SISTEMA DE GESTIÓN Y BÚSQUEDA DE SENTENCIAS JUDICIALES UTILIZANDO PROCESAMIENTO DEL LENGUAJE NATURAL

Autor: Juan Nicolás Gumy

Email: juangumy15@gmail.com

Este Trabajo Final de Grado fue presentado al Departamento de Informática de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata el 15 de noviembre de 2021, como requisito para optar al grado de

Ingeniero en Informática

Director: Ing. Ariel Podestá

Co-Director: Ing. Bruno Constanzo

SISTEMA DE GESTIÓN Y BÚSQUEDA DE SENTENCIAS JUDICIALES UTILIZANDO PROCESAMIENTO DEL LENGUAJE NATURAL

Autor: Juan Nicolás Gumy

Email: juangumy15@gmail.com

Este Trabajo Final de Grado fue presentado al Departamento de Informática de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata el 15 de noviembre de 2021, como requisito para optar al grado de

Ingeniero en Informática

Director: Ing. Ariel Podestá

Co-Director: Ing. Bruno Constanzo

En memoria de Valentín

Índice general

Índice de tablas

Índice de figuras

Agradecimientos

Quisiera dedicar este apartado a aquellas personas que merecen un reconocimiento e hicieron posible que pudiera transitar todos estos años de carrera y culminar este camino con la presentación de este Proyecto Final. Por ello, quisiera dar mis agradecimientos a:

Mi familia, pero especialmente a mis padres y hermana por brindarme su apoyo incondicional, amor y compresión constante desde que tengo memoria. Gran parte de este logro es gracias a ellos, siempre alentándome a seguir adelante en los momentos buenos como adversos.

Mis amigos y compañeros de facultad con quienes compartí todos estos años de carrera, llevándome conmigo todas las experiencias vividas , tardes de estudio, y mejores recuerdos de la facultad.

Mis directores, Ing. Ariel Podestá e Ing. Bruno Constanzo. Me han acompañado a lo largo del desarrollo del presente proyecto, siempre predispuestos a ayudar, brindarme valiosos conocimientos y asesorarme en la toma de decisiones.

A la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata y a la educación pública, por haberme permitido estudiar esta carrera universitaria que me apasiona y formarme profesionalmente como Ingeniero en Informática.

Juan Nicolás Gumy

Mar del Plata, 15 de noviembre de 2021.

Resumen

El presente Proyecto Final fue desarrollado por Juan Nicolás Gumy, estudiante de la carrera Ingeniería en Informática, perteneciente a la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

En el actual informe, se expone el análisis, diseño e implementación de un sistema web de gestión de sentencias judiciales para el Juzgado de Garantías N° 4 de la ciudad de Mar del Plata. El objetivo principal del mismo es ser una herramienta de búsqueda de jurisprudencia para el profesional del Derecho, facilitándole esta tarea en su actividad laboral de una forma mas eficiente. El sistema permite realizar búsquedas semánticas de información como así también gestionar los documentos asociados e indexarlos de una forma estructurada y organizada por diferentes criterios y categorías de contexto judicial-penal.

Para lograr este objetivo, se utilizaron distintas herramientas y soluciones informáticas que involucran el Procesamiento del Lenguaje Natural, Reconocimiento de Entidades Nombradas y búsquedas Full Text Search, entre otros.

A lo largo del informe, se realizará un recorrido por todas las fases que conformaron este trabajo, desde el análisis, elicitación de requerimientos y diseño del sistema, hasta el desarrollo final del producto esperado. Previamente, se expondrá el análisis de las herramientas utilizadas y la investigación realizada sobre la aplicación de herramientas informáticas en el ámbito legal actual.

También, se presentará el trabajo realizado correspondiente al módulo reconocedor de entidades nombradas de índole legal. Dicho módulo es el encargado de procesar el texto de la sentencia y llevar a cabo la identificación de cualquier palabra o grupo de palabras pertenecientes a una categoría semántica (entidad), previamente entrenada con ejemplos.

Por último, se expondrá la información relacionada a la gestión del proyecto del presente Trabajo Final como así también las conclusiones y lecciones aprendidas.

Nomenclatura

API	Interfaz de Programación de Aplicaciones *(Application Programming Interface)*
BSD	Licencia de software BSD *(Berkeley Software Distribution)*
CENDOJ	Centro de Documentación Judicial
CIJ	Centro de Información Judicial
CSRF	*(Cross-site Request Forgery)*
CSS	Hoja de estilo en cascada *(Cascading Style Sheets)*
DRY	Principio No te repitas *(Don't Repeat Yourself)*
DTL	Lenguaje de Templates de Django *(Django Template Language)*
DTL	Lenguaje de Templates de Django *(Django Template Language)*
ERD	*(Entity-Relation diagram)*
F	Fórmula F *(F-Measure)*
FTS	Búsqueda Texto Completo *(Full Text Search)*
HTML	Lenguaje de Marcas de Hipertexto

	(HyperText Markup Language)
$JSON$	Notacion de Objetos JavaScript
	(JavaScript Object Notation)
MER	Modelo Entidad-Relación
	(Entity-Relation Model)
ML	Aprendizaje de máquina
	(Machine Learning)
$MPTT$	*(Modified Preorder Tree Travesal)*
MVC	Modelo-Vista-Controlador
	(Model View Controller)
MVT	Modelo-Vista-Template
	(Model View Template)
NER	Reconocedor de Entidades Nombradas
	(Named Entity Recognition)
$NIST$	Instituto Nacional de Estándares y Tecnología
	(National Institute of Standards and Technology)
NLP	Procesamiento del Lenguaje Natural
	(Natural Language Processing)
$NLTK$	*(Natural Language Toolkit)*
ORM	Mapeo Objeto-Relacional
	(Object Relational Mapping)
P	Medida de Precisión
	(Precision)
POS	Etiquetado gramatical
	(Part-of-speech)
R	Medida de Recall
	(Recall)
RI	Recuperación de la Información
	(Information Retrieval)

SGBD	Sistema Gestor de Base de Datos *(Database Management System)*
SQL	Lenguaje de Consulta Estructurada *(Structured Query Language)*
SRI	Sistema de Recuperación de Información *(Information Retrieval System)*
URL	*(Uniform Resource Locator)*
UX	Experiencia de Usuario *(User Experience)*
VCS	Sistemas de Control de Versiones *(Version Control Systems)*
XML	Lenguaje de Marcado Extensible *(Extensible Markup Language)*
XSS	Secuencia de Comandos en Sitios Cruzados *(Cross-site Scripting)*

Capítulo 1

Introducción

1.1. Introducción general

En el campo del derecho, se entiende la jurisprudencia como el conjunto de sentencias y demás resoluciones judiciales emitidas en un mismo sentido por los órganos judiciales de un ordenamiento jurídico determinado. Un juez puede usar este recurso como fundamento en su escrito final independientemente de lo presentado por la parte acusadora y/o defensora.

El conjunto de sentencias dictadas por los jueces tienen un doble interés. Por un lado, sirven para valorar los distintos casos particulares. Por otro, constituyen una referencia general para comparar otras situaciones jurídicas similares. Para el derecho, son importantes los antecedentes. La idea es que un mismo hecho sea juzgado con los mismos criterios a lo largo del tiempo, partiendo de la idea de que las leyes deben impartir justicia y deben ser equitativas, sin presentar incongruencias. Gracias a la jurisprudencia, se garantiza que un ordenamiento legal se mantenga en el tiempo y tenga coherencia procesal.

En el ejercicio legal, el acceso digital a la documentación (leyes, jurisprudencia, sentencias, etc.) es un importante capital para los profesionales del derecho. Un

acceso rápido, preciso y efectivo a la información puede hacer una gran diferencia en el trabajo diario, además de producir una reducción significativa del esfuerzo requerido para obtener toda la información relevante para llevar adelante una acción.

La idea de este proyecto surge a partir de una problemática y necesidad proveniente del Juzgado de Garantías N° 4 de la ciudad de Mar del Plata, propuesta por el Laboratorio de Informática Forense (InFo-Lab) de Mar del Plata. Actualmente el Juzgado posee una gran cantidad de sentencias que son utilizadas como fuente de consulta de jurisprudencia, en diversos formatos no estructurados. Estas incluyen situaciones específicas poco comunes o casos que pueden considerarse como situaciones especiales y de interés.

El proceso de búsqueda de jurisprudencia en el juzgado incluye en la mayoría de los casos diversas tareas como apelar al diálogo entre el personal sobre el conocimiento de algún fallo en el pasado relacionado a la temática que se esté investigando, acudir a la memoria personal sobre sentencias dictadas en base a la experiencia, consulta de archivos personales y hasta la búsqueda en ficheros manuales que registran causas clasificadas según su temática, como por ejemplo medidas cautelares, sobreseimientos, estupefacientes, homicidios, nulidades, arrestos domiciliarios, unificación de sentencias, entre otras.

El personal del juzgado debe leer, analizar y revisar cada uno de estos documentos de forma manual, en busca de información relevante. Esto conlleva una gran inversión de tiempo en resolver la tarea, que podría utilizarse de una forma más eficiente en actividades más estratégicas. Sumado a este problema, se le adiciona el hecho de que al realizar un análisis manual, pueden existir sentencias o casos omitidos, debido a factores como fatiga, cansancio y al gran volumen de información a consultar.

Cabe destacar que no se posee un sistema informático o soporte digital que

facilite el acceso y análisis de dicha información de forma organizada, ni herramientas dedicadas específicamente a la gestión de los documentos. La implementación del sistema que se describirá en la siguiente sección propone resolver esta problemática y sentar una base para el tratamiento, búsqueda y recuperación de la información que proveen las sentencias judiciales, facilitando su uso a todo el personal interno del juzgado.

1.2. Objetivo del proyecto

El objetivo de este proyecto de informática es la implementación de un sistema informático para el Juzgado de Garantías N° 4 de la ciudad a utilizar por jueces, abogados y profesionales del área del derecho.

Dicho sistema tiene como propósito ser una herramienta disponible al personal, no sólo para la búsqueda automatizada de jurisprudencia, sino también para la óptima organización e indexación de la información. Para lograr este propósito, y como requerimiento inicial del proyecto, se busca aplicar técnicas del NLP para el análisis del texto judicial y corpus legal, y desarrollar un sistema web para la indexación de las sentencias y búsqueda semántica disponible para el personal del juzgado.

Adicionalmente, se busca utilizar todos los conocimientos teóricos y prácticos adquiridos a lo largo de la carrera, como así también incorporar y mejorar aquellos relacionados a aspectos técnicos como procesamiento del lenguaje natural, programación web, bases de datos, como aquellos relacionados a la gestión de proyectos.

Capítulo 2

Estado del Arte

2.1. Introducción

Previo al diseño e implementación de la solución, se han investigado otros proyectos y sistema similares al producto deseado, tanto *open-source* como de código cerrado, cuyas implicaciones involucran el campo del sector legal y en algunos casos, la inteligencia artificial. Se presentan brevemente los resultados. Luego se mencionarán algunas de las herramientas y librerías más importantes para el NLP actualmente presentes en dicho campo de estudio.

2.2. Aplicaciones y proyectos en el Área Legal

En una primera instancia, se centró la investigación en sistemas implementados en Argentina, con el objetivo de observar el ámbito local. Se encontraron algunos sistemas de búsqueda y análisis de jurisprudencia en organismos públicos del país. Algunos de ellos incluyen la utilización de herramientas de la inteligencia artificial. También se hallaron proyectos de países del exterior *open-source* que combinan técnicas del NLP con el ámbito legal. Se detallan a continuación.

2.2.1. Centro de Información Judicial

El Centro de Información Judicial es la principal agencia de noticias del Poder Judicial de la Nación. En su página web se publican noticias de actualidad del ámbito jurídico-legal nacional, así como también decisiones judiciales de trascendencia. Además, el CIJ cuenta con dos bases de consulta de jurisprudencia, ambas de carácter público y gratuito [1].

Las sentencias disponibles en las bases del CIJ carecen de una clasificación de acuerdo a un catálogo de voces o temas de referencia. Por otro lado, las bases del CIJ no disponen de una vista preliminar de los resultados arrojados y, en principio, exigen la descarga de la sentencia en formato (.pdf) para poder visualizar el contenido. Además, plantean ciertas dificultades para realizar una búsqueda temática medianamente sofisticada. No obstante ello, la mayor utilidad de las bases del CIJ radica en el gran número de sentencias que contienen, siendo una gran base de datos de sentencias disponible para el público.

2.2.2. Prometea

En 2017, la Fiscalía de la Ciudad Autónoma de Buenos Aires desarrolló Prometea, un sistema que aplica técnicas de la Inteligencia Artificial para preparar automáticamente dictámenes judiciales.

En particular, esta herramienta consiste en un sistema de software que tiene como cometido principal la automatización de tareas reiterativas y la aplicación de árboles de decisión, para la elaboración automática de dictámenes jurídicos basándose en casos análogos para cuya solución ya existen precedentes judiciales reiterados [2].

En la interfaz de usuario utiliza un asistente inteligente que incluye un chat conversacional y un asistente de voz. Ambos tipos de interacciones requieren

el empleo de técnicas para el reconocimiento del lenguaje natural (el lenguaje humano) ya sea hablado o escrito. Su idea principal apunta a la eliminación de clicks y apertura de ventanas para agilizar estas tareas.

2.2.3. Pretoria

Pretoria es un sistema de Inteligencia Artificial basado en Prometea, el programa que se utiliza en el Ministerio Público Fiscal de la Ciudad de Buenos Aires. Actualmente es aplicado en la Corte Constitucional de Colombia y fue creado por IALAB (Buenos Aires).

La Corte de Colombia recibe, en promedio, 2700 acciones de tutela por día de las cuales 1400 refieren al derecho a la salud. Las tutelas son los amparos que interponen personas que pertenecen a los sectores más vulnerables de la sociedad. El sistema contribuye a agilizar la revisión de sentencias para ayudar al juez a tomar decisiones. Por ejemplo, es capaz de leer, analizar, detectar y sugerir 32 casos prioritarios de entre 2016 sentencias escaneadas de la Corte en menos de 2 minutos, algo que para una persona humana le hubiera llevado 96 días hábiles de trabajo (en jornadas de 7 hs.), según su sitio web [3].

El sistema realiza automáticamente y sin intervención humana tres tareas:

1. La búsqueda de información de interés para la selección de las sentencias de índole salud.

2. La categorización siguiendo criterio de relevancia establecidos por la Corte Constitucional.

3. La elaboración de estadísticas que permiten visualizar de manera íntegra las tutelas presentadas en ese ámbito de la justicia.

El sistema funciona a través de una plataforma web. Consta de distintas secciones, entre ellas un buscador y estadísticas. Uno de sus creadores asegura que uno de los problemas con los que se encontraron a la hora de crear Pretoria es que no había suficientes datos de entrenamiento (disponibles y digitalizados) para desarrollar una solución, es decir que faltaba los volúmenes necesarios de información para entrenar el sistema empleando redes neuronales. Y ése fue uno de los motivos por los cuales optaron por generar otro tipo de modelo de predicciones, basado en un sistema de aprendizaje automático supervisado, siguiendo la técnica de clasificación *topic modeling*, iterando hasta lograr una tasa de acierto aceptable.

2.2.4. Centro de Documentación Judicial

El CENDOJ o Centro de Documentación Judicial es un organismo que se ubica en la órbita del Consejo General del Poder Judicial de España y que se encarga de la recopilación, digitalización y clasificación de la jurisprudencia de los tribunales españoles, entre sus numerosas funciones.

CENDOJ logró el tratamiento informático de más de 6.500.000 decisiones correspondientes a los diferentes Tribunales de España. El trabajo del Centro se ajusta a una estricta política de protección de datos personales razón por el cual insume una tarea de gran complejidad la anonimización de los datos de las personas físicas que aparecen en el contenido de las decisiones [4].

Bajo una plataforma web, el sistema se compone de un buscador de sentencias gratuito donde se puede acceder a todas las resoluciones judiciales emitidas por el Tribunal Supremo, Audiencia Nacional, Tribunales Superiores de Justicia, Audiencias Provinciales y Tribunales Militares de España. En colaboración con los magistrados, el sistema desarrolla un análisis jurídico de las sentencias: resumen,

voces y vínculos a las normas y jurisprudencia que se citan en el contenido de las mismas. Cuenta con filtros para refinar la búsqueda, temáticas de interés y presentación de estadísticas, entre otras funcionalidades.

2.2.5. Blackstone

Blackstone es un proyecto *open-source* de investigación experimental del laboratorio ICLR&D de UK (Incorporated Council of Law Reporting for England and Wales) que consiste principalmente en un modelo de spaCy [5] para el procesamiento de texto legal no estructurado. Según su documentación [6], Blackstone es el primer modelo de código abierto entrenado específicamente para su uso en textos de formato largo que contienen entidades y conceptos de derecho.

Está construido sobre la librería de NLP spaCy, y ha sido entrenado con datos que abarcan un período temporal considerable (desde textos redactados en la década de 1860 hasta el presente). Los modelos se han formado en la jurisprudencia inglesa (idioma inglés) y la librería se ha construido teniendo en cuenta las peculiaridades del sistema jurídico de Inglaterra y Gales.

Los creadores del proyecto afirman que es una versión prototipo de un proyecto altamente experimental. Como tal, la precisión de los modelos de Blackstone es baja (F1 en el NER , aproximadamente 70 %), pero mejorará con el tiempo a medida que el modelo se perfeccione.

Su pipeline de procesamiento cuenta de:

- un tokenizer, un tagger y un parser correspondiente al modelo de inglés de spaCy.
- un NER (Reconocedor de Entidades Nombradas) entrenado específicamente para textos legales. Las entidades entrenadas son: CASENAME (nombres de casos), CITATION (identificadores únicos para casos notificados y no

denunciados), INSTRUMENT (instrumentos legales escritos), PROVISION (unidad dentro de un instrumento legal escrito), COURT (corte o tribunal) y JUDGE (referencias a jueces).

- un categorizador de textos, con las siguientes categorias: AXIOM (un texto que parece postular un principio bien establecido), CONCLUSION (un texto que se asemeja a un hallazgo, celebración, determinación o conclusión), ISSUE (discución de un tema o pregunta), LEGALTEST (el texto describe una prueba legal) y UNCAT (si el texto no corresponde a las categorias mencionadas anteriormente).

- detector de abreviaciones (por ejemplo, ECtHR = European Court of Human Rights).

- *legislation linker* (componente alpha).

- detección de referencias de casos compuestos (componente alpha).

2.3. Librerías para Procesamiento del Lenguaje Natural

En esta sección se describen las principales librerías desarrolladas para el Procesamiento del Lenguaje Natural. En la actualidad, hay disponibles distintos paquetes de herramientas y librerías de amplia distribución destinadas al procesamiento lingüístico de carácter general. Estos paquetes están disponibles en distintos idiomas. Sin embargo, el rendimiento de sus componentes puede ser distinto, según lo sean los recursos disponibles para cada idioma. En el pasado, solo los expertos podían formar parte de proyectos de procesamiento del lenguaje

natural que requerían un conocimiento superior de matemáticas, aprendizaje automático y lingüística. Ahora, los desarrolladores pueden usar herramientas listas para usar que simplifican el preprocesamiento de texto para que puedan concentrarse en construir modelos de aprendizaje automático. Algunas de las bibliotecas de código abierto de NLP más conocidas son:

- NLTK (Natural Language Toolkit): es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural simbólico y estadísticos para el lenguaje de programación Python. Es una de las más populares, admite tareas como clasificación, derivación, etiquetado, análisis, razonamiento semántico y tokenización. La librería fue desarrollada por Steven Bird y Edward Loper en la Universidad de Pennsylvania y jugó un papel clave en la investigación revolucionaria en el NLP [7]. Es bastante versátil, pero compleja de utilizar. NLTK puede ser bastante lento y no cumple con las demandas del uso de producción acelerado. La curva de aprendizaje es empinada, pero se disponen de recursos como un libro llamado *Natural Language Processing with Python* para aprender más sobre los conceptos detrás de las tareas de procesamiento del lenguaje.

- CoreNLP: Esta librería fue desarrollada en la Universidad de Stanford y está escrita en Java. Aún así, está equipado con wrappers para muchos lenguajes diferentes, como Python. Con solo unas pocas líneas de código, permite la extracción de todo tipo de propiedades de texto, como el reconocimiento de entidades nombradas o el *part-of-speech tagging*. En cuanto a su performance, es rápida y funciona muy bien en entornos de producción. CoreNLP actualmente admite 6 idiomas: árabe, chino, inglés, francés, alemán y español.

- Apache OpenNLP: es un kit de herramientas desarrollado en Java basado en aprendizaje automático para el procesamiento de texto en lenguaje natural. Admite las tareas de NLP más comunes, como tokenización, segmentación de oraciones, *part-of-speech tagging*, extracción de entidades nombradas, fragmentación, análisis y resolución de correferencia. Cada una de estas funciones es accesible a través de su API. OpenNLP también incluye aprendizaje automático basado en entropía máxima y perceptrón.

- SpaCy: es una librería para el procesamiento avanzado del lenguaje natural en Python y Cython. Se diseñó para su uso en entornos de producción. SpaCy viene con modelos estadísticos previamente entrenados y vectores de palabras, y actualmente admite la tokenización para más de 60 idiomas. Cuenta con modelos de redes neuronales convolucionales de velocidad de última generación para etiquetado, análisis y reconocimiento de entidades con nombre y una fácil integración de aprendizaje profundo . Es un software comercial de código abierto, lanzado bajo la licencia MIT.

- TextBlob: está construida sobre otras dos librerías muy famosas de Python: NLTK y Pattern. La principal ventaja de TextBlob es que permite combinar el uso de las dos herramientas anteriores en un interfaz más simple. Permite análisis morfológico, extracción de entidades, análisis de opinión, traducción automática, entre otras funcionalidades. Es muy útil para diseñar prototipos. Sin embargo, también heredó las principales falencias de NLTK: es demasiado lento para entornos de producción.

- Gensim: es una librería de Python que se especializa en identificar similitudes semánticas entre dos documentos a través del modelado de espacio vectorial y el kit de herramientas de modelado de temas. Posee una gran optimización del uso de memoria y velocidad de procesamiento (con la ayuda

de la librería NumPy).

Capítulo 3

Metodología

3.1. Introducción

A lo largo de este capítulo se presentará la metodología de trabajo empleada para este proyecto, como así también las herramientas utilizadas para la realización del mismo.

3.2. Metodología de trabajo

Para la realización de este proyecto final, se contempló la aplicación de metodologías ágiles, empleadas en los ámbitos profesionales de la Ingeniería del Software, en donde el producto se va construyendo de manera iterativa e incremental, y los requerimientos y soluciones evolucionan con el tiempo según la necesidad del proyecto y de los distintos actores con poder de decisión o *stakeholders* involucrados.

Agile es un enfoque iterativo para la gestión de proyectos y el desarrollo de software que ayuda a los equipos a entregar valor a sus clientes de forma más rápida y eficiente. En lugar de realizar un lanzamiento único del producto, un

equipo ágil entrega el trabajo en pequeños, pero funcionales incrementos denominados sprints. Los requisitos, planes y resultados se evalúan continuamente, de modo que los equipos tienen un mecanismo natural para responder rápidamente a los cambios. El feedback y retroalimentación en este proceso es primordial para mejorar incremento a incremento.

Uno de los métodos ágiles mas conocidos y aplicados en la actualidad es Scrum. Dicho modelo se eligió como metodología a seguir para este trabajo, aunque no se pudieron aplicar todos los roles que la metodología exige debido a que el presente trabajo final está conformado por una única persona. Sin embargo, se trató de adoptar la esencia y filosofía de la metodología, con el objetivo de incorporar una estrategia de desarrollo iterativa e incremental, priorización y planificación de tareas, entre otras actividades. En la Fig 3.1 se puede observar un enfoque general de la metodología.

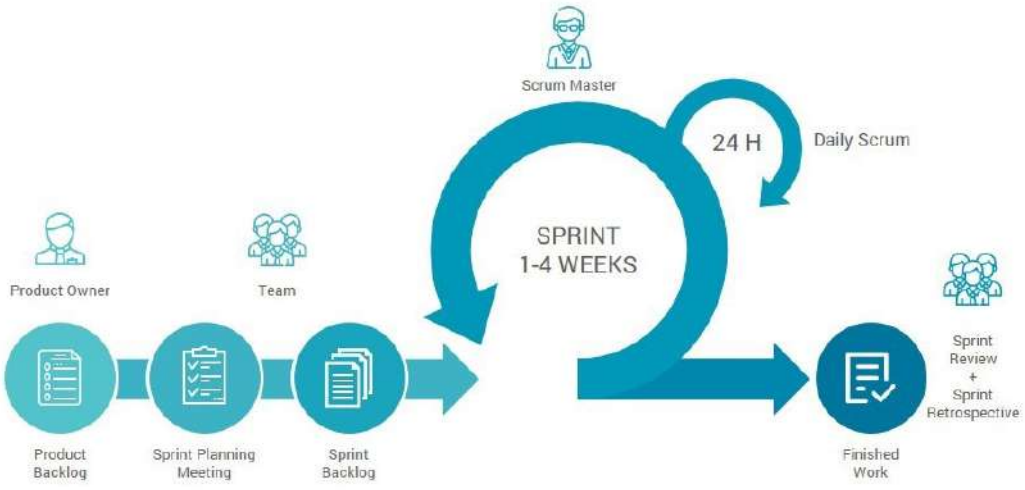


Figura 3.1: Metodología Scrum. *Fuente: Zoraida Ceballos de Mariño, SCRUM*

Una herramienta que resultó de gran utilidad en la gestión del proyecto fue

Trello, una aplicación muy utilizada tanto a nivel profesional como personal, similar a la reconocida herramienta Jira. Se basa en el método Kanban, donde se utilizan las típicas columnas *To Do*, *Doing* y *Done*. La idea principal es listar las tareas -o también denominadas *user stories*- que componen la realización del proyecto e ir colocandolas en tantas columnas sean necesarias según su estado en el sprint.

Como se ve en la Fig. 3.2 , se dispone de un Product Backlog con las tareas a realizar en los distintos sprints. Cada tarea (tarjeta representada en el tablero Kanban) posee un numero al comenzar su título el cúal es un puntaje asignado (task points) de acuerdo a la serie de Fibonacci, para poder estimar su dificultad o esfuerzo de manera creciente. También, se utilizaron *tags* con distintos colores en cada tarjeta para indicar su finalidad. Algunos de los *tags* se definieron como: Diseño, Desarrollo, FE, BE, Bugs, NLP Task, Meeting, entre otros.

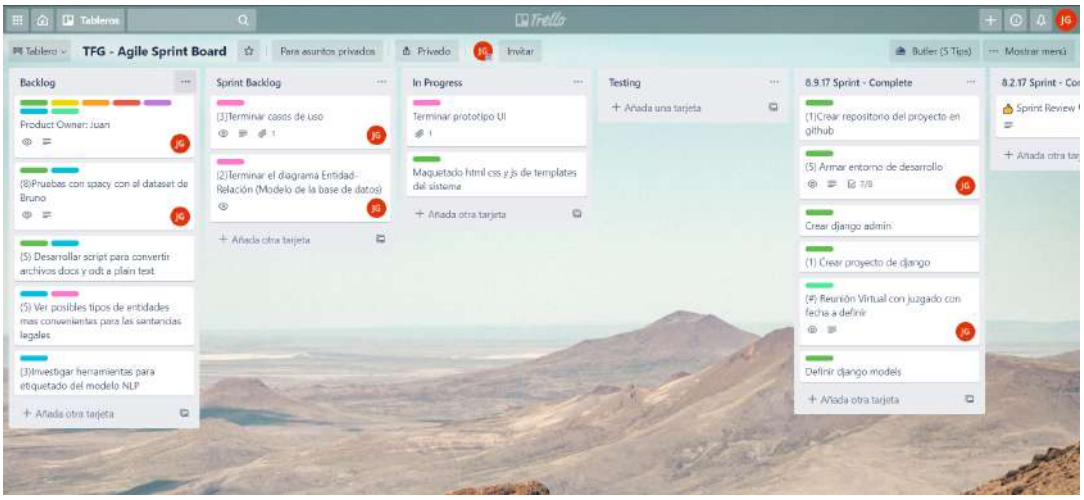


Figura 3.2: Vista de la pizarra de tareas Kanban en Trello. *Fuente: Elaboración propia*

Asimismo se mantuvo un registro de las tareas realizadas, con su numero de identificación, descripción, tiempo empleado y su estado, con el objetivo de

facilitar la gestión del proyecto. Es importante mantener un control sobre el backlog para identificar dependencias de forma anticipada y resolver los problemas o *blockers* que impedirán la resolución de una tarea.

Se organizaron reuniones virtuales con el personal del juzgado y directores del proyecto, con el objetivo de realizar consultas, validación en requerimientos y muestra del producto funcional conforme las iteraciones avanzaban. Si bien no fue posible seguir la metodología como lo indica la teoría, tener presente un proceso de trabajo facilita la organización de los tiempos y esfuerzos, aspectos súmamente necesarios en la Ingeniería del Software.

3.3. Herramientas

A lo largo de todo el proyecto fue necesario aplicar diversas herramientas de software para resolver distintos problemas. En esta sección, se describirán cuáles fueron las herramientas, lenguajes de programación y frameworks utilizados, junto con las alternativas evaluadas a la hora de elegir la opción mas adecuada. Sin dudas estas decisiones son críticas a la hora de abordar la implementación del proyecto, por lo que debieron analizarse en una etapa temprana del mismo.

3.3.1. Elección de lenguajes de programación y frameworks web

Para la elección del lenguaje de programación, se tuvieron en cuenta distintos aspectos. En primer lado, el requerimiento de trabajar con NLP hizo que se investiguen distintas alternativas de librerías y lenguajes a la hora de desarrollar en este campo de la Inteligencia Artificial. Sin dudas, el lenguaje Python fue el que más se destacó como la alternativa utilizada y popular, frente a otros lenguajes

como Java, C++ y R. Sumado a este factor, fue ampliamente aconsejado por los directores del presente proyecto final, debido a la buena experiencia que han tenido con Python en diferentes proyectos en sus ámbitos profesionales.

Python es un lenguaje interpretado, dinámico, multiparadigma y multiplataforma. Su filosofía hace hincapié en la legibilidad y simplicidad del código. Uno de los aspectos que más lo destacan es la curva de aprendizaje, la cual es relativamente asequible. No fue una dificultad aprender los aspectos mas importantes de este lenguaje para abordar el desarrollo de este proyecto, gracias a la cantidad de recursos, libros y comunidad *open source* para adquirir los conocimientos necesarios. Su gran comunidad fue un aspecto clave a la hora de consultar información, documentación, o requerir colaboración y/o consejos en distintos tipos de desarrollos.

El modo de trabajo con Python viene determinado por la combinación de módulos y paquetes que se pueden importar para añadir nuevas funcionalidades al análisis o desarrollo de un proyecto específico. Por ejemplo, es posible importar un módulo matemático cargado con todo tipo de funciones matemáticas, como raíces cuadradas y cosenos para el calculo de diferentes parámetros o, al entrenamiento de un modelo de *Machine Learning* para el Reconocimiento de Entidades Nombradas perteneciente al campo del NLP.

