Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Transformación Digital en la elaboración de Leyes usando editores WYSIWYG

Digital Transformation in Law-making using WYSIWYG editors

Álvaro González Rodríguez

La Laguna, 13 de junio de 2022

Dña. **María Elena Sánchez Nielsen**, con N.I.F. 42.848.599-J Profesora Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

C E R T I F I C A (N)

Que la presente memoria titulada:

“Transformación Digital en la elaboración de Leyes usando editores WYSIWYG”

ha sido realizada bajo su dirección por D. **Álvaro González Rodríguez**,

con N.I.F. 79.083.919-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de junio de 2022

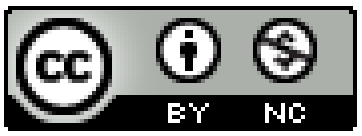
Agradecimientos

Agradecer a mi tutora por todo el apoyo, la dedicación y el trabajo realizado para ayudarme a completar este proyecto, también por tu simpatía y tranquilidad que transmitías en todas las reuniones.

Agradecer a mis amigos por haber seguido conmigo durante todo este recorrido, por haber hecho de esta experiencia un buen recuerdo.

Sobre todo agradecer a mi familia por el gran apoyo y respaldo que han sido para mi, en especial a mis padres y a mi hermano, sin ellos no hubiera llegado tan lejos. Muchas gracias.

Licencia



Resumen

El objetivo de este trabajo ha sido el desarrollo e implementación de un sistema que permita llevar a cabo propuestas de modificaciones a las diferentes partes de la ley (enmiendas) de forma sencilla y amigable utilizando editores WYSIWYG (acrónimo de "What you See is What You Get", en español, "lo que ves es lo que obtienes") y permita la visualización eficiente de los cambios propuestos a una ley.

Para ello, se realizaron una serie de tareas basadas en el estudio previo y comparativas de soluciones relacionadas con editores WYSIWYG, el desarrollo del sistema y la implementación de las tres fases esenciales implicadas en las modificaciones de una ley: inserción de los artículos, inserción de enmiendas y visualización de cambios propuestos.

Respecto a la aplicación, se han utilizado distintas tecnologías en su desarrollo. Se ha trabajado en el framework ASP.NET Core en Visual Studio 2022 con carga de trabajo para aplicaciones web, donde se hace uso de C# para el back-end y HTML para el front-end. SQL Server es el gestor de la base de datos elegido, mientras que Entity Framework Core se implementa en el código y actúa como asignador relacional de objetos.

Además, para concluir el trabajo, se ha realizado una evaluación experimental con un entorno real, es decir, introduciendo los datos de una ley y simular unas propuestas de modificaciones para la misma. De esta manera se consigue probar el correcto funcionamiento del aplicativo.

Palabras clave: editor WYSIWYG, enmiendas, ley, aplicación web, artículos, ASP.NET Core.

Abstract

The objective of this work has been the development and implementation of a system that allows making proposal for modifications to the different parts of the law (amendments) in a simple and friendly way using WYSIWYG editors (acronym for "What You See Is What You Get") and allow efficient visualization of proposed changes to a law.

A set of tasks were carried out based on the previous study and comparison of differents solutions related to WYSIWYG editors, the development of the system and the implementation of the three essential phases involved in the modificación of a law: insertion of articles, insertion of amendments and display of proposed changes.

Regarding the application, different technologies have been used in its development. The ASP.NET Core framework has been used in Visual Studio 2022 with a workload for web applications, where C# is used for the back-end and HTML for the front-end. SQL Server is the database manager selected, while Entity Framework Core is implemented in the code and acts as an object relational mapper.

In addition, to conclude the work, an experimental evaluation has been carried out with a real environment, that is, introducing the data of a law and simulating some proposed modifications to it. In this way it is possible to test the correct performance of the application.

Keywords: WYSIWYG editor, amendments, law, web application, articles, ASP.NET Core

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Motivación

En un mundo donde la tecnología cada vez es más necesaria para la realización de simples tareas cotidianas, no es de extrañar que la digitalización esté siendo una tarea fundamental a realizar en las empresas. Ya sea grande o pequeña, para un negocio es primordial no quedarse estancado respecto a la competencia.

El término digitalización [1] hace referencia a la transformación que sufre un documento al pasar de formato físico a digital, de esta manera un documento se vuelve mucho más accesible, no ocupa espacio físico y puede tenerse un control de las modificaciones mucho más estricto. Además, una empresa totalmente digitalizada puede abarcar mucho más mercado y permite adaptarse mucho más fácilmente a los cambios del mismo.

Pero la digitalización no solamente abarca el ámbito de los documentos, una empresa también puede digitalizar sus procesos, es decir, un proceso que antes conllevaba varios pasos manuales con la digitalización se puede resumir en unos simples pasos a través de un ordenador.

Es por esto último que el proceso del desarrollo de leyes se puede simplificar drásticamente si se lleva al ámbito digital. Donde en cualquier momento se puede desarrollar un artículo de la ley, esperar a sus enmiendas y ser aceptada o denegada. Por lo que este proyecto busca desarrollar una manera intuitiva para el manejo y desarrollo de leyes en el ámbito digital, donde se hará especial hincapié en el uso de editores WYSIWYG.

Un editor WYSIWYG [2], del acrónimo What You See Is What You Get, son editores que permiten ver directamente el resultado final de un documento mientras este se escribe. Este tipo de editores son fundamentales para el desarrollo del presente proyecto, ya que como se comentó anteriormente, se busca que la escritura del documento sea un proceso sencillo, por lo que de esta manera no se tendrá que tocar directamente código en HTML para el desarrollo de la ley.

1.2 Objetivos

1.2.1 Objetivo general

El objetivo principal es el de implementar una aplicación web intuitiva y cómoda para el usuario donde se pueda administrar una base de datos. En dicha base de datos se podrá insertar artículos de una ley y sus enmiendas haciendo uso de un editor WYSIWYG. Además, se tiene la posibilidad de visualizar los cambios propuestos sobre un artículo de la ley comparándolos con el texto original.

1.2.2 Objetivos específicos

Otros objetivos que serán tratados son los siguientes:

- Estudio y comparativa de las soluciones existentes en cuanto a editores WYSIWYG.
- Diseño de una base de datos.
- Diseño e implementación de la aplicación web con los requisitos comentados en el objetivo general
- Estudio y conclusiones de los resultados obtenidos mediante su evaluación experimental haciendo uso de una ley.

1.3 Tecnologías utilizadas

Al tratarse de una aplicación web, se ha optado por hacer uso de ASP.NET Core [3], un framework de Microsoft que tiene como finalidad desarrollar aplicaciones web tanto en Windows, Linux o Mac. Para trabajar ha sido necesario usar Visual Studio 2022 [25], ya que en la actualidad no existen más posibilidades de entornos que permitan trabajar en .NET. Por otro lado, se hace uso de una arquitectura modular, por lo que se pueden utilizar paquetes con funcionalidades necesarias para la aplicación. Además, se trabaja en la versión más moderna, .NET 6.

Trabajar con este tipo de frameworks trae un gran número de ventajas, destacando entre ellas la capacidad de poder utilizar distintos lenguajes de programación para distintas finalidades. Por ejemplo, en la aplicación desarrollada, el backend está escrito en C#, mientras que el frontend en HTML. También cabe destacar que en el frontend se tiene la posibilidad de añadir código en C# haciendo uso de la extensión de fichero “cshtml” que está soportado en .NET.

Gracias al framework con el que se está trabajando se puede hacer uso de Entity Framework (EF) Core [4], un mapper que permite trabajar con una base de datos con objetos .NET, el cual es compatible con varios motores de bases de datos como SQL Server.

Respecto a los patrones de diseño, como es necesario estar trabajando con una base de datos a través de una interfaz de usuario, se ha hecho uso del patrón Modelo-Vista-Controlador (MVC). Con este patrón se puede separar la lógica de negocio con las vistas o interfaces a través de un controlador.

Capítulo 2 Estudio previo

Una de las tareas fundamentales del proyecto consiste en la realización de un estudio de las soluciones existentes en cuanto a editores WYSIWYG, donde se tendrán en cuenta distintos factores que favorezcan la experiencia del usuario y las exigencias técnicas del editor.

Por otro lado, también se ha investigado la tecnología con la que se desarrollará la aplicación teniendo en cuenta la complejidad de la misma.

2.1 Editor WYSIWYG

2.1.1 ¿Qué es un editor WYSIWYG?

Del acrónimo What You See Is What You Get, un editor WYSIWYG tiene la capacidad de mostrar por pantalla el resultado final de lo que escribe un usuario sin la necesidad de codificarlo. Por ejemplo, si un usuario quiere poner una palabra en negrita no tiene que escribir la palabra deseada entre etiquetas, será el editor quien le dará la opción de poner la palabra en negrita y verlo por pantalla a través de una opción en su interfaz. Este ejemplo lo podemos ver en uno de los procesadores de texto más famosos del mercado, Microsoft Word.

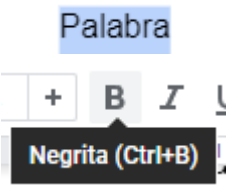
	Código	Procesador de texto
Acción	Palabra	
Resultado final	**Palabra**	**Palabra**

Tabla 2.1: *Comparación entre código y procesador de texto*

Incluso los editores WYSIWYG tienen la capacidad de crear páginas web sin la necesidad de que se tengan muchos conocimientos de HTML, aunque en la actualidad no es muy recomendable ya que muchas de las ventajas que te proporcionaban estos tipos de editores ya son recogidas en otros muchos entornos de programación o frameworks. Por lo general, las ventajas de usar un editor WYSIWYG son las siguientes

- Cambiar la fuente del texto al momento sin la necesidad de estar modificando un fichero .CSS.
- No utilizar etiquetas.
- Insertar directamente contenido multimedia.
- Dar estructura a un fichero de manera sencilla.

2.1.2 Antecedentes

La principal razón de la aparición de las técnicas WYSIWYG fue la eliminación de las etiquetas en los textos que no están formateados.

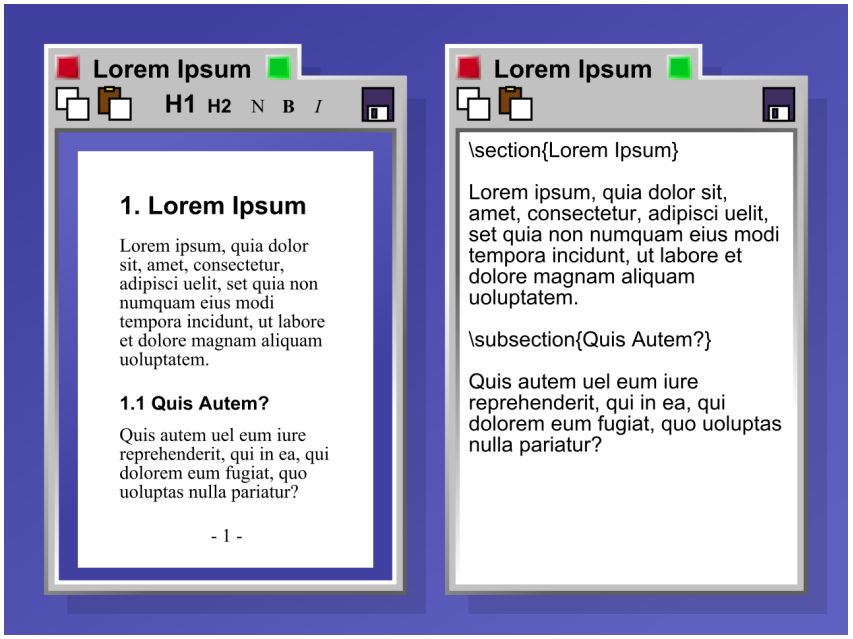


Figura 2.1: *Funcionamiento de editor WYSIWYG. De Shinobu - Own work, drawn with Inkscape, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=915515*

Bravo [16] se considera que es el primer editor que usa las técnicas WYSIWYG. Desarrollado en 1974, su principal característica era la posibilidad de cambiar el tipo de fuente de un texto, también teniendo la capacidad de visualizar los márgenes y sangrías. Esto dio pie a que cuatro años después, en 1978, Hewlett Packard, más conocido como HP, desarrolló el primer software WYSIWYG para producir presentaciones gráficas, cuyo nombre fue BRUNO [17].

