UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Aplicación de análisis de texto y aprendizaje automático en textos normados

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Julio César Romero Pérez

DIRECTOR DE TESIS

Dr. Daniel Trejo Medina

Ciudad Universitaria, Cd. Mx., 2025

VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
COMITÉ DE TITULACIÓN

Designación de sinodales de Examen Profesional

A los señores profesores: Firma y fecha de acuse

Presidente:	**DR. GERARDO EUGENIO SIERRA MARTINEZ**	_____
Vocal:	**DR. DANIEL TREJO MEDINA**	_____
Secretario:	**DR. ISMAEL EVERARDO BARCENAS PATIÑO**	_____
1er. Suplente:	**MTRA. EDITH TAPIA RANGEL**	_____
2do. Suplente:	**M.I. JORGE LEON MARTINEZ**	_____

Conforme a la encomienda que hace el Director de la Facultad a este Comité de Titulación para la integración de jurados, me permito informar a ustedes que han sido designados sinodales del Examen Profesional de: **ROMERO PEREZ JULIO CESAR**, registrado con número de cuenta **412003714** en la carrera de **INGENIERÍA EN COMPUTACIÓN**; quien ha concluido el desarrollo del tema que le fue autorizado.

Ruego a ustedes se sirvan revisar el trabajo adjunto y manifestar a la Dirección de la Facultad, si es el caso, la aceptación mediante la firma en el oficio FEX-2 conforme a lo indicado en el siguiente párrafo. No omito mencionar que **el trabajo escrito deberá contener la *Constancia de verificación de documento escrito* debidamente sellada por la Coordinación del Sistema de Bibliotecas de la Facultad de Ingeniería (CSB-FI)**, es decir, el trabajo escrito completo con las debidas señalizaciones en las secciones en las que se encuentren coincidencias, así como un "informe de originalidad" con porcentajes de coincidencias que se encuentra al final del trabajo escrito.

Por indicaciones del Sr. Director, con el fin de asegurar el pronto cumplimiento de las disposiciones normativas correspondientes y de no afectar innecesariamente los tiempos de titulación, les ruego tomar en consideración que para lo anterior cuentan ustedes con un plazo máximo de **cinco días hábiles** contados a partir del momento en que ustedes **acusen recibo de esta notificación.** Si transcurrido este plazo el interesado no tuviera observaciones de su parte, se entendería que el trabajo ha sido aprobado, por lo que deberán **firmar el oficio de aceptación del trabajo escrito.**

Doy a ustedes las más cumplidas gracias por su atención y les reitero las seguridades de mi consideración más distinguida.

Atentamente,
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Ciudad Universitaria, Cd. Mx. a 28 de noviembre de 2024.
EL PRESIDENTE DEL COMITÉ

M.C. ALEJANDRO VELÁZQUEZ MENA

FI-DIE
Lic. Ngélica Gutiérrez Vázquez
Coordinación de Titulación y Servicio Social
angelica_6@comunidad.unam.mx

NOTA: Por instrucciones del Director, deberán entregarse juntos y al mismo tiempo, el presente oficio FEX-1 y su correspondiente oficio FEX-2.

COPIA DEL ALUMNO

FEX-1
AGV

Resumen

El análisis e interpretación de textos normados es una tarea crítica y laboriosa en el ámbito legal, los métodos tradicionales de revisión manual pueden llegar a ser lentos e inexactos. El objetivo de esta tesis es desarrollar e implementar un sistema de análisis de texto capaz de procesar grandes volúmenes de textos normados, utilizando técnicas de aprendizaje automático, grandes modelos de lenguaje y procesamiento de lenguaje natural (PLN). El sistema combina técnicas de preprocesamiento de texto, procesamiento de lenguaje natural, clasificación automática, generación de resúmenes e indexado de texto para facilitar el análisis y la interpretación de los textos normados.

Utilizando datos históricos obtenidos a partir del sitio de la Suprema Corte de Justicia de la Nación, se procesaron más de mil textos normados de los años 2022 y 2023 y se pudo generar un modelo de clasificación automática que fuera capaz de discernir de manera correcta la citación de una tesis jurisprudencial, así como de una ley nombrada.

De acuerdo con los resultados aportados podemos afirmar que el sistema desarrollado permite disminuir el tiempo empleado para analizar, encontrar y comprender textos normativos en comparación con los métodos manuales, este sistema añade una herramienta más en el proceso de análisis e investigación jurídica.

Abstract

The analysis and interpretation of legal documents is a critical and laborious task of the Mexican legal system. Traditional manual revision methods can be very slow and imprecise. The main objective of this document is to develop and implement a text analysis system able to handle high volumes of legal unstructured documents using PLN techniques, Large Language models (LLMs), machine learning models using supervised learning algorithms, automatic summary generation, and a search index so the user can search, analyze and interpret Mexican normative texts.

To develop the automatic classification model, I downloaded over one thousand legal normative documents from 2022 and 2023 from the most important Mexican legal institution, the "Suprema Corte de Justicia de la Nación." This data enabled me to create an automatic classification model capable of correctly discerning the citation of a jurisprudential thesis and a referenced law inside the normative texts.

According to this thesis's results, the developed system reduces the time required to analyze, search, and understand normative texts compared to manual traditional methods. This system adds a new tool to legal analysis and research.

Agradecimientos

Agradezco a mi familia, que siempre me apoyo en momentos muy difíciles, sin ellos nunca pude haber terminado esta gran empresa, que es convertirme en ingeniero. A mi abuelo, a mi abuela principalmente. A mis profesores sin los que no hubiera podido llegar tan lejos en mi experiencia profesional, al Doctor Daniel Trejo Medina por ser más allá de un excelente profesor, alguien a quien considero mi mentor y amigo.

Gracias a la Universidad Nacional Autónoma de México por existir, y darnos la oportunidad de tener acceso a una educación de excelencia y abrirnos las puertas al mundo.

A la Facultad de Ingeniería por darnos ese lugar en el que podemos ampliar nuestros conocimientos, en ella pasé grandes años de mi vida.

Contenido

Ilustraciones

Introducción

La era digital ha impulsado un crecimiento exponencial en la generación y almacenamiento de información textual, abarcando desde documentos legales y empresariales hasta mensajes en redes sociales y correos electrónicos (Trejo, 2019a). Esta cantidad de datos plantea la necesidad de una gestión eficiente y precisa, lo cual se vuelve fundamental en ámbitos tan variados como la industria, la administración pública y la investigación académica. En este contexto, el análisis de texto y el aprendizaje automático han emergido como herramientas críticas, permitiendo a los sistemas informáticos procesar grandes volúmenes de información de manera eficiente y precisa.

La presente tesis tiene como objetivo el presentar un sistema de análisis de texto que utilice técnicas de aprendizaje automático, enfocado específicamente en textos normados. Este enfoque responde a la necesidad de automatizar, al menos en parte, la interpretación, clasificación y análisis de grandes volúmenes de estos textos, los cuales, debido a sus características estructurales y lingüísticas, requieren de metodologías precisas para la identificación de información relevante y la reducción de errores y sesgos comunes en el procesamiento manual (Sierra, 2009).

Dentro de la carrera de ingeniería en computación, la elaboración de tesis sigue siendo un desafío por la orientación técnica de la disciplina, que rara vez incluye la creación de trabajos escritos de divulgación (Muñoz, 1998, pp. 3-5). Sin embargo, esta tesis busca aplicar el conocimiento técnico adquirido durante la licenciatura, así como el criterio profesional y analítico, en el desarrollo de un sistema que contribuya a resolver un área de oportunidad actual de procesamiento textual, así como que aporte nuevos conocimientos y metodologías en el análisis de documentos normados, un área de creciente interés tanto en la academia como en la práctica profesional.

Este trabajo pretende atender problemas específicos de interpretación y procesamiento de textos normados en áreas como el derecho y la administración pública. Los textos normados, como leyes, sentencias, resoluciones y otros

documentos legales, presentan convenciones lingüísticas y estructurales que los diferencian de otros tipos de textos y que, por tanto, requieren soluciones especializadas en procesamiento de lenguaje natural (PLN) y aprendizaje automático (Trejo, 2019b). Así, esta tesis plantea que el empleo de un sistema automatizado de análisis de texto para documentos normados permitirá mejorar la eficiencia y precisión en el procesamiento de dichos textos par que puedan ser hallados más fácilmente, demostrando la viabilidad de estos enfoques para superar las limitaciones de los métodos tradicionales de análisis manual.

El lenguaje es el medio de comunicación más eficaz que el ser humano utiliza para expresarse y establecer relaciones con otros individuos, el conocimiento humano ha evolucionado, dando origen a disciplinas, ciencias y profesiones que se especializan en estudiar temas específicos, permitiendo avances que beneficien a la sociedad, este es el caso del Derecho, en el cual aunque el lenguaje natural es utilizado en la comunicación dentro de todas las ciencias, el Derecho ha empleado palabras específicas, asignándoles un significado particular acorde con su propósito operativo.

Así surgen los lenguajes de especialidad, los cuales son empleados por profesionales de cada área, y el contexto profesional establece los significados específicos de los términos (Muñoz, 1998).

Es indispensable contar con métodos que permitan gestionar toda la información escrita que se genera y que se ha generado, de acuerdo con Harris (1954) esta tarea es imposible de realizar únicamente con recursos humanos, por lo que es necesario automatizar el procesamiento de la lengua escrita; es aquí donde entra en juego el procesamiento del lenguaje natural (PLN), una rama de la inteligencia artificial (IA) que desarrolla modelos computacionales para facilitar la comunicación entre personas y máquinas.

En las dos últimas décadas, el área de PLN ha crecido y desarrollado gracias a las representaciones distribucionales, las cuales permiten una codificación semántica en espacios vectoriales densos (Pimentel, 2022), donde entre las ventajas de estos

sistemas es que no requieren datos etiquetados, siendo suficiente el uso de texto plano, el cual está disponible en grandes volúmenes. Estos sistemas, al igual que sus derivados, se fundamentan en la hipótesis distribucional de Harris (1954), que esencialmente establece que las palabras que tienen significados similares tienden a aparecer en contextos similares.

Objetivo general

Programar un sistema de análisis de texto, que emplee aprendizaje automático diseñado específicamente para textos normados, con el objetivo de automatizar en cierta forma la interpretación, clasificación y análisis de grandes volúmenes de dichos textos.

Justificación

En la licenciatura de ingeniero en computación es baja la orientación para realizar trabajos escritos de divulgación, más cuando se busca elaborar una tesis, citando al autor Muñoz (1998, pp. 3-5) la palabra *tesis* proviene del vocablo en latín *thesis*, cuyo significado es “proposición” u “opinión”, que hace referencia a la conclusión sustentada con razonamientos y argumentos válidos, de tal manera que sirva para acreditar los conocimientos del estudiante, donde podamos aplicar el criterio profesional y el análisis personal que aprendimos técnicamente en la carrea.

Considerado el entorno vigente de trabajo digital, la cantidad de información generada y almacenada en forma de texto ha experimentado un crecimiento exponencial (Trejo, 2019c), esta vasta cantidad de datos considera desde documentos legales, empresariales y científicos hasta mensajes en redes sociales y correos electrónicos; una gestión eficiente y precisa de esta información es fundamental para numerosos campos, incluyendo la industria, la investigación académica, y la administración pública.

La carrera de ingeniería en computación ha desempeñado un rol pertinente y relevante para el desarrollo de herramientas y técnicas para el procesamiento de datos no estructurados (textos); en particular, el análisis de texto y el aprendizaje automático han emergido como áreas de investigación y aplicación de negocio; estas disciplinas permiten a los sistemas informáticos comprender, interpretar y extraer información útil de grandes volúmenes de texto de manera eficiente y precisa.

El análisis de texto, aprendizaje automático prometen una serie de ventajas en el procesamiento de documentos normados; dado que los textos normados se caracterizan por seguir ciertas convenciones lingüísticas o estructurales, como normas técnicas, elementos legales, protocolos, precedentes, resoluciones, sentencias entre otros. Estas características hacen que el procesamiento automático de estos textos sea especialmente relevante y desafiante, ya que requiere la identificación y extracción precisa de información específica, es por lo anterior la justificación para llevar a cabo un trabajo de tesis en este campo se fundamenta principalmente en los siguientes puntos:

Eficiencia en el procesamiento: el análisis de texto automatizado puede reducir el tiempo y los recursos necesarios para interpretar documentos normados; a través del desarrollo de algoritmos y técnicas especializadas, es posible automatizar tareas como la identificación de secciones relevantes, la extracción de datos clave y la detección de patrones lingüísticos específicos (Trejo, 2019b).

Precisión en la interpretación: los sistemas de aprendizaje automático pueden mejorar la precisión en la interpretación de textos normados al identificar y corregir errores comunes asociados con el procesamiento humano.

Aplicaciones prácticas: la aplicación de técnicas de análisis de texto y aprendizaje automático en textos normados tiene numerosas aplicaciones prácticas en diversos campos. Por ejemplo, en el ámbito legal, estas herramientas pueden utilizarse para analizar leyes, precedentes, sentencias, resoluciones, y evaluar el cumplimiento normativo. En otras áreas como la medicina, pueden ayudar en la extracción de

información relevante de registros clínicos y en la detección temprana de enfermedades. En la industria, pueden mejorar la eficiencia en la gestión de documentos técnicos y normativos.

Contribución al estado del arte: esta no es una tesis de doctorado, y no debe ser exigida como tal, pero, el desarrollo de nuevas metodologías y enfoques para el análisis de texto y aprendizaje automático en textos normados puede contribuir al avance del estado del arte en estas áreas, ya que, al abordar desafíos específicos relacionados con la interpretación de documentos estructurados y formales, se pueden generar conocimientos y técnicas que beneficien a la comunidad académica y profesional.

Problema

En el ámbito del análisis de documentos legales y normativos, que suelen estar estructurados en un lenguaje técnico y específico, existe una creciente necesidad de procesar y entender grandes volúmenes de texto de manera eficiente y precisa. Tradicionalmente, la revisión y el análisis de estos textos se han llevado a cabo manualmente por expertos, un proceso que es intensivo en tiempo y susceptible a errores y sesgos humanos (Sierra, 2009). El problema que se pretende abordar es programar una solución de ingeniería en computación que utilice técnicas de análisis de texto y aprendizaje automático para automatizar y mejorar la interpretación, clasificación y análisis de textos normados, por ejemplo, sentencias, precedentes, resoluciones, leyes u ordenamientos, entre otros.

Hipótesis

La aplicación de técnicas de análisis de texto y aprendizaje automático en la interpretación y clasificación de textos normados resultará en una mejora representativa en la eficiencia y precisión del procesamiento de dichos textos en comparación con los métodos tradicionales basados en revisiones manuales.

Marco Teórico

Metodología

Para la investigación propuesta acerca de la aplicación de análisis de texto y aprendizaje automático sobre textos normados, se pueden combinar elementos cualitativos y cuantitativos, así como enfoques experimentales y de desarrollo de sistemas (Hernández et al, 2014), la metodología propuesta para esta tesis fue una combinación de los siguientes enfoques:

1. Revisión de literatura: se llevó a cabo una revisión de la literatura relacionada con el análisis de texto, aprendizaje automático y textos normados, con base en la cual permitió comprender el estado actual del arte en el campo, identificar las técnicas y herramientas existentes, analizar las áreas de oportunidad y problemas que aún no han sido abordados.
2. Selección de herramientas y tecnologías: se eligieron algunas de las herramientas de software y las técnicas de aprendizaje automático que se consideraron las más adecuadas para el análisis de textos normados; esto incluye algoritmos de procesamiento del lenguaje natural (PLN), técnicas de aprendizaje profundo, y plataformas de análisis de datos.
3. Recopilar datos: se obtuvo un conjunto de datos de textos normados, que incluyeron datos de 2021 del sitio de la Suprema Corte de Justica de la Nación de engroses, leyes, sentencias, resoluciones y otros documentos legales.
4. Desarrollo del modelo: se construyó y entrenó un modelo de aprendizaje automático usando los datos recopilados, lo cual implicó experimentar con distintos algoritmos, realizar ajustes para hallar una solución para el análisis y clasificación de los textos, así como el seguimiento de estos.
5. Evaluación del modelo: del modelo desarrollado y elegido para la tesis se evaluó su precisión, eficiencia y usabilidad.
6. Análisis de resultados: Se presenta el análisis de los resultados obtenidos del modelo, comparando el rendimiento del sistema con los métodos tradicionales de análisis de texto normativo.

7. Optimización y mejoras: basados en los resultados, se procedió a realizar ajustes y mejoras en el sistema para aumentar su precisión y usabilidad.
8. Documentación y discusión: se documentó lo correspondiente con el proceso de la tesis, incluyendo los métodos utilizados, los resultados obtenidos, así como las conclusiones.
9. Presentación de resultados: se presentaron los resultados de la tesis, destacando cómo el sistema desarrollado aborda el problema planteado y contribuye al campo de la ingeniería en computación y el análisis de textos normados.

Conceptos vinculados

Para la presente tesis es necesario dar un elemento básico de términos que se emplearán para mejor comprensión del desarrollo, mismos que a continuación se desarrollan.

Para fines de esta tesis se emplearon datos y documentación pública que se obtuvo de la Suprema Corte de Justicia de la Nación (SCJN), desde su sistema de consulta de asuntos, considerando una muestra de dos mil a partir de la undécima época, es decir a partir del 1 de mayo de 2021 (SCJN, 2021; s.f.).

Es pertinente mencionar que, la SCJN, carece de un portal de datos abiertos que permita obtener las resoluciones y engroses necesarios para esta tesis de manera sistematizada[1], pese a que presenta una presunta plataforma de datos abiertos[2], esta no tiene esquema básicos de conexión, o presenta un servicio consolidado.

La Undécima Época del Semanario Judicial de la Federación y su Gaceta, marcada por el Acuerdo General número 1/2021 del Pleno de la Suprema Corte de Justicia de la Nación, se distingue por introducir cambios en la estructura del Poder Judicial de la

[1] Carece de una interfase de programación de aplicaciones que facilite el hacer consultas bajo arquitectura REST, lo cual fue verificable desde el inicio de esta tesis y al dos de noviembre de 2024.
[2] La dirección es https://bj.scjn.gob.mx/datos-abiertos/conjunto-datos/tesis y https://datos.gob.mx/busca/organization/scjn, la segunda presenta datos presuntamente estadísticos, que no son accionables para efectos de la presente tesis.

Federación, especialmente en cuanto a la emisión de jurisprudencia, siguiendo las reformas constitucionales publicadas el 11 de marzo de 2021 (SCJN, 2021).

Dichas reformas incluyen, entre otras, la jurisprudencia por precedentes, lo que permite al pleno y a las salas del tribunal fijar criterios obligatorios para el resto de las autoridades jurisdiccionales con solo resolver un caso[3].

Este cambio en el sistema de creación de la jurisprudencia elimina la integración por reiteración en los asuntos de competencia del Alto Tribunal y establece que las decisiones dictadas por la Suprema Corte de Justicia de la Nación por mayoría de ocho votos, y por las Salas, por mayoría de cuatro votos, serán obligatorias para todas las autoridades jurisdiccionales de la Federación y de las entidades federativas (ídem), es decir, que busca replicar los beneficios de cada sentencia paradigmática para todas las personas en situaciones similares, promoviendo así todos los derechos para todas las personas.

Es pertinente citar ¿Qué hace la SCJN?, "es el Tribunal Constitucional que mantiene el orden constitucional impuestos a los órganos de gobierno, y hace valer los derechos y libertades de las personas. Imparte justicia a través de mecanismos que permiten vigilar que las leyes y actos de autoridad se apeguen a la Constitución, esos mecanismos son procesos llamados medios de control de constitucionalidad" (SCJN, s.f. b). La SCJN se organiza en dos salas, cada una con cinco ministros, uno de ellos fungirá como su presidente de sala. La ministra o ministro presidente no participa en las Salas, la Primera Sala: resuelve asuntos civiles y penales, la Segunda Sala: resuelve asuntos administrativos y laborales.

La SCJN es parte del Poder Judicial de la Federación, ¿Cómo está integrado el Poder Judicial de la Federación?, por la Suprema Corte de Justicia de la Nación, el Tribunal

[3] Puede ver más detalle en la liga https://www.scjn.gob.mx/sites/default/files/comunicacion_digital/2021-04/boletin_electronico_abril_2021.html

Electoral del Poder Judicial de la Federación, los Tribunales de Circuito (Colegiados y Unitarios), Juzgados de Distrito, y el Consejo de la Judicatura Federal (ídem).

Textos normados

Desde una perspectiva informática, el usuario que es principalmente de orientación de abogado puede concebir una base de datos (BD) como un repositorio digital bien estructurado, donde se almacenan y gestionan expedientes, información de clientes, datos de juzgados, ordenamientos, entre otros elementos relevantes para su práctica. A diferencia de un documento en un procesador de textos, el contenido de la BD se organiza de manera estructurada; por ejemplo, una sentencia de controversia constitucional se guarda en la BD en una tabla que podría contener al menos dos columnas. En la primera columna se listan diversos identificadores, tales como el número de expediente, el actor, los demandados, el ponente, el secretario, y el acto impugnado, mientras que en la segunda columna se registran los datos que corresponden a cada uno de estos identificadores, como el número específico, las partes involucradas y otros detalles. Así, en una misma tabla se pueden almacenar varios asuntos de manera ordenada, lo que se conoce como datos estructurados. En cambio, un archivo de procesador de texto, que consiste en texto secuencial y seccionado, se clasifica en informática como un documento no estructurado (Trejo, 2024a).

El ejemplo anterior cita una forma en la cual los textos normados se pueden almacenar en sistemas informáticos. El análisis de textos normados que se analizarán será principalmente a partir de jurisprudencia, la cual es el conjunto de razonamientos y criterios que emanan de las resoluciones que emiten la Suprema Corte de Justicia de la Nación, los Plenos Regionales y los Tribunales Colegiados de Circuito al interpretar las leyes (ídem), la jurisprudencia determina cuál es el sentido y alcance de las leyes para resolver su aplicación al resolver una controversia, además la jurisprudencia de la SCJN es obligatoria para todos los tribunales y juzgados del país.

Los textos normados son elementos sustantivos de la labor jurisdiccional, es decir, el poder y la autoridad que tiene un determinado tribunal o juez para conocer, juzgar y decidir sobre un asunto legal específico dentro de su ámbito territorial o de competencia (Escriche, 1881, p. 1143).

En el nivel jurisdiccional, especialmente a nivel federal, las labores de los juzgadores son una aplicación formal o informal de administración del conocimiento, la cual inicia desde la queja, demanda o denuncia del justiciable y se continua con el proceso de los abogados de los órganos de investigación o de impartición de justicia, donde en diversos sistemas informáticos van guardando en bases de datos o en archivos sin estructura formal y de entidad, las unidades de conocimiento que conforman las demandas, denuncias, quejas, a partir de las cuales resultan en sentencias, resoluciones, ejecutorias.

Sentencia

En el contexto legal de México una sentencia es una resolución dictada por un juez o tribunal al final de un proceso judicial, si se considera como un documento, es en esta donde se establecen las decisiones finales sobre los hechos y la aplicación del derecho en un caso específico.

La sentencia puede ser condenatoria, absolutoria o declarativa, dependiendo de los resultados del proceso y las circunstancias particulares del caso, es decir, la sentencia es la decisión final del juez respecto a la controversia planteada (Said y González, 2017, p. 352).

Resolución

Otro concepto es el de resolución, el cual es un acto jurídico emitido por una autoridad competente, como un juez, un tribunal o una autoridad administrativa, que tiene como objetivo resolver una cuestión legal o administrativa; estas resoluciones pueden ser de diversa naturaleza y pueden referirse a distintos aspectos del proceso, como medidas cautelares, incidentes procesales, recursos interpuestos durante el proceso, entre otros. En el ámbito judicial, las resoluciones son decisiones intermedias que se toman a lo largo del proceso judicial, antes de la emisión de la sentencia final (Said y González, 2017, p 350).

Ejecutoria

Una ejecutoria es una resolución judicial que ha adquirido firmeza, es decir, que ya no puede ser impugnada mediante recursos ordinarios o extraordinarios; es decir, es la resolución que ha pasado por todos los recursos legales disponibles y se considera definitiva y vinculante para las partes involucradas en el proceso. La ejecutoria es importante porque marca el momento en el cual la resolución se vuelve definitiva y puede comenzar a ser ejecutada, es decir, llevada a cabo (Said y González, 2017, p 369).

Precedente

Otra noción relevante es la de precedente, el cual puede comprenderse como una decisión judicial anterior relevante para la decisión de casos futuros (Moral, 2000), su empleo viene doctrinalmente de países que tienen un derecho de tipo ley común, como son los países anglosajones, a diferencia de los países que tienen un derecho de tipo continental o romano germánico, o ley civil, como el de México.

Los precedentes, que se generan a partir de las sentencias de los jueces, se consideran una fuente de derecho que vincula (obliga) a los jueces (Bernal et al, 2018), de aquí la relevancia de facilitar el realizar un análisis de texto para identificar razones o criterios de decisión previos (rationes decidendi) y no solo el marco normativo (leyes) (Trejo, 2019c).

Para el contexto de México, “la utilización de los precedentes judiciales en la interpretación de leyes y la justificación de decisiones jurisdiccionales es un recurso habitual en los países de tradición continental” (Martínez, 2023, p. XXIII), sin embargo, están más orientados a usar las Leyes como valor normativo.

Dentro del precedente es necesario integrar las referencias de las redes de citación las cuales “se refieren al sistema interconectado de referencias legales y jurisprudenciales utilizadas principalmente por los operadores jurídicos, abogados y académicos para fundamentar y enriquecer sus argumentos, decisiones y análisis;

estas redes son trascendentales para comprender cómo se construyen, así como también la evolución de las normativas y decisiones judiciales" (Ibidem).

Engrose

El engrose de una sentencia en la Suprema Corte de Justicia de la Nación (SCJN) de México es un proceso que sigue después de que los ministros votan un proyecto de sentencia, dicho proceso implica hacer las correcciones sugeridas por los ministros, que pueden incluir agregar, quitar o modificar ciertas consideraciones, e incluso cambiar el sentido de la resolución.

El secretario proyectista es el encargado de realizar el engrose, es decir, de integrar las correcciones indicadas en el proyecto de sentencia y luego redistribuirlo entre los ministros para su aprobación final; una vez que los ministros otorgan su visto bueno, la sentencia se imprime y se agrega al expediente.

El término "engrose" se refiere específicamente a este proceso de elaboración y ajuste final de la sentencia, antes de que sea formalmente emitida y publicada; es importante mencionar que este mecanismo de ajuste no se realiza en un solo día, ya que las correcciones pueden ser sustanciales y requieren de un consenso entre los ministros. Además, las ejecutorias deben ser firmadas por el ministro presidente, el ministro ponente y el secretario de acuerdos antes de su publicación (Lara, 2020).

Este procedimiento asegura que las sentencias sean el resultado de un proceso de revisión detallado y consensuado, reflejando la deliberación colectiva de los ministros con el objetivo de garantizar la precisión jurídica y la justicia en las decisiones de la Corte, aunque esto implique un proceso más largo y detallado para llegar a la versión final de una sentencia (Lara, 2020).

Norma

Es una regla o conjunto de reglas establecidas por una autoridad competente para regular comportamientos o procesos; en el contexto jurídico, las normas dictan cómo deben actuar los individuos y las entidades en sociedad y pueden abarcar desde

principios éticos y morales hasta directrices técnicas específicas, tienen carácter obligatorio y su incumplimiento puede acarrear sanciones. (Kelsen, 1982, p. 17).

Ley

La ley es un tipo específico de norma jurídica que ha sido formalmente promulgada por un órgano legislativo o autoridad competente; las leyes establecen mandatos, prohibiciones y permisos en diversos ámbitos de la vida social, económica, política y personal. A diferencia de otras normas, las leyes pasan por un proceso formal de discusión, aprobación y publicación antes de entrar en vigor (Kelsen, 1982, p. 239).

Tesis

De acuerdo con la SCJN, las tesis "son documentos que contienen criterios relevantes contenidos en las sentencias. Su función es la de servir de herramientas para la sistematización y difusión de los criterios que se encuentran desarrollados en las sentencias". Han sido parte, en las últimas décadas, de la tradición jurisprudencial en México. Existen dos tipos de tesis, en atención a la instancia que las emite y a su fuerza vinculante, tesis aisladas: son criterios orientadores, ilustrativos o persuasivos, más no vinculantes. Las tesis de jurisprudencia, que son criterios jurídicos de observancia obligatoria, es decir vinculantes (SCJN, s.f.). Es importante mencionar que no debe confundirse con el concepto de tesis universitaria, como es el presente documento.

Jurisprudencia

Puede entenderse de dos maneras, "como el conjunto de criterios emitidos por los operadores judiciales, que resultan obligatorios para los tribunales inferiores, por reunirse determinados requisitos formales". Asimismo, se le entiende "como el conjunto de criterios orientadores u obligatorios para el resto de los jueces o tribunales" (SCJN, s.f.).

Ordenamiento Jurídico

El ordenamiento jurídico se refiere al conjunto sistemático y jerarquizado de normas legales que rigen en un lugar determinado, ya sea un país, una región o una comunidad. Incluye diversos tipos de normas, como leyes, reglamentos, decretos, y tratados internacionales, organizados de manera que se complementen y no se contradigan entre sí; cuyo propósito es regular las relaciones sociales, garantizar el orden público, proteger derechos y establecer obligaciones (Kelsen, 1982, p. 307).

La Constitución

La Constitución Política de los Estados Unidos Mexicanos (CPEUM) es la ley fundamental sobre la cual se rige todo el sistema jurídico de México, fue promulgada el 5 de febrero de 1917, establece la estructura del gobierno federal, define los derechos y las obligaciones de los ciudadanos, así como la distribución de competencias entre los distintos niveles de gobierno (federal, estatal y municipal).

Inteligencia artificial (IA)

La IA es un campo de estudio de la informática que busca crear sistemas capaces de realizar tareas que, hasta hace poco, requerían inteligencia humana, estas tareas incluyen la toma de decisiones, el reconocimiento de patrones, el aprendizaje y la comprensión del lenguaje natural. Aplicado al análisis de texto, un subcampo del Procesamiento del Lenguaje Natural (PLN o en inglés NLP de Natural Language Processing, en adelante se emplear el acrónimo en español), la IA busca desarrollar algoritmos capaces de entender, interpretar, y generar texto de una manera que sea relevante para los usuarios humanos (Hirschberg y Manning, 2015).

Para esta tesis es conveniente mencionar que el análisis de texto mediante IA involucra varias técnicas y procesos destinados a comprender el contenido, la estructura y el significado del texto. La IA puede categorizar automáticamente textos en diferentes clases o etiquetas, facilitando su gestión y análisis; puede hacer análisis de sentimientos, que permite a las máquinas interpretar y clasificar las opiniones

expresadas en un texto, determinando si son positivas, negativas o neutrales. Otra aplicación es la generación de resúmenes de largos documentos en textos breves y concisos, permitiendo a los usuarios comprender rápidamente el contenido principal sin tener que leer todo el documento. La IA puede identificar y extraer entidades específicas (como nombres, lugares y fechas) y relaciones entre ellas, proporcionando referencias valiosas a partir de grandes volúmenes de texto. Puede además tener una generación de lenguaje natural (NLG), la cual va más allá del análisis para crear texto nuevo y coherente basado en datos y contextos específicos. (Goodfellow et al, 2016; Jurafsky y Martin, 2021)

Las aplicaciones tradicionales del análisis de texto basado en IA son muchas, las más típicas son de asistentes virtuales y chatbots para mejorar la capacidad de los bots para entender y responder a consultas humanas de manera efectiva. Apoyo en la administración del del conocimiento para facilitar la búsqueda y recuperación de información relevante en grandes bases de datos de texto. El monitoreo de medios y redes sociales y así analizar grandes cantidades de publicaciones para detectar tendencias, sentimientos y opiniones públicas (Trejo, 2019a, 2019b).

