UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UNA APLICACIÓN DE GESTIÓN DOCUMENTAL

Victor Baeza Dominguez

Tutor: José Enrique López Patiño

Tutor externo: José Ricardo López Simarro

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2021-22

Valencia, 14 de noviembre de 2021

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

En este documento se describe el trabajo desarrollado para la implementación de una aplicación de gestión documental que permita manejar, organizar y almacenar información, procedimientos y archivos de los clientes de una cierta firma de abogados desde una base de datos, suprimiendo la posibilidad de pérdida y otorgando a sus administradores control total sobre las acciones realizadas sobre dicha información. El objetivo principal es facilitar y agilizar el flujo de trabajo de la citada firma que, actualmente, presenta ciertas carencias principalmente referidas a una mala gestión de la información almacenada y control de las alteraciones que esta pueda haber sufrido.
La aplicación presentará un entorno amigable, e intuitivo, que permita al usuario navegar entre la información de los clientes que tenga asignados, y realizar sobre éstos las acciones que se le tengan permitidas dado un cierto tipo de permiso. Siendo pues, mediante permisos, la manera en la que se controlará, por parte del usuario administrador, quien, y de qué manera accede a la información de qué cliente, o a qué información en concreto.
Finalmente se realizarán diferentes registros de información que serán enviados periódicamente a los administradores y, en su caso, a los usuarios pertinentes.

Resum

En aquest document es descriu el treball desenvolupat per a la implementació d'una aplicació de gestió documental que permeti manejar, organitzar i emmagatzemar informació, procediments i fitxers dels clients d'una certa signatura d'advocats des d'una base de dades, suprimint la possibilitat de pèrdua i atorgant als seus administradors control total sobre les accions realitzades sobre aquesta informació. L'objectiu principal és facilitar i agilitzar el flux de treball de l'empresa esmentada que, actualment, presenta certes mancances principalment degudes a una mala gestió de la informació i de les alteracions que aquesta pugui haver patit.
L'aplicació presentarà un entorn amigable i intuïtiu que permeti a l'usuari navegar entre la informació dels clients que tingui assignats, i realitzar sobre aquests les accions que se li tinguin permeses atès un cert tipus de permís. Sent doncs mitjançant permisos la manera com es controlarà, per part de l'usuari administrador, qui i de quina manera accedeix a la informació de quin client, o a quina informació en concret.
Finalment es realitzaran diferents registres d'informació que serà enviada periòdicament als administradors i, si escau, als usuaris pertinents.

Abstract

This document describes the work developed for the implementation of a document management application that allows managing, organizing and storing information, procedures and files of the clients of a certain law firm from a database, eliminating the possibility of loss and giving its administrators total control over the actions carried out on said information. The main objective is to facilitate and streamline the work flow of the aforementioned company, which currently has certain shortcomings mainly due to mismanagement of the information and the alterations that it may have suffered.

The application will present a friendly and intuitive environment that allows the user to navigate between the information of the clients assigned to them, and perform the actions that are allowed given a certain type of permission. Thus, by means of permissions, the way in which the administrator user will control who and in what way accesses the information of which client, or what information in particular.

Finally, different information records will be made that will be sent periodically to the administrators and, where appropriate, to the pertinent users.

Índice

Capítulo 1. Introducción

La organización de la información constituye hoy en día un factor de éxito en las empresas. La introducción del ordenador en el mundo empresarial revolucionó por completo el sector, desarrollándose, más tarde, bases de datos y aplicaciones dedicadas a optimizar la gestión de información, dada la relevancia de este aspecto en el mundo empresarial. Más tarde fue internet quién volvió a revolucionar el sector empresarial, pasándose a controlar un volumen mucho mayor de información que debía ser organizada. Una buena organización de la información garantiza disponer de la información precisa al instante, haciéndola visible solo a aquellos usuarios autorizados, permitiendo una gran coordinación y comunicación entre los miembros de un equipo o empresa, e incrementando y simplificando las labores productivas.

Figura 1. Todo buen sistema de gestión necesita

Ibermedia es una empresa dedicada al mantenimiento, asistencia y desarrollo de servicios informáticos con mas de 20 años de experiencia. Cuenta con un formidable equipo multidisciplinar de profesionales que le permite proporcionar una atención inmediata y satisfactoria a sus clientes, manteniéndose a la vanguardia de la tecnología y la innovación, y ofreciendo un amplio abanico de servicios capaces de abarcar el conjunto de necesidades que sus clientes puedan demandar en el ámbito de la informática y las telecomunicaciones. De entre estos servicios podemos encontrar el desarrollo de aplicaciones a medida para las empresas. Estas aplicaciones permiten automatizar la mayoría de las prácticas de negocio relacionadas con los aspectos operativos o productivos de la empresa, facilitando y centralizando la información de todas las áreas que la componen.

De entre los cuantiosos clientes de Ibermedia encontramos una firma de abogados que se encuentra en situación de requerir una aplicación a medida que le permita paliar las deficiencias organizacionales de información a las que actualmente hace frente. La firma de abogados informa a Ibermedia de ciertas situaciones y circunstancias que se han dado o pueden llegar a darse en las que la información contenida por la empresa pueda llegar a verse comprometida dada la actual gestión que es realizada sobre esta.

Figura 2. Logo de Ibermedia

El desarrollo de este proyecto parte de un estudio realizado a la firma anteriormente citada. Tras un posterior estudio se determina cuales son las necesidades y requerimientos de los profesionales, de manera que se pueda adaptar el trabajo que vienen realizando a la aplicación, que contendrá y manejará la información de manera global, pudiendo ser siempre revisada por parte del usuario administrador. Partiendo pues de las dadas necesidades y requerimientos se procede a idear las funciones, módulos y estructura en general que deberá presentar la aplicación a fin de satisfacer al usuario, y que permitan facilitar y agilizar el flujo de trabajo de la empresa, todo ello presentando un entorno amigable para el menos diestro en burótica, y accesible desde cualquier tipo de dispositivo, ya sea móvil, tablet o ordenador.

