Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Aplicación web para la visualización y consulta de leyes

Web application for visualization and query laws

Jose Ramón Casero Fuentes

La Laguna, 2 de julio de 2020

Dña **María Elena Sánchez Nielsen**, con N.I.F. 42848599J profesora Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

C E R T I F I C A

Que la presente memoria titulada:

"Aplicación Web para la Visualización y Consulta de Leyes"

ha sido realizada bajo su dirección por D. **Jose Ramón Casero Fuentes**,

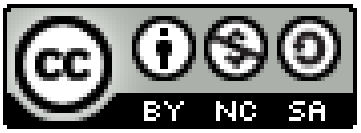
con N.I.F.25622041-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 2 de julio de 2020

Agradecimientos

Agradezco a mi familia por que me ha apoyado durante la realización de mi grado en todo y más ahora en el momento de realizar este Trabajo de Fin de Grado en tiempos de confinamiento. A todos mis amigos que se han convertido en un pilar imprescindible para mí en estos cuatro años de la carrera. A toda la gente que me apoyó y que estuvo para mí en algún momento, gracias, porque vaya donde vaya una parte de vosotros va conmigo

Licencia



Resumen

Actualmente, el volumen masivo de leyes que se tramitan y su naturaleza extensa presentan nuevos desafíos en su transformación digital. Entre ellos, se encuentran la identificación y reconocimiento de forma automática con el fin de poder llevar a cabo la clasificación de dichos documentos, para posteriormente poder procesarlos de forma automática.

El objetivo de este TFG es plantear una solución que permita la identificación y reconocimiento automático de la estructura de una ley para posteriormente permitir la consulta y búsqueda de forma sencilla y amigable. Como validación de esta prueba de concepto se ha implementado una aplicación Web que permite la importación de una ley en formato PDF(formato de documento portable), mostrar su visualización jerárquica, así como la realización de búsquedas en función de expresiones regulares y palabras claves.

Dicha aplicación ha sido validada en el contexto de las leyes del Parlamento de Canarias.

Palabras clave: Visualización, ley, expresiones regulares, procesamiento de lenguaje natural

Abstract

Nowadays, the massive number of laws of different environments are a big challenge for the digital transformation. The identification and recognition of the documents in order to classify and process them automatically is a big problem on the digital revolution.

The goal of this is work is to present a basic solution to identify and recognise the structure of a law document in order to allow querying and searching on a user friendly way. To validate this system, a web application has been created to add new laws in PDF, visualizate hierarchically and querying using regular expressions and keywords.

The applications has being developed in the context of the law of the "Parlamento de Canarias".

Keywords: Visualization, law, regular expressions, natural language processing

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Justificación

En el ámbito del derecho, la mayoría de sistemas informáticos que encontramos llevan muchos años entre nosotros, esto hace que sean sistemas relativamente antiguos con respecto a la transformación digital llevada a cabo actualmente en entornos organizacionales. Esto se traduce a que las plataformas virtuales donde podemos obtener las leyes, suelan gozar de interfaces poco intuitivas, así como los métodos de visualización se caractericen por la utilización de un formato universal y nada especializado. Asimismo, las diferentes leyes suelen ser documentos donde la localización de información no es sencilla.

Por ello, en este TFG trataremos de plantear una solución que nos permita visualizar dichas leyes de una forma más óptima, para poder obtener la información de las mismas de forma más intuitiva y rápida.

Este campo abarca un ámbito demasiado extenso para un único Trabajo de Fin de Grado, por lo que se propone una solución inicial en uno de sus apartados, que sirva para establecer una aproximación inicial para la transformación digital de dicho problema.

1.2 Esquema

En este documento podemos enumerar las siguientes partes:

- **Capítulo 2 Estado del Arte**: se aborda el estado actual en el que se encuentran las distintas plataformas y métodos de visualización de leyes, haciendo énfasis en aquellos puntos que consideramos a mejorar para nuestro sistema.
- **Capítulo 3 Metodología**: se explica el método seguido durante el desarrollo del proyecto, tanto en la etapa de investigación, como en la de creación del sistema, como en las conclusiones finales.
- **Capítulo 4 Diseño y desarrollo:** en este apartado se nombran las distintas tecnologías usadas para el desarrollo del sistema con sus distintas características, por las cuales son elegidas, además de las cuestiones que se han planteado durante el desarrollo y qué soluciones se les han dado.
- **Capítulo 5 Conclusiones y líneas futuras**: se evalúan las distintas partes del sistema conseguido y cómo mejorarlos. Se plantea futuras funcionalidades que se podrían añadir en el desarrollo del proyecto y el porqué de su necesidad. Finalmente se explica la solución final que se plantea de cara a la modernización de las plataformas relacionadas con el ámbito del derecho.
- **Capítulo 6 Presupuesto:** donde se hace un desglose del tiempo de trabajo realizado para cada una de las partes del proyecto y el coste del mismo. Además se añade el coste de un posible despliegue en un entorno real del

aplicativo.

- **Capítulo 7 Algoritmos:** se expone el código de algunos de los algoritmos utilizados para el desarrollo del aplicativo. Que se hacen mención en otras partes del documento.

1.3 Objetivos

El proyecto en cuestión, consiste en elaborar una herramienta de visualización de leyes que se desarrolle en el ámbito de las leyes del Parlamento Canario, sirviendo como solución inicial de cara a una transformación del ámbito de la visualización. Para ello esta herramienta debe de ser capaz de cumplir los siguientes puntos:

- Debe de tener una visualización amigable de la ley asimismo de partes de ella para aquellos usuarios que tratan con leyes actualmente, para que la transición a este sistema sea lo más cómoda posible. La aplicación debe de ser lo más intuitiva posible, teniendo una interfaz sencilla y adecuada.
- Será importante tener nuevos métodos de consulta debido a que los métodos actuales suelen ser bastante básicos y no permiten mucho juego. Para ello es interesante añadir procesamiento de lenguaje natural para mejorar las consultas.
- Nuevas formas de visualización, en las que podamos obtener mayor cantidad de información de forma fácil, un ejemplo de ello sería la implementación de los wordclouds para obtener una previsualización de las palabras más importantes de un documento para reconocer rápidamente la temática.
- El funcionamiento con el mayor número de leyes es un requisito esencial para hacer un sistema lo más general posible pudiéndose usar en la mayor cantidad de documentos posibles.

Como vemos este sistema debe de estar enfocado en los usuarios, aquellas personas que trabajan en el ámbito de las leyes. Además debemos de mejorar lo que ya tenemos ofreciendo un valor añadido que sería la funcionalidad de consulta o los nuevos métodos de visualización.

Capítulo 2 Estado del arte

Hoy en día en el ámbito del derecho la mayoría de herramientas utilizadas permanecen sin cambios desde hace muchos años, haciendo que queden anticuadas y no se hayan visto beneficiadas de los avances de la tecnología.

Cuando hablamos concretamente de la visualización de leyes, el formato que tenemos es un formato que lleva usándose universalmente desde hace mucho tiempo debido a sus muchas ventajas, pero al ser un formato universal no añade funcionalidades específicas o no se aprovecha de las características específicas de dicho ámbito.

IX legislatura
Año 2016

Parlamento de Canarias

Número 291
22 de septiembre

BOLETÍN OFICIAL

El texto del Boletín Oficial del Parlamento de Canarias puede ser consultado gratuitamente a través de Internet en la siguiente dirección: http://www.parcan.es

SUMARIO

PROYECTOS DE LEY

EN TRÁMITE
9L/PL-0003 Del suelo de Canarias. Página 1

PROYECTO DE LEY

EN TRÁMITE
9L/PL-0003 ***Del suelo de Canarias.***
(Registro de entrada núm. 6563, de 9/8/16).
Presidencia
La Mesa del Parlamento, en reunión celebrada el día 1 de septiembre de 2016, adoptó el acuerdo que se indica respecto del asunto de referencia:
1.- PROYECTOS DE LEY
1.1.- Del suelo de Canarias.
Acuerdo:
1.- En conformidad con lo establecido en el artículo 122 del Reglamento de la Cámara, se acuerda admitir a trámite el proyecto de ley de referencia y ordenar su publicación en el Boletín Oficial del Parlamento. Dicho proyecto de ley se acompaña de una exposición de motivos y de los siguientes antecedentes: acuerdo del Consejo de Gobierno, dictamen del Consejo Consultivo de Canarias y documentación complementaria, que quedan a disposición de los señores diputados, para su consulta, en la Secretaría General del Parlamento.
De este acuerdo se dará traslado al Gobierno y a los grupos parlamentarios.
En ejecución de dicho acuerdo y en conformidad con lo previsto en el artículo 107 del Reglamento del Parlamento de Canarias, dispongo su publicación en el Boletín Oficial del Parlamento.
En la sede del Parlamento, a 1 de septiembre de 2016.- PD EL SECRETARIO GENERAL, Salvador Iglesias Machado.

Figura 1: Visualización del formato PDF de un proyecto de ley del Parlamento de Canarias

"Las siglas PDF vienen del inglés *Portable Document Format*, es un formato de almacenamiento de documentos digitales independiente de plataformas de software o hardware. Sus principales ventajas son que es multiplataforma, puede compartirse entre distintos dispositivos sin que se modifique su aspecto ni su estructura. Puede contener cualquier combinación de texto, elementos multimedia, vínculos etc. Es un formato muy extendido, puede cifrarse, se puede generar prácticamente desde cualquier aplicación" (1).

Es un formato que lleva utilizándose desde hace mucho tiempo de forma universal, con lo cual no se aprovecha de las distintas características específicas de los documentos de ley y no añade funcionalidades específicas para estos que los harían mucho más útiles.

La visualización del mismo y las características que disponemos no dependen de sí mismo, sino que depende de un visor de terceros que no nos asegura gran utilidad. Algunas utilidades que deberían de ser imprescindibles en cualquier visualización sería añadir la posibilidad de poner puntos clave dentro de un documento para acceder rápidamente a las distintas partes en las que estamos trabajando. La posibilidad de realizar búsquedas avanzadas dentro del PDF daría mayor flexibilidad para obtener la información cuando conozcamos palabras clave, expresiones o cualquier información que podamos tener.

Una vez presentado el formato más extendido, se muestra a continuación otras alternativas planteadas para la visualización de leyes

US Code Visualization

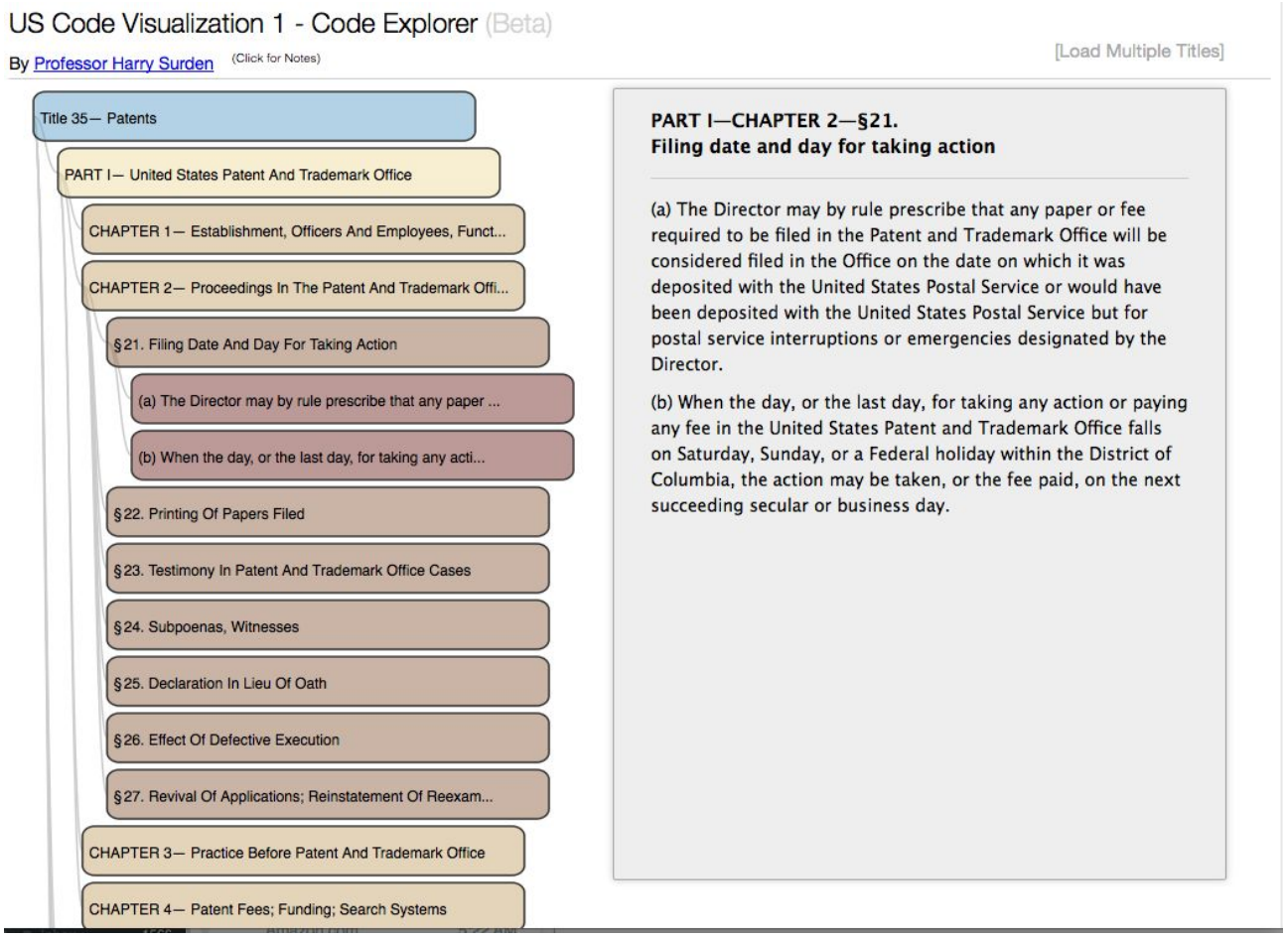


Figura 2: US Code Visualization

Esta visualización permite ver las distintas partes de una ley en una estructura de árbol, hacer click en cada una de ella y se mostrarían sus subsecciones, además del nodo clicado en grande al lado suya. No dispone de más características, por ejemplo carece de

un buscador dentro de la misma ley o de la capacidad de visualizar aumentados diferentes artículos a la vez. Es una aplicación simple, que además no se encuentra adaptada a ninguna ley aparte del código de Estados Unidos.