Aprendizaje automático

El aprendizaje automático es un subcampo de la IA que se centra en el desarrollo de algoritmos que pueden aprender de los datos y hacer pronósticos o tomar decisiones basadas en esos datos.

Se divide en tres categorías principales: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. El aprendizaje supervisado implica modelar la relación entre las características de entrada y las salidas etiquetadas. El aprendizaje no supervisado busca identificar patrones o estructuras en los datos sin etiquetar. El aprendizaje por refuerzo es un tipo de aprendizaje donde un agente aprende a tomar decisiones optimizando alguna forma de recompensa a lo largo del tiempo. El Aprendizaje Automático es fundamental para una amplia gama de aplicaciones, desde

el reconocimiento de voz y de imágenes hasta la predicción de tendencias en los mercados financieros (Goodfellow et al, 2016).

Un ejemplo podría ser el desarrollo de sistemas que simulan estrategias de litigio o negociación, donde se emplea aprendizaje por refuerzo para mejorar el rendimiento del agente en diferentes escenarios legales. Al aprender a optimizar sus decisiones, este enfoque permite una nueva forma de exploración en la toma de decisiones complejas, aportando valor no solo en la práctica profesional, sino también en el análisis y enseñanza del derecho.

La aplicabilidad del aprendizaje automático en el derecho es de suma relevancia en la era digital actual, incluso independiente de la reciente Reforma Judicial que se está viviendo en México en paralelo a terminar esta tesis, dado que la cantidad de información y datos almacenados en forma de texto legal es vastísima y sigue en crecimiento. En esta línea, el aprendizaje automático facilita el reconocimiento de patrones, la identificación de estructuras y la interpretación de documentos legales de forma eficiente, abordando el reto de gestionar grandes volúmenes de información. Además, la integración de estas tecnologías puede ayudar a reducir el margen de error en la interpretación y aplicación de la ley, así como mejorar la equidad y eficiencia en los procesos legales, dado que la IA puede analizar datos con mayor consistencia que los humanos y sin predisposiciones implícitas.

Si bien el aprendizaje automático no debe sustituir la toma de decisiones finales en el ámbito jurídico, proporciona un apoyo importante que potencia el análisis y la gestión de la información. Además, contribuye al estado del arte en el derecho, promoviendo una práctica más accesible, rápida y precisa (Trejo, 2024b). En última instancia, la implementación de esta tecnología ofrece la posibilidad de avanzar hacia un sistema jurídico más equitativo, eficiente y adaptado a los desafíos del futuro. La intersección entre el aprendizaje automático y el derecho plantea un cambio de paradigma, donde el procesamiento de datos y la interpretación de patrones no solo agilizan los procesos legales, sino que también permiten un análisis más profundo y dinámico de las implicaciones de cada caso y cada decisión en el contexto de la justicia.

Procesamiento de lenguaje natural (PLN)

El procesamiento del lenguaje natural es una rama de la IA que se centra en la interacción entre las computadoras y los humanos a través del lenguaje natural; el objetivo del PLN es permitir que las máquinas lean, comprendan e interpreten el lenguaje humano, y que puedan responder de manera que un humano pueda entender; incluye una variedad de tareas como la traducción automática, la generación de lenguaje, el resumen de textos, el reconocimiento de voz, y la comprensión lectora. El PLN utiliza técnicas de lingüística computacional, ciencia de la computación y aprendizaje automático para lograr sus objetivos (Jurafsky y Martin, 2021).

Fundamentos del análisis de texto y aprendizaje automático en el ámbito legal

El análisis de texto y el aprendizaje automático han transformado la forma en que se pueden analizar y gestionar grandes volúmenes de datos y documentos, para el caso de esta tesis el de textos normados, entre otros, resoluciones, sentencias y tesis. Estas tecnologías ofrecen herramientas que facilitarían el automatizar el procesamiento de texto, mejorar la eficiencia en la revisión de documentos, y extraer elementos que pueden detectar redes de citación, decisiones legales y evoluciones jurisprudenciales.

El análisis de texto, apoyado por técnicas de aprendizaje automático, permite procesar y entender grandes conjuntos de datos textuales; en el contexto legal, esto significa analizar el lenguaje, la estructura y el contenido de documentos como resoluciones judiciales, engroses, sentencias y tesis de jurisprudencia o asiladas, las técnicas de PLN se utilizan para desglosar el texto en componentes comprensibles para la máquina, como frases, oraciones y palabras, facilitando su análisis y clasificación (Jurafsky y Martin, 2021).

El uso del aprendizaje automático en el análisis de documentos legales no está exento de varios retos, desde la precisión de los modelos que depende de la calidad y la cantidad de los datos de entrenamiento disponibles, lo que significa que pueden existir

sesgos si los datos no son representativos o están desactualizados, aunque son elementos normados no siempre están gobernados de manera adecuada o tienen unidades de conocimiento o entidades e información que garantice su consistencia. Además, la interpretación legal requiere un nivel de análisis y comprensión que va más allá del texto superficial, lo que plantea desafíos para la automatización completa de ciertas tareas jurídicas (Trejo, 2020; 2024b).

Desde una perspectiva ética, es necesario garantizar la transparencia y la aplicabilidad de los algoritmos utilizados en el análisis jurídico, ya que los abogados y profesionales del derecho deben ser capaces de entender cómo las máquinas llegan a sus conclusiones para confiar y actuar según estas herramientas (Goodfellow, et al, 2016).

Emplear algoritmos de aprendizaje supervisado, facilita que los programas puedan ser entrenadas para clasificar documentos legales en categorías específicas, como por tipo de caso, ley, jurisprudencia relevante, criterio, precedente; lo cual puede mejorar la eficiencia en la administación de documentos y facilita la búsqueda de información relevante (Goodfellow et al, 2016).

En esta tesis la identificación y extracción de información es importante para identificar entidades legales, referencias a legislación, criterios, precedente y argumentos clave dentro de grandes volúmenes de sentencias o engroses (Hirschberg y Manning, 2015).

Antecedentes

Cumpliendo con las normativas establecidas dentro del marco práctico e ingenieril, en el presente trabajo de investigación, resultado del análisis de texto, se desarrolló un sistema capaz de procesar textos normados.

En el año 2020 se presentó en el VIII Seminario de Lingüística Forense (SeLiFo) dirigido por el Dr. Gerardo Sierra, una ponencia acerca de la administración del conocimiento en el dominio jurídico, en el cual a partir de las propuesta de Atkinson y Drew, así como de Austin y Tiersma, se procuró identificar cómo emplear las citas en sentencias de cortes menores y de expresión verbal en las mismas, para evaluar si

había una conexión entre la redacción de sentencias y el precedente judicial, que se explica más adelante.

De inicio la redacción jurídica en español tiende a ser más normativa, es decir, busca describir cómo debe comportarse un sujeto conforme a una norma. Las cláusulas suelen estar en futuro o subjuntivo (ej. "Deberá cumplir con..."); en inglés, los textos son más descriptivos y condicionales, utilizando modales como "shall", "may", "must", y "will" para indicar obligación, permiso o posibilidad. Además de la diferencia funcional del derecho romano al de ley común hacía inadecuado el emplear algoritmos tradicionales de origen norteamericano.

Estas diferencias operativas fueron consultadas con una especialista de la Facultad de Derecho, la Mtra. Guadalupe Juárez, quien amablemente detalló el punto de diferencia entre procedimiento y acto administrativo para la identificación normativa (Juárez, 2023), que nos daba sustento para poder emplear unidades de conocimiento y entidades de información para abordar desde una perspectiva de administración del conocimiento, para el análisis de textos normados, especialmente sentencias. Sin embargo, sería necesario tener acceso a los sistemas informáticos de los órganos jurisdiccionales, lo cual no era factible[4]; motivo por el cual se integraron la descarga de las sentencias mismas desde el sitio web correspondiente.

Para poner en práctica éste sistema se aplicaron los procesos de análisis correspondientes sobre los engroses provenientes de la Suprema Corte de Justicia de la Nación (SCJN) de los Estados Unidos Mexicanos; dichos documentos almacenados digitalmente en formato Word, pueden ser clasificados como documentos no estructurados debido a que no poseen una estructura bien definida en comparación a una tabla que podríamos encontrar dentro de un documento de Excel, o bien, un archivo de texto plano; éste tipo de archivos, usualmente, no cuentan con un sistema

[4] Como se citó previamente, carece la SCJN de un Portal de Datos Abiertos que expongan las sentencias, engroses o resoluciones pública con estándares de API /REST.

que emplee la tradicional aproximación de reporteo de inteligencia de negocios (BI) (Ohlhorst, 2012, p.6).

Nonaka describe que la generación y sustentación del conocimiento en las organizaciones es de cada vez mayor importancia, esto dado que vivimos en la sociedad de la información, por supuesto esta base no es ajena para el ámbito jurídico mexicano ya que cada caso que se lleva a cabo en el sistema legal genera grandes cantidades de información.

El autor propone que se debe de abandonar la idea de que las organizaciones son simplemente productores o procesadores de información, en el caso del sistema legal mexicano implicaría que no solo se concentre el trabajo de los juristas en el procesamiento y resolución de los casos que se van suscitando a lo largo del tiempo, sino que además se debe tomar en cuenta la generación de conocimiento e información (Nonaka, 1994).

Para Trejo “Los juzgadores deben interpretar y argumentar jurídicamente a partir de los materiales normativos y los casos concretos que llegan a su ponencia” (Trejo Medina, 2024a, p.55) esto empata con la idea de Nonaka de que las organizaciones típicamente se concentran en el procesamiento y en el hacer, más que en el generar conocimiento. “El conocimiento es clave para la generación de la innovación” (Nonaka, 1994), Nonaka nos propone un modelo en el que se permita que las organizaciones puedan crear y administrar el conocimiento llamado SECI.

El modelo SECI se basa en los conceptos de socialización, exteriorización, combinación e interiorización y busca que los individuos que son los principales creadores de conocimiento tácito a través del ejercicio de su trabajo diario, sean capaces de compartir y engrandecer este conocimiento y convertirlo en conocimiento explícito, menciona Nonaka que mientras más repetitiva sea la experiencia del trabajador al desempeñar sus labores, será menor la cantidad de conocimiento que se genere y será asintótica con el tiempo.

Una forma para mejorar la creación de conocimiento en los individuos es, la de hacer que los individuos experimenten distintas actividades que estén relacionadas para que de esa forma el individuo sea capaz de vincular estas experiencias distintas y crear nuevo conocimiento (Nonaka, 1994).

Para Nonaka y Takeuchi (1999) la administración del conocimiento es el proceso continuo y dinámico de creación de conocimiento organizacional, el cual permite a las empresas generar innovación y adaptarse al cambio. Esta herramienta es de vital importancia para los especialistas del derecho ya que les permite concentrarse en encontrar conocimiento que sea relevante para sus propósitos y evitar la ineficiencia de buscar en la gran cantidad de precedentes que existen, así como decisiones judiciales, documentos normativos, leyes, etc.

El tener una herramienta eficiente de administración del conocimiento es desde mi punto de vista un elemento vital para potenciar aún más la creación del conocimiento ya que se evitaría que el profesional del ámbito jurídico deba de realizar investigaciones de forma manual análogamente a lo propuesto por Nonaka, esto sería el equivalente a un trabajador cuya tarea es repetitiva y que al poco tiempo generaría poco conocimiento pues es una tarea que se mecaniza.

Contar con una herramienta de administración del conocimiento pone en disposición del profesional del ámbito jurídico las capacidades de la categorización y la organización de las citas legales, acceder de manera eficiente a precedentes determinados a partir de ciertos criterios de búsqueda, mantener una actualización continua de las decisiones judiciales ya que los precedentes cambian y la jurisprudencia evoluciona junto con la sociedad, así como la recuperación de jurisprudencia relevante permite que el profesional del ámbito jurídico sea capaz de tomar decisiones y desempeñar su rol de forma eficiente.

La administración del conocimiento permite lo que el autor Nonaka (1994) define como que el individuo tenga múltiples experiencias de trabajo y genere conocimiento tácito que después se pueda transformar en conocimiento explícito.

Gardfield (1978) planteó la pregunta sobre si el conteo de cuántas veces es citado un documento científico, puede considerarse como una medida válida de, qué tan relevante es la investigación de un individuo o grupo con respecto a otras investigaciones. Dada la basta cantidad de literatura científica producida a lo largo de los años, si tomamos en cuenta los documentos que son más ampliamente citados podemos determinar o inferir que es la investigación con más citaciones la que se ha encontrado más útil o que ha contribuido más al avance del conocimiento científico en un determinado campo.

A partir de la investigación del autor se plantea la duda sobre si es cierto que, esta medida es simplista y que puede que no refleje la complejidad del proceso de investigación, sin embargo, usando argumentos como la revisión por pares y el hecho de que la comunidad científica utiliza y cita trabajos que son realmente útiles e innovadores por encima de investigaciones obscuras y que no aporten valor, nos permite utilizar esta medida como una forma objetiva de clasificar la investigación de un grupo o de individuos.

Aplicado en el ámbito legal mexicano de forma similar, los juristas deben de determinar y encontrar precedentes legales que se acoplen a sus necesidades y de preferencia usar aquellos que históricamente han tenido mayor peso. Para ello podríamos hablar de las citaciones de documentos normativos, tesis, sentencias, precedentes, votos, etc.

Utilizando la métrica mencionada por Garfield, una tesis jurisprudencial mayormente citada en textos normados sería una tesis de mayor importancia o de mayor utilidad que alguna otra, aunque ambas aborden temas similares. Otro nivel de complejidad que le podemos agregar a esta métrica es el tiempo, ya que la jurisprudencia y las leyes evolucionan con el tiempo, es posible que algún texto jurídico ampliamente citado en el pasado caiga en desuso quizás por su obsolescencia.

Así podríamos determinar que los documentos legales que posean un mayor nivel de citación en una ventana de tiempo determinada, nos permite clasificarlos en

documentos que tienen mayor relevancia y que además están actualizados con respecto a la jurisprudencia vigente.

A estas citaciones que se generan a lo largo de múltiples (miles o decenas de miles) de documentos legales, podemos llamarla una red de citación, la cual se define como "el sistema interconectado de referencias legales y jurisprudenciales utilizadas principalmente por los operadores jurídicos… para enriquecer sus argumentos, decisiones y análisis" (Ashley, 2019 citado por Trejo, 2024b, p.55).

Las redes de citación formadas a partir de los documentos legales junto con la administración del conocimiento forman parte de las herramientas esenciales para poder generar análisis más profundos en la procuración de la ley, asegurar que esta sea consistente y disminuir los tiempos de análisis harán de este proceso un proceso más eficiente (Trejo, 2024b)

Además, el campo de la IA tiene un gran potencial de afectar el ámbito jurídico de forma amplia, mejorando la eficiencia, precisión y pronósticos que serán herramientas adicionales agregadas al conjunto ya existente de elementos que puede utilizar el profesional jurista (Trejo et al, 2024c).

Como nos explica el autor en su trabajo sobre la justicia algorítmica, la IA es capaz de generar dos tipos de modelos, los de pronóstico que están los basados en sistemas expertos (los cuales incluyen y utilizan un conjunto de reglas y proposiciones de la lógica formal para su funcionamiento) y además se encuentran los modelos que se basan en el "aprendizaje".

Los modelos basados en el aprendizaje principalmente utilizan herramientas estadísticas, funciones de minimización de error y elementos matemáticos como el descenso del gradiente, o el método de los mínimos cuadrados para aprender y de esa forma generar modelos que permitan realizar tareas como, clasificación, agrupación, regresión, etc.

Para poder implementar sistemas de administración del conocimiento aplicados en el ámbito jurídico, es necesario primero realizar un procesamiento de la información generada normalmente en documentos no estructurados en formato PDF o Word, para ello se estila la creación de sistemas informáticos conocidos como lagos de datos en los cuales es posible procesar, cargar y analizar información de forma cruda (sin ser procesada previamente).

Los lagos de datos no suelen ser estándares en cómo almacenan los datos estructurados y no estructurados, al contrario de un sistema de inteligencia de negocios tradicional que permite realizar procesos de extracción, transformación y carga de datos (ETL, por sus siglas en inglés) para estructurar los datos o información. Los lagos de datos, de acuerdo con John et al (2017), son grandes repositorios de información cruda, misma que puede ser adquirida, procesada, analizada y entregada a distintos consumidores.

Los engroses son puestos a disposición de manera pública por el Tribunal Constitucional de México mediante un buscador web, es a partir de la descarga de estos documentos que se realizó el estudio primario de su estructura; posteriormente el análisis, procesamiento, enriquecimiento y presentación dentro del sistema programado por esta tesis.

Aun cuando los engroses no cuentan con una estructura rígida de información, es posible identificar patrones dentro de estos documentos jurídicos; un escrito de este tipo está encabezado por el tipo de engrose, es decir: lo que determina si pertenece a una contradicción de criterios, un voto particular, una controversia constitucional o en su mayoría un amparo en revisión; seguido por el número de caso y el año en el que se ha llevado a cabo, así como el nombre del ministro ponente y de los secretarios del engrose.

El engrose además contiene:

- La sentencia que ha sido dictada.

- Una extensa sección de antecedentes, en la que se describe a detalle los hechos.
- Un apartado en el que se especifica si la sala de la SCJN es legalmente competente**,** es decir, si cuenta con la autoridad legal y la jurisdicción para conocer y resolver el caso (la competencia se basa en una serie de normas legales y reglamentos que definen qué tipo de casos puede resolver cada órgano judicial).
- Una sección de oportunidad en la que se decide si el caso ha sido interpuesto en tiempo y forma.
- La legitimación**,** que se centra en poder discernir si una persona o entidad es capaz de interponer una acción legal.
- El estudio de fondo**.**
- Cerca del final de un engrose encontramos una sección de suma importancia llamada: decisión**.** Después de realizar un análisis profundo sobre el recurso legal, ésta determina acciones a tomar y contiene las firmas de las personas involucradas en el dictamen, como son el ponente, los secretarios y ministros que en conjunto elaboraron el engrose.

Así, resulta evidente que los engroses cuentan con una serie de elementos clave con los que se puede trabajar ante lo ya expuesto, afirmando con ello la idea de que, si bien son documentos escritos en lenguaje natural, pueden transformarse en documentos que posean una distribución más estructurada para su posterior aprovechamiento.

Existen distintas técnicas y aproximaciones para el cumplimiento del análisis de documentos no estructurados y la producción de información estructurada, como lo son los métodos basados en algoritmos determinísticos (procedimientos que están basados en lógica y que dada una entrada producen un resultado igual siempre) y los métodos basados en algoritmos probabilísticos (no necesariamente nos entregan el mismo resultado dada una entrada) (Gupta y Singh Lehal, 2010).

Una de las motivaciones para emplear documentos no estructurados de la SCJN y cargarlos en un sistema que nos permita realizar el procesamiento y análisis de texto, es proveer una posible solución para ahorrar tiempo cuando un litigador de ese órgano jurisdiccional desee buscar y obtener una visión general de la información sin necesidad de leer todo el documento, con ello, lo que se pretende es facilitar la búsqueda y organización, así como mejorar la comprensión de documentos complejos al destacar sus puntos principales.

Entre las tareas de "estructuración" de documentos no estructurados se encuentra la generación de una síntesis que describa la información más relevante del documento original, además de esta tarea se encuentra la detección de la citación de tesis jurisprudenciales dentro de los textos normados, esto para poder posteriormente, vincular los textos normados con sus tesis jurisprudenciales y de esa manera formar una red de vínculos.

La generación de síntesis o compilaciones de textos de gran tamaño se puede llevarse a cabo mediante diversos métodos, uno de ellos se fundamenta en efectuar una transformación a cada una de las oraciones de un documento. Esta conversión consiste en transformar en vectores los significados semánticos del texto (Arroyo-Fernández et al, 2019). Conforme a lo indicado por el autor Sierra señala en su estudio sobre la representación de sentencias y su comparación de significados, es factible desarrollar un modelo que permita discernir si dos sentencias poseen significados similares. Este modelo se genera a partir de la representación vectorial de cada palabra de una oración y empleando la entropía como un indicador de la incertidumbre o diversidad de información vinculada a una palabra en un texto.

Otras virtudes de la habilidad para representar oraciones textuales en espacios vectoriales incluyen la capacidad de efectuar agrupaciones de sentencias de manera no supervisada, realizar clasificación de sentencias, análisis de sentimiento, análisis de similitud semántica, etc. Esta labor se halla principalmente condicionada por el tamaño del texto a representar (Arroyo-Fernández et al, 2019).

Para llevar a cabo la tarea de comparar dos sentencias y determinar su similitud, los autores proponen la utilización de la métrica de similitud textual semántica (STS). Esta consiste en calcular una puntuación asignada a un par de oraciones, y dicha puntuación indicaría el nivel de similitud entre las mismas. Mediante la transformación de oraciones en vectores de un espacio vectorial, somos capaces de emplear diversos métodos matemáticos para calcular similitudes entre vectores. Estos incluyen la distancia Manhattan, que cuantifica la distancia entre dos puntos siguiendo exclusivamente trayectorias horizontales y verticales, la similitud del coseno, que puede ser interpretada como la proyección de un vector sobre otro en el espacio vectorial, y la distancia Euclídea, que cuantifica la distancia lineal entre dos puntos en dicho espacio.

Una de las posibles aplicaciones en el contexto del análisis de textos normados consiste en ampliar la aplicación de estos modelos de oraciones a textos de mayor envergadura, con el objetivo de determinar de manera más extensa, por ejemplo, el significado de un texto normado e identificar otros que sean análogos. De manera similar, si disponemos de una serie de representaciones vectoriales de un texto regulado, es factible elaborar un resumen del texto mediante el análisis de los vectores. Aquellos con significados similares podrían constituir los "temas" que aborda un texto normado.

En el mundo digital podemos encontrar herramientas que permitan la aplicación de tecnologías de la información donde el problema principal requiere analizar, consolidar o hallar información, en este caso, para el ámbito jurídico hablaremos de los grandes volúmenes de datos, también conocido como *Big Data*; concepto que se volvió popular alrededor del año 2010, dado que la cantidad de información que se generaba de orden estructurado y no estructurado era mayor año con año, tenía más variabilidad y velocidad. No obstante, no existe un consenso entre los autores sobre un único significado ya que éstos van desde las 3Vs (volumen, velocidad y variabilidad), llegando incluso hasta 9Vs (Buyya et al, 2016)

Cox y Ellsworth (1997) considerados como los acuñadores del término Big Data, definen en términos relativamente modernos qué es a lo que el problema del Big Data se refiere, esto es a partir de tener grandes conjuntos de datos (Terabytes o Exabytes en la actualidad) que no pueden ser procesados en la memoria principal, ni tampoco en el disco local de un sistema, ya que exceden las capacidades de estos elementos dentro de un sistema de cómputo. Lo anterior, trasladándolo al contexto del año de 1997, era primordialmente necesario atenderse ya que los equipos de cómputo no contaban con las cantidades de memoria principal y secundaria a las que hoy en día se tiene acceso.

Las características, de acuerdo con los autores Cox y Ellsworth, que tenemos que tomar en cuenta para definir el Big Data se pueden delimitar desde tres puntos de vista: el del dominio de los datos (referente a buscar patrones en la información), el de la inteligencia de negocios (que hace énfasis en la capacidad de hacer pronósticos) y el de la estadística (que habla de las inferencias sobre los datos). Cuando mezclamos estos tres puntos de vista sobre los datos surge la definición basada en 9Vs:

- Visibilidad - Información y sus metadatos.
- Valor - Datos que contienen información valiosa.
- Veredicto - Posible acción o decisión basada en la información.
- Veracidad - Se centra en el hecho de si la información es verdadera.
- Validez - Verificación de la calidad de los datos y de evitación de sesgos.
- Variabilidad - Complejidad y variación, el tipo y número de variables en un conjunto de datos.
- Volumen - Referente a la escala y a la acumulación de los datos.
- Velocidad - Ritmo en que los datos son generados.
- Variedad – Diversidad de formatos y estructuras de datos.

Se puede resumir entonces que el problema del Big Data se centra en la necesidad de contar con sistemas de cómputo capaces de procesar grandes volúmenes de datos.

Para ello, es fundamental considerar las 9 Vs definidas: volumen, variedad, velocidad... No obstante, también debemos centrarnos en el desarrollo de tecnologías que nos ayuden a abordar el desafío. Big Data no se limita al tamaño de los datos, sino que abarca los procesos necesarios para aprovecharlos, incluyendo la adquisición, almacenamiento, búsqueda, análisis y visualización de los datos.

Como señala Ohlhorst (2012, p. 33) “tradicionalmente se pensaba que el valor principal de la información residía en los datos estructurados”; los cuales, según el autor, suelen representar menos del 20% del total de datos generados por una organización. Sin embargo, el 80% restante ha demostrado tener un gran valor, como lo evidencian diversos proyectos informáticos:

- Reconocimiento facial.
- Motores de búsqueda.
- Reconocimiento de voz.
- Sistemas de procesamiento del lenguaje natural.

En el corazón del Big data se encuentra un algoritmo que cambió la forma en la que se realizan los procesos de cómputo de grandes volúmenes de datos: el algoritmo de MapReduce, el cual se introdujo en el año 2008 desarrollado por investigadores de la empresa Google (Dean y Ghemawat, 2008).

MapReduce se puede definir en dos fases: La fase de mapeado en la que la primera tarea es que los datos se dividan en fragmentos que pueden procesarse simultáneamente de manera independiente y paralela entre sí; cada fragmento de datos se somete a una función de mapeado que lo convierte en una serie de pares clave-valor. En la segunda fase del proceso, la reducción de datos (llave-valor) se agrupan según su clave correspondiente y se envían a una función de reducción que fusiona los valores asociados a cada clave, para generar un resultado consolidado.

La filosofía que nos deja este documento es demasiado poderosa, ya que se centra en los procesos que tienen una gran cantidad de datos de entrada y considera que deben tener la capacidad de distribuir los cálculos o procedimientos complejos en

distintos servidores o equipos de cómputo, sin importar su tipo o tamaño. Es decir, hacer uso de forma masiva del cómputo distribuido.

La siguiente idea central que nos aporta este documento es la de dividir en pequeñas fracciones los datos a procesar, permitiendo con ello un manejo más sencillo de la información. El hecho de que está demostrada de forma efectiva la operatividad a través de pequeñas partes y no sobre el conjunto de datos original, es una de las principales inspiraciones en las cuales se basa la arquitectura de la solución que presenta esta tesis.

En general se busca que del total de engroses que se tienen que procesar, ya sea en tiempo real o en procesamiento por lotes, sean enrutados a distintas instancias de servicios, las cuales pueden vivir en distintos equipos de cómputo, y sean procesados de manera distribuida.

Un concepto relevante que ha revolucionado el paradigma tecnológico es el de "Transformers", se enmarca en el campo del aprendizaje profundo, una subdisciplina de la IA. Según la investigación del autor Bishop (2023), esta modalidad de red neuronal se enfoca en recibir una secuencia de vectores (palabras, píxeles de una imagen, ondas de sonido, entre otros) y aplicar una serie de pesos para transformar estos vectores a un espacio distinto.

Este tipo de red neuronal fue inicialmente empleada en el campo del procesamiento del lenguaje natural con el objetivo de efectuar traducciones automáticas entre lenguajes. No obstante, en la actualidad se emplea en una amplia gama de dominios, incluyendo servicios de chat como ChatGPT, la traducción automática de textos, la visión computacional, entre otros.

El término "Transformers" fue inicialmente introducido en el año 2017 en un estudio titulado "atención es todo lo que necesitas", representando una transformación paradigmática en relación con las redes neuronales preexistentes (Vaswani et al, 2017). Los Transformers poseen habilidades distintas a las de las redes neuronales preexistentes, tales como:

1. Un modelo fundamentado en "Transformers" tiene la capacidad de tomar toda la información proporcionada en su entrada y evaluarla simultáneamente, en contraste con sus predecesores, como las redes neuronales recurrentes que evaluaban de manera secuencial los datos de entrada. Este enfoque permite una reducción en los tiempos de entrenamiento de un modelo, dado que el entrenamiento se lleva a cabo de manera paralela.
2. Los modelos de IA fundamentados en "Transformers" incorporan un mecanismo de "atención", que facilita la comprensión del contexto de los datos procesados, y además poseen un tipo de memoria que permite el acceso a datos previamente visualizados. Contrariamente a sus predecesores que carecían de un mecanismo de atención y que, en su mayoría, estaban diseñados para conocer o retener en la memoria el dato previo que se procesó (Devlin y Chang, 2018). Este mecanismo posibilita que un modelo de tipo transformer "recuerde" el contexto de operaciones anteriores ejecutadas. Un ejemplo práctico sería el uso de ChatGPT u otros modelos, cuando se realizan múltiples solicitudes. El modelo puede tener la capacidad de retener información de respuestas previas, tales como algún resultado intermedio matemático, entre otros.
3. Los modelos de tipo "Transformar" pueden ser entrenados con grandes volúmenes de datos disponibles en línea, resultando en lo que se identifica como un modelo fundamental. Este tipo de modelos poseen la capacidad de ser adaptados posteriormente para ejecutar diversas tareas distintas a las originalmente diseñadas, como el procesamiento de tareas matemáticas o de programación o en nuestro caso la detección de entidades de índole jurídica. Esta metodología se denomina "fine-tuning" (Bishop y Bishop, 2023).
4. De esta arquitectura han emergido modelos de lenguaje, como el GPT desarrollado por OpenAI en 2018, BERT (con 330M de parámetros) desarrollado por Google en ese mismo año, GPT-2 (con 1.5 mil millones de parámetros) en 2019, GPT-3 (con 175 mil millones de parámetros) en 2020, el modelo LLaMA de Facebook, GPT-4 y, probablemente, el producto más reconocido, ChatGPT en 2023. Este tipo de modelos lingüísticos cumplen con

un objetivo general y son entrenados con grandes volúmenes de datos, así como con millones o billones de parámetros (pesos entre las interconexiones neuronales de la red neuronal) (Zhao et al, 2023).

Derivado de esta gran cantidad de parámetros se habla de un fenómeno impredecible presentado en las habilidades de estos modelos, formalmente conocido como habilidades emergentes en grandes modelos de lenguaje (Wei et al, 2022), se explica que estas habilidades no están presentes en modelos de menor escala pero que aparecen en modelos de mayor escala. Algunas de estas habilidades son:

- La capacidad de resolver tareas de tipo aritmético
- Contestar preguntas de forma correcta.
- Poder convertir una palabra de un alfabeto a otro.
- Reconstruir una palabra a partir de sus letras desordenadas.
- Comprensión multitarea, poder realizar traducciones y resúmenes.