En este documento se comentarán la metodología empleada, las fases y procesos ejecutados, y los distintos lenguajes de programación y librerías involucrados en el desarrollo de la aplicación anteriormente mencionada. En él se expondrán, entre otras cosas, los esquemas, diagramas y bocetos en general realizados durante las distintas fases del desarrollo, de manera que el lector pueda verificar los distintos procesos que han sido llevados a cabo.

Capítulo 2. Objetivos del TFG

El objetivo de este trabajo es el desarrollo de la aplicación que permitirá contener toda la información que la citada firma de abogados maneja. Se protegerá la información realizando control de acceso a los documentos mediante uso de permisos y, a su vez, se restringirá el acceso a todo aquel que no se identifique como usuario, evitando así posibles robos de información y accesos indebidos. Se llevará un registro de toda actividad realizada sobre la información que permitirá al administrador/gestor conocer en todo momento la actividad realizada por los profesionales, haya sido realizada de manera local o remota. Monitoreará posibles acciones indebidas o maliciosas contra la información, como puedan ser descargas masivas o modificación de archivos. Finalmente facilitará las labores de los profesionales, haciendo mas sencillo el hecho de compartir información y realizar notificaciones.

Ya que toda la información previamente contenida por los profesionales de la firma será importada a la base de datos de la aplicación, se pretende que el uso de la aplicación reemplace las anteriores prácticas en cuanto a almacenado y tratado de información que, a parte de poco provechosas, podían llegar a poner en riesgo la integridad de la información contenida. Con el producto final, el usuario dispondrá de toda la información de trabajo desde cualquier dispositivo que disponga de una conexión a internet, pudiendo acceder a ella, como ya se ha comentado, de manera organizada y gestionada.

Figura 3. La aplicación incluirá lo necesario para desarrollar la labor de los profesionales, asegurando, gestionando y controlando la información.

Los objetivos globales son:

- Proteger la información.
- Otorgar control sobre la información.
- Facilitar la gestión de información.
- Optimizar el flujo de trabajo.

Figura 4. El desarrollo de una aplicación requiere de varios subprocesos intermedios

Si los objetivos previos han sido alcanzados lo habrá sido, por tanto, el objetivo global final, el desarrollo de la aplicación que permita ordenar, asegurar y gestionar información que, para muchos, puede ser de vital importancia.

Capítulo 3. Metodología de trabajo del TFG

3.1 Gestión del proyecto

3.1.1 Introducción

El proyecto comienza con el estudio que se realiza a la firma de abogados en el que se determina que, efectivamente, la información esta siendo tratada de manera ineficaz e inapropiada, y se evidencia la clara necesidad de implementación de un sistema de gestión de información adecuado a las metodologías propias de los profesionales. A partir de ello se traza un esquema general de las funciones y tareas a desarrollar y se procede a plantear el desarrollo, revisando las tareas que así lo requieran. Se estudian también las herramientas que serán empleadas y las características de cada una ellas.

3.1.2 Herramientas de desarrollo

Los principales programas y lenguajes usados durante el desarrollo del proyecto se listan a continuación:

- Lenguajes de programación: HTML, CSS, Ajax, JavaScript, JQuery, PHP (Framework +Laravel).
- Sistema de administración de bases de datos: MySQL.
- IDE: Visual Studio Code.
- Solución de Paquetes Servidor Web: XAMPP
- Administrador de bases de datos: PHPMyAdmin.
- Explorador: Google Chrome.

Figura 5. HTML, CSS y JavaScript

3.1.3 HTML/CSS

HTML y CSS son los lenguajes básicos para crear una página web. Por un lado, HTML es el lenguaje que ofrece la estructura visual, cabeceras, párrafos, imágenes, introducción de texto o botones son algunos de los contenidos que este lenguaje nos permite añadir a la página web. Por otro lado, CSS es usado sobre las clases HTML, y se encarga de definir las propiedades de estilo, permitiéndonos modificar el diseño de la página web. Estos lenguajes, aunque diferentes, son usados conjuntamente por los desarrolladores web con el fin de generar un diseño limpio y fluido.

Figura 6. HTML/CSS son la base de toda página/aplicación web

3.1.4 JavaScript/JQuery/Ajax

JavaScript es, junto con HTML y CSS, una de las tecnologías centrales de la World Wide Web (www). Habilita las páginas interactivas y es una parte esencial de las aplicaciones web. En este proyecto se ha acompañado de su principal librería multiplataforma, JQuery. JQuery nos permite simplificar el modo de interactuar con los documentos HTML, manejar eventos o desarrollar animaciones. Además, JQuery nos permite generar interacciones con Ajax, una técnica de desarrollo web para crear aplicaciones interactivas, la cual nos permite efectuar cambios en alguna parte de la pagina sin que la pagina sea totalmente recargada, lo cual mejora gratamente la interactividad, velocidad y usabilidad de la aplicación o página web.

