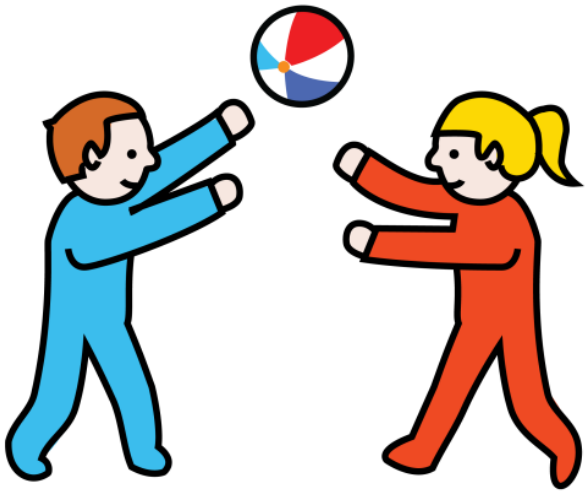


Figura 5.12: Ejemplo de consulta a “palabra” con la palabra “jugar”.

En la imagen 5.14 podemos ver un ejemplo de resultado devuelto. En este se puede observar que, para una sola palabra introducida (“comer” en este caso), se devuelven distintos resultados con bastante información en cada uno de ellos.

De esta información obtenida, en este proyecto solo se cogerá el “id_url”, siendo este el atributo que indica el nombre de la imagen alojada en el directorio del servidor de “hypatia”, el cual devolverá la imagen que se anda buscando. El acceso a este sería de la siguiente forma:

http://hypatia.fdi.ucm.es/conversor/Pictos/“id_url”

Si el servicio no encuentra la palabra en la BBDD, hace una nueva llamada pero esta vez, en vez de buscar la palabra introducida, buscaría el lema de

la misma. Es decir, si se introduce “comiendo” y no encuentra un pictograma para “comiendo”, realizaría una nueva búsqueda con la palabra “comer”.

En la imagen 5.13 podemos ver un diagrama explicativo de como funciona el servicio pictograma con la llamada a frase. La llamada a palabra funciona de manera similar por lo que el diagrama sirve para ambos casos.

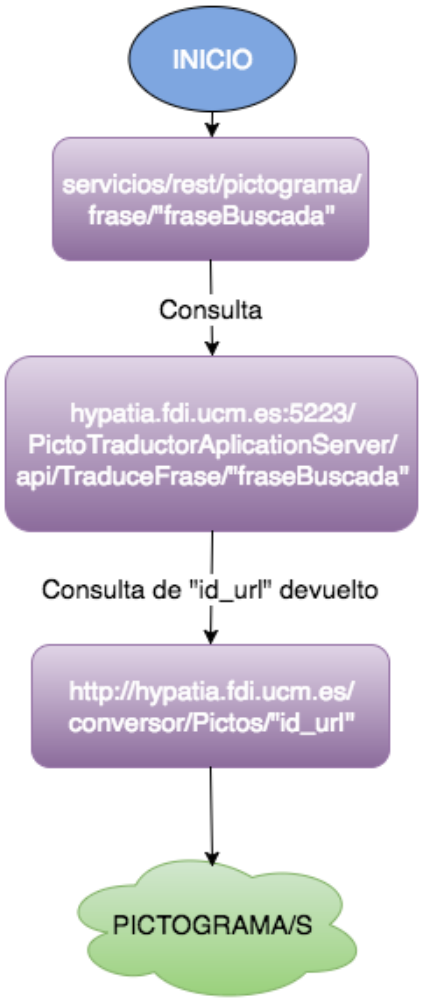


Figura 5.13: Diagrama explicativo del servicio pictograma.

El tiempo consumido en la llamada al servicio “traduceFrase” del proyecto “Conversor de texto a pictogramas” (Alonso et al. (2013 -2014)) supone un inconveniente, ya que al hacer consultas a la BBDD tarda un tiempo excesivo en mostrar la información requerida, suponiendo esto un inconveniente a la hora de llevarlo a la aplicación Android.

```
[[{"id_palabra":187,"nombre":"comer","id_url":2349,"color":3},
{"id_palabra":1927,"nombre":"comer","id_url":4609,"color":3},
{"id_palabra":3549,"nombre":"comer","id_url":6456,"color":3},
{"id_palabra":4137,"nombre":"comer","id_url":6990,"color":3},
{"id_palabra":14213,"nombre":"comer","id_url":28641,"color":3},
{"id_palabra":14250,"nombre":"comer","id_url":28669,"color":3}]]
```


Figura 5.14: Resultado devuelto de la búsqueda de "comer" en hypatia.

5.5. Servicio antónimos

En este servicio, dada una palabra, se devuelve la lista de antónimos encontrados de la misma. De nuevo, como en sinónimos, se hace una consulta a la web de sinónimos de WordReference [3], ya que esta contiene, además de la lista de sinónimos, la lista de antónimos de la palabra introducida. En la imagen 5.15 se puede ver un ejemplo de como están estos representados en WordReference.

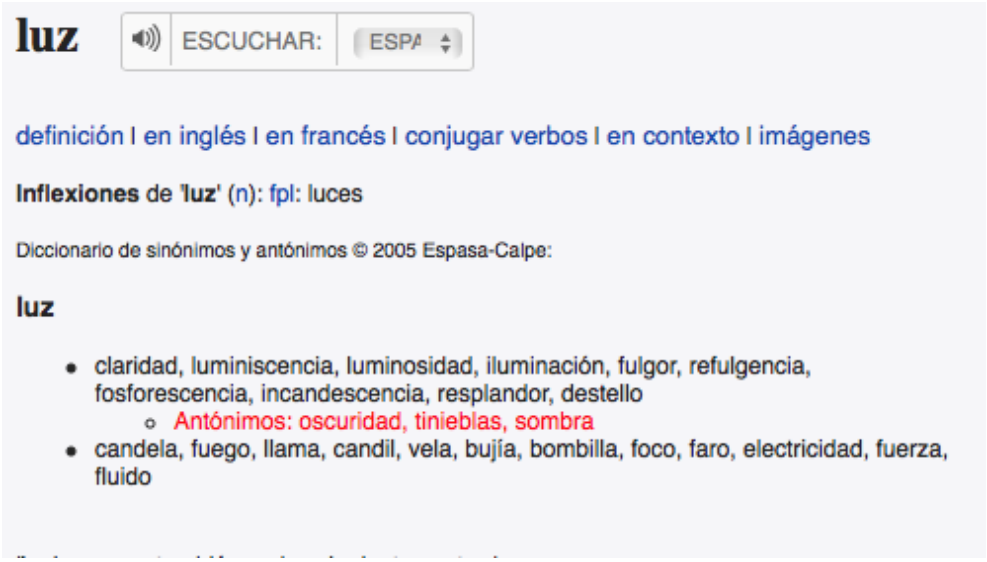


Figura 5.15: Imagen de ejemplo de representación de antónimos en WordReference.

Como se puede ver en la imagen WordReference no tiene una página dedicada para los antónimos como tiene para las definiciones o sinónimos, si no que, dentro de la de sinónimos introduce una lista de antónimos resaltados en rojo, la cual es utilizada para llevar a cabo este servicio.

[3] `http://www.wordreference.com/sinonimos/`

Funciona de una manera muy similar al de sinónimos, ya que hace una consulta a la misma página web y el resultado devuelto, en formato, es muy parecido, cambiando los nombres de las etiquetas.

A continuación se ve un ejemplo de cada uno de los formatos de las llamadas.

La llamada JSON sería:

http://sesat.fdi.ucm.es:8080/servicios/rest/antonimos/json/subir

El resultado de esta llamada se puede ver en la imagen 5.16

{"antonimos":[{"antonimo":"bajar"},
{"antonimo":"descender"},{"antonimo":"disminuir"}]}

Figura 5.16: Llamada a antónimos en formato JSON de la palabra "subir".

La llamada XML sería:

http://sesat.fdi.ucm.es:8080/servicios/rest/antonimos/xml/subir

El resultado de esta llamada se puede ver en la imagen 5.17


```
▼<antonimos>
   <antonimo>bajar</antonimo>
   <antonimo>descender</antonimo>
   <antonimo>disminuir</antonimo>
 </antonimos>
```


Figura 5.17: Llamada a antónimos en formato XML de la palabra "subir".

Este servicio, además, consta de una función principal donde se desarrollan las partes mas importantes para poder hacer la petición a la pagina correspondiente y el análisis del contenido.

En la implementación del servicio antónimos la primera acción que se realiza es conectar con la pagina mediante la URL. Se utiliza la función `URLEncoder.enode` para añadirle la palabra que el usuario escriba por la URL y tenerlo en UTF-8.

Una vez establecida la conexión con la web, se procede al análisis de la pagina html con la librería anteriormente nombrada JSOUP. Esta librería se encarga de buscar los textos que van acompañados por una etiqueta, por

ello primero se busca la etiqueta `div[class=trans clickable]` ya que si esta no se encontrara dentro del código html, será porque no existiría ningún resultado para la consulta.

Una vez encontrada la etiqueta, no se puede dar por hecho que esta sea la valida, por ello para la implementación de este servicio, primero se observó el cogido HTML y se descubrió que el resultado que se buscaba se encontraba dentro de la primera etiqueta que aparecía con nombre `div[class=trans clickable]`.

Dentro de la etiqueta `div[class=trans clickable]` se encuentran varias etiquetas `li` por ello se recorren todas con *for* hasta el fin de la etiqueta `div[class=trans clickable]`. Cuando se recorren las etiquetas `li` se extrae el texto que las acompaña y se guarda en una variable, para luego separarla por resultados, y mostrarlas con su etiqueta xml o JSON correspondiente.

Cada resultado esta separado por "," por si un resultado esta compuesto por más de una palabra, que a la hora de darle etiqueta XML o JSON no se de una a cada palabra, y se muestre como dos o mas resultados en vez de como un único resultado como debería ser.

En la figura 5.18 podemos ver un diagrama explicativo de como funciona este servicio. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

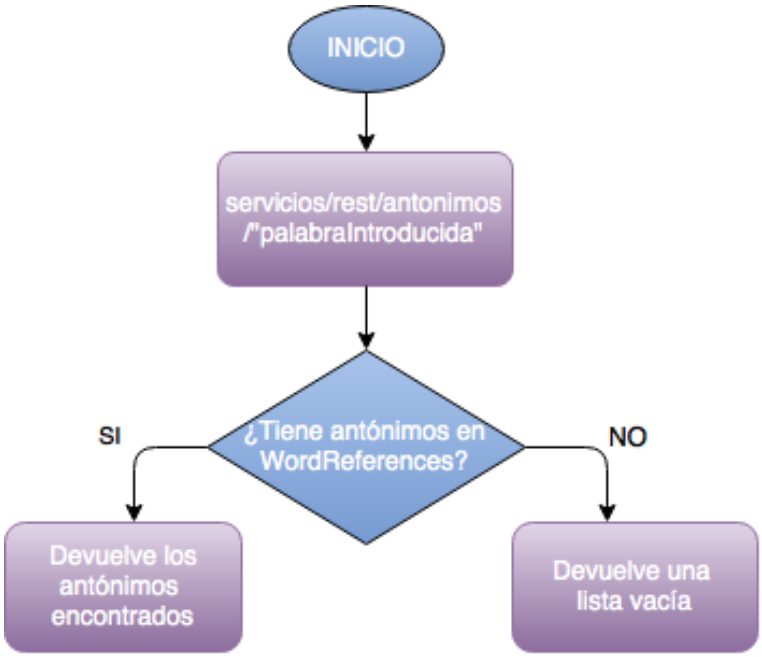
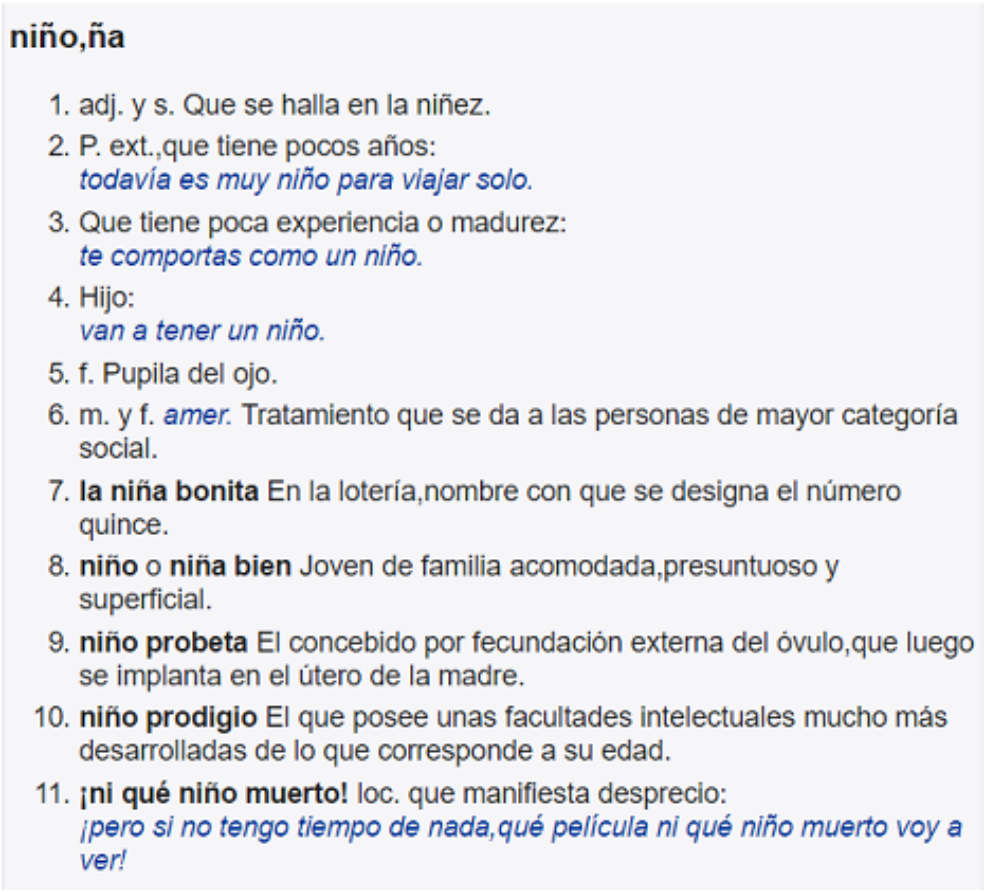


Figura 5.18: Diagrama explicativo del servicio antónimos.