Este tipo de modelos de lenguaje son ideales para realizar resúmenes de grandes documentos como los engroses, ya que es posible:

- Procesar grandes cantidades de documentos en lenguaje natural.
- Ser multifuncional, ya que existen muchos modelos de lenguaje y no estamos acoplados a ninguno de ellos se puede probar cualquiera que esté disponible y de mejores resultados. También es posible que en un futuro se liberen nuevos y más poderosos modelos que le permitan a la aplicación desempeñar su tarea de mejor forma, dado que no estamos acoplados a ningún modelo en particular, su actualización es muy simple y permite que podamos expandir las capacidades de la aplicación.
- Nos permitirá generar resúmenes de los engroses de la Suprema Corte de Justicia de la Nación para facilitar la compresión, de uno de estos documentos que suelen tener grandes cantidades de información.
- Conforme avance la tecnología seremos capaces de obtener mejores resultados.

Reforzando la idea anterior, se ha demostrado que los grandes modelos del lenguaje superan con creces otro tipo de modelos como son los de entrenamiento supervisado basados en redes neuronales, con más de 127,000 ejemplos (Radford et al, 2019) para la tarea de resumir documentos de gran tamaño y así poder tener una visión general del contenido de estos.

En la investigación de Radftord se dieron a la tarea de generar 127 mil ejemplos de textos y los resúmenes esperados, también existen otras alternativas estadísticas y basadas en heurística para la tarea de resumir textos en lenguaje natural sin embargo, dada la complejidad de la tarea de comprender el lenguaje, así como las ideas de un texto complejo y de esa manera generar un resumen, no parece haber mejor alternativa hasta el día de hoy, que utilizar grandes modelos del lenguaje LLM (Large Language Model por sus siglas en inglés) para la tarea de extraer un resumen de grandes textos como lo son los textos normados.

Además de tomar un engrose y generar un resumen utilizando un modelo LLM, es necesario poder enriquecer de mayor forma la presentación de los textos normados, por lo que se busca ser capaces de detectar las leyes que se encuentren nombradas o citadas, así como las tesis de jurisprudencia referenciadas dentro del documento.

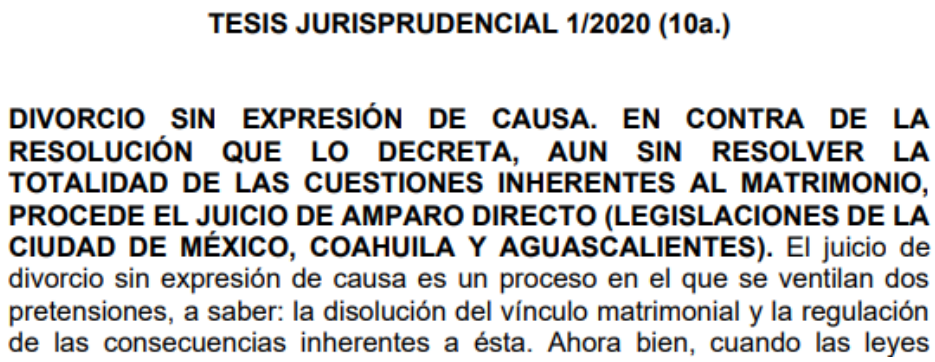
Esto no es una tarea sencilla debido a la alta variabilidad del lenguaje humano, en particular en el ámbito jurídico, ya que existen muchas formas de citar a una sola ley, reforma, tratado o documento; pongamos de ejemplo la Constitución Política de los Estados Unidos Mexicanos, misma que se puede nombrar como Carta Magna, Constitución Política o simplemente Constitución; de la misma manera en los engroses es posible que se nombren leyes que hayan cambiado de nombre o que hayan sido reemplazadas por otras a lo largo del tiempo.

Un problema en la detección de las leyes dentro de un documento podría ser que existen distintas formas de escribirlas sintácticamente hablando, uno podría acentuar una palabra mientras que otro podría omitir ese detalle, así como la colocación de puntos y comas, el uso de mayúsculas o el orden de ciertas palabras.

Además de ello sobre las tesis jurisprudenciales existe un problema parecido ya que estas poseen una convención de nombrado variable, en el ejemplo de la ilustración 1 nos encontramos con la tesis jurisprudencial 1/2020 (10a.) tomada del sitio de la Suprema Corte de Justicia de la Nación (SCJN):

Ilustración 1

Ejemplo de nombrado de tesis.



TESIS JURISPRUDENCIAL 1/2020 (10a.)

DIVORCIO SIN EXPRESIÓN DE CAUSA. EN CONTRA DE LA RESOLUCIÓN QUE LO DECRETA, AUN SIN RESOLVER LA TOTALIDAD DE LAS CUESTIONES INHERENTES AL MATRIMONIO, PROCEDE EL JUICIO DE AMPARO DIRECTO (LEGISLACIONES DE LA CIUDAD DE MÉXICO, COAHUILA Y AGUASCALIENTES). El juicio de divorcio sin expresión de causa es un proceso en el que se ventilan dos pretensiones, a saber: la disolución del vínculo matrimonial y la regulación de las consecuencias inherentes a ésta. Ahora bien, cuando las leyes

Nota. Adaptado de Gutiérrez Gatica (2024).

En ese mismo texto podemos encontrar tesis jurisprudenciales citadas:

- Contradicción de tesis 104/2019,
- tesis jurisprudencial PC. VIII. J/5 C (10ª),
- tesis jurisprudencial PC.XXX. J/18 C (10ª.),
- tesis de jurisprudencia 1a./J. 8/2012 (10a.)

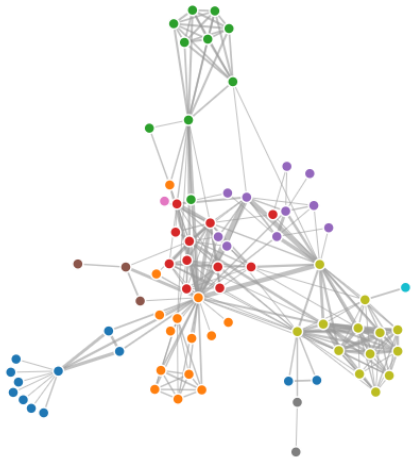
Con esto se destaca que las tesis jurisprudenciales poseen una alta variabilidad al ser nombradas, así como en la forma en la que se citan en los documentos jurídicos, lo que dificulta su detección, algunas veces estos citados llevan paréntesis, algunas otras se omiten las diagonales y en muchas más consciente o inconscientemente se agregan puntos y caracteres extras.

Una de las principales fuentes de valor de este sistema se centra en el hecho de poder detectar de forma automática esta citación o referenciación de tesis jurisprudencial

para poder facilitar la búsqueda y determinación, cuestionándose así ¿cuáles son las tesis jurisprudenciales de mayor relevancia a lo largo de los engroses procesados?, estos vínculos posteriormente se podrían graficar utilizando herramientas de grafos y dando lugar a concentraciones como las que se muestran en la siguiente figura:

Ilustración 2

Ejemplo de posible explotación automática de la citación de tesis jurisprudencial.



Nota. Tomado del sitio D3.js (2024)

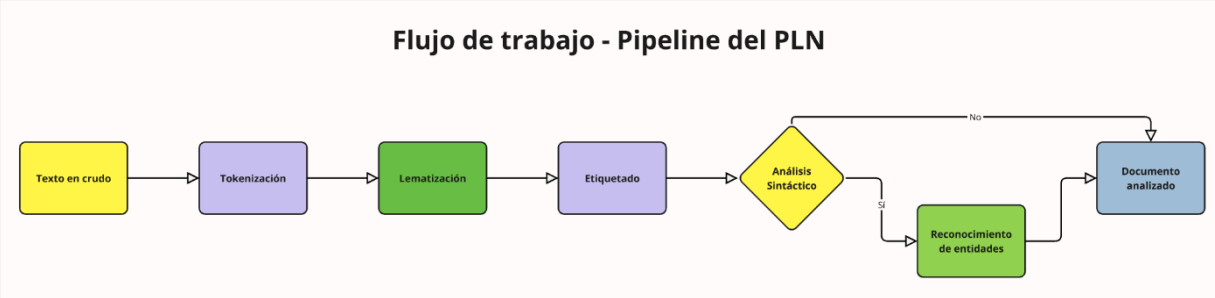
Para poder realizar esta detección de entidades en los textos se hará uso de una tecnología llamada SpaCy, la cual es una librería de código abierto escrita en el lenguaje de programación Python, que nos permite realizar “la segmentación y entrenamiento para más de 70 lenguajes, poder utilizar modelos de redes neuronales para etiquetar, analizar sintácticamente, realizar el reconocimiento de entidades, clasificación de texto, etc.” (Honnibal, 2017, sección Features), esto además utilizando la potencia de las tarjetas gráficas para realizar computo en paralelo.

Como argumenta (Vasiliev, 2020) existen una serie de tareas básicas que todo flujo de trabajo o pipeline de procesamiento del lenguaje natural posee:

- Tokenización (segmentación**)** – el proceso de tomar un texto y descomponerlo en componentes individuales, típicamente palabras, números, elementos de puntuación.
- Lematización – el proceso de tomar una palabra y reducirla a la forma raíz, por ejemplo, la palabra corro, se lematizaría como correr y la palabra integraciones se lematizaría como integrar.
- Etiquetado – se refiere al proceso de asignar una categoría a cada token del texto a analizar, verbo, sujeto, adjetivo, sustantivo, pronombre.
- Análisis Sintáctico – se refiere al proceso de identificar relaciones entre los tokens individuales, por ejemplo, un humano puede fácilmente determinar cuál es el sujeto de una oración, lo mismo con la acción que realiza el sujeto.
- Reconocimiento de entidades – una entidad nombrada es un objeto el cual puede ser una persona, un lugar, una empresa o cualquier cosa que puedas categorizar con un nombre, este proceso se encarga de clasificar cada token como una entidad. Determina si una palabra es un nombre o si hace referencia a un lugar, una fecha, etc.

Ilustración 3

Flujo de trabajo típico del PLN.



Nota. Realizado por el autor.

Una de las virtudes de elegir SpaCy es que nos podemos apalancar de todo este flujo de trabajo que ya está implementado y respaldado por una empresa y concentrarnos en agregar valor al flujo mismo. En nuestra aplicación en concreto, la visión que se

tiene es poder utilizar el módulo de reconocimiento de entidades para poder detectar las leyes nombradas dentro de los engroses de la Suprema Corte de Justicia de la Nación y además ser capaces de detectar las tesis jurisprudenciales citadas.

En el caso de SpaCy, es posible entrenar modelos para las secciones: etiquetado de palabras, para el análisis sintáctico y para el reconocimiento de entidades. Una de las ventajas de SpaCy es que ya nos entrega modelos previamente entrenados utilizando un corpus de texto en español para realizar las tareas de etiquetado general de manera automática (SpaCy, 2015).

Este etiquetado general es capaz de encontrar personas nombradas en textos, lugares, fechas y montos, sin embargo, se trata de un modelo de propósito general, el cual se queda corto para su aplicación directa en el ámbito jurídico, pues el lenguaje técnico de los temas legales propicia que este tipo de modelo genérico, no se desempeñe de manera eficiente para realizar las tareas que necesitamos en este proyecto de tesis.

El reconocimiento de entidades es el proceso del flujo de trabajo en el que SpaCy reconoce entidades nombradas. Y es en esta sección del flujo de trabajo en la que realizamos una optimización para que exista un modelo ad-hoc que sea capaz de detectar tanto de las leyes nombradas como las citaciones de tesis jurisprudenciales.

Los modelos de reconocimiento de entidades de SpaCy son modelos estadísticos basados en redes neuronales (Vasiliev, 2020), el modelo de entrenamiento que utilizan es un entrenamiento de tipo supervisado el cual a diferencia de los grandes modelos de lenguaje, de los que se habló con anterioridad en este documento, que aprenden de manera automática, se necesita que generemos un conjunto de datos de entrenamiento; ejemplos concretos a partir de los cuales, este modelo podrá “aprender” y ser capaz de desempeñar la tarea de clasificar palabras de manera automática con cierto grado de precisión.

Para esta tarea SpaCy nos pone a disposición dos formas de generar datos de prueba. La primera y más lenta, así como laboriosa, se trata de delimitar dentro de un texto

ejemplos para cada uno de los casos que queremos habilitar para que el modelo lo detecte. En la siguiente ilustración muestro dos ejemplos del proceso que se tendría que realizar para el proceso de entrenamiento supervisado:

1. Se tendría que buscar ejemplos de las partes del texto que son necesarias clasificar, en amarillo tenemos el texto de “Constitución Política de los Estados Unidos Mexicanos” el cual queremos que sea clasificado con la etiqueta “LEY” y en azul en el segundo ejemplo “1ª/J 25/2019” el cual queremos que sea clasificado con la etiqueta de “TESIS”, es importante destacar que se debe de agregar texto extra en estos ejemplos, ya que SpaCy recomienda que haya contexto para que el modelo sea capaz de generalizar y no solo “aprenda” los ejemplos que vea.
2. Determinar cuáles son las posiciones de estas entidades con respecto al texto de ejemplo, siendo los números 44, 97 y 56, 68 respectivamente.
3. Armar una gran cantidad de datos de entrenamiento para que el modelo pueda generalizar y encontrar patrones para desempeñarse de manera más eficiente, esta tarea es compleja ya que requiere que se busquen los ejemplos para cada ley que se desee detectar.

Ilustración 4

Ejemplos de entrenamiento para modelo de reconocimiento de entidades nombradas.

```
datos_entrenamiento = [
  ('con lo dispuesto en los artículos 106 de la Constitución Política
    de los Estados Unidos Mexicanos',
  {'entities':[(44, 97, 'LEY')]
  }),

  ('plasmada, entre otras, en la tesis jurisprudencial 1ª/J 25/2019,
    fue elaborado bajo su ponencia',
  { 'entities':[(56,68, 'TESIS')]
  })

  ...
]
```

Nota. Realizado por el autor.

El principal inconveniente de este método de entrenamiento es que es altamente manual; utilizando métodos de programación se podría automatizar hasta un punto limitante, pero no sería del todo escalable y confiable, además de que consumiría mucho tiempo para poder reunir una gran cantidad de ejemplos de textos para el entrenamiento. Otra dificultad es que existen alrededor de 300 leyes y seria complejo encontrar ejemplos suficientes de cada ley dentro de los textos para que el modelo pudiera generalizar patrones sobre ellas.

Para este tipo de casos, existe otro método de generación de datos de entrenamiento semiautomático, utilizando un componente basado en reglas y patrones llamado “Matcher” de SpaCy. Este componente requiere que lo alimentemos de patrones de texto, a partir de los cuales, se encargará de buscar partes del todo en un corpus que cumplan con dicho patrón.

Ilustración 5

Patrón de búsqueda para entrenamiento del modelo de reconocimiento de entidades.

```
# 1a./j. 85/2008  --> this ignores (10a.)
# 1a./j. 85/2008
pattern_a = [
    {"text": {"REGEX": r"[0-9][a-z]./[a-z]"}},
    {"IS_PUNCT": True},
    {"text": {"REGEX": r"[0-9]{2}/[0-9]{4}"}},
]
# 1a./j 85/2008 without the point
pattern_a1 = [
    {"text": {"REGEX": r"[0-9][a-z]./[a-z]"}},
    {"text": {"REGEX": r"[0-9]{2}/[0-9]{4}"}},
]
```

Nota. Hecho por el autor.

En la ilustración 5 ejemplifico dos patrones relativamente simples para poder identificar tesis jurisprudenciales, el primer patrón es capaz de identificar tesis jurisprudenciales del tipo “1a./j. 85/2008” con variantes en el año, el número de documento y las iniciales, por ejemplo, el patrón identificaría correctamente como tesis jurisprudencial “1a./j. 99/1999” o “2b./j. 01/2024”.

El segundo patrón es parecido al primero, pero toma en cuenta una de las dificultades ya mencionadas para la identificación de las tesis jurisprudenciales, la variabilidad de las formas en las que se realiza. El segundo patrón elimina la necesidad de que exista un punto después de la “j”.

Los patrones para encontrar las leyes son mucho más sencillos:

Ilustración 6

Ejemplo para generación del patrón para encontrar leyes

```
# ley organica del poder judicial de la federacion
pattern_ley_organica = [
    {"LOWER": "ley"},
    {"LOWER": "organica"},
    {"LOWER": "del"},
    {"LOWER": "poder"},
    {"LOWER": "judicial"},
    {"LOWER": "de"},
    {"LOWER": "la"},
    {"LOWER": "federacion"},
]
# ley adjetiva penal
pattern_ley_adjetiva_penal = [
    {"LOWER": "ley"},
    {"LOWER": "adjetiva"},
    {"LOWER": "penal"},
]
# ley general para prevenir y sancionar los delitos en materia de secuestro
pattern_ley_general_secuestro = [
    {"LOWER": "ley"},
    {"LOWER": "general"},
    {"LOWER": "para"},
    {"LOWER": "prevenir"},
    {"LOWER": "y"},
    {"LOWER": "sancionar"},
    {"LOWER": "los"},
    {"LOWER": "delitos"},
    {"LOWER": "en"},
    {"LOWER": "materia"},
    {"LOWER": "de"},
    {"LOWER": "secuestro"},
]
```

Nota. Hecho por el autor.

La ventaja de este método es que se puede alimentar al componente “Matcher” con una gran cantidad de texto y un conjunto de patrones, lo cual nos permite generar

miles de casos de entrenamiento para su posterior utilización en el entrenamiento del modelo de clasificación.

Esta aproximación de generación de datos de entrenamiento basada en patrones es poderosa, pero tiene sus limitaciones, la más importante a mi parecer es que, es necesario que los patrones no se traslapen ya que SpaCy no es capaz de asignar dos etiquetas a un token. Es por esta limitación que necesitamos ser cuidadosos al diseñar los patrones, para que, evitemos en la medida de lo posible que dos o más patrones distintos identifiquen una palabra con más de una categoría.

Para ser capaces de integrar la serie de tecnologías de un gran modelo de lenguaje (LLM), SpaCy en el análisis de textos no estructurados y ser capaces de ponerlo todo disponible en una sola plataforma que brinde beneficios a los usuarios, es necesario hacer uso de una herramienta básica del ingeniero en computación.

La ingeniería de software, según el autor Sommerville “es una disciplina de la ingeniería que se encarga de todos los aspectos de la producción de software desde las etapas tempranas de la especificación de un sistema, hasta el mantenimiento de este mientras se encuentra en etapa de uso” (2016, p. 7-8).

Para Farley la ingeniería de software está definida como “la aplicación de soluciones empíricas y científicas para encontrar soluciones eficientes y económicas a problemas prácticos en el software” (2021), y, desde el punto de vista de Winters et al (2020, p.3-4) “La ingeniería de software es programación integrada a lo largo del tiempo”.

Tradicionalmente se abordó el problema del desarrollo de software desde el punto de vista de la producción. Ejemplos como el de construir un puente o las cadenas de producción en serie de los automóviles han sido traídos a colación cuando de desarrollar software se trata (Farley, 2021). Los problemas de producción son distintos a los problemas del software, por ejemplo, producir se trata de tener una cierta cantidad de toneladas de vigas de acero listas para soldarse y armar la estructura de un puente, problemas logísticos para asegurar que una línea de producción jamás se interrumpa pues nunca deberías quedarte sin llantas o sin motores para seguir armando

vehículos, cada segundo que una línea de producción está detenida cuesta. Por otro lado, el desarrollo de software se trata de diseñar. Ser capaces de diseñar una pieza de software que cumpla con ciertas características funcionales.

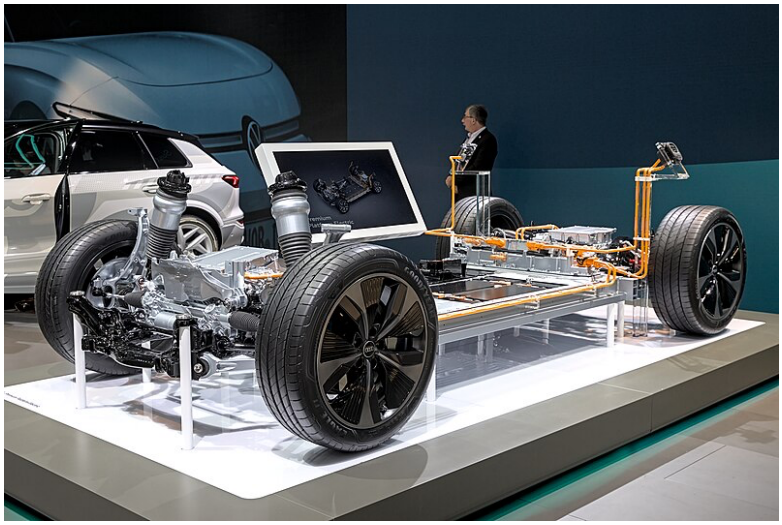
- Separación de responsabilidades.
- Cohesión.
- Modularidad.
- Abstracción.
- Bajo acoplamiento.

Cuando mencionamos que el desarrollo de software se trata de un problema de diseño más que de un problema de producción, nos referimos a un problema filosófico, de fondo más que de forma. La ingeniería de software nos permite diseñar soluciones que tengan una base, un bastidor, es a partir de una base mediante la cual podemos empezar a construir soluciones de software que escalen.

Un caso ilustrativo de cómo deberíamos diseñar software es el de los chasis de los vehículos, que constituyen una plataforma configurable que puede ser posteriormente transformada en diversos modelos. La ilustración 8 ilustra el bastidor de un automóvil, permitiendo la "experimentación" con diversos modelos. Se puede fabricar un motor de potencia reducida o aumentada, ya sea eléctrico o a gasolina, siempre que el motor satisfaga los requisitos de espacio y otros criterios designados. La posesión de una base con componentes susceptibles de intercambio facilita la flexibilidad y permite experimentar con diversas implementaciones.

Ilustración 7

El software debe ser diseñado como una plataforma con piezas que se pueden intercambiar



Nota. Fuente: Boëtius, C. (ca. 1650). Young woman warming her hands over a brazie [fotografía de Rijksmuseum Ámsterdam]. Tomada de https://commons.wikimedia.org/wiki/File%3AYoung_woman_warming_her_hands._Caesar_van_Everdingen.jpg.

Los sistemas de software son complejos y requieren de mucha experimentación, ya que existen muchas formas de programar soluciones dado un problema específico. Esta es la razón por la que el autor Sommerville propone que se realice una programación de forma iterativa, en la que se desarrolle software en pequeños incrementos, es decir pequeños entregables que nos permitan ir agregando funcionalidades al software y eventualmente llegar a la solución deseada (Sommerville, 2016).

Por otro lado, es cierto que en el desarrollo de software lo más probable es que cometamos errores ya sea al interpretar requerimientos o al no pensar en todos los posibles casos de error de una solución. Es posible que diseñemos una solución que

funcione y aun así que no cumpla con las expectativas de los requerimientos, velocidad, uso de recursos, aspectos visuales, etc.

Si abordamos el desarrollo de software desde un punto de vista más científico, podemos plantear hipótesis que nos permitan experimentar en estas piezas o partes que en conjunto formarán un sistema de software. La idea es plantear hipótesis que nos lleven por caminos distintos en los que podamos programar de cierta forma alguna pieza de software, utilizar mediciones objetivas basadas en métricas, para evaluar la eficacia de la pieza nueva de software y posteriormente tomar una decisión de si la pieza desarrollada es mejor o peor a la previamente existente.

Este proceso de formulación de hipótesis y de experimentación es el que ha llevado al conocimiento científico a crecer constantemente, pues construye conocimiento poco a poco de manera escalonada, tomando como base el conocimiento ya existente. En términos de software es prácticamente imposible diseñar una solución completa y correcta en una sola iteración, si diseñamos software con la capacidad de quitar viejas piezas y colocar nuevas, para su posterior evaluación y retroalimentación, nos acercaremos a la solución final conforme realicemos iteraciones (Farley, 2021).

En cuanto al desarrollo iterativo, uno de los marcos de trabajo más populares del desarrollo ágil es Scrum, el cual es una metodología que promueve el desarrollo iterativo, la colaboración, así como la mejora continua (Chukwurah y Aderemi, 2024). Scrum, comienza con un proyecto que principalmente contiene la visión de un sistema o de un producto a ser desarrollado. La visión es transformada por el dueño del producto en un backlog de producto, la cual es una lista de requerimientos en forma de épicas e historias de usuario.

Las historias de usuario tienen el formato de “Yo como [Rol, persona, usuario] necesito, quiero, un producto, sistema, característica, etc. Que me permita realizar [cierta necesidad de negocio]. Las historias de usuario le permiten a un equipo de desarrollo a partir del texto de la necesidad de negocio abstraer tareas de tipo técnico, que se desarrollan en Sprints, y que permitirán alcanzar estos objetivos de negocio.

Scrum utiliza iteraciones llamadas Sprints, que son periodos de tiempo definidos, típicamente entre dos a cuatro semanas de tiempo. En estos periodos de tiempo el dueño del producto le entrega al equipo una lista de historias de usuario que el equipo debe estimar si es capaz o no de convertirlas en entregables funcionales para el final del sprint. Posteriormente el equipo de desarrollo es libre de entender de mejor forma los requerimientos de las historias de usuario y evaluar técnicamente cómo construir la funcionalidad mediante un proceso creativo.

Al final de un sprint el equipo debe presentar el incremento de la funcionalidad al dueño del producto, de forma que se puedan realizar las evaluaciones de lo que se ha desarrollado y, por otro lado, se pueda adaptar el estatus del proyecto actual y evaluar sus futuros incrementos (Agile 42, 2023).

Para el desarrollo del proyecto de esta tesis se tomó en cuenta las ideas de la ingeniería de software moderna, así como el desarrollo iterativo e incremental de Scrum para iterar e irnos acercando a la primera versión de la plataforma, a la cual decidí llamar Turing en honor a Alan Turing. Principalmente se ha cuidado de diseñar el software de manera que sea capaz de ser modificado o de agregar nuevas implementaciones de partes que sean necesarias cambiarse o ser remplazadas.

Otro tema básico al escribir buen software es la elección del patrón de arquitectura de software a utilizar. Para Ford y Richards “La arquitectura de software es sobre los elementos u objetos que son difíciles de cambiar después” (2020, p. xiii). Podemos entender a la arquitectura del software como los planos sobre los que se construye un proyecto, los cimientos de un edificio o la base de un automóvil, el camino a seguir.

Uno de los retos con respecto a la arquitectura de software radica en que la mayoría de la literatura existente está llena de acrónimos y de referencias cruzadas. Soluciones que fueron utilizadas en el pasado y que eran los paradigmas dominantes, hoy en día son prácticamente historia pues el contexto y la tecnología han cambiado. Para Richards y Ford “La historia de la arquitectura de software está llena de cosas que los

arquitectos han probado, solo para darse cuenta de los dañinos efectos colaterales de las decisiones tomadas" (2020, p. 2).

La arquitectura se debe definir a partir de cuatro elementos principales:

1. En términos de la **estructura**, que se refiere a como se organizan los distintos componentes del sistema:
    - Monolito.
    - Capas.
    - Microservicio.
2. Debe incluir además **características del software** deseado:
    - Disponibilidad.
    - Confiabilidad.
    - Agilidad.
    - Seguridad.
    - Escalabilidad.
    - Tolerancia a fallos.
    - Elasticidad.
    - Recuperabilidad.
    - Desempeño.
    - Facilidad de prueba.
    - Facilidad de aprendizaje.
    - Facilidad de despliegue.
3. **Las decisiones arquitecturales**, que se refiere a las reglas a partir de las cuales un sistema debería ser construido.
4. **Los principios de diseño**, se refiere a buenas prácticas o estandarizaciones, por ejemplo, se debe preferir la comunicación asíncrona entre servicios por encima de la síncrona.

Podemos definir así que la arquitectura de software es un concepto bastante amplio, el cual abarca decisiones en la definición del cómo se construirá un sistema de

software, y que además debe cubrir conceptos como la estructura, las características y reglas deseadas del software, así como los principios de diseño sobre las cuales se construirá.

Una de las arquitecturas de software más populares y ampliamente utilizadas hoy en día es la de los microservicios, fue acuñada con ese nombre incluso antes de que fuera popular por Martin Fowler y James Lewis en una entrada de blog en marzo de 2014 (Lewis y Fowler, 2014). La arquitectura de microservicios está fuertemente inspirada en las ideas del diseño orientado por el dominio (Domain Driven Design. DDD por sus siglas en inglés), específicamente con respecto a una idea del DDD llamada contexto delimitado.

El contexto delimitado se refiere a que un microservicio está construido de forma que sus partes internas como el código, los esquemas de datos y lógica están acopladas para producir alguna funcionalidad, pero estos componentes nunca están acoplados a nada fuera de su contexto, en palabras más sencillas un “bundled context” se refiere a que un microservicio debe atender solo una necesidad de negocio particular es decir que nos permite agrupar de forma lógica modelos de dominio.

Definimos que dos o más artefactos, servicios, o microservicios están acoplados cuando, un cambio en alguno de ellos puede ser que requiera un cambio en el otro para mantenerlos funcionando correctamente (Ford et al, 2022, p.14). Una de las desventajas de los microservicios es que, uno de sus principales objetivos es el alto grado de desacoplamiento. A cambio de ello una arquitectura de microservicios tienen a duplicar mucho código pues existirán elementos comunes dentro de los microservicios y este código debe existir y ser mantenido a lo largo de los múltiples microservicios existentes en una organización.

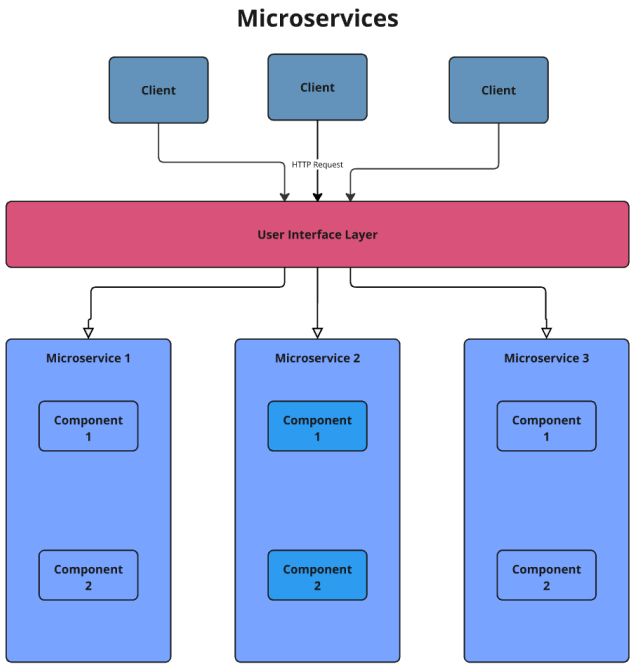
Los conceptos centrales para la implementación de microservicios son:

- Implementación de unidades separadas.
- Cada microservicio se debe desplegar de forma independiente.
- Facilidad de entrega continua y escalabilidad.

- Se trata de una arquitectura distribuida.

Ilustración 8

Arquitectura de microservicios



Nota. Hecho por el autor.

En la ilustración 8, ejemplificamos la arquitectura basada en microservicios, en ella es posible que existan múltiples interfaces de usuario, o una sola integrada por ejemplo una aplicación para dispositivo móvil y una aplicación web. La idea es que exista una capa intermedia entre el usuario y los microservicios, típicamente la conocemos como *Gateway*, el cual nos permite centralizar las peticiones y enrutarlas a los servicios correctos. En el *Gateway* podemos centralizar el proceso de autorización y autenticación y realizar un procedimiento de balanceo de carga.

