IAFlex: Flexionador nominal inteligente

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Daniel Josué Pérez Melián

TUTORIZADO POR: Francisco Javier Carreras Riudavets

Agradecimientos

A mis padres por apoyarme siempre en la carrera, a mi hermana por poner a prueba mis programas y aguantar mis bromas, a mis amigos por siempre estar ahí y a Claudia por ser mi apoyo emocional incondicional.

Resumen

Son muchas las herramientas de procesamiento de lenguaje natural, pero son muy escasos los flexionadores y, los que hay, son arcaicos e incompletos. Es por ello por lo que se busca desarrollar e implementar una solución web capaz de resolver este problema. El «Flexionador Nominal Inteligente» es una herramienta de procesamiento del lenguaje natural para el idioma español. Consiste en una aplicación web capaz de obtener todas las flexiones de una palabra dada, además de poder encontrar la forma canónica de una flexión introducida. Puede tratar sustantivos, pronombres, adjetivos, adverbios, conjunciones, etc. Y aunque reconoce los verbos no los conjuga. En este trabajo, los apreciativos diminutivos, aumentativos y despectivos se consideran flexiones nominales. Las flexiones que muestra son principalmente género y número y el superlativo para los adjetivos, entre otras menos frecuentes. Utilizando un servicio WCF y diferentes tecnologías web, muestra al usuario una interfaz amigable donde poder consultar las flexiones de cualquier palabra nominal del idioma español. Esta herramienta añadirá riqueza a la lengua española ya que será un instrumento más que tendrán todos los hispanohablantes disponible para sus consultas acerca del lenguaje.

Abstract

There are several natural language processing tools, but it is not easy to find a tool that gives all the inflections of a word, and those that already exist are archaic and incomplete. Therefore, the goal of this project is to develop and implement a web solution able to solve this problem. The “Flexionador Nominal Inteligente” (Intelligent Nominal Inflection Tool) is a natural language processing tool for the Spanish language. It consists of a web application able to retrieve all the inflections of a given word and it also can find the canonical form of a given inflection. It deals with nouns, pronouns, adjectives, adverbs, conjunctions, etc. And although it recognizes verbs, it does not conjugate them. In this work, the diminutive appreciatives, augmentative and pejoratives are considered nominal inflections. The inflections shown are mainly gender and number and the superlative for adjectives, among others less frequent. Using a WCF service and several web technologies, this tool displays a user-friendly interface where one can look up all the inflections of any nominal word of the Spanish language. This solution will add value to the Spanish language because it will be an additional tool among others that all Spanish speakers will have available for their queries about the language.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 – Introducción

«El lenguaje es el vestido de los pensamientos»

Samuel Johnson

Cada vez son más numerosos los recursos lingüísticos en internet, y entre ellos se encuentran las herramientas correspondientes al procesamiento del lenguaje natural. Existen en la actualidad numerosos conjugadores de verbos en el lenguaje español, pero, sin embargo, son muy escasos los flexionadores de formas no verbales que sean usables, completos y fiables. De hecho, en el lenguaje inglés también es difícil encontrar flexionadores, y los que se pueden encontrar están en repositorios y es necesario instalarlos para su utilización, como por ejemplo el proyecto *LemmInflect*.

Este trabajo se contextualiza, como se mencionó anteriormente, dentro del campo del procesamiento del lenguaje natural, un área del conocimiento situada dentro de la inteligencia artificial. Esta área consiste principalmente en tratar las lenguas naturales para obtener resultados de una forma concreta, fiable y automática.

En las lenguas flexivas (entre las que se encuentra el español) las palabras no verbales constan de un lexema (o raíz) y pueden tener diferentes morfemas. Se define como flexión nominal a las alteraciones que sufren las palabras mediante estos morfemas, concretamente los morfemas flexivos nominales (Navarro, 2017). Se pueden distinguir las palabras:

- Variables: tienen morfemas flexivos. En español, estos tipos de palabras son el sustantivo, el adjetivo y el pronombre, además de los verbos y,

excepcionalmente, otras categorías no flexivas como el adverbio (por ejemplo, el adverbio «cerca» con la flexión «cerquita»).

- Invariables: no poseen morfemas flexivos. Ejemplos: el adverbio, la preposición, la conjunción y la interjección (Universidad Complutense Madrid, s.f.).

En este proyecto se incluyen las palabras variables e invariables. En el caso de las palabras variables, los verbos son los únicos que no se tratan debido a que para ese propósito existen ya multitud de conjugadores verbales. Para las palabras invariables se indica que no tienen flexión.

Por la falta de una herramienta que informe de estas flexiones, se decidió diseñar e implementar una aplicación web que sea capaz de flexionar las palabras no verbales del idioma español **«IAFlex: Flexionador nominal inteligente»**. El origen de este nombre radica en la capacidad que tiene la aplicación de reconocer flexiones de forma automática y, a partir de su lema, obtener todas sus flexiones.

Este flexionador es capaz de resolver una amplia variedad de flexiones, entre las que se encuentran: el singular y plural, el femenino y masculino, los diminutivos, aumentativos y peyorativos y cualquier otra flexión que admita la categoría gramatical a la que pertenezca.

Otra de las motivaciones de este proyecto es el hecho de que existe un servicio WCF que ya está disponible por parte de la División de Lingüística Computacional del Instituto Universitario de Análisis y Aplicaciones Textuales (IATEXT) de la ULPGC que permite obtener información morfológica relevante para la resolución del problema. WCF son las siglas de *Windows Communication Foundation*, un *framework* para la creación de aplicaciones orientadas a servicios (Microsoft, 2017). Conectando este servicio a la aplicación se pueden hacer diferentes tipos de consulta dependiendo de qué información se necesite dentro del

código. Una vez obtenida dicha información, se pueden hacer todo tipo de tratamientos a esos datos para posteriormente mostrar los resultados al usuario.

Sin embargo, disponer de este servicio no supuso ni mucho menos todo el trabajo, ya que la información requería de mucho tratamiento para poder ser presentada de una forma ordenada, fiable y amigable de cara al usuario final.

Uno de los grandes problemas que presentó utilizar este servicio fue el de cómo separar y agrupar la información. Concretamente, cómo separar unas categorías gramaticales de otras para poder mostrar todas las flexiones de cada palabra ordenadamente.

Este problema se resolvería durante el desarrollo con un algoritmo que se convirtió en el más importante dentro de la aplicación. Más adelante se detallará la implementación de este, pero cabe destacar la problemática que causó tener tanta información que debía ser separada y agrupada en subcategorías (separada por género, separada por número, etc).

Capítulo 2 – Estado actual y objetivos iniciales

En la actualidad, existen muchas herramientas de procesamiento de lenguaje natural. Algunas de las más relevantes o útiles son, por ejemplo, los conjugadores verbales. Prueba de esto es que hasta la propia Real Academia Española cuenta en su web con un conjugador verbal.

Existen también aplicaciones como buscadores de sinónimos y antónimos, analizadores de rimas para poemas, comprobadores gramaticales, etc. También se pueden encontrar soluciones que expresan un número en forma de palabras, separadores de sílabas y analizadores semánticos, entre otros.

Sin embargo, es muy complicado encontrar un flexionador que proporcione de forma completa todas las flexiones nominales de una palabra. En el lenguaje español, los pocos flexionadores que se encuentran están desfasados y son incompletos, además de no estar centrados en la experiencia de usuario.

Podemos encontrar, en el lenguaje inglés, algunos flexionadores como es el caso de *LemmInflect*. Este proyecto requiere de una instalación con Python y se usa a través de línea de comandos. Para realizar una prueba, se ha instalado este proyecto y se han introducido algunos comandos con sus respectivas respuestas:

```
>>> from lemminflect import getAllInflections

>>> getAllInflections('be')
{'VB': ('be',), 'VBD': ('was', 'were'), 'VBG': ('being',), 'VBN': ('been',),
'VBP': ('am', 'are'), 'VBZ': ('is',)}

>>> getAllInflections('window')
{'NNS': ('windows',), 'NN': ('window',), 'VBD': ('windowed',), 'VBG':
('windowing',), 'VBZ': ('windows',), 'VB': ('window',), 'VBP': ('window',)}
```

En este ejemplo se ha probado tanto el verbo *to be* como el sustantivo *window* y se puede apreciar que muestra todas las flexiones de cada una de las palabras introducidas en el idioma inglés. Este proyecto presenta la gran desventaja de tener que ser instalado para poder ser utilizado. Además, no es un archivo `.exe` como el que la mayoría de usuarios están acostumbrados, sino que se trata de un módulo Python que requiere de cierta experiencia por parte del usuario para poder acceder a este recurso.

Es por estas razones por lo que se buscó crear una aplicación amigable de cara al usuario, para que tenga una buena experiencia y, al mismo tiempo, se quiso ofrecer una aplicación robusta y completa.