Otra ventaja de Python es que su uso no esta solamente limitado a la inteligencia artificial, sino que es ampliamente utilizado en el desarrollo web. Para continuar en una misma linea y poder tener una compatibilidad total a la hora de integrar el modulo encargado de reconocer entidades y el sistema web, se decidió analizar las distintas alternativas en frameworks web que ofrece Python.

Frameworks para el sistema web

Dentro de las opciones que ofrece Python en términos de frameworks para backend, se encuentran Django [8] y Flask [9].

Django es un potente framework en lenguaje Python destinado al desarrollo web escalable y de alta calidad. Fue diseñado para resolver gran parte de los problemas comúnmente conocidos en el ámbito de la programación web, por lo que uno puede concentrarse en desarrollar una aplicación sin la necesidad de "reinventar la rueda", como lo explica su documentación oficial [8]. Es gratuito y de código abierto, tiene una comunidad próspera y activa, una gran documentación y muchas opciones de soporte.

Flask, por su parte, es un framework minimalista escrito en Python que permite crear aplicaciones web de forma rápida y con un mínimo número de líneas de código. Posee una licencia BSD.

Ventajas y desventajas de Django

Algunas de las ventajas de este framework son [10]:

- Versátil: puede funcionar con cualquier framework en el lado del cliente, y puede devolver contenido en casi cualquier formato (incluyendo HTML, RSS feeds, JSON, XML, etc). Puede ser utilizado para construir cualquier tipo de sistema web. Fue creado para trabajar bajo un patrón MVT quien se encarga del manejo de controladores, esto lo caracteriza en un framework reusable y permite el desarrollo ágil.

- Seguro: ayuda a evitar varios errores comunes de seguridad informática. Django permite protección contra algunas vulnerabilidades de forma predeterminada, como la inyección SQL, ataques XSS, falsificación de solicitudes

entre sitios y *clickjacking*. Además, proporciona una manera segura de administrar cuentas de usuario y contraseñas, evitando así errores comunes como colocar informaciones de sesión en cookies donde es vulnerable (en lugar de eso las cookies solo contienen una clave y los datos se almacenan en la base de datos) o se almacenan directamente las contraseñas en un hash de contraseñas.

- Escalable: usa un componente basado en la arquitectura *shared-nothing* (cada parte de la arquitectura es independiente de las otras, y por lo tanto puede ser reemplazado o cambiado si es necesario). Teniendo en cuenta una clara separación entre las diferentes capas significa que puede escalar para aumentar el tráfico al agregar hardware en cualquier nivel: servidores de cache, servidores de bases de datos o servidores de aplicación. Un ejemplo muy famoso es Instagram, cuyo backend fue construido con este framework y escalado por su gran tráfico y volumen de usuarios.

- Mantenible: el código de Django está escrito usando principios y patrones de diseño para fomentar la creación de código mantenible y reutilizable. En particular, utiliza el principio *Don't Repeat Yourself* para que no exista una duplicación innecesaria, reduciendo la cantidad de código. También promueve la agrupación de la funcionalidad relacionada en aplicaciones reutilizables y en un nivel más bajo, agrupa código relacionado en módulos (siguiendo el patrón MVT).

- Portable: al ser escrito en Python, es multiplataforma. Puede ejecutar sus aplicaciones en muchas distribuciones de Linux, Windows y Mac OS X. Además, cuenta con el respaldo de muchos proveedores de alojamiento web, y que a menudo proporcionan una infraestructura específica y documentación para el alojamiento de sitios de Django, como por ejemplo Heroku.

- Estructura del proyecto: provee una estructura del proyecto autogenerado, muy útil a la hora de organización y optimización de tiempo y código.
- Panel de Administración: tiene un panel de administración incorporado para gestionar los modelos y relaciones generadas de la base de datos. Puede configurarse y adaptarse en base a las necesidades del proyecto.

Algunas de sus desventajas son:

- A pesar de su buena documentación, es muy extensa. La opcion de buscador que ofrece la página oficial no suele arrojar resultados significativos, por lo que el lector debe tener en claro el concepto a indagar frente a una gran cantidad de páginas de documentación.
- La curva de aprendizaje del framework puede considerarse alta. Para aprovechar la mayoría de los aspectos que brinda, se deben adquirir muchos conceptos importantes que ofrece el framework.
- A la hora de realizar un API Rest conlleva cierta condición de dificultad a comparación de Flask, ya que se debe aprender asimismo el Django Rest Framework, el cual presenta conceptos teóricos destinados a serialización de objetos y construcción de servicios Rest.

Ventajas y desventajas de Flask

Estas son algunas de las ventajas de Flask:

- Prototipado: se destaca en instalar extensiones o complementos de acuerdo al tipo de proyecto que se va a desarrollar, es decir, es muy favorable para el prototipado rápido de proyectos.

- Velocidad: su velocidad es mejor a comparación de Django. Generalmente el desempeño que tiene Flask es superior debido a su diseño minimalista que tiene en su estructura.
- Curva de aprendizaje: al tratarse de un microframework, su curva de aprendizaje es baja. Posee una extensa documentación.
- Adaptable: se integra con otras herramientas para incrementar sus funciones, como Jinja2 (motor de plantillas web) o SQLAlchemy (kit de herramientas SQL de código abierto).

Estas son algunas desventajas de Flask:

- Su sistema de autenticación de usuarios es muy básico, a comparación del potente sistema de autenticación que utiliza Django.
- Su representación de *plugins* no es tan extensa como la tiene Django.
- El ORM para conectar con las bases de datos, SQLAlchemy es externo, y no incorporado con el framework.
- Se trata de un framework que genera dificultades a la hora de realizar migraciones o pruebas unitarias, a diferencia de Django.

Elección final del framework web

Analizando todos los factores mencionados anteriormente, se decidió por utilizar Django como framework para el desarrollo web del sistema. El hecho de generar un producto para un cliente final hace que se aprovechen en casi su totalidad las herramientas y prestaciones que ofrece este framework, como interacciones con la base de datos, generación de interfaces y documentos HTML para el usuario con su sistema de renderizado de plantillas, sistema de autenticación de usuarios,

soporte para búsquedas full text search con el ORM compatible con PostgreSQL, soporte para tests unitarios, entre muchas otras características. Adquirir los conocimientos necesarios para aprender el funcionamiento del framework y su ORM no fueron un impedimento para utilizarlo, gracias a su completa documentación como así tambien los recursos de acceso libre en la web.

3.3.2. Tecnologías para el Front-end

En cuanto a las tecnologías Front-end, se decidió utilizar el motor de renderizado de *templates* que trae incorporado Django en combinación con el lenguaje JavaScript y la librería de estilos Bootstrap. Una *template* (plantilla) contiene las partes estáticas de la salida HTML deseada, así como también sintaxis especial que describe cómo se insertará el contenido dinámico. Dicha sintaxis está definida por el DTL (*Django Template Language*), donde se pueden definir variables pertenecientes de un contexto, *tags* y lógica para el tratamiento dinámico de los datos.

Para los estilos de las diferentes páginas que conforman la interfaz de usuario, se utilizó el popular framework de CSS llamado Bootstrap [11] para desarrollar sitios web responsive. Dicho framework combina CSS y JavaScript para estilizar los elementos de una página HTML y proporciona interactividad en las páginas, ofreciendo una serie de componentes que facilitan la interacción con el usuario, como menús de navegación, controles de página, modales, barras de progreso, entre otros.

3.3.3. Sistema de control de versiones

Con respecto al sistema de control de versiones, se utilizó GIT [12]. El control de versiones ayuda a rastrear y gestionar los diferentes cambios realizados en el

conjunto de archivos de código que conforman el proyecto. Git es un sistema de control de versiones distribuido de código abierto y gratuito, diseñado para gestionar y administrar el versionado de proyectos.

Como plataforma online de repositorios se utilizó Github. El mismo es un servicio basado en la nube que aloja el sistema de control de versiones Git. Éste permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso.

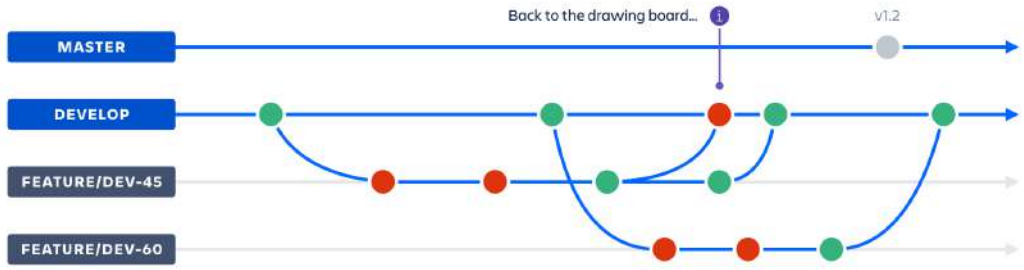


Figura 3.3: Diagrama del Git-Flow. *Fuente: Atlassian, Git Tutorial*

El flujo de trabajo que se utilizó fue el Git-flow, como se puede ver en la Fig. 3.3. Ante cada nueva *feature* o funcionalidad, o resolución de *bug*, se creó una rama local partiendo de la rama principal o maestra del repositorio remoto. Luego, se realizan los cambios pertinentes, se valida y se *mergea* a la rama principal remota creando *Pull Requests*. Cada mensaje de *commit* sigue una expresión regular conformada por el numero *id* identificatorio de la tarea junto con su descripción, para un mejor control y revisión de los cambios.

3.3.4. Entorno de Desarrollo Integrado

Como IDE se utilizó el popular editor de código fuente Visual Studio Code (VScode), el cual incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis y autocompletado de código, entre otras funcionalidades. Una de las principales ventajas es su capacidad de personalización, al poder instalar distintas extensiones o plugins para cualquier lenguaje como linters, formateadores de código, creadores de snippets de código u atajos. Es gratuito y de código abierto.

Capítulo 4

Trabajo Realizado

4.1. Introducción

En el siguiente capítulo se recopilará las distintas etapas afrontadas para la elaboración de la solución de software para implementar un sistema web encargado de la gestión y procesamiento a través de técnicas del procesamiento del lenguaje natural de las sentencias judiciales del juzgado. Como se expuso en la sección 4.2, se trató de seguir un proceso iterativo e incremental. Esto significa que se combinan elementos de un proceso lineal (Comunicación, Planeación, Modelado, Construcción y Despliegue) aplicando secuencias lineales en forma escalonada a medida que avanza el calendario de actividades, como se puede ver en la Fig. 4.1. Cada secuencia lineal (sprints) produce "incrementos" de software susceptibles de entregarse.

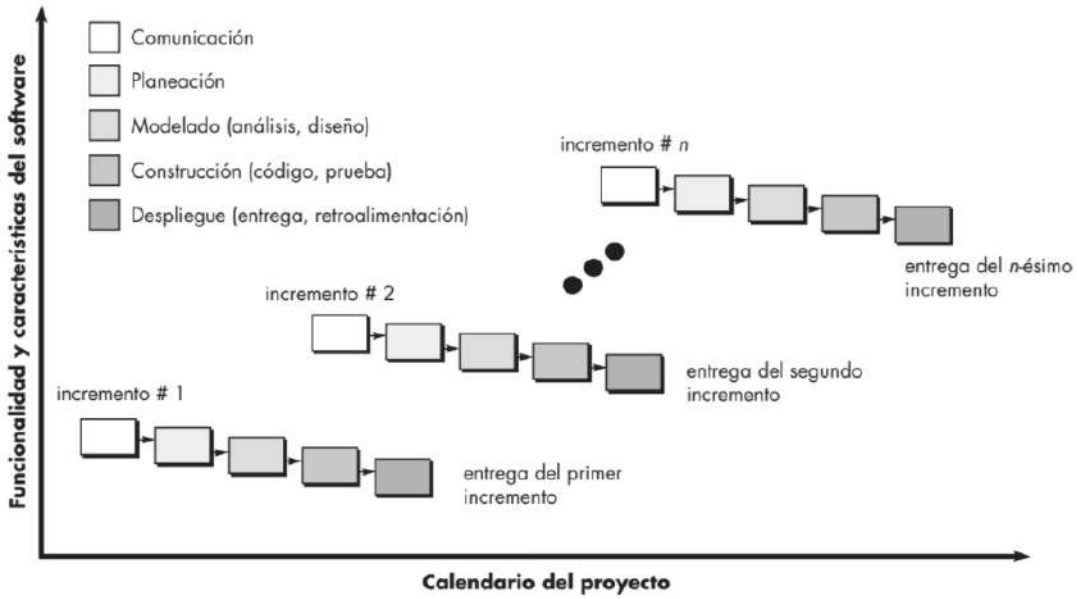


Figura 4.1: Modelo Incremental. *Fuente: Ian Sommerville. Ingeniería de software*

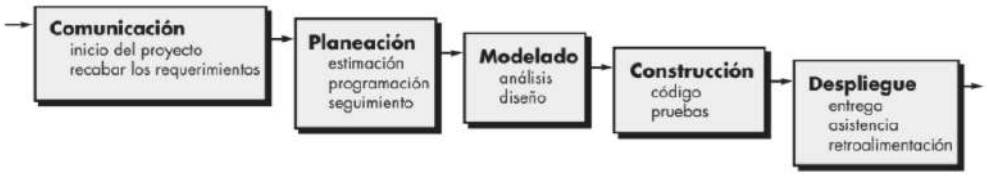


Figura 4.2: Modelo Lineal. *Fuente: Ian Sommerville. Ingeniería de software*

Adicionalmente se expondrá el desarrollo realizado para un modelo reconocedor de entidades nombradas relacionadas con el Derecho junto con su integración en el sistema web propuesto.

4.2. Elicitación de requerimientos

La Elicitación de Requerimientos es la base fundamental en el desarrollo de proyectos de software y es la fase que proporciona el impacto más alto en el diseño

y en las demás fases del ciclo de vida del producto. De realizarse apropiadamente, puede ayudar a gestionar y disminuir los cambios y correcciones en los requerimientos, lo que conlleva a la optimización de costos en el desarrollo de software por pérdidas de tiempo y una mejor gestión del riesgo.

La elicitación requiere orientar distintas técnicas para interactuar con las fuentes en las que potencialmente se encuentran los elementos para la formulación de los requerimientos. Los *stakeholders* son la principal fuente de todos los requerimientos y participan directamente en el proceso de elicitación.

En el marco de este trabajo, se emplearon distintas técnicas de elicitación de requerimientos, pero principalmente se realizaron entrevistas grupales con los integrantes del Juzgado, actores principales del sistema. Debido al contexto de pandemia originado por el COVID-19 en el que se llevó a cabo el proyecto final, dichas entrevistas se ejecutaron de manera virtual, a través de plataformas de videoconferencia. Si bien a priori esta situación podía llegar a ser un limitante para la toma de requerimientos, las distintas reuniones pactadas con el juzgado se realizaron sin inconvenientes y se pudieron obtener los requerimientos del sistema. En el Apéndice B se presenta el protocolo de entrevista empleado.

4.2.1. Proceso de consulta de jurisprudencia

Las primeras reuniones con los integrantes del juzgado se centraron principalmente en indagar sobre el dominio del problema, es decir, adquirir la mayor cantidad de información sobre cómo realizaban el proceso de búsqueda de jurisprudencia en su ámbito laboral.

La búsqueda de información de interés y jurisprudencia era un proceso mayormente manual e individual. Algunas de las tareas que incluía eran apelar al diálogo entre los integrantes del juzgado sobre el conocimiento de algún fallo en

el pasado relacionado a la temática que se esté trabajando y acudir a la memoria personal sobre sentencias dictadas en base a la experiencia. El juzgado no poseía ningún sistema informático interno que permita gestionar los archivos de sentencias dictadas de forma centralizada y disponible para todos los empleados.

Uno de los mecanismos que utilizaban, era la clasificación de algunas de las sentencias por temáticas de interés a través de un fichero elaborado manualmente. Este fichero era confeccionado por un integrante del juzgado, encargado también de su mantenimiento.

Cada ficha estaba compuesta por:

- Tema (Por ejemplo, calificación, medidas cautelares, sobreseimientos, estupefacientes, homicidios, nulidades, arrestos domiciliarios, unificación de sentencias, entre otros).
- Número de causa.
- Fecha de resolución.

Si bien el fichero era de suma utilidad, por su naturaleza manual y manuscrita, presentaba dificultades principalmente para su gestión, sin contar con el esfuerzo y tiempo dedicado para que el mismo se encuentre actualizado con las nuevas sentencias que se quisieran agregar.

Otro de los mecanismos que utilizaban era consultar jurisprudencia a través de colecciones personales de sentencias, en su mayoría en formato *.docx*, que presentaban situaciones poco comunes o casos excepcionales. Dichas colecciones dependían de cada integrante, y podían estar almacenadas en sus computadoras personales. El acceso a las mismas era disponible solamente al propietario de dicha colección, ya que era de carácter personal.

En síntesis, el juez e integrantes del juzgado debían leer, analizar y revisar cada uno de estos documentos de forma manual, en busca de información relevante. Esto conllevaba invertir demasiado tiempo en el análisis manual de la información, que podría utilizarse de una forma más eficiente en actividades más estratégicas. Además, se le adicionaba el hecho de que podían existir sentencias o casos omitidos, debido a factores como fatiga, cansancio y al gran volumen de información a consultar. En base a este proceso, se pensó en una solución que automatice la búsqueda, análisis y gestión de la jurisprudencia de la institución, para una ágil recuperación de la información, disponible como herramienta para la labor diaria del profesional.

4.2.2. Definición de requerimientos

De la información obtenida en el proceso de elicitación, se determinaron los siguientes requerimientos para el sistema:

Requerimientos Funcionales

1. Sesión de Usuario: solamente los usuarios registrados en el sistema (personal del juzgado) podrán utilizarlo. La sesión del sistema caducará si el usuario se encuentra inactivo durante un periodo determinado de tiempo, por lo que para utilizar nuevamente el sistema deberá loguearse nuevamente.
    - Login: Para el ingreso al sistema, serán requeridas las credenciales de acceso a través de una pantalla de inicio de sesión.
    - Logout: El usuario podrá cerrar sesión en el sistema, destruyéndose su sesión.
2. Perfiles de Usuario: el sistema debe contar con dos tipos de perfiles de usuarios:

- Administrador: es el perfil de usuario de mayor privilegios. Tendrá la capacidad y permisos para gestionar el sistema: carga de jurisprudencia, control y gestión de los usuarios y permisos.
- Personal: podrá ingresar al sistema para consultar la jurisprudencia pero no podrá alterar ni modificar la base de datos. Puede recibir permisos específicos para realizar algunas acciones sólo si el administrador del sistema se los otorga.

3. Carga de jurisprudencia: el sistema deberá ser capaz de permitir la carga de sentencias en formato *.docx* a la base de datos del sistema, a través de:

   - un formulario de carga que permita ingresar todos los datos pertinentes de la sentencia: título, número de causa, categorías, fecha de resolución, entre otras.
   - carga del archivo original de la sentencia en formato word (*.docx*). Se decidió que el sistema acepte únicamente este formato debido a que presenta mayor facilidad y menores inconvenientes para extraer el texto del archivo, a diferencia de formatos como PDF, dónde esta tarea resulta mucho más difícil debido a la flexibilidad que posee el formato.

     El principal problema es que PDF nunca se diseñó realmente como un formato de entrada de datos como *.docx*, sino que se diseñó como un formato de salida que brinda un control detallado sobre un documento resultante. En esencia, el formato PDF consiste en un flujo de instrucciones que describen cómo dibujar en una página. En particular, los datos de texto no se almacenan como párrafos, ni siquiera como palabras, sino como caracteres pintados en determinadas ubicaciones de la página. Como resultado, la mayor parte de la semántica del contenido

se pierde cuando un documento de texto se convierte a PDF; toda la estructura de texto implícita se convierte en una "sopa" casi amorfa de caracteres que "flotan" en las páginas.

4. Búsqueda de jurisprudencia: el sistema deberá permitir la búsqueda de sentencias a través del ingreso de términos, palabras o frases. Es decir, deberá retornar todos los resultados de las sentencias que contengan dichos términos. El usuario tendrá la posibilidad de ordenar los resultados por relevancia, fecha de resolución ascendente o descendente. El sistema debe ofrecer paginación para evitar la acumulación de muchos resultados en una sola página.

5. Búsqueda de sentencias filtrada por categorías: el sistema deberá permitir la búsqueda filtrada por diferentes categorías, como:

   - Materia de Derecho Penal.
   - Tipo de Resolución.
   - Instancia.
   - Características Especiales.

6. Búsqueda de sentencias filtrada por rango de fechas: el sistema deberá permitir la búsqueda filtrada por rango de fechas de resolución, en conjunto con los demás filtros que se deseen agregar.

7. Descarga de sentencia: el sistema deberá permitir la descarga del archivo original de una sentencia de interés.

8. Agregar a favoritos: un usuario podrá tener una colección de sentencias favoritas, pudiendo agregar o quitarlas según su interés.

9. Índice de sentencias: un usuario podrá acceder a una sección donde se encuentren un indice de las sentencias agrupadas en categorías.

10. Análisis Inteligente de una sentencia: un usuario ejecutar un análisis inteligente haciendo uso de un reconocedor de entidades nombradas (NER) que permita detectar aquellas entidades de índole legal más relevantes en el documento.

Requerimientos No Funcionales

A continuación se presentan los requerimientos no funcionales del sistema:

- Usabilidad: el sistema deberá presentar una interfaz de usuario intuitiva y fácil de utilizar por parte de un profesional del derecho. Adicionalmente debe poseer un diseño "responsive" a fin de garantizar la adecuada visualización en múltiples computadoras personales, dispositivos tablet y smartphones.

- Seguridad: el sistema debe proporcionar protección ante riesgos de seguridad mas importantes en aplicaciones web, como ataques por inyección SQL, protección contra XSS, protección contra ataques CSRF, buena administración de contraseñas en la base de datos a través de métodos de hashing robustos, entre otros.

- Disponibilidad: el sistema debe estar disponible para su uso en la mayoría del tiempo del horario laboral.

4.3. Diseño del producto

El diseño es una etapa crucial que crea una representación o modelo del software, pero, a diferencia de un modelo de requerimientos (que se centra en describir

los datos que se necesitan, la función y el comportamiento), el diseño proporciona detalles sobre arquitectura del software, estructuras de datos, interfaces y componentes que se necesitan para implementar el sistema [13]. Permite modelar el sistema o producto que se va a construir.

4.3.1. Casos de uso

Una vez definidos los requerimientos funcionales del sistema, se procedió a modelar los mismos a través de la definición de casos de uso, como se puede observar en la Fig. 4.3.

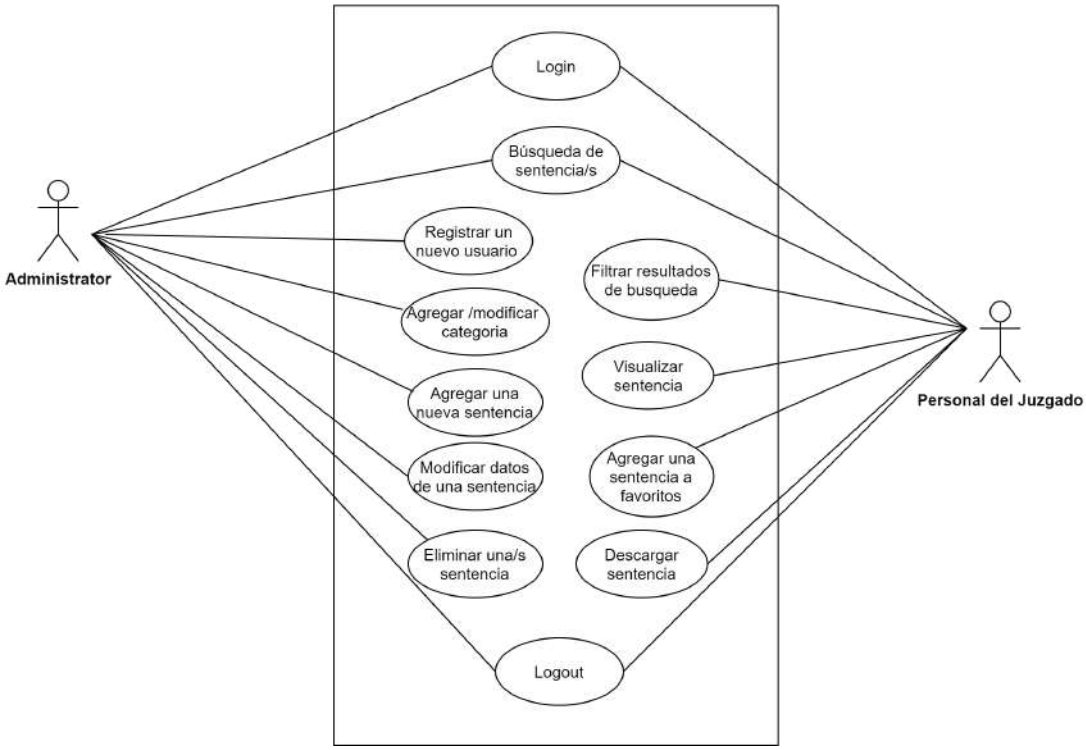


Figura 4.3: Casos de uso del sistema. *Fuente: Elaboración propia*

En el Apéndice C , se detallan los casos de usos anteriormente representados en la Fig. 4.3.

4.3.2. Arquitectura del Sistema

Dada las carácteristicas del proyecto, se optó por un sistema web que siguiera una arquitectura cliente-servidor, siguiendo el patrón Modelo-Vista-Template que proporciona Django. Como se puede observar en la Fig. 4.4, se pueden distinguir las distintas capas y módulos que conforman el sistema.

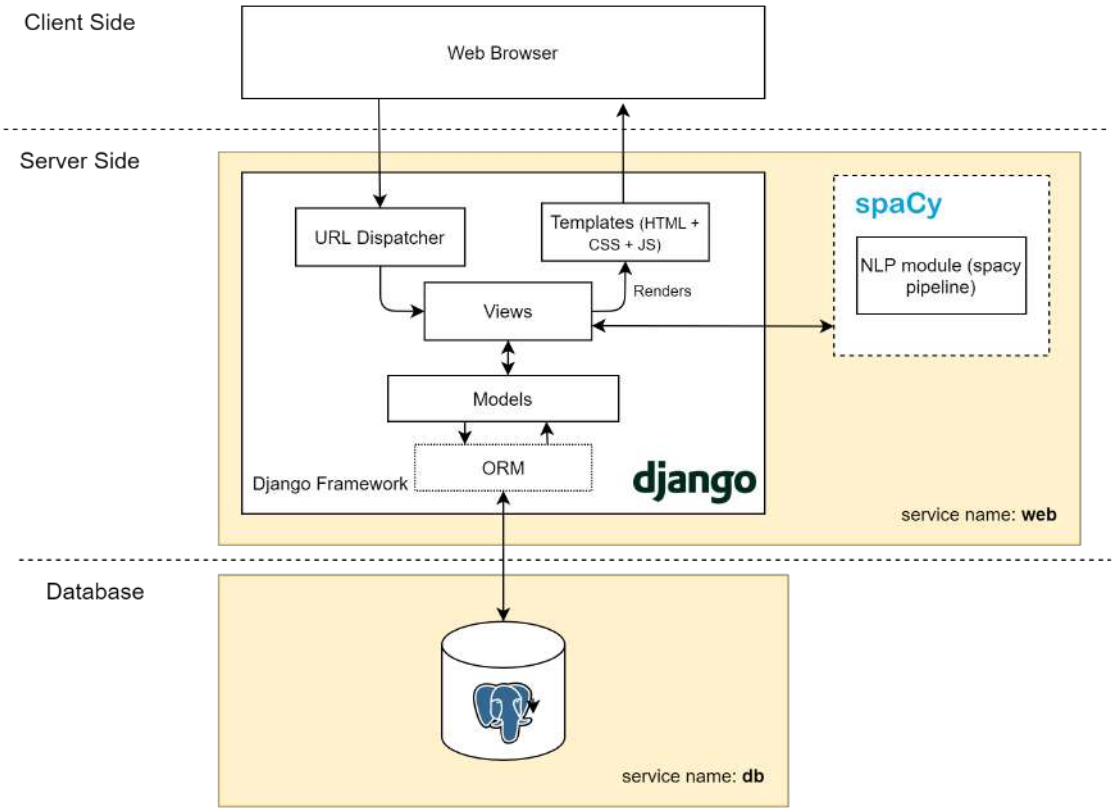


Figura 4.4: Arquitectura Web del sistema. *Fuente: Elaboración propia*

Para la capa de persistencia de datos, se utilizó el sistema de gestión de base de datos PostgreSQL. Dicho motor se comunica con el servidor backend a través del ORM que proporciona el framework Django, tanto para la realización de consultas

como para ejecutar directivas de creación y modificación de las estructuras de las tablas.

En el lado del servidor también fue necesario definir un módulo encargado de realizar la extracción de entidades nombradas, utilizando el pipeline que ofrece Spacy. Por el lado del cliente o FrontEnd, se utilizó HTML5, CSS, Bootstrap y JavaScript en conjunto con el sistema de renderizado y plantillas que proporciona el framework backend.

Patrón Modelo - Vista - Template

Django se basa en la arquitectura MVT, un patrón de diseño de software para desarrollar aplicaciones web. Este patrón es una variación del conocido Modelo-Vista-Controlador. La estructura MVT tiene las siguientes tres partes, las cuales se pueden observar en la Fig. 4.4:

- Modelo (*Model*): es la capa de acceso a la base de datos. Esta capa contiene toda la información sobre los datos: cómo acceder a estos, cómo validarlos, cómo es su estructura, cuál es el comportamiento que tienen, y las relaciones entre ellos. A lo largo del trabajo será recurrente verla a través de las clases incluidas en el archivo `models.py`.

- Vista (*View*): es la capa de la lógica de negocios. Esta capa contiene la lógica que accede al modelo y se comunica con la base de datos para recuperar datos que se transfieren a las templates para su visualización. Reciben las peticiones (GET/POST) del cliente enrutadas por el URL Dispatcher y las gestionan siguiendo las reglas del negocio. A lo largo del trabajo será recurrente verla a través de las clases y métodos controllers del archivo `views.py`.

- Template: es la capa de presentación, contiene las partes estáticas de la salida HTML deseada, así como una sintaxis especial que describe cómo se insertará el contenido dinámico. Esta parte es lo que es visible para el cliente, representa la interfaz de usuario.

Docker y Docker Compose

Para lograr una mejor facilidad tanto a la hora del desarrollo local como en un entorno de producción, se utilizó Docker y Docker Compose.

Un contenedor de Docker permite empaquetar un proyecto con todas sus dependencias únicamente necesarias en un solo binario y generar un entorno replicable, portable y estable para la ejecución de procesos. Por su parte, Docker Compose es una herramienta desarrollada para ayudar a definir y compartir aplicaciones de varios contenedores de Docker a partir de archivos YAML.