En ese mismo año se publicó al mercado WordStar [18], un procesador de texto que no requería de las etiquetas para representar palabras en negrita o en cursiva, aunque no fue hasta tres años después que se empezó a dar publicidad como un editor WYSIWYG. Dominó los mercados en los principios de los 80, pero quedó rápidamente obsoleto por culpa de su arquitectura portable, ya que era prácticamente imposible añadir funcionalidades nuevas al software sin sacrificar su rendimiento. Actualmente es un proyecto abandonado y ya no recibe actualizaciones por parte de sus desarrolladores.

Los problemas de WorldStar dieron pie a otro procesador de texto tomar su lugar y volverse uno de los procesadores más utilizados mundialmente. Este es WordPerfect [19], el cual destacó por su interfaz de usuario, donde las opciones básicas eran resaltadas y ocultaba las que menos usaban los usuarios, de esta manera conseguía una interfaz poco cargada y amigable para el usuario. Mantuvo su pico de popularidad hasta que Microsoft lanzó su propio procesador de texto para Microsoft Windows a principios de los 90.

Microsoft Word [20], en su lanzamiento llamado Multi-Tool Word, es el procesador de texto de Microsoft y actualmente es el líder del mercado. La principal característica que trajo tanto críticas como halagos fue el uso del ratón para trabajar. Esta posición de líder la ha conseguido, a parte de sus constantes actualizaciones, gracias al paquete Microsoft Office [21] cuyo paquete, además del procesador de texto, incluye un programa de hoja de cálculos, de presentaciones, sistema de base de datos, entre otros.

También hay que destacar el trabajo de LibreOffice [22], que es otro paquete de trabajo lanzado en 2011 con la ventaja de que es totalmente gratuito. Como es de esperar, el paquete incluye un procesador de texto llamado LibreOffice Writer [23] parecido en gran medida con Microsoft Word.

2.1.3 Opciones

Se han tenido en cuenta los siguientes requisitos en la búsqueda de un editor WYSIWYG para que cumpla con las expectativas:

- Open Source o comercial.
- Personalización.
- Documentación existente proporcionada por los desarrolladores.
- Uso en dispositivos móviles.

Sobre todo se ha prestado especial atención los dos primeros puntos en la elección final, aunque también se ha tomado como referencia reseñas de usuarios, fechas de lanzamiento, frecuencias de actualizaciones y dificultad en la implementación.

Dentro de un gran abanico de posibilidades, han destacado los siguientes editores:

- **CKEditor** [6]: Se trata de uno de los editores más populares del mercado siendo utilizados y aprobados por grandes marcas como Microsoft, IBM o Disney. Es de código abierto y tiene una larga trayectoria, ha pasado por más de 32.000 tests de seguridad y rendimiento a lo largo de su historia. Además, se tiene la ventaja de que su arquitectura es modular, por lo que es bastante sencillo de añadir plugins para aumentar sus funcionalidades. Por otro lado, es un editor que se ha desarrollado con la idea de colaborar con otros usuarios, con la posibilidad de observar cambios y su historial. Podría ser la mejor opción para el proyecto pero tiene la desventaja que muchas de sus características base están bloqueadas detrás de una versión de pago.
- **TinyMCE** [7]: El foco principal de este editor es la fácil implementación del mismo en la aplicación, es por eso que para su integración sólo hace falta añadir unas pocas líneas de código para que funcione correctamente. También es de código abierto y permite el añadido de plugins propios de los desarrolladores, con la desventaja de que para hacer uso de muchos de ellos hace falta pagar. Otro punto a favor es su documentación, la cual está muy bien organizada y está disponible en su página web.
- **Summernote** [8]: Es el primer editor de la lista que es completamente gratuito y de código abierto. Tiene una interfaz bastante simple pero bastante personalizable, donde se tiene la posibilidad de quitar opciones a la interfaz o añadir algunas a través de plugins propios o de terceros. También tiene la ventaja de tener plantillas de temas para la interfaz del editor, por lo que el trabajo de personalización se hace mucho más sencillo. Por desgracia, no cuenta con una buena documentación ya que esta sólo te ayuda en los primeros pasos de instalación y no te describe su API.
- **Quill** [9]: Se trata de otro editor WYSIWYG totalmente gratuito cuyo principal enfoque es el de facilitar las tareas de implementación y uso a los desarrolladores a través de una API simple. Además, otro punto a favor es su personalización, gracias a la API comentada y su capacidad de adaptarse a los dispositivos móviles. Otra característica que lo diferencia del resto de editores es el uso de Deltas [5] para el manejo de la información. Su documentación es bastante completa y muy bien organizada, esta trata desde los aspectos más básicos del editor, como su

implementación e integración en distintos entornos, hasta la definición de todos los métodos de la API.

- **SCEditor** [10]: Editor simple cuya principal característica es su poco espacio de almacenamiento y altas velocidades de respuesta. Es completamente gratuito y de código abierto, el cuál está constantemente recibiendo actualizaciones por parte de todos los contribuidores al proyecto. Tiene unos pocos plugins propios, pero se le puede añadir plugins de terceros. Su documentación se encuentra en su página web, la cual se divide en sub-apartados para facilitar la búsqueda de la información.

Se ha barajado la posibilidad de utilizar otros editores WYSIWYG pero estos se han descartado, ya sea porque el uso que los desarrolladores del editor tenían pensado eran distintos al objetivo de la aplicación que se está trabajando o porque están desactualizados respecto al resto de editores.

Jodit [11] es un editor que trabaja en puro TypeScript, sin librerías, por lo que no es una opción para una persona que no tenga muchos conocimientos de TypeScript. Froala [12], al ser un editor sin edición gratuita, está fuera del alcance de muchas personas. Aloha Editor [13] es bastante interesante ya que no trabaja con un cuadro de texto, si no que el usuario selecciona la parte de la página web que quiere editar para poder interactuar con ella y realizar cambios a tiempo real, pero este no es el objetivo del presente proyecto. Medium Editor [14] es parecido al último comentado pero lleva un tiempo sin recibir actualizaciones. Trumbowyg [15] es bastante completo, se le pueden añadir plugins y seleccionar distintos idiomas, pero, al menos en Google Chrome, tarda en cargar. A continuación se puede observar una tabla comparativa de los editores considerados:

Nombre	Open Source	Gratuita	Librerías	Plugins	Documen-tación	Móvil	QuickStart
CKEditor							
TinyMCE							
Jodit							
Summernote							
Froala							
Setka							
Aloha							
SCEditor							
Content							
Quill							
Medium							
Trumbowyg							

Tabla 2.2: *Comparativa de editores WYSIWYG*

2.1.4 Editor WYSIWYG elegido

Finalmente, después de probar gran parte de los editores comentados en el punto anterior, se ha decidido optar por usar el editor Quill. La principal característica que destaca con el resto es su gran customización gracias a su arquitectura modular y su API, al igual que su fácil implementación y aprendizaje.

En el capítulo 3 se comentará en profundidad su finalidad y uso en el presente proyecto.

Figura 2.2: *Logotipo de Quill, https://quilljs.com/*

2.2 ¿Por qué Visual Studio 2022 y ASP.NET Core?

Los framework o entorno de trabajo proporcionan herramientas que facilitan las tareas de desarrollo en una aplicación web y a la misma vez mantienen la seguridad de la misma. En la actualidad existen múltiples variedades de frameworks enfocados a distintos lenguajes de programación, es decir, si el proyecto va a realizarse en Python es mejor utilizar el framework Django o si se va a realizar en Ruby mejor utilizar Ruby on Rails, por lo que es recomendable escoger uno que facilite el desarrollo del proyecto.

En el caso que nos ocupa, el proyecto va a hacer uso de ASP.NET Core, un framework totalmente gratuito y de código abierto desarrollado por Microsoft. Su estructura modular distribuida en paquetes NuGet [24] permite añadir funcionalidades al código de manera sencilla y sin ocupar mucho espacio, por lo que es uno de sus puntos fuertes. El back-end de las aplicaciones web desarrolladas en este framework se escribe en C# mientras que su front-end principalmente en HTML. El entorno de desarrollo integrado [26], o IDE de sus siglas en inglés, utilizado es Visual Studio 2022 ya que es la única opción disponible para poder hacer uso de ASP.NET Core.

2.2.1 Entity Framework (EF) Core

Se trata de un sistema de mapeo objeto-relacional de código abierto originalmente diseñado para .NET Framework. Este tipos de sistemas se tratan como una serie de técnicas de programación que permiten trabajar con un sistema de tipos utilizando un lenguaje de programación orientado a objetos usando una base de datos relacional. En el presente proyecto, el lenguaje de programación orientado a objetos usado es C# mientras que la base de datos es SQL Server [28].

EF Core hace uso de Entity Data Model (EDM), donde principalmente se describen entidades y asociaciones junto con un esquema EDM. Las entidades representan las instancias individuales de un objeto, en C# hacen uso de las clases para ser creadas. Estas entidades están formadas por propiedades que normalmente tienen un nombre y un tipo para ser representadas, donde los tipos pueden ser booleanos, binarios, strings, etc. Por otro lado, dos entidades pueden ser relacionadas a través de las asociaciones y a estas se le pueden asociar operaciones o acciones, como puede ser la operación en borrado (OnDelete) en cascada.

2.2.2 Modelo-Vista-Controlador (MVC)

Modelo-Vista-Controlador (MVC) [27] es un patrón de arquitectura software que separa la lógica de negocio con la interfaz de usuario a través de una serie de controladores. De forma general, este patrón tiene tres componentes:

- Modelo: Representa toda la información del sistema, por lo que es el encargado de realizar todas las operaciones y consultas. Actúa cada vez que la vista, a través del controlador, requiere algo de información.
- Vista: Se trata de la interfaz de usuario.
- Controlador: Es quien responde las acciones que realiza el usuario en la vista o interfaz. Normalmente la vista envía una petición al controlador, por lo que este la recibe y responde conforme a la petición. Básicamente trabaja como intermediario entre la vista y el modelo, de esta manera se tiene una separación clara y concisa entre los datos de la aplicación y su representación visual.