Después de que el *Gateway* reciba una petición del usuario, la autentique y balancee, esta petición es enrutada hacia el microservicio correcto y procesada en el mismo, este procesamiento puede realizarse de manera síncrona o asíncrona. Es importante

considerar que pueden existir múltiples instancias de cada microservicio, esto para permitir que el sistema escale.

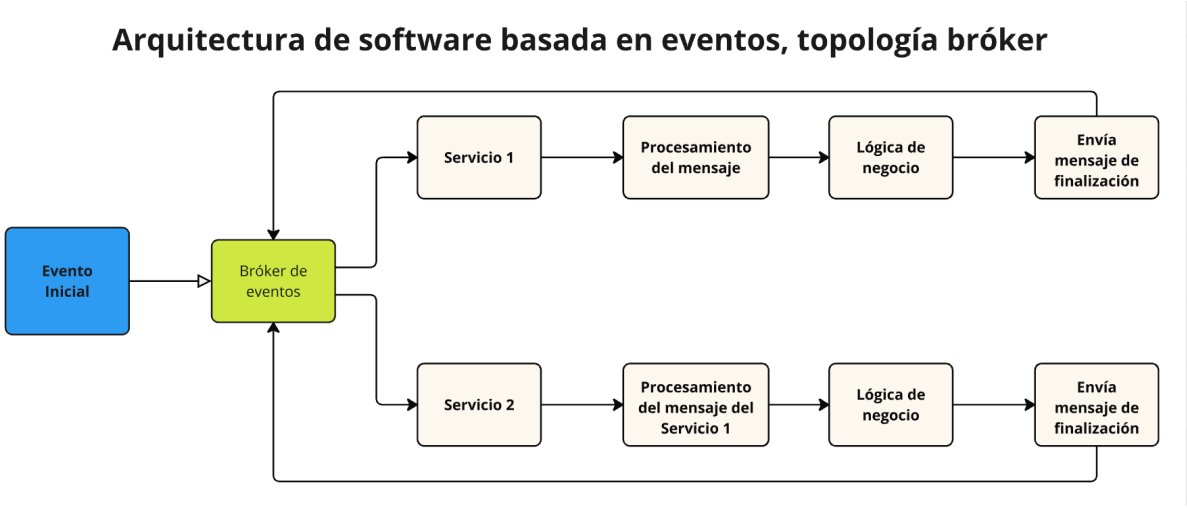
Además de esta arquitectura, existe otra arquitectura popular llamada arquitectura basada en eventos, la cual definimos como “una arquitectura distribuida y asíncrona usada para producir aplicaciones altamente escalables” como argumenta Richards (2015, p.179), esta arquitectura es altamente adaptable y se puede utilizar tanto en aplicaciones de software pequeñas como en las más grandes y complejas.

Existen 2 topologías principales de la arquitectura basada en eventos, por un lado, tenemos la topología con mediador, en esta arquitectura se puede orquestar o controlar finamente el orden de los pasos a seguir dado un determinado evento a través del mediador. La otra topología de la arquitectura basada en eventos es la del bróker, está a diferencia de la topología con mediador consiste en poseer cadenas de eventos sin la necesidad de que exista un orquestador donde los mensajes se envían a través de los servicios (ídem).

En esta arquitectura existe un evento inicial, este evento inicial es enviado a un bróker, como RabbitMQ o ActiveMQ, este evento posteriormente es consumido por un procesador de eventos; los procesadores de eventos son servicios que se encargan de realizar un cierto proceso de negocio, cuando este proceso está terminado, es el mismo procesador de eventos que notifica al bróker que ya ha terminado con su ejecución a través de un evento de procesamiento (Ford et al, 2022). Este proceso de encadenamiento se repite hasta que se han realizado todos los pasos de flujo de negocio.

Ilustración 9

Arquitectura de software basada en eventos, topología de bróker



Nota. Hecho por el autor.

En la ilustración 9, un evento inicial en azul seria recibido en el bróker, este evento inicial es consumido por un servicio el cual desencadena una serie de lógicas de negocio y cuando el procesamiento es finalizado, un evento extra es enviado al bróker, su propósito es avisar que ya ha terminado de realizar su procesamiento y que puede comenzar el siguiente.

Este evento extra es útil ya que nos permite desencadenar el inicio de otro proceso de negocio para el servicio 2. Como podemos imaginar, a partir de un solo evento podemos ir desencadenando cuantos procesos de negocio sean necesarios y orquestarlos de la manera que sea que necesitemos.

Particularmente en la solución propuesta, se utilizará una arquitectura híbrida, basada en microservicios y en eventos, por un lado el servicio principal que dará soporte a las operaciones del usuario será una arquitectura de microservicios, además de ello cuando se realiza el procesamiento de los engroses, es necesario tener bastante tiempo disponible para hacerlo, ya que los engroses serán participes de una serie de procesos de IA, procesamiento por lotes, expresiones regulares, limpieza y de procesamiento natural del lenguaje que requieren tiempo.

La estrategia propuesta es que cuando sea necesario procesar un texto normado nuevo para agregarlo a la plataforma, el usuario del sistema nos proporcione un archivo a procesar y como no es factible que un usuario espere una gran cantidad de tiempo hasta que la plataforma termine la ejecución y el análisis del engrose, utilizamos una estrategia de procesamiento asíncrono.

El usuario subirá su documento y éste será almacenado rápidamente en la plataforma, dándole respuesta al usuario casi de manera inmediata, una vez que su archivo está siendo procesado; esto genera que el usuario tenga una sensación de velocidad y alta capacidad de respuesta al utilizar el sistema. Posterior a la carga del documento, y de acuerdo con las características de las arquitecturas basadas en eventos, se producirá un evento inicial de procesamiento, el cual se enviará a nuestro bróker, en este caso un RabbitMQ. Fieles a la arquitectura, este evento inicial de procesamiento es posteriormente atendido por un servicio escrito en el lenguaje de programación Python para encadenar los procesos de procesamiento natural del lenguaje y de IA.

Es en este servicio escrito en Python en el que se empleará el gran modelo de lenguaje para generar el resumen del texto normado, además de ello se generan las palabras clave del documento, que permitirán conceptualizar el documento sin tener que leerlo de forma completa. Adicionalmente a estas tareas, será ejecutado el modelo de detección de leyes y de tesis jurisprudenciales entrenado particularmente para llevar a cabo la tarea de detectar y persistir las tesis y leyes nombradas en el texto.

Spring Boot siendo un marco de trabajo de código abierto, es capaz de crear servicios que se ejecutan sobre la máquina virtual del lenguaje de programación de Java (Heckler, 2021). Spring Boot surgió en el año 2003 como marco de trabajo y desde ese entonces se ha convertido en el marco de trabajo dominante para la programación de software basado en el lenguaje de programación Java (Luan, 2021), como nos explica el autor Meric (2024), Spring Boot nos ofrece una gran variedad de ventajas para utilizarlo aunque la mayor de ellas es su madurez como marco de trabajo y el gran ecosistema que se ha formado alrededor tanto para trabajar con bases de datos,

la nube, herramientas de monitoreo y trazabilidad, entre otros (Puig, 2024). Podemos entender así al siguiente listado como las principales características de Spring Boot:

- Configuración veloz, se puede crear fácilmente un nuevo proyecto.
- Es un marco de trabajo en el que se puede implementar microservicios de forma nativa.
- La configuración de Spring Boot es amigable para el usuario.
- Gran cantidad de herramientas para trabajar e integrar con este marco de trabajo.
- Está listo para ser llevado a la nube utilizando las herramientas de Spring Cloud.
- Es fácil implementar test para asegurar la integridad de las soluciones de negocio.
- La comunidad es enorme.

Para la creación del *Backend* de la plataforma de engroses, se ha elegido el principal servicio, un microservicio creado con la tecnología de Spring Boot, el cual se conecta con la plataforma de interfaz de usuario realizada en la tecnología Angular mediante un *Gateway* de acuerdo con un patrón arquitectónico de microservicios.

Para Khan (2019), un ingeniero perteneciente a Google y miembro del equipo de Angular, este marco de trabajo se destaca como uno de los más utilizados en la industria y que facilita la creación de aplicaciones web, así como aplicaciones nativas y de escritorio. Este marco de trabajo emplea el lenguaje de programación TypeScript, desarrollado por Microsoft y liberado al público en octubre de 2012. TypeScript ofrece la capacidad de incorporar tipos de datos al código, en contraposición al clásico JavaScript, que se caracteriza por su tipado débil. Esta característica ofrece una ventaja dado que permite identificar errores antes de su aparición en el navegador o durante la ejecución de la aplicación. Esto confiere mayor fiabilidad al código de TypeScript y al de Angular, en comparación con el código desarrollado directamente en JavaScript. Esta confiabilidad se debe a que el código de TypeScript requiere ser

compilado y transcrito a JavaScript, un proceso inviable si los tipos de datos son incorrectos.

Angular puede interpretarse también como una repositorio de bibliotecas JavaScript que se pueden emplear para la creación de aplicaciones web, empleando una arquitectura jerárquica fundamentada en componentes para la construcción de la interfaz de una página web. Una aplicación Angular se compone de distintos componentes que pueden ser conceptualizados mediante un esquema jerárquico. En el núcleo del esquema se ubica un componente primordial denominado componente de aplicación, y los componentes que el programador emplea serán componentes descendentes o subyacentes del componente de aplicación. El concepto de módulo también se encuentra presente en Angular, permitiendo la agrupación de componentes de Angular que comparten una funcionalidad comparable. Los componentes pueden incluir un componente de búsqueda, un componente de barra de navegación o un componente de tarjetas con contenido.

El lenguaje de programación Python fue desarrollado en la década de 1980 por Guido Van Rossum, y desde ese momento han emergido diversos marcos de trabajo destinados a la creación de interfaces de programación de aplicaciones (APIs), microservicios y aplicaciones de software utilizando Python. Una de las principales importancias del lenguaje de programación Python radica en que, en la actualidad, se posiciona como uno de los principales lenguajes de programación para la implementación de la IA y el procesamiento del lenguaje natural, junto con R y C++. Según la encuesta de tecnologías más populares, Python se posiciona en la tercera posición entre todos los lenguajes de programación en el año 2024 (Stack Overflow, 2024).

Para corroborar la afirmación de que Python continúa siendo el rey en la implementación de la IA, es relevante destacar que las bibliotecas de IA de mayor popularidad son:

- PyTorch.

- Scikit-Learn.
- TensorFlow.
- Keras.
- LangChain.

Todas las anteriores escritas para funcionar e integrarse con el lenguaje de programación Python, en la ilustración siguiente se puede observar que dentro del top 10 de librerías más populares, encontramos varias relacionadas con el ecosistema de Python y la IA, a nombrar, Numpy, Pandas, Scikit-Learn, TensorFlow, etc.

Ilustración 10

Marcos de trabajo más utilizados en la industria

Nota. Tomado de (Stack Overflow, 2024).

De esta información sabemos que es necesario tener al menos uno o varios servicios de Python integrados con la plataforma de análisis de engroses, para poder tener un servicio tanto de IA como de procesamiento del lenguaje natural. Para suplir esta necesidad utilizaremos la tecnología FastAPI y Python como lenguaje de programación para el análisis de los engroses e integración de funcionalidades de IA y procesamiento del lenguaje natural.

FastAPI es el símil de Spring Boot y Angular, es un marco de trabajo, pero para el lenguaje de programación Python, nos permite generar servicios y APIs con alta concurrencia y arquitecturas basadas en capas, así como microservicios. Algunas de

las alternativas a este marco de trabajo para el lenguaje Python son: el ultraligero Flask y Django, un marco de trabajo para desarrollo web completo (Lubanovic, 2024).

FastAPI se sitúa en medio entre Django un framework “pesado” que permite desarrollar sitios web desde los servicios con lógica de negocio hasta las vistas en la interfaz de usuario, utilizando una arquitectura MVC (modelo vista controlador) y Flask un marco de trabajo que realmente te deja ser libre e implementar tus servicios de la forma que tú quieras pero que no está pensando para realizar una implementación completa de servicios del lado del servidor y además interfaces de usuario.

FastAPI tiene ventajas comparado con los otros dos marcos de trabajo más populares del ecosistema de Python:

- El desempeño.
- La velocidad de desarrollo.
- La documentación autogenerada y las páginas de pruebas.

Al igual que en un proyecto de software no es viable desarrollar una implementación propia de base de datos, sino utilizar alguna que ya exista y sea confiable, para resolver la necesidad de ofrecer al usuario un motor de búsqueda potente dentro de una plataforma, el ingeniero tiene a disposición diversos motores de búsqueda, siendo un motor de búsqueda aquel componente de software que nos permite realizar recuperaciones y búsquedas de información textual. Estos componentes de software consisten básicamente en 4 elementos (Gao et al, 2012).

- Una base de datos de texto.
- Un proceso para generar un índice de palabras con el propósito de disminuir los tiempos de búsqueda.
- Un componente que nos permita realizar búsquedas.
- Filtrar, ordenar y posicionar los resultados de una operación de búsqueda. Normalmente al usuario solo le interesan los resultados más relevantes dada una consulta.

De acuerdo con Chang (2023) que analizó el desempeño de los motores de búsqueda de código abierto y los comparó con el motor comercial predominante, los motores de búsqueda más potentes son:

- Elastic Search (Comercial).
- Apache Solr (Código abierto).
- Typesense (Código abierto).
- Algolia (Comercial de código abierto).

De estas opciones la más poderosa es Elastic Search, ya que tiene características como la escalabilidad, resiliencia y alta disponibilidad, al tratarse de un sistema distribuido. Una de sus principales desventajas es que es costoso y no se recomienda para equipos de desarrollo pequeños ya que requiere de conocimiento para configurarlo correctamente.

Meili Search es un motor de búsqueda de código abierto, el motor nos provee la funcionalidad de buscar texto de manera rápida en milisegundos, la capacidad de aplicar filtros a la información, así como posicionar los resultados más relevantes de una búsqueda de manera personalizada. Meili Search es un servicio que se puede contratar y desplegar en la nube para que escale y sea resiliente, sin embargo, también es posible correr el servicio en un ambiente contenerizado e implementarlo en la infraestructura que queramos, ya sea en un ambiente de infraestructura propia o en la nube. La desventaja de esta aproximación es que no contaríamos con un servicio resiliente y de alta disponibilidad.

Por otro lado, una de las mayores ventajas de este motor de búsqueda es que se integra de manera sencilla con Angular y Spring Boot haciéndolo ideal para nuestra aplicación. Adicionalmente la comunicación tanto para cargar información nueva como para realizar búsquedas es realizada utilizando el protocolo HTTP y los verbos REST de POST y GET con objetos universales de tipo JSON. Haciendo que su uso sea extremadamente sencillo a comparación de sus contrapartes Elastic Search y Apache Solr que requieren configuraciones para funcionar en estos niveles.

Diseño

Basándome en lo presentado en la sección de antecedentes se han identificado los siguientes componentes para la solución:

1. Una aplicación en Angular que permita al usuario:
    a. Navegar a través de los engroses que se encuentren disponibles en la plataforma.
    b. Realizar una búsqueda flexible sobre la base de datos de engroses, pudiendo buscar por cualquier elemento dentro de un engrose.
    c. Ser capaz de ver el detalle de un engrose en particular.
        i. Identificar las personas nombradas en el engrose.
        ii. Identificar las tesis jurisprudenciales citadas.
        iii. Identificar las palabras claves del engrose.
        iv. Ser capaz de ver un resumen del documento para obtener una visión general del mismo.
        v. Poder consultar el contenido del documento original.
    d. Ser capaz de cargar un nuevo engrose y que éste sea procesado automáticamente.
2. Una arquitectura de microservicios escrita en Spring Boot que permita dar soporte a los casos de uso necesarios para la aplicación de Angular.
3. Un API escrita en Python con la exposición de servicios relacionados con el procesamiento natural del lenguaje, así como de IA con una orientación de arquitectura basada en eventos.
4. Un bróker de mensajería ligero, particularmente RabbitMQ para habilitar la comunicación asíncrona entre el microservicio de Spring Boot y los servicios costosos escritos en Python.
5. Una base de datos de documentos que estructure y administre la información estructurada a partir de los engroses.
6. Un motor de búsqueda que nos permita habilitar las capacidades de búsqueda completa en milisegundos de cualquier sección de un engrose.

Casos de uso

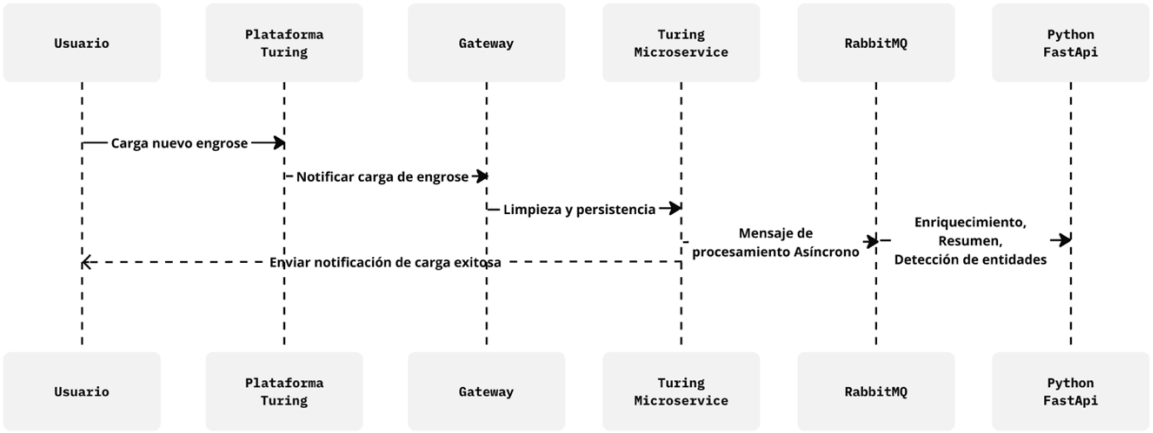
Cargar un nuevo engrose

La funcionalidad de poder agregar al sistema nuevos engroses, el cual se detalla en forma de diagrama en la imagen posterior. El flujo comienza cuando el usuario toma un archivo de engrose y lo agrega en el sistema. Se presiona un botón en la plataforma Turing y esta envía una petición HTTP hacia el *Backend*, en este caso al *Gateway*. Después, la petición se enruta hacia el microservicio Turing, en el método de cargar un nuevo documento. Este método se encarga de realizar una limpieza muy simple del documento y de persistirlo en base de datos, así como en el sistema de archivos para ser capaces de consultar el documento originalmente cargado.

Este procesamiento debería tomar de decenas a cientos de milisegundos, y antes de contestarle al usuario se envía un mensaje de procesamiento asíncrono a RabbitMQ. Este mensaje es consumido por un servicio escrito en Python que de manera asíncrona y ya sin el tiempo encima, se encarga de enriquecer el documento, encontrar las palabras clave, generar un resumen, detectar las tesis de jurisprudencia y las leyes citadas en el engrose, así como las personas nombradas y de persistir todo esto en la base de datos.

Ilustración 11

Caso de uso para cargar un engrose nuevo.



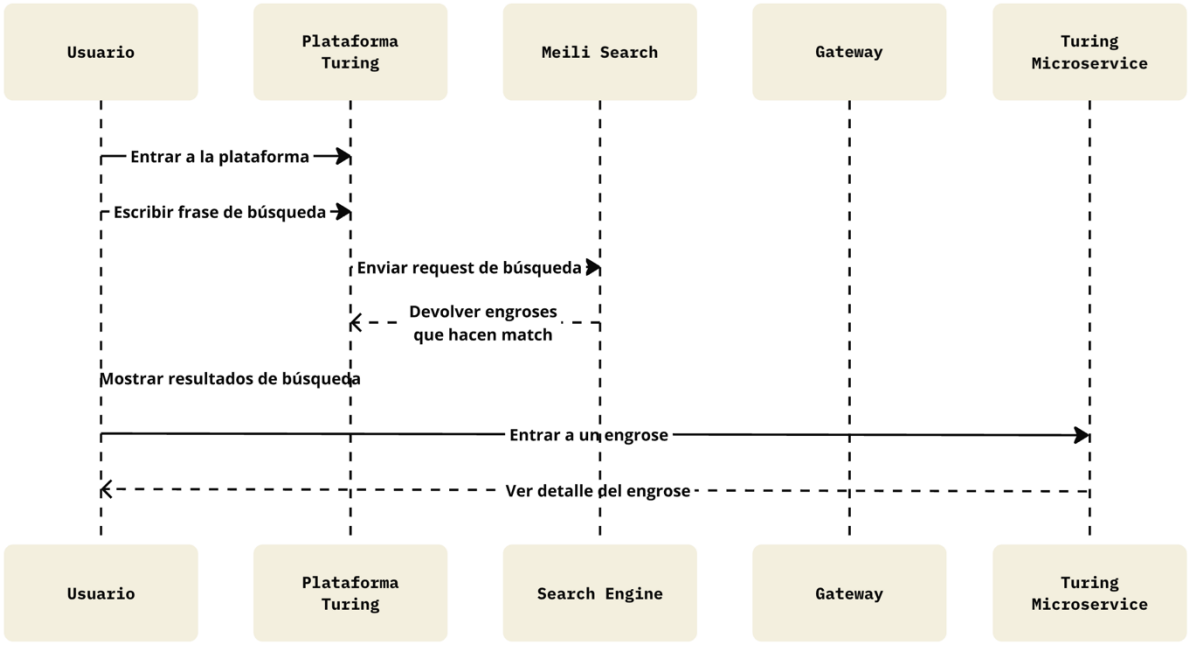
Nota. Hecho por el autor.

Búsqueda de un engrose

Este caso de uso describe el flujo para la búsqueda de documentos en la Plataforma Turing. El usuario ingresa a la plataforma y escribe una frase de búsqueda, que se envía al motor de búsqueda Meili Search a través de la Plataforma Turing. Meili Search procesa la solicitud y devuelve una lista de documentos (engroses) que coinciden sin importar acentos u otras particularidades con la frase ingresada. La Plataforma Turing muestra estos resultados al usuario, quien puede seleccionar uno de los documentos para ver sus detalles. Para obtener esta información detallada, la Plataforma Turing envía una solicitud al *Gateway*, el cual se comunica con el Microservicio Turing. Los detalles del documento seleccionado se presentan al usuario en la plataforma. Este flujo garantiza una experiencia de búsqueda eficiente, integrando múltiples componentes para la recuperación y visualización de la información.

Ilustración 12

Búsqueda de engroses con una frase



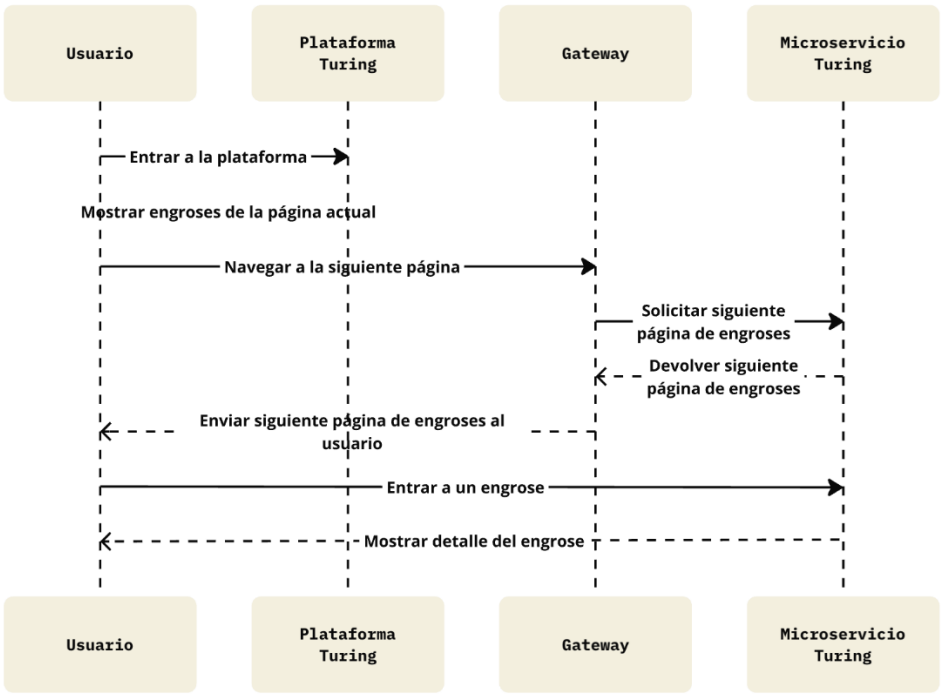
Nota. Hecho por el autor.

Navegación paginada

Este caso de uso describe el proceso de **navegación paginada** dentro de la **Plataforma Turing**. El usuario entra en la plataforma y visualiza los documentos correspondientes a la página actual. Si desea ver más documentos, navega hacia la siguiente página, lo que genera una solicitud desde la plataforma hacia el *Gateway*, que a su vez pide al Microservicio Turing la siguiente página de documentos. El microservicio devuelve los documentos de la página solicitada, que la plataforma muestra al usuario. Además, el usuario tiene la opción de seleccionar un documento (engrose) para ver sus detalles, lo que involucra otra comunicación entre la plataforma, el *Gateway* y el microservicio para mostrar la información específica del documento seleccionado. Este flujo permite una navegación eficiente entre páginas de resultados sin sobrecargar al sistema ni al usuario.

Ilustración 13

Navegación del sistema de forma paginada



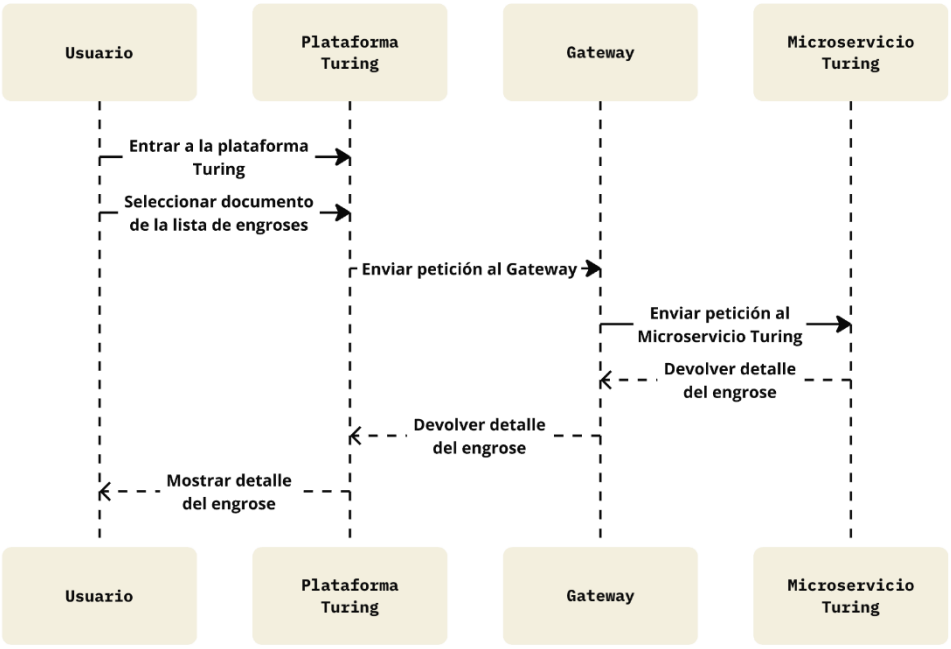
Nota. Hecho por el autor.

Visualización del detalle de un engrose

Flujo para visualizar el detalle de un documento en la Plataforma Turing. El usuario accede a la plataforma y selecciona un documento específico de la lista de resultados (engroses). La Plataforma Turing envía una solicitud al *Gateway*, el cual reenvía la petición al Microservicio Turing para obtener el detalle completo del documento seleccionado. El Microservicio Turing responde con la información detallada del documento, que se envía de vuelta a través del *Gateway* a la Plataforma Turing, donde se muestra al usuario. Este flujo asegura que el usuario pueda consultar la información detallada de un documento de manera eficiente.

Ilustración 14

Detalle de un engrose

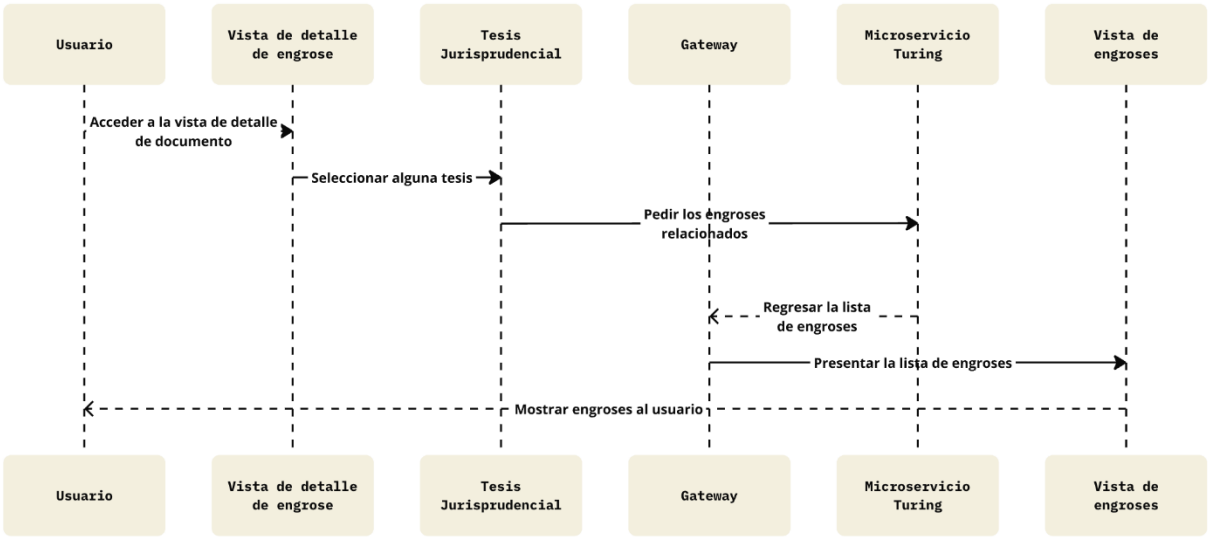


Búsqueda de documentos relacionados a una tesis jurisprudencial en particular

Un usuario puede identificar y acceder a documentos que hacen referencia a una tesis jurisprudencial específica. El usuario inicia la búsqueda al acceder a la vista de detalle de un engrose. A continuación, selecciona la tesis jurisprudencial de interés y solicita al sistema que le proporcione una lista de los engroses relacionados. A través del microservicio Turing, consulta una base de datos y devuelve una lista de engroses (documentos relacionados) que citan la tesis seleccionada. El sistema presenta esta lista al usuario en una vista de engroses, permitiéndole explorar los documentos encontrados.

Ilustración 15

Búsqueda de documentos con citas a tesis jurisprudenciales particulares



Nota. Hecho por el autor.

