Grado en Ingeniería Informática

Trabajo final de grado

Implementación de módulo de gestión documental con funcionalidad de firma digitalizada

Realizado por:
Miguel Martínez Amat

Supervisado por:
Juan Bautista García Traver
Tutorizado por:
Filiberto Pla Bañón

Fecha de lectura: 15 de Septiembre de 2021

Curso académico 2020 /2021

Resumen

En este documento se describe el análisis, desarrollo e implementación del proyecto de final de grado de Ingeniería Informática, el cual se ha desarrollado durante las 300 horas de la estancia en prácticas en la empresa Irene Solutions SL, localizada en Burriana.

Este proyecto consiste en mejorar el software de firma de documentos PDF, ya implementado en la empresa, para ofrecer una mayor funcionalidad y una visualización más adecuada para distintos dispositivos.

Para ello, se ha implementado un gestor documental de documentos PDF, donde los usuarios pueden firmar y acceder a sus documentos.

Antes de la realización de este trabajo, se ha estudiado las herramientas que emplea la empresa, así como el entorno de la misma.

Con la implementación de las mejoras detalladas en el documento se ha conseguido mejorar la gestión de los documentos que se van a firmar, ya que inicialmente no se disponía de un gestor documental.

Palabras clave

Documentos PDF, Librería Signature Pad, Librería iText, Librería PDF.js, HTML, CSS, JavaScript.

Keywords

PDF Documents, Signature Pad Library, iText Library, PDF.js Library, HTML, CSS, JavaScript.

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

El proyecto propuesto, presentado en este documento, se realiza en el marco de las prácticas enfocadas en el Proyecto Final de Grado de Ingeniería Informática. La motivación de estas prácticas se basan en aplicar los conocimientos obtenidos durante la realización del grado de Ingeniería Informática, poniendo énfasis en los conocimientos del itinerario de Tecnologías de la Información.

Este proyecto se ha llevado a cabo en la empresa Irene Solutions SL bajo la supervisión de Juan Bautista García Traver. Esta empresa presenta dos frentes de trabajo, dentro del sector de la gestión documental para la pequeña y la gran empresa. Por una parte, trata la gestión documental desde el punto de vista de la facturación y la contabilidad, teniendo como clientes tanto autónomos como empresas. Por otra parte, nos encontramos con el desarrollo e implantación de distintos programas enfocados a agilizar y, hasta cierto punto, automatizar este tipo de trabajo con enfoque hacia la gran empresa. Gran parte del abanico de estos programas está desarrollado bajo licencias públicas de código abierto GNU AFFERO v3[13].

Concretamente, el proyecto se centrará en el uso de la librería Signature Pad[1] y la librería PDF.js[2]. La librería Signature Pad[1], permite dibujar una serie de puntos, unidos por una línea que imita el trazado humano mediante el algoritmo de renderizado de Béizer. Por otro lado, para poder mostrar el documento PDF al usuario para elegir la página y poder trazar la firma posteriormente, sobre una sección de éste, se emplea la librería de Mozilla, PDF.js[2].

Para desarrollar el proyecto se analizará el entorno, las propuestas de mejora, el desarrollo, la implementación y puesta en marcha de los resultados obtenidos. En la estancia en prácticas se permite implantar y estudiar mejoras a un software en un ámbito real, lo que es de gran interés de cara al futuro.

1.2. Objetivos del proyecto

El objetivo de este proyecto es proponer e implementar mejoras al software de firma de documentos PDF, ya implementado en la empresa Irene Solutions S.L., para

ofrecer una mayor funcionalidad y una visualización más adecuada para distintos dispositivos.

El software deberá ser estudiado y analizado, con el fin de entender el funcionamiento, así como las herramientas y las bibliotecas de software que serán necesarias para el desarrollo y posterior ejecución del proyecto. Después de la finalización del estudio y análisis, se procederá a la identificación de posibles mejoras para ampliar la eficiencia y funcionalidad del producto final.

A continuación, se deberá empezar con la implementación y el desarrollo de las nuevas mejoras, así como evaluar las mismas. La finalidad de evaluar las mejoras viene dada por la posibilidad de que alguna de ellas entre en conflicto y para implementar únicamente aquellas que comportan una mejora evidente.

El objetivo principal se puede descomponer en los siguientes cuatro sub-objetivos:

- Implementar un panel de firma adaptable al dispositivo (interfaz responsive), indistintamente de si es web (ordenador) o PWA[3] (móvil).
- Desarrollar un gestor documental que permita al usuario comprobar los documentos subidos y los firmados.
- Mejorar la estructura del código para hacerlo más accesible.
- Preservar la mayor cantidad de información posible sobre el firmante, para demostrar la veracidad de la firma.

1.3. Estructura de la memoria

La memoria ha sido dividida en las partes que se muestran a continuación:

- En el primer capítulo se expone la motivación y los objetivos que impulsaron la elección de las prácticas para la realización del proyecto.
- El segundo capítulo realiza una descripción general del proyecto, de la empresa donde se realizan las prácticas y de las herramientas y metodologías empleadas para estudiar y mejorar el software actual.
- Describe la planificación del proyecto, entre las que se encuentran la metodología, recursos y costes, así como el seguimiento efectuado.
- Detalla el análisis que se ha realizado sobre el software del proyecto; además de una explicación detallada de la arquitectura utilizada para el desarrollo y la interfaz gráfica que se ha elegido.
- Expone el desarrollo seguido durante la implementación del proyecto, así como las valoraciones.
- Presentan las conclusiones. Se detallan la valoración del periodo en la estancia, tanto de la empresa como del proyecto, y una valoración personal.

Capítulo 2

Descripción del proyecto

2.1. Escenario inicial

La empresa donde se va a desarrollar el proyecto de prácticas es Irene Solutions S.L. Esta empresa, creada en 2014, se dedica a la gestión documental . Por un lado, se encarga de la facturación de la gestión documental y por otro lado, se encarga del desarrollo de aplicaciones que apoyan la gestión documental, proporcionando servicios de automatización contable y gestión de clientes. La empresa cuenta con 3 empleados: Jesús Juárez Ortíz, administrativo enfocado en la gestión documental y el trato directo con autónomos y empresas, y Manuel Diago García y Juan Bautista García Traver, desarrolladores del software utilizado en la empresa, siendo el segundo de ellos el supervisor de la estancia en prácticas.

Durante este proyecto se mejorará un software existente en la empresa. Este software se desarrolla en parte por la librería Signature Pad[1] y la librería PDF.js[2] (ambas con el código abierto disponible en GitHub) y otras herramientas complementarias para la creación de aplicaciones web (HTML ,CSS, JavaScript, etc).

La librería Signature Pad [1] permite dibujar una serie de puntos, unidos por una línea que imita el trazado humano mediante el algoritmo de Bézier. Estos puntos se guardan como una imagen y luego en la parte del servidor se insertan en el PDF mediante el uso de iText. La biblioteca de iTextSharp 5[7] está desarrollada en .NET por Irene Solutions S.L. y distribuida mediante la licencia Affero General Public License[13].

La librería PDF.js [2] permite visualizar el documento PDF mediante HTML5, lo que permite al usuario recorrer el documento para elegir la zona sobre la que quiere realizar la firma. El cometido del alumno es desarrollar una aplicación que permita subir documentos PDF para su visualización y firma.

El funcionamiento del software desarrollado se puede dividir en tres partes diferenciadas que se complementan entre sí.

- La primera parte es el gestor documental. Es la página principal del software y es la que permite al usuario poder subir los documentos PDF que desea poder firmar. Este gestor también permite almacenar los documentos PDFs y clasificarlos (subidos y firmados). Entre las funciones del gestor documental destacan la previsualización y “enviar a firmar” que son las opciones que se conectan con las otras funciones del

software. Las otras funciones se encargan de cambiar el nombre y borrar los documentos subidos.

- La segunda parte del software es la previsualización. Es la página encargada de visualizar el documento PDF y, en caso de estar visualizando en dispositivos táctiles, permite ampliarlo para ver el documento más de cerca. Esta previsualización tiene dos modalidades, una en la que solo se visualiza y una en la que se visualiza para enviar a firmar. La diferencia entre ambas es que en la opción de la firma aparece un botón que permite enviar para firmar. En esta segunda opción se puede enviar a firmar la página que se desee posicionándose encima de la página y pulsando el botón firmar.

- Finalmente la tercera parte es la firma. Es la página donde se permite realizar la firma, guardarla y descargarla. La página seleccionada en la parte anterior se envía a la firma para poder seleccionar la zona donde se quiere firmar. Para firmar, se debe hacer un doble click sobre la zona deseada y, una vez aparece el modal de la firma, usar el ratón (ordenador) o el dedo (dispositivos táctiles) para firmar. Una vez firmado se puede ver una vista previa antes de firmar definitivamente y posteriormente también es posible descargar el documento con la firma.

También hay que tener en cuenta que este software está pensado para que sea utilizado por personas con identificador, es decir, usuarios registrados. Esto se debe a que si se quiere tener veracidad en la firma hay que almacenar los mayores datos posibles.

Por ello se ha implementado, a parte de estas tres partes principales, un panel de inicio de sesión, que permite realizar las acciones necesarias para la gestión de usuarios: inicio de sesión, registro y cambio de contraseña.

El alcance de este sistema incluye desde la gestión de documentos en formato PDF mediante el gestor documental hasta la firma de los mismos.

Para disponer de un entorno adecuado de pruebas, se ha puesto a mi disposición una copia de la base de datos perteneciente a la empresa pero sin datos, para poder hacer inserciones en la base de datos de forma real sin comprometer la confidencialidad de los datos de la empresa.

2.2 Descripción de las tecnologías y herramientas

En este apartado se describe el software empleado en el desarrollo del proyecto.

- **Microsoft Visual Studio Code [3]**

  Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. Es gratuito y de código abierto.

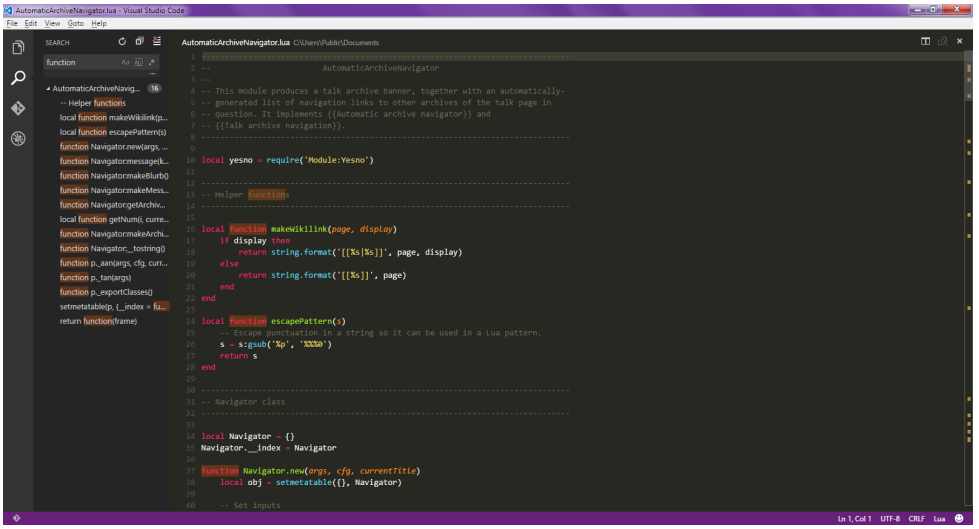

Figura 2.1. Editor de código fuente Visual Studio Code

  Es un editor de código fuente que soporta HTML, CSS y JavaScript entre otros, por lo que es perfecto para desarrollar el proyecto. Además, ha sido el editor utilizado durante una asignatura del curso por lo que ya tenía conocimientos previos.

- **Microsoft Visual Community 2019 [4]**

  Microsoft Visual Studio es un entorno de desarrollo integrado para Windows y macOS. Es compatible con múltiples lenguajes de programación, tales como C#, .NET, etc., al igual que entornos de desarrollo web, como ASP.NET.

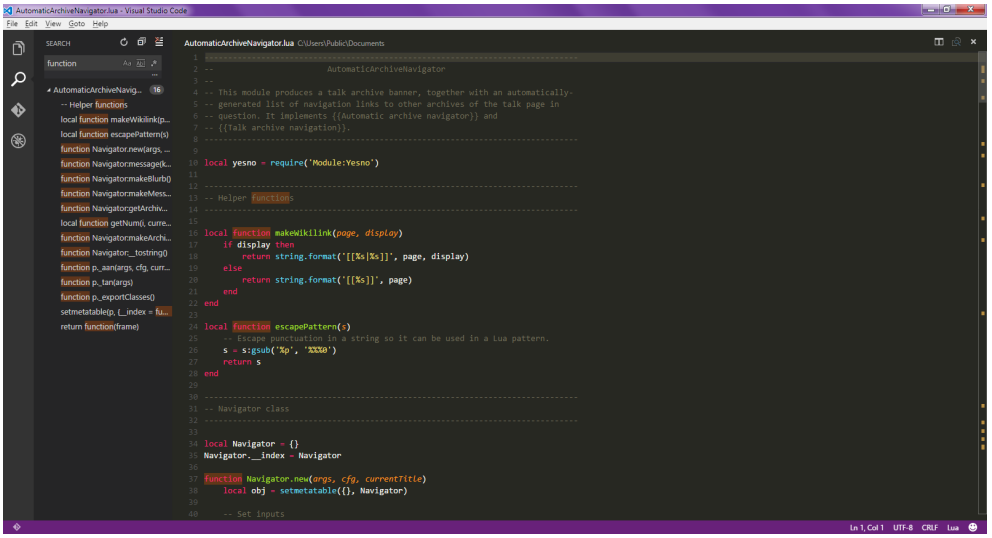

Figura 2.2 IDE de Visual Studio 2019: software de programación para Windows

La elección de este IDE se debe a las directrices del supervisor, ya que es el IDE utilizado en la empresa. Se usa este IDE debido a que soporta el lenguaje de programación C# y es el utilizado para el desarrollo del proyecto.