5.6. Servicio definiciones

Este servicio se encarga de mostrar las definiciones de la palabra introducida por el usuario. Para este, de nuevo, hemos recurrido a la página de WordReference [4] para obtener la definición, no siendo esta desde el principio la primera opción considerada para llevar a cabo este servicio, dudándose en usar el de definiciones de ElPaís [5] pero resultando imposible obtener un resultado satisfactorio de este se acabó recurriendo a la página de WordReference.

En la figura 5.19 podemos ver un ejemplo de consulta a la pagina web WordReference con la llamada:
http://www.wordreference.com/definicion/niño.



niño,ña

1. adj. y s. Que se halla en la niñez.
2. P. ext.,que tiene pocos años:
   todavía es muy niño para viajar solo.
3. Que tiene poca experiencia o madurez:
   te comportas como un niño.
4. Hijo:
   van a tener un niño.
5. f. Pupila del ojo.
6. m. y f. *amer.* Tratamiento que se da a las personas de mayor categoría social.
7. **la niña bonita** En la lotería,nombre con que se designa el número quince.
8. **niño** o **niña bien** Joven de familia acomodada,presuntuoso y superficial.
9. **niño probeta** El concebido por fecundación externa del óvulo,que luego se implanta en el útero de la madre.
10. **niño prodigio** El que posee unas facultades intelectuales mucho más desarrolladas de lo que corresponde a su edad.
11. **¡ni qué niño muerto!** loc. que manifiesta desprecio:
    ¡pero si no tengo tiempo de nada,qué película ni qué niño muerto voy a ver!

Figura 5.19: Ejemplo de la llamada a definición de WordReference con "niño".

La primera página que se analizaba para este servicio era la de WordRe-

[4] `http://www.wordreference.com/definicion/`
[5] `http://servicios.elpais.com/diccionarios/castellano/`

ference de definiciones, devolviendo los resultados obtenidos en este servicio. Inicialmente los estos quedaban mostrados íntegros, tal cual los devolvía el WordReference, pero finalmente se optó por descartar partes de la definición ya que no aportaban mayor interés como frases hechas y coloquialismos.

Este servicio contiene el método `getDefinicion()` que se encarga de recoger la información de la página WordReference y de analizar el XML recogido de la misma. Este XML está al alcance de cualquiera que quiera visualizarlo, dando en la página al botón derecho, inspeccionar código. En la figura 5.20 podemos ver recuadradas aquellas etiquetas que contienen el contenido que nos interesa.

```
<br><span class=small1>Diccionario de la lengua española &copy; 2005 Espasa-Calpe:</
onclick='redirectWR(event,"ESdefinicion")' class='trans clickable'><h3>niño,ña </h3>
<li>adj. y s. Que se halla en la niñez. <li> P. ext.,que tiene pocos años:<br><span
muy niño para viajar solo.</span> <li> Que tiene poca experiencia o madurez:<br><spa
comportas como un niño.</span> <li> Hijo:<br><span class=i>van a tener un niño.</spa
del ojo. <li> m. y f.<span class=i> amer.</span> Tratamiento que se da a las persona
categoría social. <li> <span class=b>la niña bonita</span> En la lotería,nombre con
número quince. <li> <span class=b>niño</span> o <span class=b> niña bien</span> Jove
acomodada,presuntuoso y superficial. <li> <span class=b>niño probeta</span> El conce
fecundación externa del óvulo,que luego se implanta en el útero de la madre. <li> <s
prodigio</span> El que posee unas facultades intelectuales mucho más desarrolladas d
corresponde a su edad. <li> <span class=b>¡ni qué niño muerto!</span> loc.<span clas
manifiesta desprecio:<br><span class=i>¡pero si no tengo tiempo de nada,qué película
muerto voy a ver!</span><br></ol></div><div id='FTintro'>'<b>niño</b>
```

Figura 5.20: Etiquetas de interés del código de definición en WordReference.

En el método `getDefinicion()` analizamos el XML devuelto de la página gracias a la conexión que se hace con ella previamente, utilizando la librería JSOUP.

Después de esto, se recorre todos los elementos que tengan la etiqueta <ol>, para así obtener sólo la información que se quiere mostrar. A su vez, en cada elemento con la etiqueta se recorren todas las etiquetas <li>. En cada elemento con la etiqueta recorrer los elementos se eliminan las etiquetas . Estas contienen información que no se quiere mostrar como coloquialismos o frases hechas, ejemplo, etc.

Por último se reemplazan los ":", ": ." por "." para que la frase tenga sentido. Este método devuelve un `ArrayList` con todas las definiciones obtenidas para la palabra buscada.

Este servicio, como el resto, tiene dos formatos distintos posibles para devolver el resultado, XML y JSON. A continuación se ve un ejemplo de cada uno de los formatos de las llamadas.

La llamada JSON sería:

http://sesat.fdi.ucm.es:8080/servicios/rest/definicion/json/luz

El resultado de esta llamada queda representado en la imagen 5.21 que se muestra más abajo.

{"definiciones":[{"definicion":"f. Energia que hace visible todo lo que nos rodea."},{"definicion":"Claridad que irradian los cuerpos en combustion, ignicion o incandescencia."},{"definicion":"Utensilio que sirve para alumbrar."},{"definicion":"Corriente electrica."},{"definicion":"Cada una de las aberturas por donde se da luz a un edificio. Mas en pl."},{"definicion":"pl. Inteligencia."}]}

Figura 5.21: Llamada a definición en formato JSON de la palabra "luz".

En cuanto a la llamada XML el formato de la URL para llamarlo sería de la siguiente forma:

http://sesat.fdi.ucm.es:8080/servicios/rest/definicion/xml/luz

El resultado de esta llamada se puede ver en la imagen 5.22


```
▼<definiciones>
   <definicion>f. Energía que hace visible todo lo que nos
   rodea.</definicion>
  ▼<definicion>
     Claridad que irradian los cuerpos en combustión,
     ignición o incandescencia.
   </definicion>
   <definicion>Utensilio que sirve para alumbrar.
   </definicion>
   <definicion>Corriente eléctrica.</definicion>
  ▼<definicion>
     Cada una de las aberturas por donde se da luz a un
     edificio. Más en pl.
   </definicion>
   <definicion>pl. Inteligencia.</definicion>
 </definiciones>
```


Figura 5.22: Llamada a definición en formato XML de la palabra "luz".

En la imagen 5.23 se puede ver un diagrama de como funciona el servicio de lema. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

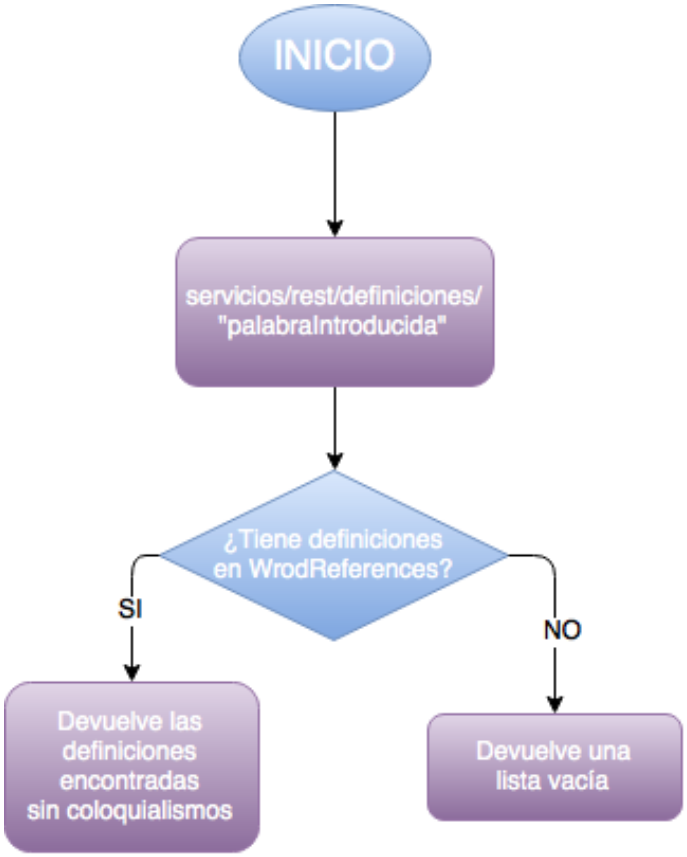


Figura 5.23: Diagrama explicativo del servicio definiciones.

5.7. Servicio traducir a inglés

Este servicio se encarga de devolver la traducción al inglés de la palabra introducida por el usuario. Para ello la idea inicial era seguir el formato llevado a cabo tanto en sinónimos como en antónimos, analizando el xml de una página que ya realizara dicha labor, para lo cual se procedió a analizar el XML de WordReference de traducción a inglés [6] usando DOCUMENT, lo cual devolvía únicamente la última palabra traducida que encontraba en la página, resultando insuficiente para tener un servicio de traducción completo.

Buscando idea en servicios de los proyectos de años anteriores se observó que en el proyecto "Apoyo a la simplificación de textos sobre nevegadores web (proyecto navega fácil)" González y Ortiz (2013 -2014) utilizaba una librería de traducción de Google que haciendo una llamada a la API de Google se podía conseguir la traducción e incluso la pronunciación de una palabra o frase en diferentes idiomas, lo cual suponía una solución de lo más

[6] `http://www.wordreference.com/es/en/translation.asp?spen=`

razonable para el servicio. Sin embargo hace unos años que Google cobra por este servicio por lo que esta idea fue descartada.

Finalmente se procedióa utilizar la página de Span¡shD!ct[7] cuyo código fuente tiene la información en un XML de una forma estructurada y sencilla de analizar para ser tratado por este servicio y descartar la información del mismo que no resultara relevante.

Como resultado en este servicio se devuelven las traducciones de la palabra introducida sin entrar en detalle de a que grupo de palabra pertenece (sustantivo, adjetivo, etc).

En las imágenes 5.24 y 5.25 se pueden ver los ejemplos de los resultados obtenidos con XML y JSON aplicados a la palabra "andar".

- Formato XML:
  <traducciones>
  <traduccion>palabraTraducida1</traduccion>
  <traduccion>palabraTraducida2</traduccion>
  </traducciones>

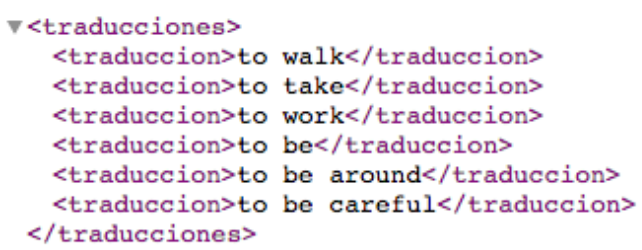


Figura 5.24: Ejemplo de traducción en XML

- Formato JSON:
  {''traducciones'': [{''traduccion'':''palabraTraducida1''},
  {''traduccion'':''palabraTraducida2''}, ...]}

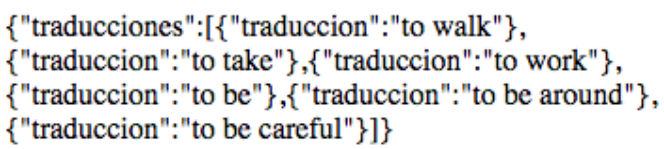


Figura 5.25: Ejemplo de traducción JSON

[7] http://www.spanishdict.com/traductor/

En la imagen 5.26 queda explicado con un diagrama sencillo como funciona este servicio. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

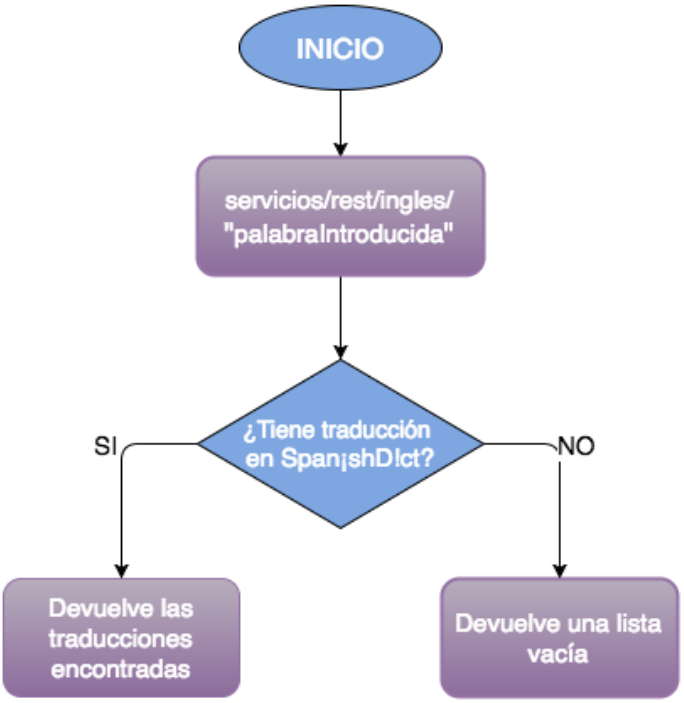


Figura 5.26: Diagrama explicativo del servicio de traducción a Inglés.