Constitución de la República Portuguesa

Figura 3: *NSM Viz Constitución de la República Portuguesa, (3)*

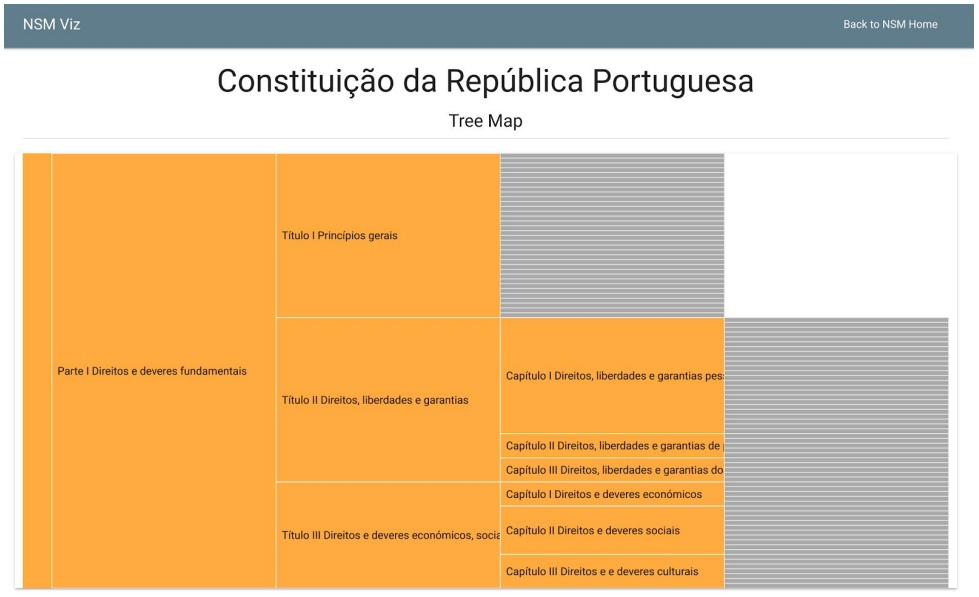


Figura 4: *NSM Viz Constitución de la República Portuguesa, Zoom(3)*

Mantiene una estructura similar a la que hemos podido ver en la herramienta anterior. Cuando hacemos click en una de las partes nos muestra esta en grande y nos

amplía la vista a sus subsecciones, dispone de un botón a la izquierda para volver al estado anterior. Como en el anterior, carece de una barra de búsqueda para localizar las distintas partes más rápidamente.

Capítulo 3 Metodología

Para realizar una planificación del proyecto, en primer lugar planteamos cuáles serían los objetivos que queríamos conseguir. Una vez esto, definimos un conjunto de tareas correspondiente a los objetivos. Durante el desarrollo de dichas tareas se irían realizando reuniones con el fin de ver la evolución del proyecto y planear los siguientes cambios, por lo que se podría decir que se ha seguido una metodología **scrum**. Veamos las distintas etapas de las que consta el proyecto:

3.1 Estudio previo y planteamiento

En esta tarea realizamos el estudio de las diferentes herramientas que se utilizarían durante el desarrollo. Tendremos que **analizar la estructura** de los diferentes documentos de ley, la estructura de cada token, para poder finalmente realizar el análisis del texto. Además tendremos que decidir el método de consulta y visualización que queremos para el sistema.

Esta etapa se realiza en el comienzo del proyecto para marcar el camino a seguir durante el mismo.

Etapa	Fecha inicio	Fecha fin
Análisis estructura	10/03/2020	17/03/2020
Análisis visualización	01/04/2020	27/04/2020

Tabla 1: Etapas estudio previo y planteamiento

3.2 Implementación

Esta etapa lleva la mayor parte del tiempo del proyecto, está dividida en las distintas partes que marcan el desarrollo del mismo. La primera parte es el desarrollo del sistema de **análisis de texto** en python, esta etapa incluye el procesamiento de texto, que consta de extracción, limpieza y tokenización (extracción de las distintas partes del texto). Posteriormente se realizará la creación de la estructura de datos que almacenará los distintos tokens extraídos anteriormente. En la segunda parte, se creará el sistema de visualización del texto de las leyes, el cual consta de una aplicación cliente-servidor, para ello desarrollaremos una **API**, en la cual se realizará todo el procesamiento de texto, y el **cliente**, que realizará las correspondientes consultas, el trabajo en este sistema permitirá detectar errores sobre el algoritmo original y realizar cambios sobre el mismo. En ambas partes realizaremos en paralelo tareas de investigación sobre las tecnologías que estemos realizando y de corrección de errores

Etapa	Fecha inicio	Fecha fin
Procesamiento de texto	17/03/2020	01/04/2020
Estructura de datos	23/03/2020	01/04/2020
Consultas	27/04/2020	01/04/2020
API	27/04/2020	-
Aplicación Web	27/04/2020	-

Tabla 2: Etapas implementación

3.3 Evaluación del funcionamiento

El proyecto se encuentra en fase final, comprobaremos el correcto funcionamiento de las distintas funcionalidades que incluye el sistema. Además se realizará una corrección de errores sobre los fallos que se encuentren, una depuración completa de las funcionalidades para conseguir el sistema final.

Etapa	Fecha inicio	Fecha fin
Prueba de aplicación	15/06/2020	-
Corrección de errores	15/06/2020	-

Tabla 3: Etapas evaluación del funcionamiento

Capítulo 4 Diseño y desarrollo

El diseño y el desarrollo de la aplicación ha estado marcado por dos etapas principales. En primer lugar una etapa de **procesamiento de texto** en la que se realizó la obtención del texto del archivo que contiene la ley, una tokenización del texto en la que obtenemos la principales partes de la ley, la elaboración de una estructura completa en la que almacenar la ley adecuada la misma. En esta fase el desarrollo sería prácticamente completa en **python**, debido a la gran flexibilidad que nos da como lenguaje. En segundo lugar una etapa de **visualización** de la ley de forma dinámica, para ello creamos una **api** con la capacidad de contener todos los métodos de la primera etapa, para realizar futuras consultas sobre ellos. Nos encargamos de desarrollar una **aplicación web**, capaz de consumir dicha api, para representar la estructura obtenida de la forma más óptima. Además de implementar otros métodos como la visualización en un texto plano optimizado o la elaboración de wordcloud. En los siguientes apartados, se describen cada uno de los aspectos principales.

4.1 Procesamiento de texto

En este apartado se explicarán las distintas herramientas utilizadas para la obtención del texto de una ley, la división del texto en los distintos tokens y la elaboración de la estructura final. Además se hablará de las distintas partes del algoritmo implementado y su justificación.

4.1.1 Extracción de texto

Habitualmente las leyes se encuentran en formato PDF, la librería elegida para obtener el texto sería pdftotext.

pdftotext

pdftotext es una librería muy sencilla y a la vez muy potente. Nos permite convertir archivos PDF a texto sin formato, con la capacidad de seleccionar un rango de páginas, mantener el diseño original del texto lo mejor posible e incluso trabajar con PDF protegidos. Tiene una gran precisión a la hora de leer el texto, lo cual nos ayuda mucho en el momento de realizar la división en tokens.

```
def read_text_from_pfd(filepath):
    with open(filepath, "rb") as f:
        pdf = pdftotext.PDF(f)
    text = ""
    for page in pdf:
        text += page
    return text
```

Figura 5: *Read text from pdf*

Como podemos ver el algoritmo de extracción de texto que nos queda es bastante sencillo, pdftotext dispone de un método de carga del PDF, y una vez cargado nos permite realizar una textura por páginas, realizando de esta manera una concatenación de las mismas, una vez esto tenemos el texto completo y podemos que proceder a realizar la tokenización del mismo.

4.1.2 Obtención de tokens

El proceso de obtención de los tokens estuvo compuesto por varias fases. En primer lugar limpiar las partes del texto que no nos aportaba información como por ejemplo los piés de página o encabezado de cada una de las páginas. Para esto empleamos distintas expresiones regulares.apartados

Las **expresiones regulares (4)** son patrones de coincidencia de texto descritos con una sintaxis formal. Los patrones se interpretan como un conjunto de instrucciones, que luego se ejecutan con una cadena como entrada para producir una subconjunto de coincidencia o una versión modificada del original. El término «expresiones regulares» con frecuencia se acorta en conversación a «regex» o «regexp» . Las expresiones pueden incluir correspondencia de texto literal, repetición, composición de patrones, ramificación y otras reglas sofisticadas. Una gran cantidad de problemas de análisis son más fáciles de resolver con una expresión regular que mediante la creación de un analizador léxico de propósito especial y un analizador sintáctico.Las expresiones regulares se usan normalmente en aplicaciones que implican procesamiento de una gran cantidad de texto". En Python la librería principal de expresiones regulares se llama **re**, para realizar dicha limpieza utilizamos un método de esta librería, el cual nos permite introducir el patrón que queremos eliminar y en este caso lo sustituimos por un string vacío ya que los espacios alrededor de dicha expresión se mantienen, veamos un ejemplo.

```
text = re.sub(r"[0-9]+ de [a-zA-Z]+ de [0-9][0-9][0-9][0-9] Boletín Oficial del Parlamento de Canarias", "", text)
```

Figura 6: Re sub elimina encabezados

Esta expresión nos permite generalizar la limpieza del encabezado en los distintos documentos de ley procedentes del Boletín Oficial del Parlamento de Canarias. Acepta expresiones del tipo `12 de junio de 2018 Boletín Oficial del Parlamento de Canarias` o `11 de marzo de 2019 Boletín Oficial del Parlamento de Canarias`.

Una vez finalizada podemos proceder por fin a realizar la tokenización, división en partes, del texto de una ley. Para este proceso necesitaría una librería de procesamiento de lenguaje natural. Donde tenemos como principal exponente **NLTK**.

NLTK (Natural Language Toolkit)

Es una plataforma líder para realizar programas en Python que trabajan con el lenguaje humano. Nos provee de una gran variedad de características como sería la clasificación, tokenización, lematización, tagging, análisis o razonamiento semántico. Esta plataforma es adecuada para lingüistas, ingenieros, estudiantes... gracias a su API, la cual es muy fácil de usar y comprender. De ella inicialmente nos interesaba su capacidad de dividir el texto en distintas frases utilizando una división adaptada al lenguaje(en nuestro caso español), ya que el objetivo era utilizarla para la tokenización.

Este tipo de tokenización según el sentido de las frases no era lo suficientemente precisa, con lo cual decidí optar por otra tokenización distinta de la misma librería, el método **word_tokenize,** este método realiza una división por los espacios, saltos de línea y los distintos signos de puntuación del español, como sería el punto, la coma, signos de interrogación, etc. Echémosle un ojo a su rendimiento:

Número de palabras	Tiempo(s)
11888	0.05373644828796387
20912	0.08794713020324707
50528	0.23798513412475586
135484	0.6478910446166992

Tabla 4: Tiempos tokenización

Como podemos ver el coste computacional de esta función es bastante bajo, el crecimiento es polinómico, lo cual la hace perfecta nuestro problema, ya que la cantidad de texto a procesar es muy variable y puede crecer incluso a documentos de cientos de páginas, era necesario que este procedimiento no consumiese demasiado tiempo para un funcionamiento eficaz del sistema.

Una vez separadas las distintas palabras del texto podemos elaborar los **tokens**. Los tokens son las distintas frases clave que conforman el texto, se encuentran separados por palabras clave y siguen una estructura. De tal manera que el proceso de **tokenización** consiste en realizar la visión de un texto en diferentes partes, atendiendo a un significado semántico y una importancia de cada una de ellas. Por ejemplo en nuestro caso realizamos una separación de los títulos, secciones, capítulos, artículos e incluso de los distintos puntos que contienen un artículo, de la manera que sea lo más exhaustiva posible para abarcar la mayor cantidad de documentos de ley posibles, además de ser flexible a los tokens en ellos. Véase el apéndice 7.1 para más información sobre el funcionamiento de este algoritmo

4.1.3 Elaboración de la estructura

Para el proceso de elaboración de la estructura, en primer lugar hay que determinar cuál es la jerarquía de los tokens. Con este fin, se ha utilizado el Proyecto de ley 9L/PL-0003 Del suelo de Canarias correspondiente al BOC número 291 del año 2016 de la IX legislatura. Analizando dicho documento podemos ver la siguiente estructura:

1. Un título contiene capítulos.
2. Un capítulo puede contener secciones o artículos.
3. Una sección contiene artículos.
4. Un artículo contiene distintos puntos.

El documento posee una estructura en árbol que no es fija, por condiciones como que un capítulo puede contener secciones y/o artículos. Si nos fijamos en otros

documentos como el Proyecto de ley 9L/PL-0017 De Calidad Agroalimentaria correspondiente al BOC número 353 del año 2018 de la IX legislatura, podemos observar una estructura diferente, aunque se mantiene la jerarquía:

1. Un título contiene capítulos.
2. Un título contiene artículos.
3. Un capítulo contiene secciones.
4. Un capítulo contiene artículos.
5. Un artículo contiene puntos.

Para estas y las estructuras diferentes que se puedan plantear y que no contemplemos, pero que mantengan esta jerarquía en los tokens, debemos plantear una estructura de datos en forma de árbol, que fuese flexible y que a su vez respete siempre la jerarquía.

Con el fin de desarrollar la estructura con forma de árbol en python. Podemos utilizar la librería **anytree**.

anytree

anytree es una librería para Python que nos permite implementar una estructura en árbol de manera bastante intuitiva, dispone de una clase Node predeterminada que dispone de un atributo id, children y parent. Además de esta facilidad para crear nuevos nodos, esta librería ofrece una gran variedad de métodos para visualizar o exportar dicho árbol. Esta visualización nos sirvió en el proceso de elaboración de la estructura para ver fácilmente el resultado obtenido pero no sería la visualización que usaría posteriormente, ya que no es dinámica ni usable para realizar grandes consultas. Con respecto a la capacidad de exportar dispone de un método para hacerlo al formato **json**, este formato nos sería de gran utilidad para realizar posteriormente la **api**, a la hora de enviar la estructura del árbol.

¿Cómo formar la estructura del árbol? Primeramente vamos explicar la estructura que se ha elegido para cada Nodo para posteriormente hablar del algoritmo de inserción que utilizaremos para así respetar la jerarquía.