Capítulo 3 Desarrollo

Una vez se tiene claro el objetivo del proyecto, se explica paso a paso el desarrollo de la aplicación web.

3.1 Primeros pasos

Antes que nada, se ha tenido que descargar Visual Studio Community 2022 con la carga de trabajo para desarrollo de ASP.NET y web. Esta carga es necesaria para el proyecto ya que sin ella no se podría compilar las aplicaciones web con ASP.NET Core.

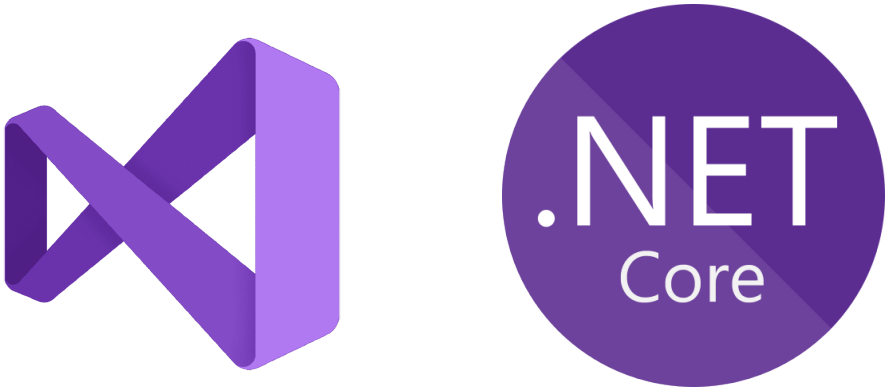


Figura 3.1 y 3.2: *Logotipo de Visual Studio (https://visualstudio.microsoft.com/es/) y ASP.NET Core (https://docs.microsoft.com/es-es/aspnet/core/?view=aspnetcore-6.0) respectivamente.*

Para la creación del proyecto se puede utilizar una plantilla para aplicaciones web que proporciona Visual Studio, la cual viene con una página de bienvenida y un pequeño controlador de la misma. Durante los primeros pasos del proyecto estos ficheros no se modificaron hasta que se empezó a trabajar con la interfaz de la aplicación. Además, se ha seleccionado el framework .NET 6, como se ha comentado anteriormente.

Como el propósito principal de la aplicación es el manejo de una base de datos de artículos de leyes, se ha instalado el paquete NuGet que contiene todas las herramientas para el funcionamiento de Entity Framework Core [29] desde un inicio del proyecto, aunque antes de empezar a trabajar sobre la aplicación es necesario tener al menos un diseño inicial de la base de datos que se va a trabajar.

Además, como .NET Core aprovecha muy bien el patrón MVC, se ha creado desde un inicio una carpeta para almacenar todos modelos que corresponden a una tabla de la base de datos.

3.2 Diseño de la base de datos

En un principio solo se tenía pensado diseñar una base de datos que tuviera la tabla de artículos y otra para la de enmiendas, pero finalmente se decidió con añadir otra para los títulos de las leyes, de esta manera se podría organizar de mejor manera los artículos de las leyes y saber si dos artículos con mismo nombre son de distintas leyes.

El diseño se realizó en MySQL Workbench [30], una aplicación con interfaz visual que permite diseñar bases de datos de forma sencilla pudiendo representar las tablas de forma visual.

Cabe destacar que lo realizado en esta herramienta solamente es un elemento orientativo y no representa el resultado final de la base de datos, ya que esta ha ido cambiando constantemente a lo largo del desarrollo del proyecto, añadiendo nuevas columnas o modificando otras en algunas de las tablas.

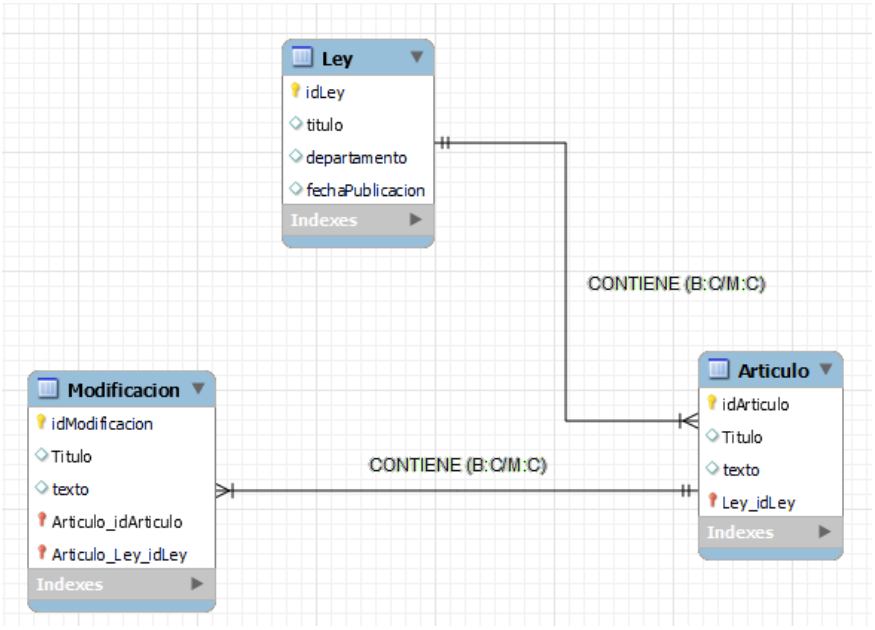


Figura 3.3: *Diseño inicial de la base de datos.*

A continuación un breve resumen del contenido de la base de datos mostrada:

- Ley: Tabla que representa el título de una ley y otros aspectos más generales relacionados con la misma. Esta tabla está abierta a cambios y pueden añadirse y quitarse columnas de forma sencilla. También es importante comentar que una ley puede contener uno o más artículos.
- Artículo: Representa el artículo de una determinada ley, por lo que en inicio sólo hace falta almacenar su contenido y título. Respecto a sus relaciones, un artículo puede tener cero o más modificaciones mientras que sólo puede estar relacionada con una ley.
- Modificación: Es la enmienda sobre un artículo de la ley, por lo que en un principio sólo hace falta almacenar el texto modificado y un título. Además, una entrada en la tabla sólo puede estar relacionada con un artículo.

3.3 Uso de EF Core

El primer paso para trabajar una aplicación con EF Core es la creación de los modelos de la base de datos. Estos modelos representan las tablas en el diseño que se mostró en el apartado anterior, por lo que será necesario la creación de tres modelos diferentes, uno para leyes, otro para artículos y otro para modificaciones. Para ello simplemente se tiene que crear una clase en C#, ya que las entidades en EF Core son representadas de esta manera, y llamarla como la tabla de la base de datos, por ejemplo Artículo, e irle añadiendo atributos públicos, los cuales representan las propiedades de una entidad. También es importante comentar que gracias a la tecnología de .NET Core se puede añadir restricciones a las propiedades de forma sencilla gracias a la validación de anotación de datos o *Data Annotations* [31], donde simplemente con una línea de código se puede especificar una restricción al modelo, por ejemplo, añadir un tamaño mínimo a la cadena de una propiedad o especificar un mensaje de error cuando no se añade un elemento necesario a la tabla.

Una vez se han creado los tres modelos necesarios, el siguiente paso es la creación del contexto de la base de datos. Para ello se tiene que hacer uso de una herencia o derivación de DbContext de EF Core. Esta derivación representa una sesión con la base de datos en la que se pueden hacer peticiones y salvar entidades, por lo que va a ser necesario representar las tres tablas que se mostraron en el apartado anterior. En el caso que nos ocupa, el contexto de la base de datos se ha denominado *LeyesTFGContext*, y este contexto incluye tres entradas, una para los artículos, otra para las modificaciones y otra para las leyes. Además, es recomendable sobreescribir la función *OnModelCreating* para especificar las relaciones entre las tablas y la manera del borrado de las mismas, por lo que se consigue un comportamiento como el requerido en el apartado anterior, donde las entradas de leyes y artículos tienen borrado en cascada. Para poder hacer uso de este contexto se tiene que especificar su uso con la inserción de dependencias en el fichero *Program.cs*, quien se podría decir que es el fichero de configuración de la aplicación y también quien lo inicia.

Al finalizar el diseño de los modelos y el contexto, se tienen que crear sus respectivos controladores y vistas. Para ello, Visual Studio proporciona una herramienta que agiliza el trabajo del desarrollador con la creación de un elemento con scaffolding, el cual permite crear un controlador básico con las acciones de creación, edición y eliminado de una base de datos y una vista general de la tabla. Haciendo uso de la plantilla que proporciona Visual Studio para un controlador de MVC con vistas que usan Entity Framework. Como se observa en la siguiente figura, hace uso de un modelo y del contexto, elementos que ya están creados. Además se tiene que especificar el nombre del controlador y también se da la opción de crear las vistas con una plantilla predeterminada.

Además, es importante comentar que para agilizar el entorno de desarrollo, se ha creado una clase que añade entradas de forma automática a la base de datos en caso de que esta se encuentre vacía, la cual se llama *SeedData*. Para ello se ha desarrollado un método dentro de la misma clase que será llamado al iniciar la aplicación.

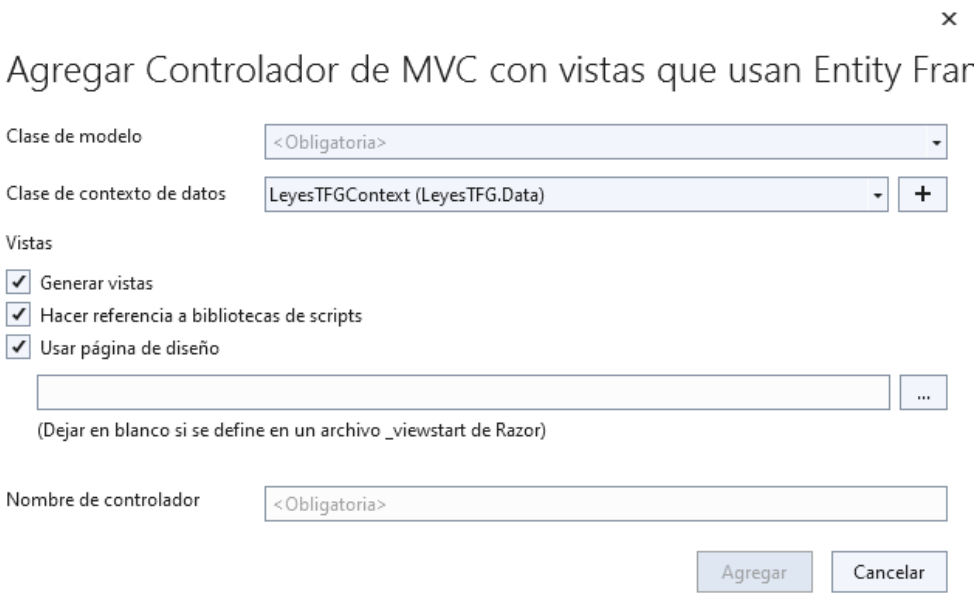


Figura 3.4: *Creación de elementos con Scaffolding.*

Para poder iniciar la aplicación y tener una primera vista de las tablas es necesario realizar una primera migración [32]. Estas migraciones son necesarias para crear y actualizar el esquema que representa los modelos y la base de datos en EF Core. Además, siempre que se realiza un cambio en los modelos es necesario realizar una migración, ya que tiene que mantenerse los datos sincronizados entre la aplicación y la base de datos.