Figura 7. JavaScript, JQuery y AJAX permiten el desarrollo de páginas interactivas

3.1.5 PHP/Laravel

PHP es un lenguaje de secuencias de comandos de código abierto ampliamente utilizado, que es especialmente adecuado para el desarrollo web, y puede ser incrustado en HTML. Para el desarrollo de este proyecto se usa Laravel, uno de los mas famosos y usados frameworks de PHP. Laravel es usado para desarrollar en PHP de manera elegante y simple, aportando múltiples funcionalidades:

- Blade: Sistema de plantillas que permiten crear vistas dinámicas en Laravel, muy útil para no repetir código. Además, acceder y nombrar datos desde la plantilla blade se torna una tarea mucho mas simple que con PHP ordinario.
- Eloquent: Aporta un elegante y simple sistema de modelos para interactuar con la base de datos, lo que hace el código menos tedioso y fácil de leer.
- Enrutamiento: Uno de los conceptos mas importantes en el desarrollo web es el sistema de enrutamiento, que permite guiar al usuario por el entramado de páginas que puede visitar, dependiendo de laURL introducida o de la parte de la web en la que éste haga

click. Laravel proporciona un sistema de rutas simple, útil y limpio, que otorga al desarrollador la posibilidad de construir rutas simple o agrupadas basadas en permisos asignados.

- Comunidad y Documentación: Es sabido que Laravel tiene una de las mejores documentaciones para desarolladores en el ámbito de la programación. Existe una amplia comunidad de gente que ha aprendido Laravel basándose únicamente en dicha documentación, ofreciendo una realimentación constante, y un mas amplio y rápido aprendizaje al aspirante a programador en Laravel.

Figura 8. Laravel.

3.1.6 MySQL/PHPMyAdmin

En el desarrollo de este proyecto se ha empleado el sistema de administración de bases de datos MySQL, que fue el visto durante la etapa de aprendizaje. MYSQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual (Licencia pública general/Licencia comercial) por Oracle Corporation, que actualemente esta considerada como la base de datos de código abierto mas popular del mundo, siendo usada por muchos sitios web populares, como Wikipedia, Google (no para búsquedas) y Youtube. Un sistema gestor de bases de datos (SGBD) es un conjunto de programas que permiten el almacenamiento, modificación y extracción de la información en una base de datos. Los usuarios pueden acceder a la información usando herramientas especificas de consulta y de generación de informes, o bien mediante aplicaciones al efecto, como será en nuestro caso. Estos sistemas también proporcionan métodos para mantener la integridad de los datos, para administrar el acceso de los usuarios a los datos y para recuperar información si el sistema se corrompe.Además, y para poder administrar y visualizar la base de datos de una manera mas cómoda (y no mediante la introducción de constantes consultas escritas a mano), se empleará PHPMyAdmin. PHPMyAdmin nos permite crear, de manera rápida y fácil, las conexiones con los servidores MySQL, administrar tablas, filas y datos, o cambiar las configuraciones relacionales de nuestra base de datos, todo desde la misma vista.

Figura 9. MySQL

3.1.7 Visual Studio Code

Visual Studio es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden añadir atajos de teclado y modificar las preferencias. Es gratuito y de código abierto.

Figura 10. Editor de texto Visual Studio Code

3.1.8 XAMPP

XAMPP es la herramienta que sostendrá y hará que nuestra aplicación funcione. XAMP es un paquete de software libre que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los interpretes para lenguajes de script PHp y Perl.
Como ventaja frente a descargar e instalar cada componente por separado y crear o editar sus ficheros de configuración por separado, XAMPP únicamente requiere una pequeña fracción de tiempo para descargarlo y ejecutarlo. XAMPP permite configurar los componentes necesarios de un servidor web de manera sencilla.

Figura 11. XAMPP

3.1.9 Bootstrap

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para el diseño de sitios y aplicaciones web. Esta biblioteca contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menus de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales

Figura 12. Bootstrap

3.1.10 Git/GitHub

Git es un software de control de versiones diseñado pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar un registro de los cambios en archivos incluyendo coordinar el trabajo que varias personas realizan sobre archivos en un repositorio de código. GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones de Git.

Figura 13. Git/GitHub

3.2 Distribución de tareas

Para que la primera versión del proyecto pueda considerarse terminada deben realizarse las siguientes tareas en las que ha sido dividido:

- **Tarea I - Investigación y recolección de datos.** Se realiza una segunda visita a la firma de abogados con el fin de comprender el flujo de trabajo y el tipo de información que se va a manejar, hecho que nos permita desarrollar las funciones que permitan implementar las tareas requeridas por los profesionales, aportando la robustez y agilidad de un buen sistema de gestión de datos. También se estudia la distribución de información que existe actualmente contenida en el servidor de la empresa, y la manera mas sencilla de desarrollar una tarea que nos permita importar la información de la que cada profesional dispone, de manera global, sin romper la estructura jerárquica que pueda haber creada en cada caso.
- **Tarea II – Ajustes de software.** Se obtienen e instalan todas las herramientas que nos permitan desarrollar la aplicación.
- **Tarea III - Esquema general de la base de datos.** Se desarrolla un primer esquema general de la base de datos con el que poder plantear las tablas y las relaciones que van a ser necesarias entre estas. El planteamiento deberá ser lógico y coherente con el fin de poder acceder a la información contenida de manera rápida y sencilla.
- **Tarea IV - Esquema general de funciones.** Una vez desarrollado el esquema de la base de datos se procede a esbozar el mapa de funciones y tareas que se generará sobre este para poder mostrar, manejar, y almacenar la información contenida en la base de datos.
- **Tarea V - Esquema general de vistas.** Se traza el esquema general de las vistas que presentaran al usuario la información requerida. Este esquema se basa plenamente en los dos previos.