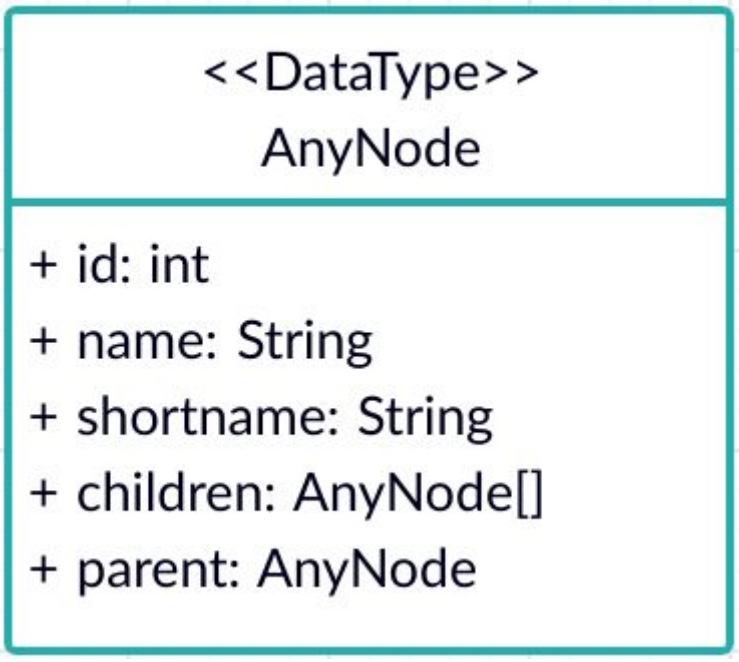


Figura 7: Tipo de dato AnyNode

id: atributo identificador de cada nodo, todos los nodos tienen un valor diferente.

name: atributo que contiene el texto completo de un token.

shortname: atributo que contiene el nombre de lost token.

children: atributo opcional, señala a los hijos de un nodo en caso de que los tuviera.

parent: atributo opcional, señala al padre de un nodo, este atributo es opcional, solo se encuentra nulo en un nodo raíz.

Ahora que tenemos la estructura básica del almacenamiento de información, es el momento de enseñar el algoritmo utilizado para la unión de ellos en forma de árbol.

create_tokens_tree

Este algoritmo, se encarga de comprobar qué tipo de nodo es cada uno de los tokens obtenidos en el paso de tokenización, cuando encontramos un título, a este le asignamos como padre el nodo raíz, ya que este es el nodo de más alto nivel. Si el token tiene la estructura de un capítulo, este solo puede descender de un título, con lo cual su valor parent es el último título que tengamos guardado. En el caso de dar con una sección, en primer lugar comprobamos si el valor del último capítulo es nulo, si no lo es este pasa a ser su padre, en el caso de que sea nulo, pasamos a comprobar el último título y si este es nulo, le asignamos como nodo padre la raíz, se han encontrado documentos cuya raíz tiene secciones. Si damos con un artículo, comprobamos los valores almacenados en los tokens de mayor nivel en orden ascendente(sección, capítulo, título) y le asignamos como padre el primero que encuentre cuyo valor no sea nulo. En cualquier otro caso se le asigna como padre el último artículo que se haya guardado. Esto es para guardar los distintos puntos que puede tener un artículo. Para que esto funcione tenemos que entender que tenemos una variable para guardar el último de los tokens de cada tipo que hemos guardado, y que cuando asignamos un token importante, como podría ser un título, el resto de las variables de memoria de nivel inferior las fijamos a nulo.

Este algoritmo nos permite crear árboles con una estructura **flexible**, la cual se puede adaptar a la de documentos con distinta estructura que los que hemos probado como modelo, de manera que esto funciona con la mayor variedad de leyes posible. Para ver más detalles sobre la implementación de este algoritmo consultar el apéndice de algoritmos 7.2

4.2 Visualización

En el proceso de visualización, primeramente establecimos el tipo de visualización que queríamos conseguir. Esta debía de ser dinámica, dar la posibilidad al usuario de interactuar con ella, además adecuada al formato de una ley. Para ello determinamos que había que realizar una aplicación web, capaz de representar la estructura que creamos previamente.

4.2.1 Tecnologías

Para llevar a cabo la aplicación web debíamos decidir en qué tecnologías iba a

realizarse, para ello creamos una pequeña demo en los principales framework de aplicaciones web tanto en el lado del cliente como del servidor.

Django y Boostrap

Como el paso de procesamiento de texto fue realizado completamente en Python, era bastante jugoso continuar el desarrollo utilizando un framework de este. Django es un framework de Python de alto nivel, enfocado en conseguir el desarrollo rápido, limpio y con un diseño pragmático de una web. Como se ha dicho anteriormente, Django hace mucho énfasis en conseguir desarrollar la página web tan rápido como se pueda, además nos ayuda a solucionar muchos errores de seguridad bastante comunes y nos permite hacer aplicaciones muy escalables. Para el frontend podemos utilizar bootstrap, bootstrap es la librería más popular para realizar diseños responsivos, rápidamente, con gran cantidad de componentes pre realizados.

Javascript con React y MaterialUI

Javascript es el segundo lenguaje de programación más usado a la hora realizar programación web, es muy completo pues nos permite programar tanto el lado del cliente, como del servidor. Es el único lenguaje apoyado por todos los principales navegadores web. **React**, por su parte es una librería de Javascript para construir interfaces de usuario, está basado en componentes que manejan su propio estado y permiten crear interfaces complejas. Es una de las librerías de frontend más usada hoy en día y debido a su gran popularidad me atrae utilizarla. **MaterialUI** sería el análogo a boostrap en la otra solución, es un framework que nos ofrece componentes de React para un desarrollo web más rápido y sencillo. Tiene gran cantidad de componentes adaptados a los estándares de diseño Material, con un diseño muy moderno. Algo menos popular que boostrap, pero con la ventaja de estar enfocado precisamente a React.

Finalmente, el desarrollo fue realizado con Javascript, React y MaterialUI, dado los conocimientos previos en Javascript y Material UI. Además nos permite implementar el patrón Flux para el manejo de datos el cual se caracteriza por dar una gran escalabilidad a la aplicación.

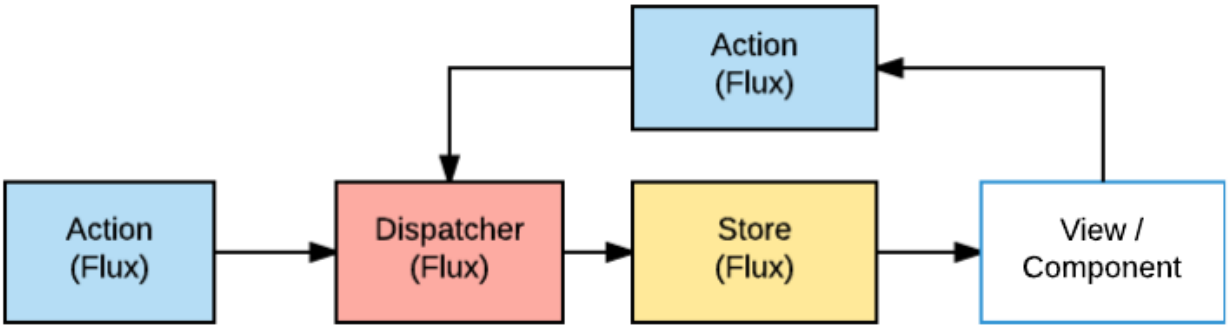


Figura 8: Patrón flux (5)

En el lado del servidor es algo más complejo, puesto que por decisión de la arquitectura, nuestra aplicación debía tener dos servidores. En primer lugar un servidor en **NodeJS**, el cual sería el que nos permite manejar los ficheros y las funcionalidades básicas de la aplicación, y en segundo lugar un servidor en Python con **Flask**, el cuál se encargaría de calcular, realizar y devolvernos la estructura de datos realizada en el

apartado de procesamiento de texto, de forma dinámica para los distintos ficheros seleccionados.

Node.js

"Nodejs es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor, basado en el lenguaje de programación Javascript, asíncrono, con E/S de datos en una arquitectura orientada a eventos. Fue creado con el enfoque ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web"(6). Este suele acompañar habitualmente a aplicaciones web realizadas en Javascript con React.

Flask

"Es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código"(7). Se adapta completamente a nuestra necesidad de ejecutar el código Python escrito anteriormente, además de no sumar demasiada complejidad al problema en sí.

Para la comunicación entre el lado del cliente y los distintos servidores. Primeramente tenía claro que quería utilizar **Express** para el servidor con Node.js

Express

"Es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona una delgada capa de características de aplicación web básicas, que no ocultan las de Node.js que tanto aman los usuarios"(8). Nos permite escribir un código mucho más sencillo en nuestro servidor web como podemos ver:

Servidor Node.js sin express	Servidor Node.js con express
``var http = require('http'); function handleRequest(request, response){ if(request.url == '/home') response.end('This is home page'); if(request.url == '/about') response.end('This is about page'); else response.end('index'); } const server = http.createServer(handleRequest).listen(5000); console.log('Server Started at port', 5000);```	```var express = require('express'); var app = express(); app.set('view engine', 'ejs'); app.get('/', function(request, response) { res.render('home'); }); app.listen(5000);```

Tabla 5: comparación express

Figura 9: código no express **(9) Figura 10**: código express **(9)**

Por parte de Flask ya nos proporciona una estructura similar a la de Express para Python.

Cuando hablamos de la parte del cliente me he decantado por **Axios** para realizar las peticiones.

Axios

"Axios es una librería de Javascript que nos permite hacer de forma sencilla solicitudes HTTP desde el lado del cliente." (11) "Está basada en promesas, por lo que al usarla podremos generar un código asíncrono bastante ordenado." (12)

4.2.2 Arquitectura

El sistema presentado sigue una arquitectura **cliente-servidor**. Esta arquitectura se caracteriza por poder lanzar múltiples clientes, todos ellos capaces de consultar el mismo servidor. Esto dota al sistema de gran escalabilidad de cara a poder hacer un despliegue online del mismo, la parte del cliente se ejecuta en el lado del navegador del usuario que ejecuta el aplicativo, mientras que el servidor se puede desplegar en cualquier parte y es capaz de ser consultado en cualquier momento.

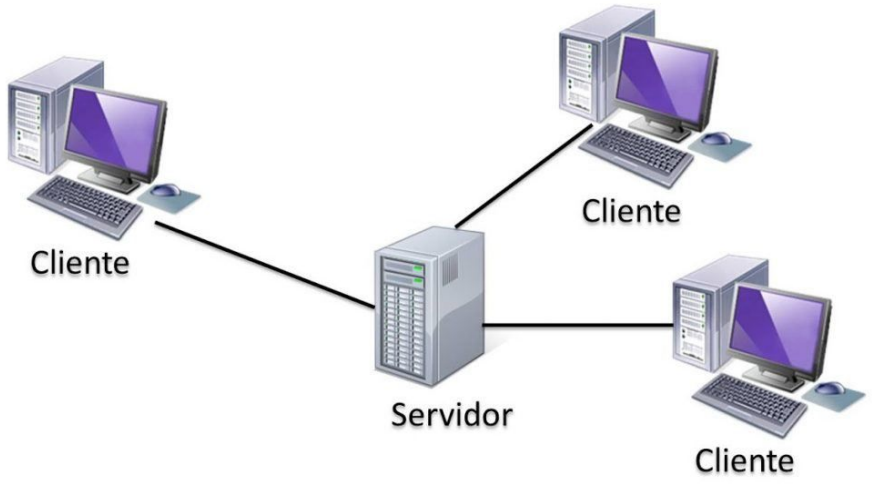


Figura 11: diagrama cliente servidor **(10)**

Otra ventaja de un sistema como este es la capacidad de poder utilizar distintos servidores desde el lado del cliente. En nuestro sistema en cuestión, disponemos de un servidor que es una **api** rest realizada en Nodejs y Express, donde se realiza todo el manejo de los documentos, tanto subida de ellos como la obtención desde el directorio; además de otra api desarrollada en python con la librería Flask, donde se realiza toda la carga y el procesado de los documentos para obtener la estructura de datos de los documentos.

4.2.3 Funcionalidades

Al comienzo del proyecto habíamos decidido cuáles serían las características que debía tener el mismo. Ahora tenemos que decidir las que tendría la visualización de la ley, los métodos que vamos a utilizar para ello. Además de las funcionalidades adicionales

que tendría que disponer la aplicación para añadir archivos nuevos de forma que sea más flexible.

Tarjetas

Viendo los diseños utilizados en las soluciones anteriores, podemos ver que ambos proponen una representación en árbol de la ley, dichos árboles no se adaptan a estilos de diseños actuales. Actualmente los estándares de diseño hablan de Material Design, el cual es un diseño que utiliza principalmente tarjetas para representar la información. ¿Cómo podemos unir una estructura en árbol con un diseño basado en tarjetas? Podemos utilizar tarjetas expandibles, en las que al seleccionarlas, se mostrarían dentro los distintos hijos de dicha parte. Veamos cómo funciona:

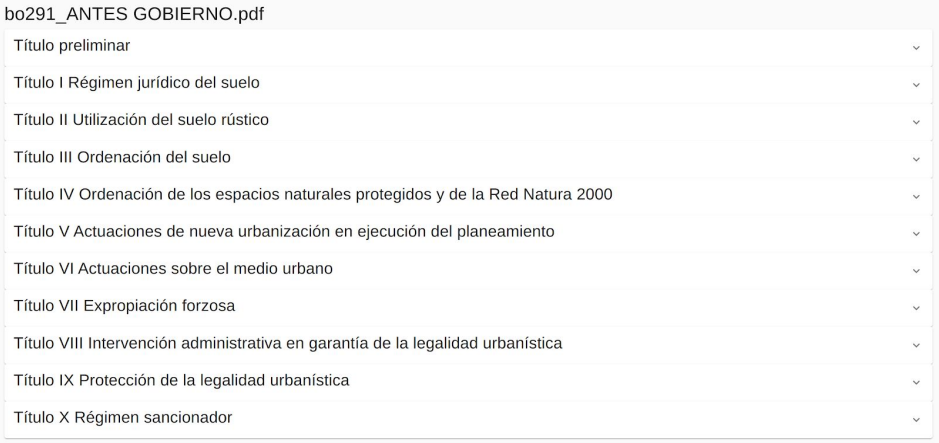


Figura 12: Vista índice tarjetas

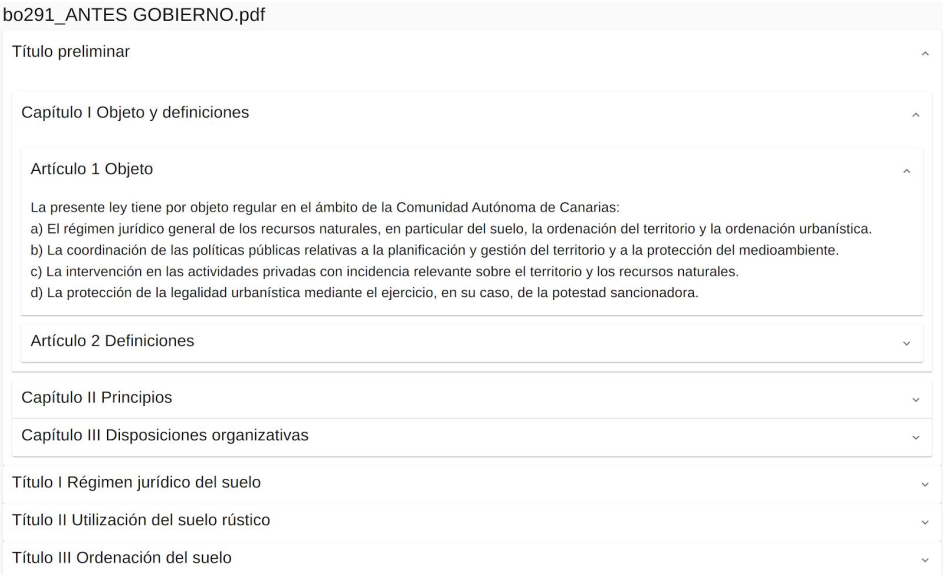


Figura 13: Vista índice tarjetas abiertas

Esta visualización tiene como características el poder tener abiertos exclusivamente aquellos artículos con los que estés trabajando, minimizando los saltos al usuario de la aplicación a la hora de manejarse entre ellos y de este modo reducir el scrolling vertical.

