ULPGC Universidad de Las Palmas de Gran Canaria | ESCUELA DE INGENIERÍA INFORMÁTICA

Trabajo de Fin de Título

Sistema asistido para la creación de Buscadores con Inteligencia Lingüística usando tecnología InteLiTextel

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Omar López Guillén

TUTORIZADO POR:

Francisco Javier Carreras Riudavets

Fecha Junio de 2024

Agradecimiento

Quiero expresar mi más profundo agradecimiento a todas las personas que me han apoyado y guiado durante la realización de este proyecto.

En primer lugar, agradezco a mis padres y a mi hermano por su apoyo incondicional. Su confianza en mí y su constante aliento han sido fundamentales para completar este trabajo.

Un agradecimiento especial a mi tutor del Trabajo de Fin de Título, Francisco Javier Carreras Riudavets. Su orientación experta, sus consejos y su paciencia han sido indispensables para la consecución de este proyecto.

También quiero reconocer a mis compañeros, quienes han contribuido con valiosas ideas y han estado presentes en los momentos difíciles. Su colaboración y amistad han sido una gran fuente de motivación.

A todos, muchas gracias por estar ahí y ayudarme a alcanzar este logro.

Resumen

Este Trabajo Fin de Título se centró en el desarrollo de una aplicación web utilizando la tecnología Microsoft ASP .NET. La aplicación permite a los usuarios registrarse y crear corpus personalizados, que son colecciones de archivos de texto de diferentes ámbitos seleccionados por los usuarios. Una vez cargados los archivos en el sistema, los usuarios pueden acceder a un buscador integrado que permite realizar búsquedas avanzadas dentro del corpus activo. El buscador facilita la búsqueda de frases específicas analizadas sintácticamente, identificando lemas, sustantivos, adjetivos, entre otros elementos. Además, durante la creación de un corpus, los usuarios especifican metadatos para cada archivo, como autor, edad y fecha de nacimiento, los cuales son utilizados por el buscador para refinar los resultados. Este sistema ofrece a los usuarios una herramienta poderosa para analizar y localizar frases específicas dentro de grandes volúmenes de texto, mejorando la eficiencia de la investigación y el análisis de datos lingüísticos.

Palabras clave: Trabajo Fin de Título, Microsoft ASP.NET, corpus textuales, análisis sintáctico, metadatos, MySQL.

Abstract

This Final Degree Project focused on developing a web application using Microsoft ASP .NET technology. The application allows users to register and create customized corpora, which are collections of text files from various fields chosen by the users. Once the files are uploaded to the system, users can access an integrated search engine that enables advanced searches within the active corpus. The search engine aids in finding specific phrases that have been syntactically analyzed, identifying lemmas, nouns, adjectives, among other elements. Additionally, during the creation of a corpus, users specify metadata for each file, such as the author, age, and date of birth, which the search engine uses to refine the results. This system offers users a powerful tool for analyzing and locating specific phrases within large volumes of text, enhancing the efficiency of research and linguistic data analysis.

Keywords: Final Degree Project, Microsoft ASP.NET, textual corpora, syntactic analysis, metadata, MySQL.

Índice general

Capítulo 1: Introducción 9
Capítulo 2: Estado actual y objetivos 10
2.1 Estado actual(Descripción general de las Plataformas) 10
2.2 Viabilidad del Proyecto 11
2.3 Objetivo 12
Capitulo 3: Competencias específicas y aportaciones del trabajo 14
3.1 Competencias 14
3.2 Aportaciones del TFT 15
Capítulo 4: Desarrollo 16
4.1 Metodología de Desarrollo 16
4.2 Lógica de la aplicación 17
4.2.1 Flujo de la aplicación 17
4.4 Herramientas empleadas 31
4.4.1 Entorno Visual Studio 31
4.4.2 ASP .NET 32
4.4.3 MySQL Workbench 33
4.5 Desarrollo de la aplicación Web 34
4.5.1 Primeros pasos en Visual Studio 34
4.5.2 Preparación de la base de datos 36
4.5.3 Login/Registro de usuarios 41
4.5.4 HomePage 43
4.5.5 Corpus 45
4.5.6 Creación de un Corpus 46
4.5.7 Buscador de un Corpus 53
Capítulo 5: Conclusiones y trabajo futuro 59
5.1 Evaluación de Resultados y Grado de Consecución de los Objetivos 59
5.2 Conclusiones Personales 60
5.3 Futuras ampliaciones 60
Capítulo 6: Bibliografía 62

Índice de figuras

Figura 1: Yext....10
Figura 2: INCEpTION....10
Figura 3: Open Semantic Search....11
Figura 4: Curatr....11
Figura 5: Flujo de la aplicación....19
Figura 6: Flujo de datos InteLiText....21
Figura 7: Página login....22
Figura 8: Página register....22
Figura 9: HomePage sin loguearse....23
Figura 10: HomePage logueado....24
Figura 11: Página corpus....25
Figura 12: Primera página creación del corpus....26
Figura 13: Segunda página de creación del corpus....27
Figura 14: Tercera página de creación del corpus....29
Figura 15: Página del buscador....30
Figura 16: Desplegable Cat. Gramatical....31
Figura 17: Desplegable de ejemplos....31
Figura 18: Empezando el proyecto....35
Figura 19: Eligiendo arquitectura....35
Figura 20: Vista del proyecto creado....36
Figura 21: Tabla users con datos....37
Figura 22: Tabla corpus con datos....37
Figura 23: Código sql de las tablas....38
Figura 24: Tabla textos con datos....39
Figura 25: Tabla frases con datos....39
Figura 26: Diccionario cat. Gramatical/identificador....40
Figura 27: Tabla palabras con datos....40
Figura 28: Código sql tabla textos....41
Figura 29: Código sql tabla frases....41
Figura 30: Código sql tabla palabras....41
Figura 31: Función VerificarUsuario....42
Figura 32: Cookie de autenticación de usuario....42

Figura 33: Función RegistrarUsuario....42
Figura 34: Función ExisteCorreo....43
Figura 35: Clase PasswordHasher....43
Figura 36: Manejo visibilidad link mis corpus....44
Figura 37: Html del link corpus....44
Figura 38: Html del icono de usuario....45
Figura 39: html listado de corpus....45
Figura 40: Función CargarCorpus....45
Figura 41: Lógica eliminar un corpus....46
Figura 42: Botón añadir metadato dinámico....46
Figura 43: Botón eliminar metadato dinámico....47
Figura 44: Función InsertarNuevoCorpus....47
Figura 45: Función CrearSchema....48
Figura 46: Función ObtenerTipoColumna....48
Figura 47: Botón para pasar a la fase 3 de creación de corpus....48
Figura 48: Vista del fichero frases.txt....49
Figura 49: Vista del fichero párrafos.txt....49
Figura 50: Componente FielUpload asp.net....50
Figura 51: Función ProcesadorArchivo....51
Figura 52: Función almacenar....52
Figura 53: Función GetFrasesAndDocuments....53
Figura 54: Función CreateMySqlQuery....54
Figura 55: Constructor de la clase DatabasePatternSinOrden....54
Figura 56: Función CreateInnerJoinCondition....55
Figura 57: Atributo QUERY_PATRON....55
Figura 58: Constructor DatabasePattern....56
Figura 59: Función GetCorpusTextualData....56
Figura 60: Función CreateMysqlQuery....57
Figura 61: Funciones de InformationExtractor....57
Figura 62: Clase Pagination....58
Figura 63: Primera página de una búsqueda....59
Figura 64: Segunda página de una búsqueda....59

Capítulo 1: Introducción

En la era de la información, la capacidad de procesar y analizar grandes cantidades de datos textuales ha cobrado una importancia crucial para múltiples industrias y campos de estudio. Las empresas, instituciones académicas y organizaciones gubernamentales se enfrentan al desafío de extraer conocimientos útiles de volúmenes masivos de datos textuales no estructurados. InteliText, una innovadora tecnología de análisis semántico desarrollada por el tutor de este proyecto ha demostrado ser una herramienta valiosa en la segmentación y comprensión del contenido textual, permitiendo a los usuarios descomponer documentos en párrafos, frases y palabras para un análisis detallado del léxico.

Este método de segmentación y procesamiento facilita la búsqueda y el análisis en textos extensos, proporcionando a los usuarios un acceso más profundo a la información. No obstante, a pesar de su utilidad, el método existente presenta una desventaja significativa: requiere un enfoque personalizado para la creación de una página web que se adapte a los documentos y metadatos específicos de cada cliente. Esto obliga a un proceso manual laborioso, limitando la escalabilidad y el alcance de la tecnología. El proyecto actual surge como una respuesta a esta problemática, proponiendo el desarrollo de una plataforma web que permita a los usuarios gestionar sus propios corpus de manera autónoma y sin la necesidad de intervención manual.

1.1 Motivación

La motivación principal de este proyecto reside en la necesidad de abordar las limitaciones de escalabilidad y accesibilidad de la solución actual. Los clientes dependen de un proceso personalizado y manual para implementar sus corpus documentales en páginas web, lo que genera un cuello de botella que limita tanto el tiempo como los recursos disponibles para atender a un público más amplio. Esta dependencia genera retrasos innecesarios y una experiencia de usuario menos eficiente.

Asimismo, existe una creciente demanda por parte de individuos, empresas y organizaciones que desean aprovechar el análisis lingüístico en sus propios documentos, con la flexibilidad de poder definir y gestionar sus propios corpus según sus necesidades específicas. La falta de autonomía y la dependencia de un intermediario dificultan la expansión de la tecnología y reducen su impacto potencial en la democratización del acceso al análisis textual avanzado.

Capítulo 2: Estado actual y objetivos

2.1 Estado actual(Descripción general de las Plataformas)

El análisis lingüístico y la búsqueda avanzada en corpus textuales se han convertido en áreas clave para empresas e investigadores que buscan extraer significado y valor de grandes volúmenes de información. Para abordar estas necesidades, han surgido diversas plataformas que ofrecen soluciones únicas para la gestión de documentos y la búsqueda lingüística.

Yext

Proporciona un servicio que utiliza la tecnología de procesamiento del lenguaje natural (NLP) y modelos de aprendizaje automático como BERT. Su enfoque se basa en comprender la intención del usuario detrás de las consultas, permitiendo búsquedas semánticas más precisas. Además, integra la inteligencia artificial en la gestión de respuestas y en el descubrimiento de información, beneficiando a empresas que desean mejorar la calidad y relevancia de la información brindada a sus clientes. [13]

Figura 1: Yext

INCEpTION

Desarrollada por la Universidad Técnica de Darmstadt, INCEpTION es una plataforma web orientada a la anotación semántica y gestión de conocimiento. Permite a los usuarios crear y manejar sus propios corpus documentales. Además, integra un marco unificado para la búsqueda de datos y la ampliación de bases de conocimiento, soportando la anotación colaborativa y multilingüe. Esto lo convierte en una herramienta valiosa para la generación de datos anotados y el desarrollo de modelos de procesamiento del lenguaje natural. [3]

Figura 2: INCEpTION

Open Semantic Search

Es un motor de búsqueda de código abierto que incorpora una plataforma de minería de texto y análisis. Ofrece funcionalidades para el análisis de documentos, reconocimiento de entidades y gestión de metadatos. Su enfoque incluye la búsqueda facetada, que ayuda a filtrar y visualizar la información desde múltiples perspectivas, proporcionando una experiencia más completa y comprensiva al usuario. [9]

Figura 3: Open Semantic Search

Curatr

Se especializa en la curation de textos históricos, particularmente para investigadores en humanidades. La plataforma permite el análisis semántico a través de lexicones personalizados y ofrece una interfaz interactiva para explorar términos relacionados con diferentes temas. Esto ayuda a descubrir textos históricos valiosos, vinculando temas de interés con términos previamente no identificados, facilitando la búsqueda de documentos significativos (ar5iv).

Objetivo. [1]

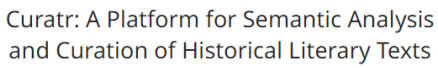


Figura 4: Curatr

2.2 Viabilidad del Proyecto

Comparando las características y funcionalidades de las plataformas mencionadas con este proyecto, hay varias oportunidades que lo hacen viable y único:

- **Flexibilidad de Gestión:** Ofrece a los usuarios la capacidad de gestionar sus propios corpus documentales sin intervención manual, lo que simplifica y acelera el proceso.

- **Búsqueda Personalizada:** Al centrarse en búsquedas basadas en inteligencia lingüística, la plataforma se alinea con las necesidades de los usuarios para encontrar información relevante en corpus documentales personalizados.

- **Escalabilidad:** La arquitectura propuesta permite que la plataforma sea escalable para diferentes tamaños de corpus y para diversos tipos de usuarios.

- **Integración Completa:** Combina análisis lingüístico avanzado, gestión de documentos, búsqueda, informes y visualización en un solo entorno, lo que proporciona una experiencia fluida para el usuario.

En conclusión, el proyecto es viable y tiene el potencial de ofrecer una solución única en un mercado donde la gestión de documentos y el análisis lingüístico avanzado son cada vez más demandados.

2.3 Objetivo

Para resolver los problemas antes mencionados y capitalizar las oportunidades existentes, el proyecto propone los siguientes objetivos:

- **Desarrollo de una plataforma web autónoma:** Crear una aplicación que permita a los usuarios registrarse, iniciar sesión y gestionar sus corpus documentales de forma independiente, sin necesidad de intervención manual.

- **Integración de InteliText:** Incorporar las funcionalidades de InteliText en la plataforma web para que los usuarios puedan aprovechar el análisis semántico avanzado de la tecnología.

- **Gestión flexible de documentos:** Ofrecer la posibilidad de que los usuarios suban, clasifiquen y personalicen la organización de sus documentos, de forma que puedan crear corpus personalizados y acceder a información relevante.

- **Análisis detallado e informes:** Proporcionar a los usuarios informes y resultados de análisis textuales detallados y personalizables, facilitando la búsqueda, exploración y extracción de información según los intereses específicos.

- **Interfaz amigable y experiencia de usuario:** Diseñar una interfaz intuitiva y un flujo de trabajo claro que facilite la creación y gestión de corpus, eliminando la complejidad innecesaria.

- **Escalabilidad y adaptabilidad:** Asegurar que la plataforma pueda crecer de forma sostenible y adaptarse a distintos tipos de usuarios y volúmenes de documentos.

- **Registro de usuarios:** Implementar un sistema de registro que permita la creación de cuentas de usuario, para facilitar la personalización de la experiencia en la plataforma.

- **Diseño y creación de una base de datos:** Crear una base de datos que contenga la información necesaria para el funcionamiento eficiente de la plataforma, incluyendo datos de usuarios, corpus documentales y metadatos.

- **Búsquedas con inteligencia lingüística:** Desarrollar un sistema que permita a cada usuario realizar búsquedas avanzadas, basadas en inteligencia lingüística, en los corpus documentales que haya creado.

Con estos objetivos, el proyecto busca revolucionar la forma en que los usuarios interactúan con el análisis semántico, eliminando las barreras para una experiencia más ágil, intuitiva y accesible.

Capitulo 3: Competencias específicas y aportaciones del trabajo

3.1 Competencias

Capacidad para planificar, concebir, desplegar y dirigir proyectos (CI2):

El proyecto demuestra esta competencia al desarrollar una plataforma web desde cero, gestionando todos los aspectos del desarrollo, desde la planificación inicial hasta la implementación y el despliegue final. Se ha liderado la puesta en marcha del sistema y contribuido a su mejora continua, lo que refleja un entendimiento profundo del impacto económico y social de las tecnologías implementadas.

Conocimiento y diseño eficiente de tipos y estructuras de datos (CI7):

Al trabajar con MySQL y estructurar esquemas y tablas de manera eficiente para soportar el análisis semántico de textos, tu proyecto ilustra la habilidad para seleccionar y utilizar estructuras de datos que optimizan la resolución de problemas específicos, en este caso, el análisis y la gestión de grandes volúmenes de datos textuales.

Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente (CI8):

En el proyecto se ha diseñado una aplicación que no solo es funcional, sino también segura. Se incorporaron aspectos críticos como la encriptación de contraseñas y la autenticación de usuarios, garantizando la robustez y eficiencia en el manejo de sesiones y datos. Esta implementación asegura que la aplicación maneje información sensible de manera segura y eficaz, cumpliendo con los estándares de seguridad informática necesarios.

Conocimiento y aplicación de las características y estructura de las bases de datos (CI12):

La habilidad para implementar y manejar una base de datos MySQL para soportar esta plataforma demuestra un profundo conocimiento de cómo las bases de datos pueden ser diseñadas y optimizadas para mejorar el rendimiento y la escalabilidad de las aplicaciones web.

Capacidad de concebir sistemas basados en tecnologías de red (TI6):

El desarrollo de una plataforma web que permite a los usuarios interactuar con un sistema basado en la nube a través de Internet, utilizando tecnologías web modernas (ASP .NET) para comercio electrónico y servicios interactivos, refleja la competencia en la creación de soluciones que aprovechan la infraestructura de red y las tecnologías emergentes.

3.2 Aportaciones del TFT

El proyecto en cuestión representa un avance significativo en la aplicación de tecnologías de procesamiento de texto avanzadas, desempeñando un papel crucial en la democratización del acceso a herramientas de big data y análisis semántico. Al facilitar el uso de estas tecnologías a una gama más amplia de usuarios y organizaciones, se contribuye a incrementar la eficiencia operativa y a fomentar decisiones más informadas y basadas en datos. Esta expansión del acceso no solo potencia la productividad organizacional, sino que también promueve un entorno más inclusivo en el ámbito tecnológico, permitiendo que una variedad de actores aproveche las ventajas del análisis de datos complejos para mejorar sus operaciones y estrategias.

Desde un enfoque técnico, el desarrollo de una plataforma robusta y escalable para el análisis de textos marca un hito importante para la comunidad tecnológica. Esta plataforma sirve como referencia y punto de partida para futuros desarrollos en el campo del procesamiento de lenguaje natural y análisis de datos. Al proporcionar un marco técnico sólido y adaptable, el proyecto impulsa la innovación y estimula el progreso continuo en técnicas y aplicaciones relacionadas, extendiendo su impacto mucho más allá de su implementación inicial.