Diagrama de base de datos

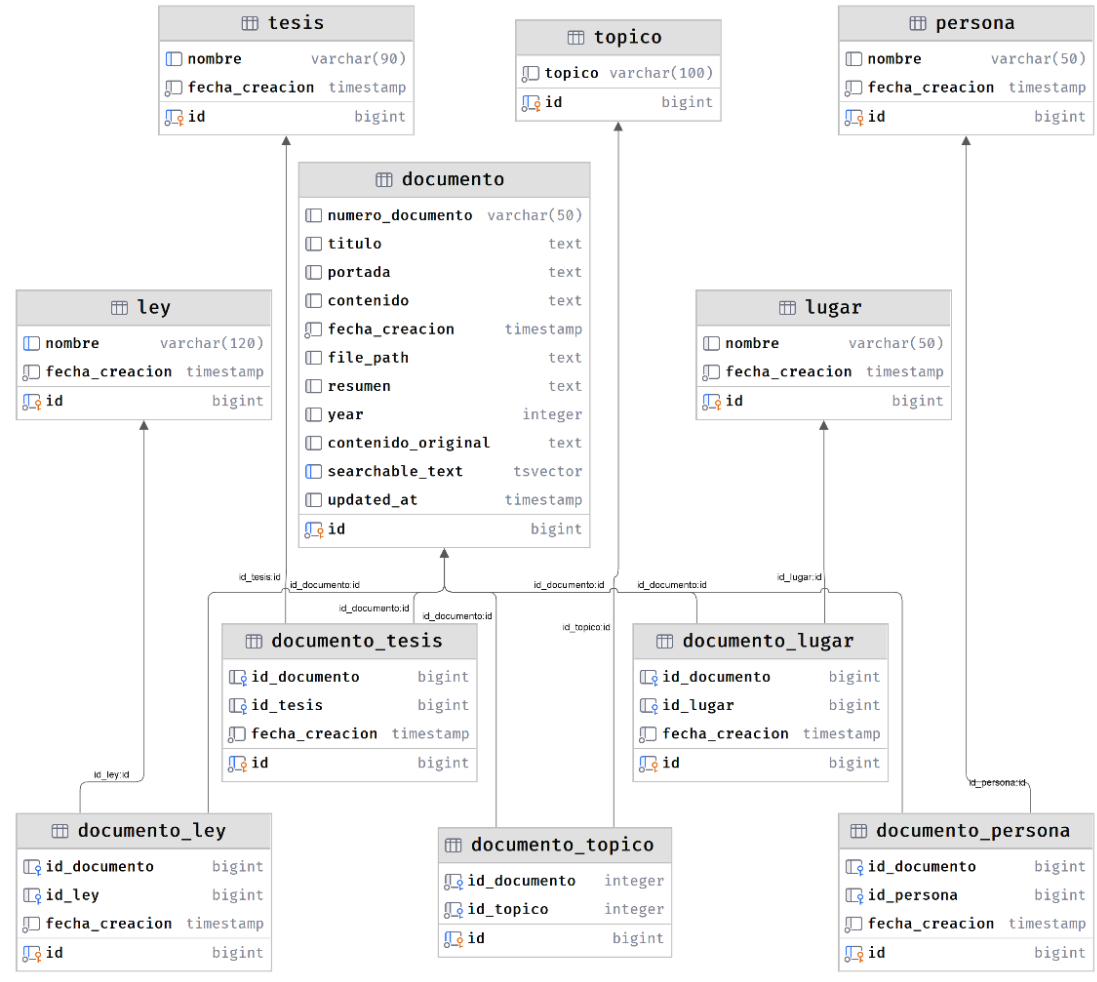
El diagrama relacional a continuación muestra una estructura de datos diseñada para gestionar información sobre documentos legales, sus temas, autores, lugares de origen y las tesis jurídicas a las que hacen referencia. El diagrama relacional y la base de datos es pequeña ya que está pensada para almacenar información relacionada a los engroses, pero no es su eje central el servicio de búsqueda de documentos, ya que eso lo tiene tomado el motor de búsqueda de texto.

La entidad central es el "documento" o engrose, que almacena metadatos como título, contenido, fecha de creación y un identificador único. Esta entidad se relaciona con otras entidades a través de tablas intermedias, como "documento_tesis", "documento_topico", "documento_lugar" y "documento_persona", lo que permite asociar múltiples tesis, tópicos, lugares y personas a un mismo documento. Además, se incluyen entidades para representar tesis jurídicas, tópicos, personas y lugares de forma independiente, con sus respectivos atributos. Esta estructura permite realizar

consultas complejas, como encontrar todos los documentos relacionados con una tesis específica o los documentos de un autor en particular, así como analizar tendencias y patrones en la producción de documentos legales.

Ilustración 16

Diagrama relacional de base de datos

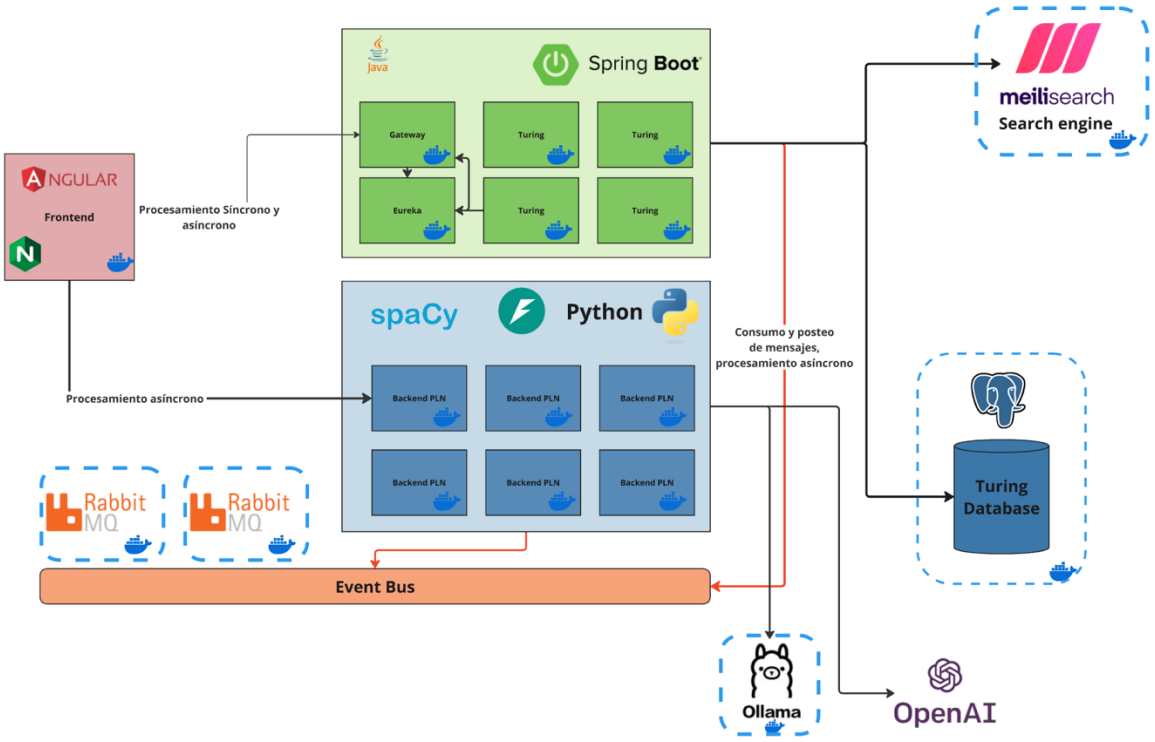


Nota. Hecho por el autor.

Arquitectura y componentes

Ilustración 17

Componentes del sistema



Nota. Hecho por el autor.

En el diagrama de la ilustración 17 de la plataforma Turing se presenta una arquitectura modular y escalable, con una clara división de responsabilidades entre sus diferentes elementos. En la parte superior, encontramos la interfaz de usuario desarrollada con Angular, que se comunica con un *Backend* construido en Spring Boot a través de *Gateway*. Este *Backend* se puede escalar creando varias instancias del microservicio Turing, los cuales se encargan de lógica principal de la aplicación. La comunicación entre los distintos componentes se facilita mediante un bus de eventos basado en RabbitMQ, lo que permite una alta desacoplación y escalabilidad. En la capa de datos, se utiliza una base de datos PostgreSQL, y se integran servicios de terceros como MeiliSearch para búsqueda y procesamiento de lenguaje natural con SpaCy y Python.

Para la tarea de generar resúmenes de los engroses, se incluyen varias implementaciones de los grandes modelos de lenguaje como Ollama en su versión que se ejecuta en local y sin salida a externos y OpenAI que implicaría un costo por cada consulta realizada y la salida de la información a terceros, en este caso OpenAI para poder hacer uso de los modelos cerrados basados en GPT.

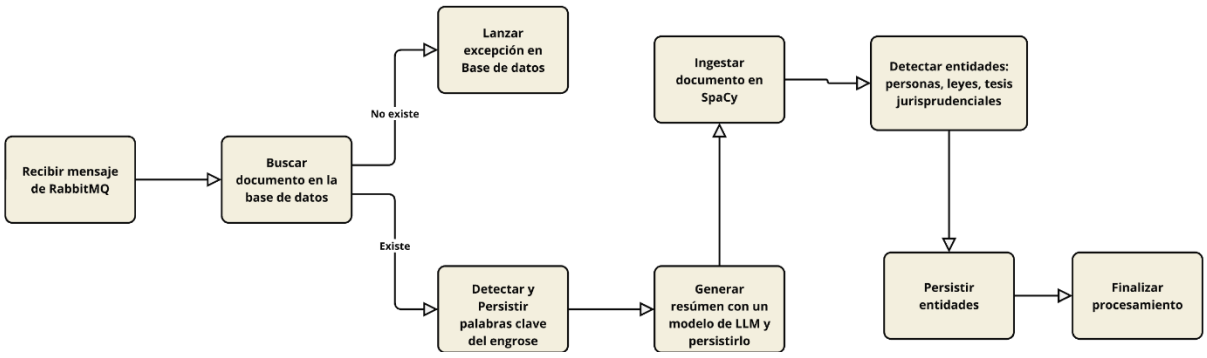
Además, en el diagrama se muestra:

- Una Arquitectura de microservicios: la plataforma se basa en una arquitectura que permite un desarrollo y despliegue independiente de cada componente utilizando las herramientas de Spring Cloud, como son el *Gateway* y Eureka.
- Comunicación asincrónica: el uso de RabbitMQ como bus de eventos facilita la comunicación asincrónica entre los diferentes componentes, lo que mejora la escalabilidad y la tolerancia a fallos, además nos permite tener más tiempo para procesar las solicitudes sin que el usuario deba esperar a que terminemos de realizar procesos que toman tiempo.
- IA: la integración de modelos de lenguaje como Ollama y OpenAI, junto con el uso de spaCy para procesamiento de lenguaje natural, muestra las capacidades de IA de la plataforma.
- Búsqueda: MeiliSearch se utiliza como motor de búsqueda, lo que permite realizar búsquedas rápidas y eficientes en grandes cantidades de datos

Algoritmo de enriquecimiento para textos normados

Ilustración 18

Algoritmo de enriquecimiento de un texto normado



Nota. Hecho por el autor.

El algoritmo de enriquecimiento de información para textos normados. Tiene como objetivo extraer y organizar información del texto de los engroses, transformando datos crudos en conocimientos estructurados y utilizables.

El proceso comienza con la recepción de un mensaje asíncrono en el bróker con el identificador único de un engrose. Este identificador único es utilizado para buscar en la base de datos el engrose y poder continuar con su procesamiento. En caso de que el engrose no se encuentre en la base de datos por alguna razón, se lanzaría una excepción en el proceso.

Posteriormente de que el engrose es recuperado de la base de datos, se utiliza su contenido para realizar un resumen corto del texto utilizando un modelo de lenguaje.

Además de que es procesado por el modelo de procesamiento natural del lenguaje para identificar y extraer entidades nombradas, como personas, leyes y tesis jurisprudenciales. Estos datos enriquecidos son posteriormente organizados y estructurados para facilitar su posterior almacenamiento.

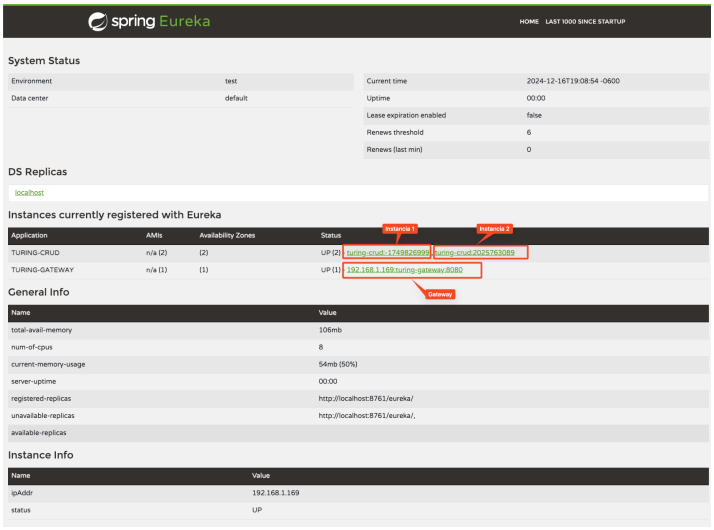
Desarrollo del sistema

Servicio principal de Eureka

Ya que la plataforma está pensada para escalar múltiples instancias de los microservicios que son posibles de ejecutarse al mismo tiempo, para poder balancear la carga entre ellos y que además la página web sea capaz de alcanzarlos utilizando un único punto de entrada, se configuró la arquitectura de spring cloud con un servidor Eureka y un *Gateway*. Para este caso Eureka nos sirve como un servicio de nombres, en ese sentido el servicio es utilizado por nuestro *Gateway* para enrutar las peticiones a distintas instancias de un microservicio. En la ilustración 19 muestro dos instancias del microservicio que sirve de *Backend* para la página web, las cuales se han auto registrado en el servidor de Eureka.

Ilustración 19

Tablero de Eureka, mostrando varias instancias de un microservicio



Nota. Hecho por el autor.

Esta auto registración es útil ya que permite tener múltiples instancias del mismo microservicio o de diferentes microservicios, en distintos servidores, no importa la dirección IP que éstos tengan. Ya que los servicios se auto registran en Eureka,

posteriormente será el *Gateway* el que se encargue de utilizar Eureka para encontrar las instancias de los microservicios adecuadas para el propósito que necesite.

Para la creación del servicio Eureka, utilizamos un microservicio en Spring Boot con una sencilla configuración, para la clase principal es necesario agregar la anotación @EnableEurekaServer detallado en la ilustración 21 en la línea 7 y en el archivo de propiedades agregar el puerto y el nombre que va a llevar nuestro servidor de Eureka, detallado en la ilustración 20.

Ilustración 20

Configuración del servidor de Eureka

Nota. Hecho por el autor.

En la línea 4 del archivo de configuración de Eureka, definimos que el microservicio no debe auto registrarse en Eureka, pues actúa como el servidor principal de nombres, y además agregamos unas configuraciones para desactivar la presentación de logs de Eureka y de Discovery en la consola, ya que nos presentan logs que realmente no nos dan mucho valor.

Ilustración 21

Archivo de propiedades del servidor de nombres de Eureka

```
spring.application.name=turing-service-registry
server.port=6060
#Eureka server configurations
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.server.enable-self-preservation=true

#Eureka server logging
logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF

    ⌘⇧L to Chat, ⌘I to Command
```

Nota. Hecho por el autor.

Para el *Gateway*, también es necesario crear un nuevo microservicio de Spring Boot, en este caso, la configuración también se encuentra centralizada en el archivo de propiedades en este caso escrito en YML, el cual se detalla en la ilustración 22.

Configuración de Eureka en el Gateway

Nuestro *Gateway* define 2 grandes grupos de propiedades, en principio las propiedades relacionadas con el enrutamiento de las peticiones que recibe el *Gateway* y posteriormente las propiedades relacionadas con Eureka.

Para empezar, definimos que el *Gateway* se encuentre en el puerto 8080, y en la sección de la línea 10 a la 15, escribimos una ruta que corresponde a los servicios de lógica de negocio que se encuentran en otro microservicio. El prefijo **lb** en la línea 11 define que nuestro *Gateway* va a realizar un balance de carga entre los microservicios que se encuentren disponibles y que sean del tipo “TURING-CRUD”.

Además, en la línea 13 definimos que todas las peticiones que lleguen a nuestro *Gateway* que comiencen con el prefijo “/turing-crud” sean enviadas a los servicios que se llamen “TURING-CRUD” como se define en la línea 11. En la línea 15 definimos que todas las peticiones que tengan el prefijo “/turing-crud” sean reescritas para

eliminar este prefijo, de forma que el microservicio reciba una petición que no lo contenga.

Ilustración 22

Propiedades del *Gateway*, conexión a Eureka y definición de las rutas necesarias para el servicio de lógica de negocio.

```yaml
server:
  port: 8080

spring:
  application:
    name: turing-gateway
  cloud:
    gateway:
      routes:
        - id: turing-service
          uri: lb://TURING-CRUD
          predicates:
            - Path=/turing-crud/**
          filters:
            - RewritePath=/turing-crud/(?<remaining>.*), /${remaining}

eureka:
  client:
    register-with-eureka: true
    service-url:
      defaultZone: http://localhost:6060/eureka
  instance:
    lease-renewal-interval-in-seconds: 30
    lease-expiration-duration-in-seconds: 90
```

Nota. Hecho por el autor.

En cuanto a la configuración de Eureka, que se define a partir de la línea 17, básicamente le indicamos a nuestro *Gateway* en qué dirección debe conectarse y que además debe registrarse en Eureka, adicionalmente definimos cuánto tiempo en segundos será utilizado para los periodos de renovación y de expiración en Eureka.

Configuración de Eureka en el microservicio principal

Para el microservicio que soporta la operación principal de la página web, se creó un microservicio igualmente en la tecnología de Spring Boot. En la cual agregué configuraciones para poderse conectar a el servidor de nombres Eureka y así, ser alcanzado desde la página web a través del *Gateway* en las líneas 19-24 se ha colocado la configuración relacionada a Eureka, en la cual se define que el microservicio debe registrarse en el servidor de nombres para estar disponible y ser capaz de ser alcanzado por el *Gateway*, además se colocan los intervalos de renovación y expiración en segundos

Ilustración 23

Archivo de propiedades del microservicio turing-crud

```
18
19  # eureka
20  eureka.client.register-with-eureka=true
21  eureka.client.service-url.defaultZone=http://localhost:6060/eureka
22  eureka.instance.leaseRenewalIntervalInSeconds=30
23  eureka.instance.leaseExpirationDurationInSeconds=90
24  eureka.instance.instance-id=${spring.application.name}:${random.int}   jromero,
```

Nota. Hecho por el autor.

Página web

Para el sitio web, al estar desarrollado en Angular, hacemos uso del servicio Http incluido con Angular para enviar peticiones y crear servicios para cada recurso que es necesario consumir. En la ilustración 23, muestro que para el marco de trabajo Angular existe un archivo muy útil que nos permite tener variables que pueden tomar distintos valores dependiendo de si estamos en un entorno productivo o si, por el contrario, nos encontramos en un entorno de desarrollo.

Ilustración 24

Archivo de ambientes de Angular

environment.development.ts

```
1   export const environment = {  no usages  jromero
2     production: false,
3     nlpBackendUrl: 'http://localhost:8000',
4     gatewayTuringUrl: 'http://localhost:8080',
5     meiliSearchHost: 'http://localhost:7700',
6     meiliSearchApiKey: '8Nt4*uHr!P2j'
7   };
```

Nota. Hecho por el autor.

Este archivo nos sirve para hacer más ágil el desarrollo ya que en el podemos poner por ejemplo, las direcciones tanto de desarrollo como productivas de los servicios de Del lado del servidor que utilizamos para nuestra aplicación, las direcciones de bases de datos de desarrollo y productivas, en nuestro caso tenemos un motor de búsqueda productivo y uno de desarrollo y de esta forma podemos realizar pruebas con el archivo ENVIRONMENT.DEVELOPMENT.TS y al salir a producción solo apuntar al archivo productivo, y no es necesario cambiar nada en el código, Angular usará las variables productivas.

Una de las ventajas de utilizar un *Gateway* para la parte de los microservicios es que, la página web solo debe conocer una única URL a la cual le realiza peticiones, así si por detrás de esta URL hay uno o cientos de microservicios eso ya es transparente y se abstrae esa complejidad del código. En la ilustración siguiente detallo el código del servicio que se encarga de traer los engroses desde el *Backend*, básicamente este servicio de engroses es capaz de traernos un documento a partir de su identificador único (línea 18 de la ilustración 25), traernos una lista de tamaño variable de documentos de forma paginada, es decir, la aplicación web puede pedir la página 1, 2, 3, …, n de tamaño m (línea 22).

Ilustración 25

Servicio de engroses en TypeScript

documento.service.ts

```typescript
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {environment} from "../../environments/environment";
import {Observable} from "rxjs";
import {Documento} from "./documento";

@Injectable({ Show usages  jromero +1 *
  providedIn: 'root'
})
export class DocumentoService {

  private baseUrl: string = environment.nlpBackendUrl + "/documentos";
  private springBackend: string = environment.gatewayTuringUrl + "/turing-crud/api/documents";

  constructor(private http: HttpClient) {  no usages  jromero
  }

  getDocumento(id: number): Observable<Documento> {  Show usages  jromero
    return this.http.get<Documento>(`${this.baseUrl}/${id}`);
  }

  getDocumentos(page: number, size: number): Observable<Documento[]> {  Show usages  jromero
    return this.http.get<Documento[]>(`${this.springBackend}/paginated/${page}/${size}`);
  }

  countDocuments(): Observable<number> {  Show usages  jromero
    return this.http.get<number>(`${this.springBackend}/count`);
  }
}
    ⌘⇧L to Chat, ⌘I to Command
```

Nota. Hecho por el autor.

Otra de las funcionalidades que posee este servicio es el de contar cuántos engroses hay actualmente cargados en el sistema. Esto me permite poder manejar la vista paginada y calcular el tamaño de las listas de forma dinámica.

Microservicio principal Turing del lado del servidor

El microservicio principal que ofrece soporte para las funciones de la aplicación web puede en general realizar las siguientes funcionalidades:

Para los **engroses** puede:

- Obtener un engrose a partir de su identificador único o lanzar una excepción si no se encuentra.
- Crear un engrose nuevo a partir de la información básica del mismo.

- Obtener una lista de engroses de forma paginada.
- Contar cuántos engroses se encuentran disponibles en la base de datos.
- Dado un identificador único de un engrose:
    - Asociar el engrose con una lista de leyes.
    - Asociar el engrose con una lista de lugares.
    - Asociar el engrose con una lista de tesis jurisprudenciales.
    - Cargar el engrose al motor de búsqueda e indexado.
    - Generar un resumen del engrose.
    - Generar el top 20 de palabras clave del engrose.

Para las **leyes** puede:

- Obtener una ley por su nombre o lanzar una excepción si la ley no existe en el sistema.
- Crear una ley a partir de su nombre.
- Obtener todas las leyes relacionadas a un engrose.
- Obtener todas las leyes conocidas por el sistema.

Para las **tesis jurisprudenciales** puede:

- Obtener todas las tesis jurisprudenciales conocidas por el sistema.
- Crear una nueva tesis.
- Obtener todas las tesis relacionadas a un engrose.
- Obtener una tesis jurisprudencial por su nombre.
- Generar una distribución de tesis, es decir dada una tesis particular, generar los datos para un histograma en dónde se muestre, cuál es el lugar de la tesis jurisprudencial comparada con el universo conocido de tesis.

Para las **palabras clave** de los engroses puede:

- Crear una palabra clave.
- Buscar una palabra clave por nombre.
- Obtener todas las palabras clave de un engrose.

- Obtener todas las palabras clave.

Para una **persona** puede:

- Obtener todas las personas relacionadas a un engrose.

Backend **de procesamiento de lenguaje natural**

En cuanto al *Backend* principal de procesamiento de lenguaje natural tiene algunas capacidades importantes:

Para los engroses:

- Es capaz de subir un archivo en formato Word al sistema.
    - Leer su contenido párrafo por párrafo y realizar una limpieza básica del texto para su posterior procesamiento.
    - Obtener una “portada” del engrose que será posteriormente la que sea presentada en la aplicación web.
    - Obtener el número de documento a partir del texto, este número de documento posee un formato de numero/año.
    - Obtener el año del documento.
    - Persistir el engrose en la base de datos.
- Es capaz de subir una carpeta de archivos en formato Word al sistema (este procesamiento está pensado para hacerse por lotes y no en la aplicación web), este procesamiento por lotes replica todos los pasos del punto anterior, aunque lo realiza en forma de lotes.
- Generar el resumen de un engrose.
- Generar las palabras clave de un engrose.

Procesamiento asíncrono

Una vez que se hace el preprocesamiento de un engrose y se persiste en la base de datos, se envía un mensaje de procesamiento a RabbitMQ, el cual detona las siguientes capacidades del *Backend*:

- Utilizando el algoritmo de Latent Dirichlet Allocation genera el top 20 de palabras clave del engrose y los persiste en el sistema.
- A través de una llamada a un gran modelo de lenguaje como Llama o inclusive algún modelo de OpenAI generamos un resumen del engrose, esta opción es sensible ya que si se utilizan modelos de OpenAI cada llamada genera un costo que se debe cubrir, mientras que, si se utiliza un modelo local, las llamadas a estos modelos no generan como tal un costo, sin embargo, es necesario contar con una tarjeta de video potente para poder dar soporte a la ejecución de un gran modelo de lenguaje de forma local.
- Detectar las personas nombradas en el engrose utilizando la librería de Python, SpaCy, posteriormente persistir las personas nombradas en el sistema.
- Detectar las tesis jurisprudenciales nombradas, así como las leyes nombradas, utilizando el modelo entrenado para realizar esa tarea en conjunción con la librería de Python SpaCy, posteriormente las entidades detectadas son persistidas en el sistema.

Desarrollo y evaluación del modelo

Para generar el modelo de clasificación de tesis jurisprudenciales, se establecieron los patrones de las trescientas leyes del marco jurídico mexicano, además de siete patrones de detección de tesis jurisprudenciales, una de las principales dificultades de definir patrones es su traslape. Para la biblioteca SpaCy un token dentro de un texto, no puede tener más de una clasificación y cuando definimos patrones que se traslapan, es posible que detectemos la citación de una tesis jurisprudencial múltiples veces. Esto sugiere que otra opción a tomar en cuenta seriamente sea la de emprender la ardua tarea de generar datos de entrenamiento de forma manual para alimentar el entrenamiento del modelo.

Adicionalmente a los patrones de clasificación de tesis y de leyes, en el entrenamiento de un modelo de aprendizaje automático supervisado, es necesario contar con datos de entrenamiento y de evaluación, en esta tarea se utilizó un corpus de aproximadamente 268,0000 tokens. Divididos en dos conjuntos, el conjunto de datos para entrenar, y el conjunto de datos para validar la efectividad del modelo.

En el fragmento de código de la ilustración previa, se leen todos los archivos de una carpeta (línea 29 y 38), cada archivo es procesado por la biblioteca de SpaCy y se le aplican los patrones de Leyes y tesis jurisprudenciales (línea 45), en la línea 49 ordenamos de manera aleatoria los documentos y posteriormente los dividimos en dos conjuntos, datos de entrenamiento y datos de verificación (línea 50 y 51 respectivamente).

Ilustración 26

Fragmento de código para generar los datos de entrenamiento y de verificación

```python
def get_custom_matcher(blank_model ) -> Matcher:  1 usage  Julio César Romero
    matcher = Matcher(blank_model.vocab)
    matcher.add( key: "LEY", law_patterns_list)
    matcher.add( key: "TESIS", thesis_patterns_list)
    return matcher

def main():  jromero +1 *
    blank_model = spacy.blank("es")
    path_folder ="../training_data/"
    file_lister = FileLister(path_folder,  supported_extensions: [".txt"])
    file_list = file_lister.list_files()

    logger.info(f"Files found: {len(file_list)}")
    nlp_documents =[]

    for file_listed in file_list:
        logger.info(f"File: {file_listed}")
        with open(file_listed, 'r') as file:
            raw_text_list = []
            content = file.read()
            raw_text_list.append(content)

            matcher = get_custom_matcher(blank_model)
            for doc in blank_model.pipe(raw_text_list):
                process_document(doc, matcher)
                nlp_documents.append(doc)

    path_to_save_bindocs = env.get_var("DOCBIN_SAVE_PATH")
    random.shuffle(nlp_documents)
    train_docs = nlp_documents[:len(nlp_documents) // 2]
    test_docs = nlp_documents[len(nlp_documents) // 2:]
    logger.info(f"Saving documents to disk")
    # Create and save a collection of training docs
    train_docbin = DocBin(docs=train_docs)
    train_docbin.to_disk(os.path.join(path_to_save_bindocs,"train.spacy"))
    # Create and save a collection of evaluation docs
    test_docbin = DocBin(docs=test_docs)
    test_docbin.to_disk(os.path.join(path_to_save_bindocs,"dev.spacy"))
```

Nota. Hecho por el autor.

Estos datos de verificación son almacenados en disco para el posterior entrenamiento del modelo. Es necesario crear una configuración de entrenamiento especificando que el proceso que vamos a entrenar es para la fase del reconocimiento de entidades y el lenguaje del texto a procesar:

```
python -m spacy init config ./config.cfg --lang es --pipeline ner
```

Evaluación del modelo

Para la parte del entrenamiento se hace uso del siguiente comando al cual hay que pasarle la configuración, una carpeta en donde queremos que se almacene el modelo resultante, los datos de entrenamiento, de evaluación si tenemos una GPU[5] potente se puede utilizar para ahorrar tiempo de entrenamiento, podemos indicarlo con el parámetro gpu-id 0.

```
python -m spacy train ./config.cfg
    --output ./output
    --paths.train C:/Users/xentinel/IdeaProjects/turing-nlp-backend/models/lawtrainningmodel/prod.text
    --paths.dev C:/Users/xentinel/IdeaProjects/turing-nlp-backend/models/lawtrainningmodel/dev.spacy
    --gpu-id 0

```

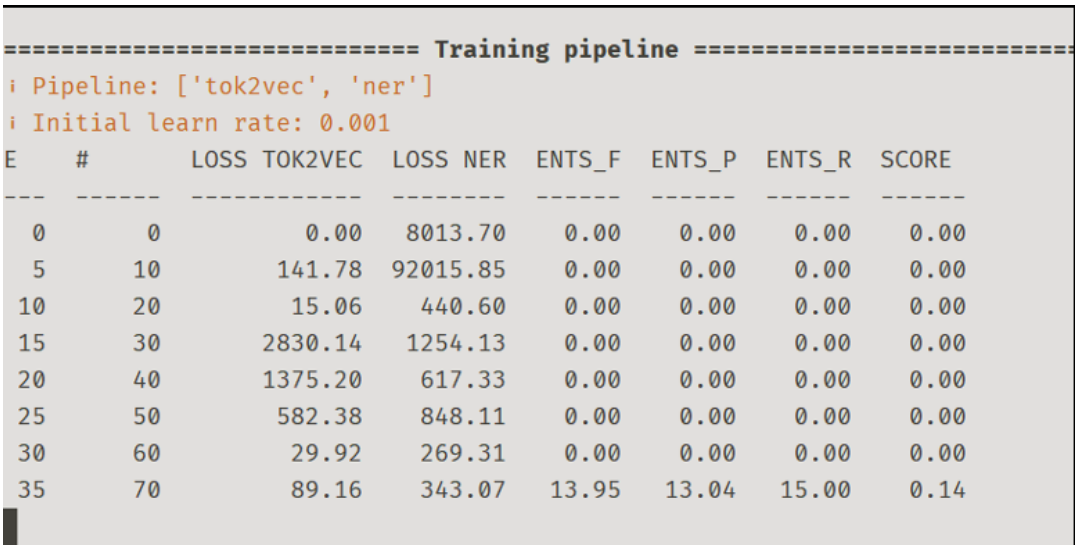
Este comando dará inicio al proceso de entrenamiento del modelo. Al ser una red neuronal, este modelo nuevo debe ser entrenado en lo que se conoce como épocas o iteraciones, cada época el modelo ajustará sus parámetros intentando minimizar la pérdida o el error que está realizando al pronosticar una determinada categoría.