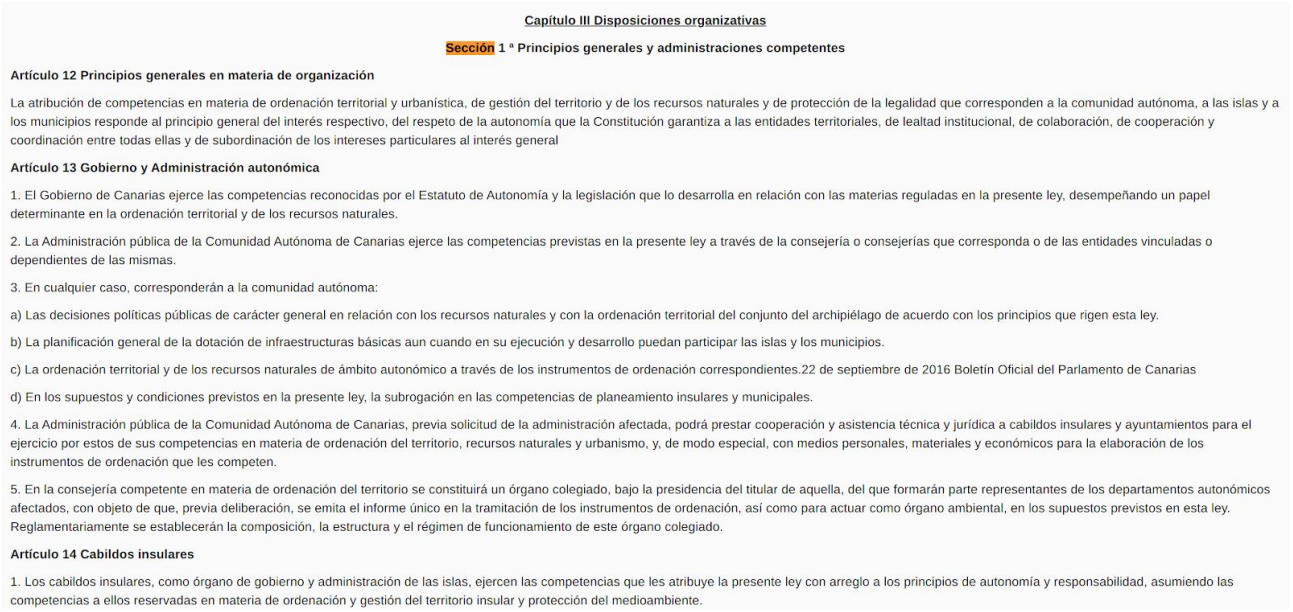
Para realizar esto se utilizan componentes prefabricados de la librería **MaterialUI**, concretamente: **<ExpansionPanel>** y sus subcaracterísticas para crear tanto el enunciado como el contenido.

```
<ExpansionPanel>
  <ExpansionPanelSummary
expandIcon={<ExpandMoreIcon />}
aria-controls="panel1a-content"
id="panel1a-header">
  <h3>{item.shortname}</h3>
  </ExpansionPanelSummary>
  <ExpansionPanelDetails>
    <Grid container>
        <Grid item xs={12}>
            <Grid container>
              <p>{item.name}</p>
            </Grid>
        </Grid>
        <Grid item xs={12}>
            <Grid container>
              {Array.isArray(item.children) ? <LawTreeCardNodeChildrenComponent children={item.children}/> : null}
            </Grid>
        </Grid>
    </Grid>
  </ExpansionPanelDetails>
</ExpansionPanel>
```

Figura 14: Código ExpansionPanel

Texto plano

Originalmente este método de visualización no se encontraba en el repertorio de maneras planteadas. Después de incluirlo, consiguió el nivel de importancia de la visualización por tarjetas. Este método nos permite visualizar el texto de un documento de ley sin necesidad de interactuar con distintas tarjetas, surgió como una alternativa para aquellos usuarios que prefieren leer distintas partes o la ley completa de seguido, de la otra manera hubiera sido muy cansado al necesitar expandir todas las tarjetas. Además de poder usar los mismo métodos de consulta de la vista por tarjetas, este también nos permite usar de forma completamente funcional el `control+f` del ordenador.



Capítulo III Disposiciones organizativas

Sección 1 ª Principios generales y administraciones competentes

Artículo 12 Principios generales en materia de organización

La atribución de competencias en materia de ordenación territorial y urbanística, de gestión del territorio y de los recursos naturales y de protección de la legalidad que corresponden a la comunidad autónoma, a las islas y a los municipios responde al principio general del interés respectivo, del respeto de la autonomía que la Constitución garantiza a las entidades territoriales, de lealtad institucional, de colaboración, de cooperación y coordinación entre todas ellas y de subordinación de los intereses particulares al interés general

Artículo 13 Gobierno y Administración autonómica

1. El Gobierno de Canarias ejerce las competencias reconocidas por el Estatuto de Autonomía y la legislación que lo desarrolla en relación con las materias reguladas en la presente ley, desempeñando un papel determinante en la ordenación territorial y de los recursos naturales.

2. La Administración pública de la Comunidad Autónoma de Canarias ejerce las competencias previstas en la presente ley a través de la consejería o consejerías que corresponda o de las entidades vinculadas o dependientes de las mismas.

3. En cualquier caso, corresponderán a la comunidad autónoma:

a) Las decisiones políticas públicas de carácter general en relación con los recursos naturales y con la ordenación territorial del conjunto del archipiélago de acuerdo con los principios que rigen esta ley.

b) La planificación general de la dotación de infraestructuras básicas aun cuando en su ejecución y desarrollo puedan participar las islas y los municipios.

c) La ordenación territorial y de los recursos naturales de ámbito autonómico a través de los instrumentos de ordenación correspondientes.22 de septiembre de 2016 Boletín Oficial del Parlamento de Canarias

d) En los supuestos y condiciones previstos en la presente ley, la subrogación en las competencias de planeamiento insulares y municipales.

4. La Administración pública de la Comunidad Autónoma de Canarias, previa solicitud de la administración afectada, podrá prestar cooperación y asistencia técnica y jurídica a cabildos insulares y ayuntamientos para el ejercicio por estos de sus competencias en materia de ordenación del territorio, recursos naturales y urbanismo, y, de modo especial, con medios personales, materiales y económicos para la elaboración de los instrumentos de ordenación que les competen.

5. En la consejería competente en materia de ordenación del territorio se constituirá un órgano colegiado, bajo la presidencia del titular de aquella, del que formarán parte representantes de los departamentos autonómicos afectados, con objeto de que, previa deliberación, se emita el informe único en la tramitación de los instrumentos de ordenación, así como para actuar como órgano ambiental, en los supuestos previstos en esta ley. Reglamentariamente se establecerán la composición, la estructura y el régimen de funcionamiento de este órgano colegiado.

Artículo 14 Cabildos insulares

1. Los cabildos insulares, como órgano de gobierno y administración de las islas, ejercen las competencias que les atribuye la presente ley con arreglo a los principios de autonomía y responsabilidad, asumiendo las competencias a ellos reservadas en materia de ordenación y gestión del territorio insular y protección del medioambiente.

Figura 15: Vista texto

Para los distintos estilos en las fuentes se ha aplicado la característica de **MaterialUI** de **<Typography>**, además de un hook de React para aplicar estilos, utilizando Expresiones Regulares de Javascript para decidir qué estilo aplicar a cada token.

```
export default function LawTreeTextNodeChildrenComponent(data) {
  const classes = useStyles();
  data = data.children
  return (
    <Grid container>
      {data.map((item) =>
      Array.isArray(item.children) ?
        <div>
          {item.shortname ?
          <div>
          <Typography className={switchStyle(item.shortname, classes)}>{item.shortname}</Typography>
          <Typography className={switchStyle(item.name, classes)}>{item.name}</Typography>
          </div>
          : <Typography className={switchStyle(item.name, classes)}>{item.name}</Typography>}
            <Grid container justify="center" spacing={2}>
              <Grid item xs={12}>
                <LawTreeTextNodeChildrenComponent children={item.children}/>
              </Grid>
            </Grid>
        </div>
        : <div>
          {item.shortname ?
          <div>
            <Typography className={switchStyle(item.shortname, classes)}>{item.shortname}</Typography>
            <Typography  className={switchStyle(item.name, classes)}>{item.name}</Typography>
          </div>
          : <Typography className={switchStyle(item.name, classes)}>{item.name}</Typography>}
        </div>
      )}
  </Grid>
);
}
```

Figura 16: Componente texto

Wordclouds

La visualización en wordclouds se ha puesto de moda hoy en día ya que permite obtener la información principal de un texto de forma rápida y sin necesidad de leerlo. Es muy interesante de cara a la visualización de leyes, ya que podríamos conocer la temática de una ley, que usualmente tiene una gran cantidad de páginas, en un vistazo. Con esto se ahorraría mucho tiempo a la hora de tratar con ella, además podemos utilizar estas palabras o expresiones principales como referencia para realizar búsquedas dentro del texto. Aquí podemos ver alguna wordcloud:

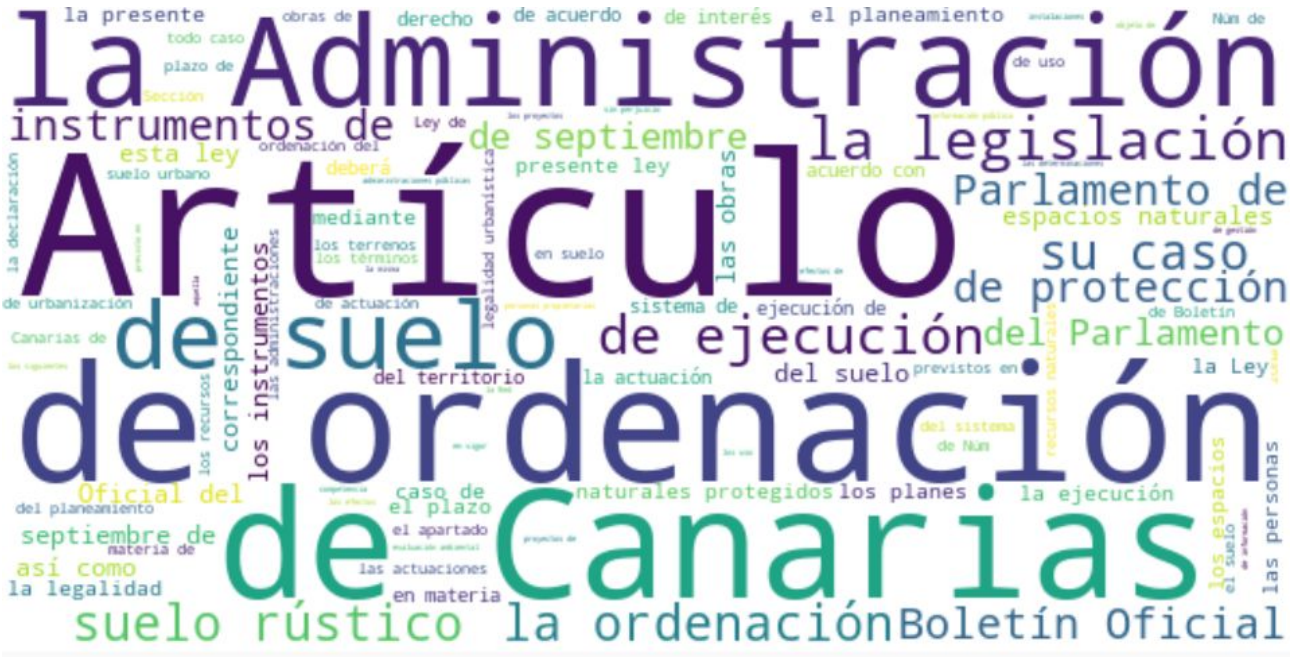


Figura 17: Wordcloud

Esta está generada con una **librería** de Python cuyo nombre es **Wordcloud**, capaz de generarlas dinámicamente a partir de un texto cualquiera, además nos permite generar wordclouds de bigramas, lo cuales son las palabras que más suelen aparecer juntas, que creo que nos permiten sacar mejor el contexto. Estos los obtenemos a partir de la **api** en python y flask.

```
def print_wordcloud(text, document):
    stopwords_spanish = stopwords.words('spanish')
    stopwords_spanish.extend(['Artículo', 'Boletín', 'Oficial', 'Parlamento', 'Canarias'])
    # Here I can add all the worlds I dont like as stopwords
    wc = WordCloud(background_color="white", max_words=100,
                   stopwords=stopwords_spanish, contour_width=3, contour_color='steelblue',
                   width=600, height=300)
    # generate word cloud
    wc.generate(text)
    # store to file
    wc.to_file("outputs/" + document + ".png")
    return document + ".png"
```

Figura 18: Generación wordcloud

Subida de archivos

La subida de archivos es una característica añadida de cara a poder elegir los distintos documentos que queremos visualizar de forma dinámica, ya que la aplicación no dispone de los documentos que se encuentran en la web del parlamento. De este modo el usuario puede subir nuevos archivos y seleccionar cuál de ellos quiere ver. Dispone de una visualización en pdf, una vez subido para poder comprobar que es el archivo correcto. No permite subir duplicados. Para el desarrollo de esto creamos una api en **nodejs** ya que era el backend original, además de la facilidad que nos da dicho framework para el manejo de los documentos y de cara a una posible escalabilidad futura. A la pantalla principal le acompaña una lista con los documentos que hay subidos previamente.