En el ámbito científico, el trabajo realizado para integrar y aplicar técnicas avanzadas de ingeniería de software y gestión de bases de datos en la resolución de problemas prácticos de análisis de textos ofrece valiosos descubrimientos. Estos hallazgos enriquecen el conocimiento existente y sientan las bases para futuras investigaciones en este campo. La experimentación y aplicación de nuevas metodologías no solo avanzan la comprensión teórica, sino que también prueban su viabilidad y efectividad en escenarios del mundo real, contribuyendo significativamente a la literatura científica y técnica sobre el tema.

Este enfoque integral refleja un entendimiento profundo de las intersecciones entre tecnología, ciencia y beneficios socioeconómicos, destacando la importancia de proyectos que no solo buscan innovar en términos técnicos, sino también en generar un impacto positivo palpable en la sociedad y la economía.

Capítulo 4: Desarrollo

4.1 Metodología de Desarrollo

Introducción a la Metodología

Este proyecto adopta la metodología de desarrollo en cascada debido a su estructura clara y secuencial, que se alinea perfectamente con la naturaleza del proyecto y el hecho de ser el único desarrollador. La metodología en cascada permite un enfoque sistemático y disciplinado, dividido en fases bien definidas que deben completarse una tras otra. Este enfoque es ideal para proyectos donde los requisitos están claramente definidos y es poco probable que cambien dramáticamente, como es el caso de la creación de una plataforma para búsquedas lingüísticas avanzadas. Dado que ya contábamos con la tecnología interna para el análisis de textos, los pasos a seguir estaban claramente definidos desde el inicio, eliminando la posibilidad de cambios radicales en el proceso.

Fase 1: Análisis de Requisitos

La fase de análisis comenzó con una serie de discusiones intensivas con mi tutor, durando entre 5 y 8 horas. Estas reuniones fueron cruciales para definir y documentar las necesidades y expectativas del proyecto. Durante esta etapa, se establecieron las bases del proyecto, definiendo claramente el propósito de la aplicación: permitir a los usuarios crear y gestionar corpus documentales, y realizar búsquedas textuales avanzadas utilizando la tecnología InteLiText.

Fase 2: Diseño del Sistema

Con los requisitos bien definidos, procedí a la etapa de diseño, que tomó aproximadamente 10 horas. Esta fase implicó la elaboración de la arquitectura general de la aplicación, especificando las relaciones entre los diversos componentes del sistema, como la interfaz de usuario, la lógica del negocio, y la base de datos. Se diseñaron también los flujos de trabajo y las interfaces, asegurando que la aplicación fuera tanto funcional como intuitiva para los usuarios finales.

Fase 3: Implementación

En esta fase de implementación, se ha dedicado aproximadamente 250 horas a traducir los diseños detallados en código funcional, utilizando ASP.NET. Esta etapa es la más extensa y técnica, ya que implica la escritura del código fuente, la configuración de la base de datos, y la integración de todas las funcionalidades y servicios planificados. También como suele ser habitual en el desarrollo de software, surgieron nuevas ideas y perspectivas que no se habían considerado inicialmente. Estos descubrimientos llevaron a varios cambios en el diseño original durante el proceso de implementación. La flexibilidad para adaptar y refinar el diseño de la aplicación fue crucial, permitiendo mejorar la funcionalidad y la usabilidad de la plataforma a medida que

emergían nuevos requisitos y oportunidades de optimización. Este enfoque iterativo aseguró que la aplicación no solo cumpliera con los requisitos originales, sino que también incorporara mejoras significativas que enriquecieran el producto final.

Fase 4: Pruebas

Una vez finalizada la implementación, seguirá la fase de pruebas, con un estimado de 30 horas dedicadas a verificar que la aplicación funcione correctamente en diferentes escenarios y condiciones. Esta etapa es vital para asegurar la calidad y la estabilidad del software, y para corregir cualquier error o problema no detectado durante la implementación.

Ajuste a la Planificación y Cambios en los Objetivos

A lo largo del proyecto, se ha mantenido una revisión constante del progreso en relación con la planificación inicial. Aunque algunos ajustes han sido necesarios, principalmente en los tiempos asignados a cada fase debido a la complejidad de ciertas tareas, los objetivos del proyecto han permanecido estables, con un enfoque firme en proporcionar una herramienta robusta para el análisis lingüístico a través de corpus documentales.

Conclusión

La metodología en cascada ha demostrado ser una elección acertada para este proyecto, proporcionando una estructura clara y permitiendo un control riguroso del desarrollo en cada etapa. Este enfoque metódico asegura que cada fase del proyecto sea completada con la máxima atención al detalle y alineación con los objetivos globales.

Este capítulo demuestra cómo cada decisión de diseño y cada selección de herramienta ha sido guiada por los principios de claridad, eficiencia, y funcionalidad, asegurando así el éxito en la creación de una plataforma innovadora para la investigación lingüística.

4.2 Lógica de la aplicación

4.2.1 Flujo de la aplicación

La aplicación web presenta un flujo lógico diseñado para manejar tanto usuarios registrados como no registrados de manera eficiente. Cuando un usuario no está registrado, tiene acceso limitado a ciertas áreas de la aplicación. Puede visitar la página principal (Home) y ver las secciónes "About Us" y “Help” para obtener información sobre la aplicación y su propósito, pero no tiene acceso a funcionalidades avanzadas.

En cambio, los usuarios registrados tienen acceso completo a la aplicación. Al iniciar sesión, pueden acceder a la página de Corpus, donde se les presenta una lista de los corpus que han creado. Si un usuario no tiene ningún corpus, se le ofrece la opción de crear uno nuevo. El proceso de creación de un corpus sigue varias etapas bien definidas. Primero, el usuario debe elegir un nombre y un acrónimo para el nuevo corpus. A continuación, debe definir los metadatos que se aplicarán a todos los ficheros que formarán parte del corpus. Estos metadatos son esenciales para la organización y búsqueda posterior dentro del corpus.

Una vez definidos los metadatos, el usuario procede a subir los ficheros uno por uno. Antes de cada subida, se requiere que el usuario rellene los metadatos correspondientes para cada fichero. Al subir cada fichero, una inteligencia lingüística se encarga de analizar el contenido, extrayendo y almacenando los datos lingüísticos relevantes en la base de datos.

Después de haber creado un corpus, el usuario puede interactuar con él de varias maneras. Una de las funcionalidades clave es el buscador de corpus. Haciendo clic en la lupa del buscador, el usuario es redirigido a una página donde puede realizar búsquedas dentro de ese corpus específico. El usuario puede introducir diversos criterios de búsqueda y optar por realizar una búsqueda vacía, lo que devolverá todos los resultados disponibles, o aplicar filtros específicos para obtener resultados más precisos.

Si los resultados de la búsqueda son numerosos, la aplicación proporciona una funcionalidad de paginación para facilitar la navegación entre los resultados. Además, las frases encontradas y su contexto relevante se muestran con los términos de búsqueda resaltados, lo que permite al usuario identificar rápidamente la información importante.

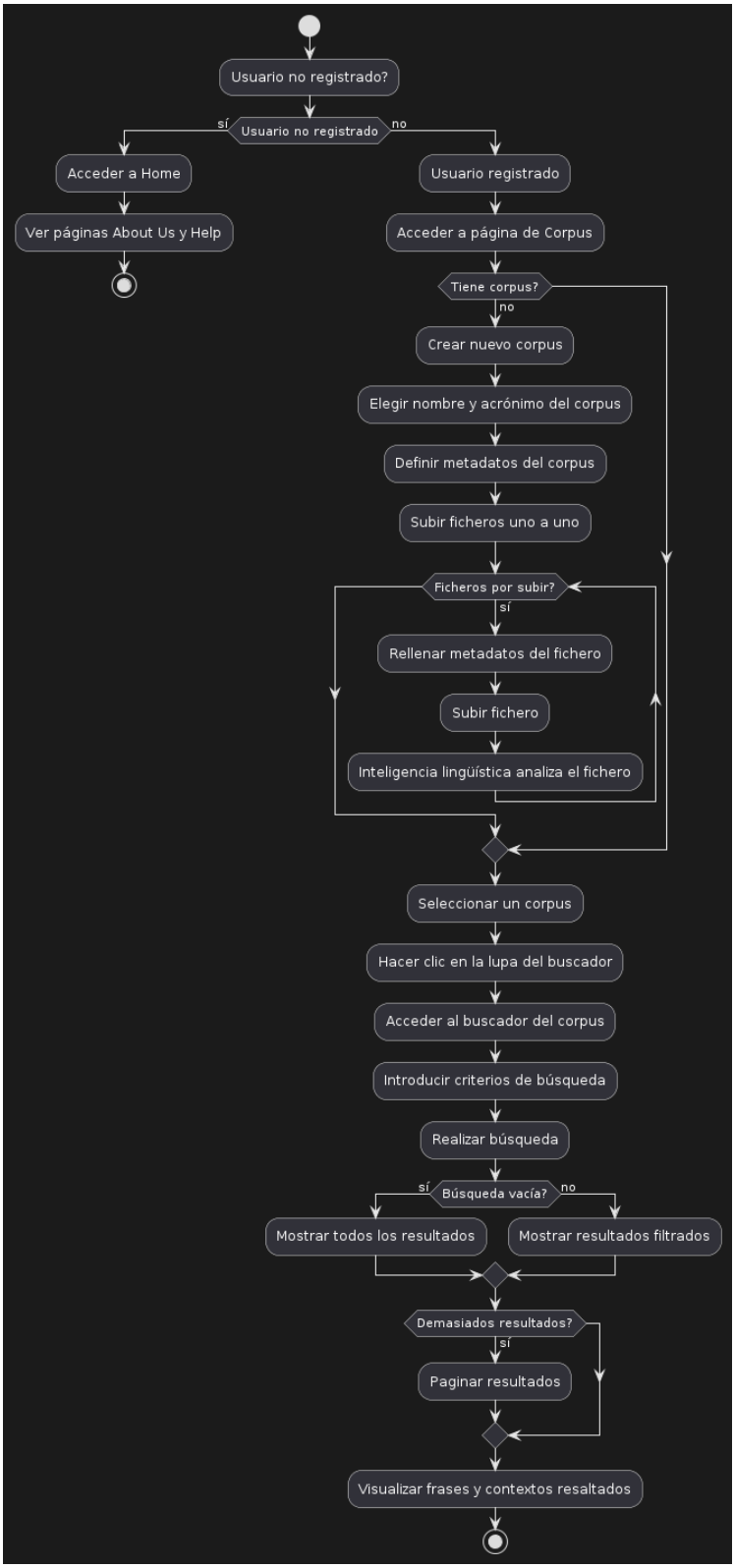


Figura 5: Flujo de la aplicación

4.2.2 Tecnología InteLiText

InteLiText es una tecnología avanzada diseñada para el análisis estructural de textos. A través de una combinación de procesamiento de texto y bases de datos, InteLiText es capaz de descomponer textos en sus componentes esenciales, como párrafos, frases y palabras, y categorizarlos de manera que sean útiles para análisis lingüísticos detallados.

Para ello se apoya de unos ficheros y de ciertas tablas en la base de datos para relacionar todas las palabras y frases entre sí.

Ficheros de Corpus

Para cada corpus procesado, se generan dos archivos clave:

- **frases.txt:** Este archivo contiene todas las frases extraídas de los textos procesados. Cada entrada en el archivo frases.txt incluye el número incremental de la frase, el texto de la frase, el byte de inicio del párrafo al que pertenece, y la longitud del párrafo.
- **parrafos.txt:** Este archivo almacena los párrafos completos, separados por el símbolo "§". La estructura de este archivo permite proporcionar contexto adicional sobre las frases cuando se realiza una búsqueda.

Base de Datos

La base de datos asociada a InteLiText se compone de tres tablas principales:

- **textos:** Esta tabla almacena información relevante sobre los textos procesados, como el nombre del archivo y su identificación dentro del sistema.
- **frases:** Contiene detalles sobre cada frase, incluyendo la longitud, el ID de la frase, el byte de inicio, y el ID del texto al que pertenece para poder acceder rápidamente a la frase a buscar en frases.txt.
- **palabras:** Registra información detallada sobre cada palabra dentro de las frases, como la categoría gramatical, la forma canónica, y la posición de la palabra dentro de la frase.

Proceso de Búsqueda

El proceso de búsqueda en InteLiText se puede describir en los siguientes pasos:

Recepción de la Consulta: El usuario introduce una consulta en el buscador de InteLiText.

Construcción de la Consulta SQL: Dependiendo de los criterios de búsqueda proporcionados por el usuario, se construye una consulta SQL dinámica que se ejecuta sobre las tablas de la base de datos.

Ejecución de la Consulta: La consulta SQL se ejecuta para obtener las frases que cumplen con los criterios especificados.

Acceso a Archivos: Se localiza la frase en el archivo frases.txt utilizando la información de la base de datos (byte de inicio y longitud). Después de localizar la frase se proporciona el contexto de la frase accediendo al archivo parrafos.txt.

Presentación de Resultados: Los resultados se presentan al usuario, resaltando las palabras clave y proporcionando el contexto necesario.

Un diagrama de flujo de datos ayuda a visualizar cómo interactúan los diferentes componentes de InTeLiTex, en la *Figura 6* se puede observar con mayor claridad.

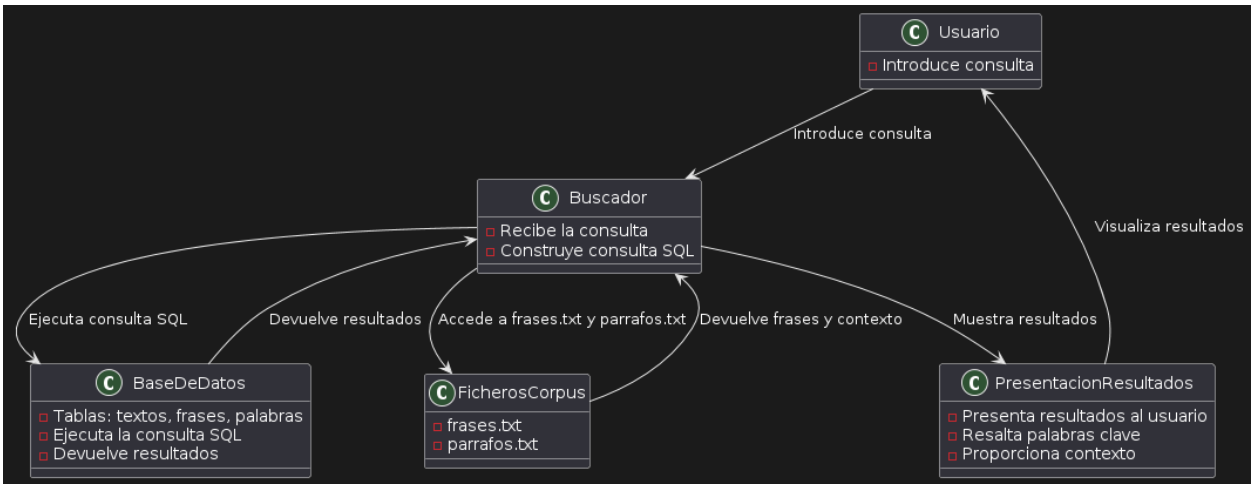


Figura 6: Flujo de datos InteLiText

4.3 Diseño de la aplicación

Pantallas y Justificación del diseño de las vistas

- **Login/Register**

El diseño de las pantallas de Login y Register de nuestra aplicación web es minimalista y moderno, diseñado para facilitar una interacción clara y eficiente. Empleamos espacios en blanco y una tipografía limpia para mejorar la legibilidad y centrar la atención del usuario en elementos

esenciales como los campos de correo electrónico y contraseña. Esta simplicidad ayuda a evitar distracciones y hace que la interfaz sea intuitiva y accesible.

Visualmente, los elementos clave como los campos de entrada y el botón de "Iniciar sesión" son prominentes y fáciles de identificar. Utilizamos un color azul brillante para el botón, lo que asegura que las acciones principales sean rápidamente reconocibles. La paleta de colores neutros contribuye a una estética moderna que no solo es agradable a la vista, sino que también facilita una alta legibilidad.

En resumen, nuestro diseño busca optimizar la experiencia del usuario mediante un enfoque centrado en la funcionalidad y la estética. Los elementos interactivos ofrecen retroalimentación visual efectiva que mejora la interacción. Este enfoque asegura que los usuarios puedan navegar el proceso de inicio de sesión de manera rápida y sin confusiones, mejorando su satisfacción y confianza en la aplicación desde el primer contacto.

Figura 7: Página login

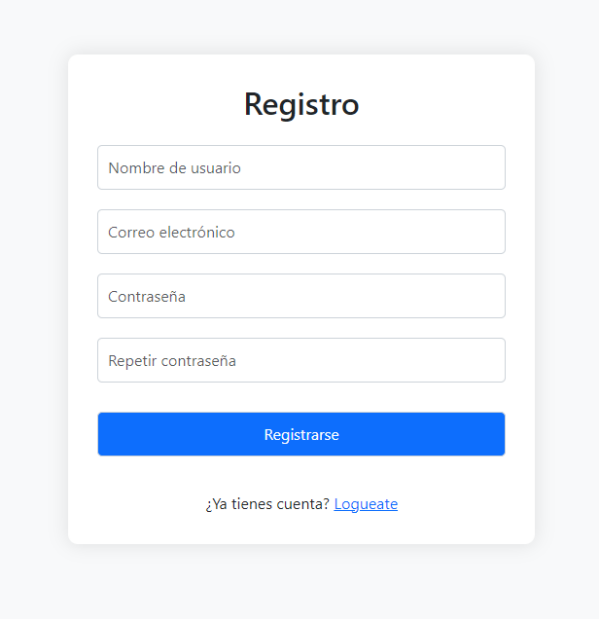


Figura 8: Página register

- **Página principal**

El diseño de la página de inicio de nuestra aplicación web se ha creado con un enfoque en la responsividad, simplicidad y modernidad. Este diseño se seleccionó para proporcionar una experiencia de usuario óptima en una variedad de dispositivos, desde ordenadores de escritorio hasta teléfonos móviles, y para asegurar que la interfaz sea visualmente agradable y funcional.

La responsividad del diseño asegura que la página de inicio se adapte perfectamente a diferentes tamaños de pantalla y dispositivos. Al usar técnicas de diseño responsivo, los elementos de la interfaz, como el menú de navegación, el contenido principal y el pie de página, se reconfiguran automáticamente para ofrecer una experiencia coherente y optimizada. Esto

permite que los usuarios accedan a la aplicación desde cualquier dispositivo con una navegación fluida y una presentación clara de la información.