5.8. Servicio lema

Servicio que devuelve el origen y la categoría gramatical de la palabra buscada. Este servicio puede ser accedido para devolver un formato XML o JSON, ambas llamadas quedan expuestas a continuación:

- *http://sesat.fdi.ucm.es:8080/servicios/rest/lema/xml/"palabraBuscada"*
- *http://sesat.fdi.ucm.es:8080/servicios/rest/lema/json/"palabraBuscada"*

En la imagen 5.27 se puede ver un ejemplo de invocación al servicio en formato XML.

Para llevar a cabo este servicio se recoge la información de la página "Busca palabra"[8] la cual te devuelve el origen y la categoría gramatical de la palabra buscada.

[8] `http://www.buscapalabra.com/categoria-gramatical-tiempo-verbal.html`

```
▼<lema>
   <palabra>consumía</palabra>
   <origen>consumir</origen>
  ▼<categoriaGramatical>
     3ª per.sin. del pretérito imperfecto de indicativo,
     verbo transitivo, verbo pronominal, verbo intransitivo,
     1ª per.sin. del pretérito imperfecto de indicativo
   </categoriaGramatical>
 </lema>
```

Figura 5.27: Ejemplo de llamada al servicio lema con la palabra "consumía" en XML.

En la imagen 5.28 podemos ver un ejemplo de resultado devuelto al consultar en la página web de "Busca palabra" la palabra "consumía". En ella se puede ver como se muestra el origen de la palabra, en este caso "consumir", así como las categorías gramaticales a las que puede pertenecer.

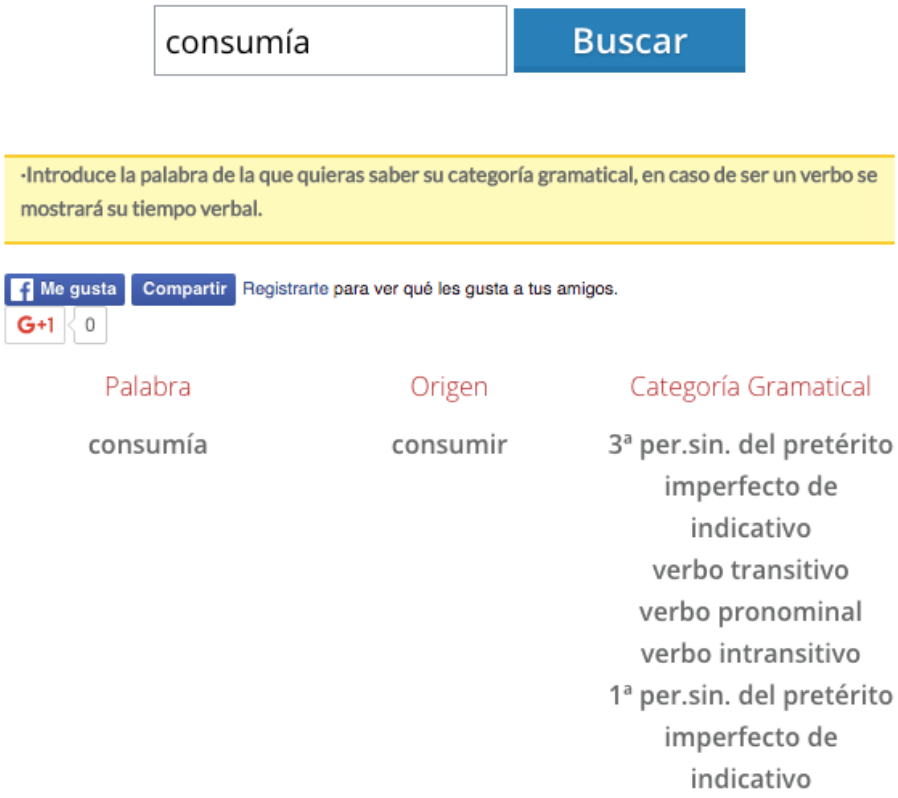


Figura 5.28: Ejemplo de consulta a "Buscar palabra".

Este servicio resulta útil, no solo para aportar información de una palabra dada, si no como herramienta para otros servicios como puede ser pictograma, que no requieran una palabra exacta para su funcionamiento, pudiendo usar el origen de la misma para mayor simplificación del problema.

En la imagen 5.29 se puede ver un diagrama de como funciona el servicio de lema.

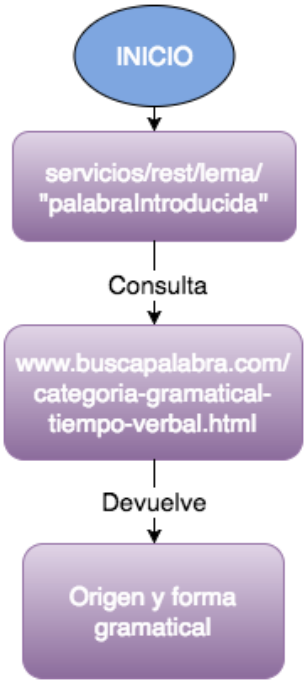


Figura 5.29: Diagrama explicativo del servicio lema.

El desarrollo de este servicio se realizó en la etapa final del proyecto por lo que, por falta de tiempo, no está del todo comprobado y desarrollado, mostrando resultados no del todo correctos en el caso de JSON, y es por eso que no se ha incluido en la aplicación de ejemplo desarrollada en Android.

5.9. Servicio convertir a palabra sencilla

El objetivo de este servicio es devolver una palabra sencilla que pertenezca a la lista de sinónimos de la palabra introducida por el usuario. Es decir, si se introduce una palabra difícil, este servicio buscará en la lista de sinónimos de la palabra introducida hasta dar con uno que al aplicarle el servicio "Palabra fácil" te devuelva *true*.

Lo primero que se hace en este servicio es llamar al visto con anterioridad de palabras sencillas (llamado palabrasSencillas) con la forma:

http://localhost:8080/servicios/rest/palabras/xml/"palabraBuscada"

que, como se ha visto antes, devuelve un *booleano* con *true* si es una palabra fácil, es decir, si se encuentra en el archivo 1000_formas.TXT y si no la encuentra devuelve *false*. Si al aplicarle a la palabra el servicio de palabrasSencillas se obtiene *true*, entonces el servicio conversión devuelve la misma palabra introducida. Si la palabra no se encuentra en el fichero quiere decir que es una palabra difícil, y hay que intentar buscar un sinónimo de esa palabra que sea sencillo. Para ello se realiza una consulta al servicio de sinónimos:

http://localhost:8080/servicios/rest/sinonimos/xml/"palabraBuscada".

Posteriormente se analiza el xml que devuelve el servicio de sinónimos con el objetivo de quedarse sólo con un String con los sinónimos analizados separados por ",", y buscar el primer sinónimo que sea una palabra sencilla dentro de esa lista, invocando nuevamente al servicio de palabrasSencillas. Si hay algún sinónimo que sea palabra sencilla, el servicio lo devolverá como resultado obtenido y, en caso contrario, devolverá la misma palabra sobre la que se ha realizado la consulta, la palabra inicial buscada por el usuario *"palabraBuscada".*

El hecho de separar los Strings por "," se debe a que hay sinónimos compuestos, es decir, que contienen más de una palabra, por lo que si se hace la división por espacios se puede coger una de las palabras de ese sinónimo y devolverse como sinónimo fácil de la misma, sin necesidad de que este lo sea. A continuación se pasa a explicar un ejemplo que aclara esta situación. Supongamos que se hace la llamada:

http://sesat.fdi.ucm.es:8080/servicios/rest/conversion/xml/comer

el resultado obtenido analizado sin "," era *<conversion>de</conversion>*, lo cual no es un sinónimo de la palabra "comer". Esto ocurre porque "comer" tiene un sinónimo que es *"Llenar de andorga"* y al separar por espacios coge "de" como sinónimo de "comer", sacado de tratar "llenar", "de" y "andorga" como tres palabras distintas, siendo "de" la palabra sencilla resultante. Esto se soluciona separando los sinónimos por ",", por lo que al hacerlo de este modo si se hace la llamada:

http://sesat.fdi.ucm.es:8080/servicios/rest/conversion/xml/comer

devuelve *<conversion>comer</conversion>*, ya que no encuentra una palabra fácil que sea sinónimo de "comer". En las imágenes 5.30 y 5.31 se pueden ver los resultados obtenidos al realizar las llamadas respectivas en XML y JSON:

http://sesat.fdi.ucm.es:8080/servicios/rest/conversion/xml/rebuscar

```
<conversion>buscar</conversion>
```

Figura 5.30: Ejemplo de llamada de conversion con la palabra "Rebuscar" en XML.

http://sesat.fdi.ucm.es:8080/servicios/rest/conversion/json/rebuscar

```
{"conversion":"buscar"}
```

Figura 5.31: Ejemplo de llamada de conversion con la palabra "Rebuscar" en XML.

Este servicio se creó, además de como servicio en si mismo, como herramienta para una demostración de funcionalidad en una aplicación Android de los servicios realizados. Así se mostraba, en una sola función, el resultado de aplicar varios de los servicios, obteniendo una utilidad fácilmente aplicable a un servicio de accesibilidad, el simplificado de palabras, y por extensión de textos.

Para terminar de aclarar como funciona este servicio en la imagen 5.32 se muestra un diagrama de flujo en el que queda explicado como se procede en los distintos casos. En este no queda especificado el formato xml o json para hacerlo genérico para ambos, ya que funcionan de forma similar.

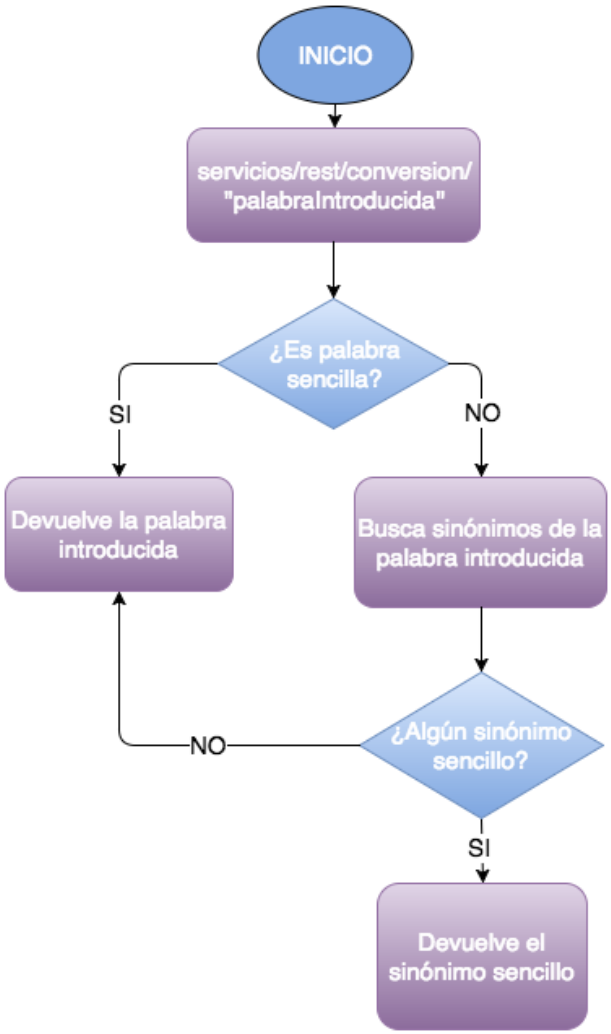


Figura 5.32: Diagrama explicativo del servicio de conversión

Capítulo 6

Desarrollo de la página Web de representación de la API

En esta sección se pasará a hablar del desarrollo de la página web que funciona como API de los servicios anteriormente explicados. En esta quedarán explicados y a disposición del público los servicios, así como se indicarán las URLs para acceder a ellos y un pequeño ejemplo de llamada y su resultado. Todos los servicios mencionados en el capítulo 5 tienen un apartado en la web cuya finalidad es explicar su funcionalidad y modo de uso.

En cuanto al diseño de la web es directo y auto explicativo en sus elementos, es decir, cada funcionalidad de la web viene definida por una breve descripción que resume lo que realiza. Además posee una estructura minimalista, teniendo únicamente los elementos necesarios para que quede claro su funcionamiento y así poder acceder a los servicios ofrecidos. Esto queda explicado más implícitamente en las secciones que veremos a continuación.

6.1. Página principal

Como se puede observar en la imagen 6.1 la pantalla inicial consta de un panel superior con el título de la web y el escudo de Informática, un panel debajo de este con un pequeño mensaje explicativo de la web y por último, siete iconos con el nombre de cada uno de los servicios que están desarrollados. Cada uno de esos siete iconos tiene un enlace al servicio que indica su nombre, por ejemplo Sinónimo te llevará a la API de sinónimos, en la cual queda detallado este servicio y la forma de invocarlo.

Como se explicaba anteriormente, la página principal es minimalista, al disponer únicamente de los elementos necesarios para acceder a cada uno de los servicios. Cada uno de ellos se ha ido añadiendo a medida que se iba incorporando en el servidor. A continuación pasamos a describir brevemente cada uno de ellos.

Figura 6.1: Página inicial de la API

- **Dificultad de palabras:** indicaciones de como utilizar el servicio que te devuelve si la palabra introducida es fácil o no devolviendo *true* o *false* respectivamente.
- **Sinónimos:** indicaciones de como utilizar el servicio que te devuelve los sinónimos encontrados de la palabra introducida.
- **Antónimos:** indicaciones de como utilizar el servicio que te devuelve los antónimos de la palabra introducida.
- **Definición:** indicaciones de como utilizar el servicio que te devuelve las definiciones de la palabra introducida.
- **Traducción inglés:** indicaciones de como utilizar el servicio que te devuelve las distintas traducciones de la palabra introducida.
- **Pictogramas:** indicaciones de como utilizar el servicio que te devuelve los pictogramas de la palabra o frase introducidas. Se entiende como

pictograma una imagen que esta relacionado con el objeto al que se refiere.