- **Tarea VI - Esquema general de tareas programadas.** Se esquematizan las tareas que deberán ser programadas en el sistema operativo y que se ejecutaran de manera periodica.
- **Tarea VII - Implementación de la base de datos.** La base de datos se crea cumpliendo el esquema fijado previamente.
- **Tarea VIII – Implementación de funciones.** Los modelos y controladores contendrán las relaciones y funciones, anteriormente esquematizadas, que administrarán y moveran la información de la base de datos a la aplicación cliente y viceversa.
- **Tarea IX - Implementación de vistas.** La información recibida desde los controladores, y la que será enviada a estos, se presentará en distintas vistas al usuario.
- **Tarea X – Implementación de tareas.** Se añadirán las tareas al sistema operativo que permitirán la ejecución de funciones de manera programada y periódica.
- **Tarea XI - Fase de testeo.** Se programará una nueva reunión con la firma de abogados con el fin de iniciar la fase de testeo general de funcionalidad de la aplicación.
- **Tarea XII - Fase de depuración.** Depuración general de la aplicación tras la fase de testeo previa en la que se corregirán errores, en el caso de que los haya, y se revisarán posibles añadidos.

3.3 Diagrama temporal

Se plantea un diagrama temporal en el que se distribuyen las tareas agrupadas por objetivos y fases: recolección, planteamiento, desarrollo y testeo. Se estima una duración determinada para cada una de ellas, aunque es probable que pueda verse alterada. Además se planifican las tareas por procesos. Como norma general, y al trabajar en un equipo, todo aquello relacionado con el back-end se realizaría por separado del front-end, preparándose ambos para integrarse una vez estén finalizados. Ya que en nuestro caso no contamos con un equipo, se prevé que la tarea relacionada con la implementación de controladores y modelos (back-end) y la relacionada con la implementación de las vistas (front-end) se realizaran de forma simultanea, ya que se testearán todos los métodos posibles desde la vista final de usuario.

Ajustes de Software	Duración
Instalación de XAMPP (Apache, PHP y MySQL).	1 hora
Instalación de Composer.	10 minutos
Instalación de Laravel.	10 minutos
Configuración e instalación de los paquetes (JQuery, Bootstrap, FontAwesome,…) y ajustes del modelo relacional.	1 hora
Creación de la base de datos MySQL y conexión con el servidor local con PHPMyAdmin.	1 hora

Tabla 1. Planificación de tiempo a emplear para los ajustes del software empleado.

Creación e implementación de la base de datos relacional	Duración
Esquema del estado inicial de la base de datos (usuarios,clientes, procedimientos, archivos, permisos).	20 horas
Implementación del estado inicial.	14 horas
Esquema de los cambios y añadidos sobre el estado inicial (datos de usuario, datos de cliente, tipos de permisos,…).	10 horas
Implementación del estado intermedio.	10 horas
Esquema de los cambios y añadidos sobre el estado intermedio (registro, notificaciones, subcarpetas, valores de columnas).	12 horas
Implementación del estado final.	10 horas

Tabla 2. Planificación de tiempo a emplear para los ajustes de la base de datos.

Creación e implementación de base lógica	Duración
Esquema general de funciones.	40 horas
Implementación de funciones.	100 horas

Esquema general de tareas.	4 horas
Implementación de tareas.	8 horas

Tabla 3. Planificación de tiempo a emplear para los ajustes de la lógica de la aplicación.

Creación e implementación de base visual	Duración
Esquema general de vistas.	40 horas
Implementación de vistas.	100 horas
Programación de Ajax y relación de rutas.	20 horas

Tabla 4. Planificación de tiempo a emplear para los ajustes de la parte visual.

Testeo y depuración	Duración
Testeo desarrollador.	40 horas
Testeo usuario.	-
Depuración.	-

Tabla 5. Planificación de tiempo a emplear para testeado y depurado.

Marzo				Abril				Mayo				Junio			
S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S4	S1	S2	S3	S5
I															
II															
	III														
		IV													
			V												
			VI												
				VII											
					VIII										
								IX							
										X					
											XI				
												XII			

Tabla 6. Diagrama temporal.

Capítulo 4. Desarrollo y resultados

4.1 Fase de investigación

4.1.1 Tarea I. Investigación y recolección de datos

El desarrollo del proyecto comienza con la fase de recolección de información, en la que se realiza una visita a la firma de abogados para conocer qué tipo de información es manejada y de qué manera está siendo almacenada (estructura jerárquica de ficheros y directorios que actualmente están en uso) por parte de los presiónales, y revisada por parte de los administradores. Tras la visita contaremos con la información necesaria para proceder con la fase de planteamiento. Una vez realizada la visita se conocen los siguientes datos:

- La firma de abogados dispone de clientes de los que tiene almacenada cierta información (nombre y apellidos, teléfono, dirección, d.n.i, etc…).
- Cada cliente puede poseer varios procedimientos y archivos asociados.
- Un cliente puede ser representado por distintos abogados en distintos procedimientos, y varios abogados pueden representar al mismo cliente en un mismo procedimiento.
- De cada procedimiento se tiene almacenada cierta información (nombre, cliente, plazo, etc,…).
- Un procedimiento posee varios archivos asociados a este.

En cuanto a la estructura jerárquica de directorios y ficheros se obtiene la siguiente información:

- Existe un directorio padre “Z:”.
- Dicho directorio contiene la información almacenada por los profesionales en directorios nombrados con el nombre del profesional.
- Los directorios de profesional contienen directorios nombrados con el nombre de los clientes asignados, además de archivos.
- Cada directorio de cliente contiene directorios nombrados con el nombre del procedimiento en el que se le represente, además de archivos.
- Cada directorio de procedimiento contiene archivos y puede contener, o no, directorios.
- Los directorios que cuelgan de un directorio de procedimiento, les llamaremos directorios de subcarpeta, pueden contener archivos, directorios, o ambos.