Para realizar la primera migración se puede hacer uso del PowerShell de Visual Studio con la siguiente instrucción:

Add-Migration NombreDeLaMigración

Esto creará una carpeta llamada *Migrations* junto con un fichero que representa los datos de la base de datos. Luego será necesario crear el esquema con el siguiente comando:

Update-Database

Una vez realizada la migración, ya se puede iniciar la aplicación sin problemas, aunque en este momento del proyecto la aplicación es muy básica, por lo que se le ha añadido más funcionalidades a los controladores para mejorar el rendimiento y la experiencia de usuario por partes iguales. Pero para poder realizar esta tarea primero hay que entender cómo funcionan los controladores en .NET Core.

Los controladores, por norma general, tienen dos funciones asociadas a una acción que puede realizar un usuario, una para poder mostrar la ventana con la información necesaria utilizando datos del modelo y otra para poder realizar cambios sobre los datos del modelo. Por ejemplo, la acción de editar un artículo tiene que tener dos métodos, uno que cargue la información del dato a editar y otro que los actualice. Sobre todo se ha tenido en cuenta los datos que necesitan información de otras tablas, el mismo método editar artículos tiene que realizar una consulta a la base de datos para poder saber el título de la ley al que está asociada. Para ello se hace uso de EF Core.

```
var leyQuery = from l in _context.Ley
               orderby l.LeyId
               select l;
ViewBag.LeyId = new SelectList(leyQuery.AsNoTracking(), "LeyId", "Titulo", LeySeleccionada);
```

Figura 3.5: *Ejemplo de consulta usando sintaxis de consultas.*

Aunque esta no es la única manera en la que se puede realizar consultas, como EF Core usa Language Integrated Query (LINQ) [33] se puede hacer uso de otro tipo de sintaxis, conocida como sintaxis de métodos. En el código que nos ocupa, normalmente se utiliza la sintaxis de consultas para realizar búsquedas un poco más complejas o que tengan algún requisito, mientras que la de métodos se usa principalmente cuando se requieren datos anidados, como se podrá observar en la siguiente figura.

```
var modificacion = await _context.Modificacion
    .Include(c => c.Articulo)
        .ThenInclude(a => a.Ley)
    .FirstOrDefaultAsync(m => m.ModificacionId == id);
```

Figura 3.6: *Consulta de modificaciones que requieran datos de artículos y leyes.*

También es importante recalcar que en todo momento los controladores necesitan conocer el contexto de la base de datos para poder acceder y conocer si existe determinada propiedad o poder actualizar cambios en la misma, es por eso que se necesita tener un atributo privado que contenga la información del contexto. Como se puede observar en las dos anteriores figuras, para poder realizar las dos consultas ha sido necesario utilizar el atributo *_context,* mientras que para poder actualizar el contexto se tiene que llamar a un método de DbContext llamado *SaveChangesAsync*.

De forma resumida, la aplicación necesita tres controladores para la base de datos, siendo estos los más complejos, y otro para la ventana de inicio. Los tres primeros controladores tienen los siguientes métodos para poder mostrar las vistas.

- Index: Carga información de la tabla.
- Details: Carga información de una entrada de la tabla.
- Create: Tiene dos métodos, uno que carga la información que se necesita para crear una entrada en la tabla y otro HttpPost para poder actualizar los datos.
- Edit: Tiene dos métodos, uno que carga la información de una entrada y otro HttpPost para poder actualizar los datos.
- Delete: Tiene dos métodos, uno que carga la entrada para mostrar los datos a borrar y otro para confirmar la acción.

Los controladores no sólo tienen los métodos anteriormente comentados, también se han añadido otros que son necesarios para poder desarrollar funcionalidades de la aplicación o incluso se han ido modificando los ya creados para que puedan realizar más acciones, en el siguiente apartado se comenta con más profundidad.

Como apunte, .NET Core contiene un atributo llamado *ModelState*, que es principalmente usado para saber si el dato que va a ser introducido en la base de datos es válido.

3.4 Funcionalidades añadidas

Con el fin de mejorar la experiencia de usuario, se han añadido funcionalidades al código que permiten visualizar y manejar información de forma más vistosa y amigable.

3.4.1 Vínculos en la página de inicio.

Además de añadir una pequeña descripción de la funcionalidad de la aplicación, en la ventana de inicio también se puede encontrar unos botones que sirven como vínculos para acceder a las distintas tablas de la base de datos, de esta manera no se tiene que introducir manualmente el enlace a la tabla en el buscador. También se ha creado una página de información que describe las tablas y sus propiedades que puede ser accedida por estos mismos botones.

Figura 3.7: *Botones para acceder a las tablas.*

Y, para acceder en cualquier momento a estas tablas, también se han añadido sus enlaces en una cabecera que está compartida en todas las ventanas de la aplicación

3.4.2 Filtro de búsqueda.

Cuando la tabla empieza a desbordarse con nuevas entradas es muy complicado buscar un determinado dato, es por eso que se pensó crear un filtro de búsqueda. Las tablas de leyes y modificaciones tienen un filtro según el título de la entrada, mientras que la de artículos busca según la ley a la que está asignada.

Para poder crear el filtro ha sido necesario modificar el método *Index* de los controladores, donde se ha añadido una variable que almacenará la información de la búsqueda. También se hace uso de un atributo de .NET Core llamado *ViewData*, un atributo que pertenece a la clase padre del controlador y su principal funcionalidad es la del paso de información entre el controlador y la vista. La idea principal es que cada vez que se pulse el botón de buscar haga que se recargue la página con el valor de búsqueda, por lo que el método del índice del controlador recogerá esa información y limitará la consulta de datos, dando como resultado una tabla que muestra las entradas con los criterios de búsqueda.

```
public async Task<IActionResult> Index(string busqueda)
{
    ViewData["Filtro"] = busqueda;
    var articulo = _context.Articulo
        .Include(c => c.Ley)
        .AsNoTracking();
    if (!String.IsNullOrEmpty(busqueda))
    {
        articulo = articulo.Where(c => c.Ley.Titulo.Contains(busqueda));
    }

    return View(await articulo.ToListAsync());
}
```

Figura 3.8: *Método Index de la tabla artículos con el filtro de búsqueda.*

3.4.3 Edición de la tabla artículos y autocompletar.

Uno de los requisitos de la aplicación es mantener la edición de los artículos fuera de su tabla y dejar constancia de los posibles cambios en la tabla de modificaciones. Con esto en mente, es un poco absurdo mantener una opción para editar artículos, por lo que la intención es vincular la creación de modificaciones con la edición de los artículos. Para ello, en la vista de los índices y de los detalles se ha sustituido el vínculo que llevaba a la edición de entradas de artículos por la de creación de modificaciones. Aunque los cambios más drásticos se los ha llevado el método *Create* que se encuentra en el controlador de modificaciones, el cual ahora recibe una variable identificador como parámetro. Si esta es nula realiza las acciones como las hacía antes de los cambios, pero si este tiene un valor numérico se realiza una consulta a la base de datos para acceder al texto y al identificador del artículo que se quiere modificar y se asignan a la nueva modificación. Esto lo que consigue es autocompletar los campos de texto e identificador del artículo para que no se tengan que añadir de cero, ya que por norma general se quiere modificar pequeñas partes del texto de un artículo para su modificación.

```
public IActionResult Create(int? id)
{
    Modificacion modificacion = null;
    if (id != null)
    {
        modificacion = RecogerDatosArticulo(id);
    } else
    {
        modificacion = new Modificacion();
        modificacion.Texto = "<p>Introduzca <stror
        modificacion.PendienteEva = true;
    }
    ListaDeArticulos();
    return View(modificacion);
}
```

```
private Modificacion RecogerDatosArticulo(int? id)
{
    var articuloTexto = from a in _context.Articulo
                        where a.ArticuloId == id
                        orderby a.ArticuloId
                        select a.Texto;

    var articuloId = from a in _context.Articulo
                         where a.ArticuloId == id
                         orderby a.ArticuloId
                         select a.ArticuloId;

    Modificacion modificacion = new Modificacion();
    modificacion.ArticuloId = articuloId.First();
    modificacion.Texto = articuloTexto.First();

    return modificacion;
}
```

Figura 3.9 y 3.10: *Modificación al método Create y RecogerDatosArticulo respectivamente*

3.4.4 Marcado de las diferencias.

Para facilitar la comparación del texto original de un artículo y su modificación se ha integrado un marcado de diferencias, donde el texto que se ha eliminado se marca en rojo y el añadido en verde. De esta manera se puede observar los cambios entre un texto y otro de forma sencilla. En un principio iba a ser una implementación propia, pero mientras se trabajaba en el algoritmo el nivel de complejidad del mismo aumentaba drásticamente y en ningún momento se alcanzaba el resultado esperado. Es por eso que se realizó una pequeña investigación y se llegó a la conclusión de que se tiene que hacer uso de la distancia de Levenshtein [34], un algoritmo que calcula el número mínimo de operaciones que hay que realizar para transformar una cadena en otra. Con este cálculo podemos saber que se ha añadido, modificado o eliminado. Para poder implementarlo en C# se ha seguido una entrada encontrada en Stack Overflow [35]. Este algoritmo recibe dos cadenas como entrada, donde la primera cadena recogerá la información de qué caracteres han variado. Para representar la cadena con las diferencias se tiene que iterar caracter por caracter y comprobar su estado.

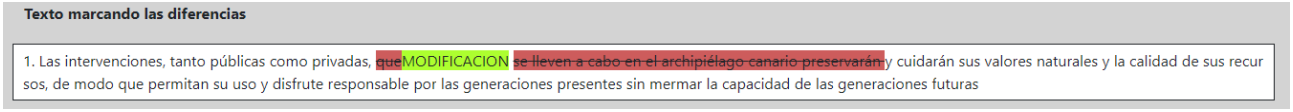


Figura 3.11: *Marcado de las diferencias*

3.4.5 Evaluación de las modificaciones.

La implementación de un sistema que permita evaluar las modificaciones es completamente necesario para la aplicación ya que de esta manera se consigue un mayor nivel de interacción entre las tablas y a la misma vez se alcanza a simular un caso de enmienda de un artículo, donde si alguna de las modificaciones es aceptada se debe de cambiar el contenido de un artículo por el de la modificación aceptada.

Para conseguir este resultado ha sido necesario crear una nueva vista para las modificaciones, la cual va a servir como ventana evaluadora. En ella se podrá observar el texto original y el texto modificado junto con el marcado de las diferencias, además de unos botones para poder aceptar o rechazar la modificación. Para poder interactuar con estos botones y con la vista se han creado tres nuevos métodos en el controlador de las modificaciones, uno de ellos para cargar los datos de la vista y otros dos para la acción de aceptar y rechazar. Además, se han añadido dos nuevas propiedades en el modelo de modificaciones, una para saber si una entrada ya ha sido evaluada y otra para saber si ha sido aceptada o rechazada. Con esto lo que se consigue es marcar una modificación como pendiente a evaluar en caso de que su valor sea *false*, en caso contrario, se comprobará el valor de la segunda propiedad, si esta es *true* querrá decir que ha sido aceptada y si es *false* no aceptada. También se han añadido avisos que bloquean la modificación hasta que el usuario no confirme su acción para evitar situaciones no deseadas, como es el cambiar el texto de un artículo por el de una modificación equivocada.