La funcionalidad del menú de usuario está diseñada para adaptarse a los diferentes estados de autenticación del usuario. Si el usuario no está logueado, al hacer clic en el icono del personaje en la parte superior derecha, será redirigido a la página de login. En cambio, si el usuario está logueado, su nombre de usuario se muestra debajo del icono del personaje. Al hacer clic en este icono, se despliega un menú con opciones para cerrar sesión o ir al perfil. Este menú desplegable está diseñado para ser intuitivo y fácil de usar, y en un futuro se podrían añadir más apartados en este menú para ofrecer funcionalidades adicionales.

Además, cuando el usuario está logueado, se desbloquea un nuevo enlace en la barra de navegación a una página llamada "Corpus". Este enlace lleva a los usuarios a una sección donde pueden ver y gestionar los corpus que han creado, así como crear nuevos corpus. Esta funcionalidad es el núcleo del proyecto, proporcionando a los usuarios una plataforma centralizada para trabajar con sus corpus de datos. La integración de esta funcionalidad de manera fluida y accesible es fundamental para la propuesta de valor de la aplicación.

La estructura del menú desplegable está diseñada pensando en la expansión futura. Se pueden agregar más opciones y apartados en el dropmenu para incluir nuevas funcionalidades o accesos directos, mejorando aún más la experiencia del usuario y la usabilidad de la aplicación.

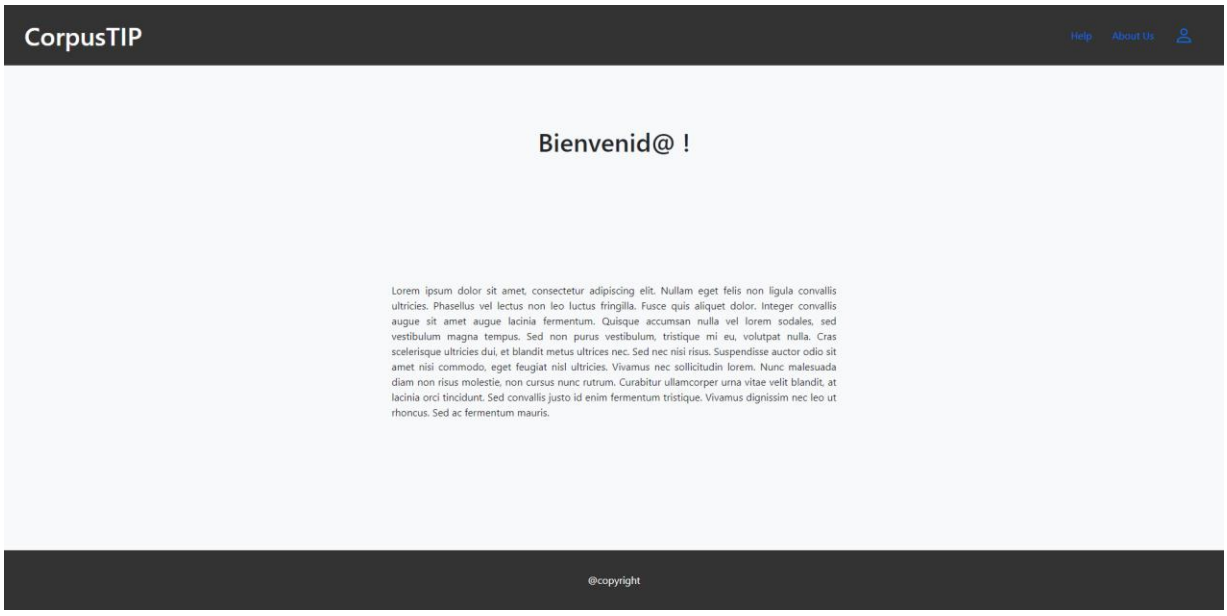


Figura 9: HomePage sin loguearse

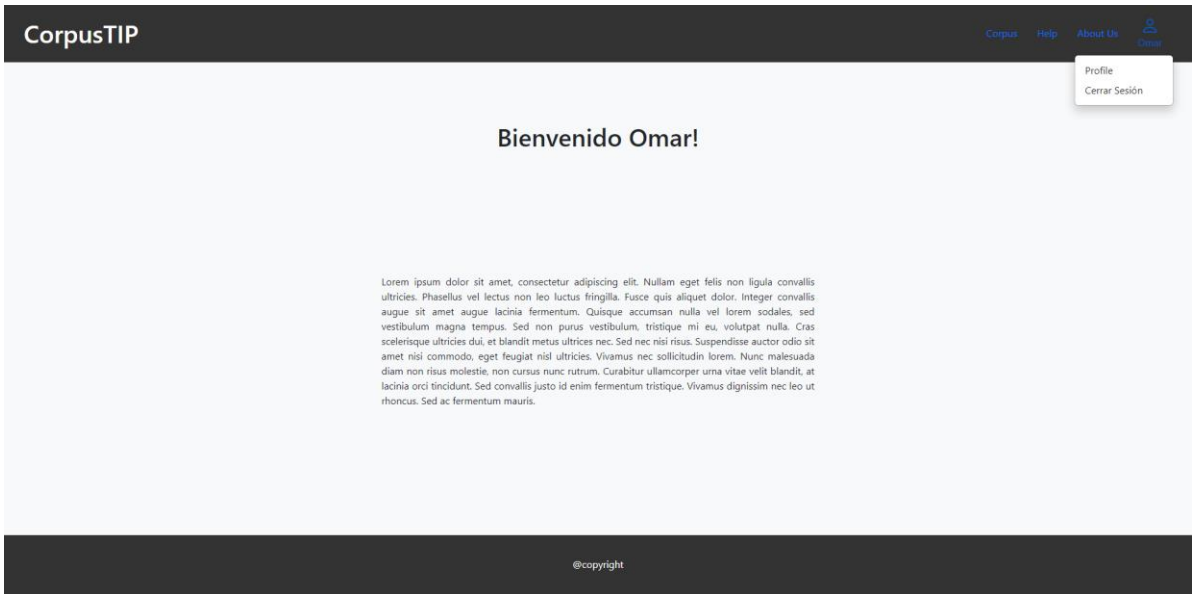


Figura 10: HomePage logueado

- **Corpus**

La página de Corpus de nuestra aplicación web está diseñada para ofrecer una funcionalidad centralizada y accesible a los usuarios que gestionan sus corpus de datos. Este diseño se centra en la usabilidad, la organización y una estética visualmente atractiva, facilitando la interacción del usuario con sus datos.

En la parte superior derecha de la página se encuentra un botón prominente para crear nuevos corpus. La decisión de ubicar el botón en esta posición se debe a la necesidad de garantizar que siempre sea visible y accesible, independientemente de la cantidad de corpus que el usuario tenga. Esto es especialmente importante en escenarios donde los usuarios gestionan numerosos corpus, evitando que el botón se pierda en un desplazamiento hacia abajo y asegurando una accesibilidad inmediata.

Los corpus se muestran en una lista organizada, uno debajo del otro. Cada entrada de la lista presenta información clave de manera clara y concisa: un identificador único y el nombre del corpus. Esta disposición facilita a los usuarios localizar rápidamente y acceder al corpus deseado, mejorando la eficiencia y la experiencia de usuario.

A la derecha de cada corpus en la lista, se encuentran cuatro iconos funcionales, cada uno con un color distintivo para añadir un toque animado y visualmente agradable. Estos iconos permiten a los usuarios realizar acciones específicas de manera intuitiva:

- **Subir Archivos (Icono Azul):** Permite a los usuarios subir nuevos archivos al corpus seleccionado.

- **Buscador (Icono Negro):** Dirige a los usuarios a una herramienta de búsqueda específica dentro del corpus.
- **Editar (Icono Naranja):** Proporciona acceso a las opciones de edición para modificar el nombre o los detalles del corpus.
- **Eliminar (Icono Rojo):** Ofrece una opción para eliminar el corpus, con un diseño que alerta al usuario sobre la importancia de la acción.

El diseño minimalista de la página, con espacios en blanco bien utilizados, asegura que la interfaz no esté sobrecargada y que los usuarios puedan concentrarse en las tareas esenciales. Los colores utilizados en los iconos no solo añaden una estética agradable, sino que también ayudan a diferenciar visualmente las acciones disponibles, facilitando una rápida identificación y uso.

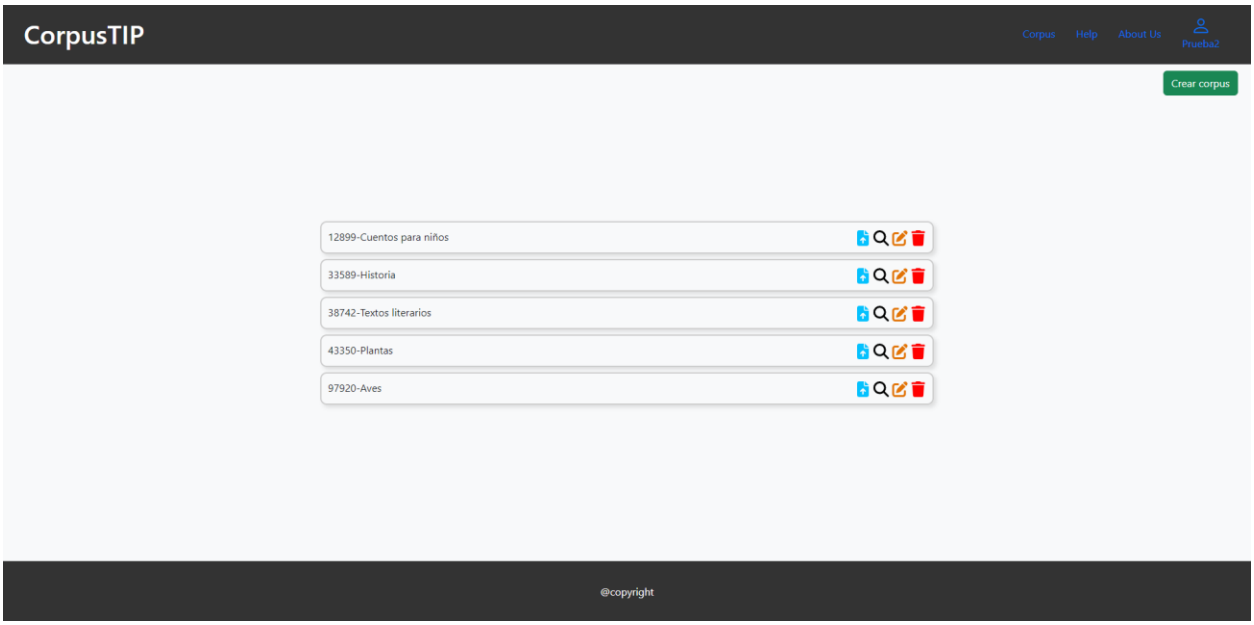


Figura 11: Página corpus

- **Crear corpus**

El proceso de creación de un corpus en nuestra aplicación web se divide en tres etapas principales para facilitar una gestión ordenada y estructurada de los datos. Estas etapas son: elegir el nombre y acrónimo del corpus, seleccionar los metadatos y sus tipos, y finalmente, subir los archivos asociados a cada metadato.

➢ **Etapa 1: Elegir Nombre y Acrónimo**

La primera etapa del proceso de creación de un corpus, como se muestra en la imagen adjunta, es elegir el nombre y el acrónimo del corpus. Esta página está diseñada para ser clara y directa, con dos campos de entrada principales: uno para el título del corpus y otro para el acrónimo. La simplicidad del diseño asegura que los usuarios puedan completar esta etapa rápidamente sin distracciones.

Los campos de entrada están claramente etiquetados y proporcionan instrucciones dentro del campo para guiar al usuario sobre qué información debe ingresar. El uso de espacios en blanco alrededor de los campos de entrada ayuda a mantener una apariencia ordenada y evitar el desorden visual.

El botón "Siguiente" se encuentra alineado a la derecha y está diseñado en un color verde distintivo que se destaca en la página. Esto no solo guía al usuario sobre el siguiente paso en el proceso, sino que también asegura que el botón sea fácilmente identificable.

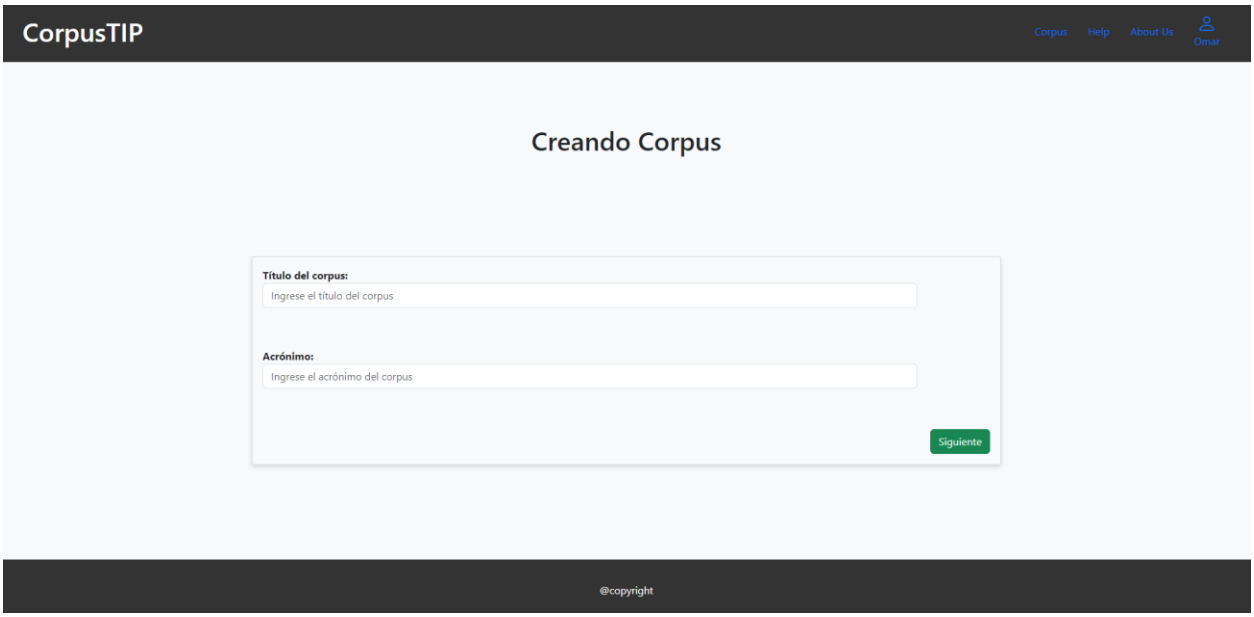


Figura 12: Primera página creación del corpus

- Etapa 2: Seleccionar Metadatos y sus Tipos

La segunda etapa del proceso de creación de un corpus, como se muestra en la imagen adjunta, está diseñada para permitir a los usuarios definir los metadatos que describirán sus datos y especificar el tipo de cada metadato. Esta página es crucial para la organización y búsqueda efectiva de los datos dentro del corpus, proporcionando una estructura clara y detallada.

La interfaz presenta una tabla donde cada fila permite al usuario ingresar un nombre para el metadato y seleccionar su tipo a través de un menú desplegable. Los tipos de metadatos disponibles incluyen opciones como texto, número, fecha, entre otros, ofreciendo la flexibilidad necesaria para describir los datos de manera precisa. Los distintos elementos que destacar de la interfaz son:

Botón de Añadir Metadato: En la parte superior de la tabla, un botón claramente etiquetado como "Añadir metadato" permite a los usuarios agregar nuevas filas a la tabla. Este botón es

esencial para que los usuarios puedan definir múltiples metadatos según las necesidades de su corpus.

Campos de Nombre de Metadato: Cada fila contiene un campo de entrada donde el usuario puede especificar el nombre del metadato. Este campo está claramente etiquetado y es lo suficientemente amplio como para acomodar nombres descriptivos.

Menú Desplegable de Tipo de Metadato: A la derecha del campo de nombre, un menú desplegable permite seleccionar el tipo de metadato. Esta funcionalidad es crucial para asegurar que los datos ingresados se manejen correctamente según su tipo (por ejemplo, texto, número).

Botón Eliminar: A la derecha de cada fila, un botón rojo etiquetado como "Eliminar" permite a los usuarios eliminar metadatos que ya no necesitan. Este botón proporciona una forma sencilla de gestionar y ajustar los metadatos a medida que se avanza en la creación del corpus.

Botón Siguiente: En la esquina inferior derecha, un botón verde claramente visible etiquetado como "Siguiente" guía al usuario hacia la siguiente etapa del proceso de creación del corpus. Este botón asegura que los usuarios puedan proceder fácilmente una vez que hayan definido todos los metadatos necesarios.

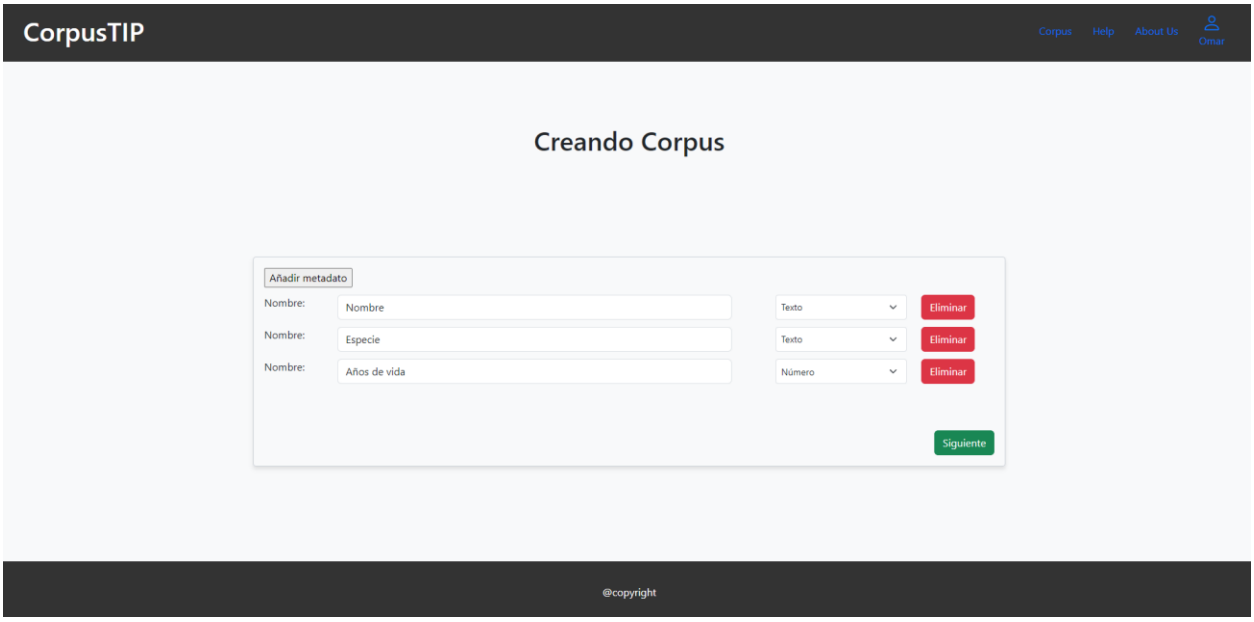


Figura 13: Segunda página de creación del corpus

- Etapa 3: Subir archivos y asociar Metadatos

La tercera etapa del proceso de creación de un corpus, está diseñada para permitir a los usuarios subir archivos de texto (.txt) y asociar cada archivo con los metadatos previamente definidos. Esta etapa es crucial para la integración de datos en el corpus y asegura que cada archivo esté correctamente etiquetado y organizado. Sus elementos mas destacables son:

Selección de Archivo: En la parte superior de la sección principal, hay un botón etiquetado como "Seleccionar archivo" que permite a los usuarios buscar y seleccionar un archivo de texto desde su dispositivo. Una vez seleccionado, el nombre del archivo se muestra junto al botón, confirmando la selección del archivo.