- **Signature Pad [1]**

  Signature Pad es una biblioteca de JavaScript para dibujar firmas. Está basado en un "lienzo" (canvas) HTML5 y utiliza la interpolación de curvas Bézier de ancho variable. Funciona en todos los navegadores de escritorio y móviles modernos y no depende de ninguna biblioteca externa.

- **PDF.js [2]**

  PDF.js es un visor de documento PDF creado con HTML5. Está impulsado por la comunidad y es compatible con Mozilla. Su objetivo es crear una plataforma basada en estándares web de uso general para analizar y representar archivos PDF.

- **iTextSharp 5 [7]**

  Biblioteca Open Source desarrollada en .NET por Bruno Lowagie, entre otros, y distribuida mediante la licencia Affero General Public License.

  Facilita la creación y manipulación de archivos PDF. Se trata de la variante de iText 5 de JAVA.

- **HTML**

  Lenguaje de marcado de hipertexto, hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia

del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Este lenguaje es empleado en el frontend del software.

- **CSS**

  Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML. Es empleado en el frontend del software para darle el estilo al HTML.

- **JavaScript**

  Es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas Es empleado en el frontend del software para darle la funcionalidad al HTML y también se encarga de la conexión con el backend.

- **C#**

  Lenguaje de programación orientado a objetos, realizado por Microsoft para el desarrollo de aplicaciones sobre la plataforma .NET. Consta de una sintaxis similar al lenguaje JAVA. Este lenguaje es empleado en el backend del software.

- **MariaDB [5]**

  MariaDB es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL (General Public License). Tiene una alta compatibilidad con MySQL ya que posee las mismas órdenes, interfaces, API y bibliotecas, siendo su objetivo poder cambiar un servidor por otro directamente.

- **HeidiSQL [6]**

  Es un software libre y de código abierto que permite conectarse a servidores MySQL (y sus derivaciones como MariaDB y Percona Server), así como Microsoft SQL Server y PostgreSQL. Sus características permiten realizar las operaciones de base de datos más comunes y avanzadas, sin embargo aún sigue en desarrollo a fin de integrar la máxima funcionalidad que se espera en una interfaz de base de datos de SQL.

2.3 Control de versiones

En este proyecto se han utilizado copias de seguridad para llevar un control de las modificaciones que se han ido realizando sobre el proyecto. Todas las copias se han comprimido en zip y se han subido a Drive.

Por otra parte, para el proceso de aprendizaje, también se han mantenido diferentes versiones en el propio proyecto para poder realizar pruebas y ejecutar distintos desarrollos a fin de mejorar el producto final.

Llevar un control sobre el trabajo realizado en cada versión del producto es parte importante en cualquier proyecto de desarrollo de software. El motivo por el que he usado este método de control de versiones, en vez de Github que es la mejor opción, es debido a que el proyecto desarrollado es un módulo independiente y en principio no va a ser compartido con otro personal de la empresa.

Capítulo 3

Planificación del proyecto

En este capítulo se presenta la metodología empleada en el proyecto, la planificación, los recursos y los costes. Así como un seguimiento del propio proyecto.

3.1 Metodología

En relación a la metodología de trabajo empleada, en este programa se aborda una metodología predictiva. Esta metodología permite crear un producto dentro un marco de tiempo específico mediante el seguimiento de objetivos claros, previamente identificados, y desarrollados hasta su consecución. Como el proyecto se ha desarrollado únicamente por una persona, el alumno en prácticas, no requiere de reuniones de equipo.

Los pasos que presenta la metodología predictiva son:

- Planificación: Estimación de la planificación y los recursos a utilizar, así como desarrollar el plan de proyecto.
- Ejecución: Llevar a cabo la planificación establecida en la etapa anterior, realizando un seguimiento y evaluaciones regulares sobre el desarrollo del proyecto.
- Control de calidad: Revisión del desarrollo para minimizar los posibles errores presentes realizando todos los cambios necesarios.
- Cierre: Documentación del proyecto desarrollado y su consiguiente puesta en marcha en caso necesario.

3.2 Planificación

La planificación del proyecto se ha realizado por el desglose de tareas mostrado a continuación que describe las tareas y sus diferentes fases en las que se divide el proyecto, así como la duración y sus dependencias. En esta planificación, se encuentran 5 grandes tareas divididas por subtareas que las complementan. Estas tareas se pueden dividir en dos grupos, las tres primeras cuya función radica en comprender el marco del proyecto, el funcionamiento de la empresa y las herramientas a emplear antes de empezar el proyecto. Tanto las tareas como su duración han sido decididos entre el alumno y el supervisor.

La duración de las tareas se representan por horas. En este caso, el horario del trabajo consiste en 25 horas semanales, repartidas en 5 horas por cada día laboral de la semana (lunes a viernes). La fecha de inicio de las prácticas data del 1 de marzo de 2021 y su finalización el 26 de mayo de 2021. La duración total de las prácticas son de 300 horas, que es la suma total de las tareas.

Tabla 3.1. Desglose de tareas sobre la planificación inicial.

Nº	**Tarea**	**Tiempo (h.)**	**Dependencias**
1	**Planificación**	**60 horas**	
1.1	Análisis del entorno	20 horas	
1.1.1	Funcionamiento en el marco de la empresa	12 horas	
1.1.2	Vista general del software	8 horas	
1.2	Estudio de las herramientas a usar	40 horas	1.1
1.2.1	IDE Visual Studio Community 2019	15 horas	
1.2.2	Librería iText	15 horas	
1.2.3	Estudio de lenguaje C#	5 horas	
1.2.4	Signature Pad	5 horas	
2	**Preparación**	**55 horas**	**1**
2.1	Adición de imagen en PDF mediante iText	35 horas	
2.1.1	Búsqueda de posibles mejoras	25 horas	
2.1.2	Actualización de la documentación existente	10 horas	2.1.1
2.2	Generación de firma con JavaScript mediante Signature Pad	20 horas	2.1
3	**Análisis**	**65 horas**	**2**
3.1	Estudio del funcionamiento de la aplicación Wefinz	65 horas	
3.1.1	Análisis del código actual del sistema, tanto cliente como servidor	35 horas	
3.1.2	Aprendizaje sobre la creación y adición de módulos en el servidor	30 horas	3.1.1
4	**Implementación**	**85 horas**	**3**

4.1	Desarrollo del código del nuevo módulo en el servidor	30 horas	
4.2	Creación de los CIEs encargados del almacenamiento de datos	20 horas	
4.3	Desarrollo de la interfaz gráfica web del cliente para el trabajo con el nuevo módulo	20 horas	4.1
4.4	Implementación de transacciones para el trabajo del módulo en la capa de negocio	15 horas	4.3
5	**Prueba y puesta en marcha**	**35 horas**	**4**
5.1	Prueba de eficiencia en el software final	15 horas	
5.2	Documentación de los nuevos procedimientos	20 horas	5.1

3.3 Estimación de recursos

La estimación de recursos consiste en la planificación y garantía de la disponibilidad de los mismos para asegurar el buen desarrollo y éxito de un proyecto. La gestión de los recursos disponibles es esencial para cualquier proyecto y debe tenerse en cuenta incluso antes de que el mismo comience.

Se puede dividir la estimación de recursos en: recursos humanos, recursos materiales, los cuales pueden ser de software o hardware, y finalmente los recursos indirectos.

En cuanto a los recursos humanos:

- **Miguel Martínez Amat**: desarrollador y analista del software.
- **Juan Bautista García Traver**: analista del software y supervisor.

En cuanto a los recursos materiales del software:

- **Microsoft Visual Studio Code**: editor de código fuente para el desarrollo del frontend del proyecto.
- **IDE Visual Studio Community 2019:** entorno de desarrollo para la implementación del proyecto
- **Signature Pad:** una biblioteca de JavaScript para dibujar firmas.
- **PDF.js:** visor de documento PDF creado con HTML5.

- **iTextSharp 5:** biblioteca Open Source para la manipulación y creación de PDFs.
- **MariaDB:** es un sistema de gestión de bases de datos derivado de MySQL con licencia GPL.
- **HeidiSQL:** es un software libre y de código abierto que permite conectarse a servidores MySQL.

En cuanto a los recursos materiales del hardware:

- **MSI GL75 Leopard 10SEK-040XES:** ordenador portátil con i7-10750H a 120HZ con el sistema operativo de Windows 10 pro.
- **Ratón inalámbrico TECKNET**

En cuanto a los costes indirectos:

- **Alquiler.**
- **Limpieza.**
- **Energía eléctrica.**
- **Comunicaciones.**
- **Mobiliario.**

3.4 Estimación de costes

Los costos se estiman para todos los recursos asignados al proyecto, es decir, recursos humanos, recursos materiales, software y hardware y posibles costes indirectos.

Coste de recursos humanos.

Para calcular los costes humanos del proyecto, hay que tener en cuenta únicamente el sueldo que tendría un programador junior. Según la plataforma web de indeed [8], el coste que haría referencia al recurso humano para un solo desarrollador es de 19.310 € al año. Con este dato podemos saber el coste que se produciría por las 300 horas del proyecto:

*Horas al mes = 40 horas semanales * 4 semanas = 160 horas / mes*

Precio mes = (19.310 €/año) / 12 meses = 1.609,16 €/ mes

*Precio mes más Seguridad Social = (1.609.16 €/mes) * 30% = 2.091,90 €/mes*

Precio hora = (2.091,90 €/mes) / (160 horas / mes) = 13,07 € / hora

Como el proyecto es de 300, el coste total por el desarrollo del proyecto en la estancia en prácticas sería de:

*Coste recursos humanos del proyecto= 300 horas * 10.05 €/hora = 3.015 €*

Coste de recursos de software.

- **IDE Visual Studio Community 2019:** Según la licencia del propio software se detalla; cualquier número de usuarios puede utilizar el software para desarrollar y probar las aplicaciones de su propiedad como parte de aprendizaje y educación del tipo en línea o presencial, o para fines de investigación académica [9]. Esto quiere decir que, en este caso, es gratuito.
- **Microsoft Visual Studio Code y Signature Pad**: Ambas están sujetas a la licencia MIT [10] por lo que su uso es gratuito.
- **iTextSharp 5:** Según se detalla en la web del software [11] la licencia de uso de la biblioteca es AGPL. Por lo tanto es un software libre y gratuito.
- **PDF.js:** Es un proyecto de código abierto publicado en GitHub [2] y por lo tanto es gratuito.
- **MariaDB y HeidiSQL:** Ambas están sujetas a la licencia GPL [12], por lo que su uso es gratuito.

Con todo esto, los recursos de software utilizados para desarrollar el proyecto no supondría ningún gasto ya que todas las herramientas tienen una licencia gratuita.

Coste recursos de hardware.

En cuanto a los costes de los recursos de hardware hay que tener en cuenta el ordenador portátil, el sistema operativo que no venia instalado y el ratón con el que se ha trabajado:

Tabla 3.2 Costes recursos del hardware

Recurso	**Coste**
MSI GL75 Leopard sin sistema operativo preinstalado	1.405,55 €

Licencia Digital Windows 10 Pro	11,90 €
Ratón inalámbrico TECKNET	15,99 €
Se tiene en cuenta que la amortización de los componentes de hardware se amortizan en 4 años y que las prácticas son alrededor de 3 meses	(1.405,55 € + 11,95 € + 15,99 €) / (4 años * 12 meses) = 29,86 € / mes
Total amortizado durante el proyecto	29,86 € * 3 meses = 89,58 €

Costes indirectos.

Alquiler: 423 € / mes

Limpieza: 60 € / mes

Energía eléctrica: 130 € / mes

Comunicaciones: 234 € / mes

Mobiliario: 187 € / mes

Estos costes indirectos son aquellos que son iguales para todas las personas de la empresa por lo que habría que dividirlo entre los 4 integrantes de la misma. De esta forma los costes indirectos quedaría de la siguiente forma:

1.034 € / 4 = 258.5 € / mes relacionados con el proyecto.

Teniendo en cuenta la estancia de alrededor de 3 meses, desde el día 1 de marzo hasta el dia 26 de mayo, el precio total de costes indirectos sería:

258,5 € / mes * 3 meses = 775,5 € / mes

Sumario de costes totales.

Una vez obtenidos los costes para cada sección, los costes totales del proyecto de 300 horas vienen detallados en la siguiente tabla (Figura 3.4):

Tabla 3.3. Costes totales

Recurso	**Coste**
Costes recursos humanos	3.015 €
Costes recursos materiales	89,58 €

Costes indirectos	775,5 €
Total costes	3.880,08 €

3.5 Seguimiento del proyecto

Durante el transcurso de las prácticas, se ha utilizado el programa del bloc de notas para llevar un diario con las tareas completadas diariamente y la información adicional necesaria para las mismas.

El diario se divide por días y las tareas se ordenan con un indicador que las hace visibles rápidamente cuando se consulta.

Este seguimiento diario ha permitido mantener el hilo del trabajo entre días, en especial en los fines de semana y festivos. Además, ayudaba a seguir la línea de desarrollo que había seguido el proyecto.

Por otra parte, también se han realizado la propuesta técnica del TFG y los informes quincenales, estructurados de la siguiente manera:

- Pequeño resumen del trabajo realizado la quincena anterior.
- Resumen de la quincena actual.
- Pequeño anticipo sobre el trabajo de la siguiente quincena.

Durante el desarrollo del trabajo, se ha ido cambiando la planificación de manera que se iban superando las tareas con una mayor rapidez de la esperada. La planificación inicial estaba realizada de acuerdo a unos parámetros establecidos entre el supervisor y el alumno en prácticas, donde se establece la duración aproximada de las tareas. Es por ello que algunas tareas han visto reducida su duración y otras han aumentado.