Los componentes que se observan en la Fig. 4.4 marcados como `service name: db` y `service name: web` corresponden a los servicios de servidor de base de datos de PostgreSQL y servidor web de Django respectivamente. Para crear un ambiente multi-contenedor con estos dos servicios (Postgres y Django), se definió el siguiente archivo `docker-compose.yml`:

```
version: "3.8"
services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    volumes:
```

```
    - ./postgres-data:/var/lib/postgresql/data
web:
  build: .
  command: python manage.py runserver 0.0.0.0:8000
  volumes:
    - .:/code
  ports:
    - "8000:8000"
  depends_on:
    - db
```

En este archivo YAML es donde se definen los servicios, redes, volúmenes y todo lo necesario para crear un ambiente con base en contenedores. Se encarga de buscar instrucciones para ejecutarlas, estas instrucciones contienen toda la configuración que será aplicada a cada contenedor iniciado por ese servicio.

A continuación se presenta una descripción simple de las principales instrucciones contenidas en el archivo docker-compose.yml anteriormente listado:

- `version`: los archivos docker-compose.yml son versionados y es muy importante indicar la versión a utilizar.
- `services`: indica los servicios a utilizar, podemos anidar n cantidad de servicios a esta instrucción, cada servicio puede tener un nombre específico. Para nuestro ejemplo utilizaremos dos servicios que representan un servidor web de django (`web`) y una base de datos PostgreSQL (`db`).
- `image`: permite especificar la imagen que se creará para instanciar el contenedor en el que estará montado el servicio. Es equivalente a utilizar el comando `docker tag`.

- `build`: se utiliza para indicar el contexto e indicar la ruta del archivo Dockerfile para construir el contenedor del servicio. En el caso de `db`, desde la imagen `postgres` de DockerHub, y en el caso de `web`, del archivo Dockerfile de la ruta correspondiente al directorio raíz del proyecto (./) .

- `depends_on`: se utiliza para establecer la dependencia y comunicación con otros servicios, en este caso el servicio `web` necesita comunicarse con el servicio `db` que se encuentra definido anteriormente.

- `ports`: se utiliza para exponer los puertos necesarios desde el contenedor.

- `environment`: permite establecer variables de entorno durante el ciclo de vida del contenedor.

4.3.3. Diseño de la Base de Datos

Una vez recopilados y analizados todos los requerimientos del sistema, se deben definir en tiempo de análisis/diseño las características de la base de datos. La notación más común para esto es el Diagrama de Entidad-Relación.

Diagrama Entidad-Relación

El ERD es una herramienta que permite realizar una abstracción o modelo de alguna situación de interés presente en el mundo real. Para esto, se modelan las cosas u objetos existentes en el dominio del sistema, sus características y sus relaciones [14]. Como podemos ver en la Fig. 4.5, se puede observar el diagrama elaborado que incluye las entidades principales del sistema, junto con sus relaciones y atributos.

Una de las principales entidades que representa la sentencia judicial en el sistema, es la denominada **Sentencia**. Una sentencia es la resolución judicial que

contiene la decisión del juez o el tribunal interviniente sobre el fondo de la cuestión que se le ha encargado juzgar. Dicha entidad posee un número identificador único como clave primaria y cuenta con varios atributos que conforman la información necesaria que debe poseer, como su título, fecha de resolución, número de causa, autor, cómo así también otros atributos necesarios para su correcta gestión en la base de datos.

Asimismo, la entidad **Sentencia** se relaciona con otras entidades. Estas asociaciones son necesarias no solo para visualizar la información de cada sentencia, sino también a la hora de realizar búsquedas filtradas por diferentes aspectos. En primer lugar, debe ser posible registrar en que **Instancia** fue dictada la sentencia. Por ejemplo, si se trata de una instancia de Juzgado de Garantías, Cámara Departamental, Provincial o Nacional. En segundo lugar, cada sentencia puede presentar una o varias **Características Especiales** que ayuden al profesional a identificar las características que hacen de esa sentencia un caso especial a la hora de consultar jurisprudencia. Por ejemplo, una sentencia puede tener características especiales como Violencia de Género, Estupefacientes, Apremios y Vejaciones, entre otras. En tercer lugar, cada **Sentencia** posee un **Autor**, registrando nombre y apellido.

También es necesario que el sistema tenga la capacidad de que un usuario pueda agregar a favoritos las sentencias de su interés. De aquí nace la relación muchos a muchos entre las entidades **Sentencia** y **Usuario**, denominada **Favoritos**, con cardinalidad N a M. Como parte del seguimiento de la actividad del usuario en el sistema, se definió la entidad **SearchHistory**. Dicha entidad es la encargada de registrar cada búsqueda realizada por un usuario del sistema (de allí su relación con **Usuario**), junto con su fecha y hora, a través del atributo timestamp.

Toda la información modelada a través de este conjunto de entidades, relaciones y atributos, debe mantenerse actualizada y consistente. Dicha labor será realizada por el administrador del sistema, por lo que no todos los usuarios pueden realizar cambios en la base de datos en términos de operaciones de escritura y no todos poseen los mismos permisos. Para esto, se hizo uso de las entidades y modelos del paquete `django.contrib.auth` que proporciona el framework Django. Como se puede ver, un **Usuario** puede pertenecer a uno o varios **Grupos**, y los **Permisos** pueden suministrarse a nivel grupo o a nivel usuario, a través del módulo de administración.

En cuanto a las restantes relaciones con las entidades **Tipo de resolución** y **Materia**, se explicarán con mayor detalle en la siguiente sección.

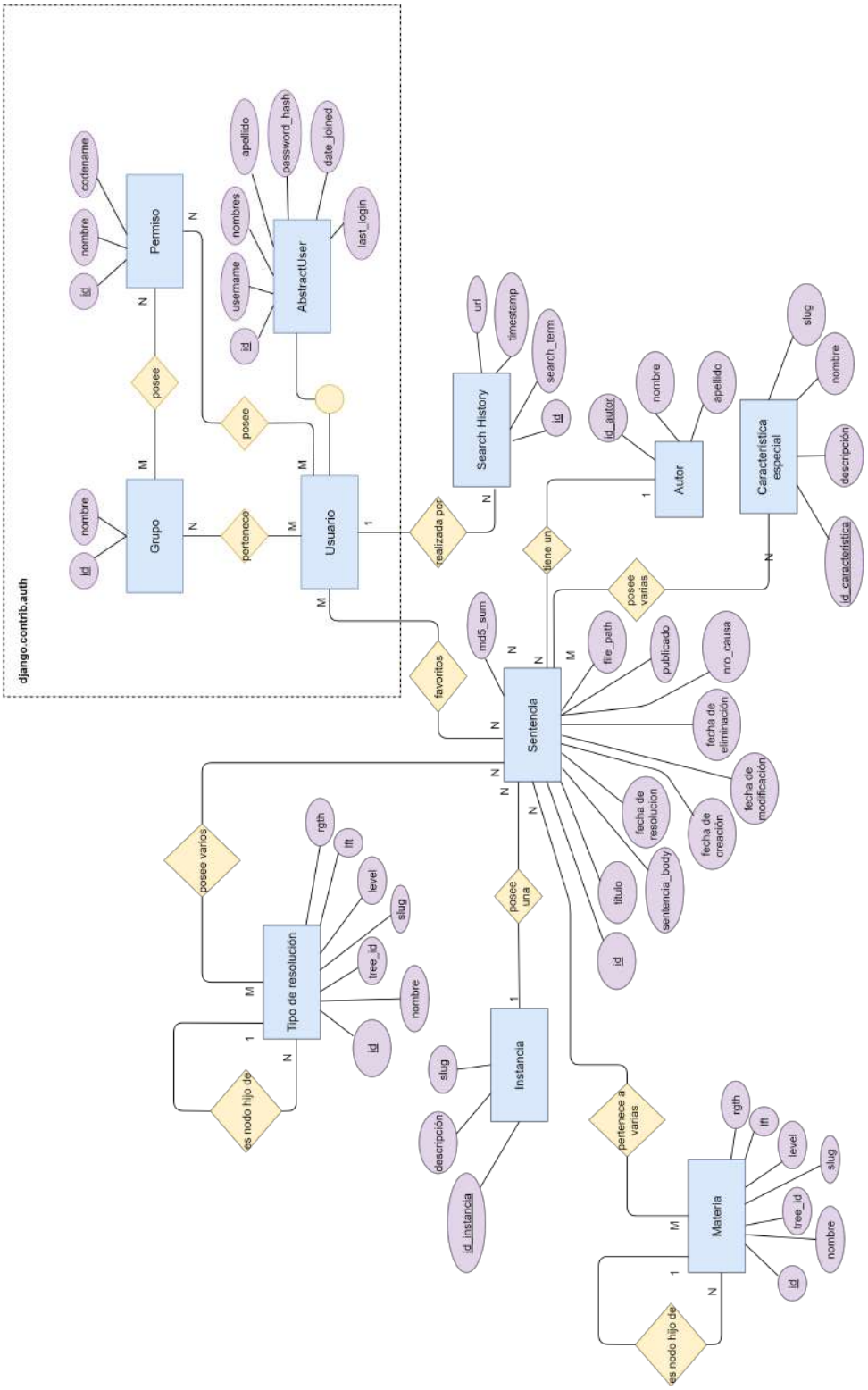


Figura 4.5: ERD. *Fuente: Elaboración propia*

Modified Preorder Tree Traversal

De la etapa de elicitación de requerimientos, se obtuvo información proveniente del juzgado sobre cómo se deberían clasificar las sentencias en base a distintos criterios. Dos de ellos, fueron la posibilidad de organizarlas por Tipo de Resolución y por Materia del Derecho Penal. A continuación se presentan las estructuras de las clasificaciones de las sentencias por estos dos criterios. Teniendo en cuenta el tipo de resolución, la clasificación provista fue la siguiente:

- **Medidas Cautelares**
- **Medidas de coerción**
  - Registros domiciliarios
  - Detención
  - Prisión Preventiva
  - Falta de Mérito
- **Resoluciones de finalización**
  - Elevaciones a juicio
    - Concedido
    - Denegado
  - Sobreseimiento
    - Materialidad
    - Autoría
  - Extinción
  - Prescripción
  - Juicio Abreviado
  - Suspensión del Proceso a Prueba
- **Cuestiones de competencia**

- Inhibitorias
  - Competencia negativa
  - Competencia positiva
- Declinatorias
  - Competencia negativa
  - Competencia positiva

Teniendo en cuenta la Materia del Derecho Penal, la clasificación provista fue la siguiente:

- **Procesal Penal**
  - Nulidades
  - Medidas de coerción
  - Medidas de investigación
    - Allanamientos
    - Intervenciones corporales
- **Penal Parte General**
  - Causales de justificación
  - Legítima Defensa
    - Supuestos de admisibilidad
    - Exceso en la LD
  - Cumplimiento del Deber
  - Supuestos de no punibilidad
- **Penal Parte Especial**
  - Delitos contra la propiedad
  - Supuestos agravados

- Estupefacientes

Como se puede observar, ambos criterios presentan una estructura jerárquica. Esto no es un dato menor ya se almacenará la información en una base de datos relacional. Los datos jerárquicos son una colección de datos en la que cada elemento tiene un padre único y cero o más hijos (con la excepción del elemento raíz, que no tiene padre). Los datos jerárquicos se pueden encontrar en una variedad de aplicaciones de bases de datos que se utilizan cotidianamente, incluidos foros y listas de correo, organigramas comerciales, categorías de administración de contenido y categorías/subcategorías de productos en tiendas *ecommerce*.

En el mundo de las base de datos relacionales, este caso es típicamente representado a través de una relación recursiva hacia misma tabla/entidad nodo. Este tipo de representación es conocida como el modelo de lista de adyacencia, o *Adjacency List Model*. En ella, cada elemento de la tabla contiene un puntero a su padre. El elemento superior o raíz es el único que posee un valor nulo para su padre. En el ORM de Django, podemos lograrlo utilizando `models.ForeignKey('self', ...)` para crear este tipo de relación recursiva.

Trabajar con el modelo de lista de adyacencia en SQL puro posee algunos inconvenientes. Antes de poder ver la ruta completa de una categoría (es decir, el camino de categorías desde la raíz hasta el destino) se debe conocer el nivel en el que reside. Además, se debe tener especial cuidado al eliminar nodos debido a la posibilidad de dejar huérfano a todo un subárbol en el proceso. Algunas de estas limitaciones se pueden abordar mediante el uso de código especial del lado del cliente o *stored procedures*. El mayor problema con este tipo de representación, es que no es escalable para arboles muy profundos. Siempre que tengamos que atravesar el árbol desde un nodo inicial dado, el motor de base de datos debe realizar una consulta por nivel de jerarquía. Se necesitará una operación *self-join*

por cada nivel en la jerarquía, y el rendimiento se degradará naturalmente con cada nivel agregado a medida que la operación de unión aumenta en complejidad. Cabe destacar que para dichas clasificaciones de sentencias, el usuario administrador podrá agregar o quitar subcategorías en su conveniencia, por lo que el árbol podría crecer en profundidad fácilmente.

Para resolver este problema, se decidió utilizar otro patrón muy conocido para almacenar estructuras jerárquicas en una base de datos relacional de forma eficiente, llamado *Modified Preorder Tree Traversal* o MPTT, por sus siglas en inglés. Este algoritmo utiliza una estructura similar a un árbol para modelar los datos, junto con un etiquetado intuitivo de los nodos asociados del árbol, lo que permite el desplazamiento basado en etiquetas. Por ejemplo, se posee la siguiente relación jerárquica entre las diferentes materias del Derecho Penal:

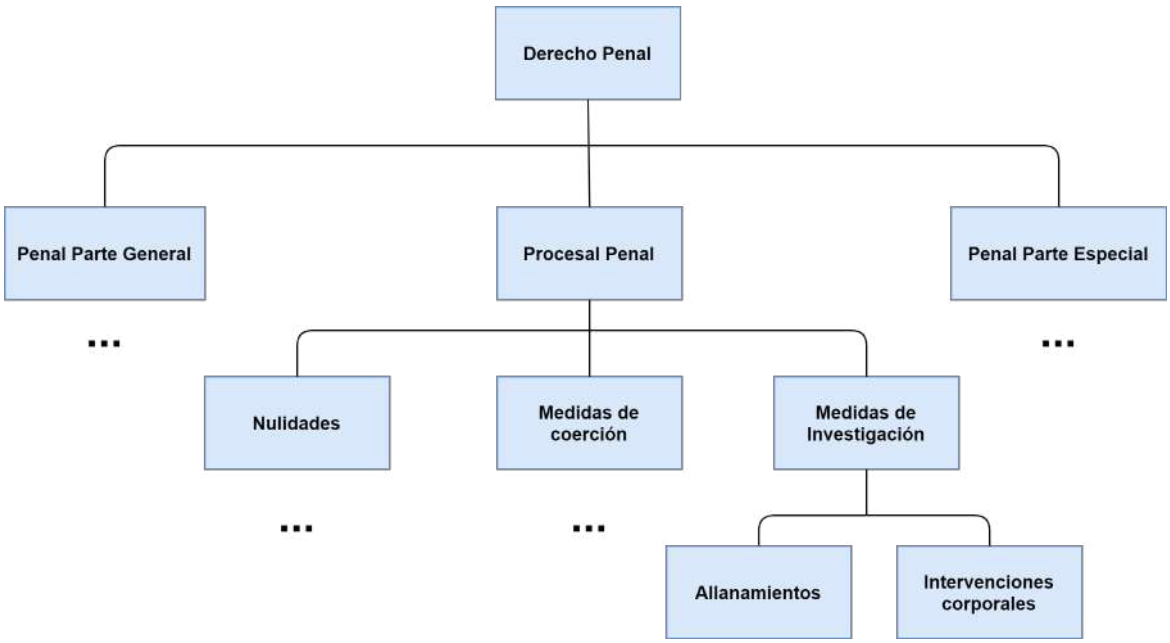


Figura 4.6: Estructura jerárquica de las materias del Derecho Penal. *Fuente: Elaboración propia*

Como se puede observar en la Fig. 4.6, solo una parte de la jerarquía parcial

es representada. El algoritmo MPTT aplica un esquema de etiquetas a los nodos, a través de los atributos lft y rgth, de la siguiente forma (siguiendo un recorrido de Pre-Orden):

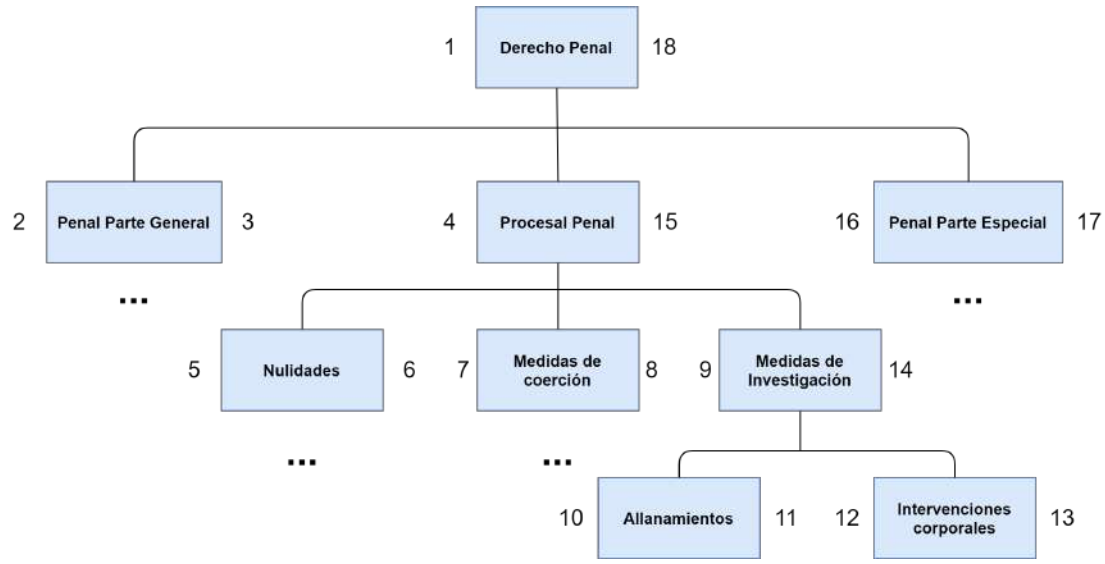


Figura 4.7: Estructura jerárquica de las materias del Derecho Penal - Atributos lft y rgth. *Fuente: Elaboración propia*

Como se observa en la Fig. 4.7, los números se asignan de manera que se pueda trazar una ruta alrededor del árbol, abarcando cada nodo. El camino comienza en la raíz y fluye hacia abajo a la izquierda.

El esquema de etiquetado comienza colocando un 1 a la izquierda del nodo raíz, *Derecho Penal* en este ejemplo. Luego, se desciende un nodo hacia la izquierda. En este nodo se incrementa la cuenta y se etiqueta un 2 a la izquierda del mismo (*Penal Parte General*). Este proceso continúa hasta el nodo secundario más bajo, llamado hoja. Luego, se etiqueta el lado derecho del nodo secundario con el siguiente incremento -3 en este caso- y se mueve lateralmente a través de los hermanos hacia el lado derecho etiquetando los lados izquierdo y derecho, incrementándose a medida que avanza. Dicho proceso recursivo se repite por cada

subárbol de la jerarquía, llegando nuevamente hasta la raíz con la cuenta final en el lado derecho del nodo (18 en este ejemplo).

Lo siguiente que se debe hacer es traducir este árbol anidado en una estructura de tabla plana. Esto se logra mediante la definición de dos columnas adicionales de valores "izquierda" y "derecha". Sin embargo, dado que izquierda y derecha son palabras clave reservadas en el lenguaje SQL, las implementaciones reales usan abreviaturas, como "lft" y "rgth".

A continuación se muestra una tabla de ejemplo de una posible implementación mínima de una tabla estructurada MPTT para las materias del Derecho Penal:

Tabla 4.1: Ejemplo de tabla MPTT para las materias del Derecho Penal.

id	nombre	parent_id	lft	rgth
1	Derecho Penal	-	1	18
2	Parte Penal General	1	2	3
3	Procesal Penal	1	4	15
4	Nulidades	3	5	6
5	Medidas de coerción	3	7	8
6	Medidas de investigación	3	9	14
7	Allanamientos	6	10	11
8	Intervenciones corporales	6	12	13
9	Penal Parte Especial	1	16	17

Ahora que los datos están organizados y anotados con los valores en las columnas "lft" y "rgt", se ha ganado más flexibilidad, control y eficiencia en la forma en que recuperamos los datos y recorremos la estructura. Usando la tabla 4.1, se puede enumerar subcategorias que forman parte de la materia **Procesal Penal**, utilizando la siguiente consulta SQL;

```
SELECT * FROM Materia WHERE lft > 4 and rgt < 15 ORDER BY lft;
```

Registrar las diferentes materias con las columnas “lft” y “rgt” de acuerdo con la estructura MPTT nos proporciona una forma mejorada de atravesar los datos y obtener información útil con menos interacciones con la base de datos y de forma más eficiente. El enfoque MPTT es beneficioso porque, con solo un par de campos adicionales, se puede determinar una estructura de árbol completa. Debido a esta economía, las operaciones de recuperación son más eficientes. En contraparte, las inserciones y los movimientos pueden ser más complejos.

Django-mptt

Para la implementación del patrón MPTT en las entidades Materia y Tipo de Resolución, se decidió utilizar la librería Django-MPTT, escrita en python y compatible con el framework Django y su ORM. La comunidad que usa y desarrolla el framework web Django ha producido el proyecto Django-MPTT [15] que amplía las funcionalidades básicas de Django e implementa MPTT para la creación de modelos. El proyecto Django-MPTT ofrece una serie de comodidades que hacen que la interacción con datos jerárquicos en la estructura MPTT sea muy conveniente al tiempo que se logran las eficiencias asociadas con la recuperación de datos MPTT, a través del ORM de Django.

Una de las clases que proporciona esta librería es `MPTTModel`. Con esta clase, podemos definir un modelo propio que herede los atributos y métodos de `MPTTModel`. Los campos que se añaden al modelo son:

- `lft` y `rgth`: descriptos en la sección anterior.
- `level`: un campo entero que indica el nivel de jerarquía de cada instancia.
- `parent_id`: un campo entero indicando el nodo padre de cada instancia.

Mientras que algunos de los métodos que se agregan, abstraén la implementación de los campos antes mencionados y son de gran utilidad para trabajar con dichas estructuras jerárquicas son:

- `get_ancestors(ascending=False, include_self=False)`
- `get_children()`
- `get_descendants(include_self=False)`
- `get_descendant_count()`
- `get_family()`
- `get_next_sibling()`
- `get_previous_sibling()`
- `get_root()`
- `get_siblings(include_self=False)`
- `insert_at(target, position='first-child', save=False)`
- `is_child_node()`
- `is_leaf_node()`
- `is_root_node()`
- `move_to(target, position='first-child')`

Otro elemento clave que podemos obtener de la librería es el tipo de campo `TreeForeignKey`, el cual se comporta esencialmente igual que el tradicional `ForeignKey` de los modelos de Django pero muestra las opciones de jerarquía de un árbol con anidación en los *forms*.

4.3.4. Diseño de la UI (Interfaz de Usuario)

La experiencia de usuario, también llamada *User Experience* o UX, se define como el conjunto de factores y elementos relacionados con el proceso de interacción de un usuario respecto a un producto o servicio. A menudo, este concepto

se aplica a la interacción con páginas web y aplicaciones móviles. Sin dudas es un factor clave a la hora de desarrollar un producto de software, porque son los usuarios finales quienes juzgarán si el producto es intuitivo, posee facilidad de uso y no genera confusión a la hora de utilizarlo. Si disponemos de un buen diseño, podremos definir de forma clara las fases de desarrollo del producto web. Para lograrlo, se llevaron algunas tareas a cabo, como:

- Realizar entrevistas para conocer a los tipos de usuarios finales.
- Analizar los requerimientos del usuario y sus necesidades.
- Analizar las interfaces y productos de la competencia, para observar su comportamiento e interacciones.
- *Wireframing* y prototipado de producto utilizando software de diseño.
- Validación del prototipado con el usuario final y *feedback* a través de reuniones virtuales.

Para el wireframing o prototipado de las diferentes pantallas, se utilizó el programa de diseño Figma, una aplicación para diseñar interfaces que se ejecuta en el navegador, pero también posee versiones en escritorio. Incluye herramientas de diseño, creación de prototipos y generación de código SVG, CSS, iOS y Android. A modo de ejemplo, se puede observar el prototipado realizado en una de las pantallas del sistema para realizar una búsqueda de sentencias por diferentes filtros en la Fig. 4.8.

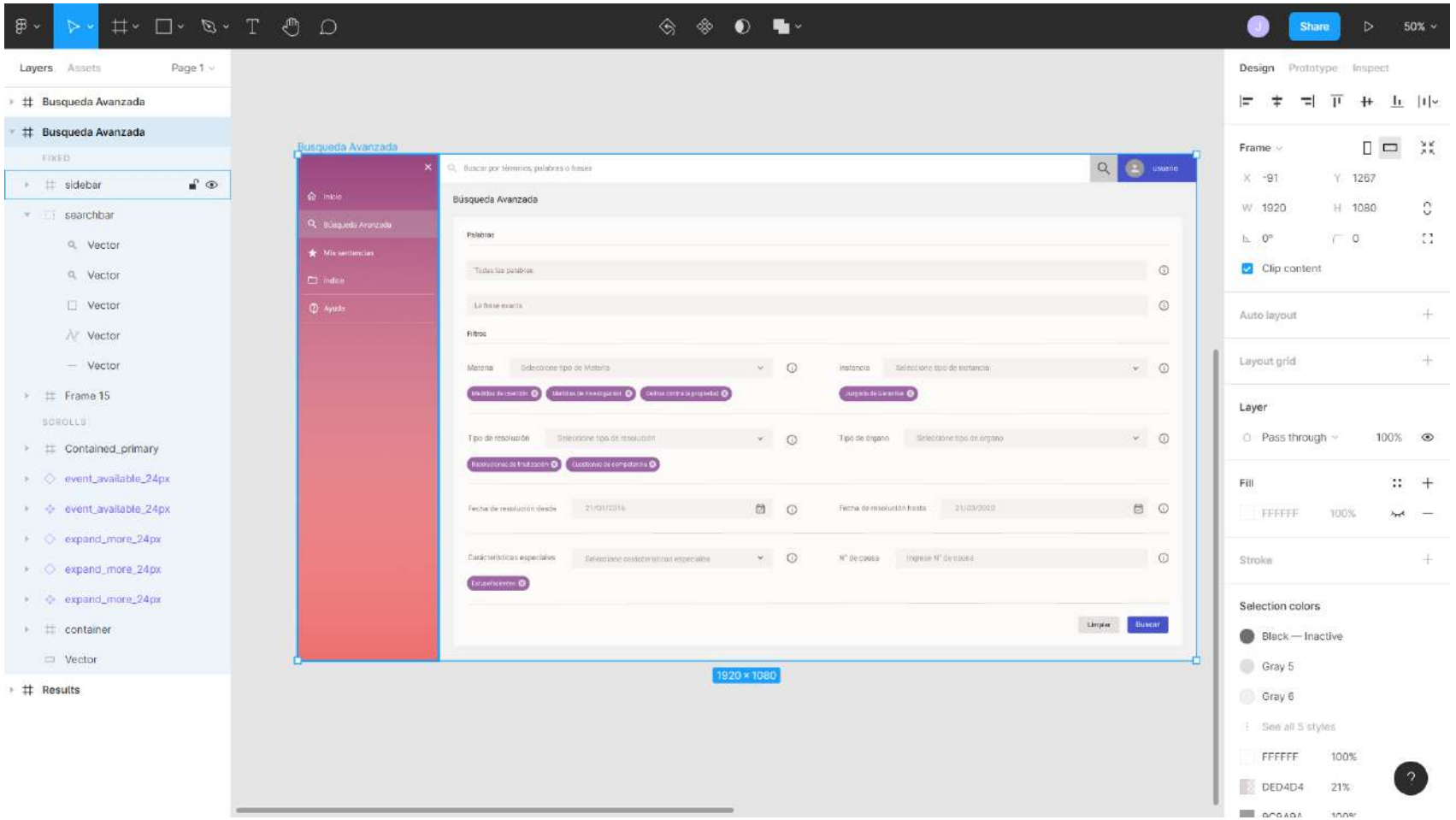


Figura 4.8: Prototipado en Figma. *Fuente: Elaboración propia*

En dicha pantalla, lo que se buscó fue tener una vista en la que el usuario pueda no solo buscar sentencias por frases o palabras, sino refinar su búsqueda por las distintas clasificaciones de las sentencias, fechas de resolución, entre otros. A la izquierda también se puede observar un menú desplegable que cumple la función de navegación por el sistema, estando presente al usuario sin importar donde se encuentre. La idea inicial fue mantener una consistencia visual a la hora de realizar búsquedas y no sobrecargar pequeñas secciones con muchos filtros, por lo que se dedicó una pantalla específicamente para esta funcionalidad.

4.4. Módulo Reconocedor de Entidades Nombradas

En esta sección se explicarán todas las actividades realizadas relacionadas a la creación de un modelo NER de índole legal, comenzando por explicar el proceso de trabajo que se adoptó, definición de entidades y fases previas y posteriores al entrenamiento del componente ner del *pipeline* de procesamiento de lenguaje natural.

4.4.1. Proceso de trabajo

Para el desarrollo de este módulo, primero se definió un proceso conformado por las etapas de Pre-Procesamiento, Procesamiento, y Carga, como se puede ver en la Fig. 4.9 , que se aplicarán sobre un corpus de documentos legales, en este caso sentencias de índole penal.

Se entenderá como “Pre-procesamiento” al conjunto de tareas que tienen relación con el proceso de extracción y limpieza de los fragmentos de textos procedentes de cada una de las fuentes; como “Procesamiento” al proceso de etiquetado de

los mismos y; como "Carga" a la adecuación de los fragmentos de dichos documentos al formato requerido para poder ser cargados como datos de entrenamiento al modelo.

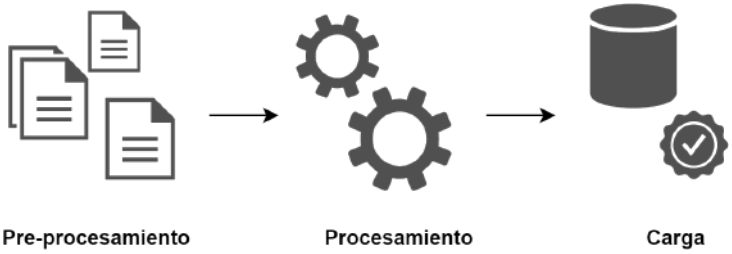


Figura 4.9: Diagrama del proceso de extracción de información para el módulo NER. *Fuente: Elaboración propia*

Para la fase de Pre-procesamiento se extraerán los textos de diferentes archivos en formato word (*.docx*) y libreOffice (*.odt*) correspondientes a sentencias judiciales facilitadas por el juzgado para su posterior uso como datos de entrenamiento. Para esto se convertirán dichos documentos a un formato *txt*.

La fase de Procesamiento constará de una etapa de procesado manual en donde se procederá a la anotación de los documentos, con la ayuda de la herramienta Doccano. Posteriormente se almacenaran texto y anotaciones en formato *JSONL*.