Campos de Metadatos: Debajo del selector de archivos, los campos de metadatos definidos en la etapa anterior se presentan de manera ordenada. Cada campo de entrada está claramente etiquetado con el nombre del metadato y permite al usuario ingresar los valores correspondientes para cada archivo. Por ejemplo, en la imagen, los metadatos incluyen "Nombre," "Especie," y "Años de vida."

Botón Subir Archivo: A la derecha del selector de archivos, un icono de carga (en azul) proporciona una acción visualmente intuitiva para subir el archivo seleccionado. Este botón asegura que el archivo sea procesado y asociado con los metadatos ingresados.

Botón Guardar: En la parte inferior derecha de la sección, un botón verde etiquetado como "Guardar" permite al usuario guardar los cambios y finalizar la subida del archivo. Este botón es crucial para confirmar que los datos ingresados y el archivo seleccionado se han procesado correctamente.

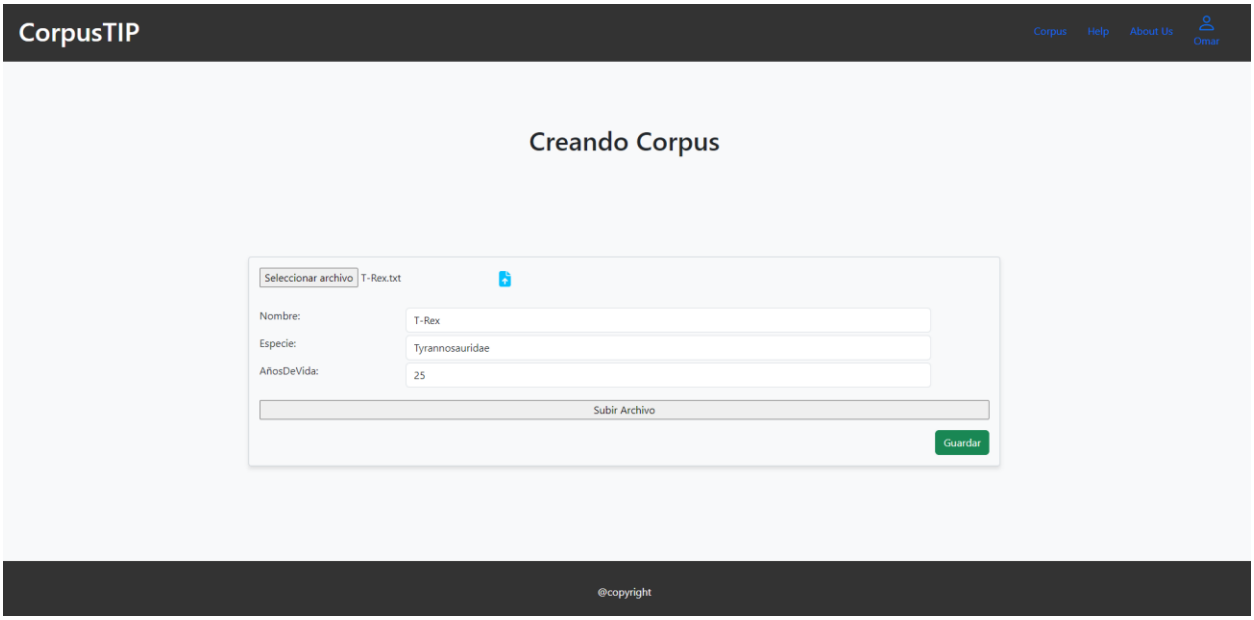


Figura 14: Tercera página de creación del corpus

- **Buscador**

La página del buscador dentro de un corpus está diseñada para proporcionar a los usuarios una herramienta robusta y flexible para la búsqueda y análisis de datos textuales. La interfaz permite realizar búsquedas avanzadas combinando diferentes criterios, asegurando así una experiencia de usuario eficiente y satisfactoria.

La interfaz del buscador está diseñada con un enfoque en la usabilidad y la claridad. En la parte superior, se encuentra un campo de entrada donde los usuarios pueden escribir los términos de búsqueda. Junto a este campo, hay un menú desplegable que permite seleccionar la categoría gramatical, como se muestra en la imagen adjunta. Las opciones incluyen adjetivo, adverbio, artículo, conjunción, interjección, preposición, pronombre, sustantivo y verbo, proporcionando una manera precisa de filtrar los resultados según la categoría gramatical.

Debajo del campo de búsqueda, hay una opción para realizar búsquedas con o sin orden, lo que añade flexibilidad a las consultas. Además, un slider permite a los usuarios seleccionar la cantidad de contexto que desean ver alrededor de los términos de búsqueda encontrados, desde solo la frase hasta un mayor fragmento de texto, lo que es útil para obtener una mejor comprensión del uso de los términos en el contexto.

Además de los criterios mencionados, los usuarios también pueden realizar búsquedas a través de metadatos asociados a los textos. Los campos de metadatos, como "Nombre," "Especie" y "Años de vida," permiten especificar valores precisos para refinar la búsqueda. Esta funcionalidad es especialmente útil para usuarios que necesitan filtrar resultados basados en atributos específicos de los datos textuales.

La interfaz permite combinar todos estos criterios de búsqueda para realizar consultas complejas. Los usuarios pueden escribir términos específicos, seleccionar categorías gramaticales, ajustar el contexto mediante el slider y filtrar por metadatos simultáneamente. Esta capacidad de combinación proporciona una herramienta poderosa para el análisis detallado y preciso de grandes volúmenes de texto.

Para facilitar el uso del buscador, se incluyen ejemplos de búsqueda que muestran cómo utilizar las diferentes funciones. Estos ejemplos son interactivos; los usuarios pueden hacer clic en ellos para probar directamente las búsquedas sugeridas. Esto no solo ayuda a los usuarios a familiarizarse con las capacidades del buscador, sino que también les proporciona una referencia rápida para construir sus propias consultas avanzadas.

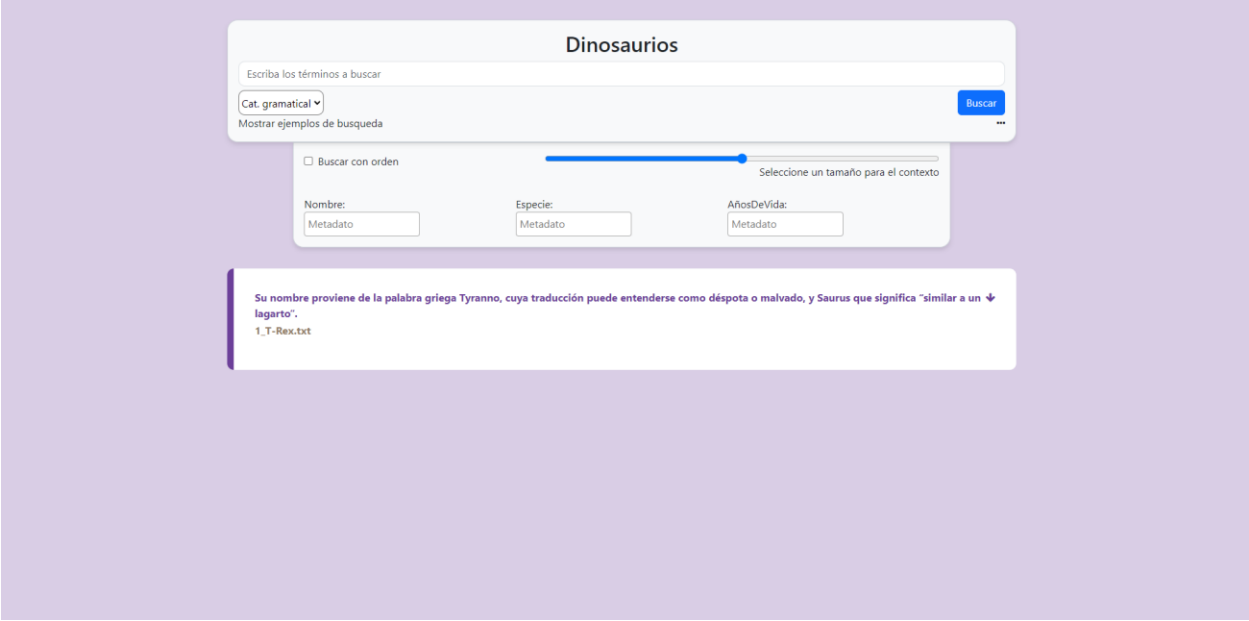


Figura 15: Página del buscador

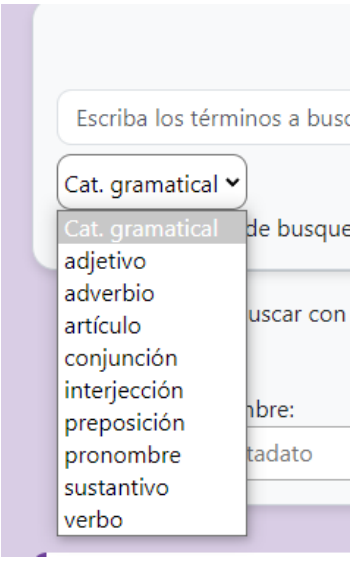


Figura 16: Desplegable Cat. Gramatical

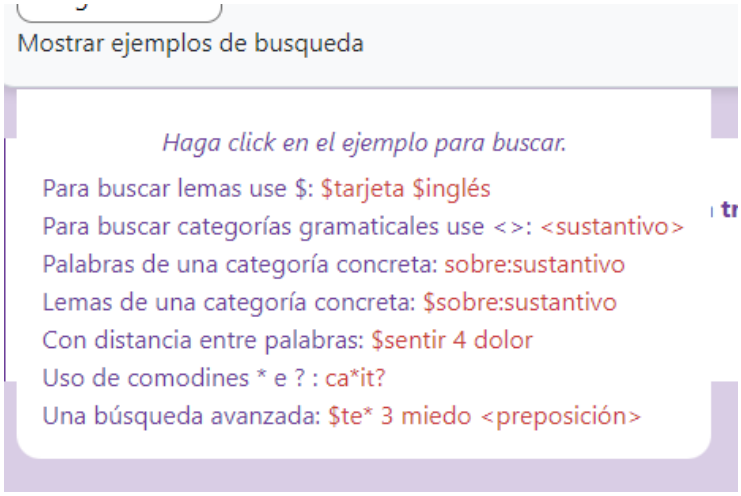


Figura 17: Desplegable de ejemplos

4.4 Herramientas empleadas

4.4.1 Entorno Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) desarrollado por Microsoft. Está diseñado para proporcionar a los desarrolladores un conjunto completo de herramientas necesarias para desarrollar aplicaciones de software eficientes y de alta calidad. Visual Studio soporta varios lenguajes de programación, incluyendo C#, VB.NET, C++, Python, y JavaScript, lo que lo hace muy versátil para diferentes tipos de proyectos.

Facilidades y Ventajas de Visual Studio

IntelliSense: Una de las características más destacadas de Visual Studio es IntelliSense, que ofrece autocompletado de código, documentación en línea y sugerencias contextuales. Esto no solo acelera el proceso de escritura de código, sino que también ayuda a reducir errores y mejora la productividad del desarrollador.

Depuración Avanzada: Visual Studio ofrece potentes herramientas de depuración que permiten a los desarrolladores identificar y corregir errores de manera eficiente. La depuración puede realizarse tanto en aplicaciones locales como en remoto, y proporciona capacidades como puntos de interrupción, inspección de variables, y seguimiento de pila.

Integración con Git: La integración nativa con sistemas de control de versiones como Git permite una gestión sencilla del código fuente. Los desarrolladores pueden realizar commits, gestionar ramas y resolver conflictos directamente desde el IDE, lo que facilita la colaboración en equipo y el control de versiones del proyecto.

Herramientas de Diseño: Visual Studio incluye diseñadores visuales para aplicaciones web, de escritorio y móviles. Estas herramientas permiten a los desarrolladores diseñar interfaces de usuario de manera intuitiva utilizando técnicas de arrastrar y soltar, y ver una vista previa en tiempo real de cómo se verá la aplicación.

Extensibilidad: La capacidad de Visual Studio para ser extendido mediante una amplia variedad de extensiones disponibles en el Visual Studio Marketplace permite a los desarrolladores personalizar el IDE según sus necesidades específicas. Esto incluye desde herramientas adicionales de depuración hasta integraciones con otros servicios y plataformas.

En resumen, Visual Studio es un IDE integral que proporciona todas las herramientas necesarias para el desarrollo de aplicaciones modernas y eficientes. Su capacidad para soportar múltiples lenguajes de programación, herramientas de depuración avanzadas, integración con sistemas de control de versiones, y su extensibilidad lo convierten en una opción ideal para el desarrollo de aplicaciones de cualquier tamaño y complejidad.

4.4.2 ASP .NET

ASP.NET es un marco de desarrollo web desarrollado por Microsoft que permite a los desarrolladores crear aplicaciones web dinámicas, servicios web y aplicaciones web en tiempo real. Proporciona una infraestructura robusta y escalable, facilitando la construcción de aplicaciones de alto rendimiento. ASP.NET es parte del marco .NET y ofrece varias tecnologías para el desarrollo web, entre las que se incluyen:

Web Forms [5]: Un modelo de programación basado en eventos que permite la creación rápida de aplicaciones web con una experiencia similar a la del desarrollo de aplicaciones de escritorio. Los desarrolladores pueden utilizar controles de servidor para generar HTML y manejar eventos del lado del servidor, lo que facilita la creación de aplicaciones web complejas sin necesidad de escribir mucho código HTML y JavaScript.

MVC (Model-View-Controller) [7]: Un patrón de diseño que separa la aplicación en tres componentes principales: el modelo, la vista y el controlador. Esto facilita la gestión del código y mejora la escalabilidad y mantenibilidad de las aplicaciones web. ASP.NET MVC permite un mayor control sobre el HTML y proporciona una mejor separación de preocupaciones en comparación con Web Forms.

Web API: Permite la creación de servicios HTTP que pueden ser consumidos por una variedad de clientes, incluyendo navegadores, dispositivos móviles y aplicaciones de escritorio. ASP.NET Web API es ideal para construir aplicaciones RESTful.

En mi proyecto, he optado por utilizar ASP.NET Web Forms para el desarrollo de la aplicación web. Web Forms ofrece una manera de construir aplicaciones web mediante un modelo de programación orientado a eventos, similar al desarrollo de aplicaciones de escritorio. Esto permite utilizar controles de servidor que generan HTML automáticamente y manejan eventos del lado del servidor, simplificando el desarrollo de aplicaciones web complejas. Web Forms es particularmente útil para desarrolladores que prefieren trabajar con una abstracción mayor sobre el HTML y JavaScript subyacentes.

4.4.3 MySQL Workbench

MySQL Workbench [8] es una herramienta visual de diseño de bases de datos, desarrollada por Oracle, que proporciona una interfaz gráfica para trabajar con bases de datos MySQL. Es una herramienta integral que facilita las tareas de administración de bases de datos, diseño de esquemas, desarrollo de consultas SQL y migración de datos. Su uso en el proyecto se justifica por las siguientes razones:

Facilidades y Ventajas de MySQL Workbench

Diseño de Esquemas: MySQL Workbench permite a los desarrolladores diseñar y modelar esquemas de bases de datos de manera visual. Con herramientas de diagramas EER (Entidad-Relación), los desarrolladores pueden crear, modificar y documentar sus esquemas de base de datos, facilitando la comprensión y comunicación del diseño de la base de datos.

Desarrollo de Consultas: La herramienta incluye un editor SQL avanzado que soporta resaltado de sintaxis, autocompletado de código y análisis de consultas. Esto permite a los

desarrolladores escribir y optimizar consultas SQL de manera eficiente, mejorando la interacción con la base de datos y la ejecución de tareas complejas.

Administración de Bases de Datos: MySQL Workbench ofrece herramientas integradas para la administración de usuarios y permisos, configuración de servidores, gestión de copias de seguridad y restauración, y monitoreo del rendimiento de la base de datos. Esto centraliza las tareas administrativas y simplifica la gestión de la base de datos.

Visualización y Monitoreo: MySQL Workbench proporciona herramientas de visualización y monitoreo que permiten a los desarrolladores y administradores observar el rendimiento de la base de datos en tiempo real. Los gráficos y reportes de rendimiento ayudan a identificar y resolver problemas rápidamente, optimizando la eficiencia de la base de datos.

En resumen, MySQL Workbench es una herramienta esencial para el diseño, desarrollo y administración de bases de datos MySQL. Su uso en el proyecto ha facilitado la creación y gestión de la base de datos, mejorando la eficiencia y efectividad del desarrollo.

4.5 Desarrollo de la aplicación Web

En este apartado se va a mostrar cómo se ha implementado la aplicación desde cero.

4.5.1 Primeros pasos en Visual Studio

Comenzamos creando el proyecto en Visual Studio. Elegimos la plantilla "ASP.NET Web Application (.NET Framework)" ya que vamos a desarrollar una aplicación web. A continuación, seleccionamos "Web Forms" como tipo de proyecto, ya que este modelo nos permite construir aplicaciones web de manera rápida utilizando un modelo de programación basado en eventos.