Se ha realizado una nueva tabla con el desglose de tareas (Tabla 3.4) con el coste, lo más aproximado posible, real dedicado a las tareas. Estas horas están basadas en el diario, anteriormente mencionado, y el propio trabajo realizado.

La principal parte del trabajo que se ha visto con el tiempo reducido son la planificación y la preparación, esto se debe a que el alumno ya tenía previo conocimiento sobre algunas de las herramientas que se han usado para el proyecto. Esto favoreció la curva de aprendizaje sobre las herramientas necesarias para el proyecto y acortar enormemente el tiempo dedicado a las tareas.

Por otra parte, el incremento de las horas en la tarea de implementación se debe a dos factores: problemas surgidos y tiempo extra. En la parte de la implementación es cuando más problemas se encontraron para realizar el proyecto y algunos de ellos ralentizaron la programación de las tareas incluso con el tiempo extra obtenido por las anteriores. Este tiempo extra compensa los problemas originados durante la implementación pero también permitieron ampliar las mejoras que se pudieron implementar en el proyecto.

TAREAS	DURACIÓN	INICIO	FINALIZACIÓN	PREDECESORAS
1.Planificación	**40 horas**	**01/03/21**	**10/03/21**	
1.1.Análisis del entorno	15 horas	01/03/21	03/03/21	
1.1.1.Funcionamiento en el marco de la empresa	10 horas	01/03/21	02/03/21	
1.1.2.Vista general del software	5 horas	03/03/21	03/03/21	
1.2.Estudio herramientas a usar	25 horas	04/03/21	10/03/21	1.1
1.2.1.IDE Visual Studio Community 2019	10 horas	04/03/21	05/03/21	
1.2.2.Librería iText	5 horas	08/03/21	08/03/21	
1.2.3.Estudio del lenguaje C#	5 horas	09/03/21	09/03/21	
1.2.4.SignaturePad	5 horas	10/03/21	10/03/21	
2.Preparación	**30 horas**	**11/03/21**	**18/03/21**	**1**
2.1.Adición de imagen en PDF mediante iText	20 horas	11/03/21	16/03/21	
2.1.1.Busqueda de posibles mejoras	15 horas	11/03/21	15/03/21	
2.1.2.Actualización de la documentación existente	5 horas	16/03/21	16/03/21	2.1.1
2.2.Generación de firma con javascript mediante SignaturePad	10 horas	17/03/21	18/03/21	2.1
3.Análisis	**65 horas**	**20/03/21**	**09/04/21**	**2**
3.1.Estudio del funcionamiento de la aplicación Wefinz	65 horas	20/03/21	07/04/21	
3.1.1.Análisis del código actual del sistema, tanto cliente como servidor	35 horas	20/03/21	30/03/21	
3.1.2.Aprendizaje sobre la creación y adición de módulos en el servidor	30 horas	31/03/21	09/04/21	3.1.1
4.Implementación	**130 horas**	**12/04/21**	**17/05/21**	**3**
4.1.Desarrollo del código del nuevo módulo en el servidor	45 horas	12/04/21	22/04/21	
4.2.Creación de los CIEs encargados del almacenamiento de los datos	20 horas	23/04/21	28/04/21	
4.3.Desarrollo de la interfaz gráfica web del cliente para el trabajo con el nuevo módulo	40 horas	29/04/21	10/05/21	4.1
4.4.Implementación de transacciones para el trabajo del módulo en la capa de negocio	25 horas	11/05/21	17/05/21	4.3
5.Prueba y puesta en marcha	**35 horas**	**18/05/21**	**26/05/21**	**4**
5.1.Prueba de eficiencia en el software final	15 horas	18/05/21	20/05/21	
5.2.Documentacion de los nuevos procedimientos	20 horas	21/05/21	26/05/21	5.1

Figura 3.4. Diagrama de Gantt sobre la planificación real.

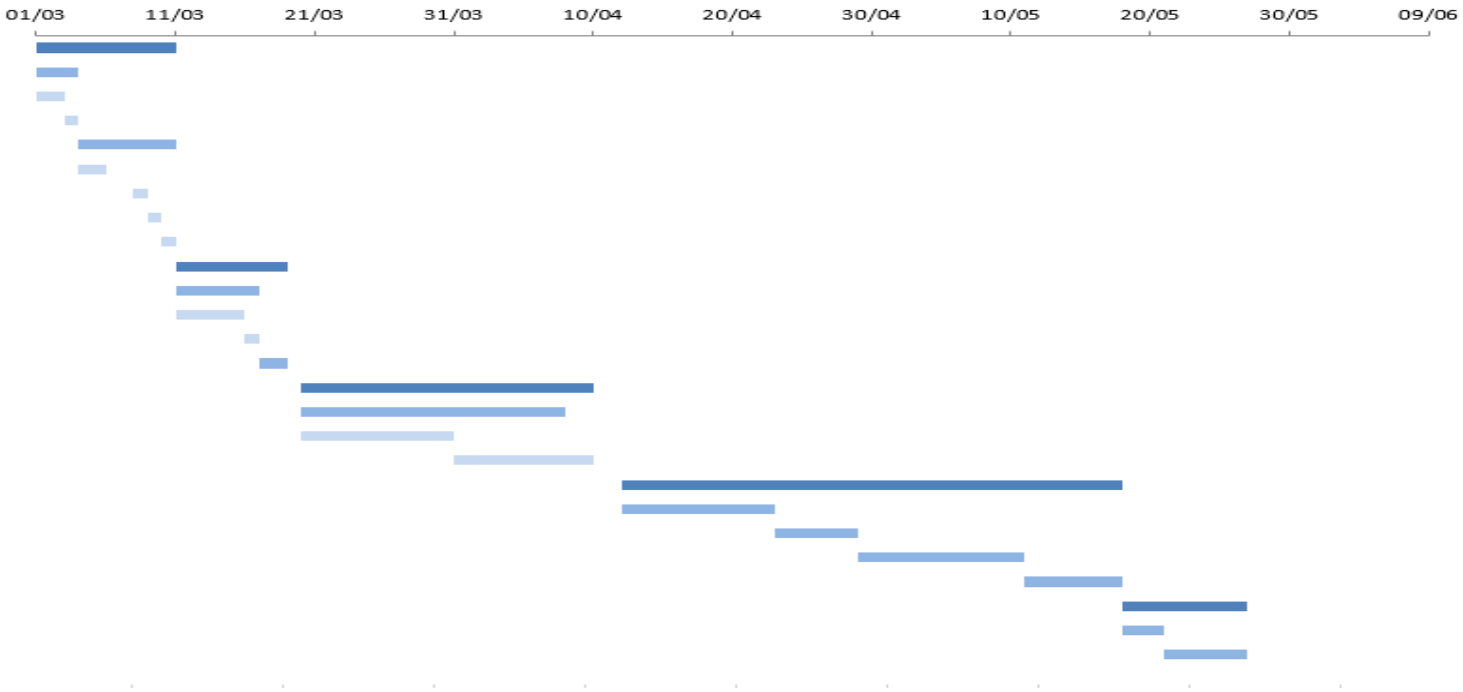

Figura 3.5 Gráfica diagrama de Gantt

Capítulo 4

Análisis y diseño del sistema

4.1 Análisis del sistema

El software desarrollado durante este proyecto mejora la funcionalidad del software de firma de documentos ya presente, añadiendo nuevas funcionalidades como el gestor documental.

El funcionamiento del nuevo sistema resultante de las mejoras se puede dividir en dos partes: por un lado el gestor documental, encargado de la función de subida y visualización de los documentos PDF, y la firma del documento, que se encarga de la parte del sistema que permite la firma.

4.1.1 Gestor documental

En primer lugar se describe la funcionalidad prevista del gestor documental. Este gestor documental permitirá, mediante el uso de un botón, que el usuario pueda subir sus propios documentos para la realización, si así lo desea, de la firma. Este botón únicamente acepta documentos PDF, por lo que no se puede subir otro tipo de documento diferente.

El nuevo documento se almacenará en la base de datos, para ello se le asignarán un serie de valores, atributos, para identificar tanto el documento como el propietario y datos importantes como:

- **DocumentName**: Nombre del documento.
- **State**: Estado del documento que por defecto es “pending”(pendiente).
- **OwnerUserID**: Identificador del usuario que sube el documento.
- **UploadDate**: Fecha en la que se sube el documento.
- **DocumentType**: Tipo de documento.

En cuanto a la visualización, el gestor documental hará uso de la biblioteca de PDF.js. PDF.js se aprovecha de JavaScript asincrónico[14] y XML[15] para descargar el archivo PDF de un servidor web y analizar su contenido. Una vez preparado, el contenido se procesa en un elemento “canvas” HTML5 utilizando comandos de dibujo específicos para estos elementos “canvas”.

El gestor consultará en la base de datos, mediante el servidor, los documentos que están asociados con el id del usuario conectado en ese momento. Estos documentos se almacenarán en un objeto result y contienen atributos de los cuales destacan:

- **Bytes**: El documento PDF en bytes.
- **Created**: Fecha de creación.
- **Extension**: Extensión del documento (en este proyecto siempre será pdf).
- **Hash**: Caracteres del documento PDF después de aplicar la función Hash.
- **UserID**: Usuario propietario del documento PDF.

Mediante el uso del atributo estado, el gestor comprueba si el documento está, o no, firmado y así los muestra de manera separada.

Tanto si el documento está firmado como si no lo está, todos tienen cuatro funcionalidades que se encuentran en los tres botones de la parte superior derecha. Estas funcionalidades se encargarán del borrado, el cambio del nombre y la firma. Para las dos primeras opciones el gestor interactúa con la base de datos y mediante el uso del id del documento lo elimina o modifica, mientras que en la opción firma el documento se envía a otra página del software donde se gestiona la firma sobre el documento.

4.1.2 Firma del documento

En esta parte del software se gestiona el documento PDF de forma individual. Hay una página anterior a la de la firma que se encarga de mostrar el documento PDF para que el usuario pueda elegir, mediante un botón, en qué página quiere que se realice la firma. El usuario únicamente debería situarse sobre la página que quiere firmar para que en el botón aparezca dicha página.

En esta página también se ha implementado un zoom para cuando el usuario está utilizando dispositivos táctiles, tales como móviles. Esta parte del sistema permite mediante el uso del “pinch” (hacer zoom con dos dedos) ampliar el documento. Aparte de esta funcionalidad, es posible del mismo modo efectuar un doble toque en la pantalla para que se haga un zoom directo y de corta distancia.

Una vez seleccionada la página donde se desea realizar la firma, el sistema realizará las siguientes fases:

1- Enviará el documento junto con el número de la página a la parte del software encargado de la misma. En este caso únicamente se mostrará la página donde se quiere firmar y permite, mediante un doble toque, seleccionar la zona donde se hará la firma.

2- Al seleccionar la zona para realizar la firma, aparece un canvas que, junto a la biblioteca Signature Pad, permitirá mediante el uso del ratón, o del dedo en caso de dispositivos portátiles, efectuar la firma. También será posible elegir el color de la firma entre varias opciones.

3- Previsualizar antes de enviar el documento a la base de datos. En esta previsualización se podrá, si se desea, cancelar la firma para hacerla de nuevo.

4- Cuando se envíe la firma a la base de datos, primero el servidor usa la biblioteca iText para la adición de la imagen de la firma en el documento PDF. Luego, mediante el servidor, se registrarán una serie de atributos para poder llevar a cabo un control de los documentos firmados, así como de dar veracidad a la firma. Esto se debe a que para que la firma sea válida debe tener registrado el Hash[16] del documento. Estos atributos son:

- **Ítems:** Almacena los puntos de los que se constituye la firma.
- **Signature Data Image:** Contiene la imagen en Base64[17].
- **PdfHash:** Caracteres del documento PDF después de aplicar la función Hash.
- **PageNumber:** Número de la página donde se encuentra la firma.
- **CoordX:** Coordenada de la posición de la firma en el eje X.
- **CoordY:** Coordenada de la posición de la firma en el eje Y.
- **PercentualWidth:** Porcentaje de la diferencia de anchura entre el tamaño de la firma y un documento en tamaño A4.
- **PercentualHeight:** Porcentaje de la diferencia de altura entre el tamaño de la firma y un documento en tamaño A4.

5- Una vez se envíe la firma a la base de datos, se le asignará el identificador del usuario y se registrará la fecha en la que se ha realizado, así como el identificador de la sesión.

4.2 Diseño de la arquitectura del sistema

El sistema que se usará para desarrollar el sistema es una arquitectura basada en Vista-Modelo-Controlador. Esto es debido a que la aplicación separa los datos de la aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Estos componentes son los siguientes (Figura 4.1)[18]:

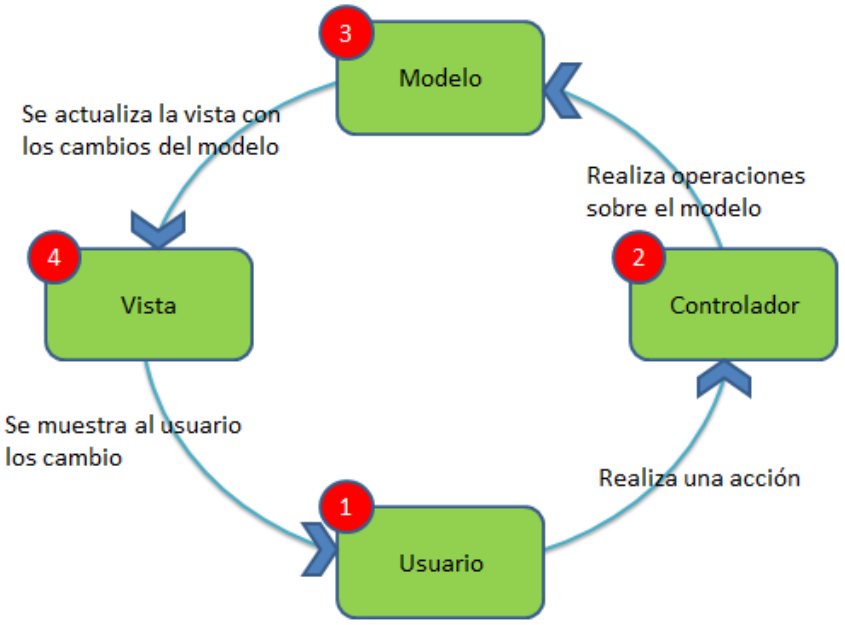

Figura 4.1. Diagrama MVC.

El Modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia, que corresponde a la base de datos en este proyecto. La Vista, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos de interacción con éste, los archivos HTML en este caso.

El Controlador, que actúa como intermediario entre el Modelo y la Vista, gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno, en este caso los archivos JavaScript.

4.3 Diseño de la interfaz gráfica

Para la visualización de la aplicación por parte del usuario, se utilizará el lenguaje HTML apoyado por el lenguaje CSS. El HTML se encargará de la estructura de la interfaz gráfica mediante el uso de etiquetas, mientras que el CSS se encargará de los estilos de la página (color, posición de los elementos, fuentes, etc).

En este proyecto, se dispone de tres páginas las cuales están estructuradas de diferentes maneras. Todas ellas tienen en común un elemento que tiene la función de header. Este proyecto se ha realizado para que se pueda ejecutar tanto en ordenador como en dispositivos móviles, por lo que la interfaz difiere entre estos. Por ello, a continuación se mostrará la interfaz perteneciente a la versión del ordenador y posteriormente la versión para móviles.

En primer lugar se mostrarán y explicarán las interfaces gráficas utilizadas para la versión de ordenador, la funcionalidad es prácticamente la misma pero la visualización difiere entre ellas.

Gestor documental

Tiene un botón para poder subir los documentos y muestra los mismo, divididos entre subidos y firmados (Figura 4.2).

Figura 4.2. Interfaz gestor documental

El gestor muestra unos canvas con la imagen y los nombres de los documentos, si se hace click sobre el canvas se puede acceder a la vista del documento.

En la parte superior derecha aparecen tres puntos los cuales desglosan un modal con las opciones disponibles para el documento PDF (Figura 4.3).

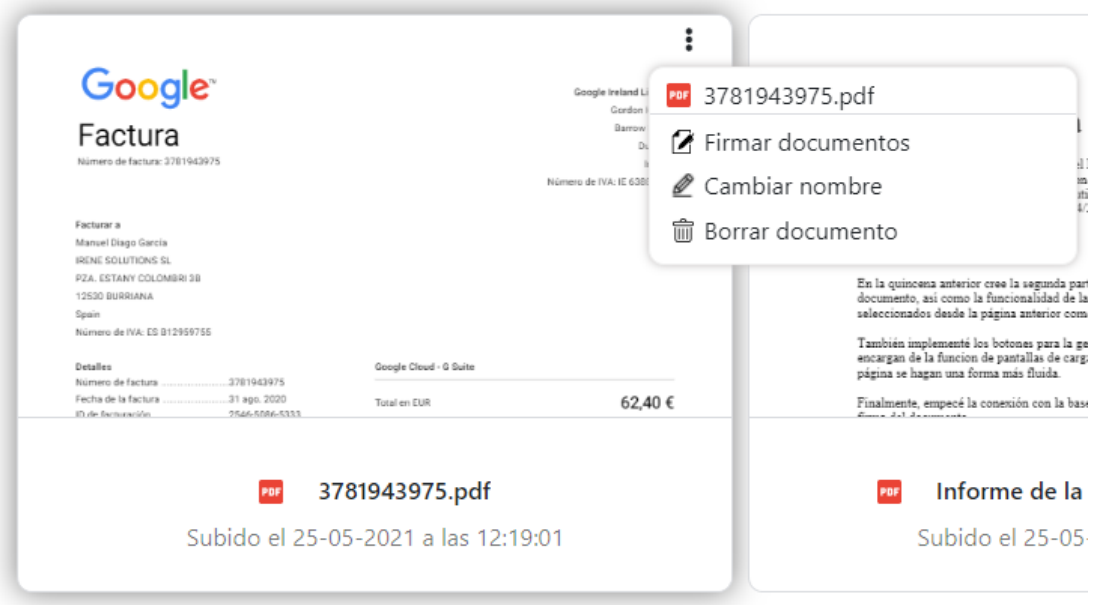

Figura 4.3. Interfaz de las opciones

Visualizador de documentos sin opción de firma

La segunda página se encarga de mostrar el documento PDF. Esta página tiene dos modos de visualización, si se ha accedido mediante el canvas únicamente aparece el

documento PDF pero no aparece el botón para firmar (Figura 4.4). Además, tampoco aparece el modal de ayuda si no se ha seleccionado la opción de firmar.

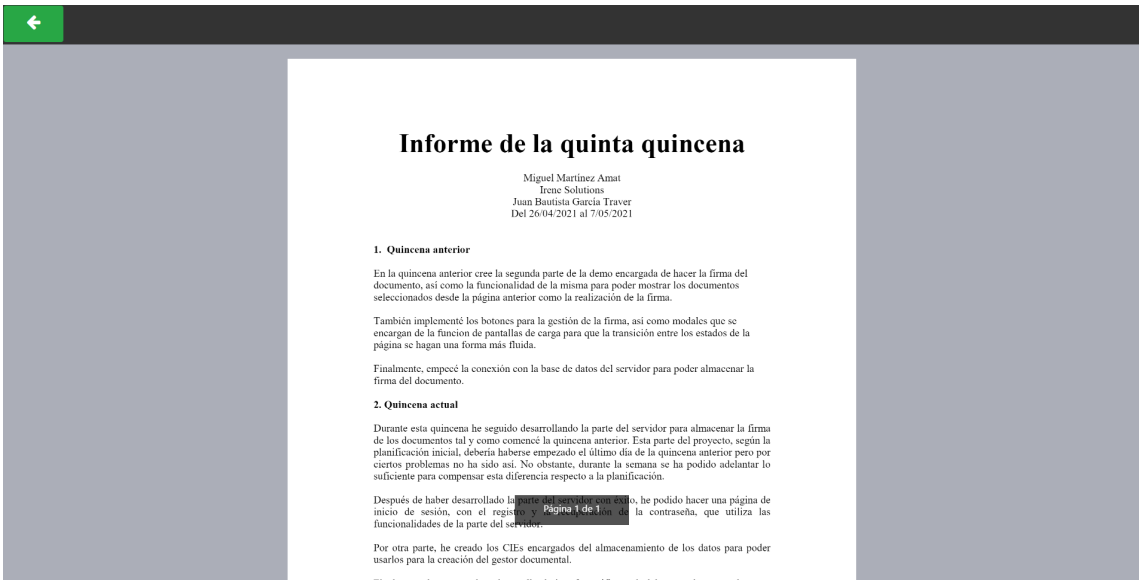

Figura 4.4. Visualización sin opción a firma

Visualizador de documentos con opción de firma

En caso de seleccionar la firma del documento, aparece un modal de ayuda con una explicación sobre el funcionamiento de la página (Figura 4.5).

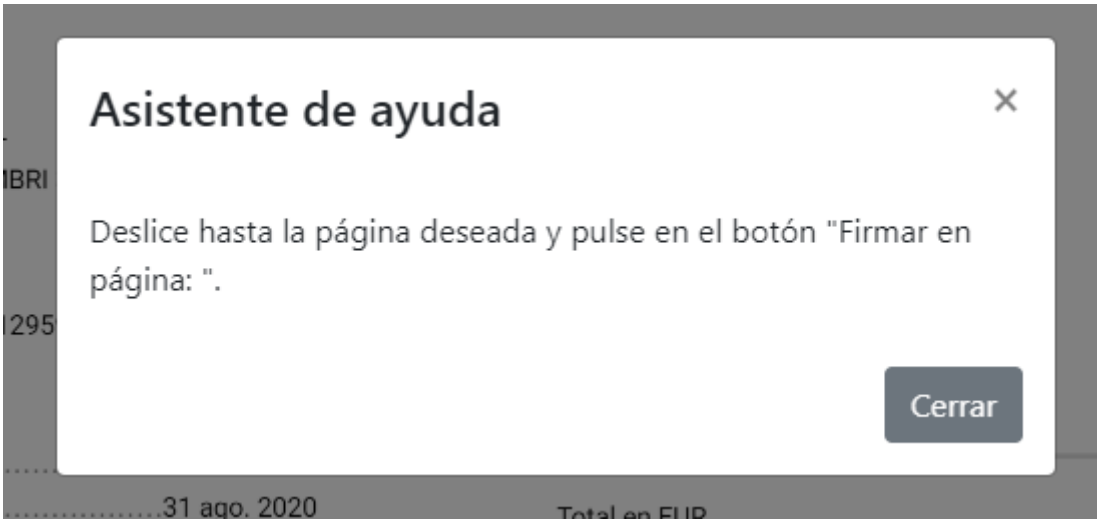

Figura 4.5. Asistente de ayuda de la visualización

Además, aparece un botón el cual, como se indica en el asistente de ayuda, permite seleccionar la página donde se quiera firmar (Figura 4.6).

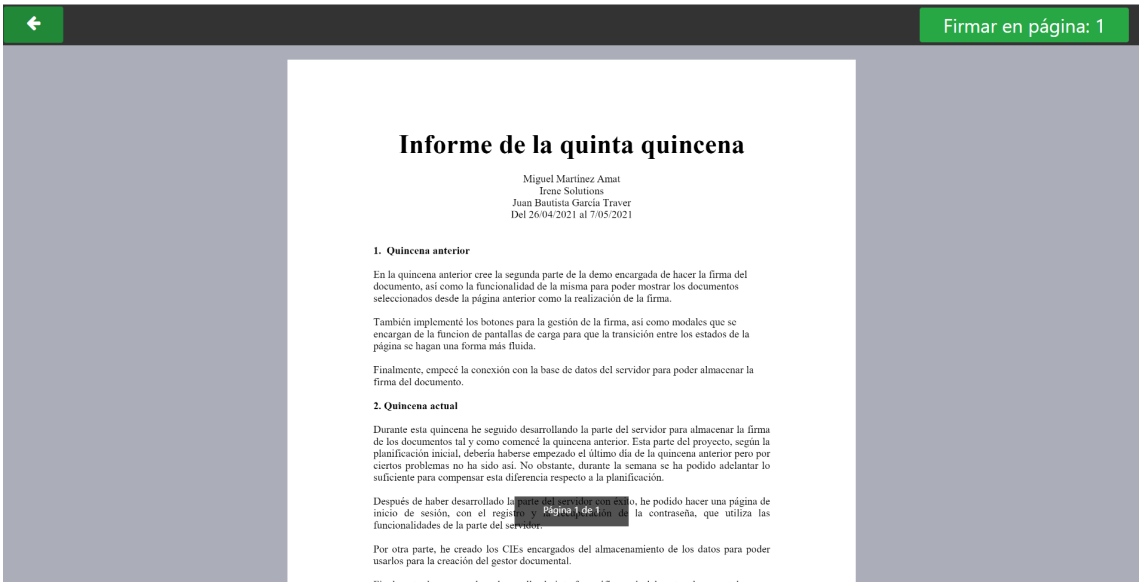

Figura 4.6. Visualización con la opción de firma

Firma de documentos

Finalmente, en la tercera página se visualiza la página seleccionada en la anterior. En este caso también aparece un modal de ayuda para facilitar el uso del software (Figura 4.7).

Figura 4.7. Modal de ayuda para la firma.

En esta página, se puede seleccionar la zona para firma con un doble click y aparece un modal para que se pueda efectuar la firma (Figura 4.8). En este modal se puede seleccionar el color de la firma.

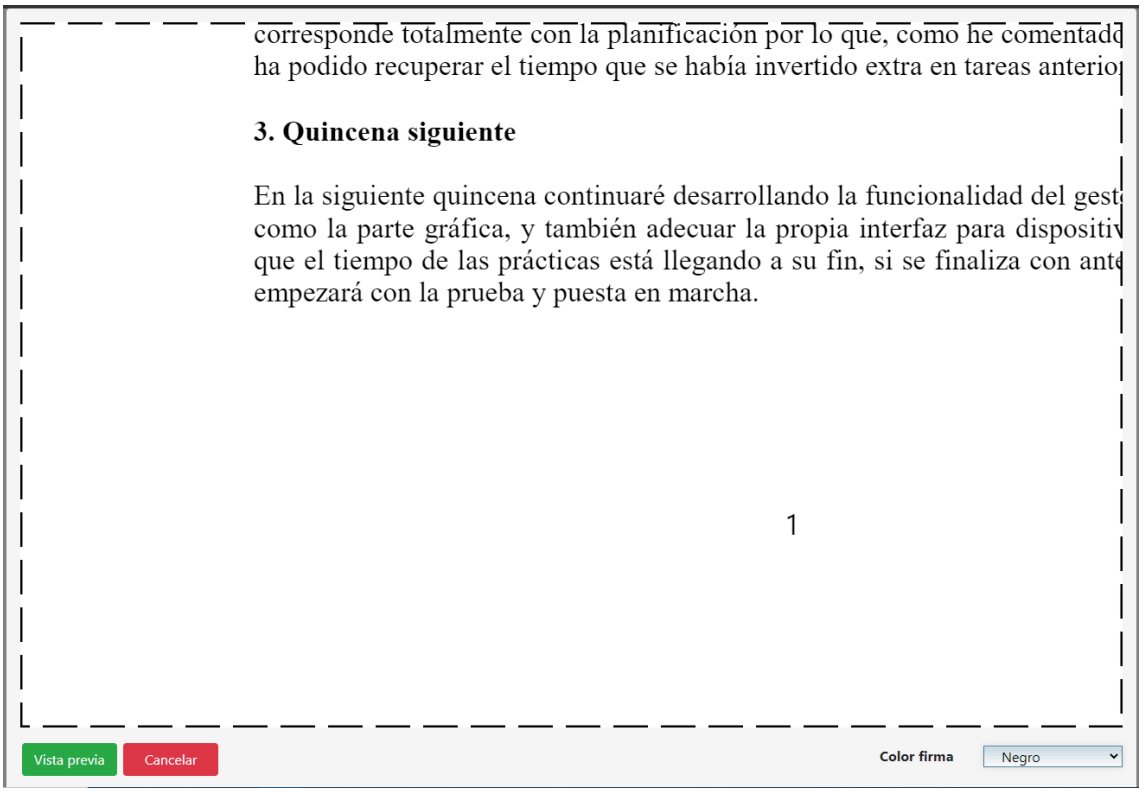

Figura 4.8. Modal para realizar la firma.