- **Conversión fácil:** indicaciones de como utilizar el servicio que te devuelve un sinónimo fácil de la palabra introducida. Si esta fuera inicialmente fácil devolvería la palabra introducida.

6.2. Páginas secundarias

Se entienden como páginas secundarias aquellas en las que quedan explicados cada uno de los servicios y las URL de acceso a los mismos, con sus ejemplos de utilización. A continuación pasaremos a describir los componentes que las forman y la funcionalidad y el estilo seguidos en cada uno de ellos.

- **Panel superior:** En este queda indicada la ruta en la que el usuario se encuentra en ese momento, es decir, si el usuario se encuentra en el servicio "Sinónimos" la ruta mostrada en el panel superior sería "Inicio /Sinónimos", ya que la página principal o inicio es considerada la página padre. Además ambos elementos en la ruta son seleccionables, tanto "Inicio" como, en este caso, "Sinónimos", llevando a la página principal en caso de "Inicio". Además, a la izquierda del todo aparece el escudo de Informática como logo de la página web. En la imagen 6.2 se puede ver un ejemplo de lo anteriormente explicado.

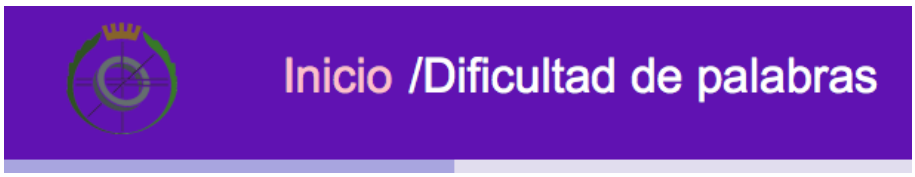


Figura 6.2: Panel superior con feedback en el posicionamiento sobre Inicio.

- **Panel izquierdo:** En él se pueden encontrar enlaces a todas las APIs de todos los servicios disponibles. Seleccionando cualquiera de ellos se accede a su API correspondiente en la cual se pasará a detallar el servicio. En este panel izquierdo, cuando se sitúa el cursor encima de alguno de los nombres de los servicios, el nombre cambia de color dándole así un feedback al usuario de que el elemento es seleccionable y que producirá una respuesta si se pincha sobre él. En la imagen 6.3 se puede ver el panel izquierdo y un ejemplo del feedback anteriormente mencionado.

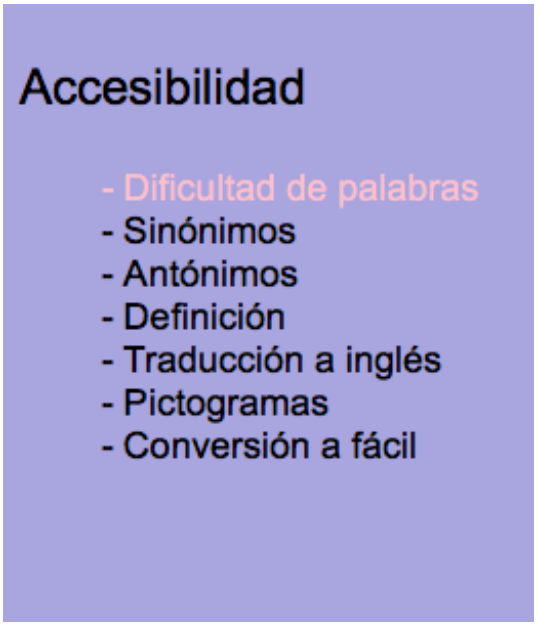


Figura 6.3: Panel izquierdo con feedback en posicionamiento de servicio.

- **Cuerpo:** En el cuerpo se encuentra toda la información necesaria referente al servicio del que se habla. Dado que el cuerpo esta dividido en varios elementos a su vez, se verá cada uno de estos por separado, según el orden en el que aparecen en la web.

  - ***Descripción:*** Lo primero que se muestra es un texto con la descripción de la funcionalidad del servicio, indicándose lo que este espera recibir como parámetro así como lo que devolverá como resultado. Este queda encabezado con el nombre del servicio al que describe. En la imagen 6.4 se puede ver un ejemplo de descripción del servicio "Comprobar palabra".

  - ***Llamadas URLs:*** Tras la descripción anteriormente mencionada se muestran las distintas opciones de llamadas a la URL. Estas son similares para todos los servicios salvo el de pictogramas, dos versiones distintas de llamada a URL, una para devolver el mensaje en formato JSON y la otra en XML. En el caso de pictogramas hay también dos opciones, pero en este caso no hacen referencia al formato de salida, ya que es siempre una o varias imágenes, sino a si se va a introducir una palabra o una frase a convertir a pictograma. En las imágenes 6.5 y 6.6 podemos ver ejemplo de cada uno de los casos mencionados anteriormente.

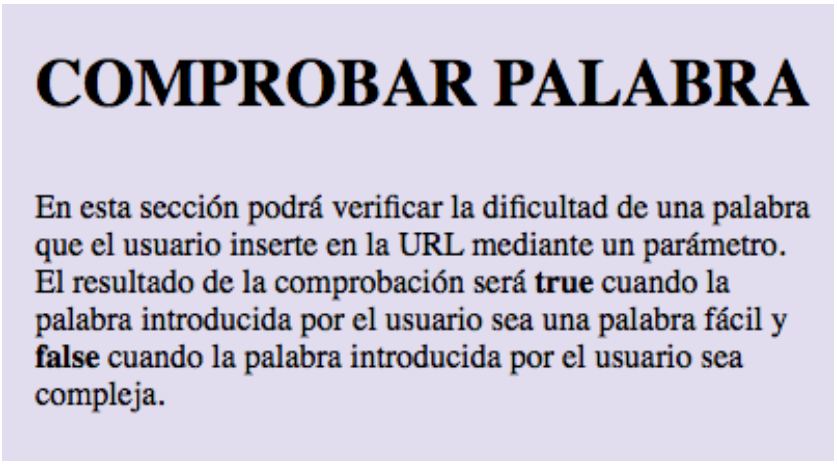


Figura 6.4: Ejemplo de descripción del servicio "Comprobar palabra".

Figura 6.5: Ejemplo de como llamar a las URLs de "Dificultad de palabra".

Figura 6.6: Ejemplo de muestra de como llamar a las URLs de "Pictograma".

- ***Información del recurso:*** Después de las llamadas a las URL aparece una pequeña tabla que te da información del servicio, en cuanto a formato que devuelve y si requiere autentificación o no. En la imagen 6.7 se ve un ejemplo.

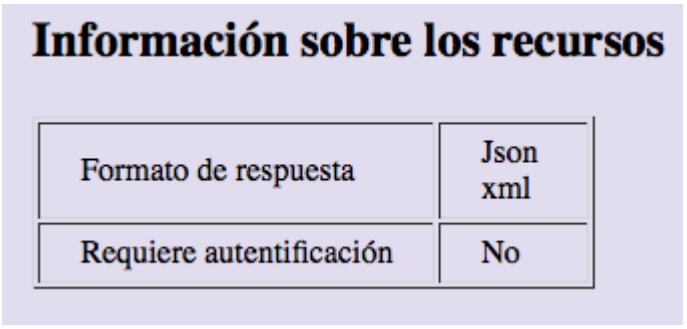


Figura 6.7: Ejemplo de tabla de información.

- ***Parámetros:*** En esta sección se especifica los parámetros que se han de introducir en la URL, tanto su forma como su funcionalidad. En el caso de pictogramas se tienen dos opciones distintas en función de si es frase o palabra, como se puede ver en la imagen 6.8. En el resto de servicios es "palabraIntroducida" a la cual se le aplica el servicio, de forma similar al formato de palabra en el servicio de convertir a pictograma.

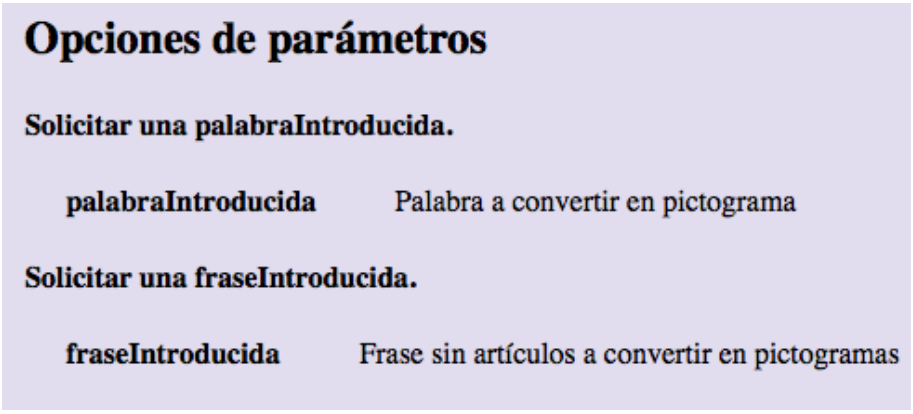


Figura 6.8: Ejemplo de parámetros.

- ***Ejemplos:*** Por último se muestran ejemplos de llamadas para cada servicio junto con su correspondiente resultado, en las distintas opciones que el servicio posee. Así en pictogramas se muestra un ejemplo de llamada para frase y otro para palabra y en el resto de servicios uno para XML y otro para JSON. En la imagen 6.9 se muestra un ejemplo de lo mencionado en este punto.

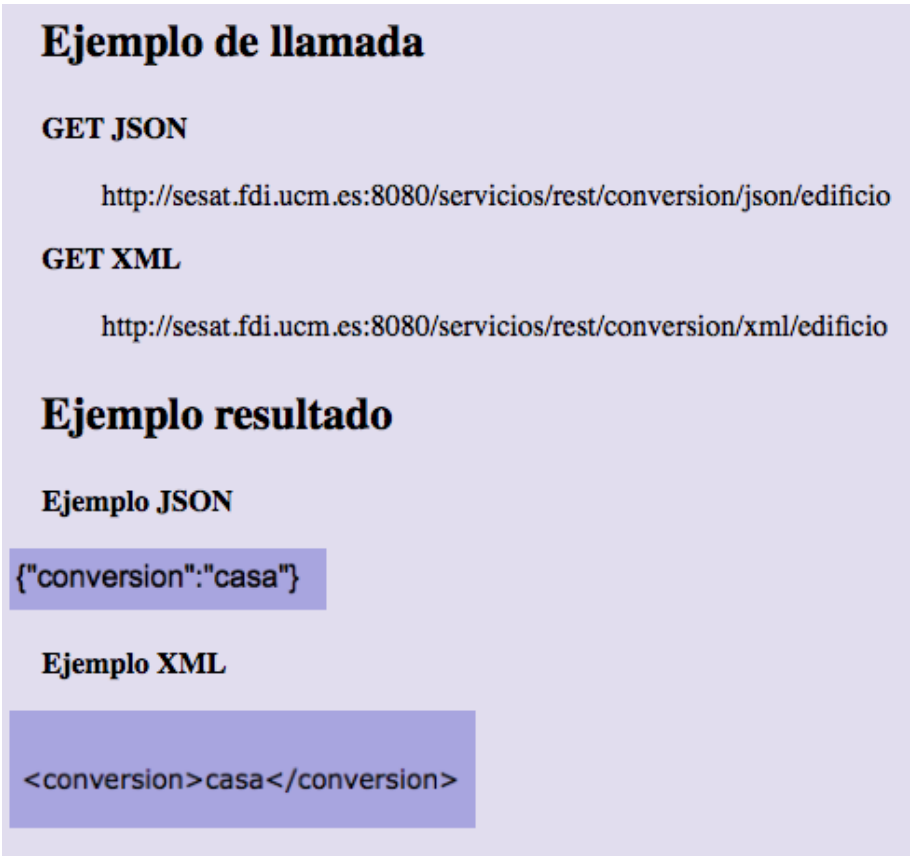


Figura 6.9: Ejemplo de llamadas a las URLs.

Capítulo 7

Desarrollo de una aplicación de ejemplo de los servicios

Tras la implementación de lo servicios se crea la aplicación Android para realizar ejemplos de utilización de los mismos. Cabe destacar en este contexto que existen dos herramientas bastante conocidas para la implementación de aplicaciones Android, el Android de Eclipse y Android Studio. Debido a que Eclipse dejo de dar soporte a comienzos del 2016 a Android esta aplicación será desarrollada en Android Studio.

Al ser una aplicación enfocada en la accesibilidad tendrá un diseño sencillo para facilitarle al usuario la utilización de la misma. Para que el diseño sea sencillo se evitará sobrecargar cualquiera de las pantallas de las que disponga la aplicación, dejando solo los elementos imprescindibles, e indicando claramente cual es la funcionalidad de cada uno de estos elementos. Esto se verá más adelante en 7.1, donde se tratará la interfaz gráfica de la aplicación.

La aplicación está pensada para usarse en una tablet Android por lo que el emulador que se usa en AndroidStudio es uno que cumple con estas características en apariencia, concretamente un Nexus 7.

El motivo por el cual se utiliza una tablet en lugar de un dispositivo móvil es que, al ser una aplicación centrada en la accesibilidad, una tablet resulta mucho más sencilla de utilizar a la hora de interaccionar con la interfaz gráfica, ya que todos los iconos son más grandes y visibles, facilitando así el manejo de nuestras opciones. En la imagen 7.4 podemos ver como queda la pantalla inicial y hacernos una idea de la facilidad de manejo que supone la misma en la pantalla de una tablet, de tamaño más grande.

7.1. Interfaz gráfica de la aplicación.

Antes de comenzar a hablar de cómo ha quedado implementada la aplicación nos centraremos en la interfaz gráfica de la misma, para ya estar familiarizados posteriormente con el diseño visual cuando pasemos a definir sus funcionalidades.