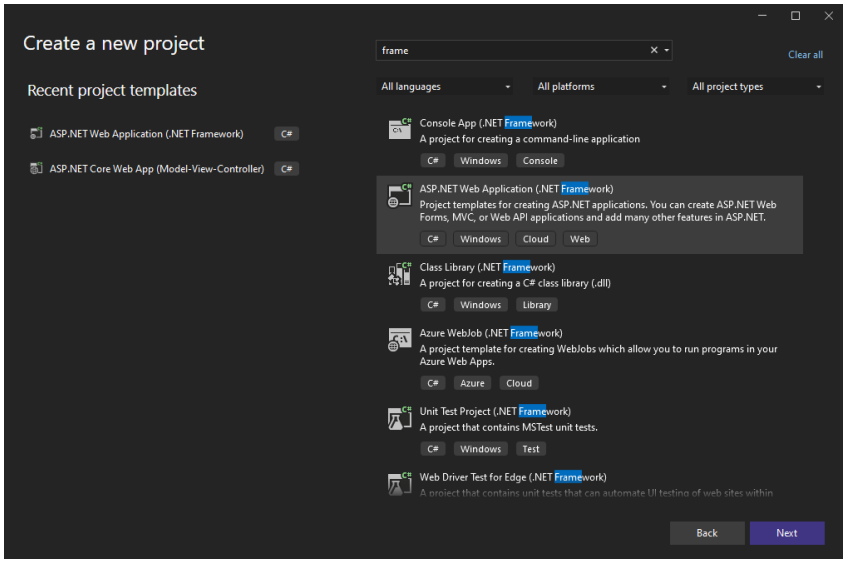


Figura 18: Empezando el proyecto

Una vez seleccionada la plantilla, se nos presenta la opción de elegir el tipo de aplicación web. Optamos por "Web Forms" porque facilita la creación de aplicaciones web dinámicas usando controles de servidor y manejando eventos del lado del servidor.

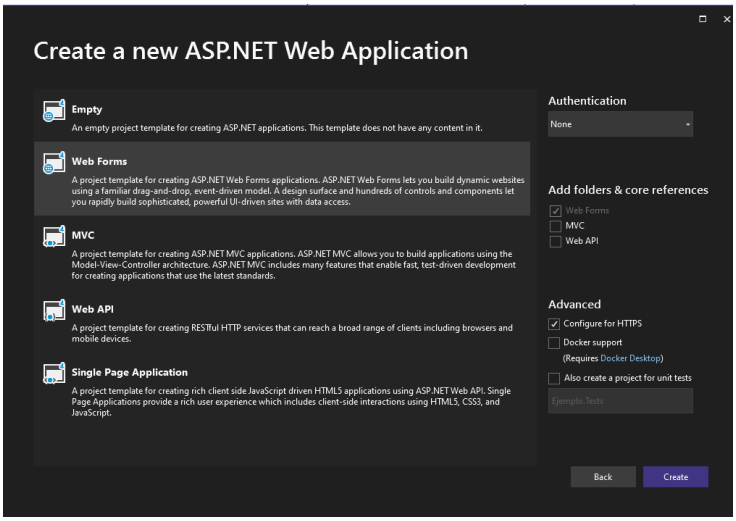


Figura 19: Eligiendo arquitectura

Después de configurar el proyecto, Visual Studio genera una estructura básica que incluye carpetas como App_Code, App_Data, Content, Scripts y Pages. Esta estructura nos proporciona una base organizada desde la cual podemos comenzar a desarrollar nuestra aplicación. A continuación, podemos ver cómo se ve la vista del proyecto en Visual Studio una vez creado.

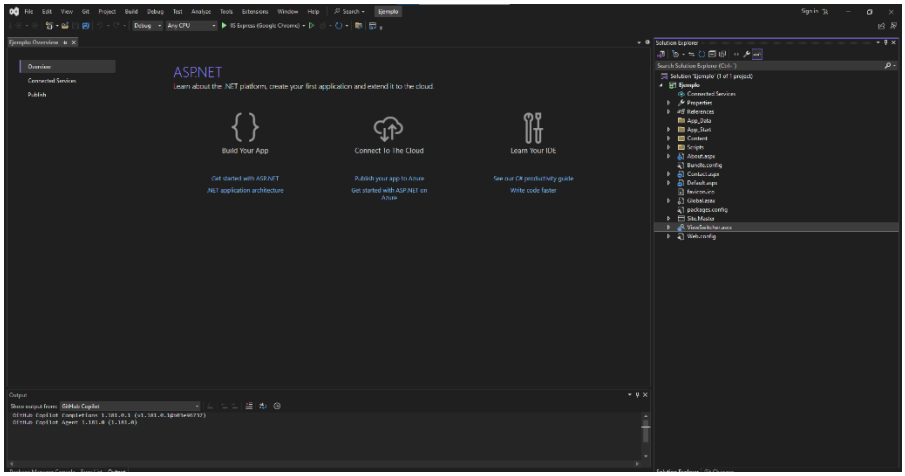


Figura 20: Vista del proyecto creado

Con esta estructura, ya podemos empezar a desarrollar el proyecto.

4.5.2 Preparación de la base de datos

La base de datos tiene distintos esquemas. El primer esquema llamado CorpusTip contiene dos tablas principales: users y corpus.

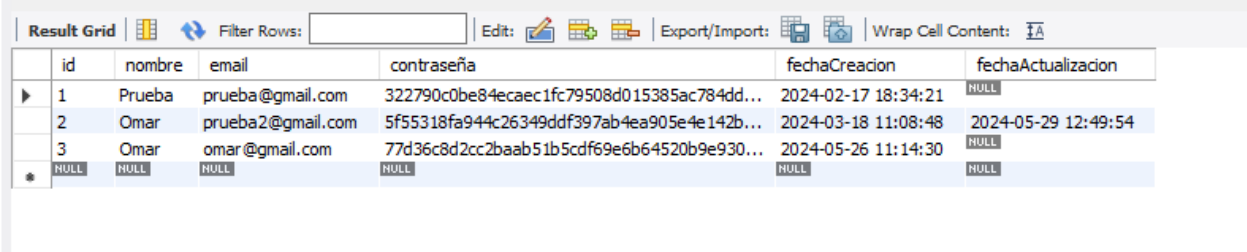
- **Tabla users**

La tabla *users* contiene información sobre los usuarios registrados en la aplicación. Sus columnas son:

- **id:** Identificador único del usuario.
- **nombre:** Nombre del usuario.
- **email:** Dirección de correo electrónico del usuario.
- **contraseña:** Contraseña del usuario, almacenada de manera cifrada.
- **fechaCreacion:** Fecha y hora en la que se creó el usuario.

- **fechaActualizacion:** Fecha y hora de la última actualización de los datos del usuario.

La estructura de la tabla *users* se vería tal como se muestra en la figura 21:



id	nombre	email	contraseña	fechaCreacion	fechaActualizacion
1	Prueba	prueba@gmail.com	322790c0be84ecaec1fc79508d015385ac784dd...	2024-02-17 18:34:21	NULL
2	Omar	prueba2@gmail.com	5f55318fa944c26349ddf397ab4ea905e4e142b...	2024-03-18 11:08:48	2024-05-29 12:49:54
3	Omar	omar@gmail.com	77d36c8d2cc2baab51b5cdf69e6b64520b9e930...	2024-05-26 11:14:30	NULL
NULL	NULL	NULL	NULL	NULL	NULL

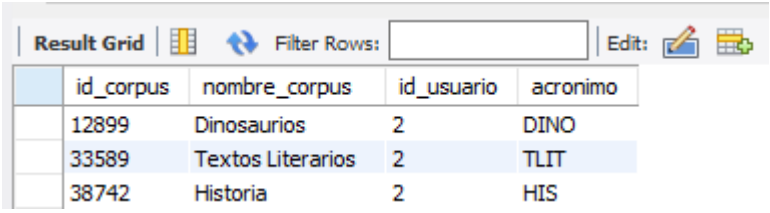
Figura 21: Tabla users con datos

- **Tabla corpus**

La tabla corpus contiene información sobre los corpus creados por los usuarios. Sus columnas son:

- **id_corpus:** Identificador único del corpus.
- **nombre_corpus:** Nombre del corpus.
- **id_usuario:** Identificador del usuario que creó el corpus, que actúa como una clave foránea referenciando la tabla *users*.
- **acronimo:** Acrónimo del corpus.

La estructura de la tabla corpus se vería tal como se muestra en la imagen:



id_corpus	nombre_corpus	id_usuario	acronimo
12899	Dinosaurios	2	DINO
33589	Textos Literarios	2	TLIT
38742	Historia	2	HIS

Figura 22: Tabla corpus con datos

En la figura 23 podemos ver como fue el código sql para crear las tablas.

```sql
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    contraseña VARCHAR(100) NOT NULL,
    fechaCreacion DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    fechaActualizacion DATETIME NULL ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE corpus (
    id_corpus INT AUTO_INCREMENT PRIMARY KEY,
    nombre_corpus VARCHAR(255) NOT NULL,
    id_usuario INT,
    acronimo VARCHAR(10),
    FOREIGN KEY (id_usuario) REFERENCES users(id)
);
```

Figura 23: Código sql de las tablas

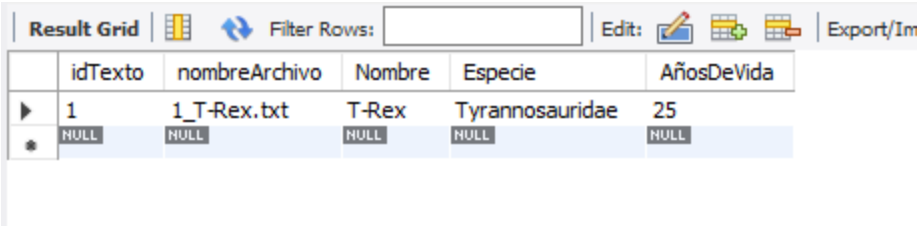
Además del schema principal CorpusTip, se crean schemas dinámicamente para cada corpus que cree un usuario. El nombre del schema es un número de 5 dígitos generado aleatoriamente seguido de un guion y el nombre del corpus, como por ejemplo “61614-Dinosaurios”

Cada schema contiene siempre tres tablas: textos, frases y palabras.

- **Tabla textos**

Esta tabla almacena la información de los archivos subidos al corpus y sus metadatos. Las columnas incluyen:

- **idTexto:** Identificador único del texto el cual es un numero incremental.
- **nombreArchivo:** Nombre del archivo junto con el identificador único(*idTexto*) del texto el cual sería el nombre que tenga el documento que suba el usuario y delante del nombre para asegurarnos de que no coincida con otro documento que se llame igual y suba el usuario pues se le añade el *idTexto* que empieza en 1 y seguido de guion bajo por ejemplo en el documento del T-Rex que vimos capitulo pasado será “1_T-Rex”.
- **Columnas de metadatos:** Incluyen tantas columnas como metadatos haya especificado el usuario en la fase de creación del corpus, cada una con el tipo de dato que haya elegido. Por ejemplo siguiendo con el ejemplo del corpus de dinosaurios el usuario puso 3 metadatos “Nombre”, “Especie” y “AñosDeVida” pues la tabla texto tendría también estas 3 tablas.



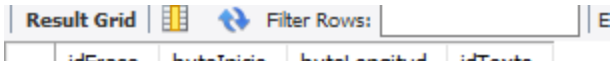
idTexto	nombreArchivo	Nombre	Especie	AñosDeVida
1	1_T-Rex.txt	T-Rex	Tyrannosauridae	25
NULL	NULL	NULL	NULL	NULL

Figura 24: Tabla textos con datos

- **Tabla frases**

Esta tabla almacena la información de las frases dentro de los textos. Las columnas incluyen:

- **idFrase:** Identificador único de la frase.
- **byteInicio:** Posición inicial de la frase en el archivo en bytes.
- **byteLongitud:** Longitud de la frase en bytes.
- **idTexto:** Identificador del texto al que pertenece la frase.



idFrase	byteInicio	byteLongitud	idTexto
1	0	143	1
2	145	176	1
3	323	155	1
4	480	150	1
5	632	48	1
6	682	99	1
7	783	176	1
8	961	84	2
9	1047	130	2
10	1179	114	2
11	1295	179	2
12	1476	41	2
13	1519	74	2
14	1595	23	2
15	1620	100	2
16	1722	35	2

Figura 25: Tabla frases con datos

- **Tabla palabras**

Esta tabla almacena la información de las palabras de cada frase. Las columnas incluyen:

- **idPalabra:** Identificador único de la palabra.
- **palabra:** La palabra en sí.
- **lema:** Lema de la palabra.
- **idCategoria:** Identificador de la categoría gramatical de la palabra. Para ella la tecnología que usamos para analizar el texto asigna una categoría a cada palabra para saber si es un vergo, adjetivo, etc. Se puede ver a que representa cada número en la figura 26.

```
public static Dictionary<string, int> CatGramaticalesAndLinguakitCodes = new Dictionary<string, int>()
{
    { "adjetivo", 1100 },
    { "adjetivo calificativo", 1101 },
    { "adjetivo posesivo", 1102 },
    { "adjetivo ordinario", 1103 },
    { "adverbio", 1200},
    { "adverbio general", 1201},
    { "adverbio negativo", 1202},
    { "artículo", 1700 },
    { "artículo determinado", 1701 },
    { "artículo indeterminado", 1702 },
    { "conjunción", 1500 },
    { "interjección", 1301},
    { "determinante", 2000},
    { "determinante artículo", 2001},
    { "determinante demostrativo", 2002},
    { "determinante indefinido", 2003},
    { "determinante posesivo", 2004},
    { "determinante interrogativo", 2005},
    { "determinante exclamativo", 2006},
    { "número", 4001},
    { "preposición", 1600},
    { "preposición latina", 1601},
    { "pronombre", 1400},
    { "pronombre demostrativo", 1401},
    { "pronombre exclamativo", 1402},
    { "pronombre indefinido", 1403},
    { "pronombre personal", 1404},
    { "pronombre relativo", 1405},
    { "pronombre interrogativo", 1406},
    { "sustantivo", 1000},
    { "sustantivo común", 1001},
    { "sustantivo propio", 1002},
    { "verbo", 3000}
};
```

Figura 26: Diccionario cat. Gramatical/identificador

- **idFrase:** Identificador de la frase a la que pertenece la palabra.
- **posicionPalabra:** Posición de la palabra dentro de la frase.

Result Grid | Filter Rows: | Edit: | Export

idPalabra	palabra	lema	idCategoria	idFrase	posicionPalabra
1	Había	haber	3000	1	1
2	una	uno	1100	1	2
3	vez	vez	1000	1	3
4	un	un	1702	1	4
5	conejito	conejito	1000	1	5
6	soñador	soñador	1000	1	6
7	que	que	1404	1	7
8	vivía	vivir	3000	1	8
9	en	en	1600	1	9
10	una	uno	1100	1	10
11	casita	casita	1000	1	11
12	en	en	1600	1	12
13	medio	medio	1013	1	13
14	del	del	1800	1	14
15	bosque	bosque	1000	1	15
16	rodeado	rodear	3000	1	16

Figura 27: Tabla palabras con datos

En las figuras 28, 29 y 30 podemos ver el código de creación de las 3 tablas, estas son iguales para todos los schemas.

```sql
CREATE TABLE `28457-Dinosaurios`.`textos` (
    idTexto INT AUTO_INCREMENT PRIMARY KEY,
    nombreArchivo VARCHAR(255) NOT NULL,
    Nombre VARCHAR(255) NOT NULL,
    Especie VARCHAR(255) NOT NULL,
    AñosDeVida INT NOT NULL
);
```

Figura 28: Código sql tabla textos

```sql
CREATE TABLE `28457-Dinosaurios`.`frases` (
    idFrase INT AUTO_INCREMENT PRIMARY KEY,
    byteInicio INT NOT NULL,
    byteLongitud INT NOT NULL,
    idTexto INT,
    FOREIGN KEY (idTexto) REFERENCES textos(idTexto)
);
```

Figura 29: Código sql tabla frases

```sql
CREATE TABLE `28457-Dinosaurios`.`palabras` (
    idPalabra INT AUTO_INCREMENT PRIMARY KEY,
    palabra VARCHAR(255) NOT NULL,
    lema VARCHAR(255) NOT NULL,
    idCategoria INT NOT NULL,
    idFrase INT,
    posicionPalabra INT NOT NULL,
    FOREIGN KEY (idFrase) REFERENCES frases(idFrase)
);
```

Figura 30: Código sql tabla palabras

4.5.3 Login/Registro de usuarios

En este apartado se explicará en detalle cómo se ha implementado el sistema de registro y login de usuarios. Para gestionar la autenticación y verificación de usuarios, se han creado varias funciones y una clase para el manejo de contraseñas.