Una vez ejecutada la firma se hace una vista previa para ver el resultado final, de este modo el usuario puede enviar la firma definitivamente o cancelar para seleccionar una nueva zona para la firma (Figura 4.9).

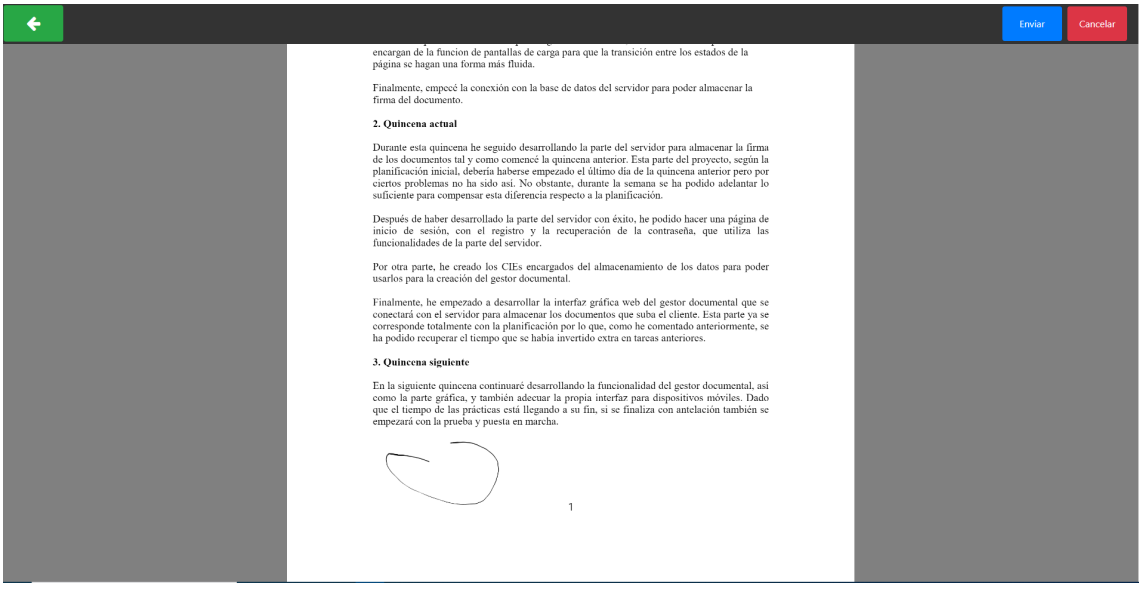

Figura 4.9. Previsualización de la firma.

Finalmente, cuando se envía la firma al servidor para que se inserte a la base de datos, aparece un mensaje que nos indica si se ha firmado correctamente y, en caso afirmativo, permite descargar el documento firmado (Figura 4.10).

Figura 4.10. Mensaje de firma correcta.

A continuación, se mostrará la interfaz perteneciente a la versión móvil que, si bien la funcionalidad es la misma, tiene un estilo diferente. Por ellos únicamente se mostrarán aquellas que difieran de la interfaz del ordenador.

Gestor documental

La interfaz del gestor documental se reduce para que la visualización de elementos sea más agradable (Figura 4.11).

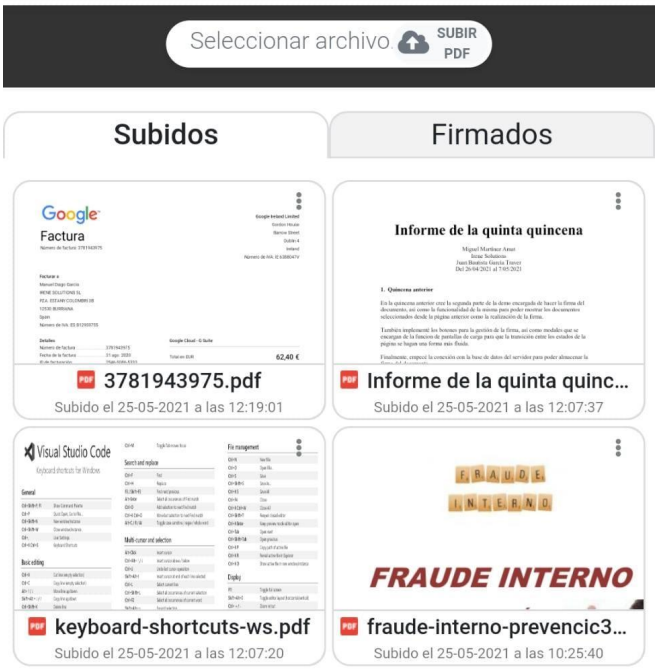

Figura 4.11. Interfaz gestor documental versión móvil.

En cuanto al modal que muestra las opciones para gestión del documento, se ha cambiado para que aparezca el modal desde la parte inferior y muestre las opciones. De esta forma, en un dispositivo de menor tamaño y con orientación vertical es más fácil interactuar con el modal (Figura 4.12).

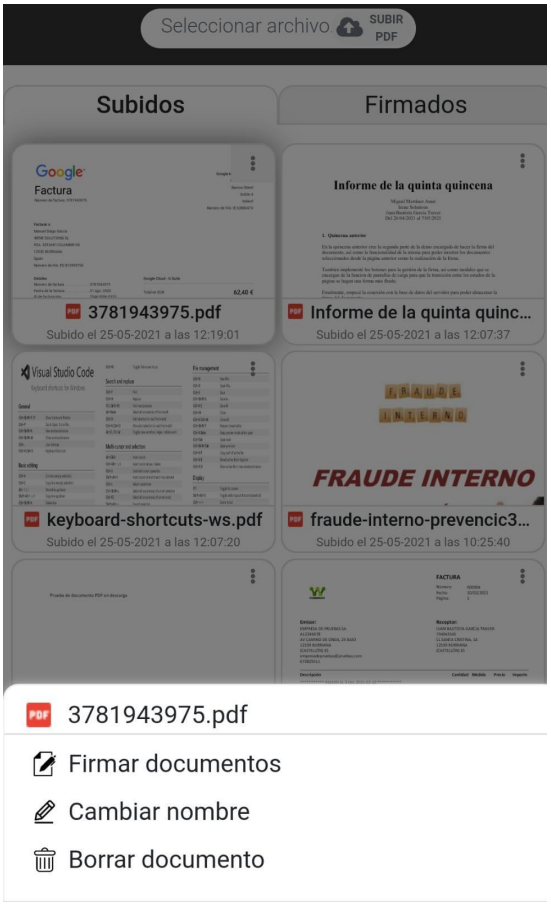

Figura 4.12. Interfaz de las opciones versión móvil

Visualizador de documentos

En cuanto a la interfaz gráfica de la visualización del documento no hay cambios notables, únicamente se han ajustado los valores para que se adapten al dispositivo (Figura 4.13 y Figura 4.14).

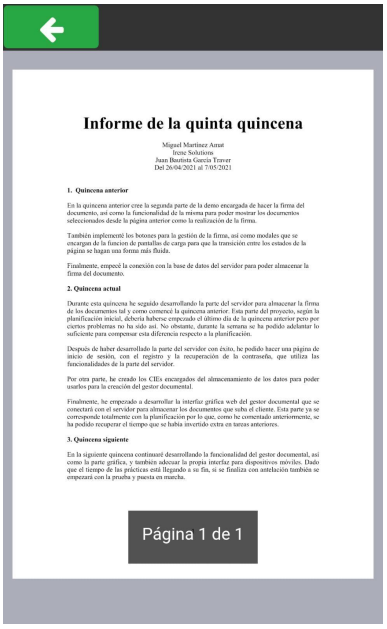

Figura 4.13. Visualización sin opción firma versión móvil

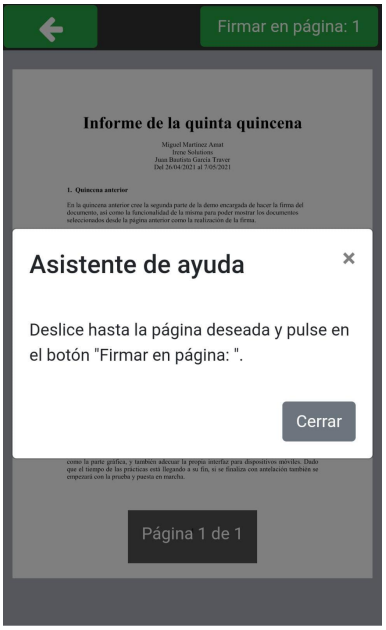

Figura 4.14. Visualización con opción de firma versión móvil.

Firma de documentos

Finalmente, la tercera página tampoco se diferencia en exceso de la versión de ordenador (Figura 4.15, Figura 4.17 y Figura 4.18). La única diferencia notable que se puede encontrar en esta versión es la redistribución de los botones a la hora de visualizar el modal para hacer la firma (Figura 4.16).

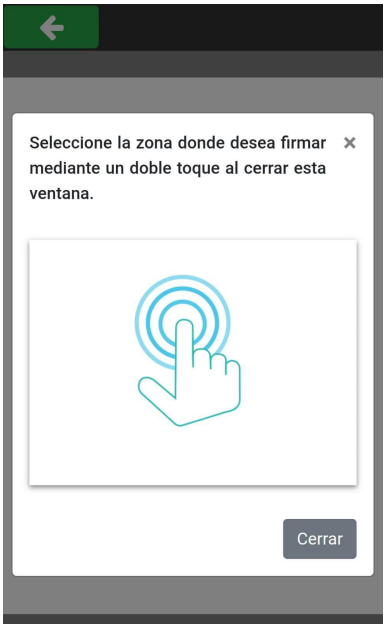

Figura 4.15. Modal de ayuda para la firma versión móvil

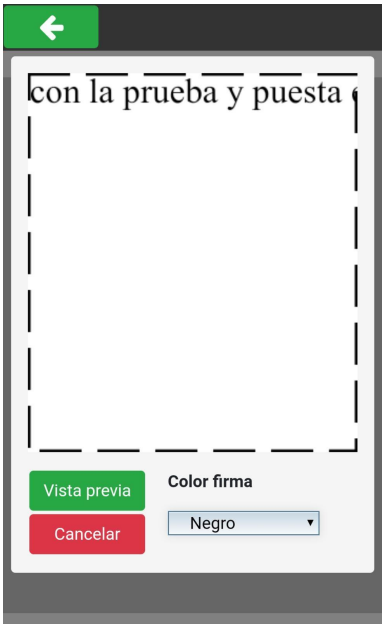

Figura 4.16. Modal para realizar la firma versión móvil.

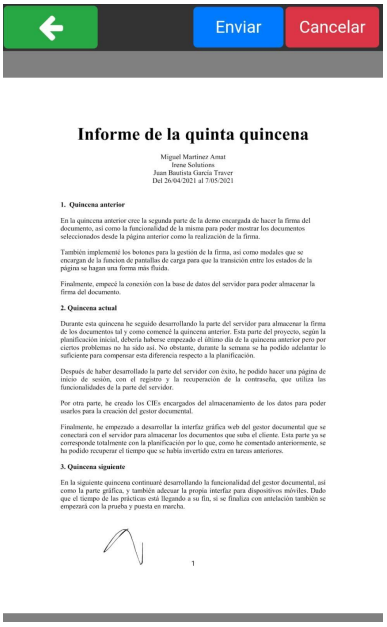

Figura 4.17. Previsualización de la firma versión móvil

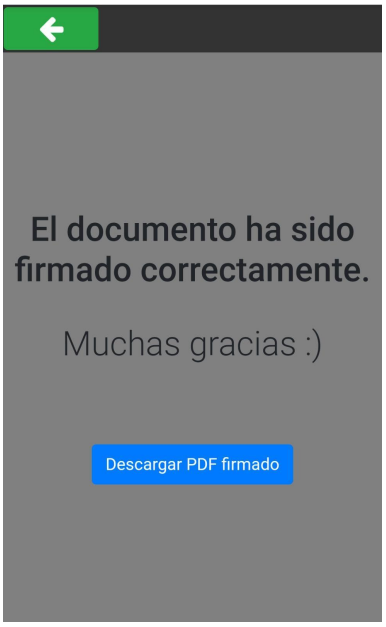

Figura 4.18. Mensaje de firma correcta versión móvil.

Capítulo 5

Implementación y pruebas

5.1 Detalles de implementación

Se detallan el proceso y la dirección tomada durante la estancia en prácticas para la implementación del proyecto. Esta implementación se compone de diversas partes por la cual ha tenido que pasar el alumno para llegar al producto final. Cabe destacar que el proyecto se ha realizado tanto para los ordenadores tradicionales como para dispositivos móviles. Por ello se han llevado a cabo algunas partes del código que solo se aplican a un dispositivo en concreto, mientras que otras partes se aplican a ambos.