Por otro lado, la División de Lingüística Computacional del IATEXT de la ULPGC cuenta con una serie de herramientas web para el procesamiento del lenguaje natural. Entre ellas, se encuentran aplicaciones como «Sílaba Tónica TIP» y «Conjugador TIP» entre otras. Desde el inicio del proyecto se consideró que, dependiendo de la calidad del resultado del proyecto, se podría incluir el «Flexionador Nominal Inteligente» como parte de este conjunto de herramientas online sobre el idioma español.

Por esta razón, durante el transcurso del proyecto se ha tenido esta motivación en mente para adaptar e integrar la aplicación con el resto de herramientas. Por ejemplo, como el flexionador no conjuga verbos, pero sí que los reconoce, se pueden enlazar para que queden vinculados y se pueda conjugar un verbo partiendo de un reconocimiento de una flexión verbal.

De esta manera, a pesar de tener en cuenta que podría ser integrado con el resto de herramientas, se tuvo la libertad desde el inicio de incorporar estilos propios y no necesariamente similares a las otras herramientas.

No obstante, los objetivos iniciales no se fueron cumpliendo a la perfección desde el primer momento, ya que hubo multitud de pequeños problemas durante la implementación que hicieron que se tuviera que cambiar la perspectiva del proyecto en algunas de sus facetas.

Gran parte de estos problemas derivaron directamente de la casuística del lenguaje español en cuanto a flexiones se trata. Ejemplos de dichos problemas son: palabras con casos particulares de flexión, mayor o menor cantidad de flexiones de una palabra, ausencia de alguno de los géneros en según qué palabras (sustantivos exclusivamente masculinos o femeninos, por ejemplo), entre otros.

Entre las palabras con casos particulares de flexión, se encuentran ocurrencias como actor/actriz, comandante/comandante, vampiro/vampiresa, rey/reina, etc. Estas situaciones están contempladas en el servicio WCF, pero, aun así, se han tenido que probar varios ejemplos extremos como estos para comprobar que la aplicación es robusta y fiable para todas las peculiaridades de la lengua española.

La mayor o menor cantidad de flexiones de una palabra fue un aspecto importante que cambió la estructura del código debido a varios factores. Existen casos extremos como las conjunciones o adverbios, que no tienen flexiones y por tanto se decidió mostrar únicamente su forma canónica. Sin embargo, algunos adverbios sí que tienen alguna flexión, como el caso de la palabra «cerca» con la flexión «cerquita», así que también se tuvieron que considerar estas ocurrencias en el código para mostrarlas en sus tarjetas correspondientes.

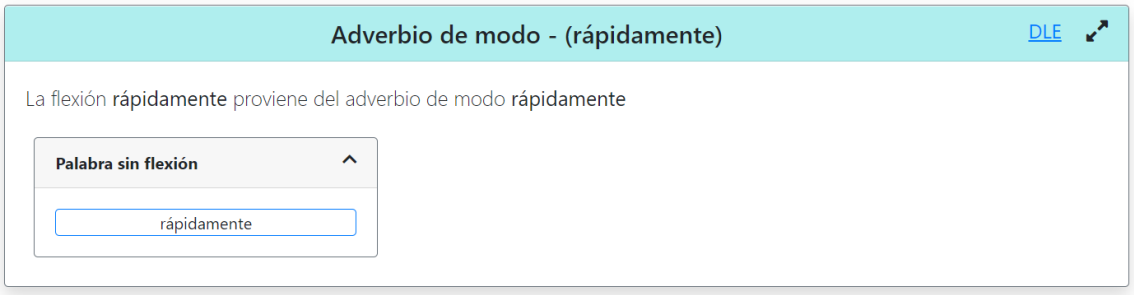

Ilustración 2-1: Disposición de una palabra sin flexión

Además, existen palabras en nuestro idioma como los determinantes demostrativos que tienen versiones de género neutro («esto», «eso» y «aquello») que dificultan la disposición en la aplicación web ya que agregan una columna extra a las tarjetas donde se muestran las flexiones, especialmente en el modo móvil.

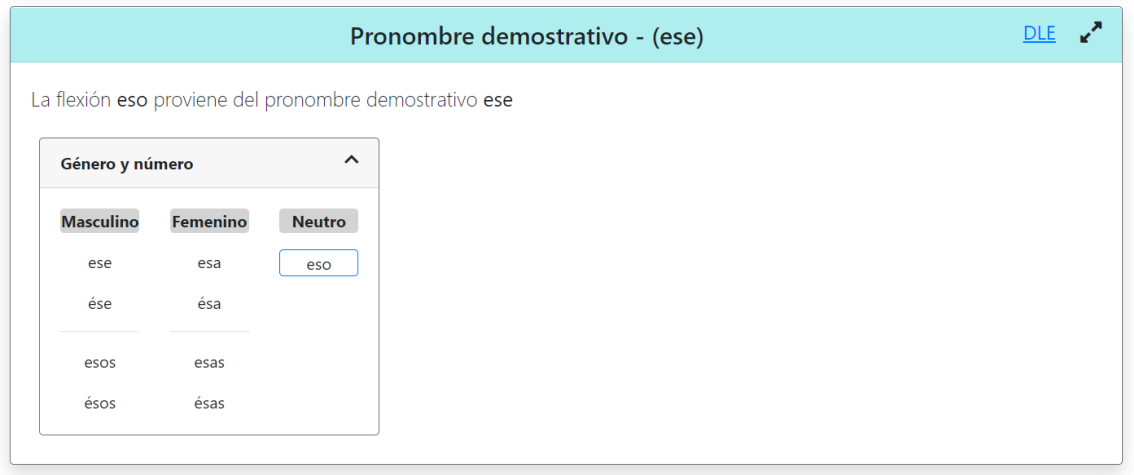

Ilustración 2-2: Disposición con género neutro

Estos son solo algunos de los casos específicos del español que se han tenido que tratar en este proyecto. Este problema fue de los más importantes durante el desarrollo de la aplicación, ya que constantemente aparecían casos (hasta el final del desarrollo) que necesitaban un tratado diferente a la hora de organizar las tarjetas y mostrar la información al usuario.

Aparte de los ya mencionados, se encontraban ocurrencias como las onomatopeyas, que también son palabras del idioma y, como tal, deben aparecer correctamente en la aplicación. El caso de las onomatopeyas se encontraba dentro de otro problema, el de diferenciar entre «de» y «de la» cuando se menciona de dónde proviene la flexión introducida.

Por ejemplo, si se introduce la palabra «o», el sistema debe enseñar un mensaje que sea «la flexión **o** proviene de la conjunción **o**», y si se introduce la palabra «perrito», el sistema enseñará el mensaje «la flexión **perrito** proviene del sustantivo masculino **perro**». Por tanto, se debe tener en cuenta el género de las palabras que componen las categorías para mostrar correctamente «de» o «de la».

De esta manera, se listan aquí todos estos casos diferentes a la hora de mostrar flexiones y que tuvieron que ser tenidos en cuenta a lo largo del desarrollo:

- Palabras «normales» que tienen versión masculina y femenina, además del mismo número de flexiones singulares y plurales
- Palabras solo masculinas o femeninas
- Palabras con distinto número de singulares y plurales
- Palabras sin flexión
- Palabras sin flexión que excepcionalmente sí tienen
- Pronombres personales
- Palabras con género neutro
- Palabras con distinta forma en masculino que en femenino

Toda esta casuística pone de manifiesto la gran importancia de diseñar la disposición de la información según el tipo de palabra que se esté procesando en la aplicación.

Capítulo 3 – Competencias específicas y aportaciones del trabajo

La competencia específica de este trabajo es la IS04, «Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales».

Para el desarrollo de esta competencia ha sido esencial el proceso de investigación y descubrimiento de las técnicas y tecnologías que existen en la actualidad, lo que se conoce como estado del arte. Se empezó buscando flexionadores que pudieran existir y que estuvieran disponibles en internet. Esta búsqueda llevó a la conclusión de que no existen otros flexionadores en la red y, si existen, están incompletos y anticuados. Para averiguar si esta falta de flexionadores se produce en otros idiomas, se realizó otra búsqueda en el idioma inglés. Los resultados de flexionadores son también escasos, se encuentran en repositorios de código que requieren una instalación y, por lo tanto, no son demasiado útiles para el público general.

Se identificó, por tanto, una falta de flexionadores en el ámbito del procesamiento del lenguaje natural que podría ser resuelta por este proyecto. Analizando el panorama actual de tecnologías se decidió crear un proyecto que supliera esta falta, con un aspecto y diseño amigables de cara al usuario.

Por otro lado, en cuanto al diseño, se pensó desde el primer momento en cómo se iba a implementar la solución web antes de empezar a programar. Se tuvieron en cuenta diferentes tecnologías para el desarrollo del diseño, que finalmente se realizó con Bootstrap debido a la experiencia previa con dicha tecnología y su fácil y rápida implementación en un proyecto ASP.NET.

Para verificar que se estaba diseñando correctamente, se realizaron múltiples reuniones cada cierto tiempo para comprobar con el tutor la implementación de los estilos, además de probar tras cada cambio cómo resultaba la aplicación en modo móvil y tableta.