Tras la recolección de información se esboza un sistema que otorgue fluidez al acceso a la información, optimizando la estructura de almacenado y estableciendo ciertas medidas y reglas de seguridad que permitan proteger la información de pérdidas y duplicados. El sistema cumplirá con los siguientes requisitos:

- Cada profesional dispondrá de un rol de usuario: administrador o usuario.
- Se realizará control de acceso mediante autenticación de usuario.
- El administrador podrá acceder a toda la información contenida en la base de datos, aunque esta haya sido eliminada desde la aplicación. Dispondrá también de la posibilidad de recuperar información eliminada.
- El usuario dispondrá únicamente de la información que se la haya asignado.
- Existirán tres tipos de permisos con los que la información podrá ser tratada por parte de un usuario: lectura, creación y modificación, que podrán ser modificados en cualquier momento.
- Al iniciarse un procedimiento, se asignarán los usuarios y los permisos de los que estos dispondrán, y se establecerán los plazos requeridos. Los usuarios que hayan sido asignados recibirán una notificación por correo electrónico indicándolo y, además,

recibirán sucesivas notificaciones conforme el vencimiento del plazo del procedimiento vaya aproximándose.

- Los usuarios podrán incluir anotaciones en los procedimientos.
- Cualquier acción que se realice sobre la información quedará reflejada en un registro. Dicho registro mostrará únicamente la información a la que el usuario tenga acceso, salvo en el caso de el administrador que se mostrará toda la información. Diariamente el registro será enviado por correo electrónico a los administradores y usuarios que estos puedan haber establecido.
- Tanto los clientes como cada uno de sus procedimientos contarán con una estructura de subdirectorios.
- De nuevo, tanto los clientes como cada uno de sus procedimientos podrán tener asignados distintos tipos de archivos con los que los usuarios trabajan: .docx, .pdf e imágenes. Estos archivos podrán ser almacenados, visualizados, actualizados y eliminados por parte de los usuarios (que dispongan de los permisos necesarios) y por parte de los administradores.
- Se realizará una importación de la información contenida actualmente de manera individual por cada uno de los usuarios, reestructurándose y organizándose acorde a lo requerido. Se ofrecerá la posibilidad de fusionar posibles clientes duplicados debido a la desestructuración actual.
- La información presentada podrá ser filtrada según los distintos campos necesarios, e indexada por medio de campo de búsqueda.
- Gestión de usuarios por parte del administrador. Posibilidad de cambiar rol de usuario, activar/desactivar usuario, eliminar, …
- Entorno amigable e intuitivo.

Una vez establecidos los requisitos considerados básicos se procede con la fase de planteamiento.

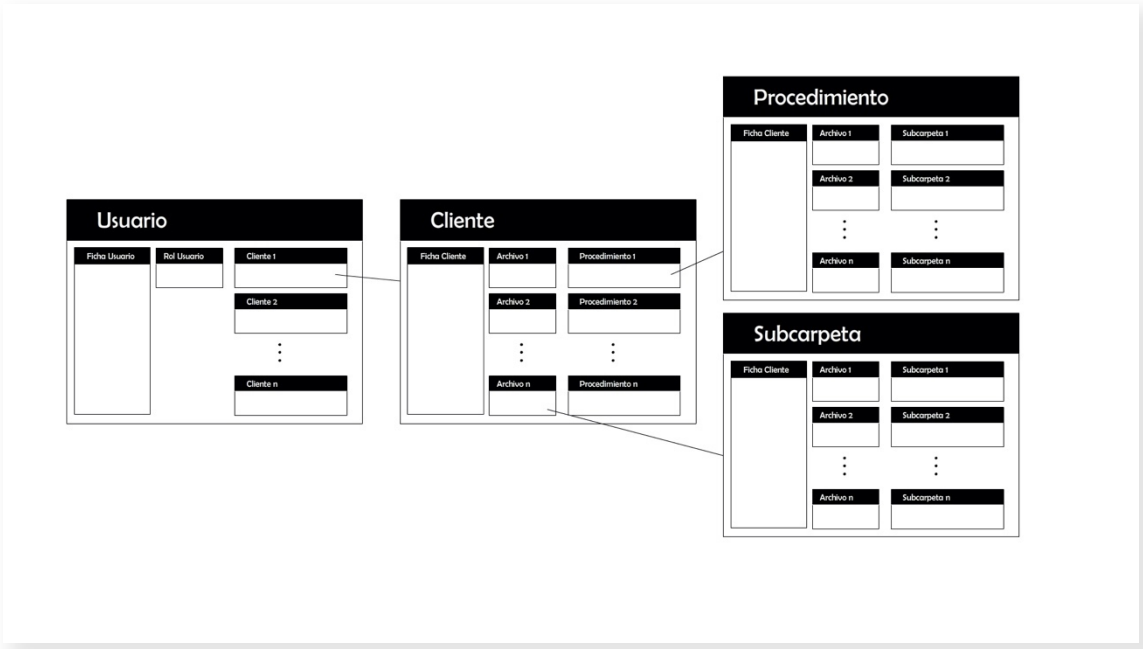


Figura 14. Esquema de objetos de la aplicación.

4.1.2 Tarea II. Ajustes de software

El primer paso será descargar XAMPP e instalarlo en el ordenador en el que trabaremos de forma local previo a migrar el proyecto al servidor remoto donde estará hospedado.