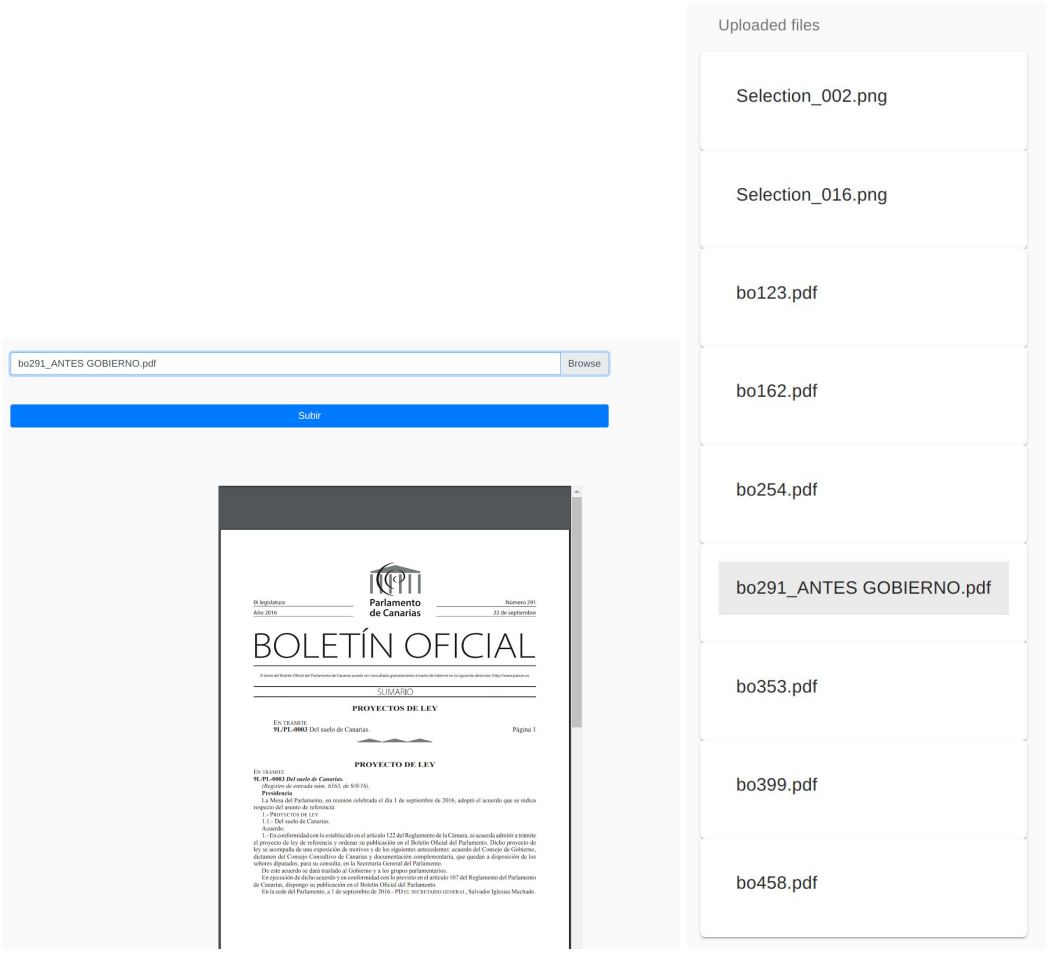


Figura 19: Subida y listado de archivos

Como en las visualizaciones anteriores, la mayoría de componentes son de Material UI: **<List>**, **<Card>**. Pero esta vez añadí algo de **bootstrap** para el campo de subida de archivos ya que **MaterialUI** carecería de este.

4.3 Consultas

Según Oxford Languages * “una consulta es la búsqueda de información en una

fuente de documentación para aprender una cosa o para aclarar una duda". Si nos ceñimos a lo que dice la Rae "una consulta es una acción y efecto de consultar, véase consultar." (13) . Consultar: "Buscar documentación o datos sobre algún asunto o materia." (14)

El objetivo principal de este proyecto era mejorar la consulta de la información sobre el texto de la ley, es decir tenemos que mejorar la obtención de la información a la hora de buscar dentro de ella, esto puede incluir la claridad de la información que se obtiene, la cantidad de información o la especificidad de la misma, a la par de la visualización de la información consultada. Para ello en primer lugar creo que tendríamos que añadir nuevos métodos de consulta para dar al usuario la posibilidad de elegir el modo más adecuado para obtener la información en cada momento. Además, cuando se realiza esta consulta debe de mostrarse la información de manera intuitiva, tal y como esperaría el propio usuario.

Todas estas consultas se realizan en el lado del servidor(la api Python), puesto que quería que la mayor parte del trabajo sobre expresiones regulares se realizase con la librería re y poder trabajarlas en el mismo lenguaje.

4.3.1 Consultas sobre tokens

La consulta sobre los tokens del árbol, consiste en realizar preguntas del tipo:

"Del título 1, capítulo 2, quiero la sección 1"

Son consultas en las que no se busca texto plano tal cual o sobre el contenido de un artículo, sino más bien para buscar sobre la estructura de la ley, Nos permiten localizar distintas partes sobre las que estamos interesados utilizando algo de procesamiento de lenguaje natural para obtener las palabras clave con la información que las acompaña. Estas reciben un preprocesamiento de texto para obtener las palabras clave, además de estandarizar el formato de los números, ya que según el tipo de token este se encuentra en números romanos o números latinos.

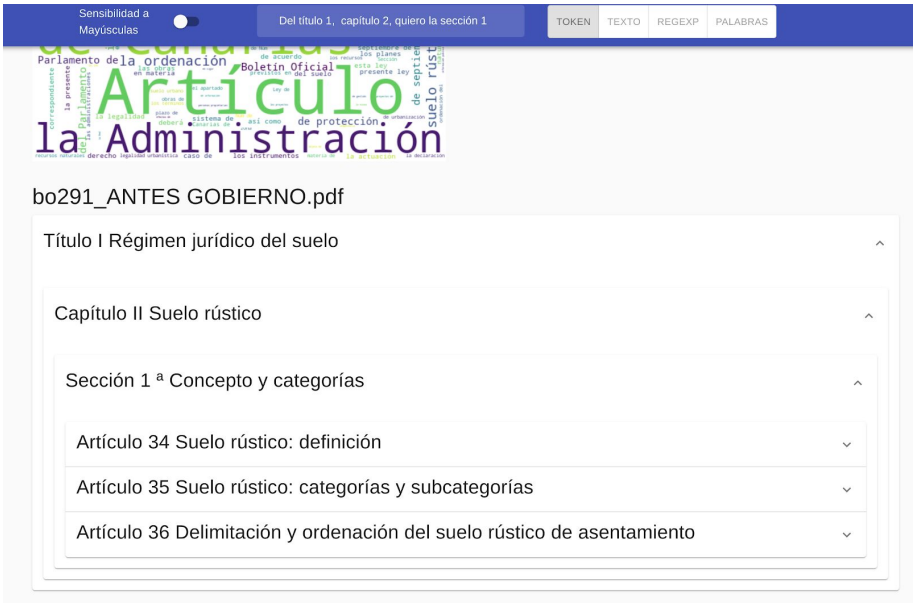


Figura 20: Consulta sobre tokens

4.3.2 Consultas sobre expresiones

Las consultas sobre expresiones se consideran las más interesantes de cara a obtener la información de la forma más optimizada posible, veamos los distintos métodos que tenemos.

A. Frases

Este tipo de consultas sea realiza cuando se quieren buscar expresiones, frases, en general un conjunto de palabras que aparecen juntos de forma ordenada. ¿Qué diferencia esto de una búsqueda de texto normal? En este caso aplicaríamos un filtro sobre el árbol mostrando únicamente los tokens que tienen alguna coincidencia sobre el texto introducido. Al introducir la expresión `la ordenación del territorio` me mostraría todo el camino hacia cualquiera de los artículos que contiene dicha expresión

Figura 21: Consulta texto

Como podemos ver esta expresión se encuentra en el artículo 14, en el resultado de la consulta vemos que el camino hasta este sería Título preliminar > Capítulo III > Sección 1 > Artículo 14.

B. Expresiones regulares

Teniendo en cuenta que gran parte de este proyecto está basado en Expresiones Regulares, creo que era obligatorio añadir la posibilidad de buscar usándolas, aunque no es algo común en el ámbito del derecho, es algo que se podría aprender a usar rápidamente y podría darle mucho potencial al uso de la aplicación si se hace correctamente. Por ejemplo aquí hemos realizado la búsqueda de todas las partes que contengan las expresiones `recursos degradados` o `recursos naturales`.

Figura 22: Consulta regexp

C. Conjuntos de Palabras

Una funcionalidad que ya se encuentra disponible en el buscador de algunos navegadores es la posibilidad de introducir una secuencia de palabras y mostrarte cualquier parte que contenga una de ellas. Es clave para no perder funcionalidades con respecto a las que ya existen, sobre todo teniendo en cuenta quien la use actualmente, su funcionamiento es el esperado viendo las filtros anteriores. Veamos por ejemplo cómo sería mostrar todos los artículos que contienen la palabra público o la palabra privado

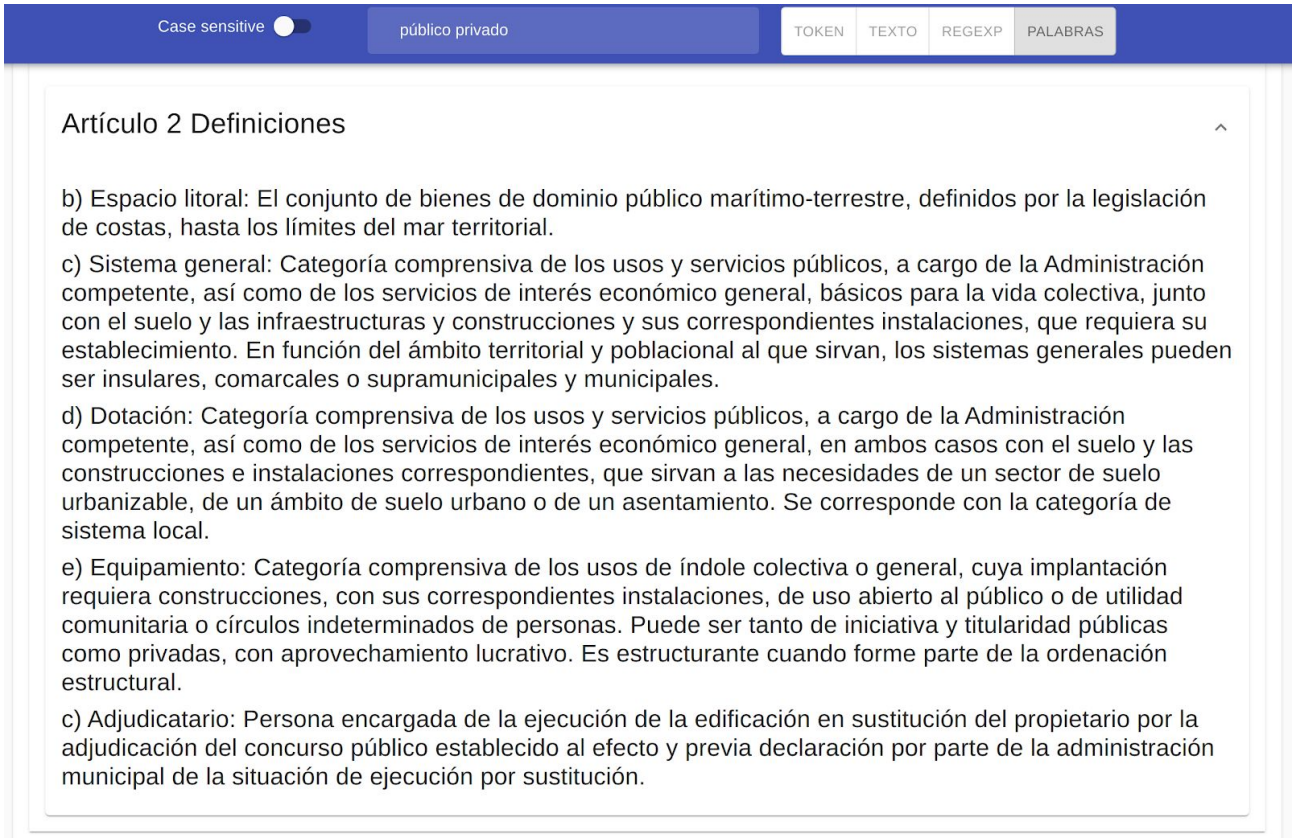


Figura 23: Consulta palabras

D. Sensibilidad a las mayúsculas

Este apartado no es un tipo de consulta nuevo específico, simplemente nos añade la oportunidad de realizar todas las consultas anteriores teniendo en cuenta las mayúsculas, es una buena opción cuando no sabemos exáctamente si una palabra se escribe con mayúsculas o no, también cuando queramos obtener todas las ocurrencias de dicha palabra independientemente de cómo esté escrita.