Por las razones previamente mencionadas, este proyecto servirá para llenar este hueco en el área de herramientas lingüísticas del lenguaje español. El objetivo es que sea utilizado en todo el mundo hispanohablante, principalmente en entornos educativos como pudieran ser colegios e institutos. Podría ser una herramienta de utilidad en estos contextos ya que es una solución completa y que ofrece todas las posibles flexiones de las palabras no verbales.

Además, la posibilidad de que sea añadido este proyecto a las herramientas existentes de la División de Lingüística Computacional del IATEXT de la ULPGC les aportaría un valor añadido a ambas partes. Esta aplicación se vería beneficiada de ser parte de un sistema de la ULPGC, mientras que las herramientas tendrían la mejora de contar con una herramienta más disponible.

Capítulo 4 – Desarrollo

El desarrollo de todo el proyecto estuvo guiado por la metodología de modelo de prototipos. En esta metodología priman las fases del desarrollo, como pueden ser la fase de requisitos, el diseño rápido, la construcción del prototipo, la evaluación del usuario, el refinado del prototipo, y la fase de implementación y mantenimiento (Guru99, s.f.). Estas fases quedan resumidas en la siguiente ilustración:

Ilustración 4-1: Fases del modelo de prototipos [adaptado de guru99.com]

Durante la fase de requisitos, se realizaron varias reuniones sobre cómo se debería desarrollar el proyecto, así como qué requisitos debía cumplir la aplicación para poder llegar a considerarse completa y fiable. Además, se debatió sobre las tecnologías que se iban a emplear y sus motivos.

Concretamente, se empezó especificando que el entorno principal de desarrollo sería ASP.NET con Visual Studio. ASP.NET es un «*framework* para construir aplicaciones web y servicios con .NET y C#» (Documentación oficial de ASP.NET). En la actualidad .NET se está enfocando en la versión .NET Core, que es la versión más reciente y además es multiplataforma. Sin embargo, en este proyecto se decidió usar .NET Framework, que es una versión que quedará sin soporte en el futuro. La razón de esta decisión es que desde un principio se sabía que esta aplicación utilizaría un sistema WCF (*Windows Communication*

Foundation) ya en funcionamiento y el producto final se instalaría en un Windows Server con IIS, así que no resultaba necesaria la funcionalidad multiplataforma y no aportaría nada usar .NET Core.

Dicho sistema WCF es capaz de realizar un reconocimiento de una palabra introducida y, posteriormente, proporciona todas las flexiones de la palabra reconocida. Este servicio web, ya implementado, cuenta con más de doscientos cincuenta mil lemas y más de seis millones de palabras del español.

La segunda fase, el diseño rápido, consistió en definir qué componentes y funcionalidades debía tener el proyecto antes de conectarlo al servicio WCF. Se decidió crear primeramente un esquema de la aplicación web, formando la estructura (cabecera, cuerpo, pie de página, etc.) que contendría más adelante las funcionalidades. Además, se creó un archivo local (en formato JSON) con unas cuantas flexiones a modo de ejemplo para cargarlas en la aplicación. Una vez revisada esta estructura con el profesor, se pasó al siguiente paso que sería conectar la aplicación al servicio WCF.

En este punto del desarrollo empezó la tercera fase llamada construcción del prototipo, teniendo ya una conexión establecida con el servicio WCF. Se aprendió cómo usar este servicio con la ayuda del tutor que guio y dio pautas sobre el uso de esta herramienta. La respuesta que se obtiene de dicho servicio cuando se realiza una consulta, tiene la siguiente estructura (ejemplo simplificado con la palabra «gato»):

[0]	*[1]*	*[2]*
…	…	…
Frecuencia: 23516	*Frecuencia: 20422*	*Frecuencia: 23516*
IdFlexion: 201	*IdFlexion: 201*	*IdFlexion: 201*
IdFormaCanonica: 65551	*IdFormaCanonica: 228426*	*IdFormaCanonica: 228425*
…	…	…

Tabla 4-1: Resultado de la consulta «gato» en el servicio WCF

Se puede observar que, entre otros campos, se reciben datos como la frecuencia de la palabra reconocida, un código para su flexión (en este caso 201 indica que las flexiones reconocidas son masculinos singulares) y un identificador de la forma canónica. La forma canónica de una palabra es su lema, es decir, la palabra base que encontraríamos en el diccionario. En este ejemplo aparecen tres formas canónicas porque cada una tiene su acepción diferenciada, aunque compartan lema. Para diferenciarlas, esta respuesta contiene un campo llamado `InfoCanonica` donde se tiene aún más información sobre cada palabra reconocida. Con esa información, podemos diferenciar estas tres palabras en: «gato» (sustantivo), «gato» (adjetivo) y «gato» (sustantivo masculino), donde cada una tiene su propio conjunto de flexiones que no necesariamente tiene por qué ser el mismo.

Una vez analizados todos los campos de información que se pueden usar y qué significan, comencé a pensar cuáles de ellos me resultarían más útiles a la hora de crear la aplicación. Los campos que aparecen en la tabla son los campos que consideré más relevantes ya que son los que contienen información sobre la flexión en concreto, el identificador de la forma canónica y la frecuencia de uso que me serviría para ordenar las más utilizadas. Cabe destacar también la relevancia del campo `InfoCanonica` mencionado anteriormente, para discernir y aclarar las diferencias existentes entre palabras con el mismo lema, además de información adicional dentro del campo que será de utilidad para la aplicación.

Una vez recopilada esta información, se consideró cómo separar y agrupar las categorías de las flexiones para mostrarlas en tarjetas en la aplicación. Todos los códigos de las flexiones están debidamente separados en categorías numéricas, por ejemplo, todos los tipos de diminutivos se encuentran en el rango 1001-1520. Sin embargo, algunas categorías como los superlativos se encuentran en el rango 501-551 y otras en rangos de diferente longitud. Es por ello por lo que separarlas mediante código debía realizarse mediante un algoritmo. Inicialmente se creó un

algoritmo que fuera separando por centenas cada categoría de flexión. Esto funcionó con algunas categorías que ocupaban rango de centenas, pero había algunas que ocupaban rango de millares, así que poco a poco se fue refinando el algoritmo para que funcionase en todos los casos.

Este algoritmo parte de una lista de tipo `InfoFlexionador`, que es una de las clases del servicio WCF que contiene la información que devuelve una consulta de una forma canónica. Esta lista inicial contiene todas las categorías de las flexiones de la palabra dada (diminutivos, aumentativos, etc.), así que se decidió crear una lista de listas llamada `infoFlexionGrupos` que contendrá cada categoría agrupada por separado. En este diagrama (ilustración 4-2) se puede apreciar de forma simplificada cómo se separa por grupos:

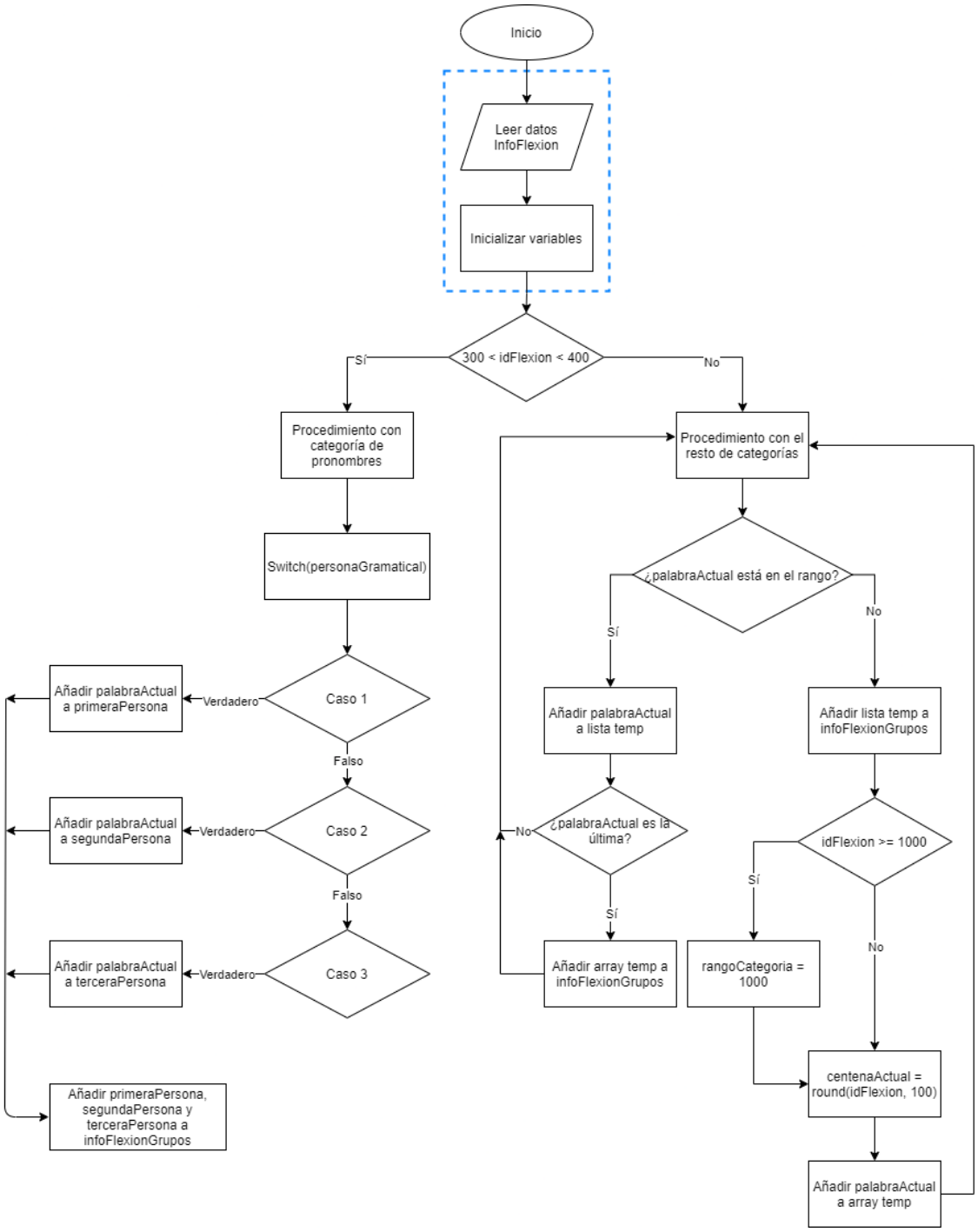

Ilustración 4-2: Diagrama del algoritmo «AgrupaInfoFlexion»

Se puede observar que existe un tratamiento diferente para la categoría de pronombres personales con respecto al resto de categorías. Esto es debido a que la disposición de los pronombres personales en la aplicación es diferente. Los pronombres se representan en tablas separados por sus personas gramaticales,

mientras que el resto de categorías (diminutivos, superlativos, despectivos, etc.) vienen representadas en forma de lista separados por género y número.

Con esta agrupación en forma de lista de listas se pueden mostrar todos los elementos en pantalla mediante el uso de bucles anidados.

En el diagrama se ve una sección delimitada en color azul al inicio. Esa parte queda explicada más en detalle en el siguiente diagrama:

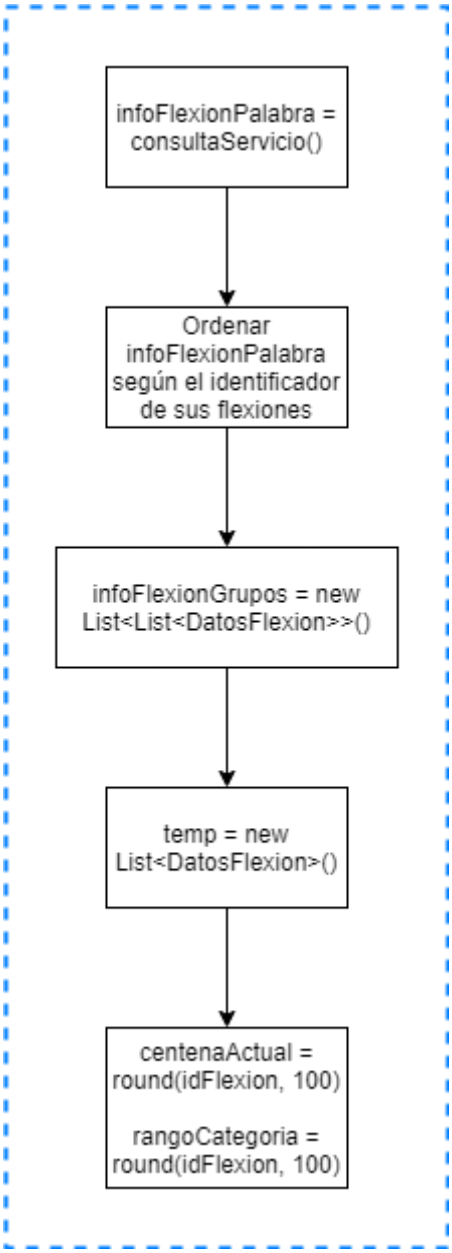

Ilustración 4-3: Detalle del diagrama de la ilustración 4-2

Esta sección es crucial para el buen funcionamiento del algoritmo en todos los casos posibles. Esto se debe a que las variables *centenaActual* y *rangoCategoria* son las que marcan los rangos de agrupación y ayudan a que la lista final quede bien separada en sus respectivos grupos. Se puede observar que estas variables son inicializadas según el código de identificación de la flexión actual (la primera, ya que se ordenó previamente la lista).

Este algoritmo es el principal del proyecto, ya que la mayoría del resto del código son métodos para devolver datos. Es por ello, por lo que debía estar refinado para todos los casos posibles y que funcionase sin problemas. Sin esta separación por grupos, sería complicado mostrar todos esos datos en pantalla.

En código C#, la parte inicial del algoritmo resulta de la siguiente manera:

```
infoFlexionPalabra =
       servicioLematizacion.ConsultaFlexionador(palabrasReconocidas[posicion].
       IdFormaCanonica);

infoFlexionPalabra.Sort((x, y) => x.IdFlexion.CompareTo(y.IdFlexion));

infoFlexionGrupos = new List<List<InfoFlexionador>>();

List<InfoFlexionador> temp = new List<InfoFlexionador>();

int centenaActual =
       (int)(Math.Round(infoFlexionPalabra[0].IdFlexion / 100d, 0) * 100);

int rangoCategoria =
       (int)(Math.Round(infoFlexionPalabra[0].IdFlexion / 100d, 0) * 100);
```

Aquí se inicializan algunas variables: *infoFlexionPalabra* (que contiene la información de la flexión en la posición *posicion*), *infoFlexionGrupos* (que es una lista de grupos de categorías de flexiones), *temp* (una lista temporal que agrupa las flexiones según sus categorías), *centenaActual* (un entero que sirve para comprobar la categoría de flexión en

la que se encuentra el bucle) y *rangoCategoria* (que sirve para aumentar el rango de las categorías, ya que a partir de los diminutivos las categorías abarcan más de una centena). Además, se aprecia que se ordena numéricamente *infoFlexionPalabra* según las categorías de las flexiones.

En este punto es donde se bifurca el algoritmo dependiendo si la flexión es un pronombre personal o no, en el caso de que lo sea se ejecuta lo siguiente:

```
List<InfoFlexionador> primeraPersona = new List<InfoFlexionador>();

List<InfoFlexionador> segundaPersona = new List<InfoFlexionador>();

List<InfoFlexionador> terceraPersona = new List<InfoFlexionador>();

esPronombrePersonal = true;

for (int i = 0; i < infoFlexionPalabra.Count; i++)
{
switch(int.Parse(DarFlexion(infoFlexionPalabra[i].IdFlexion)[0].ToString()))
      {
            case 1:
                  primeraPersona.Add(infoFlexionPalabra[i]);
                  continue;
            case 2:
                  segundaPersona.Add(infoFlexionPalabra[i]);
                  continue;
            case 3:
                  terceraPersona.Add(infoFlexionPalabra[i]);
                  continue;
            default:
                  return;
      }
}

infoFlexionGrupos.Add(primeraPersona);

infoFlexionGrupos.Add(segundaPersona);

infoFlexionGrupos.Add(terceraPersona);
```

Se puede apreciar que se crean tres subgrupos con cada persona gramatical y, una vez finalizado el bucle, se agregan todos estos subgrupos a *infoFlexionGrupos*.

Si la flexión es cualquier otra que no sea un pronombre personal, se ejecuta este fragmento de código:

```
for (int i = 0; i < infoFlexionPalabra.Count; i++)
{
	if (infoFlexionPalabra[i].IdFlexion >= centenaActual &&
	infoFlexionPalabra[i].IdFlexion < centenaActual + rangoCategoria)
	{
		temp.Add(infoFlexionPalabra[i]);

		if (i == infoFlexionPalabra.Count - 1 && temp != null)
		{
			infoFlexionGrupos.Add(temp);
		}
	}
	else
	{
		infoFlexionGrupos.Add(temp);

		temp = new List<InfoFlexionador>();

		if (infoFlexionPalabra[i].IdFlexion >= 1000)
		{
			rangoCategoria = 1000;
		}

		centenaActual =
		(int)(Math.Round(infoFlexionPalabra[i].IdFlexion / 100d, 0) *
		100);

		temp.Add(infoFlexionPalabra[i]);
	}
}
```

La primera condición comprueba que todavía se está dentro de la centena actual y del rango actual. Si es así, se añade la flexión a la lista temporal *temp* y, si la flexión actual es la última de la lista, se añade *temp* a la lista *infoFlexionGrupos*.

Si ya no se está dentro del rango, en primer lugar, se añade a *infoFlexionGrupos* la lista temporal *temp*. Después se vacía la lista temporal para la siguiente iteración y se comprueba si hay que aumentar el rango de la categoría o no. Además, se calcula el siguiente valor de la centena actual para la siguiente iteración.