Una de las ventajas de trabajar con Android Studio es que facilita el trabajo a la hora de realizar el diseño ya que te muestra una imagen con un teléfono móvil y a su izquierda una paleta donde se encuentran todos los posibles elementos a añadir en la interfaz gráfica.

Esto hace que la tarea del diseño de la aplicación sea mucho más cómoda ya que para añadir cualquiera de estos elementos tan solo hay que arrastrarlos de la paleta a la pantalla del móvil y colocarlos en la posición que se quiere que aparezcan a la hora de ejecutar la aplicación, sin necesidad de tener que estar implementando cada elemento, posición o acción.

La aplicación consta de dos pantallas cada una con su interfaz gráfica diferente. A continuación describiremos cada una de estas pantallas y los distintos elementos gráficos que las componen.

- **Pantalla principal:** en esta se encuentran todos los servicios que la aplicación ofrece, en forma de botones, y un panel en el que escribir la palabra a la que se le quiere aplicar el servicio. Una vez seleccionado el botón correspondiente al servicio e introducida la palabra este nos llevara a la pantalla secundaria que describimos más adelante. Los servicios ofrecidos son los que hemos visto anteriormente en el capítulo 5.

    - El nombre de la aplicación que siempre aparece en la parte superior de la misma, con el formato mostrado en la imagen 7.1

Figura 7.1: Cabecera de la aplicación Android

    - Un TextView, donde aparece el mensaje “Escribe aquí la palabra:” indicándole al usuario la funcionalidad que tiene el elemento posterior, para que no haya lugar a confusión.
    - Un EditText donde el usuario introduce la palabra que se le pide, a la que se le aplicará el servicio que se seleccionará posteriormente en los botones.

En la imagen 7.2 podemos ver el TextView y el EditText anteriormente mencionados.

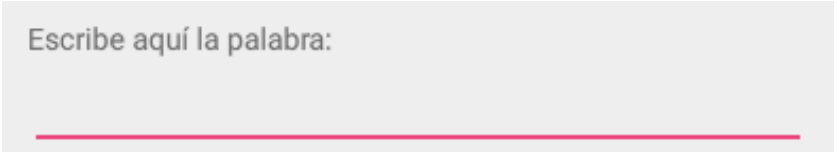


Figura 7.2: Texto de petición de palabra y el espacio para introducirla.

- Varios Buttons que son los botones donde están representados los distintos servicios ofrecidos por la aplicación, todos ellos seleccionables y con una respuesta diferente. En la imagen 7.3 podemos apreciar la disposición de los mismos.

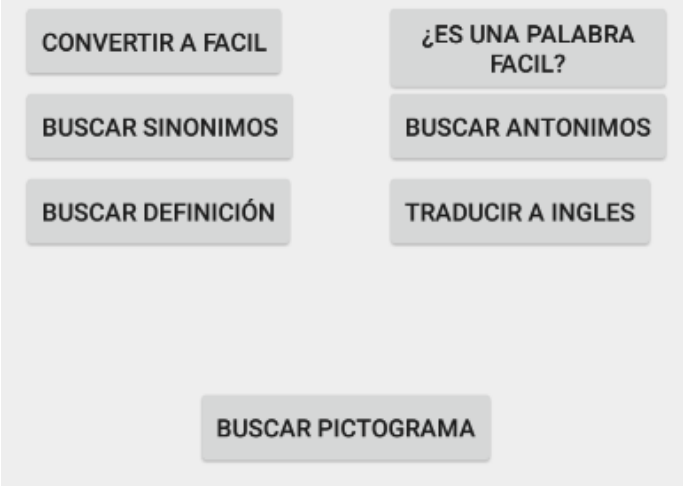


Figura 7.3: Botones con los servicios disponibles.

En la imagen 7.4 se puede observar el resultado al completo de la pantalla principal de la aplicación android, y con la cual interactuará el usuario.

- **Pantalla secundaria:** los resultados mostrados en esta pantalla dependen del servicio seleccionado en la pantalla principal, aunque el diseño gráfico seguido es similar en todos los casos, variando el mensaje y resultado mostrados.

  - Palabra fácil: esta pantalla tiene dos posibilidades de mensajes diferentes:

Figura 7.4: Pantalla principal de la aplicación Android para tablet

"La palabra *palabraIntroducida* si es una palabra sencilla" o "'La palabra *palabraIntroducida* no es una palabra sencilla".

En la imagen 7.5 podemos ver un ejemplo de resultado obtenido.

- Buscar sinónimos: en esta pantalla dependiendo de si se encuentran sinónimos de la palabra o no, se muestran los siguientes mensajes respectivamente:
"Sinónimos de *palabraIntroducida* : *todos los sinónimos mostrados en una lista*" o "No existen sinónimos de *palabraIntroducida*".

  En la imagen 7.6 se puede ver un ejemplo del resultado obtenido en este servicio.

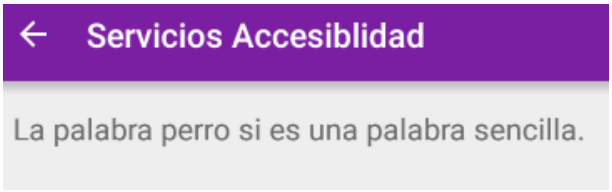


Figura 7.5: Ejemplo de resultado de la palabra sencilla “perro”

Figura 7.6: Ejemplos de sinónimos obtenidos de casa

- Buscar antónimos: en esta pantalla, dependiendo de si se encuentran antónimos de la palabra o no, se muestran los siguientes mensajes respectivamente:
  “Antónimos de *palabraIntroducida* : *todos los antónimos mostrados en una lista*” o “No existen antónimos de *palabraIntroducida*”.

  Dado que este servicio es similar al de sinónimos mostramos en este caso la pantalla en el caso de que no se encuentren antónimos en la figura 7.7

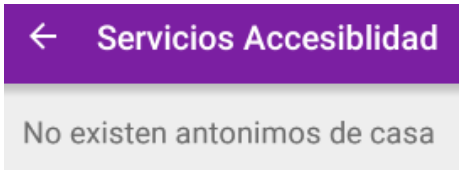


Figura 7.7: Ejemplos de sinónimos obtenidos de "casa"

- Buscar definición: en esta pantalla se muestran las definiciones encontradas para la palabra introducida. El mensaje mostrado en la pantalla es:
  "Definiciones de *palabraIntroducida* : *todas las definiciones de la palabra devueltos por el servicio*".
  En la imagen 7.8 vemos un ejemplo de las definiciones obtenidas para gato.

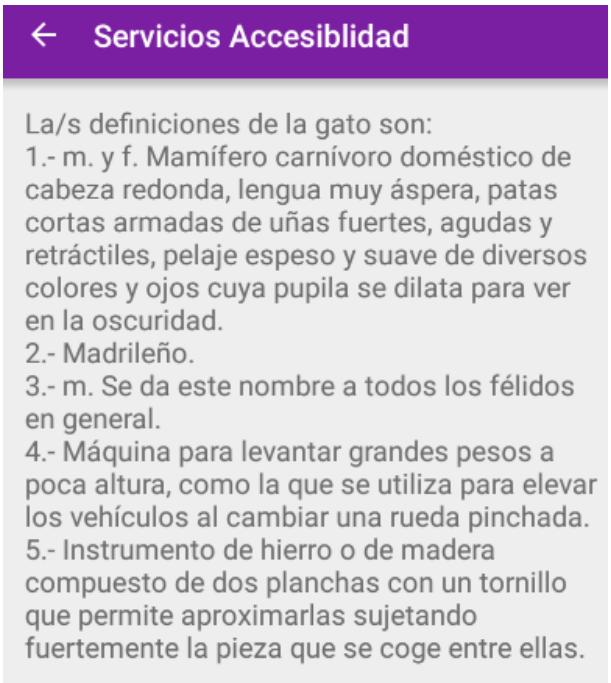


Figura 7.8: Ejemplos de definiciones obtenidas de "gato"

- Convertir a fácil: en esta pantalla se muestra un sinónimo sencillo de la palabra introducida. Tiene dos posibilidades a la hora de mostrar el resultado, que muestre un sinónimo de la palabra que sea sencillo o que te muestre la misma palabra introducida en caso de no encontrar sinónimos sencillos.
  En la imagen 7.9 podemos ver un ejemplo de conversión de "hogar" a palabra sencilla.

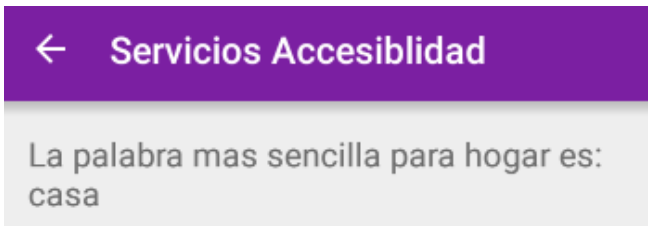


Figura 7.9: Ejemplo de conversión de "hogar" a palabra sencilla

- Traducir a inglés: en esta pantalla se muestra la traducción de la palabra al inglés. El mensaje que aparece en la pantalla es el siguiente: "Traducciones de *palabraIntroducida* : *todas las traducciones devueltas de la palabra devueltos por el servicio*".

  En la figura 7.10 podemos ver un ejemplo de traducción de casa.

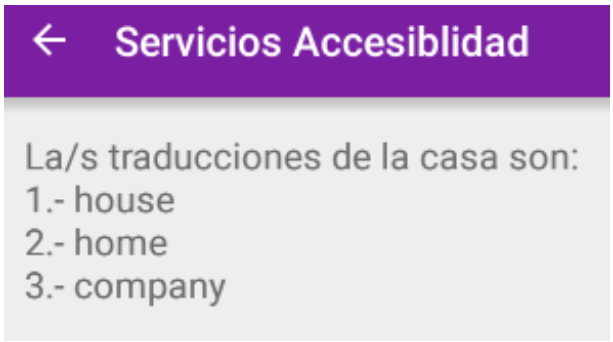


Figura 7.10: Ejemplo de traducción de "casa" al inglés

- Buscar pictograma: en esta pantalla se muestra el pictograma correspondiente a la palabra introducida. En la imagen 7.11 podemos ver un ejemplo de resultado al introducir "casa" en la pantalla principal y pulsar el botón "Buscar Pictograma".

Cómo se puede ver en las fotografías de la pantalla secundaria al lado del nombre de la aplicación aparece una flecha, cuando el usuario hace clic en esta flecha, la aplicación vuelve a la pantalla inicial donde está la opción de introducir una palabra nueva y todos botones.

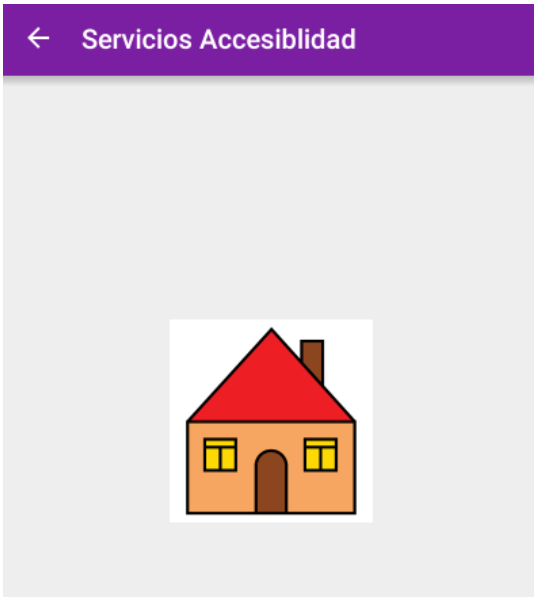


Figura 7.11: Ejemplo de pictograma de "casa"

7.2. Elementos de implementación del diseño

En cuanto a implementación en Android Studio hay dos partes importantes:

- la carpeta dónde se alojan los archivos los .java, en la cual se encuentra el código desarrollado para las funcionalidades de la aplicación.
- la carpeta en la que están los .xml, donde queda implementada la interfaz gráfica y todas sus propiedades.

En esta sección se hablará del contenido de la carpeta con archivos .xml, dond ese trata la implementación del diseño, viéndose en 7.3 la de las funcionalidades.

Cada clase .java va acompañada de su archivo .xml.

En cuanto a diseño, en las dos primeras aplicaciones de prueba se utilizó un elemento llamado *LinearLayout* y luego, tras probar distintas opciones, se acabó usando en la aplicación de servicios el *RelativeLayout*. Dentro de estos Layouts, en la aplicación han sido utilizados los siguientes elementos:

- **TextView:** se usa este elemento para mostrar todos los textos de la aplicación. Hay uno implementado para la pantalla principal y otro para la secundaria. Este elemento tiene una propiedad que hace que el texto pueda ser editable, pero no queda habilitada en la aplicación ya que este elemento lo se usa únicamente para mostrar texto, sin querer que este pueda ser modificado.

- **EditText:** dado que se requiere que el usuario introduzca la palabra a tratar se ha de utilizar un elemento para ello, siendo este el **EditText**. En este el usuario introduce la palabra a procesar por el servicio y en la pantalla secundaria se muestra el resultado obtenido del proceso.

- **Button:** este elemento representa botones que pueden ser seleccionados por el usuario para realizar determinada acción. En la aplicación se usan los botones para representar cada uno de los distintos servicios ofrecidos.

Dentro de algunos de estos elementos aparecen diferentes opciones. A continuación se detallan algunas de las que han sido utilizadas en la aplicación:

- **android:layout_width / android:layout_height** : Esta función sirve para dar medida al ancho y al largo del layout. Puede contener tres valores diferentes:

  - **match_parent:** El elemento añadido debe ser tan grande como el padre que lo contiene.
  - **wrap_content:** El elemento añadido toma el tamaño del texto que contiene.

  Se le pueden dar los valores con el número seguido de px.

- **android:text:** Es el texto que aparece en el elemento. Se puede definir de dos maneras:

  - Escribiendo directamente el texto
  - Definiendo una constante de la forma *@string/nombreElegido*.