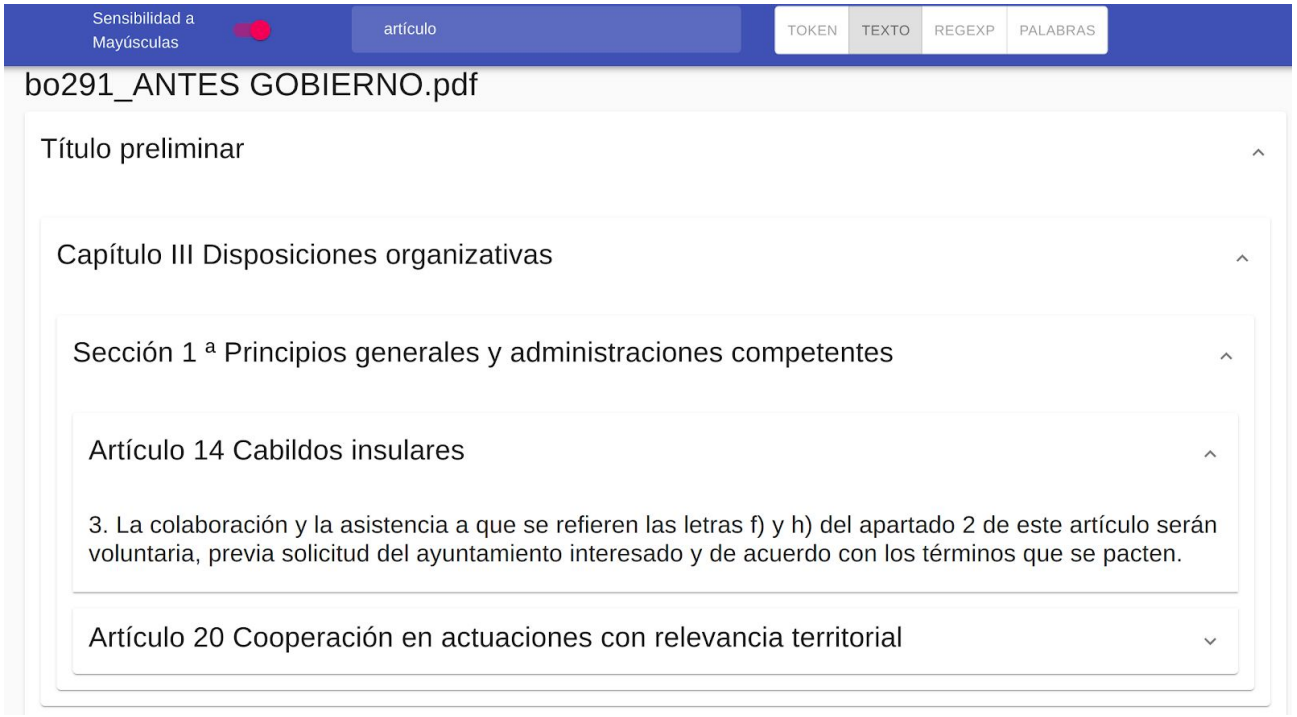


Figura 24: Consulta sensibilidad Mayúsculas

Figura 25: Consulta sensibilidad Mayúsculas no

Como podemos ver en uno de los casos al ser sensible a las mayúsculas solo encontraría aquellos artículos que incluyen la palabra ‘artículo’ tal cual, en el otro caso como todos los Artículos coinciden me muestra todos.

4.4 Evaluación

Durante el proceso de implementación y desarrollo se le pidió consejo a distintos juristas, de cara a decidir distintas variables como el tamaño de la fuente o algunas funcionalidades del sistema, por ejemplo la visualización en texto. También aportaron su opinión con respecto a la búsqueda dentro del sistema, evaluando los distintos filtros y consultas y añadiendo alguno adicional. Todo esto con el objetivo de conseguir un sistema lo más adaptado posible a aquellos potenciales usuarios. Los distintos BOC utilizados para comprobar el correcto funcionamiento de la herramienta son los siguientes:

- Proyecto de ley 9L/PL-0003 Del suelo de Canarias correspondiente al BOC número 291 del año 2016 de la IX legislatura.
- Proyecto de ley 9L/PL-0015 De Ley de Bibliotecas de Canarias. correspondiente al BOC número 87 del año 2019 de la IX legislatura.
- Proyecto de ley 9L/PL-0026 De Sociedades Cooperativas de Canarias. correspondiente al BOC número 1576 del año 2010 de la IX legislatura.
- Proyecto de ley 9L/PL-0017 De Calidad Agroalimentaria correspondiente al BOC número 353 del año 2018 de la IX legislatura.
- Proyecto de ley 9L/PL-0017 De Calidad Agroalimentaria correspondiente al boc número 6971 del año 2018 de la IX legislatura.

El total de usuarios de los que se ha recibido opinión y recomendaciones sobre la interfaz y el desarrollo de la aplicación son 10, todo esto mediante imágenes de la aplicación, mientras que físicamente han podido interactuar con ella 2 de ellos, para evaluar lo intuitivo de la interfaz y el uso de la aplicación. Para evaluar la necesidad de implementación de las diferentes funcionalidades se realizó un cuestionario a un total de 14 juristas, donde podremos ver más adelante los resultados del mismo.

4.4.1 Visualización

En la primera versión desarrollada, la única visualización de la que disponía el sistema, la cual iba a ser la principal y sobre la que iba a girar todo el desarrollo, es la visualización **mediante tarjetas**. La cual a priori recibió muy buenos comentarios, al permitir al usuario visualizar distintas partes sin tener todo el texto delante, cuando este solo trata por ejemplo con 3 artículos, solo tiene que abrir el árbol hasta ellos, disminuyendo muchísimo el scroll vertical y facilitando encontrarlo. Aunque por otra parte, algunos comentaron que si deseas leer mucho texto consecutivo o hacer una lectura general de la ley, este formato era contraproducente, aumenta el scroll y cansa mucho al usuario el tener que interactuar con las tarjetas. Aquí entra en juego el segundo modo de visualización de la aplicación para trabajar con ella, la visualización en texto plano. Durante el desarrollo de esta se ha ido consultando para decidir el tamaño de las fuentes y la localización de las palabras clave. Otra de las claves de esta visualización es que permite realizar el comando `control+f` completamente funcional, a diferencia de muchos ficheros en formato pdf

donde hay partes que no funcionan bien, por caracteres raros o imágenes. Una vez implementada dicha funcionalidad tuvo un feedback bastante positivo de cada al usuario, igualando a la otra en importancia.

4.4.2 Consultas

En relación a la funcionalidad de consultas, los diferentes usuarios llevaron a cabo las búsquedas de información que suelen realizar. La mayoría de ellos consideraron la funcionalidad de búsqueda avanzada con procesamiento de lenguaje natural, algo innecesario y no muy útil. Casi la totalidad se decantó por realizar búsquedas de texto plano o utilizar `control + f` sobre la funcionalidad de texto. Todos coincidieron en que era interesante conocer el árbol del que procedía el resultado, por ejemplo si realizamos una búsqueda y nos aparece el artículo 20, sería necesario saber a qué título, sección, etc. pertenece dicho artículo. De tal manera que se pasó a representar los resultados en vez de como un listado, utilizar la estructura de árbol implementada para la visualización general. Una vez esto esta funcionalidad se complementa muy bien con la búsqueda en la visualización en texto, ya que todos echaron en falta el marcas las coincidencias con algún formato diferente dentro de la vista, tanto en tarjetas como en texto. La fase final del desarrollo pasa por intentar arreglar esta funcionalidad, aunque el desarrollo de este cambio es bastante costoso porque el sistema de consultas se procesa actualmente en el servidor de tal manera que el cliente no tiene acceso a la información de las coincidencias.

4.4.3 Cuestionario

El conjunto de personas que respondieron al cuestionario tienen el siguiente sesgo:

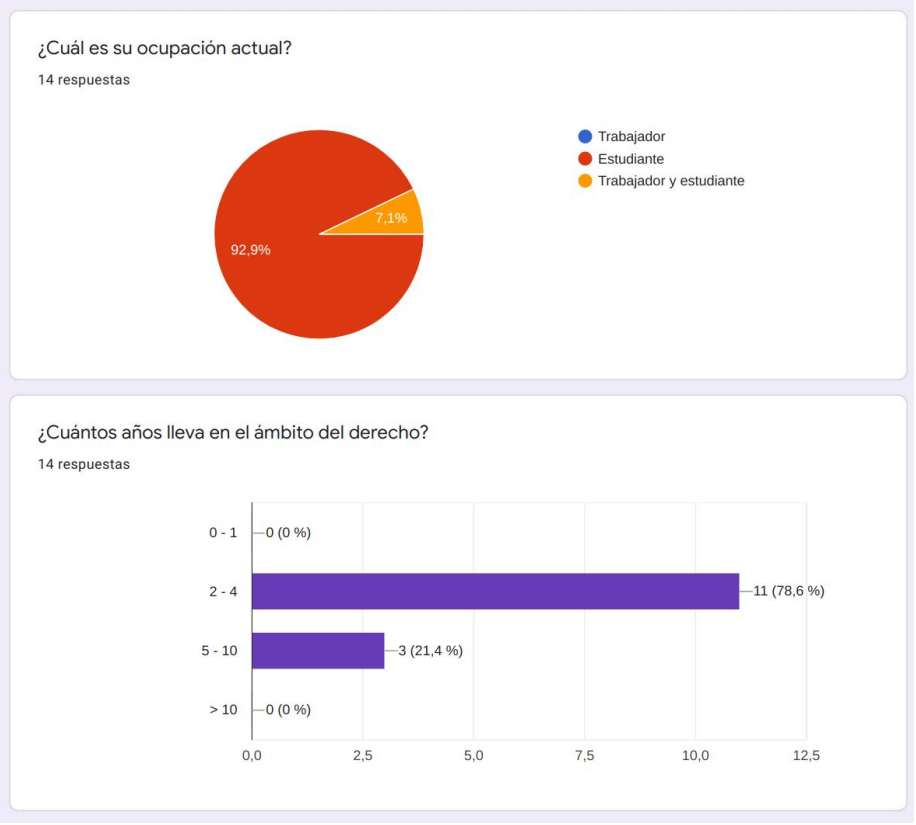


Figura 26: Participantes cuestionario

La mayoría de ellos son estudiantes que llevan en el ámbito del derecho entre dos y cuatro años, uno de ellos también ejerce actualmente. Tres de ellos llevan entre 5 y 10 años, son personas con bastante experiencia. Todos ellos respondieron distintas preguntas sobre el formato PDF, la visualización de las plataformas, búsqueda dentro de los documentos y además sobre otros métodos de visualización.

Formato PDF

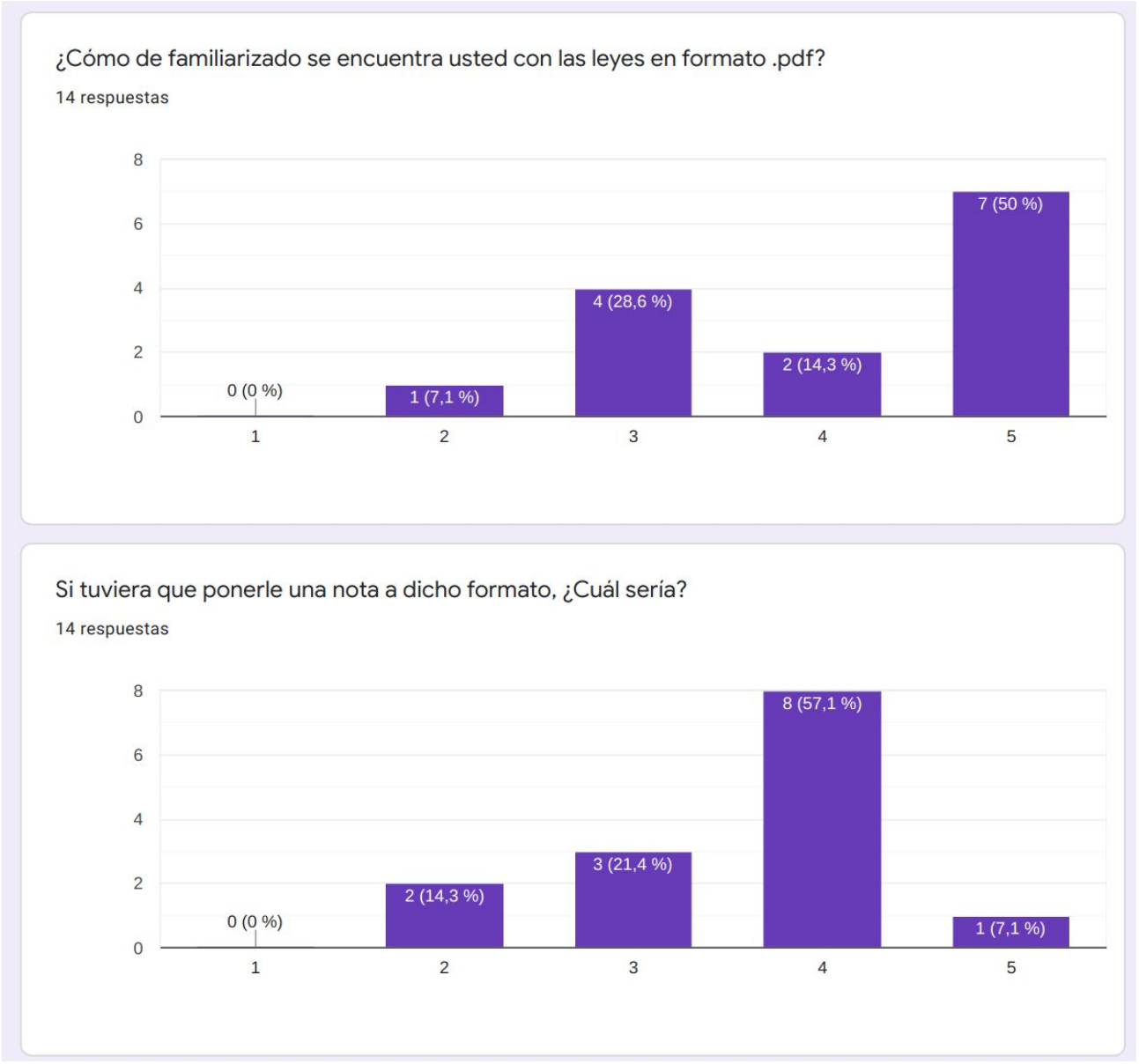


Figura 27: Evaluación PDF

Como podemos ver son personas que por lo general suelen encontrarse bastante familiarizados con este formato. La mayoría de ellos consideran que no es un formato perfecto, todavía hay margen de mejora. Se puede ver como hay un grupo de unas 5 personas que no están muy conformes con este formato, el objetivo es mejorarlo. Esta es

una manera de evaluar la necesidad de creación de este aplicativo.