En la ilustración 27 podemos ver el resultado de las primeras 35 épocas de entrenamiento, a destacar la columna LOSS NER (error de reconocimiento de entidad nombrada), que representa el error que está cometiendo el modelo al clasificar palabras como leyes o tesis jurisprudenciales.

[5] Una GPU (Unidad de Procesamiento Gráfico, por sus siglas en inglés Graphics Processing Unit) es un componente de hardware de una computadora diseñado para manejar y acelerar la creación de imágenes, videos y animaciones. Originalmente concebida para tareas gráficas en videojuegos y aplicaciones visuales, su capacidad de procesamiento paralelo ha ampliado su uso a otras áreas.

Ilustración 27

Fase de entrenamiento, primeras 35 épocas.



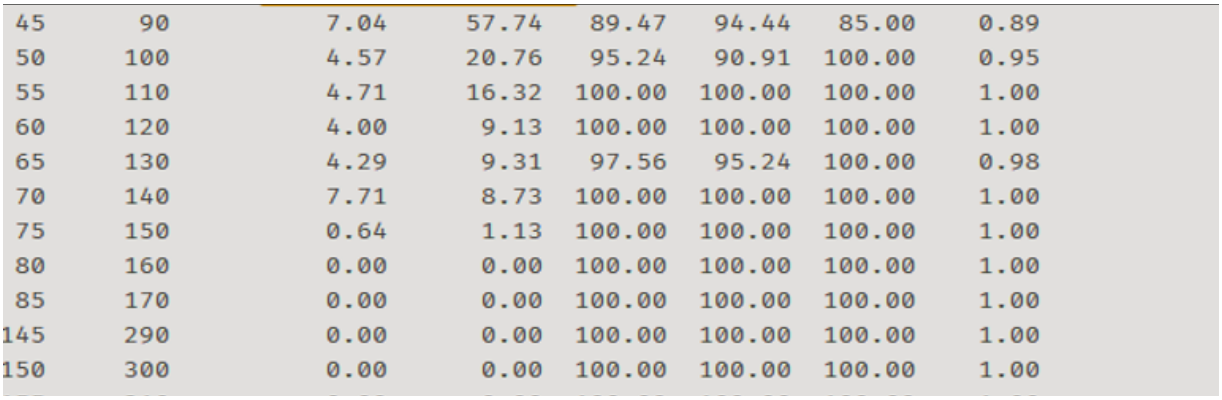
```
============================= Training pipeline =============================
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E    #       LOSS TOK2VEC  LOSS NER  ENTS_F  ENTS_P  ENTS_R  SCORE
---  ------  ------------  --------  ------  ------  ------  ------
  0       0          0.00   8013.70    0.00    0.00    0.00    0.00
  5      10        141.78  92015.85    0.00    0.00    0.00    0.00
 10      20         15.06    440.60    0.00    0.00    0.00    0.00
 15      30       2830.14   1254.13    0.00    0.00    0.00    0.00
 20      40       1375.20    617.33    0.00    0.00    0.00    0.00
 25      50        582.38    848.11    0.00    0.00    0.00    0.00
 30      60         29.92    269.31    0.00    0.00    0.00    0.00
 35      70         89.16    343.07   13.95   13.04   15.00    0.14
```

Nota. Hecho por el autor.

En la ilustración siguiente después un par de horas de entrenamiento, a partir de la época 50 en adelante notamos errores muy bajos de 20 y cayendo hasta llegar a 0.

Ilustración 28

Fase de entrenamiento después de 150 épocas.



```
 45      90          7.04     57.74   89.47   94.44   85.00    0.89
 50     100          4.57     20.76   95.24   90.91  100.00    0.95
 55     110          4.71     16.32  100.00  100.00  100.00    1.00
 60     120          4.00      9.13  100.00  100.00  100.00    1.00
 65     130          4.29      9.31   97.56   95.24  100.00    0.98
 70     140          7.71      8.73  100.00  100.00  100.00    1.00
 75     150          0.64      1.13  100.00  100.00  100.00    1.00
 80     160          0.00      0.00  100.00  100.00  100.00    1.00
 85     170          0.00      0.00  100.00  100.00  100.00    1.00
145     290          0.00      0.00  100.00  100.00  100.00    1.00
150     300          0.00      0.00  100.00  100.00  100.00    1.00
```

Nota. Hecho por el autor.

Métricas de resultados

Para ser capaces de evaluar un modelo de reconocimiento de entidades se pueden aplicar las métricas aplicables a los modelos de clasificación textuales. Las métricas más comunes aplicables a este tipo de modelos son:

- Precisión
- Recuperación o Sensibilidad
- Puntuación de F_1

La **precisión** o **confianza** denota la proporción de casos pronosticados positivamente que son efectivamente casos positivos (Powers, 2020), esto es la proporción de elementos que hemos clasificado ya sea como leyes o como tesis jurisprudenciales y que efectivamente lo son, en los resultados obtuve una precisión de más del 92% lo cual significa que en la gran mayoría de los casos vamos a obtener datos correctamente clasificados ya sea como leyes o como tesis jurisprudenciales, en el 100% de los casos para los datos de entrenamiento y evaluación que se utilizaron se encontró que el modelo de clasificación de tesis jurisprudenciales siempre las clasifica de manera correcta.

La **precisión** se puede calcular mediante la siguiente fórmula:

$$Precision = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Positivos}$$

La **Sensibilidad o recuperación** se define como la razón entre el número total de elementos detectados de forma positiva contra el número total de elementos clasificados como verdaderos positivos + falsos positivos (Buyya et al, 2016):

$$Sensibilidad = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Negativos}$$

F1-Score, o puntuación de F1, es una métrica que combina en un solo valor la precisión y la sensibilidad, esta métrica es útil para poder encontrar un balance entre ambas, se puede calcular con la siguiente fórmula:

$$F1 = 2\,(\frac{Precisión * Sensibilidad}{Precisión + Sensibilidad})$$

El resultado del proceso de entrenamiento arrojó las siguientes métricas:

Tabla 1 *Resultados relevantes para medir el desempeño del modelo entrenado*

Entidad por detectar	Precisión	Sensibilidad	Puntuación de F1
Ley nombrada	0.928571	1.0	0.962962
Tesis Jurisprudencial	1.0	0.6	0.75

Un detalle importante sobre el modelo es que se tiene un valor de sensibilidad de 0.6 para el caso de la detección de tesis jurisprudenciales, este valor es particularmente bajo comparado con la precisión, y se puede interpretar como que, el modelo no está detectando todas las tesis jurisprudenciales y las está dejando pasar sin clasificarlas como tal, para nuestra aplicación será necesario buscar alternativas quizás incrementando el número de datos de entrenamiento para que el modelo no deje sin clasificar elementos en el texto que efectivamente son tesis jurisprudenciales. Es decir, el usuario notaría que, aunque muchas de las tesis jurisprudenciales son detectadas en sus textos normados, no todas ellas serían clasificadas y marcadas dentro de la aplicación.

En cuanto a la puntuación de F1, es una medida que nos ayuda a entender el comportamiento combinando ambas puntuaciones, la precisión y la sensibilidad, para las leyes nombradas la puntuación fue de 0.96, lo cual nos dice que hay un buen balance entre las clasificaciones positivas de leyes y además el modelo deja pasar pocas leyes sin clasificarse.

Para el caso de las tesis jurisprudenciales tenemos un valor menor 0.75, indicando que tenemos un rendimiento menor, sobre todo sabiendo que el modelo deja pasar ciertas tesis sin clasificarse.

Prueba e instalación

Después de la fase de entrenamiento, el proceso genera un modelo de detección de entidades nuevo en el almacenamiento local, y debe ser empaquetado e instalado para su correcta utilización en el código. Estos comandos nos permiten empacar y probar nuestro nuevo modelo tuneado y entrenado específicamente para nuestras necesidades.

```
python -m spacy package ./output/model-best ./model --name ner_tesis --version 0.0.1
pip install es_ner_tesis-0.0.1.tar.gz   jromero, 30/09/24, 11:53 AM • Integrated the l
```

Una vez empacado, es realmente sencilla la utilización de este nuevo modelo, en mi caso lo llame “es_ner_tesis”:

```
        nlp = spacy.load("es_ner_tesis")
        doc = nlp(text)
        tesis = self.__extract_tesis__(doc)
        laws = self.__extract_law__(doc)
        return {"thesis": tesis, "law": laws}
```

La última fase de la prueba es utilizar un documento no visto antes ni en los datos de prueba ni en los datos de verificación. Haciendo uso del visualizador integrado de SpaCy, podemos ver en la ilustración inferior que nuestro modelo entrenado con patrones es capaz ahora de clasificar palabras y frases como TESIS para tesis jurisprudenciales y LEY para leyes.

Ilustración 29

Fase de prueba del modelo, SpaCy NER detecta las Leyes y Tesis nombradas.

Nota. Hecho por el autor.

Análisis de Resultado inicial

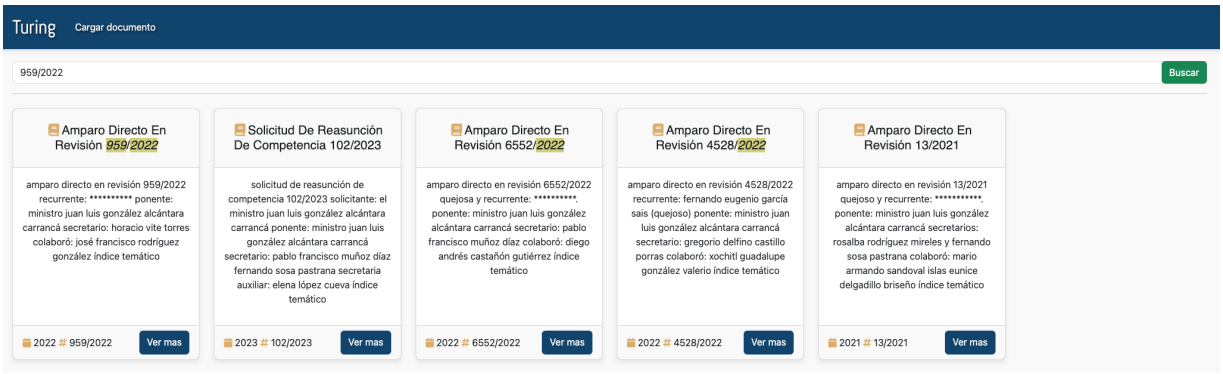
Casos de prueba

Se cargaron 1195 engroses, del año 2022 y 2023, se agregaron las vistas para poder buscar cualquier frase dentro de un documento. Para la vista de la ilustración 30, se

creó un componente en Angular que se encarga de controlar la presentación de las tarjetas de los engroses así como de la sección de búsqueda, en la ilustración 30 coloqué un ejemplo en el cual se busca un engrose en particular el 959/2022, esta búsqueda es dirigida al motor de búsqueda y los resultados que nos regresa son mostrados en pantalla, adicionalmente existe la funcionalidad de subrayar en color amarillo las coincidencias que ha se han encontrado.

Ilustración 30

Búsqueda de engroses por cualquier campo



Nota. Hecho por el autor.

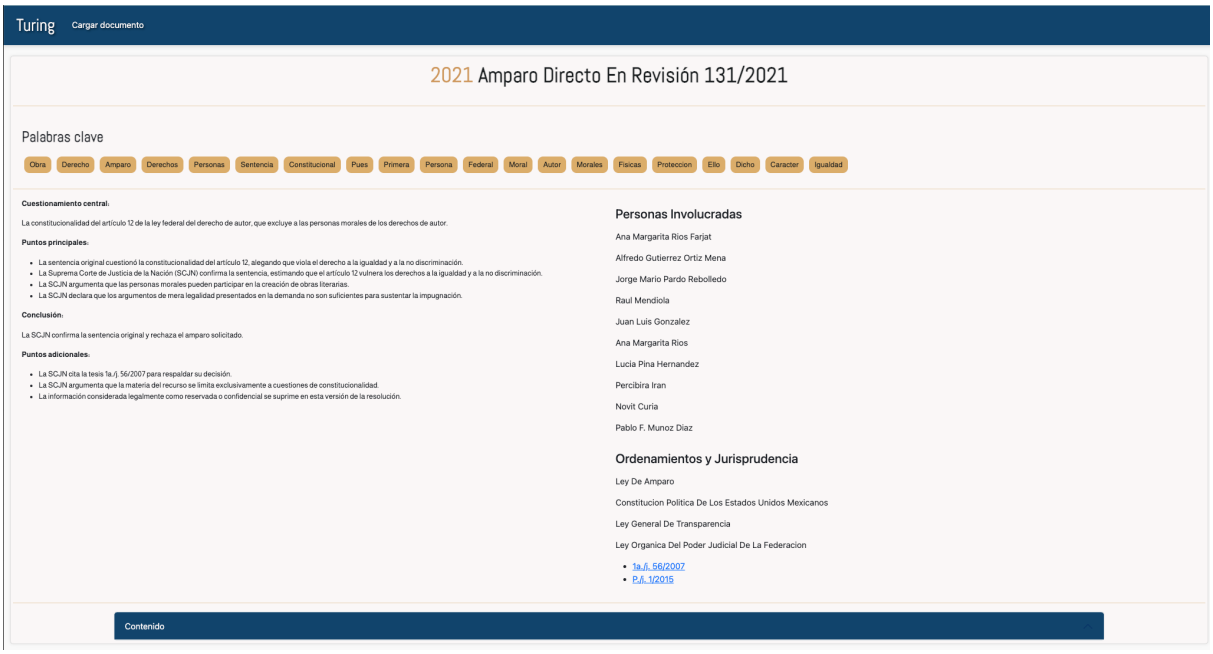
Cuando presionamos el botón de ver más, somos redirigidos a la página de detalle de un engrose en la cual nos es presentado, el nombre del engrose, su año el conjunto de palabras clave que se extrajeron.

Adicionalmente del lado izquierdo podemos ver el resultado del resumen extraído y del lado derecho en dos secciones la lista de personas nombradas que se encontraron en el documento y las tesis jurisprudenciales y los nombres de las leyes que se encontraron nombradas dentro del documento.

En la última sección de esta vista podemos hacer clic en una sección desplegable que nos va a mostrar todo el contenido del engrose tal cual se cargó en la plataforma.

Ilustración 31

Vista de detalle de un engrose



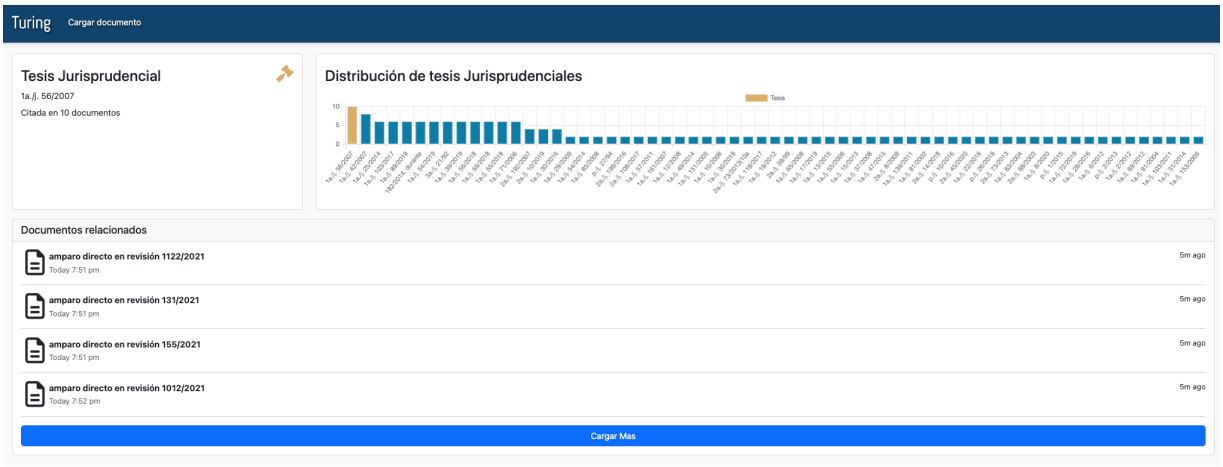
Nota. Hecho por el autor.

Al seleccionar alguna de las tesis jurisprudenciales que se encuentran en la sección inferior derecha de la ilustración 31, la aplicación web nos llevará a una sección en la que nos va a mostrar detalles acerca de la tesis jurisprudencial. Entre ellos cual es el número de engroses en los que la tesis se ha encontrado citada esto nos serviría para saber si es una tesis que es poco citada o, por el contrario, si es una tesis bastante relevante todo esto se presenta de mejor forma en la ilustración 32.

Adicionalmente a este conteo, se muestra un histograma en el cual se presenta la tesis que hemos elegido de un color distinto (dorado) para que podamos comparar su relevancia contra todo el universo de tesis jurisprudenciales existentes en el sistema. Si la tesis que estamos revisando en este momento se encuentra en la parte más alta de la distribución del lado izquierdo, podemos concluir que es una tesis altamente relevante, sin embargo, si se encuentra en la parte derecha de la distribución podemos pensar que es una tesis que se ha citado pocas veces.

Ilustración 32

Detalle de una tesis jurisprudencial



Nota. Hecho por el autor.

En la sección inferior de la vista de detalle de una tesis jurisprudencial, podemos encontrar una lista de engroses en los cuales, la tesis que estamos analizando en el momento se ha encontrado citada, esto para poder buscar otros engroses que pudiesen ser relevantes.

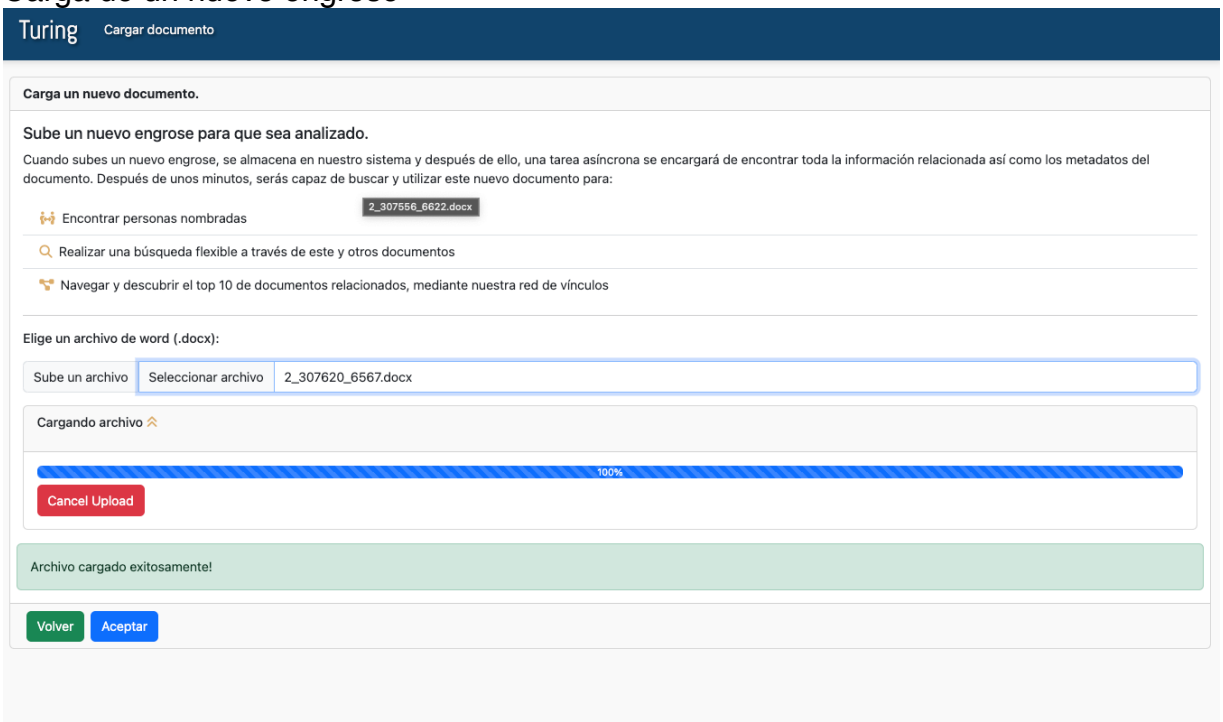
Para la vista en la que podemos cargar un engrose nuevo detallada en la ilustración 34, simplemente es necesario que el usuario se dirija a la sección de subir un archivo, presione el botón y seleccione algún engrose que desee subir a la plataforma. La plataforma lo descartará automáticamente si éste es un documento que ya se encuentra cargado previamente y si no es así, dará comienzo a su procesamiento utilizando el algoritmo previamente mencionado, pre-procesando el documento, limpiándolo y luego realizando el análisis del engrose utilizando los modelos de procesamiento natural del lenguaje.

Es conveniente mencionar que, para las pruebas del modelo, se consultó con un par de abogados que son especialistas en consulta de jurisprudencia, así como secretarios de juzgado, a quien sin darles capacitación de fondo se les explicó de manera breve el sistema, a manera de grupo de control. Ellos facilitaron valorar que el sistema sí era más rápido en las respuestas que los sistemas actuales similares, reiterando que esta

circunscrito a dos años y a la Primera Sala de la SCJN, aunque como modelo puede ser aplicado para más entornos, que a diferencia de sistemas de paga como lo es Elastic Search, el cual ya tiene embebidas funcionalidades de búsqueda, índices y recomendaciones, por lo cual no es necesarios programar IA, sino configurarlo, el cual de hecho es empleado por ejemplo por la misma SCJN, presenta resultados más afinados.

Ilustración 33

Carga de un nuevo engrose



Nota. Hecho por el autor.

Optimización y mejoras

Sin duda toda pieza de software está sujeta a mejoras, uno de los principales puntos que quedan pendientes en esta tesis es el de mejorar la detección de tesis jurisprudenciales atípicas con valores que son distintos y por lo tanto de difícil detección para el modelo, esto se puede mitigar al recopilar más datos etiquetados y de esta forma poder cubrir aún más casos reduciendo la brecha de detección.

Además, la implementación de métricas para poder medir el rendimiento del modelo a lo largo del tiempo y poder identificar problemas como la disminución de precisión. Implementar algún método para poder agregar tesis jurisprudenciales de forma manual y que esto se utilice como retroalimentación para que el modelo pueda ir aprendiendo sobre la marcha a partir de la colaboración de los usuarios.

Documentación

Controladores Spring Boot

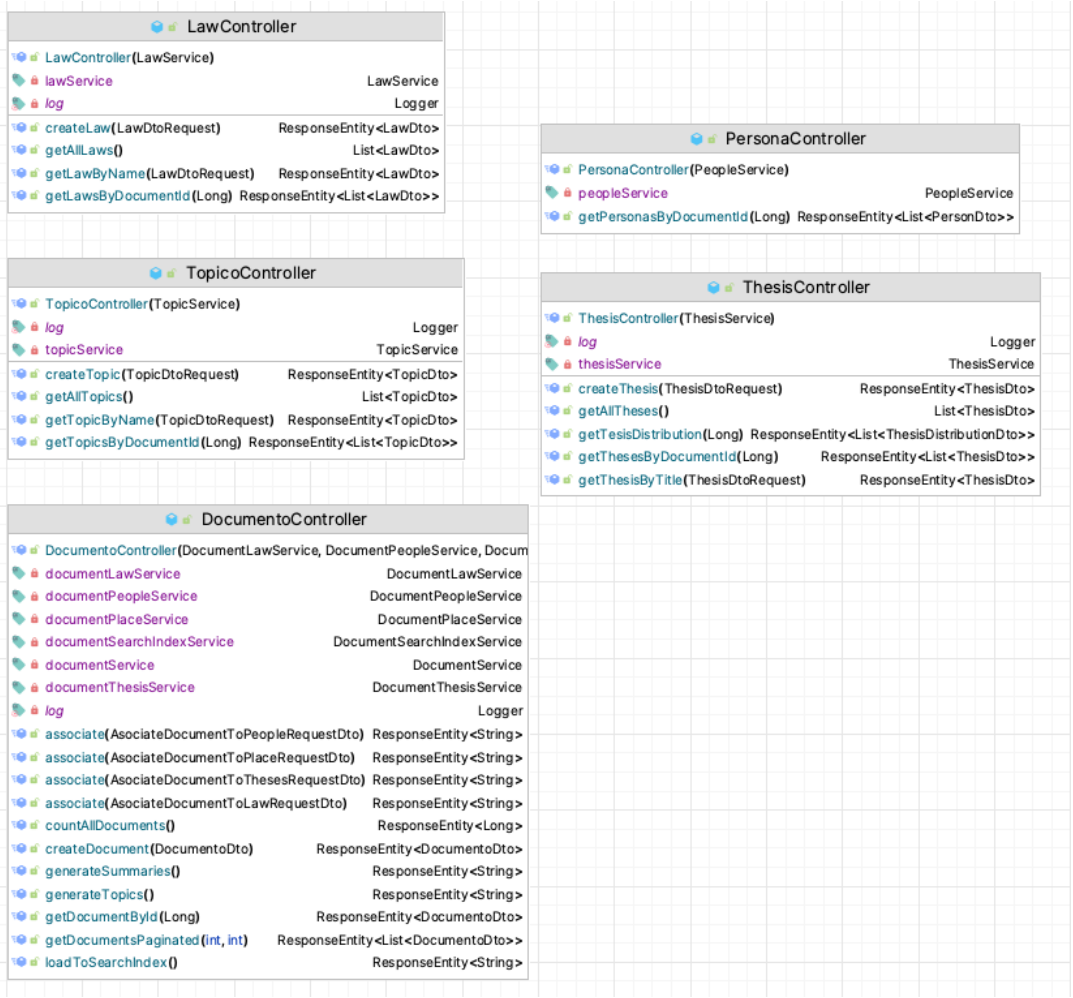
Para el microservicio se crearon cinco controladores principales, el controlador de leyes, el de palabras clave, personas nombradas, tesis jurisprudenciales y el de engroses, estos controladores tienen como única funcionalidad, ser el contacto entre el exterior (el *Gateway*) y la lógica de negocio a través de llamadas Http y los respectivos contratos de información que se debe enviar a cada recurso.

Todos los controladores tienen una URL particular para hacer las peticiones un poco más semánticas, todas ellas comienzan con el prefijo “/api2 y seguidamente se encuentra una palabra que hace referencia al recurso con el que se trabaja, por ejemplo, para el controlador de las palabras clave las direcciones comienzan con “/api/topics”, para la parte de los engroses es “api/documents”.

Todos los controladores hacen uso de la inyección de dependencias por constructor, que es la forma recomendada de Spring por sobre la inyección de dependencias utilizando la cláusula @Autowired. Asimismo, los controladores siguen el principio de diseño de software en el que una capa no debe mandar a llamar o utilizar alguna capa que este más de un nivel inferior o superior a ella. Es decir, en los controladores que se diseñaron seguí el patrón de solo llamar servicios de Spring los cuales encapsulan la lógica de negocio y el acceso y persistencia de datos.

Ilustración 34

Controladores creados para los servicios del lado del servidor en Spring Boot



Nota. Hecho por el autor.

Ilustración 35

Ejemplo del prefijo de las URLs de un controlador

```
    ⌘⇧L to Chat, ⌘I to Command  jromero, 19/10/24, 10:
@Slf4j  jromero
@RestController
@RequestMapping("/api/topics")
@CrossOrigin(origins = "http://localhost:4200")
public class TopicoController {
```

Nota. Hecho por el autor.

Por ejemplo, en el controlador de los engroses tenemos para la ilustración 36, que hacemos uso de la inyección de dependencias por constructor, esto me permite mandar a llamar a todos los servicios que sean necesarios para dar soporte a las peticiones que existen en el controlador.

Ilustración 36

Inyección de dependencias por constructor

```
13  @Slf4j  Julio César Romero +2
14  @RestController
15  @RequestMapping("/api/documents")
16  @CrossOrigin(origins = "*")
17  public class DocumentoController {
18      private final DocumentLawService documentLawService;  2 usages
19      private final DocumentPeopleService documentPeopleService;  2 usages
20      private final DocumentPlaceService documentPlaceService;  2 usages
21      private final DocumentThesisService documentThesisService;  2 usages
22      private final DocumentService documentService;  7 usages
23      private final DocumentSearchIndexService documentSearchIndexService;  2 usages
24
25      public DocumentoController(DocumentLawService documentLawService,  Julio César Romero +1
26                                 DocumentPeopleService documentPeopleService,
27                                 DocumentPlaceService documentPlaceService,   Romero, 28/09/24,
28                                 DocumentService documentService,
29                                 DocumentThesisService documentThesisService,
30                                 DocumentSearchIndexService documentSearchIndexService) {
31          this.documentLawService = documentLawService;
32          this.documentPeopleService = documentPeopleService;
33          this.documentPlaceService = documentPlaceService;
34          this.documentService = documentService;
35          this.documentThesisService = documentThesisService;
36          this.documentSearchIndexService = documentSearchIndexService;
37      }
38
```

Nota. Hecho por el autor.

Además, al ser controladores, las buenas prácticas nos dictan que en ellos debe haber la más mínima o ninguna lógica de negocio. Es por eso que en este caso para servicios como crear un nuevo engrose, obtener la siguiente página de engroses, o contar el número de engroses disponibles en la plataforma, lo único que realiza nuestro controlador es recibir la información para la petición y enrutarla hacia el servicio adecuado, por ejemplo en la ilustración 37, tenemos a partir de la línea 49 la definición de un método, el cual es capaz de crear un nuevo engrose y almacenarlo en el sistema, para ello recibe un objeto en la petición llamado DocumentoDto, este objeto recibido a

través de una petición HTTP utilizando el verbo POST es suficiente para crear el nuevo engrose.

Además, en este caso pueden ocurrir dos escenarios de error, el primero que ocurra un error desconocido, por lo cual le regresaríamos como respuesta a la aplicación web un error 500, el siguiente error seria en el cual intentemos crear un documento que ya existe previamente en el sistema, este caso daría lugar a una excepción llamada DocumentAlreadyExistingException, la cual no es una excepción mala como tal, sino que simplemente se ignoraría y la aplicación continuaría su ejecución normal.

Ilustración 37

Controlador de engroses

```
Codeium: Refactor | Explain | Docstring | ×
@PostMapping  jromero
public ResponseEntity<DocumentoDto> createDocument(@RequestBody DocumentoDto document) {
    try {
        log.info("Creando documento nuevo {}", document.numeroDocumento());
        var createdDocument = documentService.createDocument(document);
        return ResponseEntity.ok(createdDocument);
    } catch (Exception e) {
        log.error("Excepcion desconocida al crear el documento {}", e.getMessage());
        return ResponseEntity.badRequest().body(document);
    } catch (DocumentAlreadyExistingException e) {
        log.error("Error al crear el documento ya existe {}", e.getMessage());
        return ResponseEntity.badRequest().body(document);
    }
}  jromero, 05/10/24, 7:20 PM • Python API now a document is saved using a POST request to spring batch backend.

Codeium: Refactor | Explain | Docstring | ×
@GetMapping("/paginated/{page}/{size}")  jromero +1
public ResponseEntity<List<DocumentoDto>> getDocumentsPaginated(@PathVariable int page, @PathVariable int size) {
    log.info("Get Documentos Página {} size {}", page, size);
    List<DocumentoDto> documents = documentService.getDocumentsPaginated(page, size);
    return ResponseEntity.ok(documents);
}

Codeium: Refactor | Explain | Docstring | ×
@GetMapping("/count")  jromero
public ResponseEntity<Long> countAllDocuments() {
    var total = documentService.countAllDocuments();
    return ResponseEntity.ok(total);
}
```

Nota. Hecho por el autor.