5.1.1 Implementación de las llamadas al servidor para la gestión de la firma

Una vez que la segunda demo está completa. Se empieza a trabajar con la parte del servidor y la base de datos. Estas llamadas se hacen a un servidor llamado Kivu, desarrollado por la misma empresa.

Primero, se desarrolla la llamada que almacena en la base de datos la firma junto con el documento PDF. Esta llamada es un "Create" (Figura 5.14) que recibe por parámetros una variable que contiene los atributos necesarios para almacenar la firma con el documento PDF. Estos atributos están indicados y explicados en el apartado 4.1.2 de este documento.

```
Kivu.Isolutions.Firma.SignatureDataBlocks.Create(signatureDataBlock, function (result) {
```

Figura 5.14. Llamada al servidor de Kivu para crear el objeto del documento con la firma

Después, para poder comprobar el buen funcionamiento del software, se implementa el código que permite descargar el documento PDF una vez ha sido firmado. El método encargado de descargar el documento no hace una llamada directamente con el servidor, pero si utiliza una variable que contiene datos sacados del servidor (Figura 5.15).

```
PDFSigner.DownloadPDF = function () {

    let a = document.createElement("a");
    a.href = "data:application/pdf;base64," + encodeURI(PDFSigner.SignatureDataBlockResult);
    a.download = PDFSigner.PDFfile[0].Name;
    a.click();
}
```

Figura 5.16. Código que permite descargar el documento después de firmarlo

Esta variable se obtiene de un objeto Result que devuelve el servidor al llamar a la función InsertInPdf. Esta función inserta, a partir de los datos obtenidos por la función "Create", la firma en el documento. El objeto Result de este método es el que comprueba el método anterior para descargar el documento con la firma.

```
Kivu.Files.Pdf.Images.InsertInPdf({ JsonBody: JSON.stringify(pdfImageInsert) }, function (result) {

    if (result.ResultCode !== 0) {
        alert(result.ResultMessage);
    }
    else {
        PDFSigner.SignatureDataBlockResult = result.Return;
```

Figura 5.17. Código de inserción de firma en el documento PDF.

5.1.2 Creación del gestor documental con la versión definitiva del software

Para el control de los documentos, se decide desarrollar un gestor documental. Este gestor será el encargado de recibir los documentos PDF que se quieran subir y de mostrarlos. Por ello, en este punto, se elimina el botón de subir el documento PDF de la página de la visualización del documento y se implementa en el nuevo gestor documental.

Para poder mostrar y almacenar los documentos, el gestor documental debe conectarse con la base de datos a través del servidor, de la misma forma vista en el apartado anterior.

A la hora de subir nuevos documentos PDF, el gestor enviará el documento con una serie de atributos, para identificarlo, y este será enviado a la base de datos para su inserción (Figura 5.18). Si un documento comparte nombre con otro que ya ha sido subido, se le cambia el nombre para añadir al final de este un número, para mostrar que el nombre está duplicado.

```
    Metadata:
    {
        DocumentName: file.Name,
        Description: '',
        State: 'Pending',
        OwnerUserID: Kivu.Application.Credentials.User,
        UploadDate: new Date(),
        DocumentType: 'DOCUMENT'
    },
    DocTypeID: "DOCUMENT"
};

Kivu.Files.Indexer.Documents.Create(metadata, function (result) {
```

Figura 5.18. Atributos del nuevo documento y llamada al servidor

Por otra parte, para la visualización de los documentos, el gestor envía la petición al servidor con el identificador del usuario (en un principio el identificador es fijo hasta que se implementa el registro e inicio de sesión de usuarios). En esta llamada, el servidor devuelve un objeto Result (Figura 5.19) con los documentos que ese usuario tiene subidos, y el gestor los muestra.

```
(12) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
 0: {Protocol: 'http', Server: '192.168.1.72', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.0A01000000000000', …}
 1: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.0C01000000000000', …}
 2: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1101000000000000', …}
 3: {Protocol: 'http', Server: '192.168.1.73', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1301000000000000', …}
 4: {Protocol: 'http', Server: '192.168.1.73', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1401000000000000', …}
 5: {Protocol: 'http', Server: '192.168.1.73', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1501000000000000', …}
 6: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1701000000000000', …}
 7: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1801000000000000', …}
 8: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1901000000000000', …}
 9: {Protocol: 'http', Server: '192.168.1.72', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1C01000000000000', …}
 10: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.1F01000000000000', …}
 11: {Protocol: 'http', Server: 'localhost', Port: 8099, Root: 'Kivu', DocInstanceID: '0000000000000000.2201000000000000', …}
 length: 12
```

Figura 5.19. Contenido del objeto Result con los datos de los documentos en la base de datos

Una vez el gestor ha recibido los documentos del servidor, crea un canvas por cada documento y los dibuja en ellos. Cada canvas contiene: una previsualización del documento, el nombre del documento, la fecha en la que se ha subido el documento (o en la que se ha firmado) y una serie de opciones para la gestión del documento.

El gestor documental divide los documentos en dos, los documentos subidos y los documentos firmados. Estos documentos están ordenados por su fecha de subida o creación, dependiendo de si han sido subidos o firmados.

Los documentos subidos, son aquellos que no contienen ninguna firma, el documento original. Esto se debe a cómo funciona la aplicación, ya que cuando un documento es firmado, no se sobrescribe, sino que se crea un nuevo documento que contiene la firma. Por ello, cuando un documento que está en subidos es firmado, este continúa estando en el apartado de firmados.

Por otro lado, los documentos firmados, son copias del documento original con una firma insertada en ellos. Aunque un documento esté firmado, este puede volver a firmar sin problemas, la aplicación crearía una nueva firma que se relaciona de igual modo al documento, por lo que permite varias firmas en un mismo documento.Los documentos firmados mantienen el mismo nombre del original, excepto que se le añade al final del nombre un "_s" para identificar que ha sido firmado.

En cuanto a las opciones disponibles para los documentos, el gestor documental permite realizar cuatro acciones: visualizar el documento PDF, firmar el documento PDF, cambiar el nombre y borrar el documento.

Las opciones de visualizar y firmar el documento son prácticamente las mismas, a excepción de una funcionalidad. Cuando se pulsa sobre el canvas o a la opción firmar documento, el gestor envía a la página de visualización el nombre del documento y una variable que identifica si se envía a firmar o no (Figura 5.20). En la página de visualización se carga el documento desde la base de datos con el nombre y, dependiendo del valor de la variable, se muestra el botón que permite elegir la página del documento que se quiere firmar o no.

```
var parameters = {};

parameters.PDFfile = PDFfile.Name;
parameters.vista = fileType;

localStorage.removeItem('PDFfile');

localStorage.setItem('PDFfile', JSON.stringify(parameters));

document.location = "viewer.html";
```

Figura 5.20. Código para enviar el documento a la página de visualización.

De esta manera, ya no es necesario utilizar el localStorege para enviar los documentos. Esto mejora la eficiencia del software, ya que el localStorage tiene un tamaño límite y no podría ser posible enviar documentos muy extensos.

Por otro lado, a la hora de enviar la página seleccionada a la página de la firma, del mismo modo se enviaría el documento, con la diferencia de que enviara una variable con el número de página donde se desea firmar.

En lo que respecta al borrado, es una llamada al servidor en la que se le pasa como argumento el archivo y es eliminado de la base de datos. Una vez el documento es eliminado el gestor elimina de la vista el documento borrado y reorganiza la disposición de los restantes.

```
Kivu.Files.Indexer.Documents.Delete(file, function (result)
```

Figura 5.21. Código para eliminar el documento

Por otro lado, para cambiar el nombre, el gestor debe hacer una llamada al servidor con el nuevo nombre (Figura 5.22) y después, recibir el objeto Result con el documento con el nuevo nombre. Seguidamente, el gestor sobrescribe los datos del documento para actualizarlos con los nuevos.

```
Kivu.Files.Indexer.Documents.Change(file, function (result)
```

Figura 5.22. Código para cambiar el nombre el documento

5.1.3 Implementación de usuarios

Para mantener la privacidad y conocer a quién pertenecen los documentos almacenados en la base de datos, se implementa un registro de usuarios. El registro de usuarios está desarrollado a partir del existente por parte de la empresa.

Se quiere que en todo momento el usuario esté registrado, por ello se comprueban las credenciales y en caso de que no existan, se redirige la página al login para que el usuario se registre o inicie sesión (Figura 5.23).

```
// Inicio credenciales y actualizo user info en ui
Kivu.Application.Refresh(function () {
    Kivu.Application.Reset();
    document.location = "login/login.html";
});
```

Figura 5.23. Inicio de credenciales y comprobación de sesión

Todas las partes del software tienen esta funcionalidad, tanto el gestor, como la vista y la firma. Esto se debe a que hay un tiempo límite en el cual se puede estar inactivo en la aplicación, pasado este tiempo es necesario volver a iniciar sesión.

Las credenciales del usuario, se almacenan en el localStorage para que de este modo se pueda acceder a ellas desde cualquier parte de la aplicación.

Estas credenciales se usarán en el gestor para poder cargar los documentos propiedad del usuario y para que cuando se realice la firma, el nuevo documento firmado se almacene en la base de datos con el identificador del usuario (Figura 5.24).

```
OwnerUserID: Kivu.Application.Credentials.User,
```

Figura 5.24. Asignación del identificador del usuario al documento PDF.

Finalmente, una vez está completa la funcionalidad del gestor documental, se establecen los estilos para los dispositivos. En un principio se trabaja sobre el

ordenador, por lo que una vez se ha implementado todo hay que cambiar los estilos para que se adapten a los dispositivos más pequeños. Estos cambios afectan principalmente a las opciones de los documentos, como se muestra en el apartado 4.3 (Figura 4.12).

5.1.4 Desarrollo de mensajes de ayuda y pantallas de carga

Tras la implementación del gestor documental y comprobar que toda la funcionalidad está lista, se desarrollan una serie de modales que ayudan a la estética de la aplicación o a los usuarios.

Las pantallas de carga están implementadas en las tres páginas que se han desarrollado. Se compone de un fondo en blanco, que ocupa toda la aplicación, y un círculo en el centro que gira (Figura 5.25).

Figura 5.26. Pantalla de carga

Estas pantallas se muestran cuando se hace una llamada al servidor o a otra página y se ocultan cuando la ejecución ha finalizado.

```
panelTimer.style.visibility = 'hidden';
```

Figura 5.27. Código que oculta la pantalla de carga

En cuanto a los modales de ayuda, la aplicación dispone de dos: uno en la página de visualización y otra en la de firma. Ambos están descritos y se muestran en el apartado 4.3 de este trabajo.

5.2 Verificación y validación

Para validar el funcionamiento del software se ha utilizado una base de datos de prueba, proporcionada por la empresa, y documentos PDF disponibles por el alumno, en este caso principalmente documentos relacionados con las prácticas. Se han utilizado tanto documentos en horizontal y en vertical para comprobar que la posición de la firma en el documento sea la correcta, además de documentos de diferentes páginas.

En cuanto a la validación, la aplicación cumple con las especificaciones y logra su cometido. Esta validación se ha llevado a cabo con un lote de diferentes documentos de la empresa y el funcionamiento ha sido correcto en todos ellos, tanto la posición de la firma como el renderizado.

El software contiene un gestor documental que puede ser utilizado por usuarios para almacenar documentos sus documentos, visualizarlos y firmarlos. Se han desarrollado dos demos para comprobar el funcionamiento del software antes de llevarlo a su fase final.

Sin embargo, hay una funcionalidad que no ha podido ser implementada. En un principio se deseaba poder enviar vía correo electrónico o mensajería instantánea un documento a otra persona para que la firmara, pero por falta de tiempo y medios no se ha podido llevar a cabo. Habría sido necesario utilizar una url pública accesible para todo internet y un servidor para alojar el sistema.

5.2.1 Estudio de las herramientas a utilizar

Para la verificación y validación del proyecto, ha sido necesario estudiar el funcionamiento de las herramientas que se utilizaran para realizarlo.

Por ello, primero se han tenido que instalar los programas pertinentes para poder realizar el trabajo en el propio equipo. Además de la instalación de los programas, se ha creado una base de datos local, para la realización de pruebas, igual a la existente en la empresa pero vacía por motivos de privacidad y protección de datos.

Después de la instalación de los programas, se ha realizado un proceso de familiarización con Visual Studio, el entorno de desarrollo utilizado para el proyecto. Este proceso se ha llevado a cabo mientras se estudiaba el código ya existente, para poder comprobar y entender su funcionamiento.

Por otra parte, también se ha realizado un estudio sobre el funcionamiento de la biblioteca Signature Pad. Esto es, entender las clases que pertenecen a esta biblioteca para poder adaptarla al proyecto.

Además, la biblioteca PDF.js ha sido estudiada después de la biblioteca de Signature Pad, puesto que es la encargada de renderizar los documentos PDF en una imagen para que puedan ser firmados.

Finalmente, se ha realizado un estudio de la biblioteca iText, ya que es la que permite la manipulación de los documentos PDF. Esta biblioteca se emplea para la inserción de la imagen obtenida del canvas, generado por el Signature Pad y PDF.js, en el documento PDF.

5.2.2 Desarrollo de una pequeña demo con Signature Pad

Una vez realizado el estudio sobre las herramientas y las tecnologías que se van a utilizar, se ha procedido a implementar una demo de muestra con Signature Pad. Esta demo contiene las funcionalidades principales necesarias para realizar la firma, pero de momento se realiza sobre un canvas en blanco (Figura 5.1).