Por otro lado, se ha añadido otra propiedad al modelo de artículos para poder almacenar el texto original cuando este va a ser sustituido por el de una modificación. De esta manera se deja constancia de que un artículo ha sido modificado.

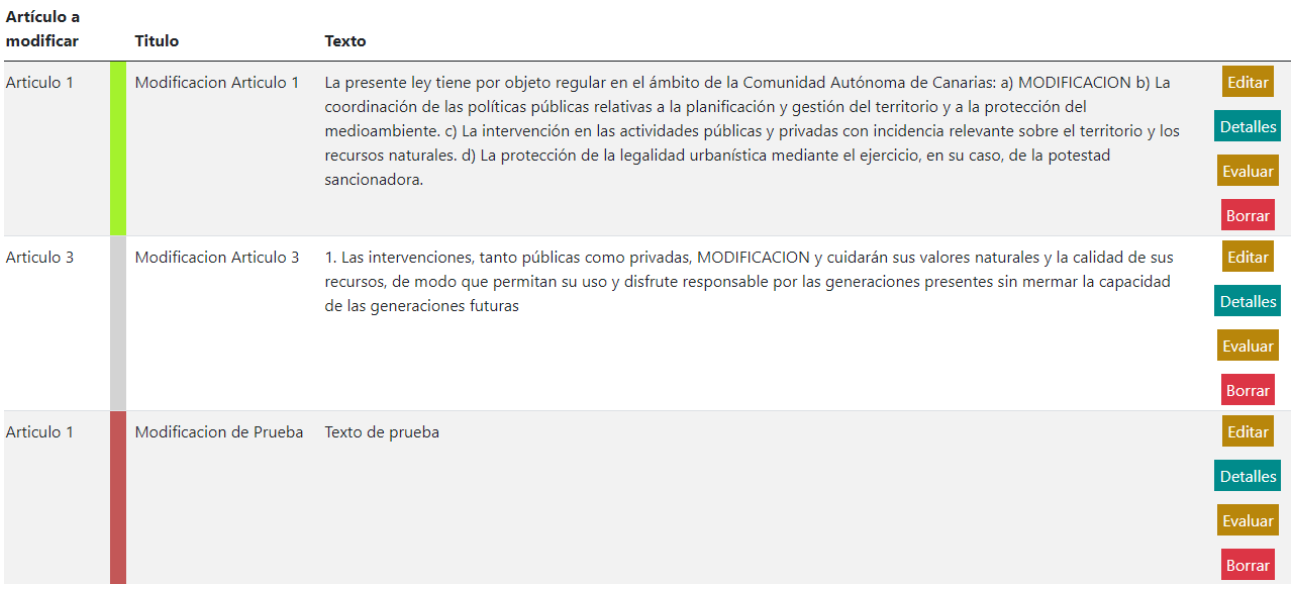


Figura 3.12: *Ejemplo de evaluación con primera entrada aceptada, segunda pendiente a evaluar y tercera rechazada.*

3.4.6 Funcionalidades menores.

Las siguientes funcionalidades son mayoritariamente para mejorar la experiencia de usuario, ya sea para facilitar el mostrado de la información o el manejo entre páginas.

- Lista de datos relacionados: Se trata de una lista que se ha añadido a la vista de detalles de leyes y artículos, donde se muestran, en el caso de las leyes, todos los

artículos relacionados con la ley y, en caso de los artículos, todas las modificaciones asociadas. Para ello en las vistas se consulta qué modificaciones tienen como uno de sus parámetros el identificador del artículo al que se está consultando y se muestran todos los resultados de esta consulta.

- Estado de los datos relacionados: También se muestra el estado de la modificación en la lista de datos de la vista de detalles de artículos. Como se comentó en uno de los puntos anteriores, el controlador es quien se encarga de comprobar el estado de las modificaciones y enviar esa información a las vistas. Para ello se usa el atributo *ViewBag* de .NET.
- Botón para volver a la página anterior: Este botón ha sido añadido en las páginas de detalles de todas las tablas. Esto se debe a que hay muchos datos interconectados y a veces no se accede a ellos a través del índice de las tablas, por lo que hay ocasiones donde el usuario prefiere volver a la página anterior antes que al índice de la tabla a la que está consultando los datos.
- Eliminación de las etiquetas del texto formateado: Al marcar las diferencias de dos textos es necesario eliminar las etiquetas de su formateo ya que en muchas ocasiones, si estas se dejan, marcará que muchas partes del texto han sido modificadas y no es el caso. El único inconveniente de esta acción es que una vez quitadas no es posible volver añadirlas, por lo que el marcado de las diferencias se mostrará sin formato.
- Rechazado de modificaciones asociadas a la modificación aceptada: Se da a entender que al aceptar una modificación el resto de modificaciones asociadas al artículo no van a ser aceptadas, por lo que se ha visto correcto realizar esta funcionalidad. Igualmente el rechazo de una modificación es prácticamente un cambio meramente visual en el resultado final ya que no tiene repercusión en el resto de datos y en cualquier momento se puede aceptar una modificación que ha sido previamente rechazada.

3.5 Integración de Quill

Hacer uso de la entrada de texto plano en el contenido de las tablas está bien cuando solamente se quiere añadir una pequeña porción de texto, como podría ser la de un título que normalmente son pocas palabras. Pero cuando se quieren introducir grandes cantidades de datos la tarea se vuelve mucho más complicada, sobre todo cuando se quiere dar formato a ese texto, ya que sin un procesador de textos como Quill se tendría que introducir a mano todas las etiquetas.

Para poder integrar Quill en el código primero hay que realizar una serie de pasos. En primer lugar hay que añadir las hojas de estilo y sus librerías. Para ello se tiene que añadir dos líneas de código al fichero *_Layout.cshtml*, que es un fichero conocido por todas las vistas del sistema, de esta manera se puede hacer uso de elementos compartidos entre vistas, como es el caso para la inicialización del editor.

Código para las hojas de estilo el cual debe de ir dentro de una cabecera.

- *<link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet">*

Código para agregar las librerías.

- *<script src="https://cdn.quilljs.com/1.3.6/quill.js"></script>*

Al iniciar el editor se le pueden asignar opciones a las barras de tareas al igual que un tema predefinido por los creadores. En la presente aplicación, se a elegido el tema llamado *'snow'* y se han agregado las siguientes opciones:

- Tamaño de los títulos.
- Tipo de letra.
- Estilo de letra.
- Cita en bloque.
- Alineación.
- Listas numeradas y sin numerar
- Sangría.
- Color de letra.
- Color de fondo.
- Limpiar formato.

Una vez iniciado sólo queda sustituir la entrada de texto plano por el procesador de texto. Para ello se ha agregado una etiqueta *div* con el identificador *'editor'*. Con esto lo que se consigue es llamar al script que inicializa el editor y mostrarlo por la vista, aunque sólo agregar esta etiqueta no es suficiente. El problema que surge es que todo el texto que se agrega al editor luego no es agregado a la propiedad de la tabla porque no están relacionados de ninguna manera. Para solventarlo hay que añadir un método al script que se usa para inicializar el editor. Este método recoge todo el contenido del editor y se lo asigna a un elemento con un determinado identificador, por lo que hay que dejar el antiguo campo que está relacionado con la propiedad de texto de la tabla y cambiarlo a oculto pero añadiendo el identificador que va a ser usado en el método del script. Por último, hay que llamar al método cada vez que se realiza un envío de información en la etiqueta del *form*. Como resultado se tiene un procesador de texto totalmente funcional e intuitivo de usar.

Figura 3.13: *Editor WYSIWYG implementado.*

Para finalizar, sólo queda mostrar el texto formateado en los detalles y en el índice de las tablas, para ello se hace uso del método *Raw* de *HtmlHelper* [36], el cual devuelve el texto formateado, esto es, sin la codificación.

Una vez ya implementado, se puede observar una notable mejoría respecto a como estaba desarrollado. El usuario tiene total libertad de formateo y manejo del texto con el único inconveniente de que el desarrollador tenga en cuenta las etiquetas que, aunque estas no se vean a simple vista, siguen existiendo y pueden llevar a resultados no deseados a la hora de trabajar con estos datos.

3.6 Diseño de las vistas

Se ha buscado que las vistas sean sencillas y accesibles, donde el número de opciones disponibles no abrumen al usuario mientras que la información que se proyecta sea clara y concisa. Además, se ha querido que en todo momento el usuario pueda acceder a las tablas, por lo que, haciendo uso del fichero *_Layout.cshtml*, en todas las páginas se encuentra una cabecera con los enlaces a todas las tablas. A continuación se podrá observar el diseño de todas las vistas de la aplicación, aunque, como el diseño de las tablas se repite, solamente se mostrarán el diseño del índice, página de creación, edición, aceptación y eliminación de las modificaciones. Además, la mayoría de las siguientes figuras están recortadas y reducidas para no ocupar mucho espacio en la memoria y por lo tanto no representan el resultado final de la aplicación.

3.6.1 Controlador de inicio.

Figura 3.14: *Página de inicio.*

La página de inicio sólo contiene una cabecera con el título del TFG, una pequeña descripción y unos botones que llevan a las distintas tablas y a una ventana de información. Esta ventana de información mayoritariamente está contenida de texto explicando los modelos y la tecnología usada.

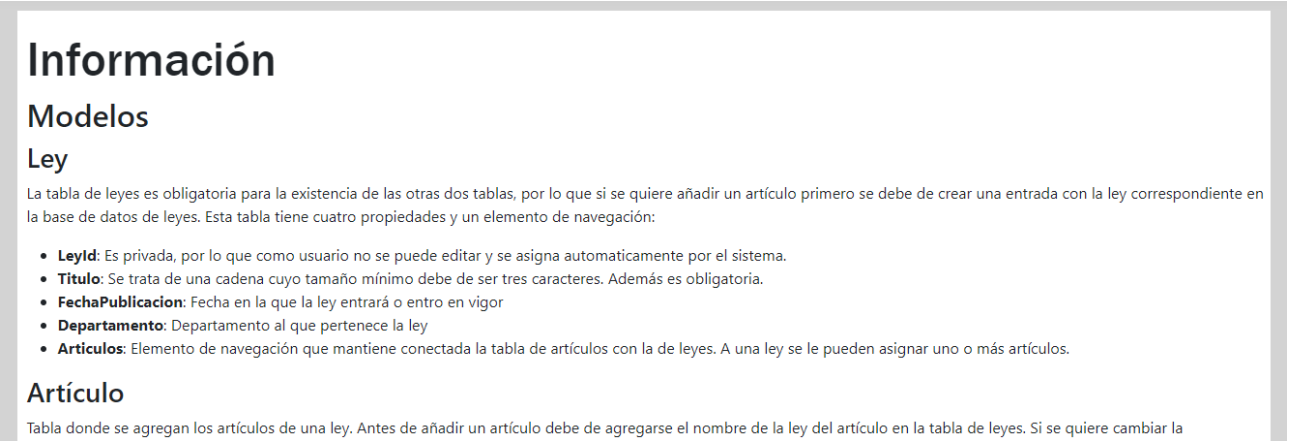


Figura 3.15: *Página de información.*

3.6.1 Controlador de tablas.

Todas las tablas tienen una vista para el índice, detalles, edición y borrado, mientras que la de modificación tiene una extra para la evaluación de una entrada.