Como se mencionó, la finalidad de los controladores es la de recibir las peticiones del mundo exterior y enrutarlas hacia los servicios, los que se encargan de realizar la lógica de negocio, en la línea 53, 67 y 73 de la ilustración 37 podemos ver que el controlador de engroses simplemente recibe información y la enruta hacia el servicio DocumentService, el cual se encarga de realizar toda la lógica del procesamiento de

un nuevo engrose o de lanzar un error si este documento ya se encuentra cargado en el sistema. Para los demás controladores de las leyes, personas nombradas, palabras clave y tesis jurisprudenciales, se sigue la misma lógica de programación en la cual el controlador recibe la información y se la pasa a un servicio adecuado para su posterior procesamiento.

Servicios

Un servicio de Spring Boot normalmente debe de ser una pieza de código que encapsule una funcionalidad sobre un objeto de dominio. Para el funcionamiento de la aplicación web, se crearon 14 servicios distintos en los cuales se encapsuló la lógica de negocio perteneciente a distintas etapas del proceso:

- **DocumentPeopleService**
    - Encargado de asociar o relacionar un engrose con *n* personas nombradas que se hayan detectado.
- **DocumentLawService**
    - Encargado de asociar o relacionar un engrose con *n* leyes nombradas que se hayan detectado.
- **DocumentTopicService**
    - Encargado de asociar o relacionar un engrose con *n* palabras clave que se hayan detectado.
- **DocumentThesisService**
    - Encargado de asociar o relacionar un engrose con *n* tesis jurisprudenciales nombradas que se hayan detectado.
- **DocumentSearchIndexService**
    - Encargado de cargar engroses en el motor de búsqueda MeiliSearch.
- **DocumentPlaceService**
    - Encargado de asociar o relacionar un engrose con *n* lugares que se hayan detectado.
- **LawService**

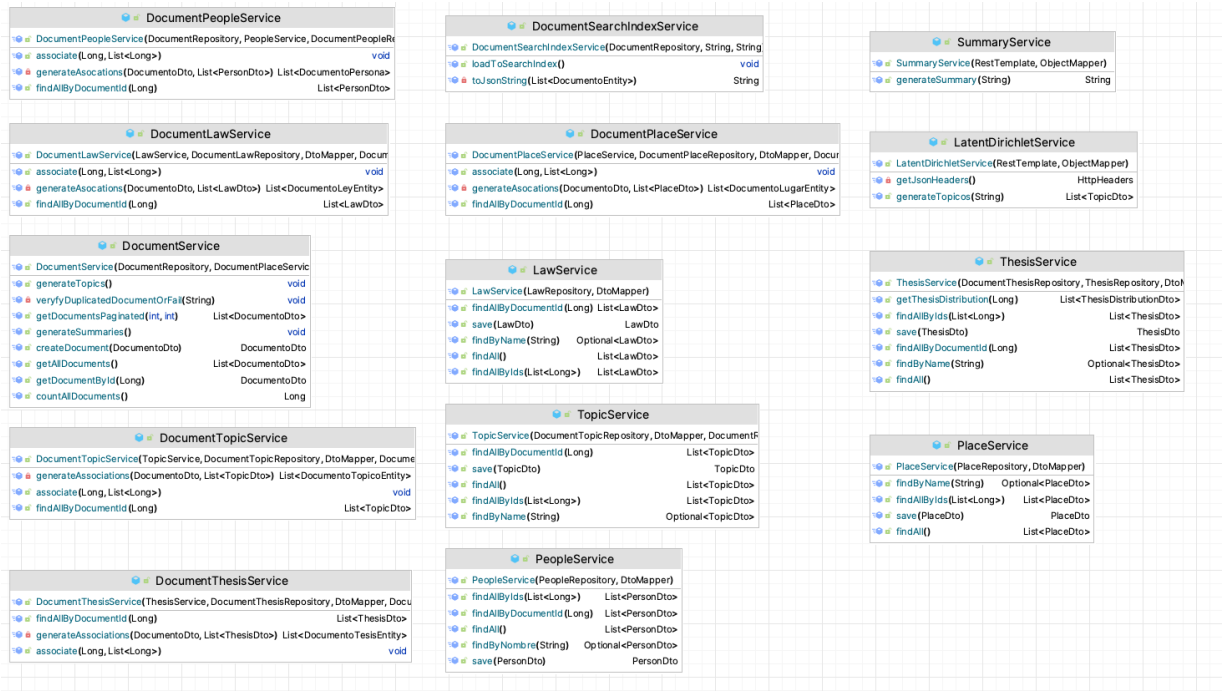
    - Encargado de crear leyes, buscar una ley a partir de su identificador único, buscar leyes por nombre, encontrar todas las leyes relacionadas a un engrose.
- **TopicService**
    - Encargado de crear palabras clave, buscar una palabra clave a partir de su identificador único, buscar palabras clave por nombre, encontrar todas las palabras clave relacionadas a un engrose.
- **PeopleService**
    - Encargado de crear personas, buscar una persona a partir de su identificador único, buscar personas por nombre, encontrar todas las personas relacionadas a un engrose.
- **PlaceService**
    - Encargado de crear lugares, buscar un lugar a partir de su identificador único, buscar lugares a partir de su nombre, encontrar todos los lugares relacionados a un engrose.
- **ThesisService**
    - Encargado de crear tesis jurisprudenciales, buscar una tesis jurisprudencial a partir de su identificador único, buscar tesis jurisprudenciales por nombre, encontrar todas las tesis jurisprudenciales relacionadas a un engrose.
- **SummaryService**
    - Encargado de generar el resumen de un engrose, realiza hasta 5 intentos por cada engrose para generar un resumen.
- **LatentDirichletService**
    - Encargado de generar las palabras clave de un engrose, realiza hasta 5 intentos por cada engrose.

En el caso de los servicios, seguimos el mismo patrón que en los controladores, realizamos la inyección de dependencias a través del constructor, y solo hacemos uso de capas que se encuentren en niveles inferiores a un servicio o en el mismo, es decir,

solo hacemos uso de la inyección de dependencias hacia otros servicios o en su caso, hacia repositorios de Spring Boot, los cuales nos permiten realizar operaciones directamente hacia la base de datos.

Ilustración 38

Servicios creados para el funcionamiento del lado del servidor



Nota. Hecho por el autor.

Además en el caso de los servicios buscamos generar métodos que tengan una sola responsabilidad buscando que sean concisos, cohesivos y que tengan mucha semántica, de forma que sean expresivos y no sea necesaria una documentación excesiva del proyecto, por ejemplo en la ilustración 39, vemos el método para crear un nuevo engrose, a partir de la línea 66 y finalizando en la línea 71, podemos observar que, el método es breve y conciso, vemos que es expresivo ya que lo primero que hace es, verificar si el documento esta duplicado y si es así el método lanza una excepción (línea 68), posteriormente el objeto de transferencia de datos (DTO) se convierte en una entidad (línea 69) y se envía a persistir a la base de datos en la línea 70.

Para el caso de los otros servicios todos están construidos a partir de esta filosofía en la que, se prioriza la semántica y la calidad del código en la que los métodos tienen una sola responsabilidad aplicando los principios S.O.L.I.D.

De acuerdo con el autor Martin, SOLID es un conjunto de buenas prácticas en diseño de software orientado a objetos que busca mejorar la calidad, mantenibilidad y escalabilidad del código; propone cinco conceptos fundamentales: responsabilidad única, apertura para extensión pero cerrado a modificaciones, sustitución coherente de clases derivadas por sus bases, diseño de interfaces específicas para cada funcionalidad, y dependencia invertida hacia abstracciones en lugar de implementaciones concretas (2008). Estos principios promueven un diseño modular y flexible, facilitando el desarrollo y la evolución de sistemas complejos.

Ilustración 39

Servicio para los engroses

```
50
    Codeium: Refactor | Explain | Docstring | ×
51  public DocumentoDto getDocumentById(Long documentId) throws Exception {  1 usage   Julio César Romero *
52      log.info("Obteniendo el documento con el identificador {}", documentId);
53
54      var documento = searchDocumentOrFail(documentId);
55      var documentDto = mapper.documentEntityToDto(documento.get());
56      var places = documentPlaceService.findAllByDocumentId(documentId);
57      var laws = documentLawService.findAllByDocumentId(documentId);
58      var people = documentPeopleService.findAllByDocumentId(documentId);
59
60      return documentDto
61              .withPlaces(places)
62              .withLaws(laws)
63              .withPeople(people);
64  }
65
    Codeium: Refactor | Explain | Docstring | ×
66  @   public DocumentoDto createDocument(DocumentoDto documentDto) throws DocumentAlreadyExistingException {
67      log.info("Creando el documento {}", documentDto.numeroDocumento());
68      veryfyDuplicatedDocumentOrFail(documentDto.numeroDocumento());
69      var documentoEntity = mapper.toDocumentoEntity(documentDto);
70      documentRepository.save(documentoEntity);
71      return mapper.documentEntityToDto(documentoEntity);
72  }
73
74
    Codeium: Refactor | Explain | Docstring | ×
75  public List<DocumentoDto> getDocumentsPaginated(int page, int size) {  1 usage   jromero
76      var documentoEntityList =  documentRepository.findAllByPage(PageRequest.of(page, size));
77      return documentoEntityList.parallelStream().map(mapper::documentEntityToDto).toList();
78  }
79
```

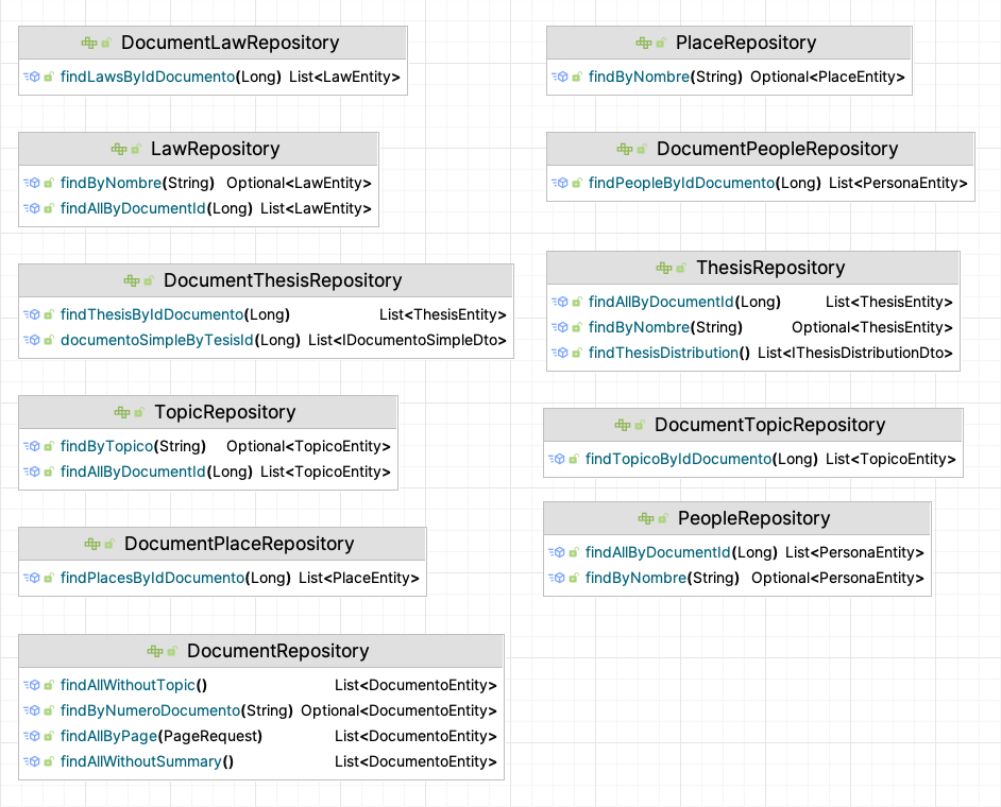
Nota. Hecho por el autor.

Repositorios

Los repositorios son implementaciones de la interfaz CrudRepository en Spring Boot que nos permiten acceder a elementos de la base de datos llamadas Entidades a través del lenguaje JPQL definido en JPA. En el caso de este proyecto, se crearon once repositorios que permiten realizar consultas hacia los objetos de negocio definidos, como persistir, actualizar, insertar y seleccionar, es decir los equivalentes del lenguaje SQL a Insert, Delete, Update y realizar un Select.

Ilustración 40

Repositorios creados para el servicio del lado del servidor



Nota. Hecho por el autor.

Otra ventaja de Spring Boot y de utilizar una interfaz como CrudRepository o JpaRepository es que ya existen muchas consultas implementadas por el equipo de Spring y no es necesario escribirlas, simplemente se pueden utilizar. Por el contrario, si es necesario generar una consulta distinta a las establecidas por la interfaz,

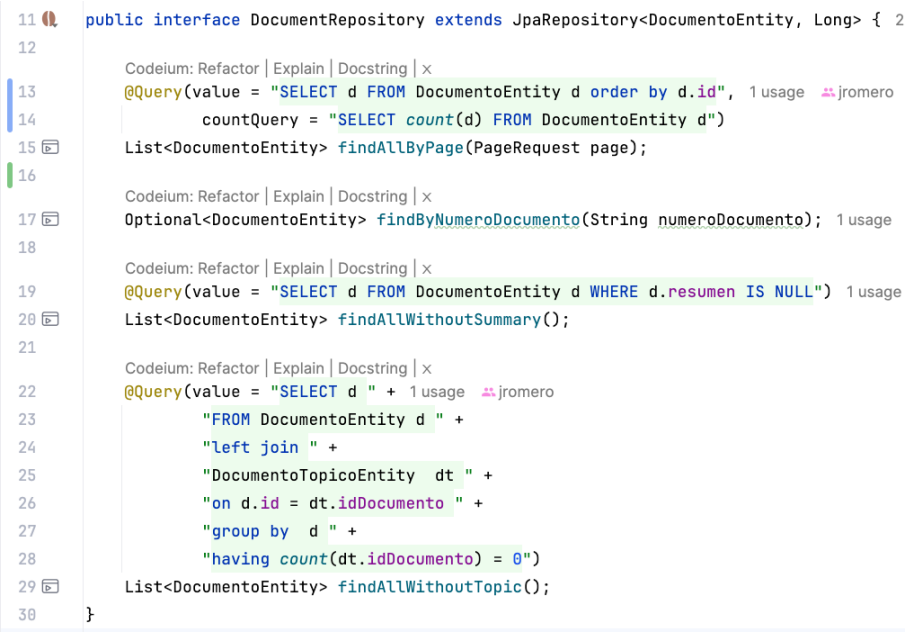
simplemente se puede hacer uso del lenguaje JPQL y escribir una nueva consulta a la medida.

En el caso del repositorio de los engroses, se definieron cuatro consultas personalizadas una un poco sofisticada que es la especificada en la línea 15 de la ilustración 41, esa consulta funciona para darle soporte a la aplicación web en la cual hay una sección para navegar a través de los engroses de forma paginada. Para ello es necesario definir en la consulta de la línea 15, cual es el objeto de negocio que se quiere paginar, en este caso engroses mapeados como DocumentoEntity. Y además agregar una consulta que le permita contar a Spring cuantos objetos de negocio hay en la base de datos.

También se agregaron consultas para poder buscar un engrose a partir de su número de documento. Poder buscar todos los documentos que aún no cuenten con un resumen y poder buscar todos los documentos que aún no tengan la lista de palabras clave definida.

Ilustración 41

Repositorio de Engroses



```
11  public interface DocumentRepository extends JpaRepository<DocumentoEntity, Long> {  2
12
        Codeium: Refactor | Explain | Docstring | ×
13      @Query(value = "SELECT d FROM DocumentoEntity d order by d.id",  1 usage  jromero
14              countQuery = "SELECT count(d) FROM DocumentoEntity d")
15      List<DocumentoEntity> findAllByPage(PageRequest page);
16
        Codeium: Refactor | Explain | Docstring | ×
17      Optional<DocumentoEntity> findByNumeroDocumento(String numeroDocumento);  1 usage
18
        Codeium: Refactor | Explain | Docstring | ×
19      @Query(value = "SELECT d FROM DocumentoEntity d WHERE d.resumen IS NULL")  1 usage
20      List<DocumentoEntity> findAllWithoutSummary();
21
        Codeium: Refactor | Explain | Docstring | ×
22      @Query(value = "SELECT d " +  1 usage  jromero
23              "FROM DocumentoEntity d " +
24              "left join " +
25              "DocumentoTopicoEntity  dt " +
26              "on d.id = dt.idDocumento " +
27              "group by  d " +
28              "having count(dt.idDocumento) = 0")
29      List<DocumentoEntity> findAllWithoutTopic();
30  }
```

Nota. Hecho por el autor.

Procesamiento natural del lenguaje del lado del servidor, Python

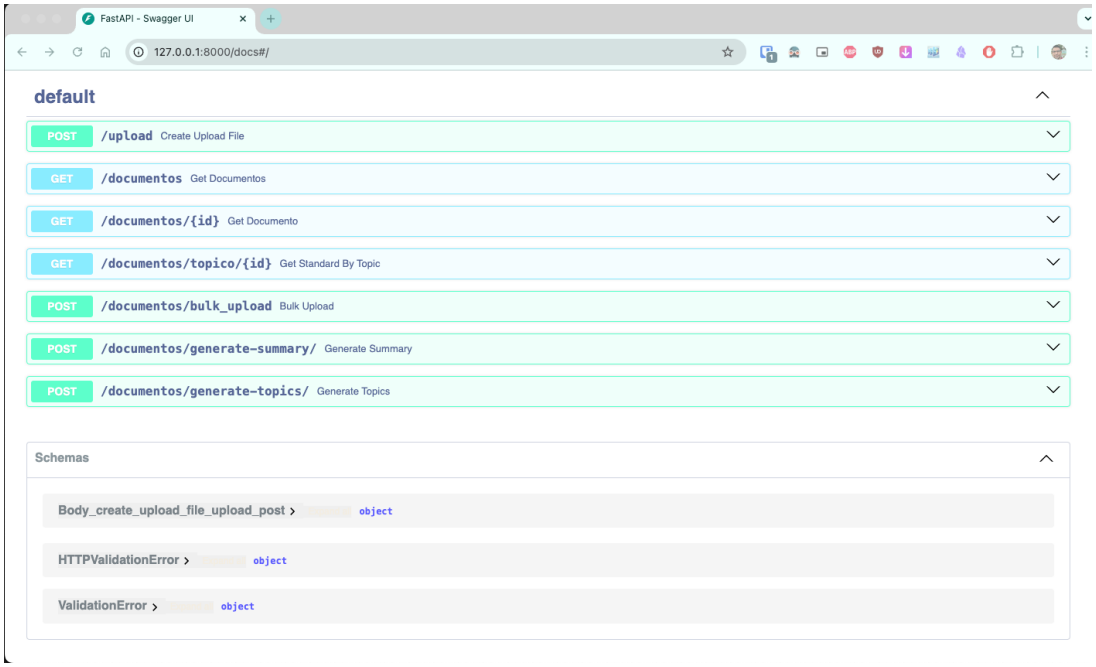
Este servicio del lado del servidor se encuentra programado en el lenguaje de programación Python, utilizando el marco de trabajo FastApi, una de las ventajas que tenemos al realizar esta programación es que ya viene integrado con Swagger que es una forma de documentar APIs o en este caso Microservicios.

Las principales funcionalidades que se encuentran disponibles en este servicio son:

- Cargar un archivo .docx al sistema.
- Obtener todos los engroses cargados.
- Obtener un engrose a partir de su identificador único.
- Obtener las palabras clave de un engrose a partir de su identificador.
- Cargar de manera masiva engroses.
- Generar el resumen de un engrose.
- Generar las palabras claves de un engrose.

Ilustración 42

Documentación autogenerada en FastAPI utilizando Swagger



Nota. Hecho por el autor.

Controladores

Al igual que los controladores realizados en la parte de Spring Boot, en el desarrollo de FastAPI buscamos que los controladores sean encargados de la comunicación vía HTTP con el exterior y no realizar lógica de negocio.

En la implementación del microservicio en Python se incluyeron funcionalidades como la de poder almacenar un texto normado a través de una petición POST subiendo al servidor el documento Word y procesándolo, retornando una respuesta en formato JSON. Un método para poder obtener los engroses cargados en el sistema (ilustración 43, línea 87), y quizás uno de los más importantes, una ruta en el microservicio para la cual le podamos pasar una carpeta que se encuentre en el servidor local y que el API escanee esa carpeta y cargue todo el contenido de esta. Este método nos permite realizar una carga inicial para posteriormente poder utilizar de forma eficiente el método de cargar un engrose a la vez vía la aplicación web.

Cabe mencionar que el propósito principal de la existencia de este servicio es poner a disposición las capacidades de Python y su ecosistema para el análisis de textos, por esta razón Python fue el lenguaje de programación en el que se desarrolló la parte de la extracción de tesis jurisprudenciales y de leyes nombradas. Para ser más preciso en la ilustración 44 se detalla la clase de Python en la que se realiza la ejecución de la extracción de las entidades nombradas.

En la línea 9 tenemos el método principal llamado “extraer entidades”, este método recibe el texto que queremos procesar para realizar la extracción de las tesis y de las leyes nombradas, posteriormente se carga nuestro modelo entrenado específicamente para realizar la tarea de la extracción (línea 12), al modelo le colocamos un nombre para saber que es capaz de clasificar entidades “es_ner_tesis” para dar a entender que es un modelo entrenado para el lenguaje español, “ner” hace referencia a la fase del proceso del procesamiento natural del lenguaje llamada reconocimiento de entidades y es de esta manera que en la línea 13 se realiza la detección de estas entidades.

Ilustración 43

Controlador de engroses en Python

```python
/documentos
@app.get("/documentos") jromero
async def get_documentos():
    document_service = DocumentoService()
    return document_service.get_all()


Codeium: Refactor | Explain | Docstring | ×
/documentos/{id}
@app.get("/documentos/{id}") jromero
async def get_documento(id: int):
    document_service = DocumentoService()
    return document_service.get(id)


Codeium: Refactor | Explain | Docstring | ×
/documentos/topico/{id}
@app.get("/documentos/topico/{id}") jromero
async def get_standard_by_topic(id: int):
    document_service = DocumentoService()
    return document_service.get_topics_by_document_id(id)


Codeium: Refactor | Explain | Docstring | ×
/documentos/bulk_upload
@app.post("/documentos/bulk_upload") Julio César Romero
async def bulk_upload(request:dict = Body(...)):
    bulk_reader = BulkReader(request.get("path"))
    bulk_reader.read()
    return JSONResponse(content={"message": "ok"}, status_code=200)

Codeium: Refactor | Explain | Docstring | ×
/documentos/generate-summary/
@app.post("/documentos/generate-summary/") jromero
async def generate_summary(request:dict = Body(...)):
    text = request.get("text")
    logger.info(f"Generating summary for text with size: {len(text)}")
    document_summarizer = DocumentSummarizerLlama()
    summary = document_summarizer.summarize(text)
    logger.info(f"Summary generated: {len(summary)}")
    return JSONResponse(content={"content": summary}, status_code=200)

Codeium: Refactor | Explain | Docstring | ×
/documentos/generate-topics/
@app.post("/documentos/generate-topics/") jromero
async def generate_topics(request:dict = Body(...)):
    text = request.get("text")
    logger.info(f"Generating topics for text with size: {len(text)}")
    latent_dirichlet_allocation = DocumentLatentDirichletAllocation()
    topics = latent_dirichlet_allocation.transform(text)
    logger.info(f"Topics generated: {topics}")
    return JSONResponse(content={"topic": topics}, status_code=200)
```

Nota. Hecho por el autor.

La parte más costosa del procesamiento se encuentra en la línea 13, una forma de hacerla más eficiente es poner a disposición de la biblioteca SpaCy una librería de Nvidia llamada CUDA, y de esta forma el procesamiento del texto y la clasificación y

detección de entidades nombradas se pueda realizar de forma paralela y aprovechando las capacidades del cómputo en los GPUs.

Ilustración 44

Extracción de tesis jurisprudenciales y de Leyes nombradas

```
class DocumentTesisAndLawExtractor:  2 usages  jromero

    Codeium: Refactor | Explain | Docstring | ×
    def extract_entities(self, text) ->dict[str, set[str]]:  1 usage  jromero
        try:

            nlp = spacy.load("es_ner_tesis")
            doc = nlp(text)
            tesis = self.__extract_tesis__(doc)
            laws = self.__extract_law__(doc)
            return {"thesis": tesis, "law": laws}

        except Exception as e:
            logger.error(f"Error extrayendo las leyes y tesis del documento: {e}")
            return {"thesis": set(), "law": set()}

    Codeium: Refactor | Explain | Docstring | ×
    def __extract_tesis__(self, doc)->set[str]:  jromero
        thesis_set = set()
        for ent in doc.ents:
            if ent.label_ == "TESIS":
                thesis_set.add(ent.text)
        logger.info(f"Se encontraron {len(thesis_set)} diferentes tesis en el documento")
        return thesis_set

    Codeium: Refactor | Explain | Docstring | ×
    def __extract_law__(self, doc) ->set[str]:  jromero
        law_set = set()
        for ent in doc.ents:
            if ent.label_ == "LEY":
                law_set.add(ent.text)
        logger.info(f"Se encontraron {len(law_set)} diferentes leyes nombradas en el documento")
        return law_set   jromero, 30/09/24, 11:53 AM • Integrated the law persister and detection
```

Nota. Hecho por el autor.

Como no todos los equipos tienen estas capacidades lo más compatible es que SpaCY realice el procedimiento utilizando los núcleos del CPU disponibles en el equipo. A pesar de esta limitación el procesamiento es bastante rápido y gracias a la arquitectura de microservicios que se implementó, no es determinante la existencia de GPUs para mejorar el rendimiento de este proceso, ya que se pueden crear más instancias de

este microservicio y realizar el procesamiento de los engroses de forma paralela gracias a las capacidades de la arquitectura basada en eventos con RabittMQ.

En los métodos de las líneas 22 y 30 de la clase de extracción de entidades, se filtran aquellas que coincidan con las palabras clave “TESIS” y “LEY”, concretamente en esta clase se hace uso de la estructura de datos SET, la cual nos permite agregar elementos a ella como si fuera una lista, pero no nos permite tener elementos duplicados. Es por esta razón que al final del procedimiento podemos obtener las leyes y tesis jurisprudenciales nombradas en el texto sin que estas se repitan y posteriormente las persistiremos en la base de datos de la aplicación.

En el caso de las palabras clave de cada texto, se utilizó de la misma forma el lenguaje de programación Python para su extracción, como resultado de esta implementación tenemos la ilustración 45 en la que se describe específicamente el desarrollo de la extracción de las palabras clave. Para comenzar el método principal se llama extraer palabras clave y parecido a la clase anterior es necesario pasarle un texto del cual queremos extraer una lista de palabras clave.

Utilizamos un objeto que permita extraer los tokens del texto y posteriormente para cada token que no sea una palabra no relevante se realiza un conteo (línea 25 de la ilustración 45), por último en la línea 47 extraemos el top de las 20 palabras más repetidas dentro del texto. Este método parte de la suposición estadística de que las palabras que más se repiten dentro de un texto son las que son más relevantes y que por lo tanto podemos obtener una idea general de lo que el texto razona con solo extraer estas palabras.

Ilustración 45

Extracción de las palabras clave

```python
class DocumentLatentDirichletAllocation:  6 usages  jromero +1

    Codeium: Refactor | Explain | Docstring | ×
    def extract_keywords(self, text: str) -> list[str]:  3 usages  jromero +1
        # Initialize regex tokenizer
        tokenizer = RegexpTokenizer(r'\w+')
        spanish_stopwords = self.get_spanish_stopwords()

        # Vectorize document using TF-IDF
        tfidf = TfidfVectorizer(lowercase=True,
                                stop_words=spanish_stopwords,
                                ngram_range=(1, 1),
                                tokenizer=tokenizer.tokenize)

        # Fit and Transform the text
        train_data = tfidf.fit_transform([text])

        # Define the number of topics or components
        num_components = 1
        # Create LDA object
        model = LatentDirichletAllocation(n_components=num_components, random_state=0)
        # Fit and Transform SVD model on data
        lda_matrix = model.fit_transform(train_data)

        logger.info(f"Latent Dirichlet Allocation Model: {lda_matrix}")
        # Get Components
        lda_components = model.components_

        # Print the topics with their terms
        terms = tfidf.get_feature_names_out()
        top_terms_list = []   jromero, 10/08/24, 4:36 PM • implementing the nlp docume
        for index, component in enumerate(lda_components):
            zipped = zip(terms, component)
            top_terms_key = sorted(zipped, key=lambda t: t[1], reverse=True)[:20]
            top_terms_list = list(dict(top_terms_key).keys())
            logger.info(f"Topics {index}: {top_terms_list}")
```

Nota. Hecho por el autor.

Aplicación web

Del lado de la aplicación web se crearon múltiples servicios en Angular, el servicio central de la aplicación es llamado DocumentoService, diseñado para interactuar con un microservicio del lado del servidor. Su propósito principal es realizar operaciones relacionadas con los textos normados, como obtener detalles, listar documentos de forma paginada y contar la cantidad total de documentos disponibles.

Este servicio encapsula toda la lógica relacionada con las solicitudes HTTP para documentos, manteniendo limpio el código de los componentes que lo consumen. Las URLs se configuran dinámicamente a partir del archivo environment, lo que facilita el

despliegue en distintos entornos (desarrollo, producción, etc.). Adicionalmente este servicio encapsula toda la lógica relacionada con las peticiones para los documentos, manteniendo limpio el código de los componentes que lo consumen.

Ilustración 46

Servicio de Engroses en la aplicación web

```
1   import {Injectable} from '@angular/core';
2   import {HttpClient} from "@angular/common/http";
3   import {environment} from "../../environments/environment";
4   import {Observable} from "rxjs";
5   import {Documento} from "./documento";
6
7   @Injectable({ Show usages  jromero +1 *
8     providedIn: 'root'
9   })
10  export class DocumentoService {
11
12    private baseUrl: string = environment.nlpBackendUrl + "/documentos";
13    private springBackend: string = environment.gatewayTuringUrl + "/turing-crud/api/documents";
14
15    constructor(private http: HttpClient) { no usages  jromero
16    }
17
18    getDocumento(id: number): Observable<Documento> { Show usages  jromero
19      return this.http.get<Documento>(`${this.baseUrl}/${id}`);
20    }
21
22    getDocumentos(page: number, size: number): Observable<Documento[]> { Show usages  jromero
23      return this.http.get<Documento[]>(`${this.springBackend}/paginated/${page}/${size}`);
24    }
25
26    countDocuments(): Observable<number> { Show usages  jromero
27      return this.http.get<number>(`${this.springBackend}/count`);
28    }
29  }
30      ⌘⇧L to Chat, ⌘I to Command
```

Nota. Hecho por el autor.