Función VerificarUsuario

Esta función se utiliza cuando un usuario intenta iniciar sesión. Verifica si el correo electrónico y la contraseña proporcionados coinciden con un registro en la base de datos.

```
1 reference
public bool VerificarUsuario(string correoElectronico, string contraseña)
{

    conexion miConexion = new conexion();

    miConexion.AbrirConecxion();

    string consulta = "SELECT COUNT(*) FROM users WHERE email = @correoElectronico AND contraseña = @contraseña";

    MySqlCommand comando = new MySqlCommand(consulta, miConexion.conectar);
    comando.Parameters.AddWithValue("@correoElectronico", correoElectronico);
    comando.Parameters.AddWithValue("@contraseña", contraseña);

    int count = Convert.ToInt32(comando.ExecuteScalar());

    miConexion.CerrarConexion();

    return count > 0;
}
```

Figura 31: Función VerificarUsuario

Una vez que la función VerificarUsuario devuelve true, es decir, que el usuario es quien dice ser, se crea una cookie en el navegador que guarda su autenticación. Esta cookie se utiliza para mantener al usuario autenticado durante el tiempo que navegue en la web, hasta que cierre sesión o el navegador

```
FormsAuthentication.RedirectFromLoginPage(correoUsuario, false);
```

Figura 32: Cookie de autenticación de usuario

Función RegistrarUsuario

Esta función se utiliza para registrar un nuevo usuario. Inserta el nombre, correo electrónico y la contraseña del usuario en la base de datos.

```
1 reference
public bool RegistrarUsuario(string nombreUsuario, string correoElectronico, string contraseña)
{

    conexion miConexion = new conexion();

    miConexion.AbrirConecxion();

    string consulta = "INSERT INTO users (nombre, email, contraseña) VALUES (@nombre, @correoElectronico, @contraseña)";

    MySqlCommand comando = new MySqlCommand(consulta, miConexion.conectar);
    comando.Parameters.AddWithValue("@nombre", nombreUsuario);
    comando.Parameters.AddWithValue("@correoElectronico", correoElectronico);
    comando.Parameters.AddWithValue("@contraseña", contraseña);

    try
    {
        comando.ExecuteNonQuery();
        System.Diagnostics.Debug.WriteLine("Usuario creado correctamente.");
        return true;
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine("Error al crear el usuario: " + ex.Message);
        return false;
    }
    finally
    {
        miConexion.CerrarConexion();
    }
}
```

Figura 33: Función RegistrarUsuario

Función ExisteCorreo

Esta función se utiliza para verificar si un correo electrónico ya está registrado en la base de datos. Es útil durante el registro para evitar duplicados.

```
1 reference
public bool ExisteCorreo(string correoElectronico)
{
    conexion miConexion = new conexion();

    try
    {
        miConexion.AbrirConecxion();

        string consulta = "SELECT COUNT(*) FROM users WHERE email = @correoElectronico";
        MySqlCommand comando = new MySqlCommand(consulta, miConexion.conectar);
        comando.Parameters.AddWithValue("@correoElectronico", correoElectronico);

        int count = Convert.ToInt32(comando.ExecuteScalar());

        return count > 0;
    }
    finally
    {
        miConexion.CerrarConexion();
    }
}
```

Figura 34: Función ExisteCorreo

Clase PasswordHasher

Este método recibe una contraseña en texto plano y devuelve su hash utilizando SHA256.

```
3 references
public class PasswordHasher
{
    3 references
    public static string HashPassword(string password)
    {
        using (SHA256 sha256Hash = SHA256.Create())
        {
            // Generar el hash
            byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(password));

            // Convertir los bytes a una cadena hexadecimal
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < bytes.Length; i++)
            {
                builder.Append(bytes[i].ToString("x2"));
            }
            return builder.ToString();
        }
    }

    1 reference
    public static bool VerificarContraseña(string contraseñaPlana, string hashAlmacenado)
    {
        // Obtener el hash de la contraseña en texto plano
        string hashEntrante = HashPassword(contraseñaPlana);

        // Comparar el hash entrante con el hash almacenado
        return hashEntrante == hashAlmacenado;
    }
}
```

Figura 35: Clase PasswordHassher

4.5.4 HomePage

La página principal de la aplicación web (Home Page) está diseñada para proporcionar información general sobre el sitio como son los links a “About Us” and “Help”. Sin embargo, hay

algunos aspectos relevantes en cuanto a la funcionalidad dinámica basada en el estado de autenticación del usuario.

Visibilidad del Link a Corpus

El link a "Corpus" solo aparece cuando el usuario está autenticado. Esto se maneja en el código del Page_Load de la siguiente manera

```
protected void Page_Load(object sender, EventArgs e)
{
    if (HttpContext.Current.User.Identity.IsAuthenticated)
    {
        mostrarMisCorpus = true;
        userLink.HRef = "#";
        userLink.Attributes.Add("onclick", "toggleUserMenu()");
    }
    else
    {
        mostrarMisCorpus = false;
        userLink.HRef = "/Login.aspx";
        userMenu.Visible = false;
    }

    phMisCorpus.Visible = mostrarMisCorpus;
    nombreUsuario.Visible = mostrarMisCorpus;
}
```

Figura 36: Manejo visibilidad link mis corpus

```
<asp:PlaceHolder ID="phMisCorpus" runat="server">
    <li>
        <a class="nav-link" href="/MisCorpus.aspx">Corpus</a>
    </li>
</asp:PlaceHolder>
```

Figura 37: Html del link corpus

Menú del Icono de Usuario

El icono de usuario tiene una funcionalidad dual.

- **Usuario No Autenticado:** Actúa como un enlace a la página de inicio de sesión (Login.aspx).
- **Usuario Autenticado:** Se convierte en un menú desplegable que permite al usuario acceder a su perfil o cerrar sesión.

```
<li class="nav-item">
    <a class="nav-link" id="userLink" href="/Login.aspx" runat="server">
        <svg xmlns="http://www.w3.org/2000/svg" width="30" height="30" fill="currentColor" class="bi bi-person" viewBox="0 0 16 16">
          <path d="M8 8a3 3 0 1 0 0-6 3 3 0 0 0 0 6m2-3a2 2 0 1 1-4 0 2 2 0 0 1 4 0m4 8c0 1-1 1-1 1H3s-1 0-1-1 1-4 6-4 6 3 6 4m-1-.004c-.001-.246-.154-
        </svg>
        <asp:PlaceHolder ID="nombreUsuario" runat="server">
            <div><%= ObtenerNombreUsuario(HttpContext.Current.User.Identity.Name) %></div>
        </asp:PlaceHolder>
    </a>
    <asp:Panel ID="userMenu" runat="server" CssClass="dropdown-menu">
        <a class="dropdown-item" href="/Profile.aspx">Profile</a>
        <asp:Button ID="btnCerrarSesion" runat="server" Text="Cerrar Sesión" CssClass="dropdown-item" OnClick="btnCerrarSesion_Click" />
    </asp:Panel>
</li>
```

Figura 38: Html del icono de usuario

4.5.5 Corpus

En la página de Corpus, se muestran los corpus pertenecientes al usuario que está autenticado. La visualización de los corpus se realiza de forma dinámica, es decir, se muestra la cantidad exacta de corpus que tiene el usuario en la base de datos. Si tiene cinco corpus, se mostrarán cinco; si tiene tres, se mostrarán tres, y así sucesivamente.

En la figura 39 se muestra un fragmento de código HTML donde utiliza un control “Repeater” para mostrar cada corpus de manera dinámica.

```
<asp:Repeater ID="rptCorpus" runat="server" OnItemCommand="rptCorpus_ItemCommand">
    <ItemTemplate>
        <div class="each-corpus">
            <asp:Label ID="lblNombreCorpus" runat="server" Text='<%# Eval("Id") + "-" + Eval("Nombre") %>'></asp:Label>
            <div class="svg-container">
                <asp:Literal ID="LiteralUploadSVG" runat="server" Text='<%# GetUploadSVG() %>'></asp:Literal>
                <asp:HyperLink style="text-decoration: none;" ID="lnkEditarCorpus" runat="server" NavigateUrl='<%# string.Format("~/Bu
                <asp:Literal ID="LiteralEditSVG" runat="server" Text='<%# GetEditSVG() %>'></asp:Literal>
                <asp:LinkButton ID="lnkEliminarCorpus" runat="server" Text='<%# GetDeleteSVG(Eval("Id"), Eval("Nombre")) %>' CommandNa
            </div>
        </div>
    </ItemTemplate>
</asp:Repeater>
```

Figura 39: html listado de corpus

En la figura 40 se puede ver el siguiente método en el code-behind que se encarga de obtener los corpus del usuario autenticado desde la base de datos y enlazarlos al “Repeater”

```
protected void CargarCorpus(int idUsuario)
{
    User usuario = new User();
    var corpus = usuario.ObtenerCorpus(idUsuario);
    rptCorpus.DataSource = corpus;
    rptCorpus.DataBind();
}
```

Figura 40: Función CargarCorpus

El botón de la papelera tiene una lógica más compleja detrás para eliminar un corpus cuya lógica está relacionada con la creación de corpus y por qué se tiene que eliminar así se explicará en el apartado de creación del corpus. A continuación, en la figura 41, se muestra cómo se maneja el evento de comando del “Repeater” para eliminar un corpus.

```
protected void rptCorpus_ItemCommand(object source, RepeaterCommandEventArgs e)
{
    if (e.CommandName == "EliminarCorpus")
    {
        // Obtener el argumento de comando que contiene el ID y el nombre del corpus
        string[] partes = e.CommandArgument.ToString().Split('-');
        string idCorpusString = partes[0];
        string nombreCorpus = partes[1];
        int idCorpus = int.Parse(idCorpusString);


        // Eliminar la carpeta correspondiente en el directorio App_Data
        string nombreCarpeta = $"{idCorpus}-{nombreCorpus}";
        string rutaCarpeta = Server.MapPath("~/App_Data/" + nombreCarpeta);

        if (Directory.Exists(rutaCarpeta))
        {
            Directory.Delete(rutaCarpeta, true);
            System.Diagnostics.Debug.WriteLine("La carpeta del corpus ha sido eliminada correctamente.");
        }
        else
        {
            System.Diagnostics.Debug.WriteLine("La carpeta del corpus no existe en el directorio App_Data.");
        }

        // Lógica para eliminar el corpus
        User miUsuario = new User();
        miUsuario.EliminarCorpus(idCorpus, nombreCorpus);

        int userId = Convert.ToInt32(Session["UserID"]);
        CargarCorpus(userId);
        UpdatePanelCorpus.Update();
    }
}
```

Figura 41: Lógica eliminar un corpus

4.5.6 Creación de un Corpus

En la fase de creación de corpus, el proceso se divide en tres etapas distintas como hemos comentado ya anteriormente. Primero, el usuario debe proporcionar un nombre y un acrónimo para el corpus. Luego, en la segunda fase, el usuario elige los metadatos que se asociarán con los archivos del corpus. Estos metadatos se crean dinámicamente, permitiendo al usuario añadir nuevos metadatos utilizando un botón que los enlaza a un “repeater”, o eliminar metadatos de la lista si es necesario.

```
protected void btnAñadir_Click(object sender, EventArgs e)
{
    Metadato nuevoMetadato = new Metadato();

    if (string.IsNullOrEmpty(nuevoMetadato.Tipo))
    {
        nuevoMetadato.Tipo = "Texto";
    }

    metadatos.Add(nuevoMetadato);

    // Vincular la lista actualizada al Repeater
    RepeaterMetadatos.DataSource = metadatos;
    RepeaterMetadatos.DataBind();
    UpdatePanelMetadatos.Update();
}
```

Figura 42: Botón añadir metadato dinámico

```
protected void lnkEliminarMetadato_Click(object sender, EventArgs e)
{
    Button lnkEliminar = (Button)sender;
    int index = Convert.ToInt32(lnkEliminar.CommandArgument);
    System.Diagnostics.Debug.WriteLine(index);
    if (index >= 0 && index < metadatos.Count)
    {
        metadatos.RemoveAt(index);
        // Vincular la lista actualizada al Repeater
        RepeaterMetadatos.DataSource = metadatos;
        RepeaterMetadatos.DataBind();
        UpdatePanelMetadatos.Update();
    }
}
```

Figura 43: Botón eliminar metadato dinámico

Cuando el usuario ha definido todos los metadatos y ha completado todos los campos, al hacer clic en continuar, se ejecuta una serie de pasos importantes. Primero, se validan los datos ingresados para asegurarse de que todos los campos estén completos. Luego, se recuperan de variables de sesión el título y acrónimo del corpus, así como el ID del usuario.

A continuación, se inserta el nuevo corpus en la base de datos utilizando la función InsertarNuevoCorpus. Esta función se encarga de generar un ID único para el corpus y almacenarlo en la tabla principal de corpus. Posteriormente, se crea un esquema de base de datos específico para este corpus.

```
public int InsertarNuevoCorpus(string tituloCorpus, int idUsuario, string acronimoCorpus)
{
    int nuevoCorpuId = -1;
    conexion miConexion = new conexion();

    try
    {
        miConexion.AbrirConecxion();

        Random rnd = new Random();
        int idCorpus = rnd.Next(10000, 99999);

        // Verificar si el idCorpus ya existe en la tabla
        string consultaVerificar = "SELECT COUNT(*) FROM corpus WHERE id_corpus = @idCorpus";
        MySqlCommand comandoVerificar = new MySqlCommand(consultaVerificar, miConexion.conectar);
        comandoVerificar.Parameters.AddWithValue("@idCorpus", idCorpus);
        int count = Convert.ToInt32(comandoVerificar.ExecuteScalar());

        // Mientras el idCorpus generado aleatoriamente esté en uso, genera uno nuevo
        while (count > 0)
        {
            idCorpus = rnd.Next(10000, 99999);

            // Verificar nuevamente si el nuevo idCorpus está en uso
            comandoVerificar.Parameters["@idCorpus"].Value = idCorpus;
            count = Convert.ToInt32(comandoVerificar.ExecuteScalar());
        }

        string consulta = "INSERT INTO corpus (id_corpus, nombre_corpus, acronimo, id_usuario) VALUES (@idCorpus, @tituloCorpus, @acronimoCorpus, @idUsuario); SELECT LAST_INSERT_ID();";
        MySqlCommand comando = new MySqlCommand(consulta, miConexion.conectar);
        comando.Parameters.AddWithValue("@idCorpus", idCorpus);
        comando.Parameters.AddWithValue("@tituloCorpus", tituloCorpus);
        comando.Parameters.AddWithValue("@acronimoCorpus", acronimoCorpus);
        comando.Parameters.AddWithValue("@idUsuario", idUsuario);

        comando.ExecuteNonQuery();
        nuevoCorpuId = idCorpus;
```

Figura 44: Función InsertarNuevoCorpus

La función CrearSchema recibe el nombre del esquema y la lista de metadatos, generando dinámicamente la tabla Textos con las columnas correspondientes a los metadatos definidos por el usuario.

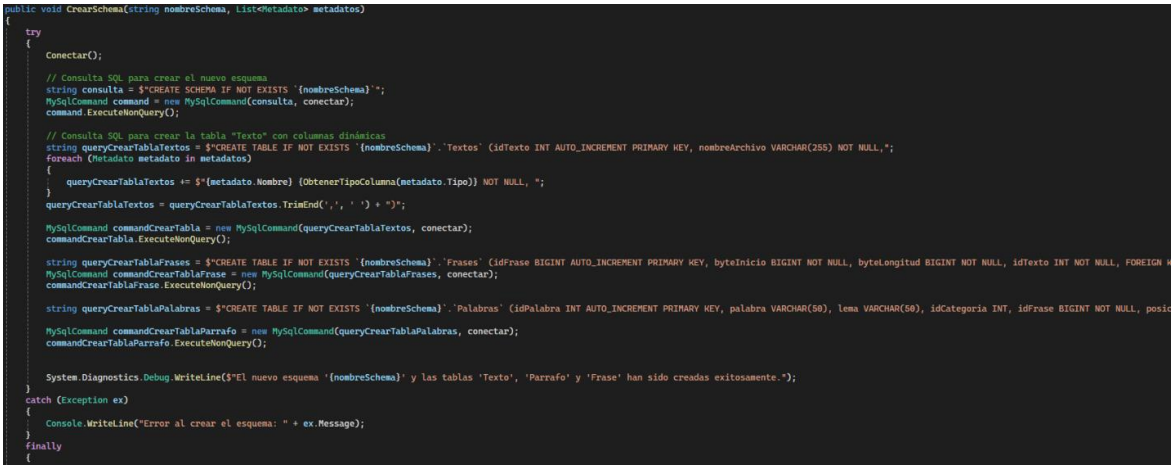


Figura 45: Función CrearSchema

La función “ObtenerTipoColumna” se utiliza para traducir los tipos de datos definidos por el usuario a tipos de datos que MySQL puede entender es muy importante para que funcione todo correctamente.

```
private string ObtenerTipoColumna(string tipo)
{
    // Convierte el tipo seleccionado por el usuario al tipo de columna correspondiente en MySQL
    switch (tipo)
    {
        case "Texto":
            return "VARCHAR(255)";
        case "Numero":
            return "INT";
        case "Fecha":
            return "DATE";
        default:
            return "VARCHAR(255)";
    }
}
```

Figura 46: Función ObtenerTipoColumna

Además de crear el esquema en la base de datos, también es necesario crear un directorio en la carpeta “App_Data” para almacenar archivos específicos del corpus. Este directorio se nombra de acuerdo con el esquema del corpus y contiene dos archivos de texto por defecto: frases.txt y parrafos.txt. Estos archivos se utilizan para almacenar todas las frases y párrafos de cada archivo que el usuario sube al corpus.

```
protected void btnSiguiente_Click(object sender, EventArgs e)
{
    bool camposValidos = ValidarCampos();

    if (camposValidos)
    {
        // Creacion del corpus
        string tituloCorpus = (string)Session["TituloCorpus"];
        string acronimoCorpus = (string)Session["AcronimoCorpus"];
        int userId = Convert.ToInt32(Session["UserID"]);
        User miUsuario = new User();
        int nuevoCorpusId = miUsuario.InsertarNuevoCorpus(tituloCorpus, userId, acronimoCorpus);

        Session["NuevoCorpusId"] = nuevoCorpusId;

        string nombreSchema = nuevoCorpusId + "-" + tituloCorpus;
        // Creacion el nuevo schema
        ControlSchema controlSchema = new ControlSchema();
        controlSchema.CrearSchema(nombreSchema, metadatos);

        // Creacion del directorio
        string rutaDirectorio = Server.MapPath("~/App_Data/" + nombreSchema);
        Directory.CreateDirectory(rutaDirectorio);

        string rutaArchivoFrases = Path.Combine(rutaDirectorio, "frases.txt");
        string rutaArchivoParrafos = Path.Combine(rutaDirectorio, "parrafos.txt");

        File.WriteAllText(rutaArchivoFrases, ""); // Crear archivo frases.txt

        File.WriteAllText(rutaArchivoParrafos, ""); // Crear archivo parrafos.txt

        Response.Redirect("~/CrearCorpus3.aspx");
    }
}
```

Figura 47: Botón para pasar a la fase 3 de creación de corpus

En resumen, este proceso asegura que cada nuevo corpus tenga su propio esquema de base de datos y directorio de almacenamiento, permitiendo una gestión estructurada y dinámica de los datos asociados a cada corpus creado por los usuarios.

Para hablar sobre los ficheros frases.txt y parrafos.txt, es esencial comprender su estructura y cómo se utilizan en el sistema.