Figura 3.16: *Índice de la tabla modificaciones.*

Estas páginas tienen el filtro de búsqueda que se comentó en el apartado de funcionalidades. Las tablas muestran partes de sus datos, siendo el caso de las modificaciones el título del artículo a modificar, su estado de aceptación, el título y el contenido respectivamente. Al lado se pueden observar los botones asociados a acciones para cada entrada de la tabla. Para añadir el contenido a la tabla se itera por el modelo de datos con un *foreach* ya que es posible gracias a las páginas Razor de ASP.NET [37] que permiten añadir código de C# en un fichero HTML.

Figura 3.17: *Creación de una modificación.*

Se puede acceder a la página de creación desde la propia tabla de modificaciones o desde la tabla de artículos, donde se autocompleta el texto con el artículo a modificar.

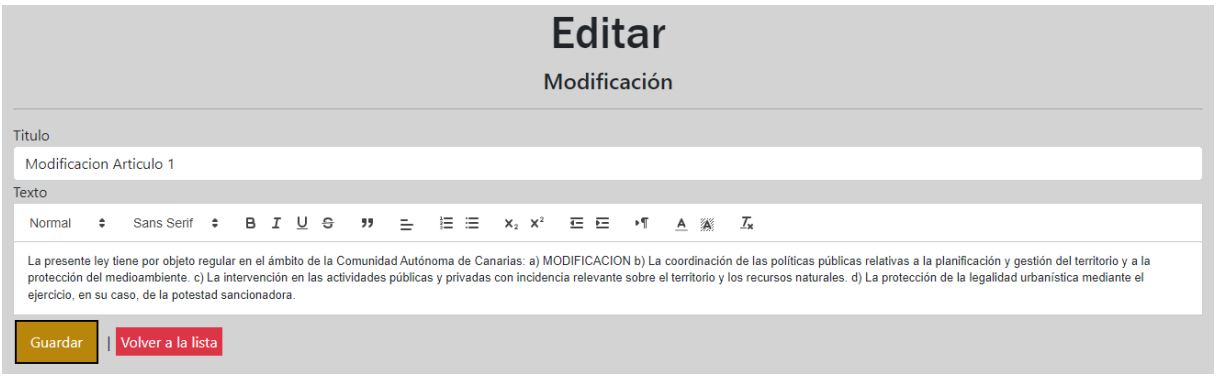


Figura 3.18: *Edición de una modificación.*

Los únicos valores editables de una modificación son su título y su contenido, por lo que en el formulario el resto de propiedades son ocultas para el usuario. Al darle al botón de guardar se cambia el contenido de la modificación, si es el de una modificación ya evaluada esta pasa a ser pendiente de evaluar.

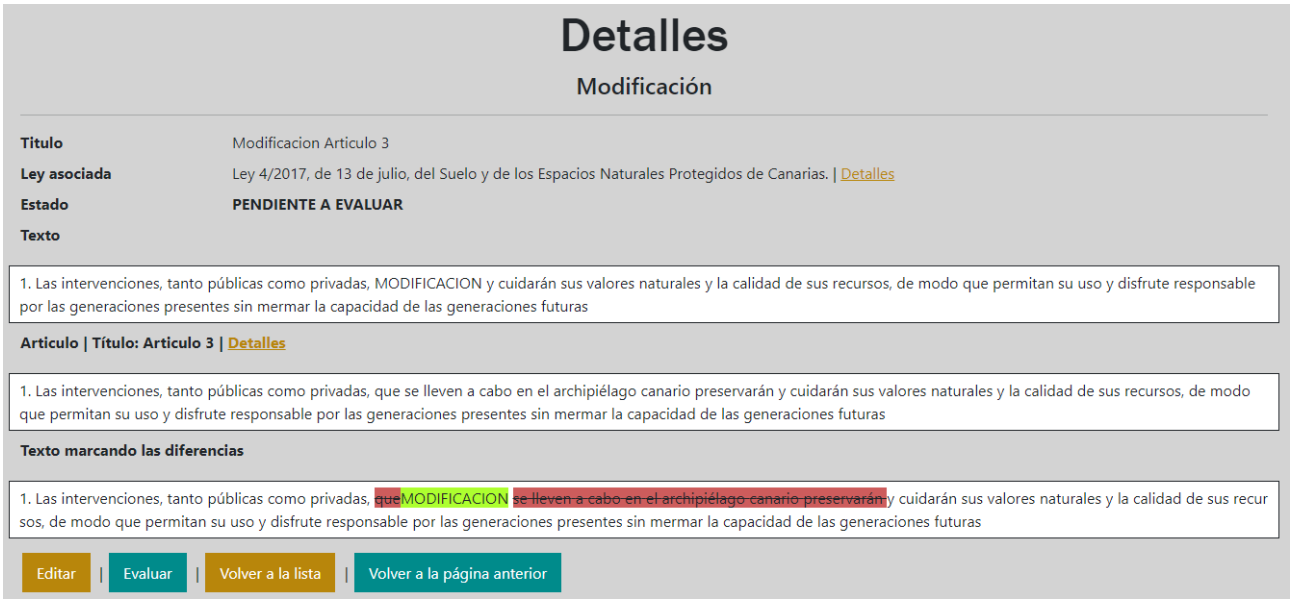


Figura 3.19: *Detalles de una modificación.*

En la vista de detalles se muestran todos los datos relacionados con la entrada que se quiere observar. En el caso de las modificaciones se muestra el título de la modificación, la ley asociada junto con un enlace a sus detalles, el estado de evaluación, el texto modificado, el título del artículo a modificar junto con un enlace a sus detalles y el texto original y un texto que marca las diferencias de los contenidos.

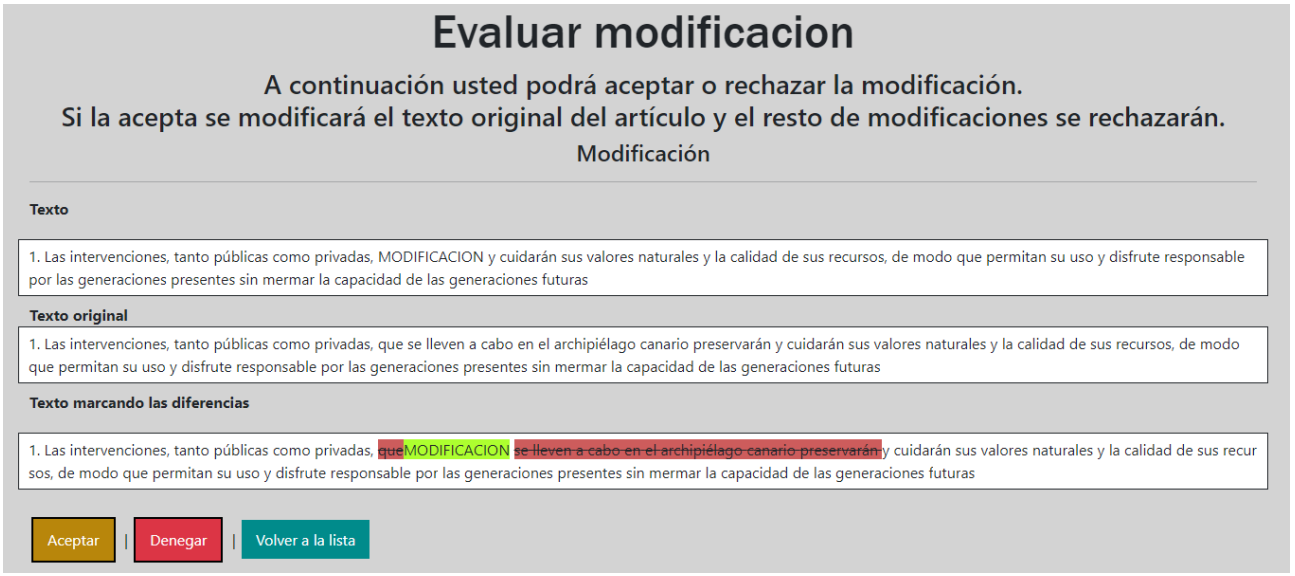


Figura 3.20: *Evaluación de una modificación.*

En la evaluación se muestra una aviso de las consecuencias que ocurrirán si se acepta una modificación además de volver a mostrar su contenido y el marcado de las diferencias. También se le vuelve a pedir confirmación al usuario a través de un mensaje por el navegador una vez se pulse el botón de aceptar o denegar.

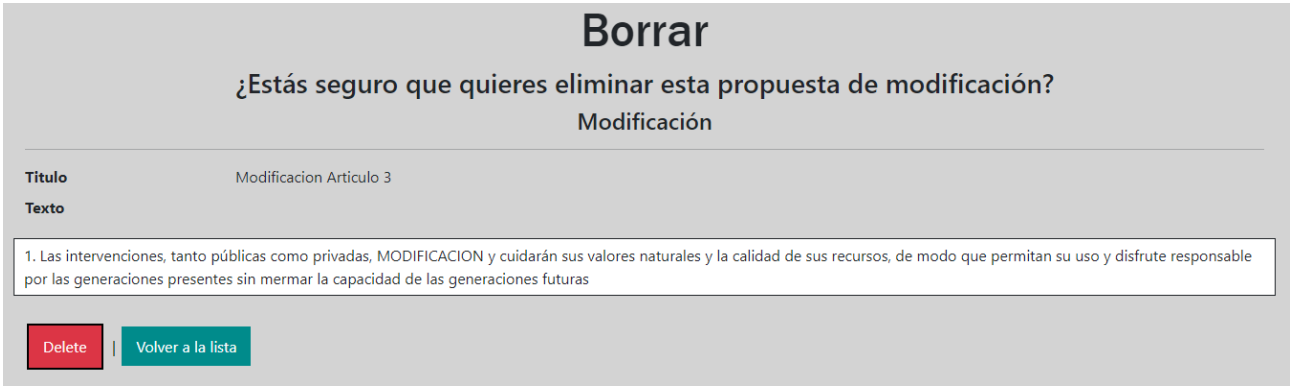


Figura 3.21: *Borrado de una modificación*

Al igual que con la ventana de evaluación, se avisa al usuario de las consecuencias del borrado, donde en la tabla de artículos se borran también todas las modificaciones asociadas y en la de leyes todos los artículos y modificaciones asociadas. Es por eso que también se pide confirmación al usuario de esta acción después de apretar el botón de borrado.

3.6.1 CSS

Por otro lado, para dar una imagen propia a la aplicación se han creado una serie de sentencias en CSS, un lenguaje de diseño gráfico. Estas sentencias han permitido diseñar las tablas de la base de datos y personalizar la ventana de inicio. También se ha hecho uso de Bootstrap [39] para agilizar el diseño y aplicar soluciones a problemas comunes a través de las opciones de sus librerías. Para agregar Bootstrap a la aplicación simplemente se tienen que añadir sus hojas de estilo en la cabecera de *_Layout.cshtml* y agregar sus librerías con las siguientes instrucciones.