La ultima fase de Carga, previa al entrenamiento del modelo de spaCy, consistirá en adecuar texto y anotaciones en el formato requerido por spaCy para el correcto entrenamiento del módulo NER.

4.4.2. Definición de Entidades

La definición de entidades a ser reconocidas por el modelo fue diseñada teniendo como objetivo principal ser una ayuda adicional al análisis de una sentencia dictada por el juzgado. Es decir, debe permitirle al profesional identificar de forma

rápida aquellos conceptos principales que conforman el documento.

Adicionalmente, la librería spaCy ofrece dos enfoques al momento de entrenar un modelo reconocedor de entidades. Uno de ellos es actualizar un modelo pre-entrenado proporcionado por spaCy para adaptarlo a un contexto específico, en este caso, el legal. La idea es agregar clases arbitrarias al sistema de reconocimiento de entidades y actualizar el modelo con nuevos ejemplos para las nuevas categorías pero así también para las existentes, con el objetivo de que el modelo no "olvide". El restante enfoque es entrenar un componente reconocedor de entidades desde cero, sin un modelo pre-entrenado, proporcionando un dataset de entrenamiento con las entidades deseadas a detectar.

A continuación se presenta el esquema de las entidades sobre las que se entrenará un modelo NER:

- CITA: Citas a instrumentos legales, leyes, constituciones.
- ORG: Organizaciones legales, instituciones, compañías, departamentos.
- LOC: Lugares y localizaciones geográficas.
- PER: Personas con sus respectivos nombres y apellidos.

4.4.3. Etiquetado de datos de entrenamiento

Para el etiquetado de datos de entrenamiento, se utilizó la herramienta open source Doccano. La misma provee de funcionalidades de anotación para clasificaciones de texto y etiquetado de entidades, sumamente útiles para llevar a cabo tareas de análisis de sentimientos en textos, reconocimiento de entidades nombradas, text summarization, entre otras. Dicha herramienta es independiente del cualquier idioma, por lo que se pueden crear datasets de entrenamiento para

cualquier lenguaje natural. Posee una interfaz web por la que se pueden crear proyectos colaborativos entre varios usuarios, con distintos roles y privilegios sobre un dataset.

El proceso de etiquetado consistió inicialmente de definir un esquema de entidades o labels, con sus respectivos identificadores y colores para diferenciar las entidades marcadas en el texto. Luego, se importó cada archivo de la sentencia a etiquetar en formato *.txt* pero respetando un formato mandatario de Doccano, para facilitar el proceso de ingesta. El mismo establece que cada línea del archivo corresponde a un ejemplo de entrenamiento a etiquetar. En este caso, cada párrafo de la sentencia debía corresponder a una linea del archivo.

Una vez cargado y procesado el archivo, Doccano detecta cada ejemplo para poder etiquetar, permitiendo marcar con el cursor o atajos del teclado las entidades que queremos que el modelo aprenda, como se puede observar en la Fig. 4.10, 4.11 y 4.12. La ventaja de poder contar con una herramienta visual diseñada específicamente para esta tarea hizo que se pudiera generar un número considerable de ejemplos en poco tiempo. De no contar con esta implementación, sin dudas hubiera sido una tarea mucho más tediosa y demandante de tiempo.

Al finalizar el etiquetado, la herramienta provee de distintos formatos de salida para almacenar el dataset de ejemplos realizado. Entre las opciones se encuentran los formatos `JSONL` y `JSONL` con los nombres de las etiquetas. Éste último fue elegido para incorporar al pipeline de entrenamiento de spaCy. El formato JSONL posee la siguiente forma:

```
{"id": 2, "text": "Peter Blackburn", "labels": [[0, 15, "PERSON"]]}
{"id": 3, "text": "President Obama", "labels": [[10, 15, "PERSON"]]}
```

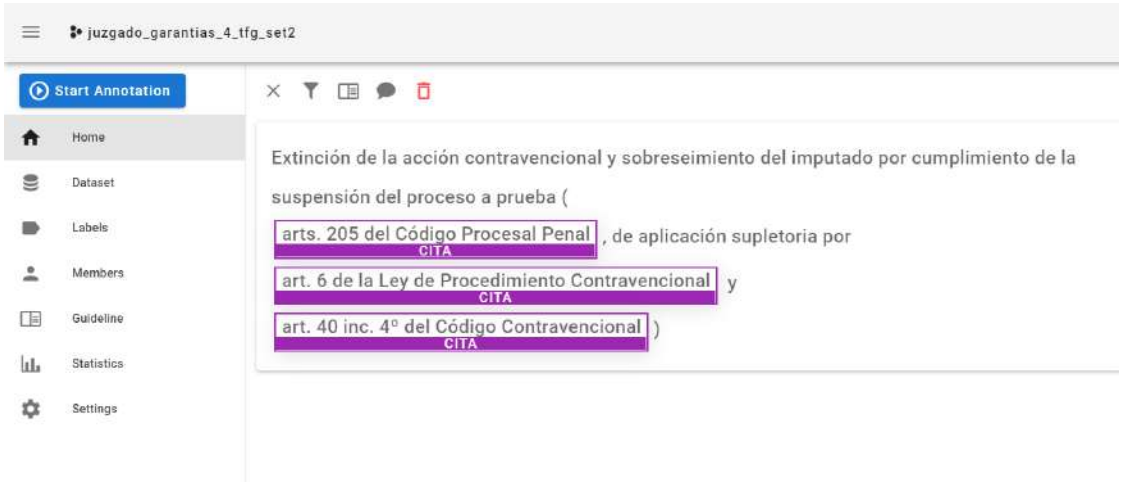


Figura 4.10: Proceso de etiquetado de ejemplos de entrenamiento a través de Doccano. *Fuente: Elaboración propia*

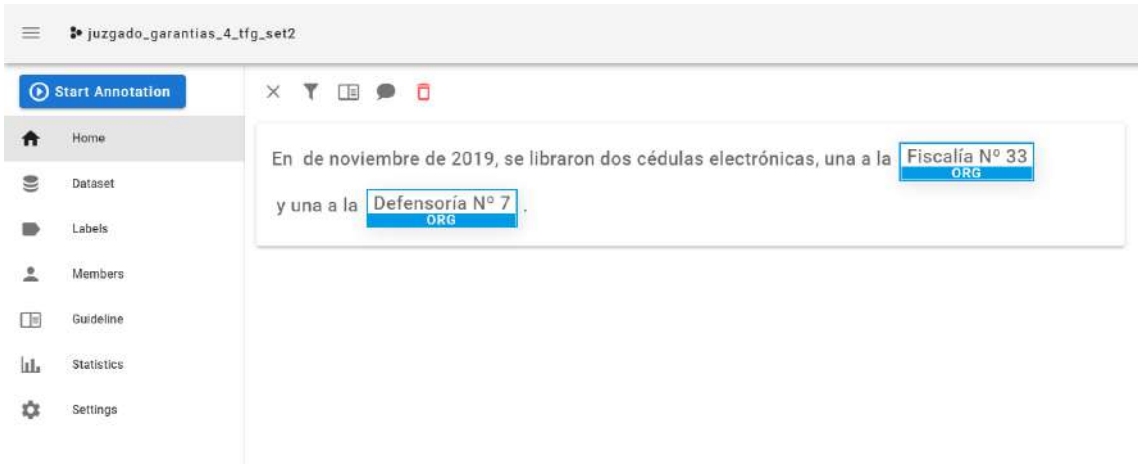


Figura 4.11: Proceso de etiquetado de ejemplos de entrenamiento a través de Doccano. *Fuente: Elaboración propia*

Figura 4.12: Proceso de etiquetado de ejemplos de entrenamiento a través de Doccano. *Fuente: Elaboración propia*

4.4.4. Datos de entrenamiento y validación

Existen tres tipos de datos implicados en la inicialización y desarrollo de modelos de inteligencia artificial: datos de entrenamiento, de validación y de evaluación (o test).

Se conoce como datos de entrenamiento a la muestra de datos empleada para ajustar el modelo. En la mayoría de casos esta muestra representa un porcentaje grande del total de los datos disponibles (en torno a un 70-80 % de los datos contenidos en el dataset).

El modelo, a lo largo de este entrenamiento irá variando los "pesos" y el "sesgo" ("weight" y "bias") para cumplir con garantías con la función la cual le ha sido encomendado, es decir, para comportarse de la manera adecuada según el caso de uso.

El conjunto de datos de evaluación o "test" se utiliza para dar una estimación imparcial de la habilidad del modelo afinado final al comparar o seleccionar entre los modelos finales (entrenados). Por lo tanto, el modelo se evalúa en la muestra retenida para dar una estimación no sesgada de la habilidad del modelo, ya que

de no ser así el modelo ya habría aprendido de estos datos previamente.

El conjunto de datos de validación, por su parte, desempeña un papel en otras formas de preparación del modelo, como la selección de características e híper-parámetros necesarios para hacer del entrenamiento un proceso efectivo.

Para el actual proyecto, se armaron 3 conjuntos de datos de entrenamiento proveniente de distintas fuentes. Por un lado, se contó con acceso a sentencias judiciales anonimizadas del Juzgado de Primera Instancia Contravencional y de Faltas N° 10 de la ciudad de Buenos Aires, gracias a la participación del InfoLab en un Hackatón que proveía el acceso a dichas sentencias. Por otro lado, fueron suministradas un conjunto de sentencias judiciales provenientes del Juzgado de Garantías N°4 de Mar del Plata.

Con la herramienta Doccano, se procedió a realizar el etiquetado como se describió anteriormente. Para dividir mejor la carga de trabajo en el etiquetado, y teniendo en cuenta las distintas fuentes, se armaron 3 conjuntos de datasets. En los primeros dos, conformados por sentencias del Juzgado N° 10 de Buenos Aires, fueron etiquetados 500 ejemplos cada uno. En el tercer dataset, constituido por sentencias del Juzgado de Garantías N° 4 de Mar del Plata, fueron etiquetados 362 ejemplos de entrenamiento. Del corpus de archivos de sentencias que fue posible reunir, se reservaron archivos para la posterior evaluación del modelo.

A continuación se presenta la distribución de etiquetas en cada dataset de ejemplos:

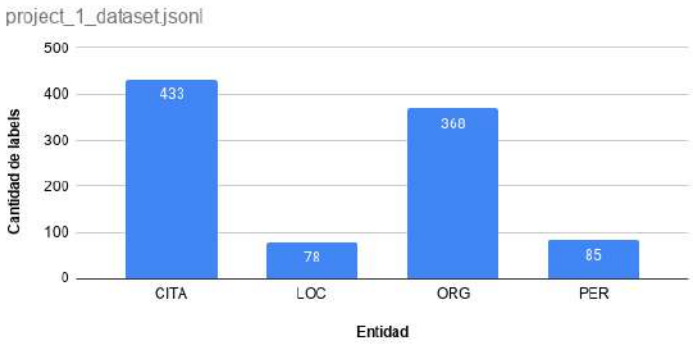


Figura 4.13: Distribución de labels en dataset N° 1. *Fuente: Elaboración propia*

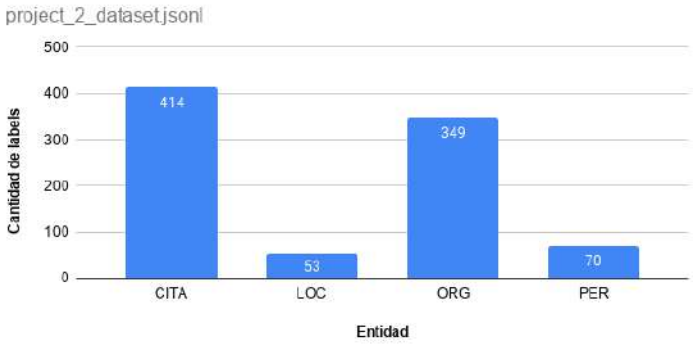


Figura 4.14: Distribución de labels en dataset N° 2. *Fuente: Elaboración propia*

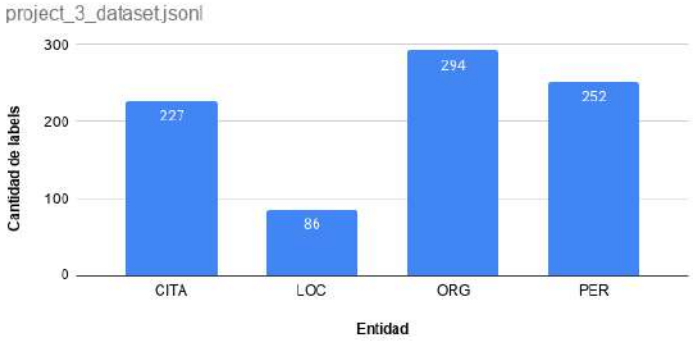


Figura 4.15: Distribución de labels en dataset N° 3. *Fuente: Elaboración propia*

Las entidades que mas predominan en los conjuntos etiquetados son CITA (Citas a instrumentos legales, leyes, doctrinas, entre otros) y ORG (organizaciones, instituciones públicas y privadas, entre otros). Esto es importante tenerlo en cuenta a la hora de evaluar los resultados del entrenamiento y la capacidad del modelo de detectar a priori una cantidad mayor de este tipo de entidades que las restantes. Asimismo, como se puede observar en las Fig. 4.13 y 4.14, al tratarse de ejemplos de sentencias anonimizadas, la cantidad de labels de la entidad PER (correspondiente a referencias de personas), es baja. Esto no ocurre en el dataset nro. 3, como se puede ver en la Fig. 4.15, ya que este conjunto de datos corresponden a sentencias sin anonimizar.

Para realizar el entrenamiento, se decidió combinar estos 3 conjuntos de datos y conformar un dataset de entrenamiento de 1163 ejemplos y un dataset de validación de 200 ejemplos.

4.4.5. Entrenamiento del modelo

Como se mencionó en la sección 3.3, se utilizó la librería spaCy. Los modelos de spaCy se basan en leyes estadísticas, es decir, que cada "decisión"que toman, como por ejemplo, qué etiqueta (sintáctica) asignar a qué palabra, o si una palabra representa una entidad nominal, es una predicción. Esta predicción se basa en los ejemplos que el modelo ha reconocido durante el entrenamiento. Para entrenar un modelo, primero necesita datos de entrenamiento; ejemplos de texto, y las etiquetas que quiere que el modelo prediga.

Como se puede ver en la Fig. 4.16, el entrenamiento es un proceso iterativo en el que las predicciones del modelo se comparan con las anotaciones de referencia (ejemplos de entrenamiento) para estimar el gradiente de pérdida. El gradiente

de pérdida se usa luego para calcular el gradiente de los pesos mediante retropropagación. Los gradientes indican cómo se deben cambiar los valores de peso para que las predicciones del modelo se vuelvan más similares a las etiquetas de referencia a lo largo del tiempo. Cuanto mayor sea la diferencia entre la predicción y el ejemplo de referencia, más significativo será el gradiente y las actualizaciones del modelo.

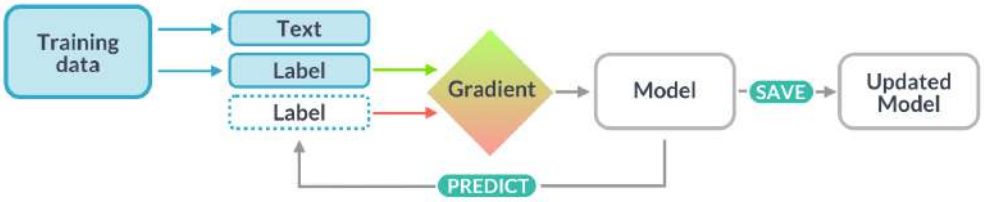


Figura 4.16: Diagrama de flujo de entrenamiento de un modelo base de spaCy. *Fuente: spaCy*

Al entrenar un modelo, el objetivo no es únicamente que pueda memorizar los ejemplos suministrados en el entrenamiento, sino que también que pueda generar una teoría o establecer patrones que sirvan para generalizar y aplicar lo aprendido en datos nuevos o no vistos por el modelo. Es por eso que los datos de entrenamiento siempre deben ser representativos de los datos que queremos procesar. Por ejemplo, un modelo entrenado en textos de novelas románticas probablemente tendrá un mal desempeño en el texto legal. Esto también significa que para saber cómo se está desempeñando el modelo y si está aprendiendo las cosas correctas, no sólo necesita datos de entrenamiento, también necesitará datos de evaluación . Si solo se prueba el modelo con los datos en los que se entrenó, no será posible determinar qué tan bien se está generalizando.

La forma recomendada por spaCy a partir de su ultima versión 3.1 [5] a la fecha (utilizada para este proyecto) es realizar el entrenamiento a través del comando

provisto `train`. Como requisito es necesario un único archivo de configuración `config.cfg` que incluya todas las configuraciones e hiperparámetros para los pipelines del modelo, como por ejemplo idioma, componentes del pipeline, hardware a utilizar (CPU o GPU), entre otros.

En una primera instancia, se puede generar un archivo de configuración base `base_config.cfg`, con los parámetros más importantes y luego llenar la configuración restante recomendada a través del comando `init fill-config`. Las configuraciones de entrenamiento siempre deben estar completas y sin valores predeterminados ocultos, para que sus experimentos sean reproducibles.

Internamente, la configuración es parseada a un diccionario. Está dividido en secciones y subsecciones, indicado por la notación de corchetes y puntos. Las subsecciones pueden definir valores, como un diccionario, o usar la @sintaxis para referirse a funciones registradas. Esto permite que la configuración no solo defina configuraciones estáticas, sino que también construya objetos como arquitecturas, programaciones, optimizadores o cualquier otro componente personalizado. En la tabla 4.2, se pueden observar las principales secciones de nivel superior de un archivo de configuración.

Asimismo es posible también sobreescribir algunos parámetros de configuración a través de la linea de comandos. Para el entrenamiento de este modelo, se decidió partir de un modelo en blanco que esté compuesto por un reconocedor de entidades nombradas (ner) y tok2vec para el lenguaje español. Previo a ejecutar el entrenamiento fue necesario desarrollar un script que convierta el dataset de ejemplos de entrenamiento de formato `.jsonl` a formato `.spacy`.

Tabla 4.2: Parámetros de configuración para el modelo

Sección	Descripción
`nlp`	Definición del objeto nlp, su tokenizador y los nombres de los componentes del pipeline de procesamiento.
`components`	Definiciones de los componentes del pipeline y sus modelos.
`paths`	Rutas a datos y otros recursos. Se reutiliza en la configuración como variables, por ejemplo ${paths.train}, y se puede sobrescribir en la CLI.
`system`	Configuraciones relacionadas con el sistema y el hardware.
`training`	Ajustes y controles del proceso de formación y evaluación.
`pretraining`	Configuraciones y controles opcionales para el pre-entrenamiento del modelo de lenguaje
`initialize`	Los recursos de datos y los argumentos pasados a los componentes cuando nlp.initialize se llama antes del entrenamiento

4.4.6. Resultados del entrenamiento

Una vez terminada la fase de entrenamiento, es el momento de evaluar el rendimiento del modelo. Para esto se hará uso del comando `evaluate` a la cual se le pasarán dos argumentos, por un lado el modelo entrenado y, por el otro, el dataset de validación. Estos últimos, también conocidos por el nombre de Gold Standards, servirán para evaluar como de bien fueron predichas las entidades en los textos de prueba.

Métricas de evaluación

Los sistemas de NLP se evalúan típicamente con respecto a su desempeño en el conjunto de pruebas de la tarea específica. Para la evaluación de los resultados de entrenamiento de un módulo NER se emplean parámetros e indicadores cuantitativos. Para ello se utilizan métricas conocidas y efectivas como la Precisión, Cobertura o Recall y F-measure o F1.

En el caso de clasificación binaria, la precisión es la medida de evaluación

común, que se define de la siguiente manera:

$$Prec = \frac{Tp + Tn}{Tp + Tn + Fp + Fn} \tag{4.1}$$

donde Tp, Tn, Fp y Fn son el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos respectivamente. Intuitivamente, el número de predicciones verdaderas se divide por el número de todas las predicciones. Para la clasificación de clases múltiples, se utiliza la puntuación F1, que es la media armónica de Precisión y Recall:

- Precisión: mide la capacidad del sistema de identificar y clasificar correctamente las entidades presentes en el texto.

$$Prec = \frac{Tp}{Tp + Fp} \tag{4.2}$$

- Recall (o exhaustividad): esta métrica mide la capacidad del algoritmo de detectar las entidades que están presentes en el texto.

$$Rec = \frac{Tp}{Tp + Fn} \tag{4.3}$$

- Fórmula F1: es una métrica que se utiliza para combinar en un solo número los resultados de las dos fórmulas anteriores.

$$F1 = 2 * \frac{P * R}{P + R} \tag{4.4}$$

En resumen, las principales métricas que retorna el entrenamiento y evaluación a partir de la version 3.1 de la librería [5] son:

Nombre	Descripción
Loss	La pérdida de entrenamiento que representa la cantidad de trabajo que le queda al optimizador. Debería disminuir, pero generalmente no a 0.
Precision (P)	Porcentaje de anotaciones previstas que fueron correctas. Debería aumentar.
Recall (R)	Porcentaje de anotaciones de referencia recuperadas. Debería aumentar.
F-Score (F)	Media armónica de Precisión (P) y Recall (R). Debería aumentar.
Speed	Velocidad de predicción en palabras por segundo (WPS). Debería permanecer estable.

Para realizar el entrenamiento se ejecutó el comando `train`, indicando el archivo de configuración, y los respectivos *paths* de salida y entrada para los dataset de entrenamiento y validación. Luego, para evaluar el modelo generado, se utilizó el comando `evaluate`. Ambos comandos retornan las puntuaciones de precisión, `recall` y F1:

```
python -m spacy train config.cfg --output ./output --paths.train
    ./train_dataset_1.spacy --paths.dev ./val_dataset_1.spacy
python -m spacy evaluate ./output/model-best ./val_dataset_1.spacy
```

En el siguiente cuadro se pueden ver los resultados obtenidos por spaCy en el reconocedor de entidades nombradas entrenado.

Tabla 4.3: Resultados en Precisión, Recall y F1 por entidad para el modelo de NER entrenado en spaCy.

Entidad	Precisión (P)	Recall (R)	F1
ORG	89.78	88.32	89.04
CITA	93.97	93.53	93.75
LOC	92.00	88.46	90.20
PER	95.29	95.29	95.29

Se puede observar un buen desempeño en las métricas de evaluación pese a no contar con un dataset muy extenso. Como era de esperar, una de las entidades que

mayor puntaje obtuvo fue `CITA`, debido a que fue la que mas ejemplos etiquetados tuvo. Esto puede sugerir que durante el entrenamiento, aquellas entidades que más hayan sido anotadas en la fase de procesamiento de los datos, más fácilmente serán reconocidas en la fase de evaluación y reconocimiento de entidades. Sin embargo, a mayor variabilidad en número de ejemplos, puede dificultarse la generalización de patrones y conducir a falsos positivos.

4.5. Implementación del producto

En la siguiente sección se profundizará sobre la implementación de las funcionalidades principales del producto, cuáles fueron los enfoques tomados y las herramientas utilizadas para materializar el diseño realizado en código.

4.5.1. Búsquedas Full Text Search sobre los documentos

Para cumplir con el requerimiento de búsqueda de sentencias a través de frases y palabras, se requirió una previa investigación enfocada en las funcionalidades típicas de un motor de búsqueda (indexación, full text search, faceted search, ranking, entre otras) y en las herramientas que se ofrecen en la actualidad para implementar este tipo de funcionalidad. Fue clave la elección de Postgres como motor de base de datos, debido a la funcionalidad full text search que provee a partir de la versión 8.3. A continuación se presentaran los conceptos principales utilizados en la implementación como así también su integración en el sistema web Django.

Postgres Full Text Search

La funcionabilidad de Full Text Search ha estado disponible a partir de la version 8.3 de Postgres (año 2008) y puede ser utilizada para buscar documentos

en base a la semántica y conocimiento del lenguaje en cuestión en lugar de una simple coincidencia de cadenas de texto, como lo realiza la conocida opción de búsqueda en bases de datos relacionales `LIKE` seguido del operador `%`.

Aunque `LIKE` es compatible con la utilización de índices para acelerar las consultas, no funciona eficientemente cuando el operador `%` es utilizado en el lado izquierdo del término de búsqueda. Esto significa que el planificador de consultas del motor tiene que ejecutar un escaneo secuencial en todos los registros cuando se utiliza este tipo de operadores *wildcard*, impactando negativamente en la performance de la consulta, como se ve en el ejemplo:

```
SELECT titulo FROM pelicula WHERE descripcion LIKE '\%brillante';
```

Esta situación puede degradarse aún más si necesitamos buscar por términos de texto en múltiples columnas del registro, ya que cada columna debe consultarse por separado usando `LIKE`, como se puede observar en la Fig. 4.17.

Debido a que `LIKE` y otros métodos simples de búsqueda carecen de soporte para idiomas y no pueden manejar variaciones de palabras o métodos de ranking, la funcionalidad Full Text Search (FTS) de Postgres es generalmente una mejor opción cuando se implementa búsquedas directamente en la base de datos. Como FTS posee soporte para varios idiomas, las búsquedas podrán coincidir no sólo con los términos de búsqueda sino también con todas las instancias de la palabra, su plural y los distintos tiempos verbales que posee.

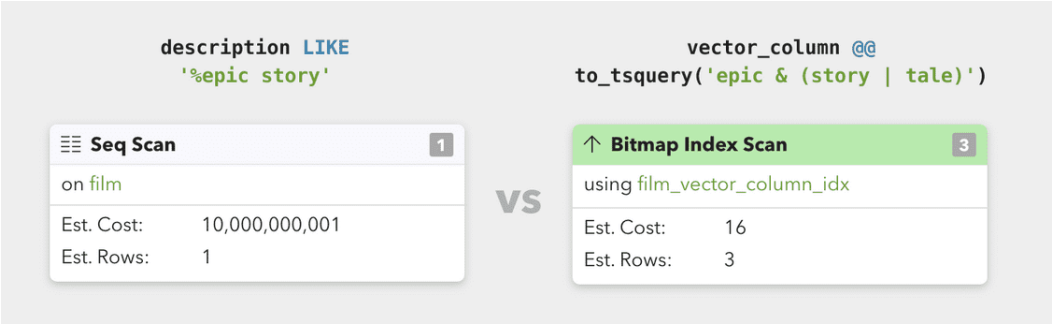


Figura 4.17: Comparación de Secuencial Scan para LIKE vs Index Scan para FTS. *Fuente: pganalyze.com, Efficient Postgres Full Text Search in Django*

PostgreSQL proporciona varias funciones nativas para la búsqueda de texto completo sobre un documento. Un documento es un concepto general utilizado en el campo del Full Text Search y básicamente, se entiende como la unidad donde se realiza la búsqueda. En una base de datos, puede ser un campo en una tabla, o la combinación de múltiples campos en una o diferentes tablas. Existen algunos conceptos claves que se deben utilizar para implementar este tipo de búsqueda [16]. A continuación se explicarán los principales.

Tipo de dato tsvector

Postgres preprocesa los documentos para su posterior búsqueda. Antes de que se pueda buscar en un texto o documento con FTS, se debe convertirlo a un tipo de dato conocido como `tsvector`.

Para convertir texto sin formato a `tsvector`, se utiliza la función `to_tsvector`. Esta función es clave en dicho pre-procesamiento: parsea el documento en tokens y reduce el texto original convirtiendo el conjunto de tokens en lexemas. Los lexemas son muy importantes ya que permiten mejorar la búsqueda con palabras relacionadas y aportan el significado básico de la palabra. Por ejemplo el lexema *histori-* puede extraerse como raíz de muchas palabras como *historia*, *historiador*

e *histórico*. Asimismo, palabras que no aportan valor como artículos, conjunciones, preposiciones y adverbios conocidas como Stop Words, son eliminadas.

En conclusión, la función `to_tsvector` retorna un conjunto ordenado de lexemas junto con los datos de sus posiciones en el texto. Dicha salida es dependiente del lenguaje natural, por lo que se debe especificar en que lenguaje se desea procesar el texto al momento de utilizar la función.

Tipo de dato tsquery

Los sistemas de búsqueda de texto tienen dos componentes principales: el texto en el que se busca y la palabra clave que se busca. En el caso de FTS, ambos componentes tienen que estar "vectorizados". De forma similar a la conversión de texto plano al tipo de dato `tsvector`, se deben convertir el/los términos de búsqueda al tipo de dato `tsquery`. Para esto, Postgres ofrece tres funciones: `to_tsquery`, `plainto_tsquery` y `phraseto_tsquery`.

Los términos de búsqueda se pueden combinar con los operadores lógicos, y se pueden usar paréntesis para agrupar operadores y determinar su orden. También pueden asignarse pesos a cada término para ponderar la consulta. La función `to_tsquery` normaliza cada token en un lexema usando la configuración especificada o predeterminada del lenguaje y descarta cualquier token que sea *stop word* de acuerdo con la configuración.

Búsqueda

Una vez que se tiene un texto convertido al tipo `tsvector` y un `tsquery` obtenido de los términos de búsqueda, se puede realizar una búsqueda FTS a través del operador `@@`.

Por ejemplo, si se posee una tabla llamada `pelicula` que contiene los títulos de películas junto con sus respectivas descripciones, se podría utilizar FTS para

encontrar todas las películas cuya descripción contenga la palabra *épico* y alguna de las palabras *cuento* o *historia*, a través de la siguiente consulta:

```
SELECT titulo, descripcion FROM pelicula
WHERE to_tsvector(descripcion) @@ to_tsquery('epico & (cuento |
    historia)')
LIMIT 10;
```

Postgres FTS en Django

El módulo `django.contrib.postgres` contiene todo el soporte para FTS en Django, a partir de la versión 1.10 [17]. A lo largo de la implementación, se utilizaron diferentes clases que provee el paquete como `SearchVector`, `SearchQuery`, `SearchRank` y `SearchHeadline` junto con el ORM de Django.

La clase `SearchVector` es la abstracción de la función `to_tsvector` de Postgres, que como se explicó anteriormente, es la responsable de retornar un vector semántico donde cada palabra es convertida a un lexema (unidad de significado léxico) con un puntero a la posición en el texto.

La clase `SearchQuery` es la abstracción de `to_tsquery` y acepta un conjunto de palabras para buscar en el vector normalizado creado utilizando `to_tsvector`. También se puede definir el tipo de búsqueda a través del parámetro `search_type`, teniendo como posibles opciones `search_type='plain'` (los términos se tratan como palabras clave independientes), `search_type='phrase'` (los términos se tratan como una sola frase), `search_type='raw'` (una consulta de búsqueda formateada con términos y operadores lógicos) y `search_type='websearch'` (una consulta de búsqueda con formato, similar a la que utilizan los motores de búsqueda web).

La clase `SearchRank` permite ordenar los resultados por algún tipo de relevancia. PostgreSQL proporciona una función de clasificación que tiene en cuenta la frecuencia con la que aparecen los términos de la consulta en el documento, la proximidad de los términos en el documento y la importancia de la parte del documento donde aparecen. Cuanto mejor sea la coincidencia, mayor será el valor del rango.

La clase `SearchHeadline` permite retornar los resultados de la búsqueda con los términos o fragmentos coincidentes resaltados para una mejor visualización.