  Más adelante se explicará que se hace con la constante definida como string más el nombre elegido.

- **android:layout_alignParentTop:** Esta función hace encajar el eje superior de este elemento con el eje superior del elemento padre. Esto sucede cuando se iguala a "True".

- **android:inputType:** Sirve para especificar el tipo de contenido que va a tener el elemento.

- **android:layout_below:** Sirve para dar un id a nuestro elemento, y así poder utilizarlo desde el fichero .java.

- **android:layout_centerHorizontal:** Esto sirve para centrar el elemento hijo con el padre en el plano horizontal. Si se iguala a "True", se centrará con el padre en caso contrario no lo hará.

- **android:layout_marginTop:** Sirve para especificar un espacio de rigor en la parte superior y que así los elementos no resulten estar inmediatamente unidos unos a otros, así como que el elemento no empiece directamente en la parte superior de la pantalla o inmediatamente después del elemento padre.

- **android:onClick:** Sirve para hacer una llamada a una función cuando un botón ha sido cliqueado. El nombre de esta función corresponde con el de la función en el .java encargada de ejecutar la funcionalidad asignada a dicho botón

Existen muchos más parámetros que se han ido viendo a lo largo de las distintas pruebas realizadas con Android Studio, pero dado que no se ponen en práctica en esta aplicación no quedarán mencionados aquí.

Una vez descritos los elementos utilizados en la aplicación y sus utilidades, pasamos a describir la implementación de los mismos, contemplando las comunicaciones entre la interfaz gráfica y el código principal.

Empezaremos centrándonos en la clase *MainActivity* que hace de puente entre la interfaz gráfica y el programa principal. Esta clase contiene un método para cada uno de los servicios implementados, todos ellos con una forma similar a la que se observa en la imagen 7.12.

```
public void convertirAfacil(View view) {
    EditText editText = (EditText) findViewById(R.id.TxtPalabra);
    String palabra = editText.getText().toString();
    intent.putExtra("Convertir", palabra);
    startActivity(intent);
}
```

Figura 7.12: Método de la clase MainActivity.

En ese método se observa la linea de código:

```
EditText editText = (EditText) findViewById(R.id.TxtPalabra);
```

con esta línea obtenemos el elemento del EditText para almacenarlo en el string llamado "palabra". Esto se realiza mediante la función
`String palabra = editText.getText().toString();`.
Una vez obtenida la información llamamos al intent:
`intent.putExtra("NombreDelServicio", palabra);`
`startActivity(intent));`.

Para poder recibir la palabra de un **EditText** se tiene que tener declarado en el *activity_ main* con el nombre de TxtPalabra.

Para implementar los diferentes servicios también nos ha hecho falta declararnos los diferentes botones en el mismo activiy_main.xml.

Como se puede observar en la figura, en la declaración del botón se tiene una función que se llama **onCLick** que cuando el usuario hace clic en un elemento (en este caso un botón) y la cual se iguala al nombre de la función que queremos que llame al producirse este clic, que es la que se encuentra en *Main_ Activiy.java*.

Al igual que se tiene un método para cada servicio también se tiene declarado un botón. Realmente este es el comienzo del programa puesto que aquí es donde se declara la interfaz y donde se llama a los primeros métodos.

Además en esta clase se tiene el método **OnCreate** que es llamado cuando se crea la actividad por primera vez.

7.3. Implementación de la aplicación

En esta sección se pasa a hablar de la implementación llevada a cabo para el desarrollo de las funcionalidades de la aplicación en si misma.

En el elemento del tipo **android:text:** que se veía en la sección anterior se pueden definir constantes de la forma: **@string/NombreConstante**, donde "NombreConstante" es el nombre que recibe la constante y con el que se quiere identificar a la misma en el programa.

En este contexto cabe mencionar que Android tiene un archivo llamado *string.xml* donde se declaran todas estas constantes, por lo que si se utiliza esta en varios lugares del programa y en algún momento se decide cambiar el valor de esta constante únicamente habrá que ir a este archivo y cambiarlo ahí, sin tener que modificar cada una de las apariciones de la misma.

A continuación se hablará del archivo *AndroidManifest.xml*. Este es un archivo que todo proyecto Android tiene que tener para su correcto funcionamiento, el cual tiene que tener necesariamente ese nombre, y que se encarga de presentar la información necesaria sobre la aplicación antes de que pueda correr el código del programa. Este archivo está compuesto por tres partes:

- En la primera parte del archivo aparecen la versión, el nombre del proyecto y el nombre del paquete que se está utilizando en la aplicación. En la figura 7.13 se puede ver como quedan representados.

- La segunda parte son los permisos que se necesitan en la aplicación. En este caso se tienen añadidos permisos a Internet y el acceso al estado de la red (Network state). En la figura 7.14 se puede ver como quedan representados.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="servicio1.servicio1">
```

Figura 7.13: Imagen de la versión, nombre de proyecto y nombre del paquete en AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 7.14: Imagen de los permisos necesarios para la aplicación.

- La tercera parte está compuesta por las actividades de la aplicación. Puesto que esta aplicación dispone de dos actividades tan solo habrá que declararse esas dos activities. Además en esta tercera parte se incluyen algunas propiedades como la definición del icono (el que aparecerá en el dispositivo móvil cuando se instale la aplicación), el tema, y si se permite Backup. En la imagen 7.15 se puede ver como todo esto queda definido.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Servicios Accesiblidad"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:name="new_T"
        android:parentActivityName="MainActivity" >
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="MainActivity" />
    </activity>
</application>
```

Figura 7.15: Imagen de las actividades que componen la aplicación.

En este proyecto las Activity extienden a **AppCompatActivity** para poder usar el **ActionBar** en la aplicación. Este **ActionBar** se usa para mostrar el nombre de la aplicación en la parte superior del dispositivo en la

forma en la que se ve en la imagen 7.4, correspondiendo este unicamente a la parte en morado en la cual aparece “Servicios Accesibilidad”.

A continuación se pasa a detallar cada uno de los componentes de la aplicación, en cuanto a clases y paquetes, para lo cual nos guiaremos con la imagen 7.16 en la que se muestra un diagrama general de la estructura y jerarquía de los paquetes y clases básicos.

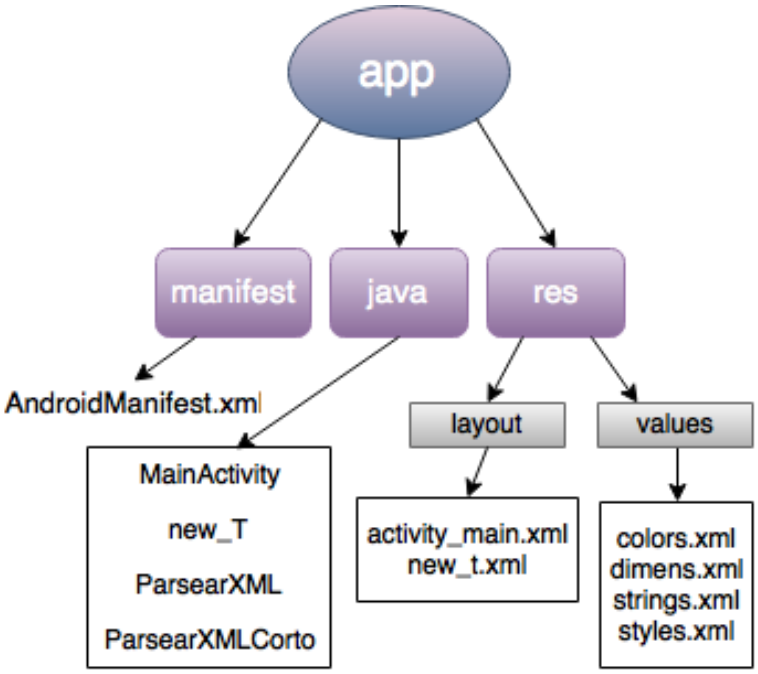


Figura 7.16: Diagrama de clases.

- **manifest:** en esta carpeta se encuentra el archivo *AndroidManifest.xml* que ha sido detallado anteriormente.
- **java>servicio1.servicio1:** es la carpeta donde se encuentran las activities y clases. Como se puede observar en la imagen 7.16 en esta aplicación se tienen cuatro en total, a las llamadas *MainActivity* y *new_ T* son las activities y *ParsearXml* y *ParsearXMLCorto* son las clases, de todas ellas se hablará más adelante con más detalle.
- **drawable:** esta carpeta está destinada a guardar las imágenes que vayan a ser utilizadas por la aplicación. En el caso de esta aplicación no se tiene almacenada ninguna imagen ya que no se usa ninguna.
- **layout:** en esta carpeta se encuentran los dos documentos *.xml* con las interfaces que componen las dos pantallas que se utilizan en la aplicación de las que se hablaba en la sección 7.1.

- **mipmap:** carpeta encargada de tener almacenados todos los iconos necesarios para la aplicación.

- **values:** en esta carpeta se encuentra el archivo string.xml anteriormente mencionado y el color.xml que ha sido editado para modificar los colores de la aplicación.

En el método **OnCreate** de la actividad *new_ T.java* queda definida la URL en función del servicio que el usuario haya elegido. Previamente se tiene declarada una variable de tipo *String* llamada **myURL** donde se encuentra la cabecera común de la URL.

En todos los **if** o **else if** se llama al método **conectar()**, menos en el servicio de pictograma, puesto que en ese se quiere devolver una imagen (**ImagenView**) y no un texto con formato XML o JSON.

En el método conectar se conecta con el Network o, en su defecto, se devuelve un error explicando que no ha sido posible realizar la conexión. Para comprobar la conectividad con el Network se usa la función **ConnectivityManager** la cual devuelve información sobre el estado de la red. Además también se usa la función **NetworkInfo** que devuelve el estado de la interfaz de red.

Si la conexión con el Network ha sido posible entonces procede a ejecutar el método **execute** de la clase llamada **DownloadWebpageTask** que se encuentra en la web de desarrolladores de Android Studio. Dentro de esta clase existen dos métodos:

- **doInBackground**: En este es donde se llama al método **downloadUrl** pasándole por parámetro la URL del servicio al que se le solicita información. Devuelve la información en formato String pasádoselo internamente al método **onPostExecute**.

- **onPostExecute:** En este método asignamos el resultado que nos devuelve el servicio al txt declarado en Android para poder mostrarlo por la pantalla.

El método **downloadUrl** recibe como parámetro la URL del servicio en formato *String*, dentro de este método se convierte esta URL tipo *String* a un objeto y se establece conexión de la forma:

```
HttpURLConnection conn =(HttpURLConnection)url.openConnection();
```

Una vez establecida la conexión se procede a tratar los diferentes resultados posibles. Esto se lleva a cabo mediante instrucciones *if* y *else if*. Dentro de cada if se llama a un método llamado **parsear** dentro de las clases **ParsearXML** o **ParsearXmlCorto**.

Las respuestas de los servicios que se tratan son las de tipo XML, pero, dado que se tienen tres tipos de salidas distintas en este formato se tratarán estos tres tipos de salida por separado. A continuación se explica brevemente cada uno de los tipos de salidas mencionados:

- **Formato vacío:** es aquel que se produce cuando alguno de los servicios no devuelve ningún resultado, representándose este de la forma expuesta en la figura 7.17.

```
<sinonimos/>
```

Figura 7.17: Ejemplo formato de salida vacía.

- **Formato simple:** es aquel que se produce cuando el servicio devuelve una única salida por lo que el formato XML solo está compuesto por una etiqueta, que representa ese único resultado. En la imagen 7.18 vemos un ejemplo de formato simple.

```
<conversion>casa</conversion>
```

Figura 7.18: Ejemplo formato de salida simple.

- **Formato compuesto:** es aquel que esta compuesto por varias etiquetas, es decir, la salida devuelve varios resultados. Por ejemplo, en el caso de sinónimos se tiene una etiqueta general "*sinonimos*", y dentro de esta otra "*sinonimo*" para cada uno de los valores devueltos, como se muestra en la imagen 7.19.

```
▼<sinonimos>
   <sinonimo>lecho</sinonimo>
   <sinonimo>catre</sinonimo>
   <sinonimo>piltra</sinonimo>
   <sinonimo>hamaca</sinonimo>
   <sinonimo>yacija</sinonimo>
   <sinonimo>litera</sinonimo>
   <sinonimo>camastro</sinonimo>
 </sinonimos>
```

Figura 7.19: Ejemplo formato de salida compuesto.

Teniendo esto en cuenta, dependiendo del tipo de salida que devuelva el servicio se llama a una función de parsear u a otra. Si la salida es de

formato simple se utiliza la clase **ParsearXmlCorto** y si la salida es de formato compuesto se usa **ParsearXml**. El caso de que devuelva un formato vacío queda contemplado en las dos. Ambas clases quedarán explicadas a continuación.

- **ParsearXml:** Esta clase, en su método **parsear**, recibe como parámetros el nombre de la clase (etiqueta general) y nombre de la etiqueta simple, además del **InputStream**. En esta función se utiliza una clase de Android llamada **XmlPullParser** que sirve para parsear texto en XML. Esta función la explicaremos más adelante.
- **ParsearXmlCorto:** Esta clase tiene el mismo comportamiento que la clase **ParsearXml** con la diferencia de que en esta solo nos interesa una etiqueta, al estar solo compuesta por una, por lo que no será necesario llamar a la función **leerVarios**.

La clase `XmlPullParser` mencionada anteriormente se encarga de leer el documento XML por etiquetas. Primero busca la etiqueta general y dentro de ella la sencilla, la cual tiene el texto que se quiere obtener. Por esta razón se tienen dos clases `parsear`. En el `ParsearXml` se pasa por parámetro tanto la etiqueta sencilla como la general y en el `ParsearXmlCorto` tan solo se pasa una etiqueta puesto que solo tiene una y la segunda se pasa un parámetro nulo.