Plataformas

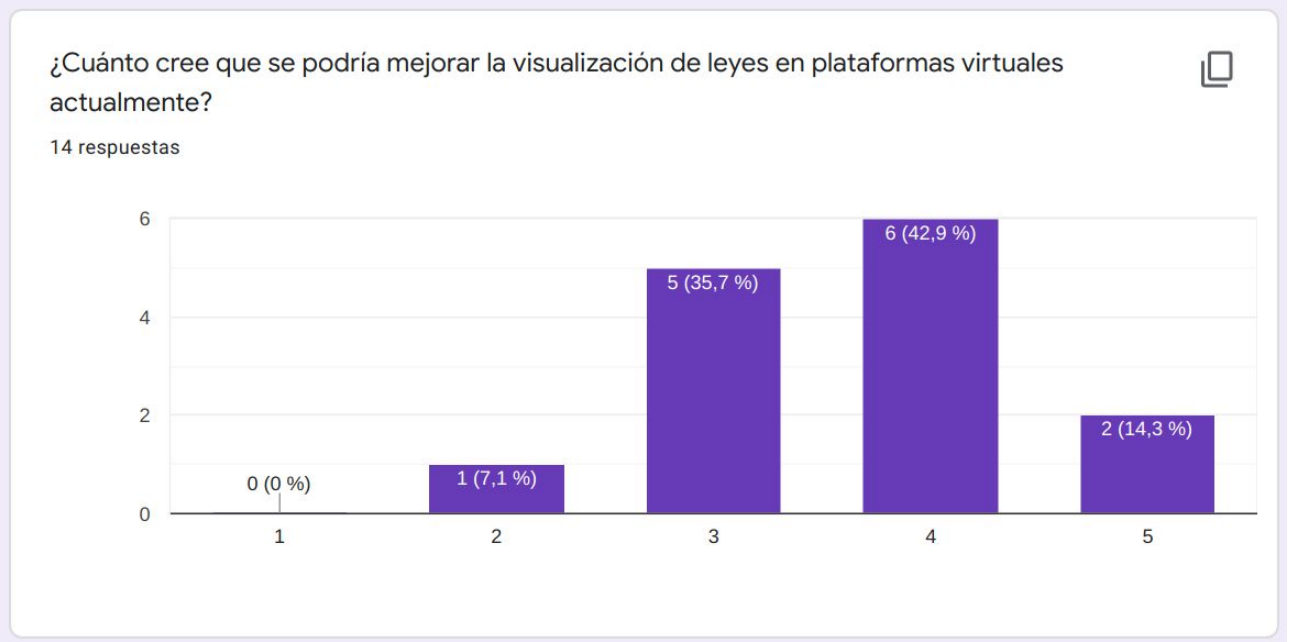


Figura 28: Mejora plataformas virtuales

Con respecto a las plataformas virtuales por lo general, se consideran muy mejorables, se ve que prácticamente la totalidad de los usuarios se encuentran insatisfechos con el funcionamiento de las mismas. Para mejorar esto tendríamos que modernizar las plataformas virtuales que se utilizan hoy en día.

Búsqueda

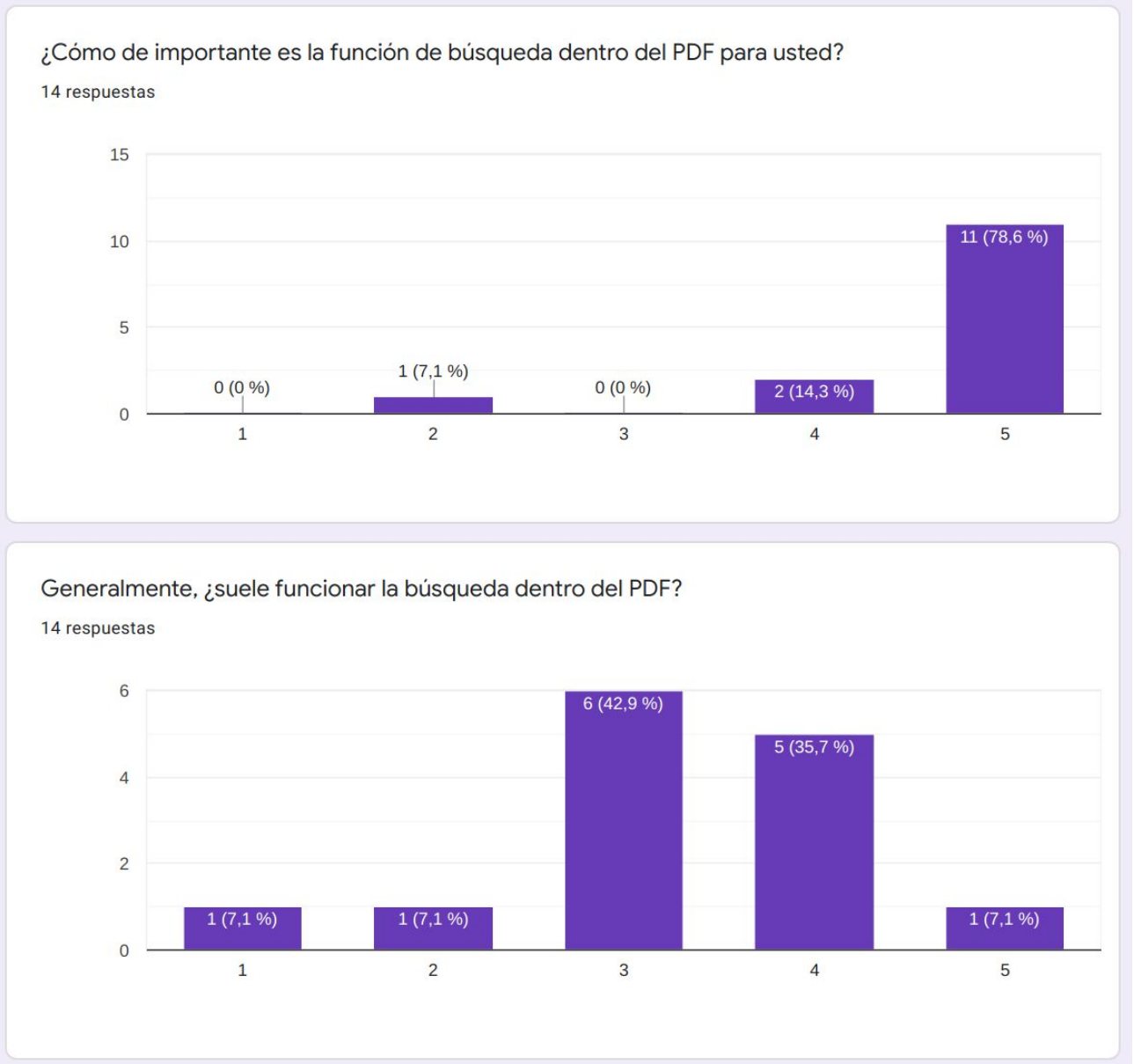


Figura 29: Evaluación búsqueda

Para evaluar la importancia de las búsquedas para los juristas se les formularon las preguntas ¿Cómo de importante es la función de búsqueda dentro del PDF para usted? la cual recibió la máxima puntuación por la mayoría de usuarios, con lo cual podemos decir que es de máxima importancia. Y ¿suele funcionar la búsqueda dentro del PDF? Donde los resultados fueron muy repartidos, aunque teniendo en cuenta que la mayoría de ellos le dieron la máxima importancia, y ahora los resultados indican que no funciona muy bien normalmente, podemos dejar claro que este campo es muy importante mejorarlo. Para ello hemos desarrollado los distintos métodos de consulta dentro de nuestra aplicación.

Métodos de visualización

Figura 30: Modos de visualización

La primera de las preguntas, que sería si el usuario suele leer todo el texto de la ley de seguido, está enfocada a esa funcionalidad de visualización de texto plano la cual se asemeja más a un PDF tradicional, donde aumenta la comodidad para leer grandes fragmentos de texto seguidos, pocos usuarios suelen realizar esta lectura, aunque son una cantidad a tener en cuenta. Por otro lugar, a la pregunta de no importarte dar saltos entre las distintas partes de una ley, los resultados están muy repartidos, el objetivo de esta pregunta sería conocer cómo de útil sería la visualización en tarjetas para ver estas diferentes partes de forma cómoda. Estos resultados indican que mejorar la visualización por partes debe de ser algo con bastante prioridad.

Nuevos métodos de visualización

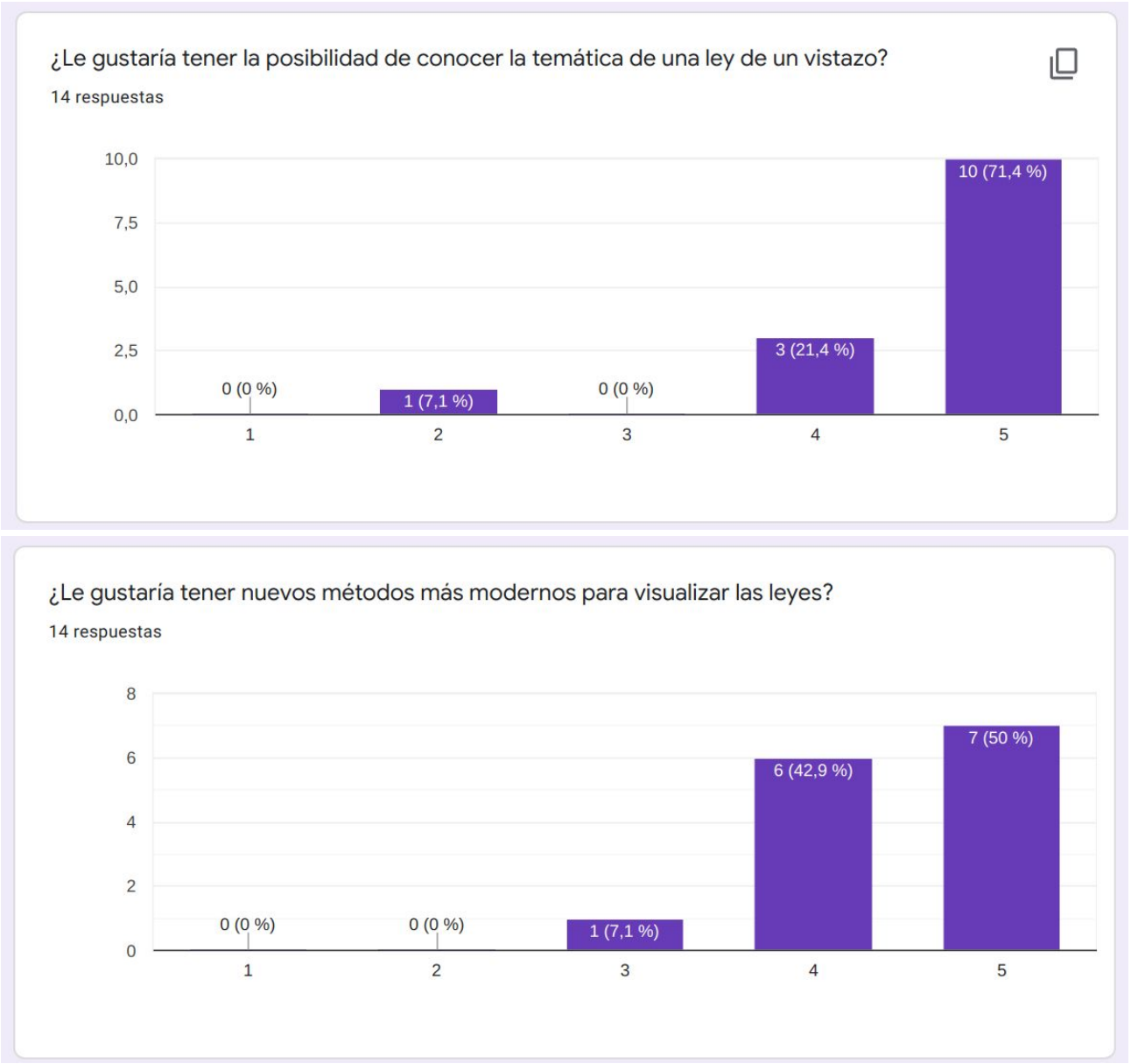


Figura 31: Nuevos métodos de visualización

Para evaluar la importancia de los wordclouds se preguntó a los sujetos sobre la posibilidad de conocer la temática de una ley de un vistazo, lo cual recibió una respuesta casi unánime afirmativa, esto nos indica que esta herramienta nos puede ser bastante útil a la hora de visualizar de forma rápida los documentos de ley y saber de qué tratan. La siguiente pregunta se utilizó para conocer la predisposición de estos usuarios a utilizar estos métodos y tener nuevas maneras de ver estos documentos de ley. La respuesta fue afirmativa, esto nos indica que este tipo de herramientas pueden tener aceptación entre ellos.

Capítulo 5 Conclusiones y líneas futuras

En este Trabajo Fin de Grado se ha desarrollado un aplicativo que permite reconocer la estructura de una ley, permitiendo su visualización de forma jerárquica en un navegador Web junto con la representación visual de los términos que conforman la ley mediante una nube de etiquetas. Asimismo, dicho aplicativo permite la realización de cuatro tipo de consultas diferentes.

La validación del prototipo se ha llevado a cabo por expertos en el ámbito del derecho, las principales conclusiones obtenidas son:

- En relación a la utilización de documentos pdf el principal problema viene a raíz de cómo se generan, con un pequeño cambio en su creación se podrían conseguir unos resultados mucho mejores. Los PDF que se encuentran con leyes generalmente no disponen de los **enlaces a las distintas partes que se nombran** dentro del mismo, sería muy interesante que estos estuviesen presentes, incluso con la posibilidad de añadir los de documentos nombrados. Esta funcionalidad mejoraría en gran medida el manejo del sistema.
- En relación a la gestión de versiones de una ley se podría implementar un **gestor de versiones** tal y como sería el de github o el de Google Drive. Este gestor de versiones nos permite, dentro de un documento, ver las partes que han cambiado en sus distintas versiones. Esta ha sido una funcionalidad muy señalada por la mayoría de los juristas, sería esencial tenerlo en un sistema final.
- En relación a los nuevos métodos de visualización podemos añadir que la visualización a través **wordclouds**, ha recibido un feedback bastante positivo y la posibilidad de incluirlo en los futuros sistemas, además de investigar otras formas similares.
- En relación a las plataformas donde obtenemos los documentos de ley, aunque esta propuesta sale fuera del ámbito concreto de este trabajo, sería una buena línea de trabajo futuro para modernizar el ámbito del derecho, la posibilidad de modernizar las **plataformas** donde se obtiene las diferentes leyes, como por ejemplo la del BOE o la del Parlamento de Canarias. Añadiendo una visualización de la página más moderna, basada en estándares material design, dando la posibilidad de crear usuarios y almacenar las distintas leyes con las que se suele tratar frecuentemente. Además de añadir un visor de documentos propio más avanzado que incluya las características señaladas en el resto de apartados.