Para concentrar la lógica de la búsqueda full text search, se definió un custom Model Manager llamado `SentenciaManager`. Previamente se debió definir en el modelo `Sentencia`, los campos por los que queremos buscar texto en ellos, los cuales fueron el titulo y el cuerpo de la sentencia.

En el método `search` 4.5.5, se definieron en primer lugar los vectores semánticos de búsquedas utilizando `SeachVector`: cada título y cuerpo de la sentencia serán convertidos al tipo `tsvector`, indicando el diccionario a utilizar para el pre-procesamiento (spanish) y su peso (`weight`) a tener en cuenta en el ranking de resultados. Se definió que en primer lugar un resultado de coincidencia en el título de la sentencia asigne más puntaje en el ranking seguido por las apariciones en el cuerpo de la misma.

Luego, se convirtió el texto a buscar al formato `tsquery`, pudiendo elegir entre modo frase o modo websearch, indicando también el diccionario a utilizar para las operaciones de stemming y eliminación de stopwords que realiza el motor.

Para utilizar la función de Postgres de ranking, se instanció el objecto SearchRank pasando como parámetros los vectores semánticos y el texto a buscar en formato `tsquery`. También se utilizó otra funcionalidad que ofrece Postgres llamada *Trigram Similarity* para adicionar otro criterio de relevancia de resultados. Un *trigram* es un grupo de tres caracteres consecutivos tomados de una cadena

de texto. Se puede evaluar la similitud de dos cadenas por el número de *trigrams* que comparten. Para esto previamente se debe tener activado el módulo `pg_trgm` en nuestro motor de base de datos.

Con todos estos elementos, ya podemos definir el `QuerySet` (una lista de objetos de un modelo determinado recuperados de la base de datos por el ORM) que retornará el método search, el cual filtrará los resultados relevantes, los ordenará en base a su ranking y resaltará las apariciones a través de tags html para una mejor visualización. Dicho método será invocado por el *controller* o *ClassView* encargado de gestionar la búsqueda de sentencias.

Algoritmo 4.1 SentenciaManager

```
from django.contrib.postgres.search import SearchQuery,
   SearchRank, SearchVector, TrigramSimilarity, SearchHeadline,

from django.db import models


class SentenciaManager(models.Manager):
    def search(self, search_text, modo, highlight):
        search_vectors = SearchVector(
            "titulo", weight="A", config="spanish"
        ) + SearchVector("sentencia_body", weight="B",
           config="spanish")

        search_query = SearchQuery(search_text, search_type=modo,
           config="spanish")

        search_rank = SearchRank(search_vectors, search_query)
        trigram_similarity = TrigramSimilarity("sentencia_body",
           search_text)
        qs = (
            self.get_queryset()
            .annotate(
                search=search_vectors,
                headline=SearchHeadline(
                    "sentencia_body",
                    search_query,
                    start_sel="<b>",
                    stop_sel="</b>",
                    min_words=30,
                    max_words=60,
                    max_fragments=2,
                    fragment_delimiter="...<br><br>",
                    highlight_all=highlight,
                ),
            )
            .filter(search=search_query)
            .annotate(rank=search_rank + trigram_similarity)
            .order_by("-rank")
        )

        return qs
```

Como se puede observar en la Fig. 4.18, los resultados de búsqueda son presentados al usuario a través de una pantalla que lista todos los posibles resultados ante el término de búsqueda, con la posibilidad de realizar filtrado por distintas categorías y ordenarlos por varios criterios (relevancia, fecha de resolución

ascendente o descendente).

Los términos relevantes detectados en la sentencia son resaltados en letra negrita, junto a su extracto de contexto en el documento (utilizando el atributo `headline`). Cabe destacar que el motor también detecta variaciones del término de búsqueda gracias a las normalizaciones de tokens a *lexemas* y los considera como relevantes, como es el caso de las palabras *allanó*, *allanándose*, *allanamientos* con respecto al término *allanamiento*, lo que resulta que la búsqueda sea más flexible y no solo se limite a una búsqueda literal sino semántica.

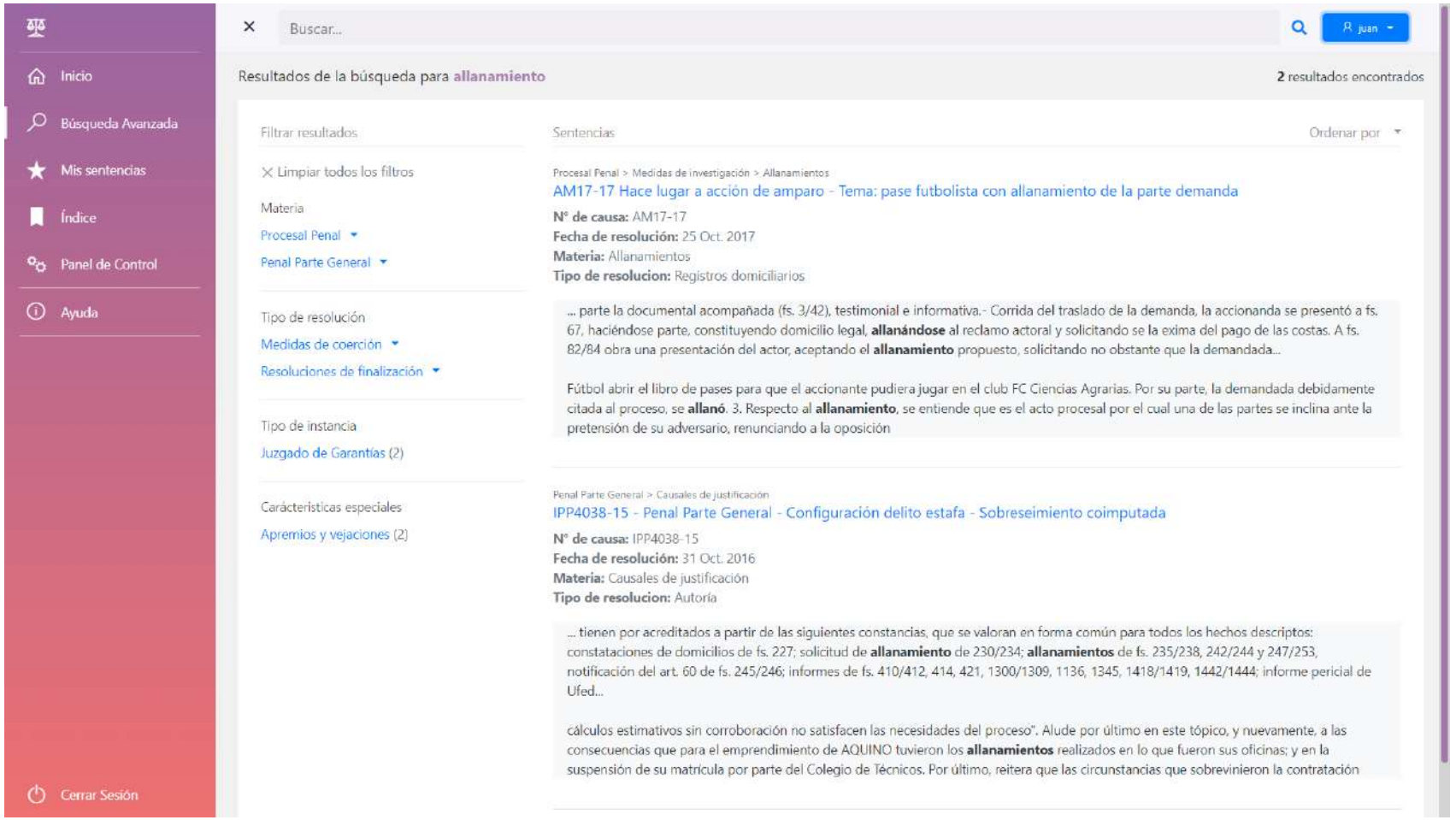


Figura 4.18: Resultados de la búsqueda. *Fuente: Elaboración propia*

4.5.2. Búsqueda Avanzada y filtros

Para el desarrollo de este requerimiento, se implementó una pantalla de búsqueda destinada específicamente a esta tarea. La misma le permite al usuario elegir y aplicar distintos filtros de búsqueda al conjunto de sentencias almacenadas en la base de datos, como se puede observar en la Fig. 4.19. Los filtros disponibles son:

- Búsqueda por coincidencia de palabras en el título y contenido de las sentencias (términos analizados de forma independiente). Se pueden incluir algunos operadores en la búsqueda como:
  - Texto sin comillas: serán procesados como términos separados con operadores '&'.
  - Texto entre comillas: serán procesados como términos separados con operadores `FOLLOWED BY`.
  - OR: operador lógico 'O'.
  - - : operador lógico 'NOT'.
- Búsqueda por coincidencia de frase (es decir, se busca por sentencias que contengan la frase exacta).
- Materia de Derecho Penal: campo de selección múltiple con vista en árbol de las posibles categorías en materia de derecho que pertenece una sentencia. Se pueden seleccionar una o más de una materia a través de los checkboxes.
- Instancia: campo de selección múltiple para elegir el tipo de instancia de la/s sentencia.
- Tipo de Resolución: campo de selección múltiple con vista en árbol de las

posibles tipo de resolución que posee una sentencia. Se pueden seleccionar una o más de una materia a través de los checkboxes.

- Características Especiales: campo de selección múltiple para elegir las características especiales que puede tener una sentencia a buscar.

- Fechas de resolución: Rango de fechas de resolución de las sentencias a buscar. Cada campo despliega un componente del tipo calendario para facilitar la elección de la fecha.

- Número de causa: Campo para buscar sentencias por su número de causa registrado en el sistema.

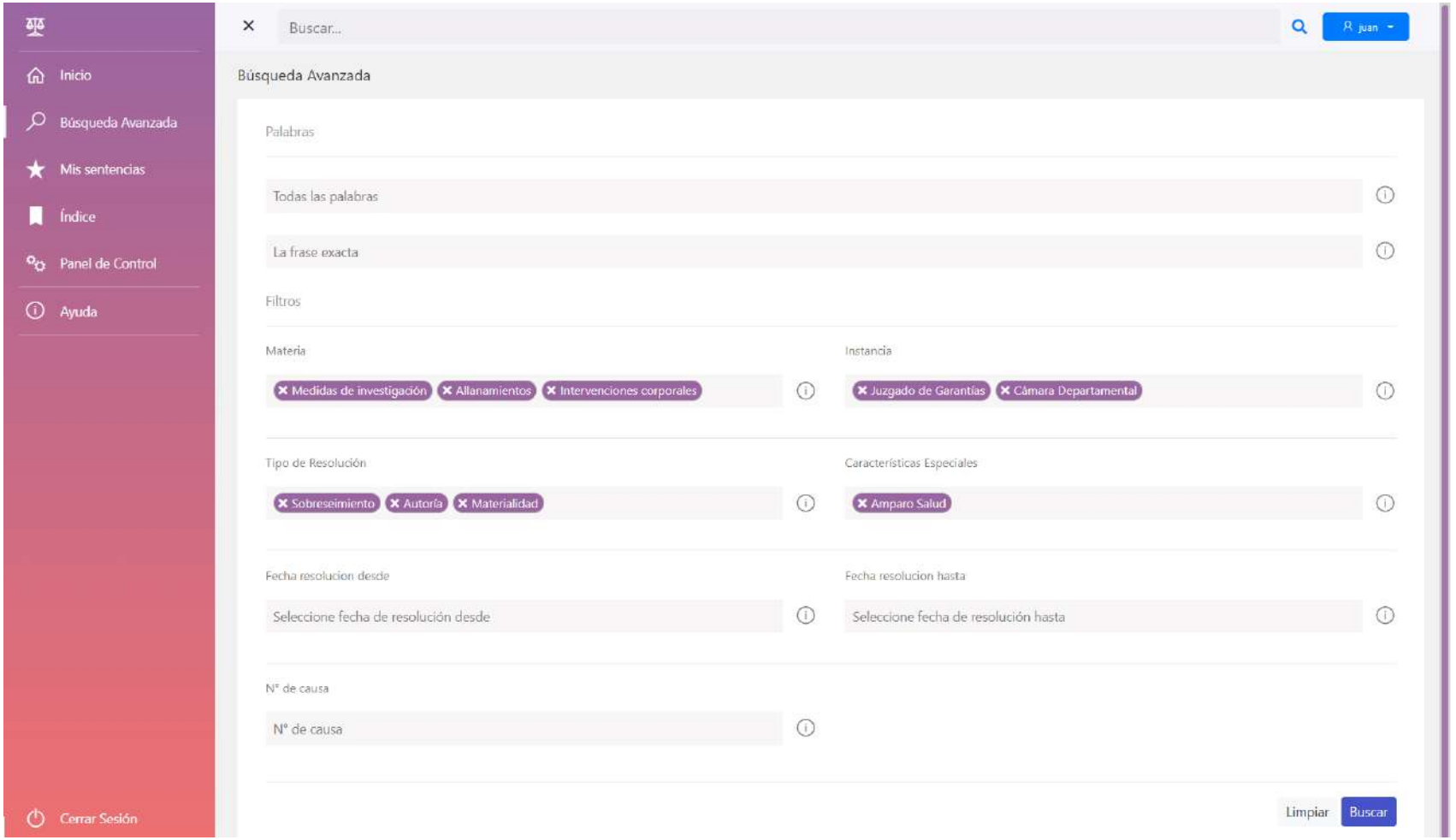


Figura 4.19: Búsqueda Avanzada. *Fuente: Elaboración propia*

Previo a implementar estas operaciones de filtrado, se debieron tener definidos los modelos necesarios en el archivo `models.py` que representan las tablas de la base de datos. Se utilizó la API que ofrece el ORM de Django, una abstracción de la base de datos para crear, recuperar, actualizar o borrar objetos representados a través de los modelos. En la Fig. 4.20 se pueden observar todos los modelos resultantes diseñados en Diagrama Entidad-Relación junto con algunos necesarios para los módulos de autenticación y sesión de usuarios.

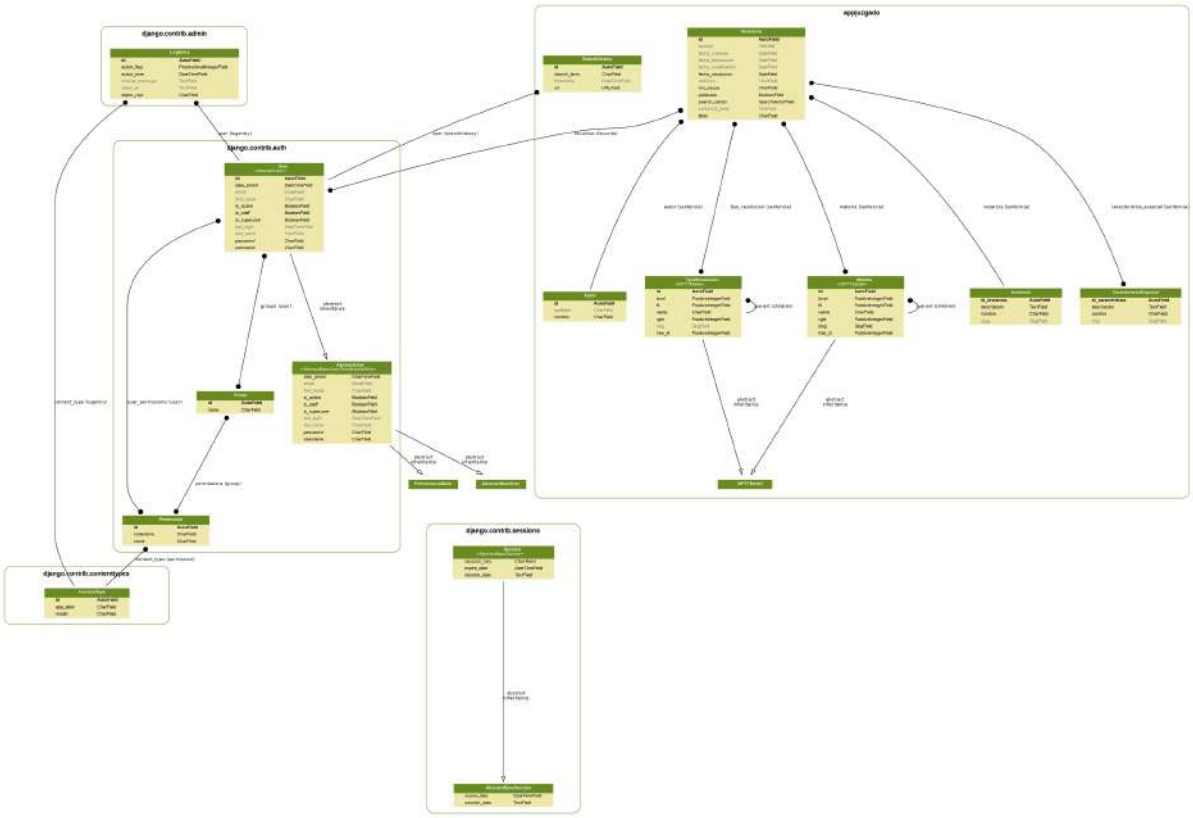


Figura 4.20: Modelos del sistema. *Fuente: Elaboración propia*

El objeto que representa una query o consulta a la base de datos es denominado `QuerySet`. El mismo se conforma de una colección de objetos de la base de datos, y puede poseer cero, uno o múltiples filtros, que reducen los resultados de la

consulta en función de los parámetros dados. En términos de SQL, un `QuerySet` equivale a una declaración `SELECT` y un filtro es una cláusula de limitación como `WHERE` o `LIMIT`.

Por ejemplo, si se quieren filtrar sentencias por su número de causa, bastaría con realizar la operación, a través del método `.filter()`:

```
Sentencia.objects.filter(nro_causa='12346')
```

Se pueden encadenar tantos filtros se deseen, y la ventaja es que el acto de crear un `QuerySet` no implica ninguna actividad de base de datos, sólo hasta que el mismo es evaluado. La operación del ejemplo anterior se traduce a la siguiente operación SQL, la cual se puede visualizar a través del atributo `query` del objeto `QuerySet`:

```
SELECT * FROM "appjuzgado_sentencia" WHERE
    "appjuzgado_sentencia"."nro_causa" = '123456'
```

Una vez recuperado la colección de objetos `Sentencia` que cumplen con los criterios de búsqueda, se procede a definir un contexto que requiere la *template* encargada de mostrar la información en la interfaz de usuario. Un contexto es un diccionario en el que las claves son nombres que se usan en la *template* para acceder los datos y los valores son los datos que necesitamos enviar a la *template*. El mismo no solo se compone de la colección de objetos sentencias, sino con atributos adicionales para mostrar la cantidad de sentencias que cumplen cada condición de filtrado, cantidad de resultados, paginación, entre otros.

Los resultados asimismo pueden ordenarse por relevancia, por fecha de resolución mas reciente a más antiguo y por fecha de resolución mas antiguo a más reciente.

Autocompletado

Una funcionalidad muy presente en sistemas de recuperación de información y motores de búsqueda es el autocompletado. La misma mejora la experiencia del usuario al buscar información, ofreciéndole sugerencias para completar el término de búsqueda a medida que va escribiendo. Se decidió implementar esta funcionalidad para complementar el requerimiento de búsqueda FTS. Para esto, se implementó autocompletado agrupado en 2 secciones: búsquedas relacionadas y documentos, como se puede ver en la Fig. 4.21.

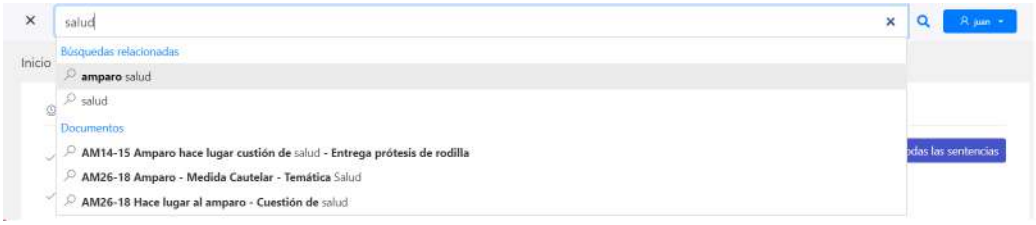


Figura 4.21: Autocompletado en el sistema. *Fuente: Elaboración propia*

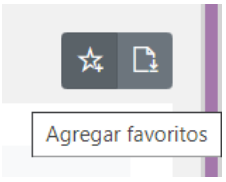
La sección "Búsquedas relacionadas" ofrece al usuario aquellos términos o búsquedas realizadas por otros usuarios del sistema. Para esto, fue necesario crear un modelo que registre los términos buscados por los usuarios, junto con un *endpoint* del sistema destinado a obtener dichas búsquedas relacionadas y popular el contenido del componente de la interfaz encargado de listar las sugerencias, que tienen un máximo de 5 valores posibles. La sección "Documentos" está destinada a proporcionar aquellos términos de búsqueda presentes en los títulos de las sentencias almacenadas en el sistema, con un máximo de 5 sugerencias. El autocompletado se ejecutará una vez que el usuario haya introducido al menos 2 caracteres en su búsqueda.

4.5.3. Detalle de una Sentencia

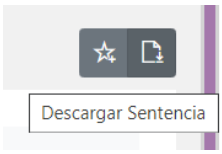
Una vez definida la búsqueda, fue necesario construir una pantalla que permita visualizar la sentencia judicial (Fig. 4.24), junto con sus detalles. Esta pantalla posee varias funcionalidades que se describirán a continuación.

En primer lugar, se visualizan los datos de la sentencia para dar contexto a la misma. En la parte superior su título, seguido por su N° de causa, fecha de resolución, instancia perteneciente, entre otros. Algunos de estos datos poseen hipervínculos que conducen a la sección Índice del sistema, donde se pueden ver la totalidad de sentencias por las diferentes categorías en términos de Instancia, Materia, Tipo de Resolución y Características Especiales.

Asimismo, en la parte superior, se encuentran dos botones tanto para agregar a favoritos la sentencia a la colección del usuario como para descargar el archivo original de la sentencia (en formato *.docx*), como se ven en las Fig. 4.22. Las sentencias agregadas a favoritos por el usuario serán incluidas en la sección Favoritos del sistema, accesible a través del menú de navegación principal.



(a) Agregar a favoritos



(b) Descargar

Figura 4.22: Opciones para la sentencia. *Fuente: Elaboración propia*

Además, se cuenta con una sección de búsqueda de palabras dentro del documento, sobre la parte izquierda de la vista. Dicha funcionalidad está diseñada para que el profesional busque los términos relevantes a lo largo del documento,

con el fin de ahorrar tiempo en su lectura. Se acompaña de dos botones para navegar por cada aparición del termino en la sentencia y cada término coincidente se resaltará en color naranja. Dicha funcionalidad fue implementada a través de la libreria `mark.js` [18] de JavaScript.

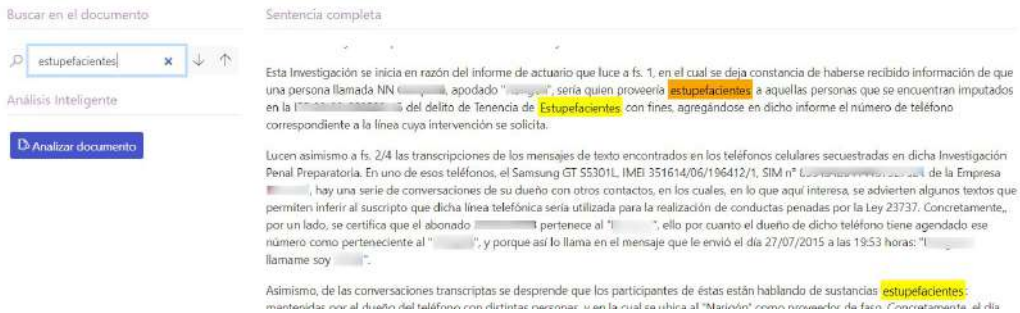


Figura 4.23: Búsqueda en el documento. *Fuente: Elaboración propia*

Una mejor visualización de la vista de detalle se presenta a continuación en la Fig. 4.24:

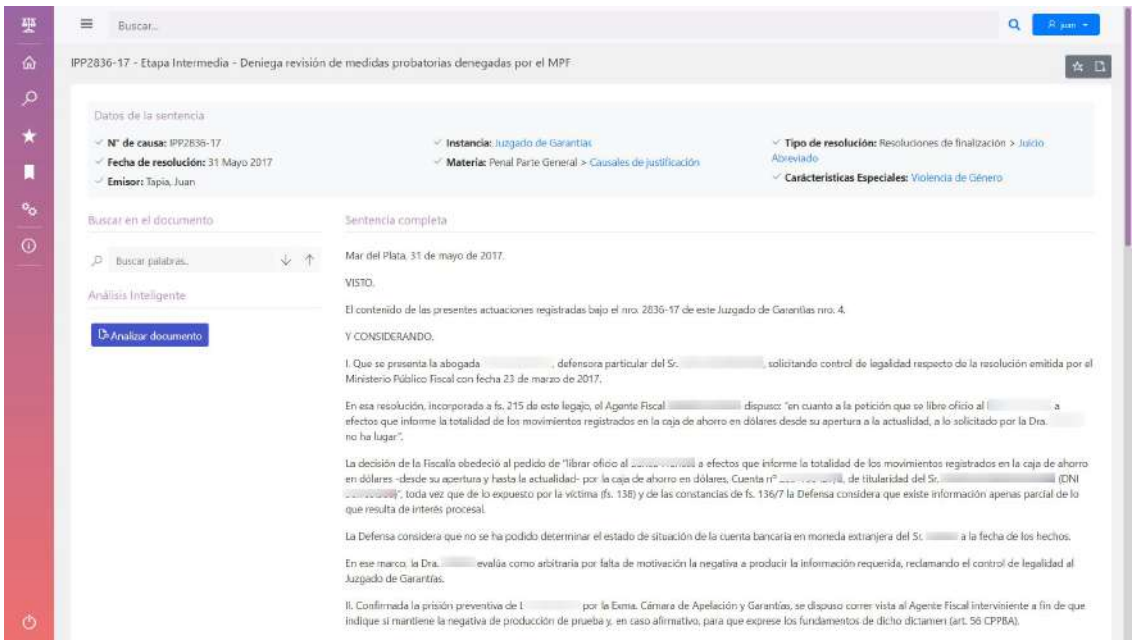


Figura 4.24: Pantalla Detalle de una sentencia. *Fuente: Elaboración propia*

4.5.4. Análisis Inteligente - Módulo NER

Se desarrolló la funcionalidad de realizar un "análisis inteligente " en donde el profesional de derecho pueda identificar con mayor facilidad los términos y conceptos más significativos de la sentencia . Aquí se hizo uso del modelo reconocedor de entidades nombradas (NER) de índole legal entrenado con spaCy, abordado en la sección 5.4.

Como se puede observar en las Fig. 4.26 y 4.25, el usuario puede visualizar el conjunto de entidades detectadas por el modelo en el texto de la sentencia, pudiendo diferenciar el tipo de predicción a través de sus colores. A la izquierda, se encuentra con el listado de entidades detectadas agrupadas por su tipo y cantidad de apariciones en el documento, pudiendo resultar en Citas o Referencias Legales (color violeta), Organizaciones o Entidades legales (color celeste), Personas (color verde) y Lugares o Localizaciones (color naranja). Esta información es sumamente útil porque se puede obtener una predicción en cuanto al contenido del documento y poder resumir con mayor precisión los aspectos claves de la sentencia.

Para el resaltado de las entidades en el texto, se utilizó e hizo uso de la librería `displacy-ent.js` del lado del navegador. Como se puede notar en la Fig. 4.26 para el caso de la entidad Cita, el modelo permite detectar diversos formatos de referencias y citas de instrumentos legales, como por ejemplo leyes, decretos, ordenanzas, acuerdos, resoluciones, artículos, códigos y/o fallos. Para integrar el modelo entrenado a Django, se expuso un endpoint que consume displacy el cual analiza el texto suministrado y retorna el conjunto de entidades detectadas. Cada entidad posee su identificación, a demás de su color correspondiente. En el caso de no detectar algún tipo de entidad especifica, no se incluirán en el listado de resultados por entidad.

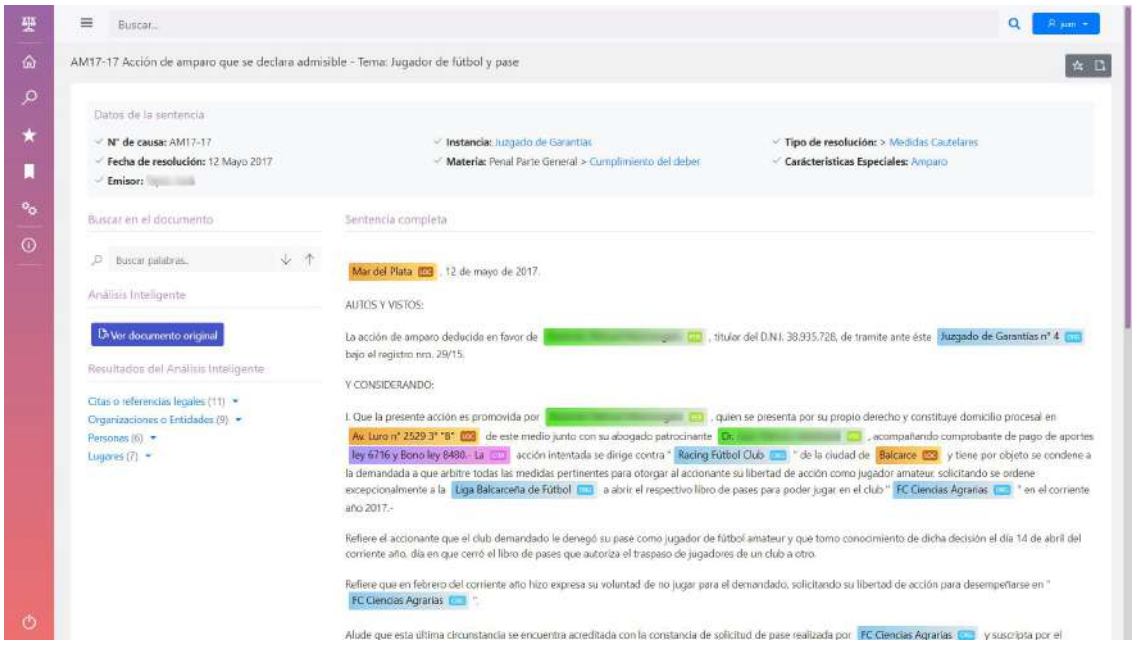


Figura 4.25: Resultados del Análisis Inteligente. *Fuente: Elaboración propia*

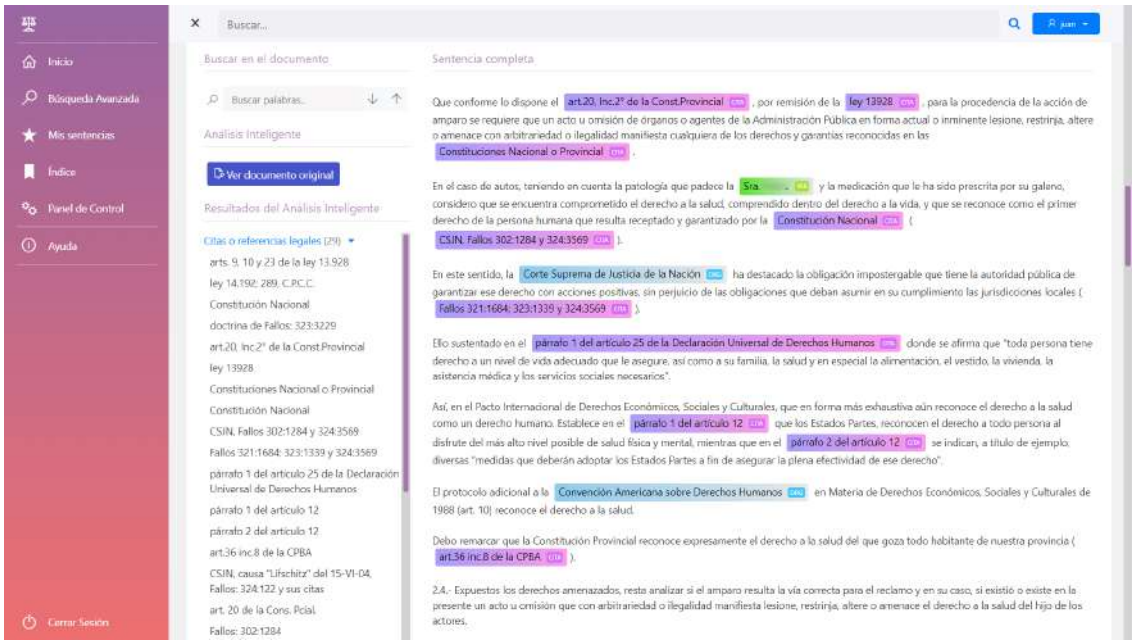


Figura 4.26: Resultados del Análisis Inteligente. *Fuente: Elaboración propia*

4.5.5. Panel de Administrador

Una de las partes más poderosas de Django es su interfaz de administración. Habiendo definido los modelos, la aplicación de administración de Django puede usar los mismos para construir automáticamente un área dentro del sistema que se puede utilizar para crear, consultar, actualizar y borrar registros.