- Hojas de estilo:

  <link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet">

- Agregar las librerías:

  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>

Para poder hacer uso de las sentencias implementadas en CSS, se añade el nombre de la sentencia al campo *class* de la etiqueta.

3.7 Diagrama de clases

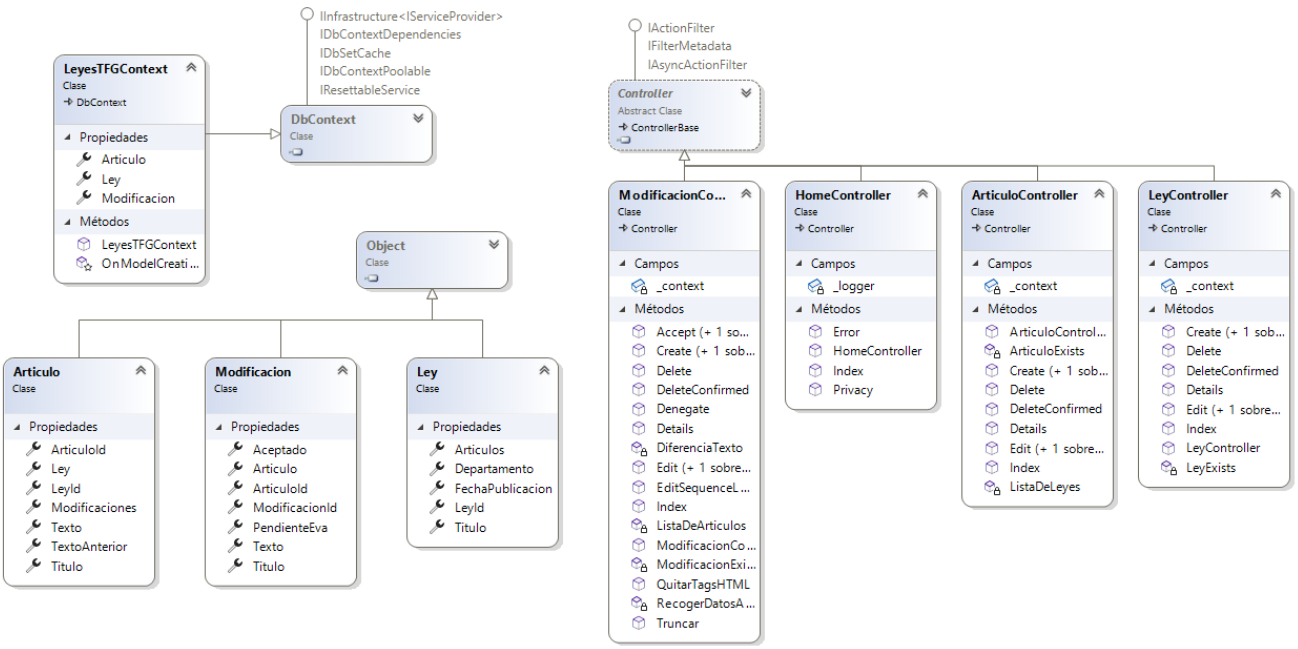


Figura 3.22: *Diagrama de clases*

LeyesTFGContext hereda los elementos de *DbContext*, clase añadida con los elementos de EF Core. Todos los controladores son implementaciones de la clase abstracta *Controller* y los modelos son simplemente objetos de C# con sus propiedades siendo atributos públicos de la clase, ya que así es como EF Core identifica las entidades. Respecto a los métodos de los controladores, modificación es quien tiene más métodos ya que es quien tiene más funcionalidades y, por lo tanto, quien tiene más interacciones con el usuario. Por otro lado, como se puede observar en el diagrama, las vistas no son añadidas ya que estas no son clases si no que páginas escritas en HTML.

3.8 Enlace a GitHub

En el siguiente enlace se puede encontrar el resultado del trabajo realizado, donde se puede observar con mucha más claridad la estructura del proyecto y el propio diagrama de clases, el cual se encuentra en la raíz de la solución.

https://github.com/AlvaroGonzalezRodriguez/LeyesTFG

Capítulo 4 Caso de estudio: Evaluación con leyes.

Una vez presentado las características e interfaz del aplicativo solo resta evaluar su funcionamiento añadiendo una ley real a la base de datos. Para el caso de estudio se han elegido los 5 primeros artículos de la Ley Orgánica 2/2006, de 3 de mayo, de Educación [38], donde se simulará que se quieren evaluar dos propuestas de enmiendas a cada artículo.

4.1 Introducción de la ley

El primer paso es añadir el título de la ley a la tabla de leyes. Para ello se tiene que ir a la pestañas de leyes a través de la página de inicio y pulsar el botón de “Nueva ley”. Una vez allí se introducen los valores de título, fecha de publicación y departamento. Al pulsar el botón de crear se tendrá como resultado una nueva entrada en la base de datos sin artículos asociados.

4.2 Agregar los artículos a modificar

Una vez agregado el título de la ley ya se pueden introducir los nuevos artículos a su tabla correspondiente. Desde la cabecera de la aplicación web se puede acceder a la sección de artículos, donde se deberá pulsar “Nuevo artículo”. Para crear los 5 artículos se tienen que introducir los datos uno a uno, siendo necesario indicar un título, texto o contenido y la ley asociada. Es importante recordar que una vez introducido el contenido del artículo, este no se puede editar a no ser que sea a través de una entrada aceptada de la tabla de modificaciones. Al finalizar, la página de detalles de la ley se verá de la siguiente manera:

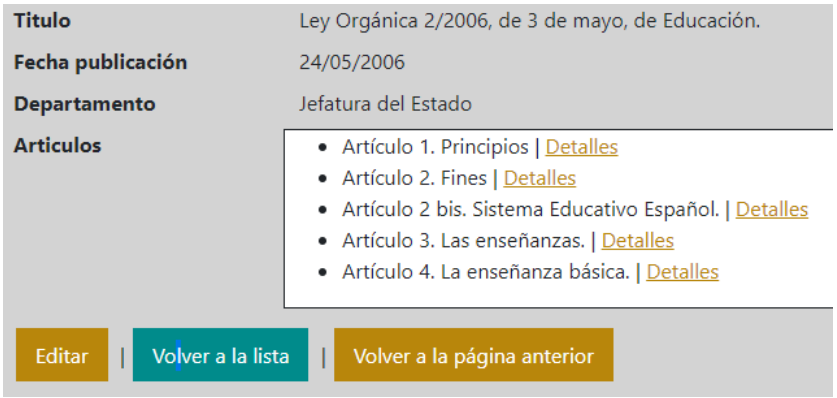


Figura 4.1: *Detalles de la ley de prueba.*

4.3 Agregar las enmiendas

La forma más óptima de agregar una modificación de un artículo es a través del botón “Modificar” que está disponible en todas las entradas de la tabla de artículos, ya que de esta manera se autocompleta los campos de la creación con el artículo a modificar. Se simulará que se insertan, eliminan y modifican distintas partes de los artículos. Una vez añadidas las modificaciones se puede observar en cualquiera de los artículos sus modificaciones asociadas y si estas están aceptadas o no.

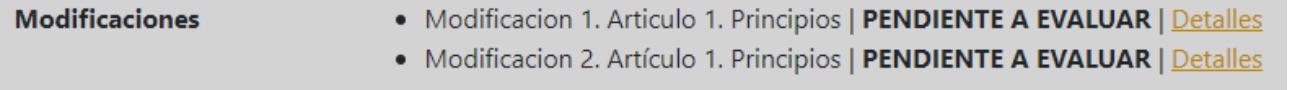


Figura 4.2: *Modificaciones asociadas al artículo 1.*

4.4 Evaluar las enmiendas y comportamientos de las mismas.

Para el presente caso de prueba, se ha seleccionado como aceptada la primera modificación de cada artículo a excepción del artículo 2, donde no se ha aceptado ninguna de ellas. Al pulsar el botón “Evaluar” dentro de la tabla de modificaciones se inicia el proceso de evaluación, donde se podrá comparar los dos textos con sus respectivas diferencias. En el caso del artículo 2, será necesario rechazar manualmente las dos modificaciones para que se vea reflejado en la pestaña de detalles que ninguna de las dos ha sido aceptada, mientras que para el resto de artículos sólo hace falta aceptar una de las modificaciones para que la otra sea rechazada automáticamente.