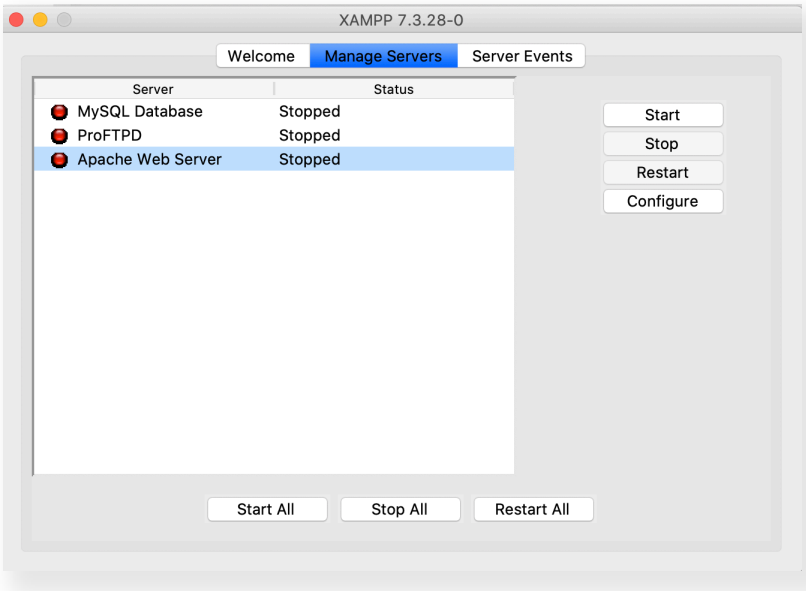


Figura 15. Servidores de XAMPP

Una vez instalado ya contaremos con las dependencias necesarias para correr nuestro proyecto de manera local y gestionar nuestra base de datos sin necesidad de servidores externos.
Disponiendo de la infraestructura necesaria para implementar la base de datos procederemos con la instalación de Laravel, desde el cual generaremos los modelos y las relaciones existentes entre ellos. Para la instalación de Laravel en nuestro proyecto, de sus distintas opciones nos decantamos por la instalación a través de Composer. Composer es un sistema de gestión de paquetes para programar en PHP, el cual provee los formatos estándar necesarios para manejar dependencias y librerías de PHP. Composer trabaja e instala dependencias o librerías desde la línea de comandos. Tambien permite al usuario instalar las aplicaciones PHP que estén disponibles en el "Packagist", el repositorio principal que contiene todos los paquetes disponibles. Dispone, además, de capacidad de auto-descarga para las librerías necesarias que se especifiquen en la información de arranque para así facilitar el uso del código de terceros. Ofrece varios parámetros, entre los que se incluyen:

- require : añade el parámetro de la librería al archivo *composer.json* y lo instala.
- install : instala todas las librerías de *composer.json*. Es el comando que se usa para descargar todas las dependencias PHP desde el repositorio.
- update: actualiza las librerías de *composer.json* de acuerdo a las versiones permitidas que se señalen.
- remove: desinstala una librería y la elimina de *composer.json*.
-

El archivo composer.json es un archivo JSON ubicado en la carpeta raíz del proyecto PHP. Su propósito es especificar las propiedades, metadatos y dependencias de un proyecto común, y es parte de una amplia gama de proyectos existentes.

Una vez nuestro proyecto en Laravel ha sido creado ya podemos acceder al sistema de ficheros y archivos que han sido generados durante la instalación. Para ello, desde Visual Studio, abrimos

la carpeta del proyecto, se nos muestra la pantalla principal del editor, en la que podemos observar la estructura citada:

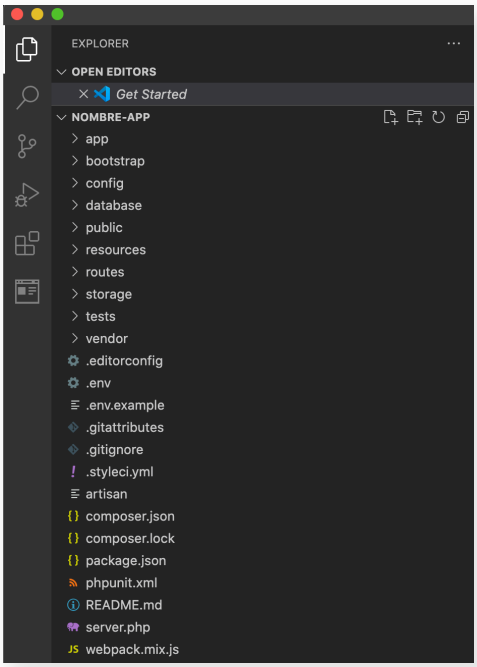


Figura 16. Explorador de archivos de Visual Studio

En dicha estructura podemos encontrar el archivo .env, que contiene algunos valores de configuración comunes que pueden diferir según si nuestra aplicación se ejecuta localmente o en un servidor web de producción. Estos valores luego se recuperan de varios archivos de configuración de Laravel dentro del directorio **config** usando la función *env* de Laravel. También podemos encontrar el archivo *.env.example,* que puede servirnos si desarrollamos junto a un equipo al permitirnos colocar valores de marcador de posición en el archivo de configuración de ejemplo que permitirán al resto de desarrolladores conocer qué variables de entorno son necesarias para ejecutar la aplicación. Es importante mencionar el archivo *.env*, pues será modificado en posteriores tareas.

Figura 17. Archivo .env

Para el desarrollo del fron-end de nuestra aplicación emplearemos Bootstrap. Para instalar Bootstrap introduciremos los siguientes comandos en la línea de comandos del editor de código:

- `php artisan ui bootstrap`
- `npm install` . Descarga el sistema de gestión de paquetes por defecto para Node.js, un entorno de ejecución para JavaScript.
- `npm run dev`. Compila los archivos js y css para que puedan ser utilizados.

```
    <link href="https://cdn.jsdelivr.net/gh/gitbrent/bootstrap4-
toggle@3.6.1/css/bootstrap4-toggle.min.css" rel="stylesheet">

<script src="/vendor/js/bootstrap.min.js"></script>
```

4.2 Fase de planteamiento

4.2.1 Tarea III. Esquema general de la base de datos

Una vez establecidos los requisitos básicos se procede con la fase de planteamiento, que inicia con el desarrollo del esquema general de las tablas que contendrá la base de datos, de sus campos y de la relaciones que existirán entre ellas, a fin de contener una estructura simple y presentar la información de manera correcta y eficaz. Dichas tablas y relaciones serán en las que nos basemos a la hora de proceder con la siguiente fase, la de desarrollo, en la que las relaciones se verán reflejadas en los modelos/objetos que se manejan del lado del servidor previamente a mostrar la información al usuario o administrador. Con unas relaciones coherentemente establecidas seremos capaces de archivar y obtener la información con cortas líneas de código, lo que agilizará la labor de implementación durante la fase de desarrollo. A continuación, se muestra el esquema provisional para la estructuración de la información contenida en la base de datos, que puede verse modificado durante la fase de desarrollo en el caso de que surjan detalles o inconvenientes no tenidos en cuenta.