De igual manera se crearon componentes para encapsular partes de la aplicación web, quizás el más importante de ellos el que nos permite presentar la información detallada de los textos normados, para el caso de Angular los componentes son elementos sencillos que se enlazan con la vista, en la ilustración 47 se detalla el componente que gestiona la lógica de la presentación del detalle del texto normado.

Para mostrar el detalle del documento en el método de la línea 39, hacemos uso de observables, esto es llamadas asíncronas hacia servicios del lado del servidor, las cuales esperamos que se resuelvan y en consecuencia actuamos al respecto. La primera llamada asíncrona es para obtener el texto normado desde el lado del servidor

y si esta llamada se resuelve correctamente, entonces se envían cuatro promesas más para traer las palabras clave, las leyes, tesis jurisprudenciales y las personas relacionadas a ese documento.

Ilustración 47

Detalle de un engrose, componente de Angular

```
15  @Component({  Show usages  jromero *
16    selector: 'app-documento-detail',
17    templateUrl: './documento-detail.component.html',
18    styleUrls: ['./documento-detail.component.css']   jromero, 11/08/24, 3:54
19  })
20  export class DocumentoDetailComponent implements OnInit{
21    documento: Documento | undefined;
22    topicos: Topico[] = [];
23    people: Person[] = [];
24    laws: Law[] = [];
25    theses: Thesis[] = [];
26
27    constructor(private route: ActivatedRoute,  no usages  jromero *
28                private documentoService: DocumentoService,
29                private topicoService: TopicoService,
30                private peopleService: PeopleService,
31                private lawService: LawService,
32                private thesisService: ThesisService)
33    { }
34
35    ngOnInit(): void {  no usages  jromero
36      this.getDocumento();
37    }
38
39    private getDocumento():void {  Show usages  jromero
40      const id:number = Number(this.route.snapshot.paramMap.get('id'));
41      this.documentoService.getDocumento(id)
42        .subscribe(documento : Documento  => {
43          this.documento = documento
44          this.getTopicos();
45          this.getPeople();
46          this.getLaws();
47          this.getThesis();
48        });
49    }
```

Nota. Hecho por el autor.

Otra funcionalidad importante es la de poder subir un archivo en formato Word desde el navegador hasta nuestro servidor, para esta tarea se creó un componente en la aplicación de Angular, este componente esta enlazado a la vista en la ilustración 49, específicamente en la línea 36, en la que se define un evento “change” en un input de HTML de tipo file. Dicho evento ejecuta la función onFileSelected descrita en la ilustración 48, línea 23. En ella se toma el archivo que elige el usuario desde la aplicación web en su navegador y se agrega a una forma, esta forma después es enviada al servidor específicamente en el microservicio escrito en Python.

Este microservicio tomara esa información <enviada desde la página web y la almacenara. En esta lógica se agregó una barra de progreso para que el usuario pueda ver cuál es el progreso que lleva su archivo al cargar.

Ilustración 48

Controlador para subir un archivo en Angular

```
export class DocumentUploaderComponent {
  finished: boolean = false;

  constructor(private http: HttpClient, private router: Router) { no usages jromero
  }

  onFileSelected(event: Event) : void { Show usages jromero *
    const target = event.target as HTMLInputElement;
    let file : File | null | undefined = target.files?.item(0);
    if (file) {
      this.fileName = file.name;
      this.loading = true;
      this.finished = false;


      const formData = new FormData();

      formData.append("document", file);
      formData.append("filename", this.fileName);

      const upload$ : Observable<HttpEvent<Object>> = this.http.post(`${environment.nlpBackendUrl}/upload`, formData, {
        reportProgress: true,
        observe: 'events'
      })
        .pipe(
          finalize(() : void => this.reset())
        );

      this.uploadSub = upload$.subscribe(event : HttpEvent<Object> => {
        if (event.type == HttpEventType.UploadProgress) {
          this.progress = Math.round(100 * (event.loaded / event.total!));
          console.log(this.progress);
        }
      });

    }
    this.finished = true;   jromero, 31/07/24, 3:02 PM • Frontend app for uploading a document
    console.log("Load finished", this.finished)
  }

  cancelUpload() : void { Show usages jromero
    if (this.uploadSub) {
      this.uploadSub.unsubscribe();
      this.reset();
    }
  }
```

Nota. Hecho por el autor.

Ilustración 49

Lógica en la vista para subir un archivo

```
<p class="fw-medium">Elige un archivo de word (.docx):</p>

<div class="input-group mb-3">
  <label class="input-group-text" for="inputGroupFile01">Sube un archivo</label>
  <input #fileUpload (change)="onFileSelected($event)" type="file" class="form-control" id="inputGroupFile01"
         accept=".docx">
</div>

<div class="card" *ngIf="loading">
  <div class="card-header">
    <p class="fw-medium">Cargando archivo <i class="fa-solid fa-angles-up"></i></p>
  </div>
  <div class="card-body">   jromero, 31/07/24, 3:02 PM • Frontend app for uploading a document
    <div class="progress">
      <div class="progress-bar progress-bar-striped progress-bar-animated"
           role="progressbar" [style.width.%]="progress" [attr.aria-valuenow]="progress"
           aria-valuemin="0" aria-valuemax="100">
        {{ progress }}%
      </div>
    </div>
    <button class="btn btn-danger mt-2" (click)="cancelUpload()">Cancel Upload</button>
```

Nota. Hecho por el autor.

Conclusiones

Partiendo de la hipótesis de trabajo propuesta para la presente tesis, se puede afirmar que "La aplicación de técnicas de análisis de texto y aprendizaje automático en la interpretación y clasificación de textos normados resultará en una mejora en la eficiencia y precisión del procesamiento de dichos textos en comparación con los métodos tradicionales basados en revisiones manuales", sí se cumple.

Sustentando lo anterior en los siguientes elementos:

Resultados cuantitativos: Se utilizó un conjunto de más de 1,000 textos normados procesados mediante técnicas de aprendizaje automático y modelos como SpaCy y grandes modelos de lenguaje (LLMs). El sistema automatizado redujo el tiempo necesario para procesar y analizar textos normativos en comparación con los métodos manuales, además de demostrar un alto grado de precisión en la clasificación y detección de entidades como leyes y tesis jurisprudenciales.

Es conveniente mencionar que una tesis de licenciatura en ingeniería en computación tiene como principal objetivo demostrar la capacidad del estudiante, en este caso el autor de la presente, para integrar y aplicar los conocimientos adquiridos durante la carrera a través de la resolución de un problema técnico específico, utilizando metodologías y herramientas ya establecidas en el campo, y su alcance está orientado hacia el diseño, implementación o mejora de sistemas computacionales, es decir, procurar mostrar la comprensión y aplicación práctica de conceptos fundamentales, mostrando habilidades de análisis, programación y diseño en un contexto como lo son los textos normados.

En contraste, una tesis de doctorado en este ámbito tiene un alcance mucho mayor, ya que busca generar conocimiento original y contribuir de manera sustantiva al avance de la disciplina, incluso esto lleva a la identificación de lagunas en el conocimiento existente, el planteamiento de hipótesis innovadoras y la validación de estas mediante metodologías rigurosas. Además, debe demostrar un impacto

potencial en la teoría, la práctica o ambas, proponiendo soluciones que trascienden las limitaciones técnicas actuales. Este documento cumple con el objetivo de una tesis de licenciatura y tiene algunos alcances de posgrado, al identificar áreas de oportunidad en la teoría vigente de textos normados en el contexto mexicano.

La implementación de técnicas de procesamiento de lenguaje natural y algoritmos supervisados optimizó tareas previamente complejas, como la identificación de entidades legales, generación de resúmenes y clasificación de documentos; así como el uso de grandes modelos de lenguaje permitió manejar la variabilidad y complejidad inherentes al lenguaje jurídico encontrado en los engroses.

El trabajo tiene un impacto práctico, la plataforma desarrollada facilita la interpretación de textos normados, así como la creación de redes de citación, mejorando la navegación y comprensión de documentos extensos. Se comprobó que el sistema puede integrarse fácilmente en entornos legales, contribuyendo a la eficiencia de abogados y juzgadores.

Los resultados experimentales incluyeron métricas como precisión, recall y la puntuación F1, que fueron superiores en los métodos automáticos frente a los manuales. Se implementaron iteraciones en el diseño del sistema, siguiendo principios de ingeniería de software, para optimizar su desempeño; los experimentos realizados y los resultados presentados en la tesis validan la hipótesis planteada.

Por otra parte, considerando que un objetivo de esta tesis fue la programación de un sistema de análisis de texto y autoaprendizaje para automatizar el análisis e interpretación de textos normados, mismo que se cumple con la generación del modelo de reconocimiento de entidades que nos permite identificar las tesis jurisprudenciales y las leyes citadas en los 1,195 textos normados que se tienen registrados del año 2022 al 2023, obtenidos del sitio de la Suprema Corte de Justicia de la Nación.

Además, se automatizó la interpretación de los textos normados usando el modelo de código abierto Llama 3.2, además se tiene una implementación con el modelo privativo comercial gpt-4o y se presentaron los resultados en una aplicación web de forma

concisa haciendo más sencillo el procesamiento de los engroses comparado con el método tradicional de revisión manual. Se agregó la capacidad de poder buscar de forma eficiente y rápida información dentro de cualquier parte de los textos normados cargados en la aplicación.

Entre las limitaciones en el desarrollo de la presente, está el acceso a grandes modelos de lenguaje del estado del arte que sean capaces de comprender documentos de tan larga longitud como lo son los textos normados de forma económica, esto debido a que su costo es elevado, y la alternativa de utilizar modelos de código abierto requiere una inversión en infraestructura para su ejecución. Adicionalmente en el caso del entrenamiento de un modelo personalizado para la detección de las tesis jurisprudenciales y de las leyes nombradas es necesario desarrollar un método para poder generar datos de entrenamiento con una mayor diversidad ya que con los generados previamente el modelo es capaz de detectar tesis jurisprudenciales y leyes nombradas, pero omite algunas que tienen una estructura o una sintaxis distinta.

Podría ser que utilizando algún otro tipo de modelo o utilizando múltiples modelos se pueda minimizar el número de tesis y leyes que no son detectadas de forma automática por el modelo de aprendizaje.

Esfuerzos como el realizado por el Grupo de Ingeniería de Lingüística (GIL) donde los alumnos de la Facultad podemos obtener artículos de Corpus Lingüísticos, amplió el interés de analizar otra parte de una minería de datos vía IA, teniendo la oportunidad de presentar una ponencia en el Seminario de Lingüística Forense (SeLiFo) en su VIII edición, que fue donde visualice como alumno la posibilidad de realizar esta tesis, además, en la Facultad de Derecho, en el Instituto de Investigaciones Jurídicas tienen un avance doctrinal importantísimo, sin entrar al fondo de ingeniería aún, no obstante se reconoce la labor que en materia de IA están desarrollando, entre otros la Mtra. Guadalupe Juárez de la Facultad de Derecho, el Dr. César Cáceres Nieto, así como mi tutor de la presente.

En esta tesis aplica lo aprendido, principalmente en la bases de la materia de ingeniería de software que facilitó comprender como emplear la IA y el procesamiento natural del lenguaje en el ámbito jurídico mexicano; si bien algebra lineal brinda fundamentos, el aplicarlo con programación en un proyecto brindó mayor certeza. Se sugiere que se continúen los esfuerzos para poder generar más elementos de investigación desde la academia para su aplicación en los órganos jurisdiccionales, facilitando el acceso a la información jurídica, haciendo con ello más sencilla su interpretación y búsqueda, para ministros, magistrados, jueces, abogados y servidores públicos, para personas que carecen de conocimiento especializado en la materia, incluso que la Facultad de Ingeniería tuviera mayor alcance para colaborar por ejemplo con la Facultad de Derecho.

Esta tesis de ingeniería tiene a su nivel de licenciatura, una contribución al campo de estudio técnico de la IA, con un sustento teórico aplicado en lo jurídico, que tiene potencial de ser implementado en cortes nacionales o internacionales. Que incluso siguiendo el buen ejemplo y desempeño del GIL, brinda bases para proseguir con estudios de posgrado a partir de este trabajo.

Fuentes

Agile 42. (2023, July 31). *Scrum in a nutshell*. https://www.agile42.com/en/agile-community/agile-info-center/scrum-nutshell

Arroyo-Fernández, I., Méndez-Cruz, C.-F., Sierra, G., Torres-Moreno, J.-M., Sidorov, G. (2019). Unsupervised sentence representations as word information series: Revisiting TF–IDF. *Computer Speech & Language*, *56*, 107–129. https://doi.org/10.1016/j.csl.2019.01.005

Ashley, K. (2019). *Automatically Extracting Meaning from Legal Texts: Opportunities and Challenges*. Ssrn.com. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4083158

Atkinson, J. M., Drew, P. (1979). *Order in Court: The Organization of Verbal Behavior in Judicial Settings.* Humanities Press.

Austin, A. (1993). The Reliability of Citation Counts in Judgments on Promotion, Tenure and Status. *Arizona Law Review* 35, 829– 40.

Bernal Pulido, C., Camarena , R., y Martínez Verástegui, A. (2018). *El precedente en la Suprema Corte de Justicia de la Nación*. Centro de Estudios Constitucionales SCJN.

Bishop, C. M., y Bishop, H. (2023). *Deep Learning*. Springer Nature.

Buyya, R., Calheiros, R. N., Dastjerdi, A. V. (2016). *Big data : principles and paradigms*. Elsevier/Morgan Kaufmann.

Chang, X. (2023). The Analysis of Open Source Search Engines. *Highlights in Science Engineering and Technology*, *32*, 32–42. https://doi.org/10.54097/hset.v32i.4933

Chukwurah, E. G., Aderemi, S. (2024). Elevating team performance with scrum: insights from successful US technology companies. *Engineering Science & Technology Journal*, *5*(4), 1357–1371. https://doi.org/10.51594/estj.v5i4.1038

Cox, M., Ellsworth, D. (1997). Application-controlled demand paging for out-of-core visualization. *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, 1–3. https://doi.org/10.1109/visual.1997.663888

D3.js. (2024). D3 Force-directed Graph [Imágen en línea]. In *Force-directed graph, canvas*. https://observablehq.com/@d3/force-directed-graph-canvas/2?collection=@d3/d3-force

Dean, J., Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113. https://doi.org/10.1145/1327452.1327492

Devlin, J., Chang, M.W. (2018). *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing*. https://research.google/blog/open-sourcing-bert-state-of-the-art-pre-training-for-natural-language-processing/

Escriche, J. (1881). *Diccionario razonado de legislacion y jurisprudencia*. Imprenta Eduardo Cuesta. https://www.rae.es/sites/default/files/biblioteca/pdf/4_A_56/4_A_56.pdf

explosion/spaCy Industrial-strength PLN. (2021, March 23). GitHub. https://github.com/explosion/spaCy

Farley, D. (2021). *Modern Software Engineering*. Addison-Wesley Professional.

Ford, N., Richards, M., Sadalage, P. J., Dehghani, Z. (2022). *Software architecture : the hard parts : modern trade-off analysis for distributed architectures*. O'Reilly.

Gao, R., Li, D., Li, W., Dong, Y. (2012). Application of Full Text Search Engine Based on Lucene. *Advances in Internet of Things*, *2*(4), 106–109. https://doi.org/10.4236/ait.2012.24013

Garfield, E. (1979). Is citation analysis a legitimate evaluation tool? *Scientometrics*, *1*(4), 359–375. https://doi.org/10.1007/bf02019306

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. The MIT Press. https://www.deeplearningbook.org/

Gupta, V., Singh Lehal, G. (2010). A survey of Text Summarization Extractive Techniques, *Journal of Emerging Technologies in Web Intelligence*, 2 (3),258-268.

Gutiérrez Gatica, M. de los Á. (2024). *TESIS JURISPRUDENCIAL 1/2020 (10a.)*. 1–20. https://www.scjn.gob.mx/sites/default/files/tesis/documento/2020-12/TESIS%20JURISPRUDENCIALES%202020_PRIMERA%20SALA.pdf

Harris, Z. S. (1954). Distributional Structure, *WORD, 10*(2-3), 146-162.

Heckler, M. (2021). *Spring Boot: Up and running*. O'Reilly Media.

Hernández Sampieri, R., Fernández Collado, C., Baptista Lucio, M. (2014). *Metodología de la Investigación*. McGraw Hill.

Hirschberg, J., Manning, C. D. (2015). Advances in natural language processing. *Science*, *349*(6245), 261–266. https://doi.org/10.1126/science.aaa8685

Honnibal, M. (2017, November 12). *spaCy's NER model · spaCy Universe*. SpaCy's NER Model. https://spacy.io/universe/project/video-spacys-ner-model

John, T., Misra, P., Benjamin, T. (2017). *Data Lake for Enterprises*. Packt Publishing.

Juárez Quezada, G. (2023). El procedimiento y el Acto Administrativo en México. En R. Contreras Bustamante (Ed.), *Derecho Procesal Administrativo Mexicano*. Tirant lo blanch.

Jurafsky, D., Martin, J. H. (2021). *Speech and Language Processing : an Introduction to Natural Language processing, Computational linguistics, and Speech Recognition* (3a ed.). Dorling Kindersley Pvt, Ltd. https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

Kelsen, H., Vernengo, R. J. (1982). *Teoría pura del derecho* (2da reimpresión). UNAM.

Khan, M. Z. (2019). *Angular projects : build nine real-world applications from scratch using Angular 8 and TypeScript*. Packt Publishing.

Lara Chagoyán, R. (2020). *El Constitucionalismo mexicano en transformación. Avances y retrocesos. Colección IECEQ*. Instituto de Estudios Constitucionales del Estado de Querétaro.

Lewis, J., Fowler, M. (2014, March 25). *Microservices*. Martinfowler.com. https://martinfowler.com/articles/microservices.html

Luan, X. (2021). *IMPLEMENTATION AND ANALYSIS OF SOFTWARE DEVELOPMENT IN SPRING BOOT*. https://scholarworks.calstate.edu/downloads/zg64ts132

Lubanovic, B. (2024). *FastAPI*. O'Reilly Media, Inc.

Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Pearson Education.

Martínez Verástegui, A. (2023). *Teoría y práctica del precedente judicial en Iberoamérica* (2da ed.). Tirant lo blanch.

Meric, A. (2024). *Mastering Spring Boot 3.0*. Packt Publishing Ltd.

Migl, A. (2023). Premium Platform Electric IAA 2023 [Imagen en línea]. In *Wikimedia*.

https://commons.wikimedia.org/wiki/File:Premium_Platform_Electric_IAA_2023_1X7A0298.jpg

Moral Soriano, L. (2000). A progressive foundation of precedents. *Archiv für Rechts-und Sozial-philosophie, 86,* (3), 327-350.

Muñoz C. (1998). *Cómo elaborar y asesorar una investigación de tesis*. Prentice Hall Hispanoamericana.

Nonaka, I. (1994). A Dynamic Theory of Organizational Knowledge Creation. *Organization Science*, *5*(1), 14–37.

Nonaka, I., Takeuchi, I. (1999). *The knowledge-creating company* (1er ed.). Harvard Business Press.

Ohlhorst, F. J. (2012). *Big Data Analytics* (1st ed.). John Wiley & Sons.

Pimentel Alarcón, A. E. (2022). *Detección de oposición semántica mediante patrones sintácticos yuxtapuestos* [Tesis Doctorado]. Universidad Nacional Autónoma de México.

Puig, F. M. (2024). *Spring Boot 3.0 Cookbook*. Packt Publishing Ltd.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*. https://insightcivic.s3.us-east-1.amazonaws.com/language-models.pdf

Richards, M. (2015). *Software Architecture Patterns. Understanding common architecture patterns and when to use them* (2nd ed.). O'Reilly Media, Inc.

Richards, M. A., Ford, N. (2020). *Fundamentals of software architecture : An engineering approach*. O'Relly.

Said, A., González Gutiérrez, I. M. (2017). *Teoría General del proceso*. IURE Editores.

Sierra, G. (2009). Extracción de contextos definitorios en textos de especialidad a partir del reconocimiento de patrones lingüísticos, *Inguamática 1* (2), 13–37.

Sommerville, I. (2016). *Software engineering* (10a ed.). Pearson Education Limited.

spaCy. (2015). *spaCy · Industrial-strength Natural Language Processing in Python*. SpaCy. https://spacy.io/

Stack Overflow. (2024). *Technology | 2024 Stack Overflow Developer Survey*. Stackoverflow.co. https://survey.stackoverflow.co/2024/technology

Suprema Corte de Justicia de la Nación [SCJN]. (2021, April 8). *Acuerdo General Número 1/2021, de ocho de abril de dos mil veintiuno, del Pleno de la Suprema Corte de Justicia de la Nación, por el que se determina el inicio de la Undécima Época del Semanario Judicial de la Federación*. https://bit.ly/3uPXgIi

Suprema Corte de Justicia de la Nación [SCJN]. (s.f.). *¿Qué hace la Suprema Corte de Justicia de la Nación?* https://www.scjn.gob.mx/conoce-la-corte/que-hace-la-scjn. (s.f.b).

Suprema Corte de Justicia de la Nación [SCJN]. (s.f.). *Formulario de consulta de los asuntos de la SCJN.* https://www2.scjn.gob.mx/consultatematica/paginaspub/tematicapub.aspx. (s.f.).

Tiersma, P. (1993). Linguistic Issues in the Law. *Language, 69*(1), 113-137. doi:10.2307/416418

Trejo Medina, D. (2019a). *Administración del Conocimiento aplicado en el dominio jurídico, caso de precedentes.* http://dx.doi.org/10.13140/RG.2.2.22303.51361

Trejo Medina, D. (2019b). *Big data, una oportunidad de mejora en las organizaciones*. DIDAC.

Trejo Medina, D. (2019c). *Ciencia de datos, la ironía de la teoría.* DIDAC.

Trejo Medina, D. (2020). *Gobierno de datos para directores*. DIDAC.

Trejo Medina, D. (2024a). Justicia algorítmica: desafíos y falacias de la inteligencia artificial en la impartición de justicia. *Poder Judicial Del Estado de México, Escuela Judicial Del Estado de México*, *21*(21), 43–78. https://exlegibus.pjedomex.gob.mx/index.php/exlegibus/article/view/472

Trejo Medina, D. (2024b). Las redes de citación y su relevancia para la administración del conocimiento de los precedentes judiciales en México. *Revista Oficial Del Poder Judicial*, *16*(22), 53–77. https://doi.org/10.35292/ropj.v16i22.998

Trejo, D., Juárez Quezada, G., González Jiménez, C. (2024c). *Inteligencia Artificial. Fundamentos para abogados y estudiantes de Derecho.*

Vasiliev, Y. (2020). *Natural Language Processing Using Python*. O'Reilly Media.

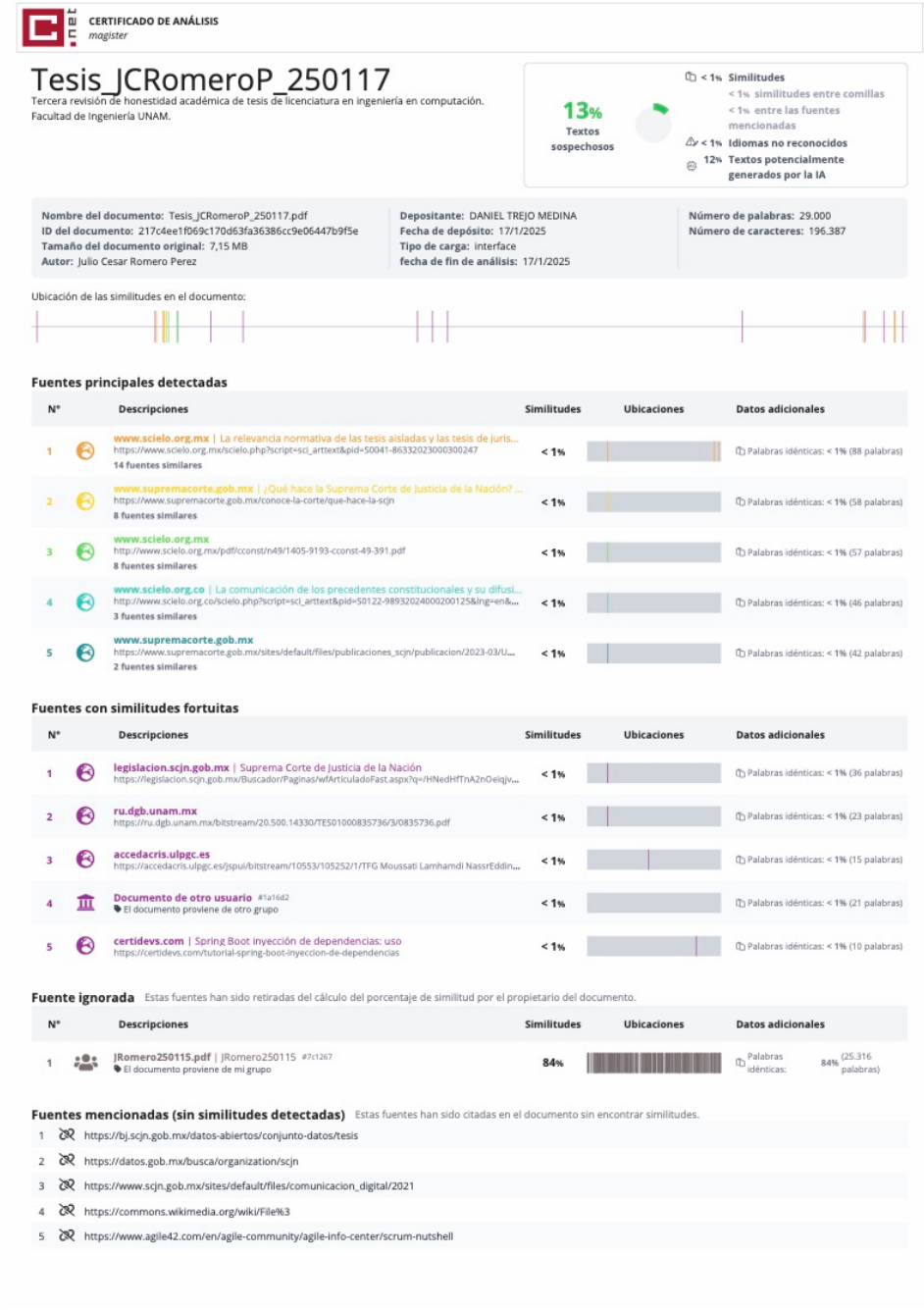
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017, June 12). *Attention Is All You Need*. ArXiv. https://arxiv.org/abs/1706.03762

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., Fedus, W. (2022). Emergent Abilities of Large Language Models. *ArXiv:2206.07682 [Cs]*. https://arxiv.org/abs/2206.07682

Winters, T., Manshreck , T., Wright, H. (2020). *Software Engineering at Google: Lessons Learned from Programming over Time.* O'Reilly & Associates Inc.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P. (2023). A Survey of Large Language Models. *ArXiv:2303.18223 [Cs]*. https://arxiv.org/abs/2303.18223

Resultado del análisis de honestidad académica con Compilatio una vez aprobada por los revisores y sinodales de la tesis. Se ignoró la fuente que subió otro revisor previo a la presente entrega final.



CERTIFICADO DE ANÁLISIS
magister

Tesis_JCRomeroP_250117

Tercera revisión de honestidad académica de tesis de licenciatura en ingeniería en computación.
Facultad de Ingeniería UNAM.

13% Textos sospechosos

- < 1% Similitudes
  - < 1% similitudes entre comillas
  - < 1% entre las fuentes mencionadas
- < 1% Idiomas no reconocidos
- 12% Textos potencialmente generados por la IA

Nombre del documento: Tesis_JCRomeroP_250117.pdf
ID del documento: 217c4ee1f069c170d63fa36386cc9e06447b9f5e
Tamaño del documento original: 7,15 MB
Autor: Julio Cesar Romero Perez

Depositante: DANIEL TREJO MEDINA
Fecha de depósito: 17/1/2025
Tipo de carga: interface
fecha de fin de análisis: 17/1/2025

Número de palabras: 29.000
Número de caracteres: 196.387

Ubicación de las similitudes en el documento:

Fuentes principales detectadas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	www.scielo.org.mx | La relevancia normativa de las tesis aisladas y las tesis de juris... https://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S0041-86332023000300247 14 fuentes similares	< 1%		Palabras idénticas: < 1% (88 palabras)
2	www.supremacorte.gob.mx | ¿Qué hace la Suprema Corte de Justicia de la Nación? ... https://www.supremacorte.gob.mx/conoce-la-corte/que-hace-la-scjn 8 fuentes similares	< 1%		Palabras idénticas: < 1% (58 palabras)
3	www.scielo.org.mx http://www.scielo.org.mx/pdf/cconst/n49/1405-9193-cconst-49-391.pdf 8 fuentes similares	< 1%		Palabras idénticas: < 1% (57 palabras)
4	www.scielo.org.co | La comunicación de los precedentes constitucionales y su difusi... http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0122-98932024000200125&lng=en&... 3 fuentes similares	< 1%		Palabras idénticas: < 1% (46 palabras)
5	www.supremacorte.gob.mx https://www.supremacorte.gob.mx/sites/default/files/publicaciones_scjn/publicacion/2023-03/U... 2 fuentes similares	< 1%		Palabras idénticas: < 1% (42 palabras)

Fuentes con similitudes fortuitas

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	legislacion.scjn.gob.mx | Suprema Corte de Justicia de la Nación https://legislacion.scjn.gob.mx/Buscador/Paginas/wfArticuladoFast.aspx?q=/HNedHfTnAZnOeiqJv...	< 1%		Palabras idénticas: < 1% (36 palabras)
2	ru.dgb.unam.mx https://ru.dgb.unam.mx/bitstream/20.500.14330/TES01000835736/3/0835736.pdf	< 1%		Palabras idénticas: < 1% (23 palabras)
3	accedacris.ulpgc.es https://accedacris.ulpgc.es/jspui/bitstream/10553/105252/1/TFG Moussati Lamhamdi NassrEddin...	< 1%		Palabras idénticas: < 1% (15 palabras)
4	Documento de otro usuario #1a16d2 El documento proviene de otro grupo	< 1%		Palabras idénticas: < 1% (21 palabras)
5	certidevs.com | Spring Boot inyección de dependencias: uso https://certidevs.com/tutorial-spring-boot-inyeccion-de-dependencias	< 1%		Palabras idénticas: < 1% (10 palabras)

Fuente ignorada Estas fuentes han sido retiradas del cálculo del porcentaje de similitud por el propietario del documento.

N°	Descripciones	Similitudes	Ubicaciones	Datos adicionales
1	JRomero250115.pdf | JRomero250115 #7c1267 El documento proviene de mi grupo	84%		Palabras idénticas: 84% (25.316 palabras)

Fuentes mencionadas (sin similitudes detectadas) Estas fuentes han sido citadas en el documento sin encontrar similitudes.

1. https://bj.scjn.gob.mx/datos-abiertos/conjunto-datos/tesis
2. https://datos.gob.mx/busca/organization/scjn
3. https://www.scjn.gob.mx/sites/default/files/comunicacion_digital/2021
4. https://commons.wikimedia.org/wiki/File%3
5. https://www.agile42.com/en/agile-community/agile-info-center/scrum-nutshell