El resto de métodos de los que se habla, son métodos como *QuitaVerbos* que, como su nombre indica, quita las formas verbales de la lista de palabras reconocidas y las almacena en una lista aparte. Esto se debe a que el flexionador es eminentemente nominal y no se encarga de flexionar los verbos. Sin embargo, es capaz de detectar estas formas verbales y las almacena aparte para mostrarlas al final en una tarjeta separada. En dicha tarjeta, se enlaza a la herramienta «Conjugador TIP» de la División de Lingüística Computacional del IATEXT de la ULPGC. De esta manera se ofrece al usuario la posibilidad de conjugar el verbo que introdujo en una página externa, además de que le resalta la forma verbal concreta. Otros métodos interesantes del flexionador son *DarCabecera* y *DarCabeceraGenero*. El primero se encarga de devolver una *string* con el nombre de la categoría de la flexión (diminutivos, superlativos, etc.) mientras que el segundo es capaz de discernir entre masculino, femenino y neutro para poder hacer la subdivisión dentro de una categoría. Por ejemplo, dentro de la tarjeta de diminutivos, se encuentra una separación entre masculino y femenino, cuyas cabeceras vienen dadas por este método.

Todo estos métodos y algoritmos forman parte de lo que se conoce como *code-behind*, es decir, el código que hay detrás de la presentación. La gran ventaja de esta técnica es la separación de la lógica (en C#) de la aplicación y la presentación (principalmente en HTML), consiguiendo así un código más limpio y legible.

Sin embargo, no todo el código se encuentra en el *code-behind*, también existe una parte muy importante del código en la parte de presentación, llamado *in-line code*. Este código se aplica en medio del HTML mediante el uso de etiquetas llamadas «expresiones en línea de ASP.NET» (Microsoft, 2020). Estas etiquetas tienen la estructura `<% ... %>`, con ellas podemos incrustar código C# dentro del HTML, lo cual resulta extremadamente útil especialmente para el uso de bucles.

Estas etiquetas pueden llevar un símbolo detrás del primer símbolo de porcentaje y cada uno tiene una funcionalidad diferente (Microsoft, 2020):

- `<%= … %>` es equivalente a usar la instrucción `Response.Write(…)`
- `<%@ … %>` indica una expresión de directiva
- `<%# … %>` es una expresión de vinculación de datos (*data-binding*)
- `<%$ … %>` se utiliza para construir expresiones
- `<%-- … -- %>` indica un bloque de comentarios en el lado del servidor
- `<%: … %>` es como la primera, pero la salida está codificada en HTML

Gran parte del *in-line code* de este proyecto utiliza la última opción de estas etiquetas. Esto hace que se simplifique mucho el código en lugar de escribir `Response.Write(…)` constantemente. Esa instrucción imprime en el HTML el valor que se ponga dentro de los paréntesis, lo cual resulta muy útil para imprimir palabras, clases de estilos CSS, etc. Se utilizó esta opción en lugar de la primera porque la primera tiene vulnerabilidades de tipo XSS. La última opción mitiga esto codificando la salida en HTML (Haack, 2009).

Además, estas etiquetas permiten escribir varias líneas de código incrustado sin necesidad de tener que repetirlas en cada línea. De esta manera resulta un código mucho más legible, especialmente porque este proyecto consta de múltiples bucles anidados que pueden hacer difícil su lectura.

Como se mencionó anteriormente, para la presentación visual de la aplicación se utilizó Bootstrap, debido a la facilidad que ofrece para resolver este tipo de representaciones en tarjetas y por la experiencia previa utilizando esta biblioteca de código. Estas tarjetas se denominan «cards» en la biblioteca y son el componente principal de la aplicación. Presentan una gran utilidad debido a la necesidad de mostrar tarjetas anidadas para todas las categorías de todas las flexiones de cada forma canónica reconocida.

Por otro lado, esta tecnología ofrece varias herramientas para convertir el diseño de una página web en un diseño *responsive*. Esto se refiere a la capacidad que tiene dicho diseño de adaptarse según el tamaño de pantalla del dispositivo desde donde se esté visualizando, así como dependiendo de la orientación de este.

Así, se ha conseguido cambiar la disposición de las tarjetas en función del tamaño de pantalla, ofreciendo para el modo móvil una vista apilada de las tarjetas sin menguar las funcionalidades del diseño.

Para que esta tecnología funcione, la parte quizá más importante es incluir entre las *meta tags* una etiqueta de la siguiente forma:

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

Así, se consigue que el diseño *responsive* se comporte de forma adecuada adaptando el ancho al tamaño de la ventana del dispositivo. La escala inicial, por su parte, controla el nivel de *zoom* cuando la página se carga por primera vez (Mozilla, s.f.).

Una vez establecida esta etiqueta, se pueden utilizar todos los componentes de Bootstrap para crear un diseño acorde a las necesidades del proyecto. Aparte de las ya mencionadas «cards», existen componentes totalmente *responsive* como es

el sistema *grid* que permite organizar los elementos en una malla con contenedores, filas, columnas, etc. Este sistema se utiliza agregando clases CSS a un elemento `div`, de forma que Bootstrap pueda reconocer y aplicar los estilos necesarios. Un ejemplo de uso de Bootstrap en el flexionador es el siguiente:

```
<div class="card shadow border-secondary mt-4 m-2">
	<div class="card-header cabecera-titulo">
		<h2>IAFlex: Flexionador nominal inteligente</h2>
	</div>
	<div class="card-body">
		<div class="input-group mb-3">
			<asp:TextBox class="form-control" ID="palabraEntrada"
				runat="server">
			</asp:TextBox>
			<asp:Button class="input-group-append btn btn-secondary"
				ID="botonBusqueda" runat="server" Text="Buscar..."
				OnClick="BotonBusqueda_Click"
			/>
		</div>
	</div>
</div>
```

Con este fragmento de código se muestra la cabecera de la aplicación web. Se trata de una tarjeta que muestra el título del proyecto, además del campo de introducción de palabras para buscar. Se aprecia que un `div` con clase «card» contiene dos `div` (uno para la cabecera de la tarjeta y otro para el cuerpo de esta).

Además, se puede ver otra clase Bootstrap que es la de «input-group». Esta clase consigue agrupar al `TextBox` y al `Button`, ambos elementos de entrada que aparecen juntos en la tarjeta. De esta forma, aparece la caja de texto primero y, a su derecha, el botón de «Buscar…».

Como se vio en el diagrama del algoritmo, existen dos tipos de tratamiento de la información dependiendo de qué categoría gramatical se trate. Tras tratar la información, en la aplicación existen dos disposiciones de flexiones muy diferenciadas.

Si se trata de un pronombre personal, la disposición de las flexiones se muestra en una gran tabla con varias secciones con cada uno de los pronombres separados por persona gramatical, género y número. Si se trata de cualquier otra categoría gramatical, la disposición se basa en una tarjeta grande con varias «subtarjetas» desplegables con cada una de las subcategorías (diminutivos, aumentativos, despectivos, etc.).

En la siguiente ilustración se puede observar cuál es la disposición que se muestra al usuario cuando este introduce algún pronombre personal:

IAFlex: Flexionador nominal inteligente

ella Buscar... á é í ó ú ü ñ

Pronombre personal - (yo)

DLE

La flexión **ella** proviene del pronombre personal **yo**

Personas gramaticales

Persona	Pronombres			
1ª	Singular		Plural	
	Masculino	Femenino	Masculino	Femenino
	yo	yo	nosotros	nosotras
	me	me	nos	nos
	me	me	nos	nos
	conmigo	mí	nos	nos
	mí	conmigo	nosotros	nosotras
2ª	Singular		Plural	
	Masculino	Femenino	Masculino	Femenino
	tú	tú	vosotros	vosotras
	usted	usted	ustedes	ustedes
	vos	vos	os	os
	os	os	os	os
	te	te	vosotros	vosotras
	os	os	ustedes	ustedes
	te	te		
	ti	contigo		
	contigo	ti		
	usted	usted		
	vos	vos		
3ª	Singular		Plural	
	Masculino	Femenino	Masculino	Femenino
	él	ella	ellos	ellas
	lo	la	los	las
	se	le	se	les
	le	se	les	se
	consigo	ella	ellos	ellas
	él	consigo		
	sí	sí		

Ilustración 4-4: Disposición de pronombres personales

Se puede apreciar que algunos pronombres se ven repetidos. Esto es debido a que estos tienen funciones diferentes (función de sujeto, función de complemento directo, función de complemento preposicional, etc.). Para que no resulte confuso, cuando el usuario pasa el cursor por encima de alguno de los pronombres aparece un pequeño *tooltip* que explica su función como se observa aquí:

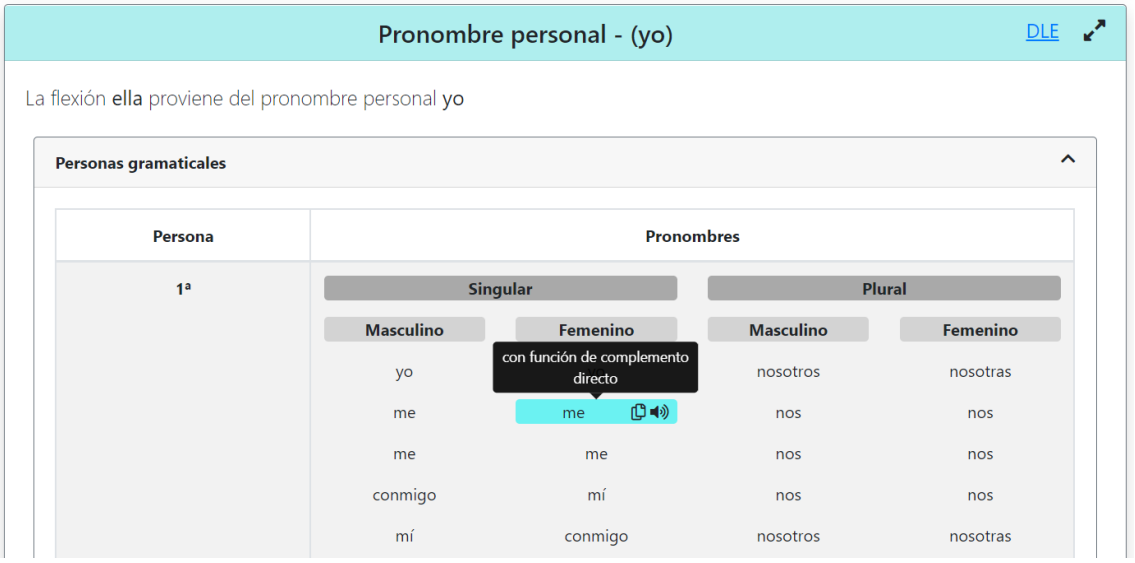

Ilustración 4-5: Tooltip con breve explicación de la función del pronombre

Además, en la ilustración 4-3 se muestra con un borde azul la flexión que se introdujo inicialmente (en este caso «ella»). Nótese la inclusión de un pequeño texto debajo de la cabecera que indica que «ella» es una flexión de la forma canónica «yo». Esto se puede observar también en el otro tipo de disposición, no solo con los pronombres personales.

Por otro lado, tenemos la disposición del resto de categorías gramaticales (sustantivos, adjetivos, etc.) que se muestra en la siguiente ilustración:

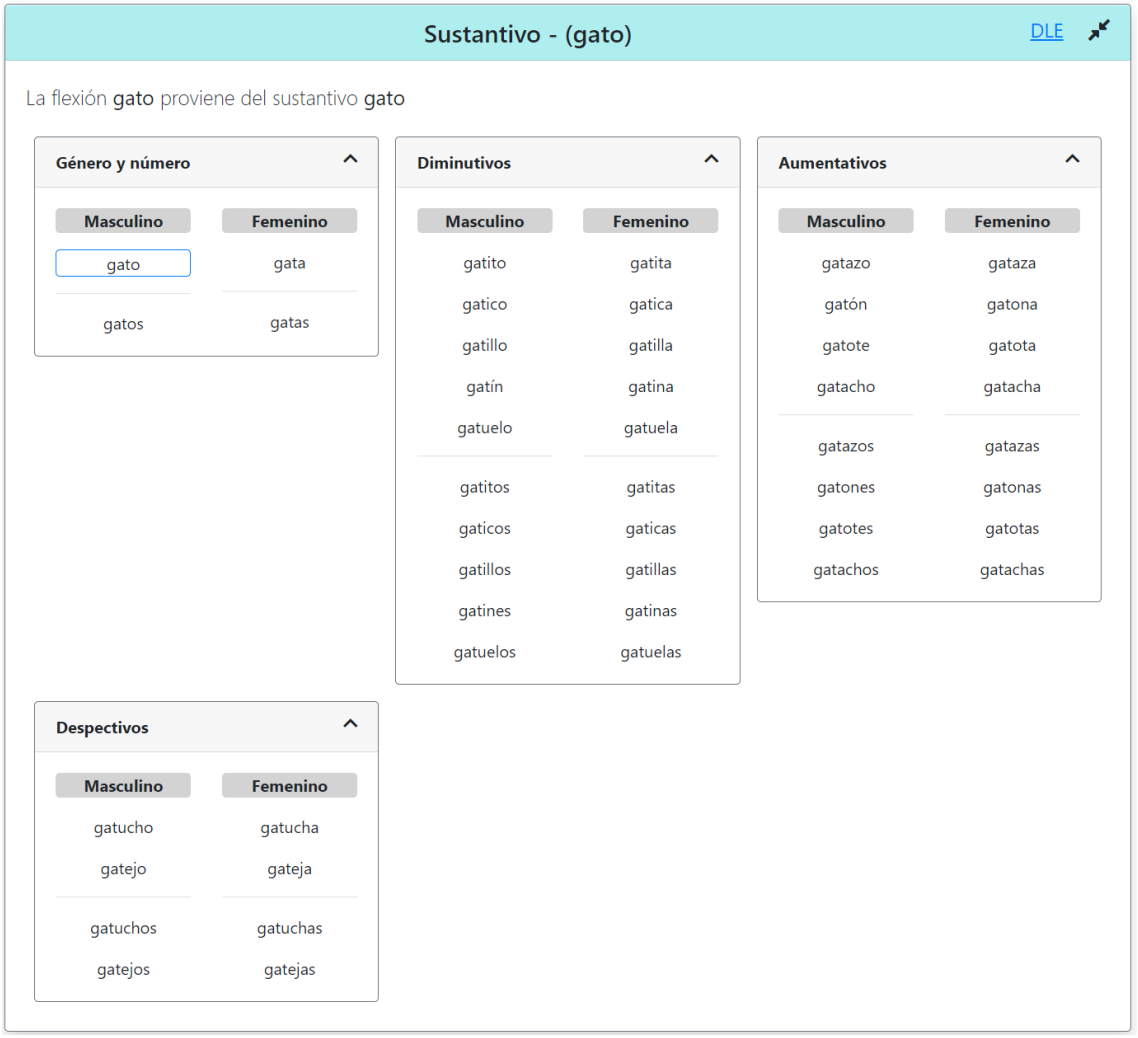

Ilustración 4-6: Disposición del resto de categorías gramaticales

En este caso se muestran las subcategorías desplegadas, organizadas en columnas por género y número. Se aprecia, además, que cada subcategoría se puede desplegar o plegar mediante la pulsación de un icono, y también arriba a la derecha se pueden expandir o contraer todas las tarjetas a la vez.

Bootstrap, así como otras dependencias que se mencionarán más adelante, están controladas por el gestor de paquetes NuGet. Este gestor permite agrupar bibliotecas de código en paquetes fácilmente utilizables desde su interfaz gráfica

(Microsoft, s.f.). Además, permite actualizar o degradar la versión de cualquiera de sus paquetes. Esto fue necesario en algunos casos donde aparecían conflictos entre versiones de paquetes que se vieron solventadas al cambiar sus versiones por unas algo más antiguas.

Entre las dependencias instaladas se encuentran: «jQuery» (para algunos pequeños *scripts* donde era necesario código jQuery), «Font Awesome» (para incluir iconos gráficos de forma sencilla, mediante el uso de clases CSS), «toastr» (para el mensaje que aparece como una notificación al copiar una flexión al portapapeles), «popper.js» (para el uso de *tooltips*, es decir, pequeños paneles explicativos que aparecen al pasar el ratón por encima de aquellos elementos que requieren una breve explicación o descripción).

Un ejemplo de uso de jQuery es el *script* que se creó para mostrar un icono de carga. Este icono solo aparece mientras se espera respuesta del servicio WCF. Así, se otorga al usuario la información de que la página no se encuentra bloqueada ni colgada, sino que está esperando por datos. Esto resulta de especial utilidad para usuarios con dispositivos antiguos o conexiones lentas, ya que generalmente tardará más la respuesta del servicio. Se tuvo en mente principalmente potenciales usuarios de Hispanoamérica, donde la conexión internet es por lo general de peor calidad.