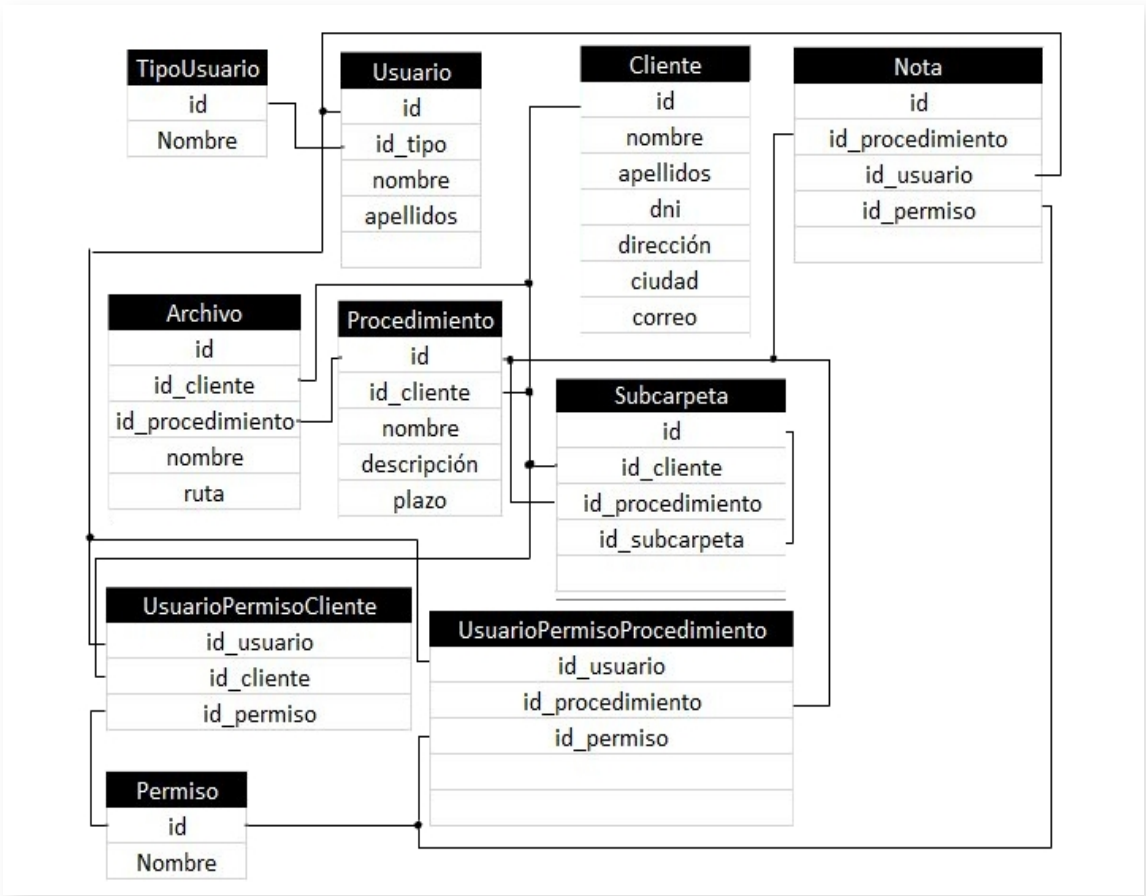


Figura 18. Esquema general de la base de datos

4.2.2 Tarea IV. Esquema general de funciones

Con el esquema general de la base de datos listo podemos continuar la fase de planteamiento iniciando la cuarta tarea, que se basará en establecer las funciones necesarias que permitan manejar la información desde la base de datos a la vista de usuario y viceversa. Atendiendo a los requisitos que el sistema debe cumplir, se traza un esquema de funciones para los objetos que serán manejados por la aplicación en el que se esboza de qué manera podrá ser accedida dicha información. Debemos tener presente que un usuario podrá únicamente acceder a las funciones que le sean permitidas dado su rol de usuario.