Fichero frases.txt

El archivo frases.txt contiene información detallada sobre cada frase en el corpus. La estructura de este archivo incluye un número incremental que comienza en 1, seguido de la frase, el byte de inicio del párrafo al que pertenece y finalmente el byte de longitud total del párrafo. Por ejemplo, en la imagen proporcionada, las tres primeras frases pertenecen al mismo párrafo, lo cual se puede verificar al observar que el byte de inicio es 0 y el byte de longitud del párrafo es 454 para las tres frases. Esta organización permite descomponer los textos en frases individuales mientras se mantiene la referencia a sus párrafos de origen.

```
1       Había una vez un conejito soñador que vivía en una casita en medio del bosque, rodeado de libros y fantasía, pero no tenía amigos.      0       454
2       Todos le habían dado de lado porque se pasaba el día contando historias imaginarias sobre hazañas caballerescas, aventuras submarinas y expediciones extraterrestres.   0       454
3       Siempre estaba inventando aventuras como si las hubiera vivido de verdad, hasta que sus amigos se cansaron de escucharle y acabó quedándose solo.       0       454
4       Al principio el conejito se sintió muy triste y empezó a pensar que sus historias eran muy aburridas y por eso nadie las quería escuchar.       456     181
5       Pero pese a eso continuó escribiendo.   456     181
6       Las historias del conejito eran increíbles y le permitían vivir todo tipo de aventuras. 639     258
7       Se imaginaba vestido de caballero salvando a inocentes princesas o sintiendo el frío del mar sobre su traje de buzo mientras exploraba las profundidades del océano.    639     258
8       Había una vez una dulce niña que quería mucho a su madre y a su abuela. 897     301
9       Les ayudaba en todo lo que podía y como era tan buena el día de su cumpleaños su abuela le regaló una caperuza roja.    897     301
10      Como le gustaba tanto e iba con ella a todas partes, pronto todos empezaron a llamarla Caperucita roja. 897     301
11      Un día la abuela de Caperucita, que vivía en el bosque, enfermó y la madre de Caperucita le pidió que le llevara una cesta con una torta y un tarro de mantequilla.     1200    199
12      Caperucita aceptó encantada.    1200    199
13      Ten mucho cuidado Caperucita, y no te entretengas en el bosque. 1401    65
14      ¡Sí mamá!       1468    14
15      La niña caminaba tranquilamente por el bosque cuando el lobo la vio y se acercó a ella. 1484    91
16      ¿Dónde vas Caperucita?  1577    26
17      A casa de mi abuelita a llevarle esta cesta con una torta y mantequilla.        1605    74
18      Yo también quería ir a verla…. así que, ¿por qué no hacemos una carrera?        1681    139
19      Tú ve por ese camino de aquí que yo iré por este otro.  1681    139
```

Figura 48: Vista del fichero frases.txt

Fichero parrafos.txt

El archivo parrafos.txt contiene los párrafos completos del texto, separados entre ellos por el signo "§". Este archivo se utiliza en el buscador para proporcionar más contexto al usuario, permitiéndole identificar a qué zona del texto pertenece una frase específica. En la imagen proporcionada, se puede ver cómo los párrafos están claramente delimitados por el signo "§", facilitando así la búsqueda, recuperación de fragmentos textuales con mayor contexto y evita que se muestre información del contexto que no corresponde a la búsqueda.

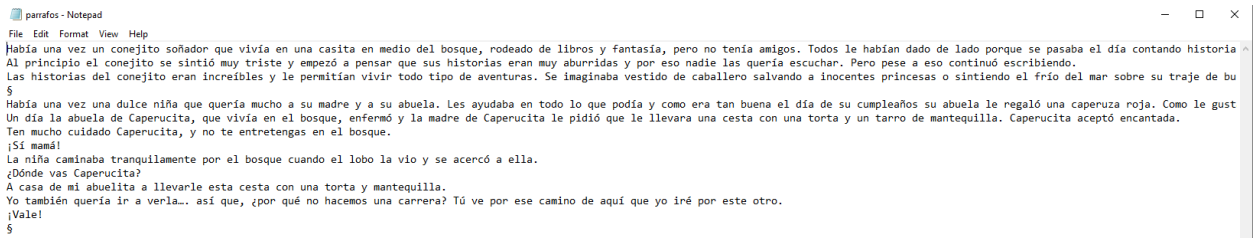


Figura 49: Vista del fichero párrafos.txt

En resumen, estos archivos no solo almacenan las frases y párrafos, sino que también proporcionan una estructura que permite una navegación eficiente y contextualizada del contenido, esencial para las funciones de búsqueda y análisis del corpus.

En la última fase, los usuarios pueden subir los archivos uno a uno, asociándolos con los metadatos que introdujeron en la fase anterior. Aquí se detallan los pasos y el código relacionado con esta fase:

Selección de Archivo y Metadatos

Primero, el usuario debe seleccionar un archivo desde su dispositivo utilizando el componente "*asp:FileUpload*", que es una etiqueta de ASP.NET diseñada para permitir la selección de archivos.

```
<asp:FileUpload ID="FileUpload1" runat="server" />
```

Figura 50: Componente FielUpload asp.net

Además de seleccionar el archivo, el usuario debe rellenar los metadatos correspondientes para dicho archivo

Validación y Procesamiento del Archivo

Una vez que el usuario ha seleccionado el archivo y ha proporcionado los metadatos, al hacer clic en "Subir Archivo", se llama a la función "*btnSubirArchivo_Click*". Esta función se encarga de validar que todos los campos estén correctamente rellenados. Si la validación es exitosa, se llama a *ProcesadorArchivo*, que es el encargado de procesar y analizar el archivo y realiza varios pasos cruciales:

- **Obtención de Nombre y Ruta del Corpus:** obtiene el nombre completo del corpus utilizando la función "*ObtenerNombreCorpus*" y además define la ruta donde se almacenarán los archivos del corpus dentro de "*App_Data*"
- **Generación de ID para el Archivo y guardado:** Genera un ID único para el archivo que se está subiendo, combinando este ID con el nombre original del archivo. Por ejemplo, si es el primer archivo y se llama "T-Rex.txt", el nombre resultante será "1_T-Rex.txt". Y luego este lo guarda en la ruta especificada
- **Procesamiento del Texto:** Crea una instancia de la clase "*ProcesadorTextos*" y llama a sus métodos para analizar el texto, dividiéndolo en palabras y párrafos, y para rellenar la base de datos.

```
private void ProcesadorArchivo()
{
    string nombreCarpetaCorpus = ObtenerNombreCorpus();

    string rutaCarpetaCorpus = Server.MapPath("~/App_Data/" + nombreCarpetaCorpus);

    int nuevoIdArchivo = ObtenerProximoIdArchivo();

    string nombreArchivo = Path.GetFileName(FileUpload1.FileName);

    string nombreNuevoArchivo = $"{nuevoIdArchivo}_{nombreArchivo}";

    FileUpload1.SaveAs(Path.Combine(rutaCarpetaCorpus, nombreNuevoArchivo));
    Session["NombreUltimoArchivo"] = nombreNuevoArchivo;


    ProcesadorTextos procesar = new ProcesadorTextos(nombreCarpetaCorpus, nombreNuevoArchivo, actualizarMetadatos());

    string contenidoArchivo = "";
    using (StreamReader sr = new StreamReader(rutaCarpetaCorpus + "/" + nombreNuevoArchivo))
    {
        // Leer todo el contenido del archivo y almacenarlo en la variable
        contenidoArchivo = sr.ReadToEnd();
    }

    procesar.almacenar(contenidoArchivo);
    lblMensajeError.Text = "Es correcto";
}
```

Figura 51: Función ProcesadorArchivo

Clase ProcesadorTextos

La función almacenar se beneficia de una inteligencia desarrollada por el profesor que descompone el texto en niveles jerárquicos: párrafos, frases y palabras. Esta inteligencia se implementa a través de una serie de clases especializadas que permiten un procesamiento estructurado y detallado del texto.

La clase *Paragraph* se encarga de identificar y extraer los párrafos del texto completo. Cada objeto de esta clase representa un párrafo y contiene métodos para manipular y acceder a su contenido. Uno de los métodos clave de esta clase es *GetSentences*, que permite dividir un párrafo en frases. Este método devuelve una lista de objetos de la clase *InfoUnaFrase*, cada uno de los cuales representa una frase dentro del párrafo.

La clase *InfoUnaFrase* representa una frase individual dentro de un párrafo. Esta clase contiene propiedades y métodos para acceder a las palabras que componen la frase. Uno de los métodos importantes de *InfoUnaFrase* es *GetWords*, que divide una frase en palabras. Este método devuelve una lista de objetos de la clase *InfoUnaPalabra*.

La clase *InfoUnaPalabra* representa una palabra individual dentro de una frase y proporciona información detallada sobre cada palabra, como su lema (la forma básica de la palabra), forma canónica (la forma estándar), y otras características gramaticales.

En cuanto al funcionamiento detallado de la función almacenar, primero utiliza la clase *Paragraph* para dividir el texto en párrafos mediante el método *GetParagraphs*, que devuelve una lista de objetos *Paragraph*. A continuación, cada objeto *Paragraph* utiliza el método *GetSentences* para obtener las frases contenidas en él, las cuales se representan como objetos *InfoUnaFrase*. Finalmente, los objetos *InfoUnaFrase* utilizan el método *GetWords* para dividir la frase en palabras, y cada palabra se representa como un objeto *InfoUnaPalabra*, que incluye detalles como el lema y la forma canónica.

En resumen, la función almacenar utiliza una jerarquía de clases (*Paragraph*, *InfoUnaFrase*, *InfoUnaPalabra*) proporcionadas por la inteligencia del profesor para descomponer y analizar el texto de manera estructurada. Cada nivel de esta jerarquía (párrafos, frases y palabras) se procesa secuencialmente, lo que permite obtener información detallada y organizada del texto, facilitando así su almacenamiento y análisis en la base de datos del proyecto.

```
public void almacenar(string contenidoArchivo)
{
    try
    {
        Text document = new ProcesarTextos.Text("", contenidoArchivo);

        int idTexto = controlSchema.InsertarTexto(this.nombreSchema, this.nombreArchivo, this.metadatos);
        long offsetBytes = (new FileInfo(rutaFicheroIndiceParrafos)).Length;

        foreach (Paragraph paragraph in document.GetParagraphs())
        {
            long tamBytesParrafo = Encoding.UTF8.GetByteCount(paragraph.getText() + "\r\n");

            List<InfoUnaFrase> frases = new List<InfoUnaFrase>();

            foreach (Sentence sentence in paragraph.GetSentences())
            {
                InfoUnaFrase frase = new InfoUnaFrase();
                frase.Frase = sentence.getText();
                frases.Add( frase );
            }

            List<InfoUnaFrase> frasesReconocidas = servicioLematizacion.ReconocerFrases(frases, "es", false);

            foreach (InfoUnaFrase frase in frasesReconocidas)
            {
                string registroFrase = (++this.ultimoIdFrase) + "\t" + frase.Frase + "\t" + offsetBytes.ToString() + "\t" + tamBytesParrafo;
                long numBytesFrase = Encoding.UTF8.GetByteCount(registroFrase);

                long offsetBytesFrase = (new FileInfo(rutaFicheroIndiceFrases)).Length;
```

```
                controlSchema.InsertarFrase(this.nombreSchema, offsetBytesFrase, numBytesFrase, idTexto);

                ficheroIndiceFrases.WriteLine(registroFrase);
                ficheroIndiceFrases.Flush();

                int posPalabra = 0;
                List<InfoUnaPalabra> palabras = frase.Palabras;
                foreach(InfoUnaPalabra palabra in palabras)
                {
                    //Los signos de puntuación y caracteres especiales no cuentan
                    if (palabra.PosMark != "") continue;
                    int idCategoria = palabra.IdCategoria;
                    string formaCanonica = palabra.FormaCanonica;
                    if (formaCanonica.Contains("_")) formaCanonica = palabra.Palabra.ToLower();

                    posPalabra++;
                    controlSchema.InsertarPalabra(this.nombreSchema, palabra.Palabra, formaCanonica, idCategoria, this.ultimoIdFrase, posPalabra);
                }
            }
            offsetBytes = offsetBytes + tamBytesParrafo + 2;
            ficheroIndiceParrafos.WriteLine(paragraph.getText());
            ficheroIndiceParrafos.Flush();
        }
        string delimitadorArticulos = "§";
        ficheroIndiceParrafos.WriteLine(delimitadorArticulos);
        ficheroIndiceParrafos.Flush();
    }
```

Figura 52: Función almacenar

4.5.7 Buscador de un Corpus

En el buscador de la aplicación, se pueden presentar tres casos distintos dependiendo de la forma en que el usuario realice la búsqueda. Estos casos son:

Búsqueda vacía: El usuario no introduce ningún criterio de búsqueda y presiona el botón de buscar.

Búsqueda sin orden: El usuario introduce criterios de búsqueda pero no especifica un orden para los resultados.

Búsqueda con orden: El usuario introduce criterios de búsqueda y especifica un orden para los resultados.

Para saber en que caso estamos hemos creado la clase *InformacionDocumentos* que es la encarga de recoger todos los datos introducidos por el usuario y así podemos saber que tipo de búsqueda hacer.

DatabaseBusquedaVacia

Para gestionar el caso de una búsqueda vacía, se creó la clase *DatabaseBusquedaVacia*. Esta clase se encarga de ejecutar una búsqueda en la base de datos cuando no se han proporcionado criterios específicos por parte del usuario. El método más importante en esta clase es *GetFrasesAndDocuments*, que realiza toda la lógica necesaria para obtener las frases y sus documentos correspondientes.

```
public List<CorpusFrase> GetFrasesAndDocuments()
{
    List<CorpusFrase> result;
    try
    {
        string query = CreateMySqlQuery(this.InputOptions);
        Debug.WriteLine(query);
        DBConnection.OpenConnection();
        MySqlCommand command = new MySqlCommand(query);
        MySqlDataReader reader = this.DBQuery.ExecuteQueryAndGetReader(command);
        MySqlDataReaderToCorpusTexualDataList(reader);
        result = this.CorpusFrases;
        return result;
    }
    catch (Exception)
    {
        this.CorpusFrases = new List<CorpusFrase>();
        result = this.CorpusFrases;
    }
    finally
    {
        DBConnection.CloseConnection();
    }
    return result;
}
```

Figura 53: Función GetFrasesAndDocuments

Dentro de *GetFrasesAndDocuments*, se llama al método *CreateMySqlQuery*, que forma una consulta SQL basada en las opciones de entrada proporcionadas.

```
private string CreateMySqlQuery(InputOptions inputOptions)
{
    string titulo = CreateTituloSQLStatement(inputOptions);


    string AllSQLConditions = titulo;
    if (!AllSQLConditions.Equals(string.Empty)) AllSQLConditions = "WHERE " + AllSQLConditions;

    return string.Format(QUERY_INPUT_VACIO, AllSQLConditions, CreateLimit());
}
```

Figura 54: Función CreateMySqlQuery

Este método se encarga de formar la consulta SQL de manera dinámica. Primero, crea la parte del título de la consulta utilizando CreateTituloSQLStatement. Luego, verifica si hay condiciones adicionales y las añade a la consulta SQL. Finalmente, forma la consulta completa utilizando una plantilla de consulta (QUERY_INPUT_VACIO) y añade cualquier límite necesario.

En resumen, DatabaseBusquedaVacia maneja la lógica para ejecutar una búsqueda vacía, obteniendo las frases y documentos necesarios de la base de datos. La utilización de métodos auxiliares como CreateMySqlQuery permite formar consultas SQL dinámicas y flexibles, adaptándose a las necesidades de la aplicación y asegurando que los resultados sean precisos y relevantes.

DatabasePatternSinOrden

Ahora vamos a explicar la clase *DatabasePatternSinOrden*. Al crear una instancia de esta clase, se le pasan los datos de búsqueda que ha introducido el usuario (*InputOptions*), el *InputSliceData* que define el rango de contexto deseado (más o menos texto alrededor de la frase encontrada), y el nombre del corpus en el que se está realizando la búsqueda.