Capítulo 8

Conclusión del proyecto y trabajo futuro

Para acabar con este proyecto y tener una visión más general y funcional del mismo pasaremos a explicar las conclusiones sacadas de su realización así como las posibles aplicaciones a trabajos futuros.

8.1. Conclusión

Una vez finalizado el proyecto analizamos la evolución del mismo y sus aspectos generales. Inicialmente la idea era crear una API de los servicios de accesibilidad ya existentes de años anteriores, facilitando así el acceso a ellos. La tarea de generación de una API de servicios de accesibilidad ha sido llevada a cabo, salvo que, en vez de ser creada a partir de los servicios existentes, la hemos creado de los servicios que nosotras nos hemos ido creando. Así la motivación inicial de nuestro proyecto, la API, ha sido cubierta, junto con el desarrollo de los servicios de los que se hablan en ella.

Además, al ir evolucionando el proyecto nos dimos cuenta que crear una aplicación que mostrara un ejemplo de uso de los servicios ayudaría a otros desarrolladores a coger ideas de posibles usos por lo que procedimos al desarrollo de la aplicación.

Nuestros servicios son REST ya que nos parecía apropiado al centrarse en el protocolo HTTP y XML para los datos y no ser necesaria ninguna capa adicional. Con este tipo de servicios se pueden implementar métodos GET, siendo este el que más nos interesaba a nosotras ya que todos nuestros servicios se basan en peticiones GET. Además nuestra intención era devolver dos formatos de salida, XML y JSON, y ambos son soportados por REST.

Inicialmente desarrollamos los servicios enfocándolos más a un usuario final, tratando los resultados como algo auto-explicativo y de fácil comprensión

para el que lo utilizara. Fue con el desarrollo más avanzado de los servicios y con el comienzo de la aplicación con lo que llegamos a la conclusión de que estos servicios no estaban destinados a un usuario final, sino a un desarrollador intermedio, que interpretaría los resultados de estos servicios y, ya si, se encargaría de darle un resultado claro al usuario final.

Debido a esto, finalmente los resultados de los servicios están enfocados a facilitar su análisis futuro más que a ser auto-explicativos, para lo cual la API tiene un papel fundamental, explicando las funcionalidades de cada uno de los servicios así como resultados que devuelven y modo de uso.

Tanto el desarrollo de la API como el de la aplicación Android nos hicieron terminar de entender bien la funcionalidad de los servicios. Así mismo, a medida que comprendíamos mejor los conceptos nos dimos cuenta de la importancia que tiene una API en cuanto a hacer accesible un servicio y dejar claro su funcionamiento y lo que se espera de él.

Tras la búsqueda de información vimos que los servicios de accesibilidad resultaban especialmente útiles en dispositivos móviles o tablets, siendo estos más accesibles en todo momento. Es por ello que nos decantamos por enfocar nuestra aplicación de ejemplo en un dispositivo Android.

En cuanto a la implementación de la aplicación Android se han utilizado los elementos proporcionados por la propia plataforma y el Material Design para la interfaz y el diseño.

La tecnología desarrollada en este proyecto tiene como objetivo ayudar a otros desarrolladores a implementar funcionalidades de accesibilidad, llamando a los servicios creados y sin necesidad de tener que reimplementar código ya existente.

Por lo anteriormente mencionado llegamos a la conclusión de que los servicios de accesibilidad, aunque cada vez más desarrollados, aún tienen mucho campo por tratar, y es por eso que este proyecto puede tener grandes aplicaciones de cara al futuro, lo cual se verá en la siguiente sección.

8.2. Trabajo futuro

Con la creación de los servicios hemos observado que el campo de la accesibilidad es mucho más amplio de lo que considerábamos al comienzo, y que aún quedan muchos servicios de accesibilidad que se pueden desarrollar.

Es por eso que este proyecto no solo sirve de cara al futuro como herramienta para el desarrollo de aplicaciones que se apoyen en nuestros servicios, si no que tiene grandes posibilidades en cuanto a ser ampliado y conseguir que sea cada vez más completo, consiguiendo que las limitaciones en cuanto accesibilidad acaben por no suponer una limitación.

En cuanto al proyecto en sí se pueden aplicar en el futuro diversas mejoras y cambios que completen sus funcionalidades. Así, como se decía anteriormente, se pueden añadir nuevos servicios al servidor haciéndolo más completo. Por ejemplo, en la versión actual del servidor se tiene únicamente traducción a inglés, pudiendo ser esto en el futuro ampliado a mayor variedad de idiomas así como a frases, y no únicamente a palabras.

Igualmente se podrían solventar y mejorar alguno de los servicios creados en el proyecto, como podría ser el de pictogramas, haciendo que las consultas se realicen a mayor velocidad, ya que la demora de estas solicitudes supone un problema a la hora de utilizar este servicio.

También se podría desarrollar el servicio de lematización, ya que al ser añadido al final este no está del todo completo y resultaría de gran utilidad en aplicaciones futuras como puede ser devolver el infinitivo de un tiempo verbal complejo para facilitar la comprensión de un texto.

En este contexto la API irá evolucionando con el desarrollo de los servicios, para mantener estos siempre accesibles y bien detallados.

De cara a la aplicación Android, siendo ahora un diseño de ejemplo, podría seguir desarrollándose, ampliando así mismo los nuevos servicios que pudieran ir surgiendo, hasta ser una aplicación totalmente disponible para los usuarios, pasando de ser un ejemplo de uso de los servicios a una herramienta para la accesibilidad.

Además la aplicación podría ser llevada a dispositivos con sistema operativo IOS para poner esta herramienta a disposición del mayor número de gente posible.

Capítulo 9

Project conclusions and future work

To conclude this project and have a more general and functional view of it we will now explain the conclisions that we have made of its implementation as well as the possible future work aplications.

9.1. Conclusions

Once the project was finished we analized its evolution and its general aspects. At the begining the idea was to create an API of the accesibility services of previous years, making the access to them easier. The task of generating an accesibility service API has been made, though the API is not about the existing projects but about the services that we have been creating. This is how our initial project, the API, has been covered, and also the development of the services covered in the API.

Moreover, while the project was evolving we realized that creating an aplication that shows how to use the services would help other developers to have some ideas of possible uses, and that is why we decided to develop the aplication.

Our services are REST as we thought that it is more appropriate for being focused in the HTTP protocol and in the XMLs data processing and for not needing any aditional layer. With this kind of services GETs methods can be implemented, being this the one that we are more interested in, because all our services are GET requests based. Also our intention were returning two exit formats, XML and JSON, and REST gives support to both of them.

At the begining we developed the services being more focused in the final user, processing the results as a self-explained messages, easy to understand for the ones that uses them. It wasn't until the really advanced development

of the service and the begining of the aplication that we realized that the services weren't meant to be used for a final user but for a developer who would interpret the result and will return the clear, self-explained result to the final user.

Because of this, the services results are focused in making it easier its parse to a future developer rather than to be self-explained, which is the work of the API, explaining the functions of each one of the services and what to expect from it.

Both the API development and the Android aplication made us finally understand properly the services's function. Also, as we were understanding better the concepts we were realizing the importance of the API to make easier the access to the services and to explain how they work and what to expect from them.

After the information search we saw that accesibility services were specially useful in movile devices and tablets, being this more accesible at every moment. That is why we choosed focus our example's aplication in an Android device.

With regard to the Android's application implementation it's been used the elements that are supplied by the platform and the Material Design for the interface and the design.

The developed technology in this project has as a goal helping other developers to implement accessibility functions, by using the created services and without having to code everything from zero.

To sum up we can conclude that the accesibility services, though they are more developed everyday, still have a long way to go, and that's why this project could have great aplications in the future, which will be seen in the next section.

9.2. Future work

With the creation of the services, we have observed that accessibility's field is much bigger than we thought at the begining, and that there are still many accessibility services that can be developed.

That is why this project is not only facing the future as a tool for the development of an aplication, but it also has huge posibilities in regard of the posibility of being extended and being each day more complet, making the limitations of the languaje not be a limitation anymore.

In regards to the project, improvements and changes can be made in the future to make its functionalities more complete. Also new services can be

added to the server in the future, making it more complete. For example, in the current version of the server we only have a translation to english, in the future this could be extended to other languajes aswell.

It could be also fixed and improved some service already created in this project, as the pictograph one, making the request faster, being the time that the current service takes to return the image too long.

Iin this context the API will be evolving with the developing of the services, so they are always going to be accessible and well described.

Regarding the Android application, it could keep developing, adding the new services that can be make until it becomes an actual aplication for the users, becoming a useful tool instead of an example. accesibilidad.

The application could also be implemented in IOS operating systems, making it accessible for more people.

Capítulo 10

Aportaciones individuales

10.1. Sheila Plaza Estévez

El comienzo del trabajo se centró en la búsqueda de información así como recursos que pudieran ser útiles para su desarrollo. Se estuvo investigando qué tipo de servicios serían más útiles para desarrollar en un futuro en una aplicación. Nuestros tutores nos sirvieron de gran ayuda en este comienzo, puesto que nuestros conocimientos sobre este tema eran bastante escasos, pero con su ayuda y la información recaudada de Internet y algunos libros, poco a poco pudimos ir teniendo la información necesaria para desarrollar el proyecto.

Igualmente se debatió qué tipo de aplicación podría desarrollarse para servicios de accesibilidad y cuál sería la mejor plataforma para ello.

Este proyecto ha sido desarrollado por tres persona, y una de ellas fuera del país durante el primer cuatrimestre, lo cual no fue ningún impedimento puesto que nos organizamos de tal forma que esa persona pudiera estar al día de todo lo que se hablaba en nuestra reuniones, tanto con los directores del proyecto como entre las dos que estábamos en Madrid, y en como avanzábamos, para que ella pudiese avanzar en paralelo. Para ello lo mejor fue utilizar un plan desde el comienzo y seguirlo durante todo el año para no perder el control del tiempo ni de las tareas. En este contexto se decidieron usar herramientas bastante útiles para la planificación de tareas así como para estar comunicadas: Trello, Google Drive, Skype y Whatssap.

Trello ha sido utilizado para la planificación y asignación de tareas a los diferentes miembros del grupo. En Google Drive se abrieron dos carpetas, una carpeta con los componentes del grupo y los directores para poder compartir la información que se fuese teniendo además de subir todas las actas de las reuniones que se iban haciendo para que así la persona del extranjero pudiera saber de que se hablaban en las reuniones que ella estaba ausente. La segunda carpeta del Drive la realizamos sólo para los componentes del grupo donde

a la vez se encontraban las diferentes carpetas de nuestros trabajos y las distintas actualizaciones de los mismos.

Durante el primer periodo del proyecto mis tareas fueron recoger toda la información posible para poder desarrollar el trabajo y a la vez entender que son los servicios web rest, y como se usan. Mientras se buscaba información, como la búsqueda tenía una gran cantidad de resultados, decidimos empezar a escribir la memoria tal y como nuestros tutores nos habían recomendado puesto que si la dejábamos para el final luego no nos acordaríamos de muchas cosas.

La búsqueda de la información nos llevó unos meses, mas o menos de septiembre a primeros de noviembre. Después de haber entendido lo que era un servicio web rest y como funcionaba procedimos a la realización mas inicial del proyecto.

Desde noviembre hasta febrero estuve creando la primera versión de nuestra web API y creando algunos servicios web REST de prueba para ir entendiéndolos mejor y asentar conocimientos. Uno de los servicios de prueba, con varias mejoras, serviría más adelante, convirtiéndose en el servicio de palabra sencilla. Este primer cuatrimestre mi disponibilidad era bastante baja por lo que los avances por mi parte se veían a paso más lento.

En una de las reuniones antes de comenzar con todos los servicios, decidimos cuales deberíamos desarrollar primero y cuales en un futuro si acabáramos a tiempo los primarios. Nuestros tutores pusieron a nuestra disposición algunos de los trabajos de fin de carrera de otros años que estaban relacionados con los servicios que nosotras queríamos hacer.

En segundo cuatrimestre mi disponibilidad era bastante más elevada por lo que me dedique a desarrollar algunos servicios web REST para poder terminar lo ante posible la web y empezar con una aplicación de ejemplo.

Al principio habíamos acordado que los servicios rest serían traducidos a java de otros proyectos escritos en PHP, Java, etc. Cuando me dispuse a traducir el servicio de sinónimos de un trabajo anterior en PHP a Java, me fue muy complicado, ya que yo no entiendo de PHP, por lo que antes de estar gastando tiempo en traducir pensé que sería más sencillo si empezaba a escribirlo desde cero, y así fue.

Durante los primeros meses del segundo cuatrimestre me dediqué a desarrollar algunos servicios analizando el código html de una página web. Esto parecía complicado pero la librería JSOUP hizo que el análisis fuera más sencillo.

Antes de comenzar con los servicios, estuve mirando que página se ajustaba más a las necesidades que estábamos buscando. Algunas de las candidatas eran:

- Sinonimos.com[1]
- Sinonimos.org[2]
- ElPaís[3]

La de ElPaís fue una de las páginas con las que empecé a trabajar puesto que en trabajos anteriores habían trabajado con ella. Pero buscando un poco más, me di cuenta que la página WordReferences[4] daban respuesta a casi todos los servicios que nosotras queriamos implentar, por ello decidí que esta seria la mejor, ya que una vez aprendiese a parsearla para un servicio podría parsearla para los demás.

Para dar uso a nuestros servicios, pensamos que podríamos desarrollar una aplicación de ejemplo, algunas de las alternativas eran:

- Un ejemplo dentro de la página web.
- Una aplicación Java para ordenador.
- Una aplicación Android para móvil o tablet.