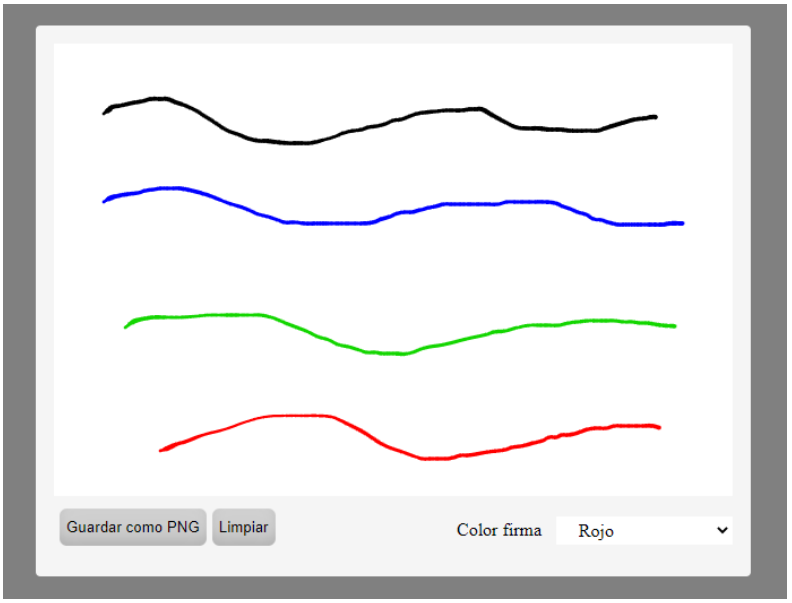

Figura 5.1. Demo con Signature Pad

Como se puede observar, esta demo ya contiene la mayoría de las funcionalidades que serán adaptadas finalmente al proyecto. En esta demo, es posible descargar la imagen, eliminar los puntos para dejarlo en blanco y elegir el color para los puntos de la firma.

Para que el movimiento con el ratón a la hora de realizar la firma sea preciso, es necesario redefinir el tamaño del canvas (Figura 5.2), ya que es posible que en algunas escalas la firma no sea precisa.

```
PanelFirma.ResizeCanvas = function() {
    var ratio = Math.max(window.devicePixelRatio || 1,1);

    canvas.width = canvas.offsetWidth * ratio;
    canvas.height = canvas.offsetHeight * ratio;
    canvas.getContext("2d").scale(ratio, ratio);

    signaturePad.clear();
```

Figura 5.2. Método para redefinir el tamaño del canvas.

5.2.3 Desarrollo de una segunda demo con PDF.js

Una parte importante del proyecto es poder visualizar los documentos PDF para poder firmarlos. En esta demo se implementa la biblioteca PDF.js para permitir que se pueda subir, mediante un botón, un documento PDF y este se visualice (Figura 5.3).

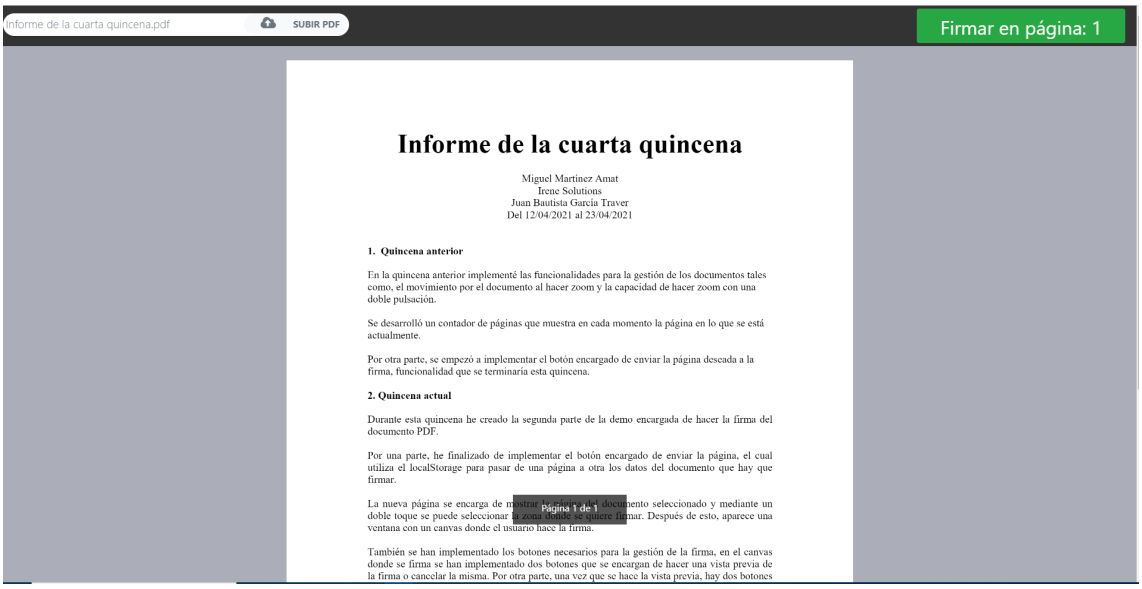

Figura 5.3. Demo con PDF.js

Esta demo no solo tiene la función de mostrar el documento, también permite elegir una página para enviarla a firmar.

Por un lado, el software obtiene el documento del botón superior izquierda y lo renderiza en diferentes canvas según el número de páginas (en la Figura 5.3 solo hay una página) y muestra un pie de foto con las páginas totales y la actual.

Para cada página, el programa tiene que determinar el tamaño del canvas con el tamaño del Viewport [19] y la escala en la que se encuentra (Figura 5.4).

```
pdfInstance.getPage(pageNumber).then(function (page) {
    console.log('Page loaded');
    console.log(numCanvas);
    console.log(i);
    console.log(page);

    var scale = 1;
    var viewport = page.getViewport({ scale: scale });

        // Prepare canvas using PDF page dimensions
    var canvas = document.getElementById("mostrar" + (page._pageIndex + 1));
    var context = canvas.getContext('2d');
    canvas.height = viewport.height;
    canvas.width = viewport.width;

    numCanvas++;
        // Render PDF page into canvas context
    var renderContext = {
        canvasContext: context,
        viewport: viewport
    };
    var renderTask = page.render(renderContext);
    renderTask.promise.then(function () {
        console.log('Page rendered');
    });
});
```

Figura 5.4. Código de renderizado para insertar la páginas del documento PDF en los canvas

Una vez implementada la visualización del documento PDF, se ha desarrollado la funcionalidad del pinch (zoom con dos dedos) para los dispositivos táctiles (Figura 5.5). Esto se debe a que el proyecto está desarrollado también para dispositivos móviles.

```
if (e.touches.length == 2 && pinch) {
    if (e.scale !== 1) { e.preventDefault(); }
    const pinchDistance = Math.hypot((e.touches[1].pageX - e.touches[0].pageX), (e.touches[1].pageY - e.touches[0].pageY));
    const originX = startX + container.scrollLeft;
    const originY = startY + container.scrollTop;

    deltaX = (((e.touches[0].pageX + e.touches[1].pageX) / 2) - originX) * 2; // x2 for accelarated movement
    deltaY = (((e.touches[0].pageY + e.touches[1].pageY) / 2) - originY) * 2; // x2 for accelarated movement
    pinchScale = pinchDistance / initialPinchDistance;
    pinchScale = Math.min(Math.max(minScale, pinchScale), maxScale);  //La escala será como mucho 4 y como mínimo 1

    var transformCords = getBoundCoordsPinch();
    transformCords.x && (deltaX = transformCords.x);
    transformCords.y && (deltaY = transformCords.y);

    if (pinchScale == 1) { deltaX = 0; deltaY = 0; }       //Solo se permitira el movimiento del elemento por la pantalla si
    viewer.style.transform = `translate3d(${deltaX}px, ${deltaY}px, 0) scale(${pinchScale})`;
    viewer.style.transformOrigin = `${originX}px ${originY}px`;

    position.x = deltaX;    //Para que el move empiece desde la cordenada del pinch
    position.y = deltaY;
```

Figura 5.5. Parte del código encargado del funcionamiento del zoom.

En esta parte del proyecto fue cuando se produjo el mayor problema, el control del movimiento del documento cuando se hacía el zoom no funcionaba en lo absoluto. Se deseaba que el movimiento se limitará por el tamaño del dispositivo de forma que aunque se moviera el documento este no desapareciera por los márgenes. Esto llevó a retrasos en la programación inicial, pero era algo necesario para el software ya que era para visualizar el documento en dispositivos móviles.

Finalmente, se encontró una solución mediante el uso de un método auxiliar que calculaba la posición con los márgenes establecidos (Figura 5.6).

```
//Este método se encarga de limitar el movimiento de la imagen por los margenes cuando se mueve por la pantalla
//Solo se podrá mover cuando la escala es mayor que 1
getBoundCoordsPinch = function () {
    var newValues = {
        x: 0,
        y:0
    }
        , widthScale = (pinchScale - 1) * container.offsetWidth / 2
        , limitLeft = deltaX * pinchScale - widthScale
        , limitRight = deltaX * pinchScale + widthScale
        , heightScale = (pinchScale - 1) * container.offsetHeight / (container.offsetHeight / window.screen.height * 10)
        , limitTop = deltaY * pinchScale - heightScale
        , limitBot = deltaY * pinchScale + heightScale;
    return limitLeft > 0 && (newValues.x = widthScale / pinchScale),
        limitRight < 0 && (newValues.x = -widthScale / pinchScale),
        limitTop > 0 && (newValues.y = heightScale / pinchScale),
        limitBot < 0 && (newValues.y = -heightScale / pinchScale),
        newValues;
}
```

Figura 5.6. Código encargado de controlar los límites del movimiento del documento mientras se realiza el zoom

Este método, calcula los límites disponibles para el documento dependiendo de la escala, la posición actual y el tamaño del contenedor del documento que sirve como limitador. Después, recalcula el valor de las coordenadas y estos se sobreescribe la posición original si esta sobrepasa los límites.

Por otro lado, a parte del pinch, también se implementa el zoom con un doble toque y el movimiento por el documento una vez ampliado.

El zoom mediante el doble toque, amplía la zona pulsada un porcentaje fijo y, cuando se vuelve a hacer la misma acción, el zoom vuelve al valor original (Figura 5.7).

```
// Este método se encarga de comprobar si se ha hecho un dobletap y, en ese caso, se encarga de la funcionalidad
detectDoubleTap = function (e) {
    var time = (new Date()).getTime();

    if (e.touches.length > 1) {
        lastTouchStart = null;
    }

    if (time - lastTouchStart < 300) {
        doubleTap = true;
        if (currentScale == 1) {

            originX = e.touches[0].pageX;
            originY = e.touches[0].pageY

            currentScale = 3;

            viewer.style.transform = ` scale(${currentScale})`;
            viewer.style.transformOrigin = `${originX}px ${originY}px`;

        } else {
            currentScale = 1;
            viewer.style.transform = `scale(${currentScale})`;
            position.x = 0;
            position.y = 0;
            viewer.style.transform = `translate3d(${position.x}px, ${position.y}px, 0) scale(${currentScale})`;
        }

    } else {
        doubleTap = false;
    }

    if (e.touches.length == 1) {
        lastTouchStart = time;
    }
```

Figura 5.7. Código encargado de la funcionalidad del zoom con doble toque.

El método encargado de realizar el movimiento por el documento es más complicado, ya que tiene que prevenir que el movimiento sobrepase el margen del dispositivo (como ocurre con el pinch).

Este método recibe como parámetro la posición de la pantalla donde se ha pulsado y la va comparando con la anterior registrada para saber hacia qué dirección se está moviendo el documento por la pantalla. Además, calcula en cada momento que no se sobrepase el margen del dispositivo para que la imagen no se pierda de la pantalla (Figura 5.8).

```
doMove = function (relativeX, relativeY) {
    if (lastX && lastY) {
        var deltaX = relativeX - lastX;
        var deltaY = relativeY - lastY;

        if (deltaX > 0) {
            if (viewer.getBoundingClientRect().x < 0)                                   //Controla el margen izquierdo
                position.x += deltaX;
        }
        if (deltaX < 0) {
            if (viewer.getBoundingClientRect().right > container.getBoundingClientRect().right)       // Controla el margen derecho
                position.x += deltaX;
        }
        if (deltaY > 0) {
            if (viewer.getBoundingClientRect().y < 0)                                   // Controla el margen superior
                position.y += deltaY;
        }
        //Cuanto más se desliza hacia abajo el bottom va disminuyendo, por lo que para que pare antes de que
        //el documento desaparezca de la pantalla se para cuando el valor es igual al de la pantalla

        if (deltaY < 0) {
            if (viewer.getBoundingClientRect().bottom > window.innerHeight)              // Controla el margen inferior
                position.y += deltaY;
        }

    }
    viewer.style.transform = `translate3d(${position.x}px, ${position.y}px, 0) scale(${currentScale})`;

    lastX = relativeX;
    lastY = relativeY;
```

Figura 5.8. Código de la funcionalidad del movimiento del documento por la pantalla del dispositivo.

Este método también tiene que tener en cuenta que el número del pie de página puede cambiar mientras se esté moviendo por el documento, por ello tiene un fragmento de código para controlar esta posibilidad (Figura 5.9).

```
// Esta parte del código se encarga de calcular y llamar al metodo de actualizar de la numeración de la página
var height = viewer.getBoundingClientRect().height / currentScale;
var pageHeight = height / totalPagesCount;
var currentPos = (viewer.getBoundingClientRect().y / currentScale) - (pageHeight / maxScale +1);

currentPage = ~~(-currentPos / pageHeight) + 1;

updateNumPage();
```

Figura 5.9. Código que actualiza el pie de página si con el movimiento este cambia.

Para enviar el documento desde una página a la otra se utiliza el LocalStorage transformado el documento a formato Json. Esta es una solución temporal, ya que posteriormente se implementarán llamadas al servidor que permitirán utilizar la base de datos para almacenar los documentos y únicamente se necesitarán enviar los identificadores y pocos datos más entre páginas.

Una vez enviado el documento, aparece la página seleccionada anteriormente (Figura 5.11) y solo después de seleccionar la zona para la firma con un doble toque, ya sea con el ratón en ordenador o con el dedo en la pantalla con dispositivos táctiles, aparece el modal de la firma con la parte del documento seleccionada para su firma.