Para comprobar que se muestra el icono en estas circunstancias, se utilizó una herramienta integrada en el navegador Chrome/Brave que permite simular la

carga de la página con un dispositivo peor. A esta herramienta se puede acceder pulsando F12:

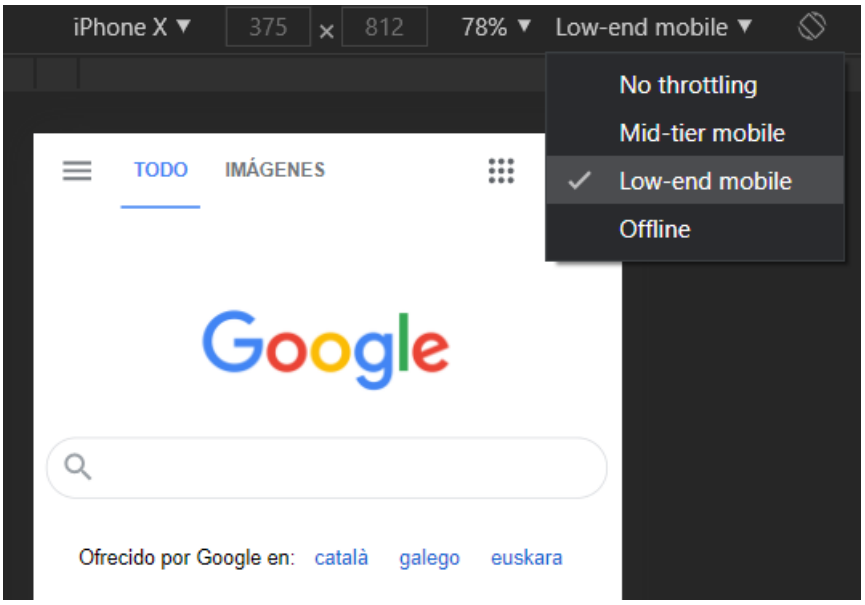

Ilustración 4-7: Simulación en navegador de carga con dispositivo lento

En cuanto a la dependencia «Font Awesome», se ha utilizado una variedad de iconos para indicar distintas funcionalidades. Precisamente en el mencionado icono de carga, se usa uno de esta galería de iconos. También se han usado iconos en los campos de las flexiones, para que cuando el usuario pase el ratón por encima de una flexión aparezcan dos iconos: uno de copia y otro de altavoz. El de copia sirve para copiar la flexión seleccionada, mientras que el del altavoz sirve para que el navegador pronuncie en audio la flexión (síntesis de habla).

El mecanismo de copia se ha realizado con un pequeño *script* que hace uso de «Clipboard API». Esta API es una tecnología que consta de un objeto «Clipboard», con el que se puede leer y escribir en el portapapeles del usuario. De esta manera, se pueden implementar fácilmente soluciones para cortar, copiar y pegar en una aplicación web (Mozilla, s.f.). Sin embargo, en navegadores más antiguos como Internet Explorer, esta tecnología no funciona. Es por ello, por lo que se agregó un caso especial para solucionar este problema. En caso de que el

navegador que el usuario esté usando en ese momento sea Internet Explorer, se ejecutará la instrucción `document.execCommand("copy")`. Esta instrucción se encuentra en estado obsoleto, pero funciona adecuadamente en este caso. Además, en ambos casos se muestra un *toast*, es decir, un pequeño mensaje de notificación que informa al usuario de si el texto se ha copiado a su portapapeles de forma satisfactoria o no.

Ilustración 4-8: Texto copiado satisfactoriamente

El mecanismo de síntesis de habla se basa también en un pequeño *script* que hace uso de la tecnología «SpeechSynthesis». Esta tecnología se usa, además de otras funcionalidades, para reproducir una pronunciación (Mozilla, s.f.).

En este *script* se recupera la flexión sobre la que se ha hecho clic y, posteriormente, se pronuncia en idioma español.

Otra funcionalidad implementada en el proyecto ha sido la de la internacionalización. Es decir, adaptar la aplicación web del flexionador para que pueda ser utilizado en otros idiomas. En este caso, se ha decidido elegir el inglés como idioma adicional. Sin embargo, debido a la forma en la que se ha implementado la internacionalización, se puede adaptar fácilmente para otros idiomas si es que fuera necesario. Se puede decir entonces que es una aplicación bastante escalable en cuanto a la internacionalización.

El funcionamiento de esta tecnología se basa en el uso de recursos globales que contienen las cadenas de caracteres que deben ser traducidas o adaptadas. Se ha creado un directorio llamado *App_GlobalResources*, y dentro del mismo se encuentran tres archivos:

- *Resource.resx*
- *Resource.es.resx*
- *Resource.en.resx*

El primero es un archivo que contiene las cadenas de caracteres que se usarán cuando el lenguaje del navegador del usuario sea otro que no sea el español o el inglés. En este caso se eligió que se muestren en español estos textos cuando el lenguaje es otro.

El segundo contiene las cadenas de caracteres en el idioma español. Estos tres archivos contienen una clave (que será usada en el código para recuperarlas) y un valor. Por último, el tercer archivo contiene las mismas claves que los demás, pero lo que cambia son los valores de las cadenas, que serán las traducciones al inglés. Esto es prueba de que la internacionalización es escalable, ya que para agregar otro idioma solo habría que añadir un nuevo archivo (por ejemplo *Resource.fr.resx*) que contuviera las adaptaciones al idioma en cuestión.

Nombre	***Valor***
…	…
Buscar	*Search...*
Cabecera	*IAFlex: Intelligent nominal inflection tool*
Conjugar	*Conjugate*
ConjugarEnTIP	*Conjugate verb in «Conjugador TIP»*
Despectivos	*Pejoratives*
Diminutivos	*Diminutives*
DLE	*Dictionary of the Spanish language*
Femenino	*Feminine*
…	…

Tabla 4-2: Fragmento del archivo Resource.en.resx

Para testar que la internacionalización funcionaba de forma adecuada, se instaló en el navegador una extensión capaz de cambiar el idioma del navegador. Al aplicar el idioma *en* se observa este resultado:

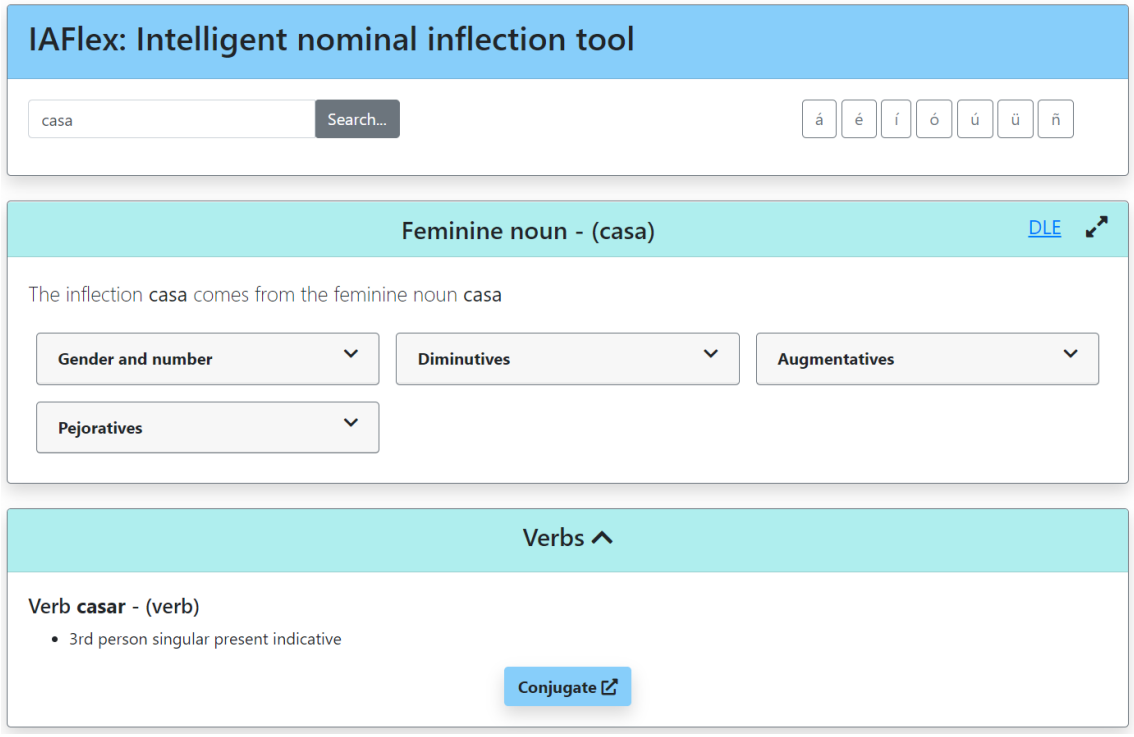

Ilustración 4-9: Ejemplo de búsqueda en inglés

Cabe destacar que, tanto las categorías gramaticales como algunos otros parámetros, se encontraban ya traducidos en el sistema WCF así que se adaptó el código para obtener esos datos dependiendo del idioma establecido por el navegador.

Por otro lado, una de las funcionalidades que también se implementó fue la de reescribir las rutas URL para mejorar el posicionamiento SEO (*Search Engine Optimization*) en cuanto a las búsquedas.

El funcionamiento se basa en aplicar reglas que aparentan generar direcciones URL con las flexiones de las palabras, que son tratadas para simular que el usuario ha introducido una flexión en concreto. Esto produce que cuando en un motor de búsqueda (como Google) muestre a un usuario los resultados de la

consulta «flexiones de niño», dicho motor mostrará una URL de la siguiente forma: *http://[...]/flexionar-niño.html?palabra=CbGvKxE3Cb*.