Para esto, es necesario registrar en el archivo `admin.py` aquellos modelos que se definan necesarios para gestionar desde esta interfaz. La representación de un modelo en la interfaz de administrador está dada por la clase `admin.ModelAdmin`. Con el objetivo de configurar cómo los modelos son representados o qué datos se quieren visualizar, se debe crear una clase que herede de `admin.ModelAdmin` y registrarla a través del método `admin.site.register`.

La clase `ModelAdmin` es muy flexible. Posee varias opciones para lidiar con la personalización de la interfaz. Todas las opciones se definen en la subclase que herede de `ModelAdmin`, como se puede ver en el siguiente ejemplo:

Algoritmo 4.2 admin.py

```
1  from django.contrib import admin
2  from mptt.admin import DraggableMPTTAdmin,
       TreeRelatedFieldListFilter
3  from django.forms import TextInput
4  from .models import *
5
6  # Ejemplo de ModelAdmin para el modelo Sentencia
7
8  class SentenciasAdmin(admin.ModelAdmin):
9      list_display = ('titulo', 'nro_causa','fecha_resolucion',
           'fecha_creacion', 'publicado')
10     list_filter = (('materia', TreeRelatedFieldListFilter),
           ('tipo_resolucion',
           TreeRelatedFieldListFilter),'publicado')
11     formfield_overrides = {
12         models.CharField: {'widget':
               TextInput(attrs={'size':'100'})},
13     }
14     exclude = ('sentencia_body',)
15
16 admin.site.register(Sentencia, SentenciasAdmin)
```

En el ejemplo anterior, pueden definirse qué campos del modelo listar o visualizar a través del atributo `list_display`, cómo también qué columnas se desea que posean filtros para refinar los resultados a través de `list_filter`. También se puede sobreescribir el formato por defecto de los campos pertenecientes al form del modelo, como en este caso aumentar su tamaño para una mejor experiencia de usuario. Por ultimo, es posible excluir algún campo del modelo que no queremos mostrar, a través de `exclude`. Como resultado, los cambios reflejados en el panel de administrador donde podemos crear, editar o eliminar sentencias, se puede apreciar en la Fig. 4.27:

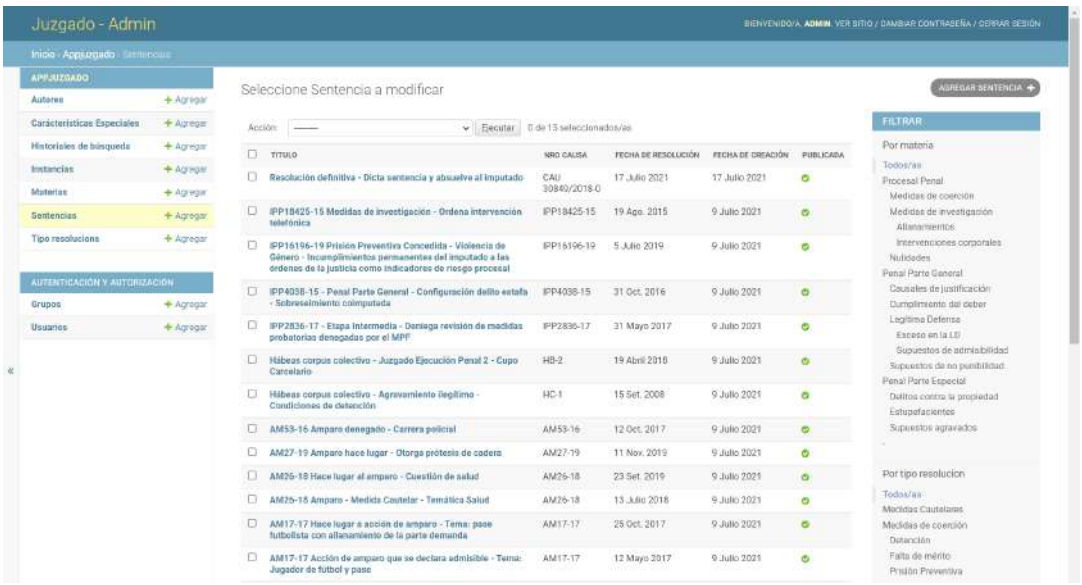


Figura 4.27: Panel de administrador - Modelo Sentencia. *Fuente: Elaboración propia*

Adicionalmente, si el usuario administrador desea agregar más materias pertenecientes a las ramas del Derecho Penal, y ubicar su posición dentro de la jerarquía de clasificación, puede realizarlo a través de la siguiente interfaz, destinada a gestionar el modelo Materia. En la misma se pueden expandir/colapsar

las categorías, como así también reubicar su posición, como se ve en la Fig. 4.28:

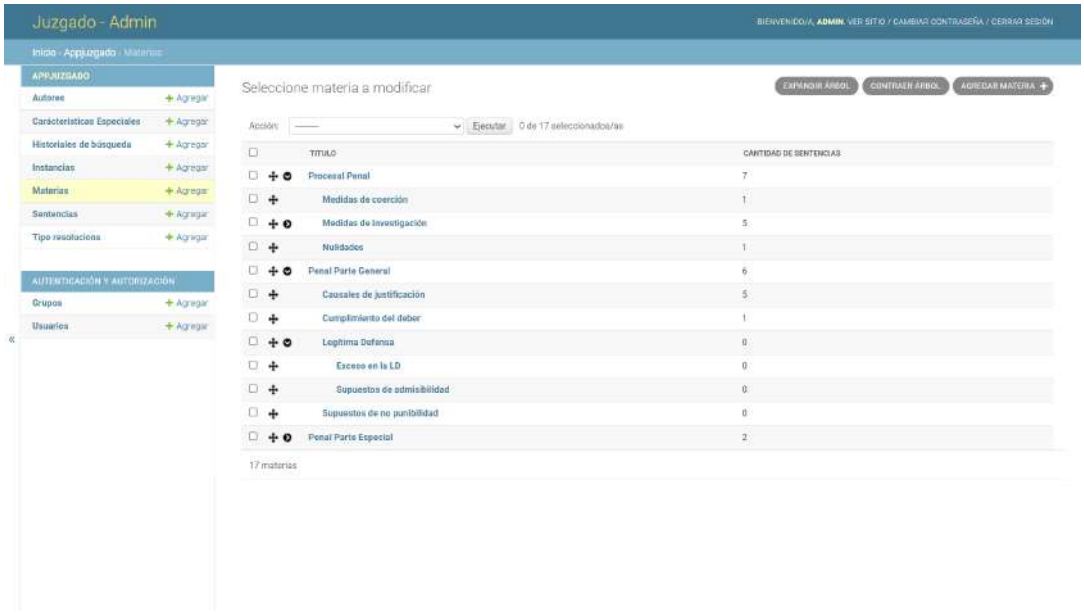


Figura 4.28: Panel de administrador - Modelo Materia. *Fuente: Elaboración propia*

En el panel es posible gestionar las cuentas de los usuarios, como crear grupos. Asimismo, los permisos para tanto para lectura como escritura de los modelos del sistema pueden otorgarse a nivel usuario como a nivel grupo. El superusuario del sistema es quien tiene la capacidad de hacerlo, pudiendo restringir la funcionalidad del panel de administración a un scope o ámbito adecuado por usuario. En base a esto, es posible definir distintos tipos de usuarios, que tengan funcionalidad parcial según los permisos otorgados por el administrador del sistema.

4.5.6. Aspectos de Seguridad

A continuación se enuncian algunos aspectos de seguridad relevantes para el desarrollo del sistema así como las mitigaciones que ofrece el framework elegido para el desarrollo del mismo.

Protección contra Cross Site Scripting

Segun OSWAP [19], Cross Site Scripting es un tipo de vulnerabilidad informática o agujero de seguridad típico de las aplicaciones Web, que puede permitir a una tercera persona o usuario malicioso inyectar en páginas web visitadas por un usuario código JavaScript o en otro lenguaje similar de scripting. XSS es un vector de ataque que puede ser utilizado para robar información delicada, secuestrar sesiones de usuario, y comprometer el navegador de la victima, subyugando la integridad del sistema.

Esta situación es habitualmente causada al no validar correctamente los datos de entrada que son usados en una aplicación, o no sanear la salida adecuadamente para su presentación como página web. Generalmente se logra almacenando los scripts maliciosos en la base de datos donde serán recuperados y mostrados a otros usuarios, o haciendo que los usuarios hagan clic en un enlace que hará que el navegador del usuario ejecute el JavaScript del atacante. Sin embargo, los ataques XSS pueden tener su origen en cualquier fuente de datos que no sea de confianza, como cookies o servicios web, siempre que los datos no estén lo suficientemente desinfectados antes de incluirlos en una página.

Al utilizar el sistema de plantillas de Django y generar el documento HTML a partir de estas, existe una protección contra la mayoría (aunque no todos) de los ataques XSS habituales, al escapar caracteres específicos que pueden ser potencialmente peligrosos. De forma automática, cada plantilla escapa automáticamente de la salida de cada etiqueta de variable. Específicamente, estos cinco caracteres se escapan:

- < es convertido a <
- > es convertido a >

- ' (comillas simple) es convertido a '
- "(comillas doble) es convertido a "
- & es convertido &

Protección Cross Site Request Forgery

Los ataques CSRF permiten que un usuario malicioso ejecute acciones utilizando las credenciales de otro usuario sin el conocimiento o consentimiento de este último. Django posee protección incorporada contra la mayoría de los tipos de ataques CSRF, siempre que se haya habilitado y utilizado donde sea apropiado.

La protección que provee Django para mitigar la vulnerabilidad CSRF funciona al verificar un secreto o token de seguridad en cada solicitud POST o formulario que expone el sistema. Esto asegura que un usuario malintencionado no pueda reproducir un formulario POST en el sitio web y que otro usuario que haya iniciado sesión envíe ese formulario sin saberlo. El usuario malintencionado debería conocer dicho token, que es específico del usuario (mediante una cookie). Este token único para cada usuario es generado al momento de inicio de sesión y enviado en cada petición que el usuario realice.

Para utilizar dicha protección, se debe asegurar que el middleware CSRF esté activado y para cualquier plantilla que use un formulario POST, incorporar la etiqueta `csrf_token` para cada solicitud con una URL interna.

Protección a SQL injection

La inyección SQL es un tipo de ataque en el que un usuario malintencionado puede ejecutar código SQL arbitrario en una base de datos. Esto puede provocar la eliminación de registros o la filtración de datos. Aunque es una de las vulnerabilidades mas conocidas a la fecha, debe considerarse como una vulnerabilidad

con gravedad de alto impacto.

El ataque de inyección SQL ocurre cuando un dato no deseado ingresa a un programa desde una fuente que no es de confianza y luego dichos datos se utilizan para construir dinámicamente una consulta maliciosa de SQL.

Django proporciona protección contra SQL injection. Las *querysets* del ORM de django estan protegidas debido a que son construidas a través de parametrización. El código SQL de una consulta se define por separado de los parámetros de la consulta. Dado que los parámetros pueden ser proporcionados por el usuario y, por lo tanto, no seguros, el controlador de la base de datos sanitiza estas entradas, removiendo aquellos caracteres especiales que podrían insertarse para explotar esta vulnerabilidad.

Password Hashing

La administración de contraseñas en la base de datos es algo que generalmente no debería reinventarse innecesariamente, debido a que representa un punto crítico y puede compromenter la seguridad del sistema si no se realiza correctamente. De forma predeterminada, Django usa el algoritmo PBKDF2 con un hash SHA256, un mecanismo de extensión de contraseña recomendado por NIST. Esto debería ser suficiente para la mayoría de los usuarios: es bastante seguro y requiere una gran cantidad de tiempo de computación para romperse. Sin embargo, si los requerimientos cambian, se puede elegir un algoritmo diferente o incluso utilizar un algoritmo personalizado para adaptarse a la situación de seguridad que sea necesaria (como por ejemplo los algoritmos de hash argon2 y bcrypt).

Capítulo 5

Gestión del Proyecto

5.1. Análisis FODA

Previo al inicio del proyecto, y en el momento que se decidió la temática del mismo, se analizaron todos los factores que implicaban afrontar este proyecto informático. Se presentan a continuación las fortalezas, amenazas, oportunidades y debilidades con las que el autor se encontró en el comienzo del desarrollo del presente proyecto.

Fortalezas:

- El estudiante que llevará a cabo el proyecto cuenta con conocimientos para afrontar el trabajo y está en constante aprendizaje de nuevas herramientas y conceptos teóricos.
- El estudiante se encuentra guiado por profesionales en el área de informática que le serán de gran ayuda para la realización del proyecto.

Oportunidades:

- En el campo del derecho, este sistema sería una solución novedosa que aplica técnicas de la inteligencia artificial para facilitar y agilizar la tarea de profesionales del área. No se han encontrado precedentes de aplicaciones similares en la ciudad de Mar del Plata.
- En las reuniones que se tuvieron con el personal del juzgado, se ha observado una buena predisposición para trabajar en las necesidades que surjan a lo largo del proyecto.

Debilidades:

- El entrenamiento de la red neuronal puede requerir una gran cantidad de datos y etiquetado para lograr una tasa de acierto aceptable.
- El volumen de datos a procesar puede incluir múltiples formatos, pudiendo complejizar el proceso de ingesta.
- Puede resultar limitado el hardware que dispone el alumno para realizar el entrenamiento de un modelo que se adapte a las necesidades del juzgado.

Amenazas:

- En el contexto actual este proyecto está situado bajo una pandemia sin precedentes, pudiendo impactar negativamente en los plazos de tiempo, debido a las limitaciones que presenta este escenario de aislamiento. Asimismo puede afectar principalmente etapas de relevamiento y pruebas en el juzgado.
- Existe una potencial competencia en el mercado actual de este tipo de software aplicado a otras regiones del país.

5.2. Métricas del proyecto

Se analiza aquí la duración del proyecto en horas, organizadas según las siguientes categorías:

- Análisis e Ingeniería de Requerimientos: implica las reuniones con los actores involucrados y futuros usuarios finales para obtener requerimientos funcionales y no funcionales. Análisis y definición de los requerimientos obtenidos y comprensión del dominio del problema. **Tiempo total**: 40 horas.

- Investigación del mercado: tiempo empleado en analizar los productos de software/soluciones similares al objetivo del proyecto en el mercado bajo un punto de vista funcional. **Tiempo total**: 15 horas.

- Investigación Técnica: tiempo referido a la investigación de posibles herramientas tecnológicas a utilizar (plataformas, lenguajes, librerías, frameworks, etc.), y la capacitación en los mismos. **Tiempo total**: 20 horas.

- Gestión del proyecto. Tareas de gestión, reuniones de revisión con los directores del Proyecto Final, confección de propuesta, a efectos de tomar decisiones y obtener feedback/retroalimentación. **Tiempo total**: 18,5 horas.

- Diseño del sistema: comprende los tiempos involucrados en etapas de diseño de arquitectura y módulos, diseño de los casos de uso del sistema, diseño de la base de datos y diagrama entidad-relación, *wireframes* y diseño de la interfaz web. **Tiempo total**: 90 horas.

- Modulo reconocedor de Entidades Nombradas (NER): tiempo empleado para el desarrollo y entrenamiento de un modelo NER reconocedor de entidades legales utilizando la librería spaCy.

Estas incluyen, entre otras tareas:

- Etiquetado manual de entidades en ejemplos de entrenamiento extraídos de sentencias judiciales.
- Pruebas con matcher, expresiones regulares y pipeline de spacy & displacy.
- Entrenamiento del modelo y verificación de resultados.

Tiempo total: 55 horas.

- Desarrollo y codificación del sistema: tiempo empleado en la programación del sistema, siguiendo la arquitectura MVT.

  Estas incluyen, entre otras tareas de desarrollo:

  - Maquetación en html, css y js de pantallas del sistema.
  - Preparación de entorno de desarrollo.
  - Creación y configuración de repositorio en github.
  - Creación de django admin.
  - Integración de Authentication system.
  - Creación de vistas (ClassView) y templates para pantalla de login.
  - Creación de template base del sistema.
  - Creación de estructura urls, views (class based views) y templates.
  - Codificación de modelos (datos - entidades del la base de datos).
  - Codificación de vistas y urls para las pantallas de búsqueda avanzada, e indice de sentencias.
  - Implementación carga de archivo word de una sentencia en el módulo administrador.

- Implementación de filtros de búsqueda, muestra de resultados, paginación y agregaciones/agrupaciones en los resultados.
- Implementación de búsqueda de contenido a través de full text search.
- Implementación de funcionalidad agregar sentencia a los favoritos del usuario.
- Implementación de funcionalidad indice de sentencias por materia, instancia, características especiales y tipo de resolución.
- Implementación funcionalidad "ordenar resultados por".
- Implementación de resaltado de palabras en la pantalla de detalle de la sentencia.
- Implementación autocompletado de la barra superior de búsqueda.
- Integración módulo NER. Carga del modelo custom en django settings e integración con `displacy-ent.js`.
- Test Unitarios para ClassView SearchResults.
- Visualización de filtros en pantalla de resultados.
- Aplicación de md5 y sh4 a los nombres de archivos para evitar archivos duplicados.
- Codificación funcionalidad historial de busqueda para un usuario.
- BugFixes. Arreglos en estilos y *media querys* para soportar pantallas de diversas resoluciones (responsive). Refactorizaciones de código.

Tiempo total: 160 horas.

- Testing: tiempo empleado para verificacion y validacion de requerimientos funcionales/no funcionales. Ad hoc testing y pruebas de integración entre módulos. **Tiempo total**: 30 horas.

- Redacción del Informe TFG : tiempo empleado en la redacción y documentación del presente informe. **Tiempo total**: 80 horas.

Tiempo total empleado: 508,5 horas.

El siguiente esquema (Fig. 5.1) muestra la distribución horaria y porcentajes de dedicación a cada actividad.

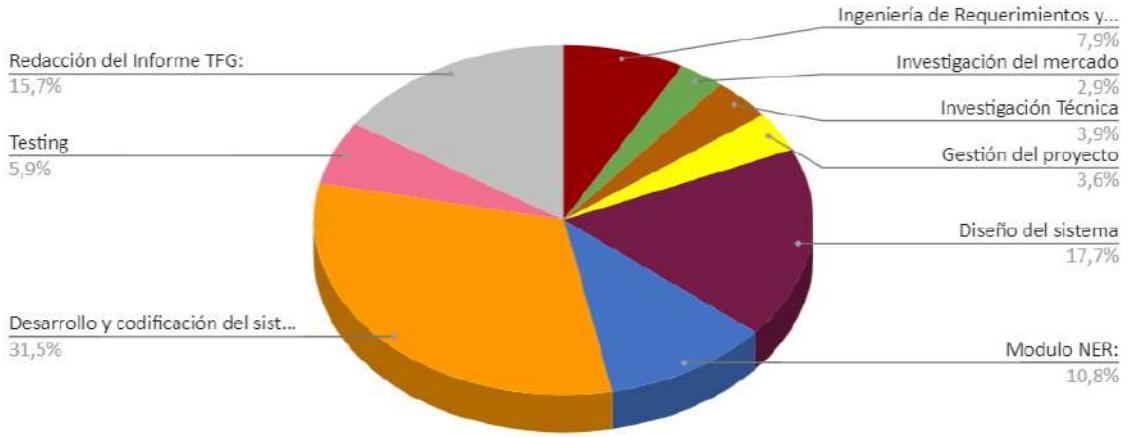


Figura 5.1: Distribución horaria de las actividades del TFG. *Fuente: Elaboración propia*

5.3. Análisis de la distribución horaria

Como se puede observar, la mayor parte del tiempo invertido fue dedicada a tareas de Desarrollo y Codificación del sistema, ya que el propósito de este trabajo desde un principio fue la entrega de un producto funcional, el cual se ve reflejado en los resultados obtenidos.

Asimismo, como parte del proceso de construcción de un producto de software, se invirtió mucho tiempo en la Ingeniería de Requerimientos y el Diseño del sistema, procurando tomar las decisiones correctas que minimizaran los riesgos y

complicaciones a lo largo del proyecto.

Con respecto a los tiempos de testing, se puede observar un porcentaje menor frente a las demás tareas de desarrollo, ya que básicamente se comprobó para cada funcionalidad implementada, que la misma funcione acorde al diseño y libre de errores, a través de pruebas manuales Ad Hoc. El hecho de ser un solo integrante en el proyecto influye en este aspecto, ya que como lo indica la teoría, las pruebas de software debe realizarlas otra persona distinta al desarrollador para evitar sesgos u omisiones de escenarios de prueba.

En cuanto a la Investigación del mercado e Investigación técnica, las mismas fueron realizadas en etapas iniciales del trabajo, como complemento a las actividades de Ingeniería de Requerimientos. Como se aclaró anteriormente, fue necesario invertir tiempo en capacitación y aprendizaje de las herramientas de software y librerías utilizadas en este proyecto, aunque la formación académica recibida y personal facilitó este proceso junto con los recursos disponibles en la red de forma gratuita y de fácil acceso.

La redacción del presente informe también fue una actividad significativa en el proyecto, realizada de forma paralela con las demás actividades, pero de forma más intensiva en las etapas finales del mismo. Si bien pudo resultar difícil plasmar todo lo trabajado, resultó ser un ejercicio sumamente útil para mejorar las habilidades de comunicación y competencias escritas de un ingeniero.

5.4. Análisis entre la planificación original vs ejecución del proyecto

En la propuesta del proyecto final se realizó una planificación en formato de Gantt que estimaba, antes de comenzar el proyecto, una distribución de etapas y

actividades a realizar junto con su estimación en semanas.

Resultó compleja realizar dicha estimación debido a la incertidumbre que se poseía en ese entonces en cuanto a etapas y/o actividades a abordar, sumado a que la misma se realizó bajo el contexto de la pandemia COVID-19. El protocolo de proyecto inicialmente presentado, se puede observar en la el Apéndice D.

Para poder realizar un análisis frente al tiempo ejecutado en horas [Hs] del proyecto, se tomó un promedio de dedicación semanal de 20 hs (4hs por día), teniendo en cuenta que el presente autor se encontraba cursando las materias de 5to año de la carrera, por aquel entonces. En la tabla 5.1 se puede observar las estimaciones en horas a partir del Gantt. Aquellas etapas cuyo tiempo de estimación aparece como 0, significa que no habían sido tenido en cuenta en la planificación inicial.

Tabla 5.1: Tiempo Estimado por Etapas.

Etapa	Estimado [Hs]
Ingeniería de Requerimientos y Análisis	100
Investigación del mercado	20
Investigación Técnica	20
Gestión del proyecto	0
Diseño del sistema	80
Modulo reconocedor de Entidades Nombradas (NER):	160
Desarrollo y codificación del sistema	200
Testing	120
Redacción del Informe TFG:	0
Total [Hs]	700

En la Fig 5.2, se puede observar la comparación entre las horas planificadas y las horas ejecutadas. Se puede observar una desviación entre el tiempo planificado y el tiempo ejecutado de un 27,35 %, donde el tiempo ejecutado resultó ser menor

que el planificado en horas.

Figura 5.2: Gráfico Planificado Vs. Ejecutado *Fuente: Elaboración propia*

A continuación, en la Fig. 5.3, se puede observar más en detalle la comparativa entre la planificación y la ejecución del proyecto por etapas. Si bien es sumamente positivo que la estimación haya resultado mayor que lo ejecutado, vale la pena detenerse y realizar un análisis de los desvíos, ya que en algunas etapas puede observarse una sobre-estimación y en otras, una estimación nula, por lo que no todas las etapas han sido bien estimadas. En la siguiente subsección se analizarán estos desvíos para obtener conclusiones que ayuden a mejorar la planificación en proyectos futuros.

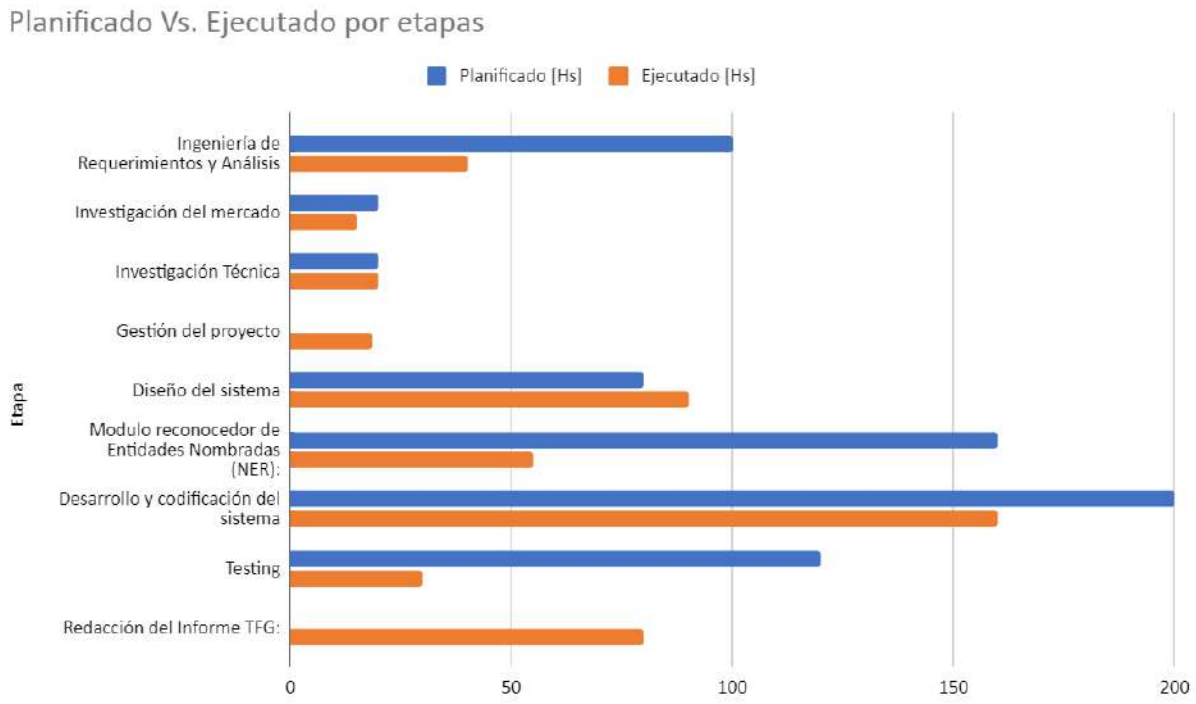


Figura 5.3: Gráfico Planificado Vs. Ejecutado por etapas *Fuente: Elaboración propia*

5.4.1. Desvíos en los tiempos planificados vs. ejecutados

Del grafico anterior, podemos establecer los siguientes desvíos:

- Hubo etapas que no fueron consideradas en la planificación original y debieron ser incluidas inicialmente, cómo la redacción del informe del trabajo final y las tareas dedicadas a la gestión del proyecto (reuniones pre-liminares, confección del protocolo y propuesta). Esto es un error de planeamiento, ya que fueron etapas importantes a la hora del abordaje del trabajo.

- Para la estimación de las tareas relacionadas al módulo NER, el tiempo estimado supera ampliamente al real. Esto tiene su fundamento en el desconocimiento de la tecnología a utilizar (spaCy) como el volumen de ejemplos

de entrenamiento a etiquetar para lograr obtener resultados aceptables, ya que no se disponía de una dimensión del corpus de documentos del juzgado para realizar el entrenamiento. La elección de la tecnología NLP a utilizar fue un gran acierto, ya que con un número relativamente bajo de ejemplos se logró desarrollar un modelo NER en poco tiempo. Esta sugerencia por parte de mis directores facilitó y aceleró de manera significativa los esfuerzos en el desarrollo del módulo reconocedor de entidades nombradas.

- La etapa de testing fue estimada de una forma distinta a la aplicada en el desarrollo del sistema. Se pensó inicialmente realizar pruebas unitarias o *test suites* a cada componente, clase o función, en paralelo al desarrollo. Esto en la práctica no fue posible aplicar esta idea, ya que algunos componentes resultaron difíciles de testear a través de código, sobre todo la interfaz de usuario realizada a través del sistema de templates. En su lugar, fueron realizadas pruebas manuales Ad Hoc, para corroborar y validar cada funcionalidad que se iba implementando.

- Con respecto al Análisis e Ingeniería de requerimientos, el tiempo ejecutado fue menor al planificado. Si bien al momento de realizar la planificación inicial se sabia la funcionalidad principal del sistema, existía cierta incertidumbre respecto a la cantidad final de requerimientos del sistema, por esto se tomo un tiempo alto de estimación. Sin embargo, se puede observar que las tareas de diseño, el tiempo ejecutado fue mayor al planificado.

- Con respecto al desarrollo y codificación del sistema, la estimación resultó correcta. La elección del framework backend Django aceleró bastante el proceso, al contar con muchas herramientas que facilitaron el desarrollo de tareas complejas que hubieran llevado más tiempo de desarrollo al utilizar otra herramienta, como es el caso del módulo administrador del sistema.

5.5. Análisis entre los plazos del proyecto (planificación vs real)

A continuación, se realizará un análisis de los plazos del presente proyecto final, con su posterior detalle de los desvios que resultaron en el proceso de trabajo.