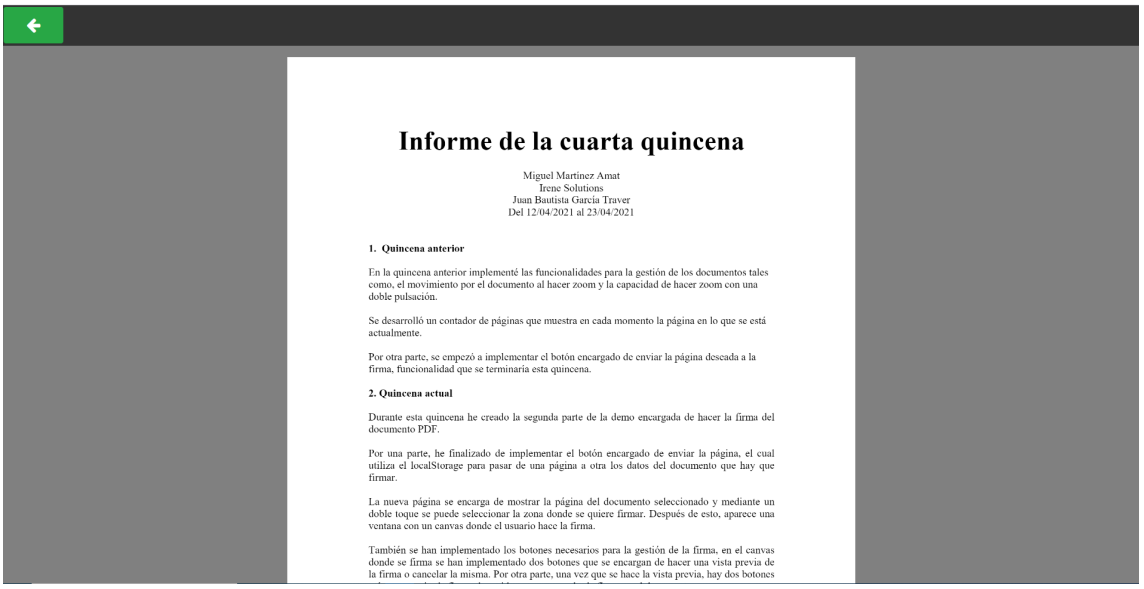

Figura 5.10. Página que permite seleccionar la zona de la firma

Este modal contendrá un canvas, en el cual se pondrá la página del documento, con las funcionalidades de la demo anterior (Figura 5.10).

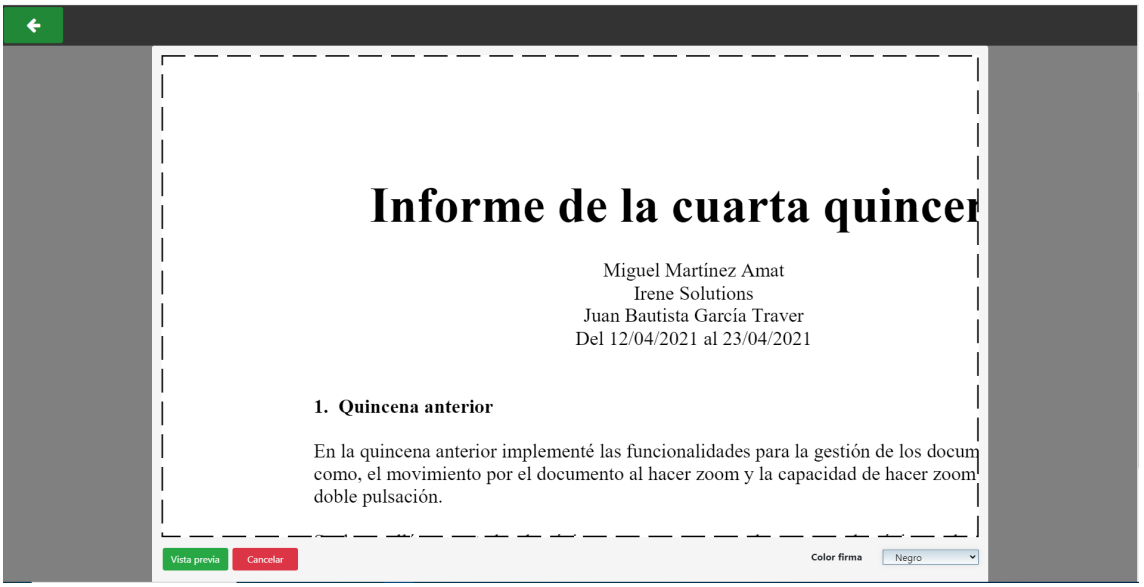

Figura 5.11. Canvas que permite la firma con la funcionalidad de la primera demo

En esta parte de la demo, se cambia la funcionalidad respecto a la anterior en cuanto a la funcionalidad de la vista. Ahora, es posible hacer una vista previa de la firma antes de enviarlo al servidor para almacenarlo a la base de datos, funcionalidad que aún no está implementada (Figura 5.12).

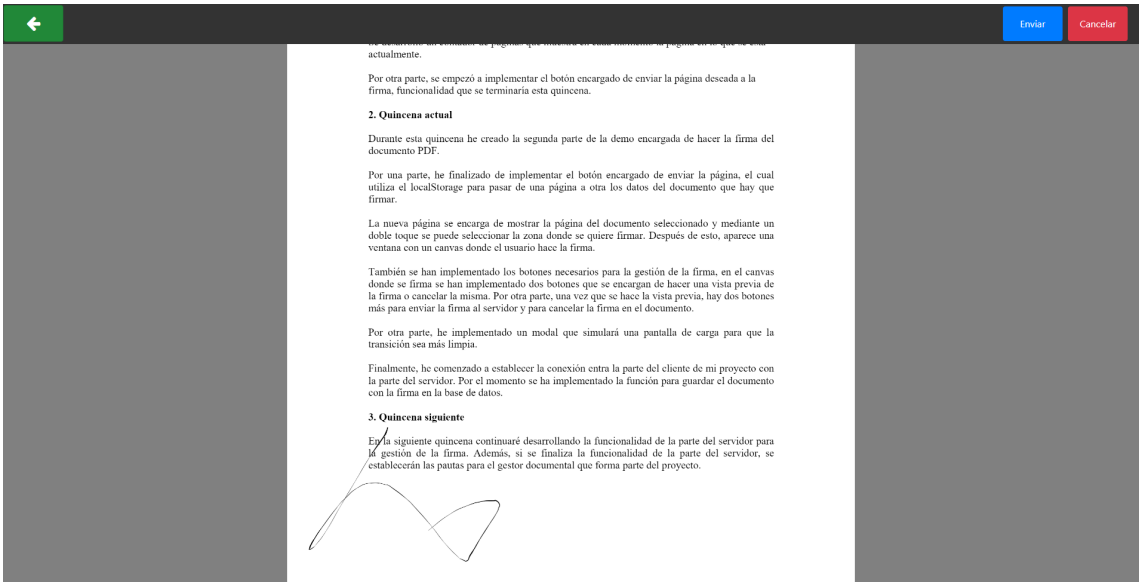

Figura 5.12. Vista previa de la firma

En esta parte del desarrollo, se encuentra otro de los problemas que dificultan la implementación del proyecto. Este problema viene relacionado con el canvas de la página de la firma, donde el tamaño de la página no se ajustaba al canvas. Es decir, el programa insertaba la página entera en el canvas y esto provocaba que la imagen no se ajustara con la zona de firma.

Para solucionar este problema, calculamos el tamaño del canvas dependiendo del tamaño de la pantalla. Para pantallas de más de 1340 píxeles, se tiene en cuenta un recuadro de 15x9.6 cm sobre un A4 (21x29.7 cm) vertical. Para pantallas más pequeñas, se tiene en cuenta el tamaño de la pantalla y se le aplica un porcentaje, 87 por ciento para la anchura y un 50 por ciento para la altura (Figura 5.13).

```
if (window.screen.width > 1340) {
    var signBoardWidth = (15 * canvas.width) / 21;
    var signBoardHeight = (9.6 * canvas.height) / 29.7;


    percentualSizes.SignBoardWidth = (15 * 100) / 21;
    percentualSizes.SignBoardHeight = (9.6 * 100) / 29.7;

} else {
    var signBoardWidth = ( canvas.width / (canvas.width / window.screen.width)) * 0.87;
    var signBoardHeight = (canvas.height / (canvas.height / window.screen.height)) * 0.5;


    percentualSizes.SignBoardWidth = (100 / (canvas.width / window.screen.width)) * 0.87;
    percentualSizes.SignBoardHeight = (100 / (canvas.height / window.screen.height)) * 0.5;
}
```

Figura 5.13. Código para el cálculo del canvas de la firma

Capítulo 6

Conclusiones

En este proyecto se ha llevado a cabo la implementación de un gestor documental con funcionalidad de firma digitalizada, mejorando la funcionalidad de firma previamente desarrollada por la empresa, Irene Solutions S.L. Se ha desarrollado el proyecto en base al código desarrollado previamente, mejorandolo conforme a las necesidades y las directrices del supervisor. Se han utilizado tecnologías muy potentes que están en constante desarrollo y mejora como son HTML y JavaScript. Las bibliotecas que se han utilizado son todas de código abierto, por lo que están impulsadas por la comunidad del mismo modo tienen mucha capacidad de crecimiento a largo plazo. Como el sistema contiene versión tanto como para móvil como para ordenador, se convierte en un software mucho más accesible. La encriptación de la firma aporta seguridad para evitar falsificaciones, mientras que almacenar la fecha en la que se efectuó la firma aporta veracidad, conociendo el momento de la firma y si ha sido modificada.

Ha sido un proyecto muy instructivo, desarrollando tecnologías nuevas de las que desconocía su existencia, con grandes posibilidades para desarrollar futuro código. También cabe destacar, la experiencia obtenida sobre los lenguajes de programación utilizados durante la práctica, de los cuales he podido aprender nuevas formas de utilizarlas.

Por otra parte, la experiencia en la empresa me ha permitido conocer cómo se desarrolla un proyecto en un ámbito más profesional, no solo a la hora de desarrollarlo sino también de comercializarlo y distribuirlo. Las estrategias sugeridas por el tutor, desarrolladas por su experiencia, también me han resultado de gran utilidad, tanto para la realización de la aplicación como para la realización de la memoria.

En cuanto a la parte personal, ha sido una experiencia enriquecedora, donde se me ha permitido desarrollarme y mejorar tanto a nivel profesional como personal. También he adquirido experiencia en realizar trabajos de investigación acerca de cómo funciona un software que ya está diseñado. Por otro lado, me ha permitido comprobar la experiencia laboral de un programador en una empresa y las diferencias que aparecen en comparación a hacer un proyecto mientras estudias.

Bibliografía

[1] Repositorio de la biblioteca Signature Pad. https://github.com/szimek/signature_pad
[fecha de consulta: 24 de mayo, 2021]

[2] Repositorio de la biblioteca PDF.js. https://github.com/mozilla/pdf.js
[fecha de consulta: 24 de mayo, 2021]

[3] Progressive Web Apps
https://www.iebschool.com/blog/progressive-web-apps-analitica-usabilidad/
[fecha de consulta: 25 de junio, 2021]

[4] IDE Visual Studio Community 2017.
https://visualstudio.microsoft.com/es/vs/community/
[fecha de consulta: 25 de mayo, 2021]

[5] Sistema de gestión de base de datos MariaDB https://mariadb.org/
[fecha de consulta: 25 de mayo, 2021]

[6] HeidiSQL https://www.heidisql.com/ [fecha de consulta: 25 de mayo, 2021]

[7] Software iText PDF. https://itextpdf.com/es [fecha de consulta: 25 de mayo, 2021]

[8] Web indeed https://es.indeed.com/career/programador-junior/salaries
[fecha de consulta: 26 de mayo, 2021]

[9] Licencia IDE Visual Studio Community
https://visualstudio.microsoft.com/es/license-terms/mlt553321/
[fecha de consulta: 26 de mayo, 2021]

[10] Licencia MIT https://es.wikipedia.org/wiki/Licencia_MIT
[fecha de consulta: 26 de mayo, 2021]

[11] Licencia iText 5
https://kb.itextpdf.com/home/it5kb/faq/is-itext-java-library-free-of-charge-or-are-there-any-fees-to-be-paid [fecha de consulta: 26 de mayo, 2021]

[12] Licencia GPL https://es.wikipedia.org/wiki/GNU_General_Public_License
[fecha de consulta: 26 de mayo, 2021]

[13] GNU Affero General Public License 3.0.
https://www.gnu.org/licenses/agpl-3.0.en.html [fecha de consulta: 27 de mayo, 2021]

[14] JavasScript asincrono
https://developer.mozilla.org/es/docs/Learn/JavaScript/Asynchronous
[fecha de consulta: 28 de mayo, 2021]

[15] XML https://es.wikipedia.org/wiki/Extensible_Markup_Language

[fecha de consulta: 28 de mayo, 2021]

[16] Hash https://es.wikipedia.org/wiki/Funci%C3%B3n_hash
[fecha de consulta: 31 de mayo, 2021]

[17] Base 64 https://es.wikipedia.org/wiki/Base64
[fecha de consulta: 31 de mayo, 2021]

[18] Modelo-Vista-Controlador
https://codigofacilito.com/articulos/mvc-model-view-controller-explicado
[fecha de consulta: 1 de junio, 2021]

[19] Viewport meta tag
https://developer.mozilla.org/en-US/docs/Web/HTML/Viewport_meta_tag
[fecha de consulta: 6 de junio, 2021]

[20] Microsoft Visual Studio Code. https://code.visualstudio.com/
[fecha de consulta: 25 de mayo, 2021]