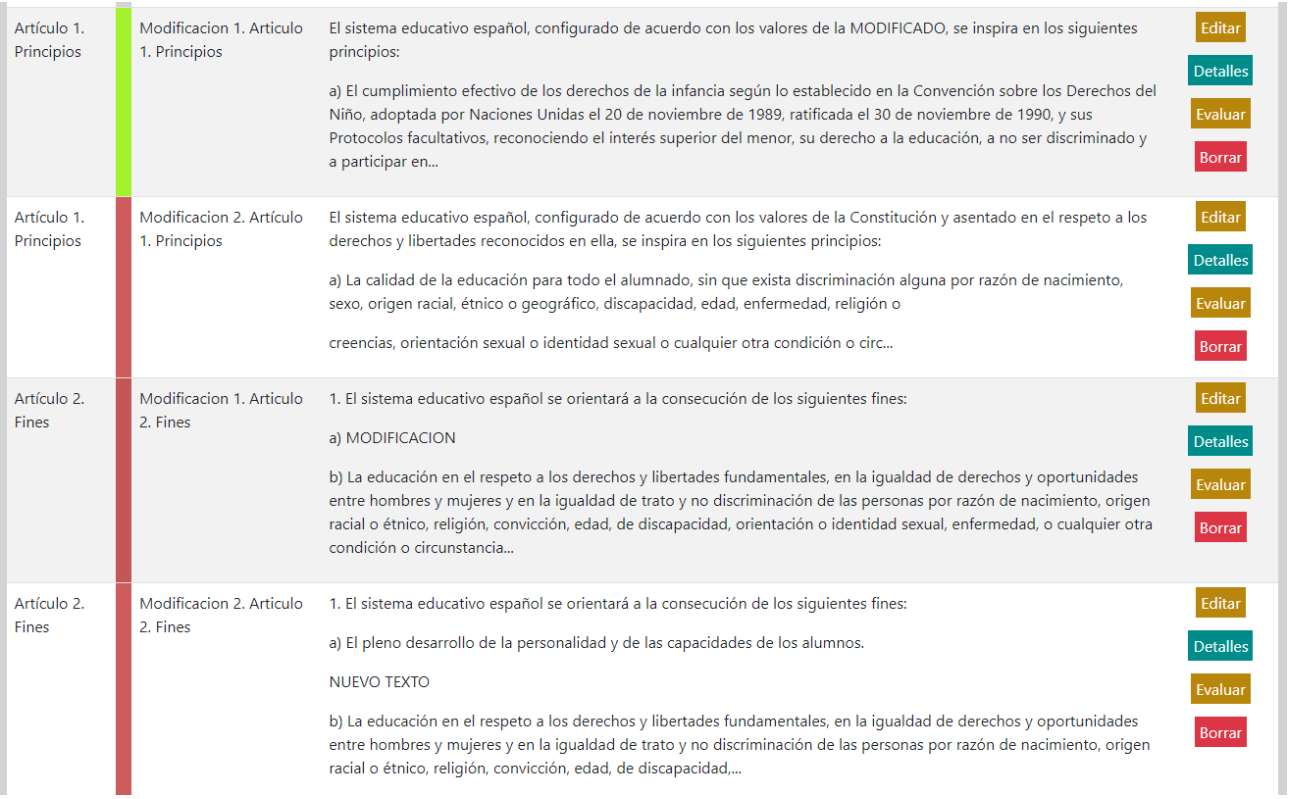


Figura 4.3: *Modificaciones aceptadas y rechazadas*

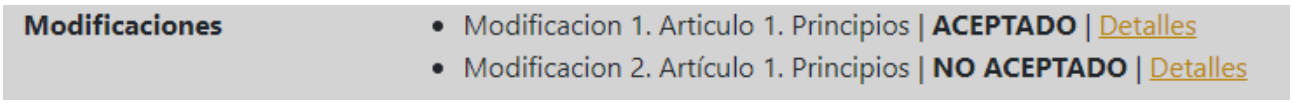


Figura 4.4: *Los detalles del artículo 1 han cambiado.*

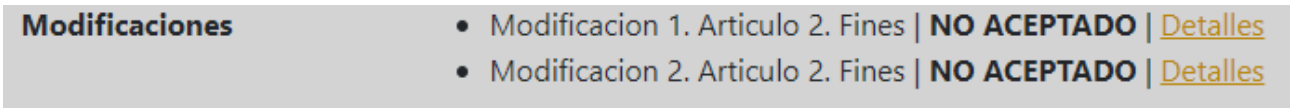


Figura 4.5: *Los detalles del artículo 2 no han cambiado.*

Como se puede observar, el uso de la aplicación es bastante intuitivo. También cabe destacar que si un artículo es modificado no quiere decir que ya no pueda volver a ser editado. Si se acepta otra modificación sobre el mismo artículo se volverán a sobreescribir los datos. Además, si una modificación aceptada es borrada de la base de datos no afecta en absoluto al artículo que modifica, ya que sus datos ya han sido sobreescritos.

Capítulo 5 Conclusiones y líneas futuras

Como resultado del trabajo realizado se ha desarrollado una aplicación web que maneja una base de datos que facilita el proceso de enmiendas de los artículos de una ley de manera totalmente digital, haciendo uso del framework ASP.NET Core. Como resumen final, estas son las tres principales características que definen al sistema:

- Manejo de una tabla relacionada con leyes.
- Manejo de una tabla relacionada con artículos de leyes.
- Manejo de una tabla relacionada con modificaciones de artículos.

Haciendo uso del aplicativo, el usuario se puede dar cuenta de lo rápido y cómodo que es trabajar en un entorno digital y, por tanto, abandonar los métodos tradicionales en lo que respecta al proceso de enmiendas de los artículos.

Ha sido la primera vez que hago uso de este framework, que se me fue introducido en el desarrollo de las prácticas externas y me pareció que su uso era la mejor opción, sobre todo al descubrir que gracias a EF Core se puede implementar una base de datos de una manera tan sencilla y útil.

El desarrollo del presente sistema me ha permitido adquirir nuevos conocimientos en el desarrollo de aplicaciones web, tanto en el back-end como en el front-end. También ha sido un reto, ya que nunca había trabajado una aplicación de tal envergadura, donde hay que diseñar una propia base de datos y una interfaz que no sea muy complicada de usar mientras se mantienen todas las funcionalidades. Además, nunca había trabajado con CSS ni con C#, por lo que este primer acercamiento ha sido una experiencia totalmente nueva y gratificante.

Respecto a las líneas futuras, si el proyecto tuviera éxito, se podría considerar nuevas características y funcionalidades:

- Sistema de usuarios: Un sistema de usuarios es ideal para comprobar y mantener un seguimiento de quién crea y edita entradas en una base de datos. Este sistema de usuarios también podría tener un sistema de chats entre usuarios para poder tener una mejor coordinación y comunicación. Además, se implementaría una serie de roles que limiten acciones de usuarios, como evitar que usuarios puedan evaluar modificaciones o evitar que las añadan a la tabla.
- Posibilidad de añadir comentarios: En lo que respecta a la evaluación de las modificaciones, los comentarios podrían ayudar a otros usuarios saber porque se ha aceptado o rechazado una modificación. Estos comentarios se mostrarían una vez se ha evaluado la modificación en la pestaña de detalles de artículos y modificaciones.

- Posibilidad de descargar el contenido de los artículos: Añadir una opción para descargar en formato pdf el contenido de los artículos, ya sea el contenido individual de un artículo o todos los artículos relacionados con una misma ley.
- Añadir sección para evaluar el sistema: Posibilidad de que un usuario pueda realizar una encuesta para evaluar el sistema y que sirva como retroalimentación para los desarrolladores.
- Soporte en inglés: Dar la posibilidad de tener la interfaz traducida en inglés y de añadir un traductor que ayude a cambiar el contenido de un artículo al inglés, pero que este no sobrescriba los datos si no que sea solamente un cambio visual.

Capítulo 6 Summary and Conclusions

As a result of the current work, a web application has been developed. It manages a database that helps in the process of amending the articles of a law in a totally digital way, using the ASP.NET Core framework. As a summary, these are the three main characteristics that define the system.

- Management of a table related to laws.
- Management of a table related to articles.
- Management of a table related to article modifications.

By using the application, the user can realize how fast and comfortable it is to work in a digital environment and, therefore, they will stop using traditional methods regarding the article amendment process.

It has been the first time that I have used this framework, which was introduced to me in the development of external externships. It seemed to me that using .NET was the best option, especially when I discovered that thanks to EF Core, a database can be implemented in such an easy and useful way.

The development of this system has allowed me to acquire new knowledge in the development of web applications, both in the back-end and front-end. It has also been a challenge, since I had never worked on an application with this magnitude, where you have to design your own database and an interface that is not very complicated to use while maintaining all the functionality. Also, I had never worked with CSS or C#, so this first approach has been a completely new and rewarding experience.

Regarding future lines, if the project was successful, new features and functionalities could be considered:

- User system: A user system is ideal for checking and tracking who creates and edits entries in the database. This user system could also have a chat system between users to have better coordination and communication. In addition, users could have roles that limit user actions, such as preventing users from evaluating changes or preventing them from adding entries to the table.
- Possibility of adding comments: Regarding the evaluation of the modifications, the comments could help other users to know why a modification has been accepted or rejected. These comments would be displayed once the modification has been evaluated in the articles and modifications details tab.
- Possibility of downloading the content of the articles: Add an option to download the content of the articles in pdf, either the individual content of an article or all the articles related to the same law.

- Add section to evaluate the system: Possibility that a user can carry out a test to evaluate the system and then the developers can use the information as feedback.
- Support in English: Give the possibility of having the interface translated into English and adding a translator that helps to change the content of an article to English, but does not overwrite the data but is only a visual change.

Capítulo 7 Presupuesto

A continuación se presenta una estimación del presupuesto necesario estimado para la realización del presente proyecto.

Tarea	Tiempo	Coste	Coste final
Estudio previo de editores WYSIWYG y tecnología a usar	25 horas	15€/hora	375€
Elección de editor y tecnologías adecuadas	10 horas	15€/hora	150€
Diseño de la base de datos	5 horas	20€/hora	100€
Desarrollo de la aplicación	175 horas	20€/hora	3500€
Evaluación experimental	10 horas	15€/hora	150€
TOTAL			4275€

Tabla 7.1: *Presupuesto*

Como se observa en la tabla, el presupuesto final es de 4275€ suponiendo que sólo un programador está a cargo del mismo, con un total de 225 horas.

Bibliografía

[1] Digitalización. url: https://es.wikipedia.org/wiki/Digitalizaci%C3%B3n

[2] Editor WYSIWYG. url: https://es.wikipedia.org/wiki/WYSIWYG

[3] ASP.NET Core.

url:https://docs.microsoft.com/es-es/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

[4] Entity Framework Core. url: https://docs.microsoft.com/es-es/ef/core/

[5] Quill - Delta. url: https://quilljs.com/docs/delta/

[6] CKEditor. url: https://ckeditor.com/

[7] TinyMCE. url: https://www.tiny.cloud/

[8] Summernote. url: https://summernote.org/

[9] Quill. url: https://quilljs.com/

[10] SCEditor. url: https://www.sceditor.com/

[11] Jodit. url: https://xdsoft.net/jodit/

[12] Froala. url: https://froala.com/wysiwyg-editor/docs/

[13] Aloha Editor. url: https://www.alohaeditor.org/guides/

[14] Medium Editor. url: https://github.com/yabwe/medium-editor/wiki

[15] Trumbowyg. url: https://alex-d.github.io/Trumbowyg/

[16] Bravo. url: https://en.wikipedia.org/wiki/Bravo_(editor)

[17] BRUNO. url: https://en.wikipedia.org/wiki/Bruno_(software)

[18] WordStar. url: https://en.wikipedia.org/wiki/WordStar

[19] WordPerfect. url: https://www.wordperfect.com/en/

[20] Microsoft Word. url: https://www.microsoft.com/es-es/microsoft-365/word

[21] Microsoft Office. url: https://www.office.com/

[22] LibreOffice. url: https://es.libreoffice.org/

[23] LibreOffice Writer. url: https://en.wikipedia.org/wiki/LibreOffice_Writer

[24] NuGet. url: https://docs.microsoft.com/en-us/nuget/what-is-nuget

[25] Visual Studio. url: https://visualstudio.microsoft.com/es/

[26] El concepto de IDE. url: https://www.redhat.com/es/topics/middleware/what-is-ide

[27] Modelo-Vista-Controlador.

url: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[28] Microsoft SQL Server. url: https://es.wikipedia.org/wiki/Microsoft_SQL_Server

[29] EF Core paquetes NuGet.

url:https://docs.microsoft.com/es-es/ef/core/what-is-new/nuget-packages

[30] MYSQL Workbench. url: https://www.mysql.com/products/workbench/

[31] DataAnnotations.
url:https://docs.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/models-data/validation-with-the-data-annotation-validators-cs

[32] Migraciones EF Core.

url: https://docs.microsoft.com/es-es/ef/core/managing-schemas/migrations/?tabs=vs

[33] Language Integrated Query (LINQ).
url:https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/

[34] Distancia de Levenshtein.

url:https://www.wikiwand.com/es/Distancia_de_Levenshtein#:~:text=La%20distancia%20de%20Levenshtein%2C%20distancia,y%20ciencias%20de%20la%20computaci%C3%B3n.

[35] Implementación de distancia de Levenshtein.

url: https://stackoverflow.com/questions/52162091/detect-differences-between-two-strings

[36] HtmlHelper.Raw Método.
url:https://docs.microsoft.com/es-es/dotnet/api/system.web.mvc.htmlhelper.raw?view=aspnet-mvc-5.2

[37] Páginas Razor.
url:https://docs.microsoft.com/es-es/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-6.0&tabs=visual-studio

[38] Ley Orgánica 2/2006, de 3 de mayo, de Educación

url:https://www.boe.es/buscar/pdf/2006/BOE-A-2006-7899-consolidado.pdf

[39] Bootstrap. url: https://getbootstrap.com/