Planificación Original

Fecha de Inicio prevista: **10/08/2020**

Fecha de Finalización prevista: **15/03/2021**

Ejecución del Proyecto

Diversos factores hicieron que la planificación del proyecto fuera ajustándose y cambiando, resultando la siguiente ejecución:

Fecha de Inicio Real: **11/09/2020**

Fecha de Finalización Real: **15/09/2021**

Como se puede observar, la entrega del trabajo demoró 6 meses más de lo planificado originalmente. El proceso del trabajo tuvo periodos con mucha actividad y otros con menor o nula por distintos motivos y factores, los cuales se explicarán el la siguiente sección.

5.5.1. Desvíos en los plazos

- Como se mencionó anteriormente, la planificación realizada a largo plazo no pudo ser respetada, aunque este factor fue analizado como un posible riesgo en la propuesta realizada, principalmente por el contexto de pandemia del COVID-19. Algunas devoluciones, correcciones y consultas de información

se vieron afectadas en los plazos debido a la alteración de la actividad laboral del juzgado al tener que realizar todos los procesos de manera remota sin poder acceder físicamente al establecimiento.

- Algunos procesos como la toma de requerimientos u obtención de documentos y archivos para entrenar el modelo NER, tuvieron algunos periodos de demora debido a la coordinación de entrevistas con el juzgado y obtención de la información necesaria. Sin embargo, esto no impidió que se trabaje de forma paralela en otras actividades necesarias para el proyecto, pero provocaron algunos intervalos de espera.

- El proyecto tuvo periodos de actividad mas fuertes que otros. Esto se debió a factores exógenos que son ajenos al proyecto, pero que influyeron en la demora de los plazos de entrega. A partir del comienzo del presente año, el autor de este informe comenzó a trabajar de forma full-time, por lo que la disponibilidad horaria semanal se vio notablemente afectada, factor no tenido en cuenta en la planificación original. Fue requerido tiempos de capacitación a nivel personal para afrontar otras obligaciones laborales, por lo que hubo periodos de inactividad en el proyecto. Si bien era sabido que tomar esta oportunidad afectaría negativamente los plazos estimados para el desarrollo y entrega del presente trabajo, también se evaluó que se adquiriría mucha experiencia en cuanto a aprendizaje por estar desempeñándose en un ámbito profesional del desarrollo e Ingeniería del Software.

- Los periodos mas fuertes de desarrollo del sistema tuvieron lugar entre el ultimo trimestre del 2020 y el primer trimestre del año 2021, mientras que los periodos restantes no fue posible dedicarle el tiempo deseado inicialmente al reducirse la disponibilidad horaria del autor del presente informe.

5.6. Lecciones aprendidas

Una vez concluido el desarrollo del proyecto, y realizando un análisis objetivo y retrospectivo, se pudieron identificar las distintas lecciones aprendidas, las cuales pueden observarse bajo distintos puntos de vista.

Desde el punto de vista técnico, algunas de las lecciones aprendidas fueron:

- Utilicé y trabajé con herramientas de software vigentes en el mercado, lo cual significó invertir tiempo no solo en capacitación, si no también en un análisis de las distintas alternativas. Haber realizado dicho análisis en una etapa temprana fue clave para minimizar los riesgos en los tiempos de desarrollo. Me llevo esta valiosa lección, ya que un ingeniero debe poseer la capacidad de poder identificar cual es la herramienta más adecuada a utilizar para resolver un problema bajo un contexto específico (y no perder de vista el criterio "depende para qué"). Hoy en día la tendencia al comenzar un desarrollo informático es utilizar las tecnologías de "moda", pero no siempre son las adecuadas bajo distintos contextos.

- Si bien la combinación de Django Templates y librerías JavaScript como *jquery* para el frontend permitieron resolver algunos desafíos a la hora de implementar funcionalidades, se experimentaron algunas limitaciones y dificultad en la separación de componentes de UI reutilizables como lo ofrecen librerías de frontend como React, Angular o Vue. Si bien se sabia que utilizar estas últimas tecnologías implicarían desplegar dos aplicaciones distintas (una para el cliente y otra para el servidor), me deja la experiencia de haber trabajado con el framework Django pudiendo conocer no solo lo bueno que ofrece, sino también sus limitaciones y desventajas.

- Realizar distintos artefactos de diseño permitieron tener un concepto claro

a la hora de implementar y codificar el sistema, y representan un insumo esencial para el futuro mantenimiento del sistema. Aprendí que documentar y confeccionar dichos diagramas deben responder a una necesidad, y bajo un proceso ágil, debe ser lo necesario y tener un propósito objetivo, a diferencia de otras metodologías tradicionales, donde suelen realizarse artefactos de diseño que poco se ajustan a las necesidades reales de un sistema informático desde el punto de vista de un desarrollador, quien debe materializar el diseño en código.

Desde el punto de vista de competencias relacionadas a la gestión del proyecto, algunas de las lecciones aprendidas fueron:

- Una vez concluido el proyecto, y habiendo realizado el análisis de los plazos y tiempos del mismo, existieron indudablemente desvíos. Como se explicó anteriormente, factores externos como la pandemia y compromisos laborales hicieron que los plazos de entrega se demoren. Como aprendizaje, puedo observar que la estimación de los plazos no fue la más adecuada y quizá fue planificada con un criterio muy optimista, ya que no se consideró la modificación del factor importante de horas de dedicación y la posibilidad de que las mismas cambien al comenzar a ejercer una actividad laboral. Otra lección adquirida es tratar de ajustarse al calendario planificado lo más posible, ya que si bien el tiempo en horas (hs) planificado fue correcto, la fecha de finalización fue 6 meses después de lo previsto.

- Haber realizado dicho análisis comparativo en retrospectiva de las métricas del proyecto me significó un aprendizaje en términos de planificación y gestión de proyectos, ya que en el mundo del software, estimar correctamente las horas de trabajo a dedicar es crucial, debido a que el recurso principal en los proyectos informáticos son las horas de trabajo intelectual. Por ejemplo,

una mala estimación en proyectos con fines de lucro, puede llegar a significar grandes pérdidas de dinero y esfuerzo.

- El haber realizado un proyecto de esta magnitud significa un aprendizaje muy valioso para la organización y priorización de tareas a realizar.

Capítulo 6

Trabajos a futuro

En esta sección se enuncian algunas de las propuestas mediante las cuales el sistema podría mejorarse como así también funcionalidades que se podrían implementar a futuro:

- Exploración y adición de componentes adicionales al pipeline de spaCy: la librería utilizada ofrece componentes adicionales al NER que pueden complementar el analisis linguistico de una sentencia. Podrian incorporarse elementos como categorizador de textos, part-of-speech tagging o entity linking, para establecer relaciones entre diferentes entidades.

- Mejoras en el entrenamiento: proporcionar y recopilar más ejemplos de entrenamiento para seguir mejorando la capacidad predictiva y métricas del modelo NER.

- Capacidad de exportar un informe con el análisis inteligente realizado y estadísticas: si bien el sistema proporciona la vista del documento junto con las entidades detectadas, seria útil la posibilidad de auto-generar un informe en un formato como pdf, con los resultados del análisis junto con las estadísticas obtenidas.

- Mejoras en el sistema de búsqueda: reutilizar el detector de entidades para fortalecer la búsqueda de sentencias no solo a través de full text search sino utilizando *tags* que resuman el contenido de la sentencia con mayor precisión. En base a esto, también podrían ofrecerse sugerencias de búsquedas en caso de no encontrar lo requerido.

- Formatos de entrada: posibilidad de agregar mayor variedad de formatos de archivos a analizar y almacenar en el sistema.

- Expansión a varios juzgados: el sistema está preparado para adaptarse no sólo a un Juzgado en específico, sino que es flexible a aplicarse en distintas instituciones. Sus funcionalidades fueron pensadas para ser aprovechables en varios contextos, por lo que podría mejorarse para ser un producto disponible a muchos juzgados de la ciudad.

Capítulo 7

Conclusiones

En este trabajo se ha abordado el análisis, diseño e implementación de un sistema informático para la gestión, búsqueda y análisis de sentencias judiciales del Juzgado de Garantías N°4 de la ciudad de Mar del Plata, con el objetivo principal de ser una herramienta para la labor diaria y permitir generar una mayor agilidad en el proceso de búsqueda de jurisprudencia. A continuación, se expondrán las siguientes conclusiones respecto al producto, al proyecto y a título personal.

7.0.1. Respecto al producto

Se ha podido construir un sistema de software que mejora la eficiencia en la búsqueda de jurisprudencia. Los sectores jurídico y legal trabajan con grandes volúmenes de información, analizada de manera manual, invirtiendo una gran cantidad de tiempo en revisar cada sentencia. El sistema resuelve esta problemática pudiendo analizar y buscar por información relevante en mucho menor tiempo que si se realizara de forma manual.

El sistema fue diseñado para aportar mejoras en la organización y administración de las sentencias digitales del juzgado, conformando un soporte digital

centralizado de sentencias, donde todos los integrantes del juzgado puedan no solo consultar, sino también agregar y gestionar la información decisiones, sentencias y fallos dictados por el juzgado u otros organismos jurídicos.

Asimismo, el usuario ha mostrado gran aceptación al producto y ha manifestado su conformidad a lo largo de las reuniones donde se ha mostrado el sistema en funcionamiento. Se ha podido diseñar y materializar una interfaz de usuario intuitiva enfocada en la facilidad de uso. No solo fue construida para ser visualizada en computadoras, sino también se adapta a todo tipo de resoluciones, como dispositivos móviles y/o tablets.

Se ha podido implementar un sistema que utilice técnicas del NLP, como se había establecido en el requerimiento inicial del trabajo. La aplicación permite analizar el texto extraído de las sentencias judiciales aplicando un Reconocedor de Entidades Nombradas para detectar los datos y conceptos relevantes del documento en distintas categorías en texto legal.

7.0.2. Respecto al proyecto

El proyecto fue el primero en su magnitud, donde pude aplicar todos los conceptos aprendidos a lo largo de la carrera universitaria. Se logró identificar una problemática perteneciente a la actividad laboral del juzgado. El esfuerzo principal de este trabajo se centró en el análisis, diseño e implementación de un proyecto de ingeniería, comenzando desde el planeamiento inicial hasta su ejecución final. Si bien la planificación original no consideró algunos aspectos importantes que surgieron a lo largo del proyecto, puedo concluir que detectar dichos errores y desvíos conforman un gran aprendizaje a futuro para no volver a cometerlos.

Apliqué diversas técnicas y herramientas para resolver distintas problemáticas; desde el prototipado, realización artefactos de diseño para una mejor toma de decisiones, modelos de datos. El proyecto permitió utilizar varias tecnologías y aprender muchos conceptos técnicos. Algunas decisiones tomadas que si bien cumplieron con el objetivo establecido, con la experiencia aprendida a lo largo del desarrollo, podrían haber sido mejores. Es el caso por ejemplo de las tecnologías frontend utilizadas, como se mencionó en el capitulo 5.

La redacción del presente informe me permitió expresar y sintetizar los épicos mas importantes del proyecto. Si bien no fue una tarea fácil poder plasmar todos los acontecimientos involucrados en el proceso, resultó ser un ejercicio invaluable para mejorar en mis habilidades de comunicación.

7.0.3. A título personal

Recapitulando y comenzando con entender el dominio de la problemática propuesta por el InfoLab, fue imprescindible adquirir algunos conceptos teóricos y prácticos necesarios para afrontar este proyecto. En primera instancia, se investigaron algunos casos prácticos presentes en el mercado actual que puedan dar indicios sobre la viabilidad de la solución propuesta. Esta actividad resultó de suma utilidad, ya que permitió observar cómo las herramientas informáticas interactúan y agregan valor en la actividad laboral de profesionales del Derecho y Organismos Judiciales.

Luego, fue necesario realizar un relevamiento de los requerimientos funcionales y no funcionales que el sistema debía poseer. Este proceso fue realizado junto con los usuarios finales del sistema, a través de reuniones con el personal del Juzgado, realizando consultas y resolución de dudas, obtención de información y recibiendo validaciones y *feedback* sobre la solución propuesta. Aquí mismo quiero resaltar

la importancia de esta actividad y la experiencia que me dejó al involucrarme con profesionales de un ámbito completamente distinto a la Ingeniería, como lo es el Derecho. La habilidad de poder comunicarse interdisciplinariamente y colaborar con profesionales de otros ámbitos representa una experiencia sumamente gratificante. Aquí toma mucha importancia la competencia de comunicación que debe poseer un ingeniero. Para lograr una buena productividad en cada reunión, fue necesario transmitir mis ideas de manera clara y precisa, sin abrumar en tecnicismos y expresar los conceptos de una manera entendible para alguien que no posee todo el contexto técnico involucrado.

Otro aspecto que quiero resaltar, es una reflexión personal sobre el hecho de haber existido un solo integrante en este proyecto. Esta carácteristica particular del proyecto presenta ventajas y desventajas, las cuales pude experimentar a lo largo del mismo. Como ventaja, puedo decir que tuve que ejercer y tomar distintos roles a lo largo del proceso de desarrollo, por lo que se puede decir que mis tareas fueron multi-roles: planificador de proyecto, analista de requerimientos, arquitecto de una solución informática, diseñador y desarrollador, entre otros. Esta multi-actividad me permitió comprender cada eslabón del proceso y adquirir experiencia en cada una de las actividades desarrolladas. Al ser un sólo integrante, se gana velocidad en la toma de decisiones. Pero este aspecto al mismo tiempo puede volverse un punto negativo. Si bien cada duda o decisión importante fue consultada con mis tutores, se pierde un ingrediente importantísimo en un proyecto, que es el trabajo en equipo si el mismo hubiera sido conformado por varios estudiantes. La posibilidad de debatir, intercambiar ideas, obtener distintos puntos de vista ante una posible solución, y formar parte de ese proceso de aprendizaje mutuo, son factores que al conformar un proyecto unipersonal, no se pueden experimentar de la misma forma que realizándolo entre varios integrantes.

Como resultado, se obtuvo un sistema que es de utilidad para la búsqueda

de jurisprudencia y que cumple con los requerimientos pactados con el cliente y el alcance de este proyecto final. Sin dudas este trabajo establece una base para posibles mejoras y adiciones de cara al futuro, y permitió aplicar principios de la Ingeniera Informática para resolver una problemática de otra disciplina profesional como lo es el Derecho.

Capítulo 8

Bibliografía

[1] Secretaría General de Capacitación y Jurisprudencia Defensoria General de la Nación. Guía de herramientas para la búsqueda de jurisprudencia nacional en internet. *Ministerio Público de la Defensa*, Febrero 2014.

[2] Pablo Fillottrani Elsa Estevez, Sebastián Linares Lejarraga. Prometea - transformando la administracion de justicia con herramientas de inteligencia artificial. 2020.

[3] IALAB. Pretoria, 2020. URL https://ialab.com.ar/pretoria/. [Online; Último acceso 19-Enero-2021].

[4] FAM -Federación Argentina de la Magistratura y la Funcion Judicial. Cendoj: Vanguardia global de tecnologÍas aplicadas a la tarea judicial, 2019. URL https://www.fam.org.ar/noticia/cendoj-vanguardia-global-de-tecnologias-aplicadas-a-la-tarea-judicial. [Online; Último acceso 19-Enero-2021].

[5] What's new in v3.1 - spacy usage documentation, Agosto 2021. URL https://spacy.io/usage/v3-1. [Online; Último acceso 14-Agosto-2021].

[6] ICLR - Incorporated Council of Law Reporting (UK). Blackstone - open source natural language processing for legal texts, 2019. URL https://research.iclr.co.uk/blackstone. [Online; Último acceso 19-Enero-2021].

[7] Dominik Kozaczko. 8 best python natural language processing (nlp) libraries, Junio 2018. URL https://sunscrapers.com/blog/8-best-python-natural-language-processing-nlp-libraries/. [Online; Último acceso 21-Enero-2021].

[8] Django documentation, Mayo 2021. URL https://docs.djangoproject.com/en/3.2/. [Online; Último acceso 1-Mayo-2021].

[9] Welcome to flask - flask documentation, Mayo 2021. URL https://flask.palletsprojects.com/en/1.1.x/. [Online; Último acceso 4-Mayo-2021].

[10] Introducción a django, Mayo 2021. URL https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction. [Online; Último acceso 1-Mayo-2021].

[11] Introduction - bootstrap, Mayo 2021. URL https://getbootstrap.com/docs/4.6/getting-started/introduction/. [Online; Último acceso 10-Mayo-2021].

[12] Git, Mayo 2021. URL https://git-scm.com/. [Online; Último acceso 4-Mayo-2021].

[13] Roger S. Pressman. Ingeniería del software, un enfoque práctico. séptima edicion. *University of Connecticut - ISBN : 978-607-15-0314-5*, 2010.

[14] Cecilia Ruz Sergio D'Arrigo, Leticia Seijas. Apunte de modelización. *Departamento de Computación - Facultad de Ciencias Exactas y Naturales - UBA*.

[15] Django mptt documentation, Mayo 2021. URL https://django-mptt.readthedocs.io/en/latest/index.html. [Online; Último acceso 30-Mayo-2021].

[16] Postgresql: Documentation: 13: 12.3. controlling text search, Junio 2021. URL https://www.postgresql.org/docs/current/textsearch-controls.html. [Online; Último acceso 20-Junio-2021].

[17] Full text search — django documentation, Junio 2021. URL https://docs.djangoproject.com/en/3.2/ref/contrib/postgres/search/. [Online; Último acceso 20-Junio-2021].

[18] mark.js: Javascript keyword highlight, Julio 2021. URL https://markjs.io/. [Online; Último acceso 19-Julio-2021].

[19] Cross site scripting (xss) software attack — oswap software foundation, Agosto 2021. URL https://owasp.org/www-community/attacks/xss/. [Online; Último acceso 14-Agosto-2021].

[20] IAARHub. Introducción al procesamiento del lenguaje natural, 2016. URL https://iaarbook.github.io/procesamiento-del-lenguaje-natural/. [Online; Último acceso 18-Enero-2021].

[21] Juan Ignacio Velez Facundo Matías Ramos. Integración de técnicas de procesamiento de lenguaje natural a través de servicios web. *Universidad Nacional del Centro de la Provincia de Buenos Aires*, Mayo 2016.

[22] Peter Russell, Stuart; Norvig. Inteligencia artificial: Un enfoque moderno (3ra edición). *Prentice Hall*, page 229, Diciembre 2009.

[23] Matthew Kirk. Thoughtful machine learning with python - a test-driven approach. *ISBN: 978-1-491-92413-6*, page 15, Enero 2017.

[24] LucidImagination. Capabilities of full text search system, 2010. URL https://web.archive.org/web/20101223192214/http://www.lucidimagination.com/full-text-search. [Online; Último acceso 18-Enero-2021].

[25] Fernando R.A. Bordignon Gabriel H. Tolosa. Introducción a la recuperación de información - conceptos, modelos y algoritmos básicos. *Laboratorio de Redes de Datos, División Estadística y Sistemas - Departamento de Ciencias Básicas -Universidad Nacional de Luján*, pages 10–13, Abril 2007.

Apéndice A

Definiciones teóricas

A.1. Introducción

Para cumplir con los objetivos planteados a lo largo de todo el proyecto, se requirió investigar y analizar diferentes conceptos teóricos para llevar a cabo una solución óptima. Para el posible interés del lector, se incluye a continuación una descripción teórica sobre aquellos conceptos importantes a la hora de desarrollar este trabajo.

A.2. Procesamiento del lenguaje natural

El lenguaje humano se apoya en la capacidad de comunicarse por medio de signos lingüísticos (usualmente secuencias sonoras y signos gráficos, pero también con gestos y señas). Es una de las herramientas centrales en nuestra vida social y profesional. Entre otras cosas, actúa como un medio para transmitir ideas, información, opiniones y sentimientos; así como para persuadir, pedir información, o dar órdenes. Asimismo, el lenguaje humano es algo que está en constante evolución; y que puede llegar a ser sumamente ambiguo y variable. Sí se toma por

ejemplo la frase *comí una pizza con amigos* comparada con *comí una pizza con aceitunas*; aunque se puede observar que su estructura es la misma, su significado es distinto. De la misma manera, un mismo mensaje puede ser expresado de formas diferentes; *comí una pizza con amigos* puede también ser expresado como *compartí una pizza con amigos*. [20]

Los seres humanos se caracterizan por ser muy buenos a la hora de producir e interpretar el lenguaje. Pueden expresar, percibir e interpretar significados muy elaborados en fracción de segundos casi sin dificultades, pero al mismo tiempo se les dificulta mucho entender y describir formalmente las reglas que lo gobiernan. Por este motivo, entender y producir el lenguaje humano por medio de una computadora es un problema muy difícil de resolver. Éste problema, es el campo de estudio de lo que en inteligencia artificial se conoce como Procesamiento del Lenguaje Natural o NLP por sus siglas en inglés.

El Procesamiento del Lenguaje Natural, es una rama de la Inteligencia Artificial cuyo objetivo es facilitar la interacción entre humano y máquina a través de lenguajes naturales (por ejemplo ingles, español, francés, etc.). En particular se encarga de cómo programar las computadoras para procesar y analizar grandes cantidades de datos del lenguaje natural. El resultado es una computadora capaz de "comprender" el contenido de los documentos, incluidos los matices contextuales del idioma que contienen.

A.2.1. Niveles de Procesamiento

Existe una clasificación de los diversos tipos de procesamientos que se pueden realizar sobre un texto. Esta se puede ver más claramente como una estructura de niveles, permitiendo así diferenciar los aspectos que trata cada nivel de proceso y la complejidad que posee cada uno. Cada uno de estos representa un tipo

de análisis que se debe efectuar al texto de entrada para extraer información específica. [21]

En general, en el NLP se utilizan seis niveles de comprensión con el objetivo de descubrir el significado de un discurso. Estos niveles son:

- Nivel fonético: se presta atención a la fonética, la forma en que las palabras son pronunciadas. Este nivel es importante cuando se procesa la palabra hablada (auditivamente), no así cuando se trabaja con texto escrito.

- Nivel morfológico: se encarga de analizar la composición de las palabras. El análisis de este nivel consiste en determinar la forma, clase o categoría gramatical de cada palabra dentro de una oración. Teniendo en cuenta este nivel, un sistema NLP es capaz de desglosar una palabra y obtener el significado a través del significado de cada uno de sus morfemas. Por ejemplo, si se busca interpretar la palabra "Libros", se obtiene que "Libr-" es el lexema, que "-o-" y "-s" son morfemas dependientes flexivos de masculino y plural.

- Nivel léxico: se encarga del significado individual de cada palabra. Para poder realizar el procesamiento léxico, se debe tratar cada palabra por sí misma, y etiquetarla con una parte del discurso dependiendo del contexto en la cual se encuentra. Cada palabra que compone un texto lleva asociada un conjunto de datos morfológicos, sintácticos y semánticos. El objetivo específico de este nivel es analizar cada una de estas palabras para saber su significado y función dentro de una oración.

- Nivel sintáctico: se encarga de analizar la función de cada palabra dentro de una oración, descubriendo así la estructura u organización de la misma. El resultado de este procesamiento será una representación de la oración

analizada que mostrará la relación entre las palabras que la conforman, como se puede ver en la Fig. A.1

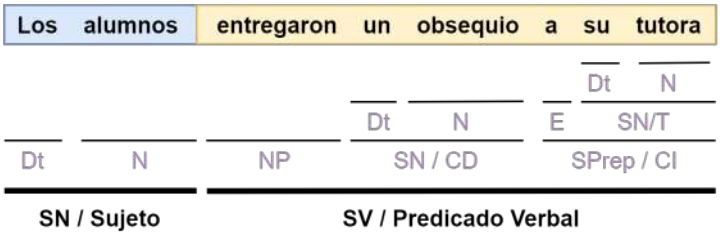


Figura A.1: Ejemplo de análisis sintáctico. *Fuente: Elaboración propia*

Se puede observar mediante el ejemplo anterior, como dentro del nivel sintáctico se identifican los componentes de una oración permitiendo conocer la función que cumple el mismo.

- Nivel semántico: este nivel es un complemento del anterior, en el análisis semántico se busca entender el significado de la oración. Las palabras pueden tener múltiples significados, la idea principal es identificar el significado apropiado por medio del contexto de la oración.

- Nivel discursivo: se encarga de trabajar con unidades de texto más grande que los niveles anteriores. Hace foco en el texto como un todo para alcanzar el significado haciendo conexiones entre las oraciones. Dos de los procedimientos que son realizados por este nivel son la resolución de anáforas, y el reconocimiento de la estructura del texto.

- Nivel pragmático: este nivel se ocupa del análisis de oraciones y cómo se usan en diferentes situaciones. Además, también cómo su significado cambia dependiendo de la situación.

A.2.2. Técnicas del NLP

A continuación, se describirán algunas de las técnicas más comunes utilizadas por los diferentes sistemas NLP para procesar texto escrito en lenguaje natural. Cada técnica de NLP puede implicar uno o varios niveles de procesamiento explicados en la sección anterior. [21]

Detección de oraciones

La detección de oraciones es una de las técnicas básicas correspondiente al nivel de procesamiento sintáctico. Si bien parece una tarea simple, detectar oraciones tiene ciertas dificultades a la hora de procesar títulos, abreviaturas, lista de elementos, y otros componentes que no siguen un patrón de texto plano.

Segmentación por palabras

Esta técnica, más conocida como analizador léxico o "tokenizer", pertenece al nivel léxico y consiste en la identificación de tokens, los cuales son unidades lingüísticas como palabras, puntuación, números, caracteres alfanuméricos, etc.

Una forma de identificar tokens en idiomas modernos que utilizan un sistema de escritura basado en el griego, como el inglés y otros idiomas europeos, se realiza delimitando espacios en blanco con límites de palabra, entre comillas, paréntesis y puntuación. El trato con las abreviaciones es similar a la detección de oraciones, ya que no existen normas universalmente aceptadas para muchas abreviaturas y acrónimos. El enfoque más adoptado para el reconocimiento de abreviaturas es mantener una lista de palabras recortadas reconocidas. Algunos ejemplos que pueden traer problemas son las direcciones de emails, palabras con apostrofes, URLs, ciudades, entre otros.

Etiquetado gramatical o Part of Speech-tagging

Una vez ejecutadas las dos técnicas previamente explicadas, se puede realizar el proceso de etiquetar las palabras según el rol que cumplen dentro de una oración. Este proceso de NLP se conoce como Etiquetado gramatical o Part-of-Speech (POS tagging).

Este proceso se encarga de asignar a cada una de las palabras de un texto su categoría gramatical de acuerdo a la definición de la misma o el contexto en que aparece, por ejemplo, sustantivo, adjetivo, adverbio, etc, como se puede observar en la Fig. A.2. Para ello es necesario establecer las relaciones de una palabra con sus adyacentes dentro de una frase o de un párrafo. Un mismo token puede tener múltiples etiquetas POS, pero solo una es válida dependiendo del contexto.

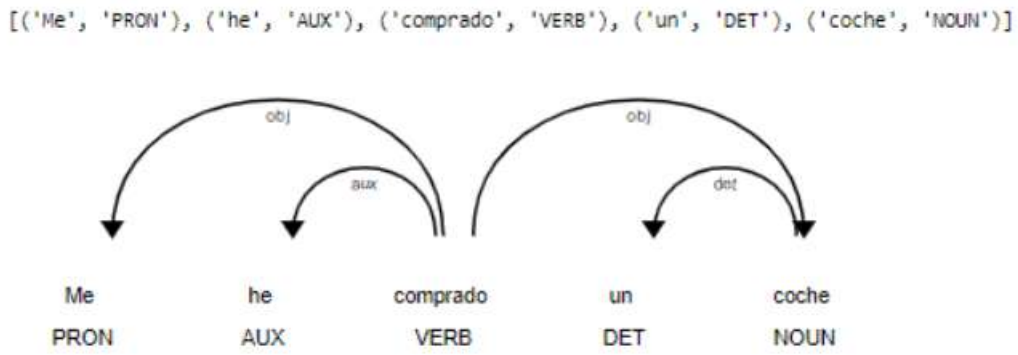


Figura A.2: Ejemplo de Part of Speech-tagging. *Fuente: Elaboración propia*

Segmentación morfológica

Otra técnica de NLP muy utilizada en el procesamiento de texto, es la segmentación morfológica o en morfemas. Un morfema es el fragmento mínimo capaz de expresar el significado de una palabra, es decir, la unidad significativa más pequeña de un idioma. Un morfema no es idéntico a la palabra, ya que este puede estar acompañado por ejemplo, por prefijos o sufijos, mientras que una palabra,

por definición, es independiente.

Esta técnica incrementa su complejidad en función del idioma. La identificación de morfemas permite el análisis en profundidad de una palabra dentro de un fragmento de texto, proporcionando información específica como puede ser el género, modo y tiempo, entre otras. Mediante el análisis realizado con esta técnica se puede ubicar de manera precisa cada palabra de cada oración.

Eliminación de "Stop Words"

La técnica "Stop Word" es utilizada para excluir palabras muy comunes que suelen tener poco valor para recuperar información que necesita el usuario. La cantidad de ocurrencias de una palabra en el texto determina si es o no una *stop word*. Dentro de este grupo se encuentran los artículos, los pronombres, las preposiciones, y las conjunciones, los cuales suelen aparecer con mucha frecuencia en los textos y no aportan valor. Esta técnica permite reducir el tamaño del texto para analizar, eliminando aproximadamente el 30 % o 40 % de dichas palabras. Además, se mejora la eficiencia, ya que la selección de palabras claves es más precisa.

Reconocimiento de Entidades Nombradas

La técnica NER, por sus siglas en inglés, busca y clasifica elementos del texto que pertenecen a categorías o entidades predefinidas como pueden ser nombres de personas, organizaciones, lugares, expresiones temporales, cantidades, porcentajes, etc.

A continuación en la Fig. A.3 se puede observar un ejemplo de etiquetado NER. La oración de ejemplo es: "Pedro compró 1000 acciones de las empresas Apple y Samsung en el año 2020." , donde ORG hace referencia a la entidad "Organización" y PER a la entidad "Persona".

Figura A.3: Ejemplo de Reconocimiento de Entidades Nombradas (NER). *Fuente: Elaboración propia*

Se han creado sistemas NER empleando técnicas basadas en gramáticas lingüísticas y modelos estadísticos, por ejemplo de aprendizaje de máquina. Los sistemas basados en gramáticas y elaborados con reglas manualmente obtienen generalmente una mejor precisión, pero a expensas de un menor recobrado y meses de trabajo de lingüistas computacionales experimentados. El NER estadístico comúnmente requieren un conjunto grande de datos de entrenamiento anotados manualmente. Se han propuesto métodos semisupervisados para evitar parte del esfuerzo de anotación.

Lematización

Esta técnica es un proceso lingüístico que consiste en obtener la forma base de una palabra. Por ejemplo, *decir* es el lema de *dije*, pero también de *diré* o *dijéramos*; *bello* es el lema de *bellas*; *mesa* es el lema de *mesas*.