Aunque el objetivo de este TFG, no ha sido proporcionar un sistema final para su utilización, la mayoría de los usuarios finales que lo han testeado creen que es una mejoría con respecto al sistema que se tiene actualmente, sobre todo la visión en tarjetas que nos evita dar muchos saltos dentro del documento. Además de que la función de `control + f ` funciona perfectamente sobre el texto, cosa que no podemos asegurar en los todos los ficheros con formato PDF. Algunas deficiencias que tiene y que podrían

mejorarse si se trabajase sobre el mismo es que tal y como ocurre cuando se utiliza la función de buscar en el navegador, cuando se apliquen los distintos filtros las **palabras o expresiones** que se han buscado **aparezcan remarcadas** en algún tipo de subrayado.

El sistema sigue sin obtener una **tokenización perfecta**, debido a los caracteres especiales que se introducen en el método de obtener el texto de los PDF, esto se podría solucionar más adelante si se diese la posibilidad de crear las leyes dentro de la aplicación, aunque para ello tendría que estar ya implantada en un sistema final con identificación y autenticación. Otra alternativa para mejorar esta tokenización sería buscar otra herramienta para obtener dicho texto, tal vez se podría probar en otros lenguajes.

Dentro de esta misma aplicación se podría implementar una **agrupación de los pdf** por metadatos o por datos introducidos en un cuestionario al añadirlo, de manera que podamos ordenarlos por fecha, edición, tipo, etc. o añadirle un nombre descripción, puesto que actualmente al descargarlos de las páginas donde se encuentran, reciben un nombre un poco abstracto para el usuario.

Al igual que con los ficheros en formato PDF, se podrían **guardar los distintos artículos** más utilizados de cada ley, de tal manera que se optimizaría su obtención durante el tiempo que se traten con ellos.

Este tipo de **filtrados de búsqueda** han tenido gran aceptación en los juristas que los han probado, dentro de la plataforma estaría muy bien incluirlos. En la aplicación actual se han implementado en el lado del servidor, de este modo se podrían realizar en Python y para un sistema prototipo esta carga no iba a ser muy grande. De cara a un sistema final el filtrado debería de realizarse en el lado del cliente, para disminuir el número de consultas que se hacen al servidor, la carga de este y además, aumentar mucho la velocidad de la aplicación.

Summary and Conclusions

In this work, it has been developed an application that can recognize the structure of a law, allowing the visualization hierarchically on a web navigator alongside a visualization of the keywords on wordclouds. Furthermore we can query on four different ways.

The validation of the app is been done by experts on the laws environment, here we have the conclusions:

- Relative to using PDF documents, the main problem comes from they way they are being generated, with a small change on the creation we would get much better results. Nowadays in the PDF we can see differents part where another part or document is named and there is no **link** to go there. Working with PDF would be much intuitive and would be much more user friendly.
- Relative to versions managing of a law we could use a **version manager** like it is on github or the Google Drive's one. In this version manager we could see the changes made between different versions of the same document and when they were then done. This is a key feature in a final system, a lot of lawyers has said that.
- Relative to the new visualization methods we can say that the **wordclouds** received a very positive feedback and adding them to future systems would be important,investigate about other similar methods would be fantastic.
- Relative to the **web platforms** where we get the different soft law documents, although this isn't in the ambit of this work, we could modernize them using material design standards and adding new features like the creation of users and storing most used laws. Also, we could add a viewer of specific purpose for the laws. Furthermore the features named in the other parts.

Although the aim of this work hasn't been to provide a final system to be used, most of the final users who has tested it believe that it's better than the actual systems, especially the card view where we can open the parts we are using. In addition the searching function of the navigator works perfect on the text, it doesn't happen always on PDFs. There are some things that could be improved if it's development continues, for example **emphasize the words and expressions** which are searched using the integrated searching bar.

The **tokenization** isn't perfect yet, because of some extrange symbols that we get on the text subtracted from the PDF, we could solve it using another method of extraction maybe in another language or using another library. An alternative would be to change the generation of the documents to work perfect with the actual method.

In the application we could **group the pdf** by metadata or by different characteristics added. In this way ordering by date of creation, date of edition, etc. or adding a name would be very useful. Inside each law we could mark the most important articles allowing the user to get them easily when they are working with them.

Talking about the **query methods**, most of the lawyers who has used them say that they think they are useful to add in future systems. In the current solution they work in the

server side, because they are easier to implement on Python and in this prototype wouldn’t be a big load, but in a final system it would be crucial to make them work in the client side to boost the speed and reduce the load to the server.

Capítulo 6 Presupuesto

Para el desarrollo de este proyecto, no se ha necesitado ningún tipo de material físico, con lo cual el coste de este proyecto se puede resumir en el tiempo de trabajo, del cual se realizará un desglose más adelante. También se añadirá información sobre el coste de un posible despliegue.

6.1 Coste de desarrollo

Para el coste por hora se ha usado el sueldo de un desarrollador medio de 2000€ al mes, sin tener en cuenta los impuestos. El desglose del precio se hará en las distintas tareas realizadas.

Tarea	Cantidad [Horas]	Coste Unitario [€/hora]	Coste total [€]
Análisis	20	12,5	250
procesamiento de texto	60	12,5	750
Aplicación Web	80	12,5	1000
Api	40	12,5	500
Total	200	12,5	2500

Tabla 6: Coste de desarrollo

6.2 Coste de despliegue

Para el coste del despliegue se ha tomado como referencia los costes del servicio de alojamiento web **arsys**.

Servicio	Precio total [€]	Precio mes [€]	Descripción
Hosting	249,99 €	10,42 €	Plan profesional, 2 años
Dominio	23,49 €	0.99 €	Dominio .es, 2 años

Tabla 7: Coste de despliegue

¿Qué variables se han tenido en cuenta para elegir estos servicios?

En primer lugar el plan de hosting que se ofrece incluye un plan con tráfico de web ilimitado, para ello se ha supuesto un uso generalizado de dicha aplicación web.

Se ha decidido un dominio “.es” ya que es el de nuestro país y su precio estándar, sin tener en cuenta posibles ofertas ni descuentos.

Capítulo 7 Algoritmo

7.1 Algoritmo Tokenización

```
    '''
    Tokenization algorithm
    Jose Ramón Casero Fuentes
    20th June 2020
    This algorithm divide a law text on different tokens by their
importance
    Each word in text is processed and added to last token of create a new
token
    utils.py
    '''
    def tokenize(token):
       subtokens = [""]
       start_time = time.time()
       words = nltk.word_tokenize(token)
       num_words = len(words)
       duration = time.time() - start_time
       print("Time transcurred: ", duration)
       print("Num words: ", num_words)
       num_words = len(words)
       count = 0
       ## Never two tokens following
       last_is_token = False
       for word in words:
           word = word.strip()
               ### Is token and, common nearly words using with similar
expresion
            if not last_is_token and (is_titulo(word) or is_capitulo(word)
or is_seccion(word) or is_articulo(word)) and (count < len(words)-2 and
((words[count + 1] != "o") and (words[count + 1] != "habilitante") and
(words[count + 1] != "habilitantes") and (words[count + 1] != "de"))):
               subtokens.append(word)
               last_is_token = True
           ### Token of type: a), b), c)
              elif not last_is_token and len(word) == 1 and count + 1 <
len(words) and count > 0 and words[count + 1] == ')' and word != '.' and
```

```
words[count - 1] != 'letras' and words[count - 1] != ',' and words[count - 1] != 'letra' and words[count - 1] != 'y' and words[count - 1] != 'apartado':
                subtokens.append(word)
                last_is_token = True
            ### Token of type: 1. 2. 3. 11.
              elif not last_is_token and word.isnumeric() and count > 1 and count + 1 < len(words) and (words[count - 1][len(words[count - 1]) - 1] == '.' or words[count + 1] == '.') and words[count - 2] != 'Núm' and words[count - 2] != 'Grupo' and words[count - 1] != 'Natura':
                subtokens.append(word)
                last_is_token = True
            ### Token of type: : - first of enum
                elif not last_is_token and word == '-' and count > 1 and words[count - 1] == ':' and words[count - 1] != '.':
                subtokens.append(word)
                last_is_token = True
            ### Token of type: 1.º 2.º 3.ª 11.ª
              elif not last_is_token and re.match("[0-9]+[.][ªº]", word) and not is_seccion(words[count - 1]):
                subtokens.append(word)
                last_is_token = True
            ### Token of type Grupo 1, Grupo 2, Grupo 3
            elif not last_is_token and word == 'Grupo':
                subtokens.append(word)
                last_is_token = True
            else:
                subtokens[len(subtokens) - 1] += (' ' + word)
                if word != '.':
                    last_is_token = False
            count += 1
        return subtokens
```

7.2 Algoritmo Creación Árbol

```
    '''
    Create tokens tree algorithm
    Jose Ramón Casero Fuentes
    20th June 2020
```

```
    Once tokenized, this algorithm receive the different tokens and create
an hierarchy between them according to their formar
    utils.py
    '''
    def create_tokens_tree(tokens, file):
       id = 1
       print(file)
       root = AnyNode(id=0, name=file, shortname=file)
       # Memory variable, used to store last token received of each type
       last_titulo = None
       last_capitulo = None
       last_seccion = None
       last_articulo = None
       current_node = None
       for token in tokens:
           token = token.replace("/^\s*\s*$/", "")

           if is_titulo(token):
               if not (re.match(".*[.].*", token[0:140])):
                     # formatToken is used to delete extra whitespaces added
by tokenizing algorithm in each appart.
                   token = formatToken(token)
                      # shortname is made by main words of the token, which
are always at the beginning
                   # name the rest of the token
                         current_node = AnyNode(id=id, parent=root, name="
".join(token.split('.')[2:]), shortname=" ".join(token.split('.')[0:2]))
                   last_titulo = current_node
                   # delete all minor nodes stored
                   last_capitulo = None
                   last_seccion = None
                   last_articulo = None
           elif is_capitulo(token):
               token = formatToken(token)
                   current_node = AnyNode(id=id, parent=last_titulo, name="
".join(token.split('.')[2:]), shortname=" ".join(token.split('.')[0:2]))
               last_capitulo = current_node
               last_seccion = None
               last_articulo = None
           elif is_seccion(token):
```

```
                # Check minor not null stored node
                if last_capitulo is None:
                    if last_titulo is None:
                            current_node = AnyNode(id=id, parent=root, name="
".join(token.split('.')[2:]), shortname=" ".join(token.split('.')[0:2]))
                    else:
                            current_node = AnyNode(id=id, parent=last_titulo,
name="               ".join(token.split('.')[2:]),               shortname="
".join(token.split('.')[0:2]))
                else:
                        current_node = AnyNode(id=id, parent=last_capitulo,
name="               ".join(token.split('.')[2:]),               shortname="
".join(token.split('.')[0:2]))
                last_seccion = current_node
                last_articulo = None
            elif is_articulo(token):
                token = formatToken(token)
                if last_seccion is None:
                    if last_capitulo is None:
                        if last_titulo is None:
                                current_node = AnyNode(id=id, parent=root,
name="               ".join(token.split('.')[2:]),               shortname="
".join(token.split('.')[0:2]))
                        else:
                                                current_node = AnyNode(id=id,
parent=last_titulo,    name="    ".join(token.split('.')[2:]),    shortname="
".join(token.split('.')[0:2]))
                    else:
                            current_node = AnyNode(id=id, parent=last_capitulo,
name="               ".join(token.split('.')[2:]),               shortname="
".join(token.split('.')[0:2]))
                else:
                        current_node = AnyNode(id=id, parent=last_seccion,
name="               ".join(token.split('.')[2:]),               shortname="
".join(token.split('.')[0:2]))
                last_articulo = current_node
            else:
                # is a part of an articulo
                token = formatToken(token)
                        current_node = AnyNode(id=id, parent=last_articulo,
```

```
name=token, shortname="")
            id += 1
        return root
```

Capítulo 8 Bibliografía

(1) PDF. (n.f) En Wikipedia. Recuperado el 17 de Junio de 2020 de https://es.wikipedia.org/w/index.php?title=PDF&oldid=126909663
(2) Code Explorer Visual Tool. (04, Septiembre, 2015), Recuperado el 17 de Junio de 2020 de http://www.openlawlab.com/2015/04/06/code-explorer-visual-tool/
(3) Constituição da República Portuguesa, Recuperado el 17 de Junio de 2020 de http://nsm-viz.nrc.pt/tree-map/const-pt
(4) Expresión regular. (n.f) En Wikipedia. Recuperado el 17 de Junio de 2020 de https://es.wikipedia.org/w/index.php?title=Expresi%C3%B3n_regular&oldid=126383428
(5) Edu Salguero. El patrón flux. (09, Marzo, 2018), Recuperado el 17 de Junio de 2020 de https://medium.com/@edusalguero/patron-flux-475d858af405
(6) Nodejs. (n.f) En Wikipedia. Recuperado el 17 de Junio de 2020 de https://es.wikipedia.org/w/index.php?title=Node.js&oldid=126903366
(7) Flask. (n.f) En Wikipedia. Recuperado el 17 de Junio de 2020 de https://es.wikipedia.org/w/index.php?title=Flask&oldid=125578927
(8) Express. (n.f) Express Infraestructura web rápida, minimalista y flexible para Node.js, Recuperado el 17 de Junio de 2020 de https://expressjs.com/es/
(9) https://medium.com/@stupid_arnob/create-node-js-server-using-express-without-express-c52e9a72fe1e
(10) Selección de arquitecturas y herramientas de programación. (06, Octubre, 2013), Recuperado el 20 de Junio de 2020 de https://entreunosyceros.net/arquitecturas-y-herramientas-programacion/
(11) Jose Mª Baquero García. Analizamos las características de la librería Axios, un ligero cliente HTTP para JavaScript. (05, Abril, 2019), Recuperado el 17 de Junio de 2020 de https://www.arsys.es/blog/programacion/axios/#:~:text=Axios%20es%20una%20librer%C3%ADa%20JavaScript,recibiremos%20respuestas%20f%C3%A1ciles%20de%20procesar.
(12) Miguel Angel Alvarez. Librería Axios: cliente HTTP para Javascript. (14, Septiembre, 2018), Recuperado el 17 de Junio de 2020 de https://desarrolloweb.com/articulos/axios-ajax-cliente-http-javascript.html
(13) Consulta. RAE, Recuperado el 17 de Junio de 2020 de https://dle.rae.es/consulta
(14) Consultar. RAE, Recuperado el 17 de Junio de 2020 de https://dle.rae.es/consultar