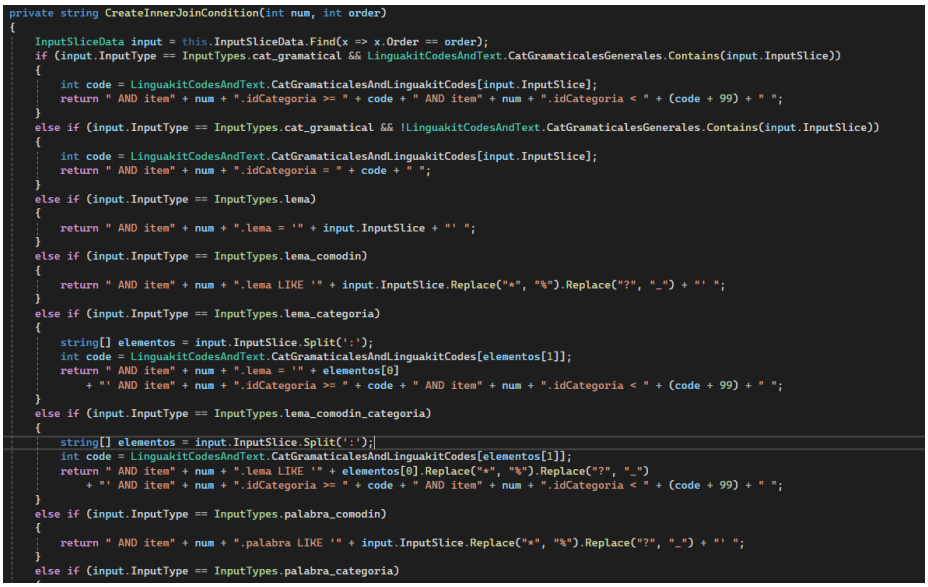
La clase se encarga de buscar todas las frases que coincidan con los criterios de búsqueda y las almacena en un atributo llamado *CorpusFrases*. Este proceso implica la creación y ejecución de una consulta SQL personalizada basada en los datos de entrada.

```
2 references
public DatabasePatternSinOrden(InputOptions InputOptions, List<InputSliceData> InputSliceData, string corpusName)
{
    this.corpusName = corpusName;
    this.QUERY_SELECT_ELEMENTO = $"SELECT idFrase, palabra, posicionPalabra FROM `{corpusName}`.palabras" + " AS item WHERE idFrase IN ({0}) AND {1}";
    this.TotalFrases = 0;
    this.CorpusFrases = new List<CorpusFrase>();
    this.InputOptions = InputOptions;
    this.InputSliceData = InputSliceData;
    this.DBConnection = new DatabaseConnection(corpusName);
    DBConnection.CreateConnection();
    this.DBQuery = new DatabaseQuery(this.DBConnection);
    string idFrases = GetAllIdFrases();
    if (idFrases != string.Empty) GetAllCorpusFrases(idFrases);


}
```

Figura 55: Constructor de la clase DatabasePatternSinOrden

Esta clase contiene varias funciones importantes, similares a las de la clase *DatabaseBusquedaVacia*, como *GetTotalFrases* y *CreateQueryForGetIdFrases*. Además, tiene una función crucial llamada *CreateInnerJoinCondition*, que se encarga de montar la consulta SQL y transformar los términos de búsqueda introducidos por el usuario en una forma comprensible por SQL. Esta función evalúa los diferentes tipos de entrada (como categorías gramaticales, lemas, palabras, etc.) y construye las condiciones de la consulta de manera dinámica para asegurar que los resultados sean precisos y relevantes según los criterios especificados por el usuario.



```
private string CreateInnerJoinCondition(int num, int order)
{
    InputSliceData input = this.InputSliceData.Find(x => x.Order == order);
    if (input.InputType == InputTypes.cat_gramatical && LinguakitCodesAndText.CatGramaticalesGenerales.Contains(input.InputSlice))
    {
        int code = LinguakitCodesAndText.CatGramaticalesAndLinguakitCodes[input.InputSlice];
        return " AND item" + num + ".idCategoria >= " + code + " AND item" + num + ".idCategoria < " + (code + 99) + " ";
    }
    else if (input.InputType == InputTypes.cat_gramatical && !LinguakitCodesAndText.CatGramaticalesGenerales.Contains(input.InputSlice))
    {
        int code = LinguakitCodesAndText.CatGramaticalesAndLinguakitCodes[input.InputSlice];
        return " AND item" + num + ".idCategoria = " + code + " ";
    }
    else if (input.InputType == InputTypes.lema)
    {
        return " AND item" + num + ".lema = '" + input.InputSlice + "' ";
    }
    else if (input.InputType == InputTypes.lema_comodin)
    {
        return " AND item" + num + ".lema LIKE '" + input.InputSlice.Replace("*", "%").Replace("?", "_") + "' ";
    }
    else if (input.InputType == InputTypes.lema_categoria)
    {
        string[] elementos = input.InputSlice.Split(':');
        int code = LinguakitCodesAndText.CatGramaticalesAndLinguakitCodes[elementos[1]];
        return " AND item" + num + ".lema = '" + elementos[0]
            + "' AND item" + num + ".idCategoria >= " + code + " AND item" + num + ".idCategoria < " + (code + 99) + " ";
    }
    else if (input.InputType == InputTypes.lema_comodin_categoria)
    {
        string[] elementos = input.InputSlice.Split(':');
        int code = LinguakitCodesAndText.CatGramaticalesAndLinguakitCodes[elementos[1]];
        return " AND item" + num + ".lema LIKE '" + elementos[0].Replace("*", "%").Replace("?", "_")
            + "' AND item" + num + ".idCategoria >= " + code + " AND item" + num + ".idCategoria < " + (code + 99) + " ";
    }
    else if (input.InputType == InputTypes.palabra_comodin)
    {
        return " AND item" + num + ".palabra LIKE '" + input.InputSlice.Replace("*", "%").Replace("?", "_") + "' ";
    }
    else if (input.InputType == InputTypes.palabra_categoria)
```

Figura 56: Función CreateInnerJoinCondition

DatabasePattern

Esta clase es fundamental para realizar búsquedas ordenadas en base a las entradas del usuario. Contiene un atributo *QUERY_*PATRON, que es una cadena de consulta SQL predefinida. Esta cadena contiene marcadores de posición {0}, {1}, etc., que se reemplazarán dinámicamente con fragmentos de SQL generados por otros métodos de la clase. Este enfoque permite construir consultas SQL complejas de manera flexible y dinámica.

```
private const string QUERY_PATRON = "SELECT filtro.idFrase, fr.byteInicio, fr.byteLongitud, txt.idTexto, txt.nombreArchivo, {0} " +
    "FROM (SELECT DISTINCT item1.idFrase FROM {1} {2} " +
    "INNER JOIN frases fr ON item1.idFrase = fr.idFrase " +
    "INNER JOIN textos txt ON fr.idTexto = txt.idTexto WHERE {3} {4}) AS filtro " +
    "INNER JOIN frases fr ON filtro.idFrase = fr.idFrase " +
    "INNER JOIN textos txt ON fr.idTexto = txt.idTexto {5}";
```

Figura 57: Atributo QUERY_PATRON

Además el constructor es parecido al *DatabasePatterSinOrden* ya que recibe los mismos 3 parámetros, el *inputOptions*, *inputSliceData* y el *corpusName*. Dentro del constructor, se

inicializan varios atributos y se llama al método *GetCorpusTextualData*, que es el encargado de ejecutar la consulta y almacenar los resultados.

```
public DatabasePattern(InputOptions InputOptions, List<InputSliceData> InputSliceData, string corpusName)
{
    this.corpusName = corpusName;
    this.NumTotalApariciones = 0;
    this.TotalFrases = 0;
    this.DBConnection = new DatabaseConnection(corpusName);
    DBConnection.CreateConnection();
    this.DBQuery = new DatabaseQuery(this.DBConnection);
    this.InputOptions = InputOptions;
    this.InputSliceData = InputSliceData;
    GetCorpusTextualData();
}
```

Figura 58: Constructor DatabasePattern

Sus métodos más destacables son *GetCorpusTextualData* el cual se encarga de crear y ejecutar la consulta SQL. Primero, construye la consulta llamando al método *CreateMysqlQuery*. Luego, ejecuta la consulta y transforma los resultados en datos que pueden ser utilizados por la aplicación.

```
private void GetCorpusTextualData()
{
    string query = CreateMysqlQuery();
    Debug.WriteLine("Query: " + (object)query);
    DBConnection.OpenConnection();
    MySqlCommand command = new MySqlCommand(query);
    MySqlDataReader reader = this.DBQuery.ExecuteQueryAndGetReader(command);
    if (reader == null) this.MySqlError = true;
    else
    {
        this.TransformMySqlReaderToData(reader, this.InputOptions.NumeroElementos);
        reader.Close();
        TotalFrases = -1;
        this.queryTotal = "SELECT Count(DISTINCT(item1.idFrase)) " + query.Substring(query.IndexOf("item1.idFrase") + 13, query.IndexOf("LIMIT") - query.I
    }
    DBConnection.CloseConnection();
}
```

Figura 59: Función GetCorpusTextualData

Y por otro lado tenemos a *CreateMysqlQuery*, este método construye la consulta SQL dinámica utilizando la plantilla QUERY_PATRON. Llama a varios métodos auxiliares como *CreateColumns*, *CreateFromForFiltro*, *CreateAllInnerJoins*, *CreateWhereOfFiltro* y *CreateLimit*, que generan las diferentes partes de la consulta SQL.

```
private string CreateMysqlQuery()
{
    this.numItem = 1;
    return string.Format(QUERY_PATRON,
        CreateColumns(),
        CreateFromForFiltro(),
        CreateAllInnerJoins(2),
        CreateWhereOfFiltro(),
        CreateLimit(),
        CreateAllInnerJoins(1),
        CreateFrasesAndDocsOptions());
}
```

Figura 60: Función CreateMysqlQuery

Con esto podemos llegar a la conclusión de que la principal diferencia entre *DatabasePattern* y *DatabasePatternSinOrden* radica en cómo se maneja el orden de los datos de entrada. Mientras que *DatabasePattern* respeta y utiliza el orden de los datos ingresados por el usuario para construir la consulta SQL, *DatabasePatternSinOrden* ignora el orden y busca todas las coincidencias posibles sin considerar la secuencia.

Para finalizar, hay que destacar las clases *InformationExtractor* y *Pagination*, las cuales son fundamentales para la manipulación y visualización de los resultados de búsqueda en nuestra aplicación.

La clase *InformationExtractor* está diseñada para resaltar y subrayar las frases encontradas y las palabras clave dentro de esas frases. Los métodos más importantes de esta clase son:

- **HighlightFrases:** Este método toma una lista de objetos *CorpusFrase* y recorre cada uno de ellos. Para cada frase, busca las palabras clave y las resalta. Esto se logra mediante la modificación de las propiedades de cada frase para indicar visualmente las palabras importantes.
- **MergeFrasesAndParrafos:** Este método combina las frases y los párrafos en un solo diccionario. Esto facilita la relación entre frases y el contexto de los párrafos a los que pertenecen, proporcionando una mejor experiencia de usuario al mostrar los resultados.

```
public Dictionary<FraseData, ParrafoAndDocumento> HighlightFrases(List<CorpusFrase> corpusFrases)
{
    foreach (CorpusFrase corpusFrase in corpusFrases)
    {
        this.Information.Where(x => x.Key.IdFrase == corpusFrase.Id_Frase)
            .ToList()  // Convertir a List
            .ForEach(y => y.Key.FraseResaltada = this.highlightText
                .ResaltarFrase(y.Key.Frase, corpusFrase.Elementos
                    .Select(x => new Posiciones(x.Posicion, x.Posicion)).ToList()));
    }
    return this.Information;
}

// Método para juntar las frases y los párrafos en un diccionario.
2 references
private Dictionary<FraseData, ParrafoAndDocumento> MergeFrasesAndParrafos(List<FraseData> frases, List<ParrafoAndDocumento> parrafosAndDocu)
{
    return frases.Zip(parrafosAndDocu, (a, b) => new { a, b }).ToDictionary(x => x.a, x => x.b);
}
```

Figura 61: Funciones de InformationExtractor

La clase *Pagination* es esencial para manejar la paginación de los resultados de búsqueda cuando hay muchas frases encontradas. La implementación de la paginación asegura que la interfaz de usuario permanezca limpia y manejable. A continuación, se detallan los aspectos clave de esta clase:

- **Constructor:** Al crear una instancia de *Pagination*, se le pasa una lista de *CorpusFrase*, el tamaño del párrafo (es decir, cuántas frases se deben mostrar por página), una bandera para indicar si está vacío y el nombre del corpus. La clase se encarga de inicializar el extractor de información y otros atributos necesarios.
- **GetInformation:** Este método es el encargado de extraer y resaltar la información necesaria. Primero, verifica si hay frases y si estas no contienen elementos. Luego, utiliza el extractor de información (*InformationExtractor*) para extraer y resaltar las frases según sea necesario.

```
public class Pagination
{
    private List<CorpusFrase> CorpusFrases;
    private readonly int paragraphSize;
    private InformationExtractor infoExtractor; // Clase para extraer la información de los archivos.

    1 reference
    public Pagination(List<CorpusFrase> CorpusFrases, int ParagraphSize, bool empty, string corpusName)
    {
        this.CorpusFrases = CorpusFrases;
        if (!empty)
        {
            paragraphSize = ParagraphSize;
            this.infoExtractor = new InformationExtractor(corpusName, this.paragraphSize);
        }
    }

    1 reference
    public Dictionary<FraseData, ParrafoAndDocumento> GetInformation()
    {
        if (CorpusFrases.Count > 0 && CorpusFrases[0].Elementos.Count == 0)
        {
            return this.infoExtractor.ExtractInformationFromFiles(CorpusFrases);
        }
        else
        {
            this.infoExtractor.ExtractInformationFromFiles(CorpusFrases);
            return this.infoExtractor.HighlightFrases(CorpusFrases);
        }
    }

}
```

Figura 62: Clase Pagination

Por último destacar que en la interfaz de usuario, se ha implementado una navegación intuitiva que permite a los usuarios moverse fácilmente entre las páginas de resultados. Las flechas hacia abajo permiten cambiar de página, y las flechas dobles permiten saltar directamente a la primera página, mejorando la navegación y accesibilidad de la aplicación.

Figura 63: Primera página de una búsqueda

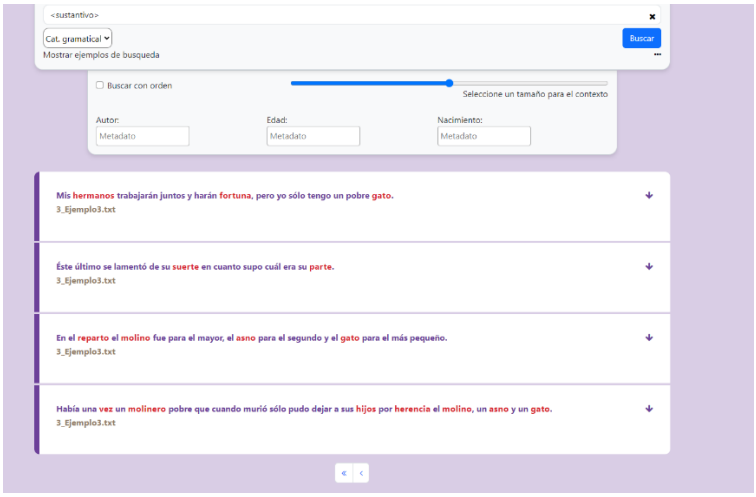


Figura 64: Segunda página de una búsqueda

Capítulo 5: Conclusiones y trabajo futuro

5.1 Evaluación de Resultados y Grado de Consecución de los Objetivos

Durante el desarrollo de este proyecto, se han logrado cumplir los objetivos establecidos inicialmente. La aplicación web creada permite a los usuarios gestionar corpus textuales de manera eficiente, facilitando la búsqueda de categorías gramaticales y variantes de palabras. El proyecto ha resultado en una plataforma funcional que automatiza y optimiza tareas que, de otro modo, consumirían mucho tiempo.

La implementación de una base de datos robusta y la creación de esquemas dinámicos para cada corpus han sido aspectos clave que han permitido un almacenamiento estructurado y eficiente de los datos. El uso de ASP.NET ha sido fundamental en este proceso, proporcionando las herramientas necesarias para desarrollar una aplicación web potente y escalable. La integración de componentes de ASP.NET y el manejo de autenticación, han demostrado ser valiosas adiciones a la funcionalidad del proyecto.

5.2 Conclusiones Personales

Desde hace tiempo tenía pensado realizar una aplicación web, pero desconocía totalmente ASP.NET. El desarrollo de este proyecto me ha permitido conocer a fondo las ventajas de esta tecnología y su complejidad. He aprendido no solo sobre ASP.NET, sino también sobre bases de datos y su gestión eficiente. Este proyecto me ha servido de mucho, ya que he podido explorar distintos enfoques de desarrollo y aportar mi esfuerzo para que futuros usuarios puedan automatizar su tiempo en la búsqueda de categorías gramaticales o variantes de palabras. Ver cómo esta herramienta puede ayudar a otros a aprender me complace enormemente.

5.3 Futuras ampliaciones

- **Implementación del Apartado de Profile del Usuario:** Un aspecto crucial para el desarrollo futuro de la aplicación es la implementación de una sección de perfil de usuario. Este apartado permitirá a los usuarios ver y gestionar su información personal, como su nombre, correo electrónico y contraseña. Además, incluirá opciones para personalizar sus preferencias dentro de la aplicación y gestionar sus corpus de manera más eficiente.

- **Capacidad para Añadir Más Textos a un Corpus Existente:** Otro desarrollo futuro importante es permitir que los usuarios puedan añadir más textos a un corpus ya creado. Actualmente, los corpus se crean con un conjunto inicial de textos, pero la posibilidad de ampliar estos corpus con nuevos textos mejorará significativamente la flexibilidad y utilidad de la aplicación. Este desarrollo implicará la adaptación de las funciones de carga y procesamiento de textos para integrarse con los corpus existentes sin necesidad de crear nuevos desde cero.

- **Ampliación del Sistema de Búsqueda**: Ampliar el sistema de búsqueda para incluir funcionalidades avanzadas será un objetivo clave. Esto podría incluir búsquedas semánticas y contextuales, permitiendo a los usuarios encontrar no solo categorías gramaticales, sino también sinónimos, antónimos y contextos específicos. Además, se pueden implementar filtros más detallados.

- **Análisis Estadístico de Palabras y Tipos de Palabra**: Implementar herramientas de análisis estadístico permitirá a los usuarios generar informes detallados sobre la frecuencia de palabras, tipos de palabras y otras métricas relevantes. Esto proporcionará una comprensión más profunda de los textos y facilitará la investigación y el análisis lingüístico. La integración de visualizaciones gráficas para estos análisis también mejorará la accesibilidad y comprensión de los datos.

- **Mejora de la Interfaz de Usuario:** Finalmente, mejorar la interfaz de usuario es un objetivo continuo. Una interfaz más intuitiva y accesible, con mejoras en la usabilidad y la experiencia del usuario, es esencial para aumentar la eficiencia y satisfacción de los usuarios. Esto podría incluir la implementación de características como arrastrar y soltar para la carga de archivos, visualizaciones gráficas interactivas, y una navegación más fluida y moderna.

Capítulo 6: Bibliografía

[1] Curatr. (Mayo de 2024). Obtenido de https://ar5iv.labs.arxiv.org

[2] IAtext. (Abril-Mayo de 2024). Obtenido de https://cmtm.iatext.ulpgc.es/modecan/Buscador

[3] INCEpTION. (Mayo de 2024). Obtenido de https://inception-project.github.io/

[4] Microsoft. (26 de Febrero de 2024). Obtenido de https://dotnet.microsoft.com/es-es/learn/aspnet/what-is-aspnet

[5] Microsoft. (28 de Febrero de 2024). Obtenido de https://learn.microsoft.com/en-us/aspnet/web-forms/

[6] Microsoft. (Marzo-Mayo de 2024). Obtenido de https://learn.microsoft.com/en-us/answers/tags/97/dotnet

[7] MVC, A. .. (Mayo de 2024). Obtenido de https://learn.microsoft.com/en-us/aspnet/mvc/

[8] MySQL. (Febrero-Mayo de 2024). Obtenido de https://www.mysql.com/

[9] OpenAI. (Junio de 2024). *ChatGPT*. Recuperado el 4 de June de 2024, de https://chatgpt.com/?oai-dm=1

[10] Search, O. S. (Mayo de 2024). Obtenido de https://opensemanticsearch.org/

[11] StackOverflow. (Febrero-Mayo de 2024). Obtenido de https://stackoverflow.com/

[12] ULPGC-IAtext. (Febrero de 2024). Obtenido de https://iatext.ulpgc.es/sites/default/files/InteLiText.pdf

[13] Yext. (Mayo de 2024). Obtenido de https://www.yext.com/