Esto aporta una ventaja en cuanto al posicionamiento SEO, ya que hará que Google se «crea» que la aplicación web tiene una página para la flexión «niño», cuando en realidad lo que se hace es una petición al servidor como si el usuario hubiese introducido la palabra «niño» en el buscador.

Esta técnica es ampliamente utilizada en sitios web que tienen algún tipo de consulta del usuario, como por ejemplo webs que ofrecen definiciones de palabras, sinónimos, antónimos, etc. En esta ilustración se puede observar un ejemplo:

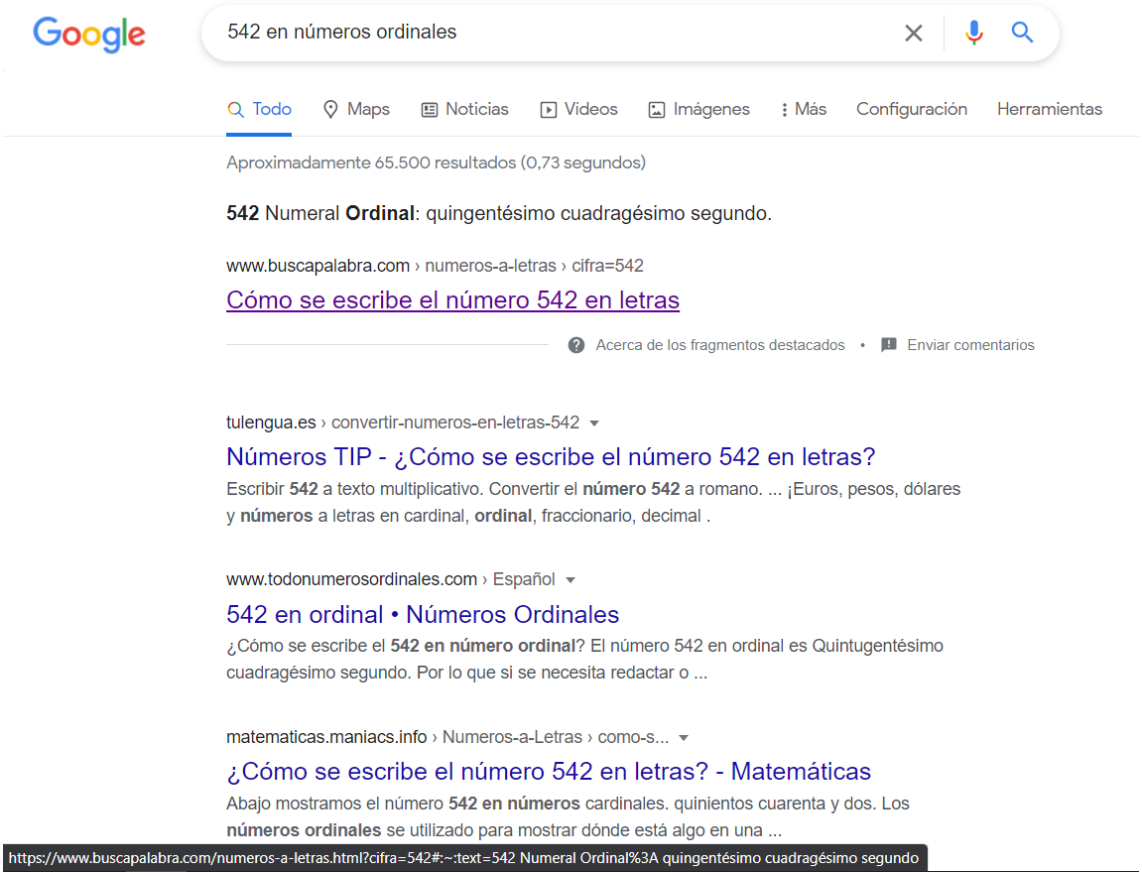

Ilustración 4-10: Explicación de posicionamiento con generación de URLs

Se aprecia en esta consulta el detalle que muestra la URL en la caja negra de la parte inferior de la ilustración. Existe en la URL un parámetro que es

?cifra=542. Esto es bastante similar a lo que se ha implementado en este proyecto. De esta forma se consigue «generar» direcciones ficticias que la aplicación en sí no tiene, pero que es capaz de gestionar para realizar la consulta al servidor que utilice.

Sin embargo, en el ejemplo del flexionador citado anteriormente (*http://[...]/flexionar-niño.html?palabra=CbGvKxE3Cb*) se observa que hay, además de la palabra «niño», un parámetro llamado «palabra». Este parámetro es un cifrado de la palabra «niño», utilizando un método personalizado de cifrado, para que los usuarios no puedan intentar obtener información de qué palabras dispone el servicio, de cómo está estructurado, de cómo atacarlo, etc. Es decir, añadir este parámetro agrega seguridad a la aplicación ya que solo si coincide la codificación de la palabra se realiza la consulta.

Capítulo 5 – Conclusiones y trabajo futuro

Durante el desarrollo de este proyecto se han logrado los objetivos planteados al inicio. Estos objetivos eran principalmente llegar a implementar una solución web útil y sencilla capaz de obtener todas las flexiones nominales de la lengua española. Además, se hizo mucho hincapié en que fuera completo, ya que los proyectos existentes de flexionadores no son del todo íntegros. Razón por la cual se probaron multitud de casos extremos que tiene el idioma español para asegurar que el producto final tiene una buena calidad.

Se pone de manifiesto, además, la gran riqueza que tiene nuestro lenguaje aportando este enorme abanico de posibilidades que da a las palabras para formarse y derivarse, dando lugar a una multitud de apreciativos que hacen de este un lenguaje diverso y rico. Y es que esta fue una de las grandes razones por las que se inició este proyecto, por querer aportar algo más a la lingüística en forma de herramienta online que todos tendrán disponible.

En fechas cercanas al inicio del desarrollo se comentó la posibilidad de incluir este proyecto en el resto de herramientas de la División de Lingüística Computacional del IATEXT de la ULPGC. En el momento de escribir esta memoria aún no se ha incluido, pero se espera que en un futuro próximo esté integrado como una herramienta más en este servicio online. Esto demuestra, por un lado, el grado de completitud de la solución y, por otro, el valor que añade a la lengua el tener una utilidad así.

Además, el proyecto presenta una gran escalabilidad al estar desarrollado con tecnologías modulares y bastante conocidas. Especialmente con el caso de Bootstrap, que aporta modularidad en cuanto al aspecto de las hojas de estilos, lo

cual hará que se pueda ajustar el diseño al que tienen el resto de herramientas de la División de Lingüística anteriormente mencionada.

Otras adiciones que se podrían realizar dentro de este trabajo en un futuro tras la entrega de esta memoria serían:

- Accesibilidad: añadiendo más utilidades para las personas con algún tipo de discapacidad
- Protección frente a ataques: proteger el punto de entrada al servicio WCF para evitar ataques de denegación de servicio (*DDoS*)
- Incorporar un corrector o sugerencias ante palabras mal escritas por el usuario

Bibliografía

- Guru99. (s.f.). *Prototyping Model in Software Engineering: Methodology, Process, Approach*. Recuperado el 1 de mayo de 2021, de Guru99: https://www.guru99.com/software-engineering-prototyping-model.html
- Haack, P. (25 de septiembre de 2009). *Html Encoding Code Blocks With ASP.NET 4*. Recuperado el 3 de mayo de 2021, de You've Been Haacked: http://haacked.com/archive/2009/09/25/html-encoding-code-nuggets.aspx/
- Microsoft. (2017). *What Is Windows Communication Foundation*. Recuperado el 17 de abril de 2021, de Microsoft Docs: https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf
- Microsoft. (26 de marzo de 2020). *Expresiones en línea de ASP.NET en .NET Framework*. Recuperado el 3 de mayo de 2021, de Microsoft Docs: https://docs.microsoft.com/es-ES/troubleshoot/aspnet/inline-expressions
- Microsoft. (s.f.). *NuGet documentation*. Recuperado el 8 de mayo de 2021, de Microsoft Docs: https://docs.microsoft.com/en-us/nuget/
- Mozilla. (s.f.). *Navigator.clipboard*. Recuperado el 11 de mayo de 2021, de MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/Navigator/clipboard
- Mozilla. (s.f.). *SpeechSynthesis*. Recuperado el 11 de mayo de 2021, de MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/API/SpeechSynthesis
- Mozilla. (s.f.). *Using the viewport meta tag to control layout on mobile browsers*. Recuperado el 8 de mayo de 2021, de MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag
- Navarro, J. (2017). *Definición de Flexión (Lingüística)*. Recuperado el 20 de abril de 2021, de Definición ABC: https://www.definicionabc.com/general/flexion-linguistica.php

- Universidad Complutense Madrid. (s.f.). *Flexión nominal*. Recuperado el 20 de abril de 2021, de Plataforma gramatical de enseñanza de español como lengua extranjera: https://www.ucm.es/plataformaele/flexion-nominal