Decidimos que la mejor opción sería Android puesto que hoy en día el móvil o la tablet es usado por la mayoría de las personas y el sistema operativo Android, según estudios realizados por un periódico, es usado en un 88 % de los dispositivos. Otra de las razones por la que elegimos Android es porque mientras buscábamos información sobre cómo realizar servicios web rest, encontrábamos bastante sobre Android, y decidimos aprovechar este avance.

Una vez decidido la plataforma que usamos para nuestra aplicación de ejemplo, comencé a investigar qué programa podría utilizar. Como ya contamos en capítulos anteriores, en un principio comencé con Eclipse, sin darme cuenta de que estaba obsoleto para android. Decidí utilizar Eclipse porque era un entorno que ya conocía. Viendo que con este programa no podía desarrollar, comencé con Android Studio. Una vez habituada a este entorno de desarrollo, comence con la implementación de toda nuestra aplicación Android, utilice un emulador de una Tablet Nexus 7.

Para terminar con este apartado de mis aportaciones al trabajo, sólo añadir que en la memoria he aportado todo el desarrollo de Android, la parte de servicios que desarrolle en su comienzo, parte del resumen y conclusiones y posibles adaptaciones al futuro.

[1] `http://www.sinonimos.com/`
[2] `http://www.sinonimos.org/`
[3] `http://servicios.elpais.com/diccionarios/sinonimos-antonimos/`
[4] http://www.wordreference.com/

10.2. Nerea Ramírez Lamela

Inicialmente, acordamos que nuestro proyecto se iba a desarrollar en JAVA, utilizando servicios rest.

Decidimos que íbamos a realizar actas con los puntos tratados en cada reunión, y posteriormente subirlas a una carpeta compartida en el drive. El propósito de estas actas, era mantener informada a una de las integrantes del grupo, debido a que se iba a encontrar fuera durante el primer cuatrimestre y no iba a poder asistir a dichas reuniones. Esto no ha supuesto ningún problema ya que gracias a aplicaciones como Skype, drive como repositorio y whatsapp hemos estado constantemente en contacto.

Para mejorar nuestra coordinación y organización creamos dos archivos donde en uno de ellos se escribían las tareas a realizar y en el otro las dudas que nos surgían y no conseguíamos resolver para preguntárselo a nuestros tutores en la próxima reunión.

La primera toma de contacto con el proyecto, fue familiarizarse con la estructura y el funcionamiento de una API y con los servicios rest. Para ello, realizamos búsquedas en internet de APIS ya creadas, donde pudimos observar diferentes diseños y recopilamos ideas para la nuestra.

En los cuatro primeros meses, mis tareas se basaron en la búsqueda de información, la creación del primer prototipo de la página web de la API, entender el funcionamiento del servidor proporcionado por nuestros tutores y el funcionamiento de los servicios rest.

Para ello, me instale todas las herramientas necesarias para la generación de servicios rest, e investigue sobre todo lo relacionado con la tecnología que íbamos a tener que utilizar, ya que nuestros conocimientos en este ámbito eran bastante escasos.

Conseguimos recopilar mucha información, por lo que siguiendo los consejos que nos habían dado y para que no se nos olvidarán detalles del progreso del proyecto, empezamos a escribir la memoria dejando plasmado toda la información recolectada.

Gracias a la ayuda de nuestros tutores, consultas en Internet y libros, conseguimos obtener nociones básicas para poder empezar a realizar nuestros primeros ejemplos de servicios. Por ejemplo, el primer servicio que desarrolle, fue una agenda simple, realizando un servicio que hiciera una petición HTTP con el método GET.

Una vez que estuvimos familiarizadas con las tecnologías, creamos el primer prototipo del servicio “palabra sencilla” del proyecto, generado con eclipse y utilizando como servidor local, “Tomcat”. A finales del cuatrimestre, nuestros tutores nos facilitaron un servidor de la universidad “http://sesat.fdi.ucm.es/”,

para que pudiéramos almacenar nuestros archivos y servicios. Tuve que configurar y realizar modificaciones en los permisos de las carpetas, a través de una consola Linux, para que nos permitiera poder subir nuestros archivos .war que contenían el código desarrollado.

Al mismo tiempo que realizaba las tareas comentadas antes, creé el primer prototipo de la API, que sólo contenía el esqueleto de la página web, ya que no teníamos definido los servicios que iban a estar incluidos en nuestro proyecto.

En esta última etapa del curso, mi disponibilidad bajó considerablemente, por lo que mis avances tardaban más tiempo en verse reflejados.

En una reunión del mes de Enero, definimos qué proyectos íbamos a desarrollar primero, dejando los restantes en segundo plano, con el pensamiento de ser desarrollados en un futuro, en caso de que hubiera tiempo.

Cuando me dispuse a definir lo servicios que se encontraban en los proyectos marcados como primarios, y a ver la forma de integrarlos en nuestra Api, me di cuenta de que salvo uno de ellos, los demás tenían una estructura completamente distinta a la que nosotras teníamos pensado, para desarrollar nuestro proyecto. A consecuencia de esto, tomamos la decisión de implementar nuestros propios servicios para darles una estructura homogénea y sencilla usando como referencia estos proyectos.

Los proyectos a desarrollar elegidos fueron ?Apoyo a la simplificación de textos sobre navegadores web? y ?TraductorTextoPictos?. El criterio seguido para esta selección, fue la dificultad que iba a suponer traducir los servicios a servicios rest en cada proyecto según la tecnología utilizada.

En esta última etapa me dediqué a desarrollar, ampliar y mejorar todos los servicios que están implementados en el proyecto, ya que no pudimos utilizar los servicios creados en años anteriores , a excepción del caso de pictogramas.

Por último, comentar que en el desarrollo de la memoria he aportado la gran mayoría de los servicios, información sobre el análisis de los proyectos de años anteriores, parte de conclusiones y posibles adaptaciones al futuro.

10.3. Carmen Acosta Morales

La primera reunión con nuestros directores del proyecto la tuvimos justo antes de que yo me fuera de Erasmus el primer cuatrimestre. En ella se habló de la idea inicial del proyecto, caminos que podía ir tomando a medida que evolucionara y posibles aplicaciones que se podían hacer del mismo. Igualmente nuestros tutores nos dieron una serie de recomendaciones iniciales

y pautas a seguir durante el proyecto para que no se nos echara el tiempo encima y ser lo más eficaces posible.

Además acordamos que para que yo estuviera al día de las actualizaciones de las reuniones y de lo que se iba acordando, se tomarían actas de las mismas, las cuales además servían para no olvidarnos de lo acordado en una reunión.

Debido a que yo iba a estar fuera el primer cuatrimestre acordamos que para que estuviera al día de las actualizaciones de las reuniones y de lo que se iba acordando en ellas, se tomarían actas de las mismas, las cuales además servirían para no olvidarnos de todos los puntos tratados durante una reunión.

Además, fuera del ámbito de las reuniones, para solucionar el asunto de la distancia acordamos utilizar una serie de herramientas para que no supusiera un problema y poder ir avanzando yo también desde la distancia en una versión actualizada de lo llevado a cavo por las que estaban en Madrid. Estas herramientas han sido: Google Drive, Skype, Whatsapp y Trello. A las que más uso les hemos dado han sido:

- Google Drive, ya que hemos ido guardando en ella todas las versiones de las distintas partes del proyecto para que cada integrante del grupo pudiera acceder a ellas cuando lo necesitara.
- Whasapp ya que resultaba la forma de comunicación mas inmediata para informar de las novedades que no requirieran de mucha explicación. Para las novedades más extensas utilizamos Skype.

Al comiendo del curso me informé de todo lo que pudiera ser de utilidad para el desarrollo de nuestro proyecto. Así empecé a buscar información sobre REST, SOAP, APIs, servicios de accesibilidad, trabajo con servidores, etc. Una vez que me hube familiarizado más con estos conceptos, totalmente nuevos para mi en un inicio, empecé a buscar información de una forma más enfocada a ir escribiendo el "Estado de la Cuestión" de la memoria.

Una vez recopilada información suficiente acerca del proyecto en general me dediqué a buscar información sobre LaTeX y empecé a trabajar con la platilla TeXiS. Para ello me instalé TeXstudio para poder ir desarrollando la memoria en LaTeX ya que nos parecía que le daba mejor formato al diseño de la misma. TeXstudio no fue el primer programa que probé para trabajar en LaTeX, pero debido a que los otros que probé me dieron problemas los directores del proyecto me recomendaron trabajar con TeXstudio y, salvo un inconveniente inicial con la bibliografía, todo funcionó correctamente.

Una vez que estuvimos más informadas empezamos a dejar más afianzadas nuestras ideas de qué desarrollar en nuestro proyecto. Desde un principio

tuvimos claro que íbamos a desarrollar una API de servicios web de accesibilidad y que para ello tendríamos que unificar los servicios de proyectos anteriores para, posteriormente, facilitar su acceso. Cuando empezamos a trabajar con los servicios llegamos a la conclusión que lo mejor que podíamos hacer era empezarlos de nuevo, cogiendo ideas de los de años anteriores.

Al inicio del segundo cuatrimestre me dediqué a desarrollar el servicio de "conversión a fácil", así como a tratar algunas modificaciones de los servicios más desarrollados. A medida que los servicios iban siendo acabados empezamos a realizar el desarrollo de la aplicación Android, con la cual estuve inicialmente trabajando. Cabe destacar que antes de usar Android Studio para la aplicación intentamos desarrollarla con el Android de Eclipse, pero este había dejado de dar soporte a Android por lo que nos pasamos a Android Studio.

Finalmente realicé la actualización de la página web, realizando algunos cambios en el diseño, tanto de la página principal como de aquellas que contienen cada uno de los servicios, dándole más consistencia a unas con otras. Igualmente fui actualizando todos los enlaces a los servicios, ya que , debido a que estos habían ido evolucionando y extendiéndose, la página web se había quedado obsoleta en ese sentido, así como los ejemplos y descripciones de los servicios que contiene.

Para el desarrollo de la página web buscamos distintas APIs que nos pudieran servir de referencia, y así obtuvimos unas ideas iniciales de cómo queríamos que fuera nuestra API. Decidimos que la sencillez y la eficacia eran fundamentales por lo que hicimos una web en la que cada servicio quedara bien explicado, con la URL de cada uno de los servicios, en sus distintas formas, y un ejemplo de llamada con su resultado para que quedara clara la funcionalidad.

Así mismo, a medida que íbamos desarrollando el proyecto e íbamos escribiendo la memoria, me iba encargando de estructurar esta en LaTeX y de incorporar las aportaciones ajenas o propias a la misma. Además desarrollé en la misma el tema de la página web, así como algunos servicios, introducción y estado de la cuestión.

Bibliografía

Alonso, A. H.-G., Díaz, C. M. y García, S. P. *Conversor de texto a pictogramas*. Trabajo de fin de grado, Facultad de Informática, UCM, 2013 -2014.

Apple. Apple voiceover. 2016. Disponible en `http://www.apple.com/es/accessibility/osx/voiceover/` (último acceso, Junio, 2016).

Burke, B. *RESTful Java with JAX-RS*. O'Reilly Media, Noviembre, 2009. ISBN 978-1-4493-6134-1.

FESOCA. Webvisual. 2016. Disponible en `http://www.webvisual.tv/` (último acceso, Junio, 2016).

González, A. B. y Ortiz, J. G. *Apoyo a la simplificación de textos sobre navegadores web*. Trabajo de fin de grado, Facultad de Informática, UCM, 2013 -2014.

Google. Google talkback. Abril, 2016. Disponible en `https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=es` (último acceso, Junio, 2016).

Google. Google braileback. Diciembre, 2015. Disponible en `https://play.google.com/store/apps/details?id=com.googlecode.eyesfree.brailleback&hl=es` (último acceso, Junio, 2016).

IBM. Ibm api types. 2016. Disponible en `https://developer.ibm.com/apiconnect/documentation/api\discretionary{-}{}{}101/types-apis/` (último acceso, Junio, 2016).

Linares, J. D. y Flores, J. L. G. *Aplicación multiplataforma para conversión personalizada de textos*. Trabajo de fin de grado, Facultad de Informática, UCM, 2013 -2014.

Maddox, S. ffeathers api types. 2016. Disponible en `https://ffeathers.wordpress.com/2014/02/16/api\discretionary{-}{}{}types/` (último acceso, Junio, 2016).

MARTÍN, C. R. y CALLEJA, J. P. G. *Editor predictivo de mensajes en pictogramas*. Trabajo de fin de grado, Facultad de Informática, UCM, 2013 -2014.

PROMEDIA, G. Pictotraductor. 2013. Disponible en `http://www.pictotraductor.com/` (último acceso, Junio, 2016).

WIKIPEDIA. Wikipedia xml. Abril, 2016. Disponible en `https://es.wikipedia.org/wiki/Extensible_Markup_Language` (último acceso, Abril, 2016).

WIKIPEDIA. Wikipedia wsdl. Agosto, 2015. Disponible en `https://es.wikipedia.org/wiki/WSDL` (último acceso, Abril, 2016).

WIKIPEDIA. Wikipedia jsonrpc. Junio, 2016a. Disponible en `https://en.wikipedia.org/wiki/JSON\discretionary{-}{}{}RPC` (último acceso, Junio, 2016).

WIKIPEDIA. Wikipedia rest english. Junio, 2016b. Disponible en `https://en.wikipedia.org/wiki/Representational_state_transfer` (último acceso, Junio, 2016).

WIKIPEDIA. Wikipedia rest. Marzo, 2016a. Disponible en `https://es.wikipedia.org/wiki/Representational_State_Transfer` (último acceso, Abril, 2016).

WIKIPEDIA. Wikipedia soap. Marzo, 2016b. Disponible en `https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol` (último acceso, Diciembre, 2015).

WIKIPEDIA. Wikipedia xmlrpc. Mayo, 2016. Disponible en `https://en.wikipedia.org/wiki/XML\discretionary{-}{}{}RPC` (último acceso, Junio, 2016).