La lematización puede realizarse automáticamente mediante programas de análisis morfológico. Existen diversos grados de lematización posible: se puede realizar una una lematización puramente morfológica, o bien emplear una lematización sintáctica que tenga en cuenta el contexto en el que aparece la palabra. Por ejemplo, en un análisis morfológico la palabra *ama* tendría dos lemas: el sustantivo *ama* y el verbo *amar*. Sin embargo, en un contexto sintáctico (es decir, en una oración), podemos desambiguarlo y optar por un único lema. Así, en *El ama de llaves abrió la puerta*, *ama* es sustantivo, mientras que en *María ama a Pedro*, *ama* es del verbo *amar*. Para poder hacer este tipo de lematización es necesario,

por lo tanto, hacer un análisis sintáctico.

Stemming

Las palabras están morfológicamente estructuradas en prefijos, sufijos y una raíz. La técnica de Stemming busca un concepto de la palabra eliminando tanto prefijos como sufijos y obteniendo la raíz. De esta manera, se efectúa una reducción de la palabra a su mínimo elemento con significado. Un término que es reducido a su común denominador simplifica la recuperación de documentos cuyas palabras tenga la misma raíz. De este modo "jugar" sería la palabra raíz para "jugó" o "jugaba". Esta aplicación puede apreciarse sensiblemente similar a la lematización.

A.3. Aprendizaje Automático

El Aprendizaje Automático o *Machine Learning* es una rama de la Inteligencia Artificial, el cual tiene por objetivo desarrollar técnicas que permitan a los programas de computadoras *aprender*. Se dice que un agente aprende cuando su desempeño mejora con la experiencia; es decir, cuando la habilidad no estaba presente en su genotipo o rasgos de nacimiento [22]. Posee una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica.

Actualmente, estas técnicas se encuentran en un completo auge. Algunas de las razones son:

- Por un lado la capacidad computacional de las computadoras ha ido aumentando siendo posible actualmente tratar problemas que antiguamente

no se podían tratar. Este aumento ha sido vertical (mejora de la capacidad individual de computación) y también horizontal (aumento de la capacidad de cómputo al trabajar con varios ordenadores al mismo tiempo usando el paradigma del *Big Data*).

- Por otro lado la revolución y masificación de los datos, motivada por la digitalización ha supuesto un aumento los de datos que pueden ser procesados y modelizados para obtener conocimiento de ellos.

En muchas ocasiones el campo de actuación del aprendizaje automático se solapa con el de la estadística inferencial, ya que las dos disciplinas se basan en el análisis de datos. Sin embargo, el aprendizaje automático incorpora las preocupaciones de la complejidad computacional de los problemas. El aprendizaje automático también está estrechamente relacionado con el reconocimiento de patrones. Principalmente sus algoritmos o métodos extraen patrones de los datos [23].

A.3.1. Tipos de Aprendizaje Automático

Los algoritmos de aprendizaje automático pueden clasificarse de 4 formas distintas: aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semisupervisado y aprendizaje por refuerzo.

Aprendizaje supervisado

En este tipo de aprendizaje se parte de un conocimiento a priori. El objetivo es, mediante datos de entrenamiento, deducir una función matemática que haga lo mejor posible el mapeo entre unas entradas y una salida. Los datos de entrenamiento constan de tuplas (X,Y), siendo X las variables que predicen una determinada salida Y.

La variable a predecir Y puede ser una variable cuantitativa (como en el caso de problemas de regresión) o cualitativa (como en el caso de problemas de clasificación).

Aprendizaje no supervisado

Al contrario que en el aprendizaje supervisado, en este caso no existe conocimiento a priori. Aquí ya no se tienen tuplas (X,Y), simplemente se tienen conjunto de datos X. El objetivo del aprendizaje no supervisado es modelizar la estructura o distribución de los datos para aprender sobre ellos. Sirve tanto para entender como para resumir un conjunto de datos. Se llama no supervisado porque, contrariamente al supervisado, tiende a ser más subjetivo ya que no tiene respuestas correctas. Los algoritmos sirven para descubrir y presentar estructuras relevantes en los datos.

Aprendizaje semisupervisado

El aprendizaje semisupervisado se encuentra en un punto intermedio entre el aprendizaje supervisado y el no supervisado. Lo que se posee ahora son datos etiquetados como no etiquetados, es decir, además de tener tuplas (X,Y), se tienen conjunto de datos sólo de X de los que no se sabe su respuesta Y.

El desafío se encuentra en combinar datos etiquetados y no etiquetados para construir un modelo supervisado que sea mejor ya que:

1. La cantidad de datos etiquetados se puede aumentar, lo que generalmente mejora los resultados de los modelos.

2. Hay un alto coste de etiquetar los datos X sin etiquetas.

Supone que los datos etiquetados y no etiquetados provienen de la misma

distribución. Por otro lado, puede existir un sesgo en la elección de datos no etiquetados.

Entre los métodos de aprendizaje semisupervisado se encuentran:

- *Self-training*
- *Co-training*
- *Assemble*
- *Re-Weighting*

Aprendizaje por refuerzo

En los casos de aprendizaje supervisado se cuenta con tuplas (X,Y). Sin embargo, el caso de aprendizaje por refuerzo lo que tenemos son problemas no supervisados que sólo reciben re-alimentaciones o refuerzos (por ejemplo, gana o pierde).

Se sustituye la información supervisada (Y) por información del tipo acción/-reacción. El objetivo en el aprendizaje por refuerzo es aprender a mapear situaciones de acciones para maximizar una cierta función de recompensa.

En estos problemas un agente aprende por prueba y error en un ambiente dinámico e incierto. En cada interacción el agente recibe como entrada un indicador de estado actual y selecciona una determinada acción que maximice una función de refuerzo o recompensa a largo plazo. Este proceso de decisión secuencial se puede caracterizar como un proceso de Markov.

A.4. Full-Text Search

En el ámbito de recuperación de información, se denomina Full-Text Search al conjunto de técnicas diseñadas para realizar búsquedas de texto en una base

de datos de documentos de texto.

Muchos sitios web y aplicaciones que se utilizan en la vida cotidiana (como por ejemplo motores de búsqueda, procesadores de texto, etc.) proveen funcionalidades que utilizan Full-Text Search para la búsqueda de textos.

A.4.1. Indexamiento

Las técnicas de Full-Text Search se apoyan en la creación de índices de texto. Cuando se desea realizar una búsqueda de texto en un numero pequeño de documentos, es posible para el motor de búsqueda realizar un escaneo de el contenido de los documentos con cada consulta. Esta estrategia, llamada *serial scanning*, es lo que algunas herramientas como el comando `grep` de linux realizan cuando ejecutan una búsqueda. Sin embargo, cuando la cantidad de documentos para buscar es potencialmente grande, o la cantidad de consultas para realizar es sustancial, el problema de la búsqueda de Full-Text Search a menudo se divide en dos tareas: indexación y búsqueda.

La etapa de indexación escaneará el texto de todos los documentos y creará una lista de términos de búsqueda (a menudo llamado índice). En la etapa de búsqueda, al realizar una consulta específica, solo se hace referencia al índice, en lugar del texto de los documentos originales. [24]

El indexador generará una entrada en el índice para cada término o palabra que se encuentre en el documento y posiblemente almacene su posición relativa dentro del mismo. Algunas implementaciones realizan un parseo del documento para obtener los tokens, luego producen una transformación de tokens a lexemas e ignoran las *stop words* (como las palabras “en”, “y”) que son comunes y no tienen el significado suficiente para ser útiles en la búsqueda.

Cómo funciona la búsqueda mediante índice invertido

Por ejemplo, se posee un buscador que obtiene los términos de búsqueda a partir de 2 documentos que contienen diferentes textos.

- Documento 1: "Salvador Dalí nació en Figueras."
- Documento 2: "Figueras es una ciudad de Gerona."

El buscador creará un índice con las diferentes palabras que aparecen, indicando el documento en el que aparece, como se puede ver en la tabla A.1 . Generalmente, se excluirán algunas palabras stop words ("en", "de", ...), como se mencionó anteriormente.

Actualmente, algunos sistemas de gestión de bases de datos relacionales (SGBD) cuentan con soporte para la indexación y recuperación de documentos de texto.

ID	Palabra	Documento N°
1	Salvador	1
2	Dalí	1
3	nació	1
4	Figueras	1,2
5	ciudad	2
6	Gerona	2

Tabla A.1: Índice Invertido

A.4.2. Arquitectura básica de un SRI

Se puede decir que la recuperación de información intenta resolver el problema de "encontrar y rankear documentos relevantes que satisfagan la necesidad de información de un usuario, expresada en un determinado lenguaje de consulta"

[25]. Sin embargo, existe un problema que dificulta sobremanera esta tarea y consiste en poder "compatibilizar" y comparar el lenguaje en que está expresada tal necesidad de información y el lenguaje de los documentos.

Para cumplir con sus objetivos, un SRI debe realizar algunas tareas básicas, las cuales se encuentran – fundamentalmente – planteadas en cuestiones computacionales, a saber [25]:

- Representación lógica de los documentos y – opcionalmente – almacenamiento del original. Algunos sistemas solo almacenan porciones de los documentos y otros lo hacen de manera completa.
- Representación de la necesidad de información del usuario en forma de consulta.
- Evaluación de los documentos respecto de una consulta para establecer la relevancia de cada uno.
- Ranking de los documentos considerados relevantes para formar el "conjunto solución" o respuesta.
- Presentación de la respuesta al usuario.
- Retroalimentación o refinamiento de las consultas (para aumentar la calidad de la respuesta)

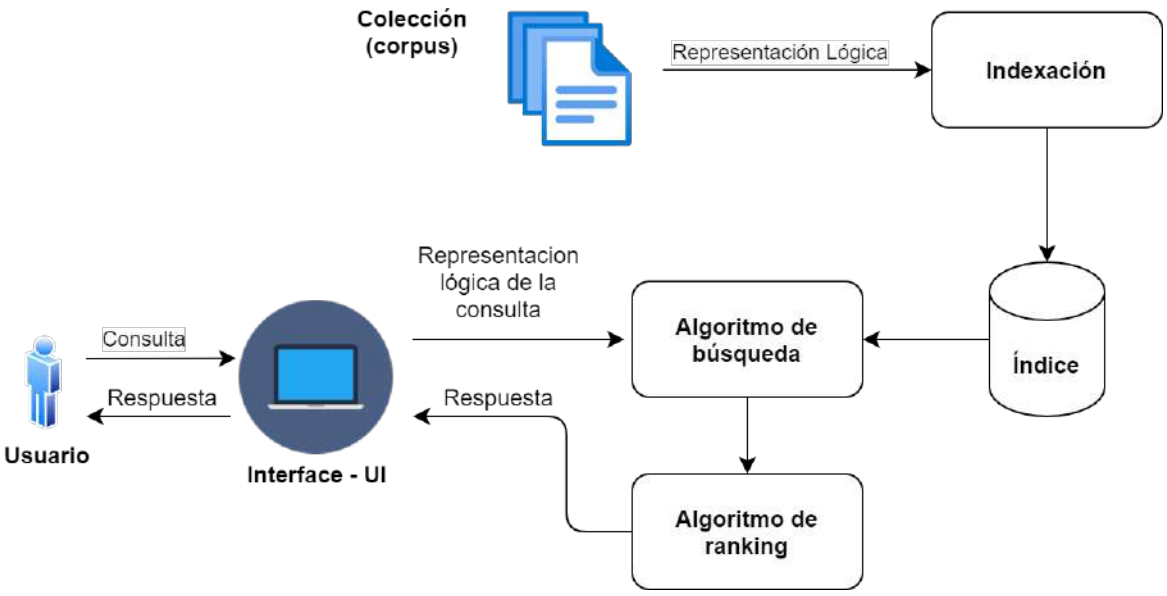


Figura A.4: Arquitectura básica de un SRI. *Fuente: Elaboración propia*

En la Fig. A.4 se puede apreciar con mayor detalle la arquitectura básica de un SRI, el tratamiento de los documentos y la interacción con el usuario. Como se puede observar, se parte de un conjunto de documentos de texto, los cuales están compuestos por sucesiones de palabras que forman estructuras gramaticales (por ejemplo, oraciones y párrafos). Tales documentos están escritos en lenguaje natural y expresan ideas de su autor sobre un determinado tema. El conjunto de todos los documentos con los que se trata y sobre los que se deben realizar operaciones de RI se denomina *corpus*, *colección* o *base de datos textual o documental*.

Para poder realizar operaciones sobre un corpus, es necesario obtener primero una representación lógica de todos sus documentos, la cual puede consistir en un conjunto de términos, frases u otras unidades (sintácticas o semánticas) que permitan – de alguna manera – caracterizarlos. Por ejemplo, la representación de los documentos mediante un conjunto de sus términos se la conoce como "bolsa de palabras" (*bag of words*).

A partir de la representación lógica de los documentos, existe un proceso (indexación) que llevará a cabo la construcción de estructuras de datos (normalmente denominadas índices) que la almacene. Estas estructuras darán luego soporte para búsquedas eficientes.

El algoritmo de búsqueda acepta como entrada una expresión de consulta o *query* de un usuario y verificará en el índice cuáles documentos pueden satisfacerlo. Luego, un algoritmo de ranking determinará la relevancia de cada documento y retornará una lista con la respuesta. Se establece que el primer ítem de dicha lista corresponde al documento más relevante respecto de a la consulta y así sucesivamente en orden decreciente.

Por ultimo, la interfaz de usuario permite que éste especifique la consulta mediante una expresión escrita en un lenguaje preestablecido y además, sirve para mostrar las respuestas retornadas por el sistema.

Apéndice B

Protocolo de entrevista

En la fase de elicitación de requerimientos, descripta en el Capítulo 4 , se llevaron a cabo diferentes preguntas a los profesionales del Derecho e integrantes del juzgado. A continuación se presentan las principales.

1. ¿Cómo se resuelve actualmente el proceso de consulta de jurisprudencia?
2. ¿Qué pasos se realizan?
3. ¿Qué aspectos se suelen buscar manualmente?
4. ¿Cuánto tiempo aproximado se tarda al consultar la información requerida manualmente?
5. ¿Cómo es utilizada la información posteriormente, una vez obtenida?
6. ¿Qué funcionalidades debería incluir el sistema?
7. ¿Qué tipo de filtros serían necesarios en las búsquedas?
8. ¿Cuáles son las categorías/temáticas que el sistema debería clasificar/organizar la información?

9. ¿Cuál es el volumen de archivos que cuenta el juzgado para consultar jurisprudencia?

10. ¿Cuáles son las secciones mas importantes de una sentencia?

11. ¿Cómo están compuestas las "fichas" mencionadas en la entrevista inicial? ¿Qué aspectos relevantes presenta?

12. ¿Cada qué periodo de tiempo se van agregando nuevos documentos a consultar?

13. ¿Cómo está compuesto el juzgado?

14. ¿Qué personal del juzgado utilizaría el sistema?

15. ¿Cuántas personas utilizarían el sistema?

16. ¿Todos utilizarían el sistema de igual forma o existen restricciones?

17. ¿Con qué sistemas informáticos trabaja el juzgado?

18. ¿Qué hardware (equipos, servidores) posee el juzgado?

19. ¿De dónde provienen los documentos/archivos a indexar? ¿En qué formato?

Apéndice C

Casos de uso

Los casos de uso son una técnica para la especificación de requisitos funcionales propuesta inicialmente por Ivar Jacobson. Modelan la funcionalidad del sistema tal como la perciben los agentes externos, denominados actores, que interactúan con el sistema desde un punto de vista particular. Sus componentes principales son:

- Sujeto: sistema que se modela.
- Casos de uso: unidades funcionales completas.
- Actores: entidades externas que interactúan con el sistema.

En la fase de elicitación de requerimientos, descripta en el Capítulo 4, se modelaron los siguientes casos de uso del sistema:

Tabla C.1: Caso de uso N° 1 - Login

Caso de uso N° 1	Login
Descripción	El caso de uso permite el ingreso al sistema introduciendo nombre de usuario y contraseña.
Actor	Usuario Personal o Administrador.
Precondiciones	El usuario debe estar registrado en el sistema.
Flujo Normal	1. El usuario ingresa a la url /login. 2. El usuario ingresa su nombre de usuario. 3. El usuario ingresa su contraseña. 4. El usuario selecciona la opción “Ingresar”. 5. El sistema comprueba las credenciales correctas del usuario. 6. El sistema redirige al usuario a la página de Inicio.
Flujo alterno	1. El usuario ingresa a la url /login. 2. El usuario ingresa su nombre de usuario. 3. El usuario ingresa su contraseña (errónea). 4. El usuario selecciona la opción “Ingresar”. 5. El sistema comprueba las credenciales del usuario. 6. El sistema indica que las credenciales de ingreso no son correctas, restringiendo el ingreso al sistema.
Resultado	El usuario se encuentra logueado y puede utilizar las funcionalidades del sistema según su perfil.
Prioridad	Alta.

Tabla C.2: Caso de uso N° 2 - Búsqueda de sentencia/s

Caso de uso N° 2	Búsqueda de sentencia/s
Descripción	El caso de uso permite la búsqueda de sentencias de interés a través de dos formas: búsqueda simple o búsqueda avanzada. La búsqueda simple implica buscar por términos, palabras o frases que se incluyan en el contenido de una sentencia. La búsqueda avanzada adiciona a la búsqueda simple la posibilidad de agregar criterios adicionales a la búsqueda como tipo de materia, tipo de resolución, instancia, temática, rango de fechas de la resolución, entre otros.
Actor	Usuario Personal o Administrador.
Precondiciones	El usuario debe estar logueado en el sistema. El usuario debe tener perfil de Administrador o Personal.
Flujo Normal	1. El usuario ingresa al apartado buscador. 2. El usuario ingresa un término de búsqueda. Por ejemplo “estafa”. 3. El usuario selecciona la opción “Buscar”. 4. El sistema busca en la base de datos aquellas sentencias que contengan el término “estafa” en sus campos. 5. El sistema redirige al usuario a la pantalla de Resultados de la búsqueda, informando la cantidad de resultados y opciones de ordenamiento.
Flujo alterno	1. El usuario ingresa al apartado buscador. 2. El usuario ingresa un término de búsqueda. Por ejemplo “estafa”. 3. El usuario despliega la opción “Búsqueda Avanzada”. 4. El usuario selecciona algunos filtros de búsqueda, como rango de fechas, materia e instancia. 5. El usuario selecciona la opción “Buscar”. 6. El sistema busca en la base de datos aquellas sentencias que contengan el término “estafa” en sus campos, se encuentren en el rango de fechas indicado y pertenezcan a la materia e instancia indicadas. 7. El sistema redirige al usuario a la pantalla de Resultados de la búsqueda, informando la cantidad de resultados y opciones de ordenamiento.
Resultado	El usuario visualiza los resultados de su búsqueda, pudiendo elegir por alguno de su interés.
Prioridad	Alta.

Tabla C.3: Caso de uso N° 3 - Registrar un nuevo usuario

Caso de uso N° 3	Registrar un nuevo usuario
Descripción	El caso de uso permite el registro de un nuevo usuario al sistema.
Actor	Administrador.
Precondiciones	El actor debe tener perfil de administrador. El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor ingresa a la url /admin. 2. El actor selecciona la opción “Agregar usuario” en el apartado “Gestión de usuarios”. 3. El actor completa el formulario con los datos requeridos del usuario. 4. El usuario selecciona la opción “Guardar”. 5. El sistema comprueba el formulario ingresado. 6. El sistema muestra un aviso de “Usuario creado exitosamente”.
Flujo alterno	Usuario ya existente: 1. El actor ingresa a la url /admin. 2. El actor selecciona la opción “Agregar usuario” en el apartado “Gestión de usuarios”. 3. El actor completa el formulario con los datos requeridos (nombre de usuario erróneo) del usuario. 4. El usuario selecciona la opción “Guardar”. 5. El sistema comprueba el formulario ingresado. 6. El sistema muestra un aviso de error de “Usuario ya existente”. Contraseñas no coincidentes: 1. El actor ingresa a la url /admin. 2. El actor selecciona la opción “Agregar usuario” en el apartado “Gestión de usuarios”. 3. El actor completa el formulario con los datos requeridos, completa los campos contraseña y confirmar contraseña , pero estos no coinciden. 4. El usuario selecciona la opción “Guardar”. 5. El sistema comprueba el formulario ingresado. 6. El sistema muestra un aviso de error de “Deben coincidir las contraseñas” y no registra el nuevo usuario.
Resultado	El usuario se encuentra registrado en el sistema y se encuentra habilitado para su uso.
Prioridad	Alta.

Tabla C.4: Caso de uso N° 4 - Agregar una nueva sentencia al sistema

Caso de uso N° 4	Agregar una nueva sentencia al sistema
Descripción	El caso de uso permite indexar una nueva sentencia al sistema, estando disponible para su posterior consulta.
Actor	Administrador.
Precondiciones	El actor debe tener perfil de administrador. El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor ingresa a la url /admin. 2. El actor ingresa al apartado sentencias y selecciona la opción “Agregar sentencia”. 3. El actor completa los campos requeridos de la sentencia con datos válidos. 4. El actor sube el archivo digital de la sentencia a través de la opción “Subir archivo”. 5. El actor selecciona la opción “Guardar”. 6. El sistema comprueba los datos del formulario y agrega la nueva sentencia en la base de datos. 7. El sistema muestra un mensaje de carga de sentencia exitosa.
Flujo alterno	3. El actor completa con datos erróneos o campos requeridos faltantes. 4. El actor sube adjunto un archivo de formato no compatible. 6. El sistema comprueba los datos del formulario y al ser erróneos no carga la nueva sentencia al sistema. 7. El sistema muestra un mensaje de error indicando al usuario los campos erróneos.
Resultado	Una nueva sentencia ha sido cargada e indexada en el sistema, pudiendo ser consultada posteriormente por cualquier usuario del sistema.
Prioridad	Alta.

Tabla C.5: Caso de uso N° 5 - Modificar datos de una sentencia

Caso de uso N° 5	Modificar datos de una sentencia
Descripción	El caso de uso permite modificar los datos de una sentencia existente del sistema, como materia, instancia, temática, tipo de resolución, fecha de resolución, entre otros.
Actor	Administrador.
Precondiciones	El actor debe tener perfil de administrador. El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor ingresa a la url /admin. 2. El actor ingresa al apartado sentencias. 3. El actor visualiza las sentencias y busca la sentencia que desea modificar. 4. El actor selecciona la opción “Cambiar”. 5. El actor modifica los datos de interés de la sentencia. 6. El actor selecciona la opción “Guardar”. 7. El sistema comprueba los datos del formulario y modifica la sentencia existente en la base de datos. 8. El sistema muestra un mensaje de modificación de sentencia exitosa.
Flujo alterno	5. El actor modifica la sentencia con datos erróneos o campos requeridos faltantes. El actor sube adjunto un archivo de formato no compatible. 7. El sistema comprueba los datos del formulario y al ser erróneos no modifica la nueva sentencia al sistema. 8. El sistema muestra un mensaje de error indicando al usuario los campos erróneos.
Resultado	Una sentencia existente en el sistema ha sido modificada, reflejando sus cambios para posteriores búsquedas.
Prioridad	Alta.

Tabla C.6: Caso de uso N° 6 - Eliminar una sentencia del sistema

Caso de uso N° 6	Eliminar una sentencia del sistema
Descripción	El caso de uso permite eliminar una sentencia existente del sistema.
Actor	Administrador.
Precondiciones	El actor debe tener perfil de administrador. El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor ingresa a la url /admin. 2. El actor ingresa al apartado sentencias. 3. El actor visualiza las sentencias y busca la sentencia que desea eliminar. 4. El actor selecciona la opción Eliminar. 5. El sistema solicita la confirmación de eliminado para la sentencia seleccionada. 6. El actor confirma la acción. 7. El sistema muestra un mensaje de sentencia eliminada exitosamente.
Flujo alterno	–
Resultado	Una sentencia existente en el sistema ha sido eliminada, siendo ésta ya no disponible para posteriores búsquedas.
Prioridad	Alta.

Tabla C.7: Caso de uso N° 7 - Filtrar resultado de una búsqueda

Caso de uso N° 7	Filtrar resultado de una búsqueda
Descripción	El caso de uso permite filtrar los resultados de una búsqueda de jurisprudencia, refinando por criterios y filtros.
Actor	Usuario Administrador o Personal.
Precondiciones	El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor busca una sentencia por términos o frases. 2. El actor visualiza los resultados de las sentencias. 3. El actor visualiza la sección de filtros a la izquierda de la pantalla. 4. El actor selecciona los filtros por los que desea refinar la búsqueda. 5. El actor selecciona la opción Filtrar. 6. El sistema realiza el nuevo filtrado de resultados mostrando una pantalla de carga. 7. El sistema muestra los nuevos resultados de la búsqueda junto a los filtros aplicados, con opciones de ordenamiento y número de resultados por página.
Flujo alterno	1. El actor busca una sentencia por términos o frases. 2. El sistema no encuentra resultados para la búsqueda.
Resultado	La búsqueda del actor ha sido filtrada refinando los resultados.
Prioridad	Alta.

Tabla C.8: Caso de uso N° 8 - Visualizar una sentencia

Caso de uso N° 8	Visualizar una sentencia
Descripción	El caso de uso permite visualizar una sentencia, con sus datos completos, categorías y temáticas.
Actor	Usuario Administrador o Personal.
Precondiciones	El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor busca una sentencia por términos o frases. 2. El actor visualiza los resultados de las sentencias. 3. El actor selecciona una sentencia de su interés de la lista de resultados. 4. El actor visualiza una sección sobre los datos de la sentencia: título, número de causa, fecha de la sentencia, órgano perteneciente, instancia, temáticas, entre otros. 5. El actor visualiza el cuerpo de la sentencia, con los términos de búsqueda resaltados en color. 6. El actor puede introducir términos a buscar dentro del cuerpo de la sentencia, presionando posteriormente el botón en forma de lupa.
Flujo alterno	–
Resultado	Visualización en detalle de una sentencia de interés.
Prioridad	Alta.

Tabla C.9: Caso de uso N° 9 - Agregar una sentencia a favoritos

Caso de uso N° 9	Agregar una sentencia a favoritos
Descripción	El caso de uso permite agregar una sentencia a la colección de favoritos del usuario logueado en el sistema.
Actor	Usuario Administrador o Personal.
Precondiciones	El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor busca una sentencia por términos o frases. 2. El actor visualiza los resultados de las sentencias. 3. El actor selecciona una sentencia de su interés de la lista de resultados. 4. El actor selecciona la opción “Agregar a favoritos”. 5. La sentencia es agregada a favoritos. 6. El actor ingresa a la sección Perfil - “Mis sentencias”. 7. La sentencia es listada en primer lugar (por orden y fecha de favoritos) junto con las demás sentencias.
Flujo alterno	–
Resultado	Sentencia agregada a los favoritos del usuario.
Prioridad	Media.

Tabla C.10: Caso de uso N° 10 - Descargar una sentencia

Caso de uso N° 10	Descargar una sentencia
Descripción	El caso de uso permite descargar el archivo original de una sentencia del sistema.
Actor	Usuario Administrador o Personal.
Precondiciones	El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor busca una sentencia por términos o frases o bien encuentra una sentencia a través de la sección “Índice”. 2. El actor visualiza los resultados de las sentencias. 3. El actor selecciona una sentencia de su interés de la lista de resultados. 4. El actor selecciona la opción “Descargar archivo original”. 5. El actor elige dónde guardar el archivo y es descargado a través del navegador.
Flujo alterno	4. Por problemas de conectividad el archivo no ha sido descargado, intentar nuevamente.
Resultado	Archivo de la sentencia descargado en la computadora del actor.
Prioridad	Media.

Tabla C.11: Caso de uso N° 11 - Logout

Caso de uso N° 11	Logout
Descripción	El caso de uso permite desloguear al usuario del sistema.
Actor	Usuario Administrador o Personal.
Precondiciones	El actor debe estar logueado en el sistema.
Flujo Normal	1. El actor ingresa al apartado Perfil. 2. El actor selecciona la opción “Cerrar Sesión”. 3. El sistema redirige al actor a la pantalla de Login del sistema.
Flujo alterno	–
Resultado	Usuario deslogueado del sistema. Para utilizarlo, debe iniciar sesión nuevamente.
Prioridad	Alta.

Apéndice D

Diagrama de Gantt

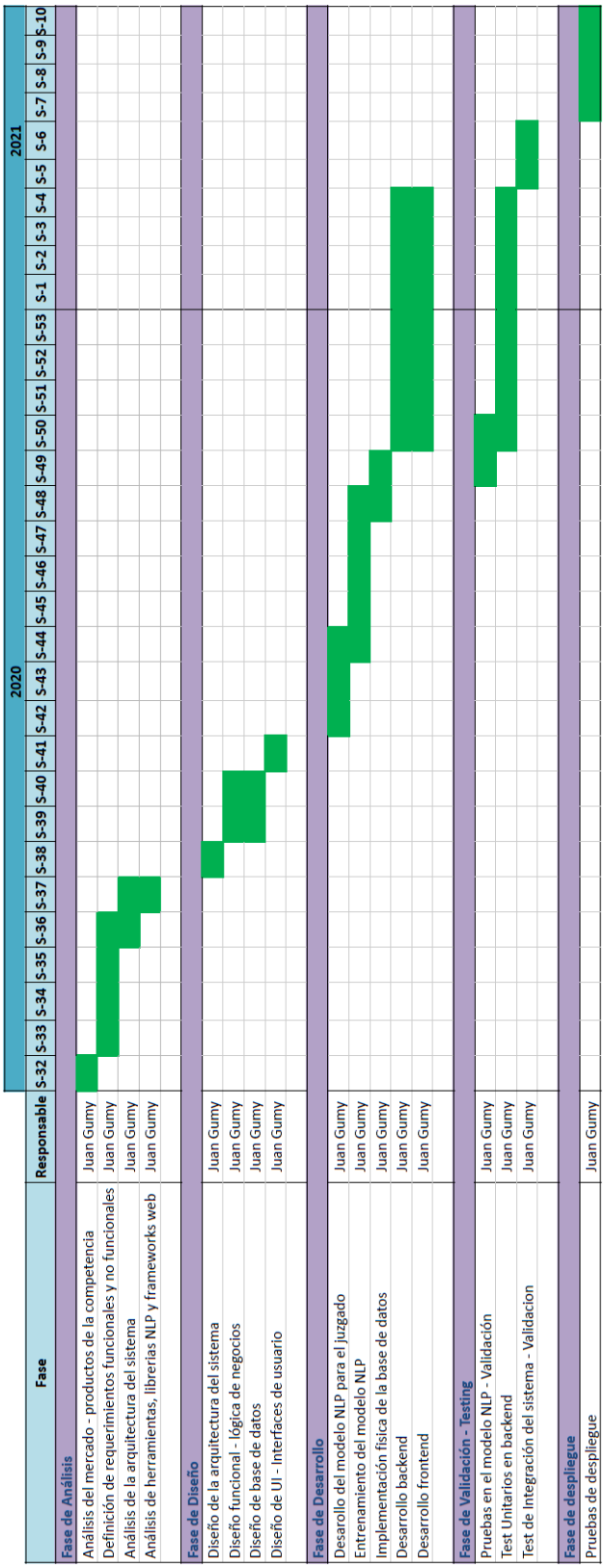


Figura D.1: Diagrama de Gantt planificacion original *Fuente: Elaboración propia*