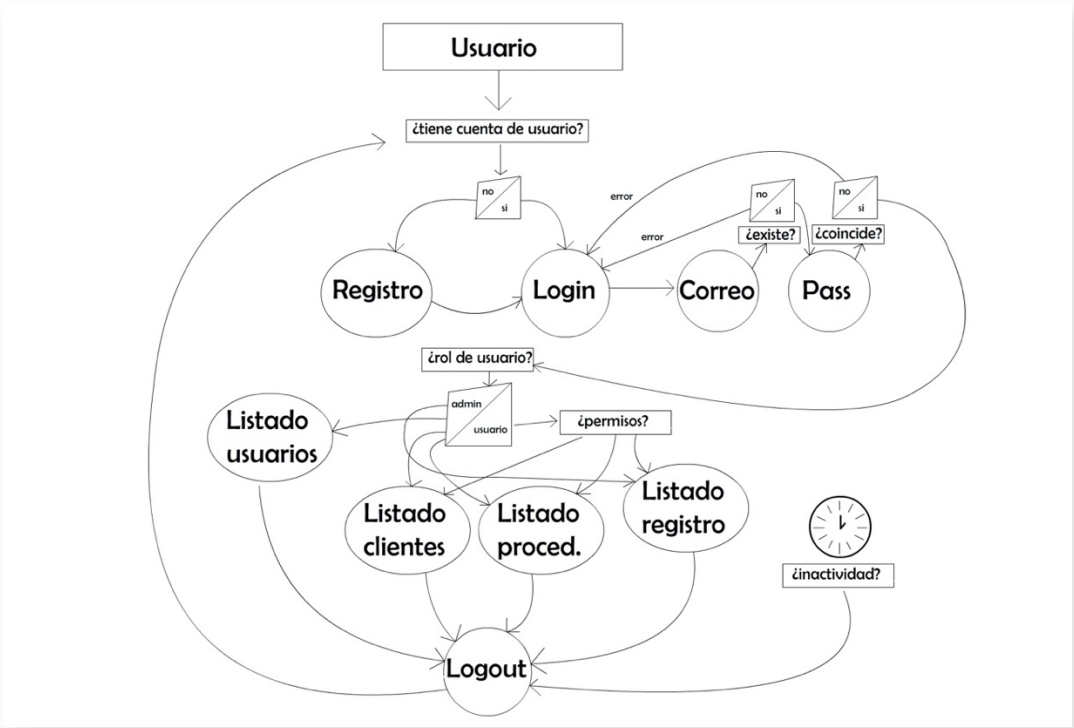


Figura 19. Diagrama de uso de la aplicación

A continuación se comentan brevemente las funciones permitidas a los usuarios según roles de usuario:

- **Usuario**. Un usuario corriente únicamente podrá efectuar los procesos de registr, inicio y cierre de sesión. El usuario administrador podrá realizar acciones de edición/creación sobre los usuarios, además de la posibilidad de listar todos los usuarios e importarlos, entre otros.

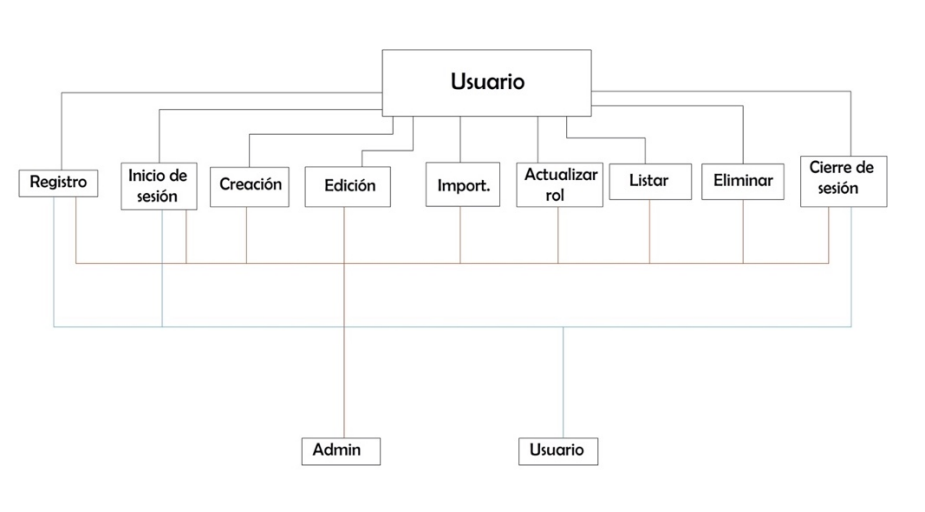


Figura 20. Funciones asociadas al objeto "usuario"

- **Cliente**. El usuario corriente listará aquellos clientes, con sus procedimientos y archivos, que tenga asignados mediante el uso de permisos, y podrá realizar acciones de edición/creado en aquellos sobre los que tenga asignados los permisos pertinentes. Podrá

además realizar búsquedas de dichos clientes desde la vista de usuario. El administrador podrá aparte actualizar los permisos asociados a dicho cliente e importar clientes del directorio de trabajo actual.

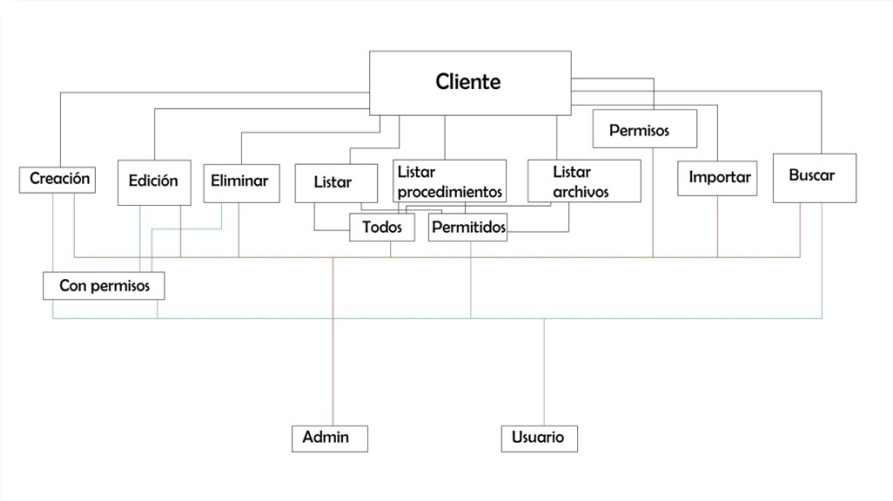


Figura 21. Funciones asociadas al objeto "cliente"

- **Procedimiento**. El usuario corriente listará aquellos procedimientos, con sus archivos, que tenga asignados mediante el uso de permisos, y podrá realizar acciones de edición/creado en aquellos sobre los que tenga asignados los permisos pertinentes. De nuevo, podrá además realizar búsquedas de dichos procedimientos desde la vista de usuario. El administrador podrá aparte actualizar los permisos asociados a dicho procedimiento.

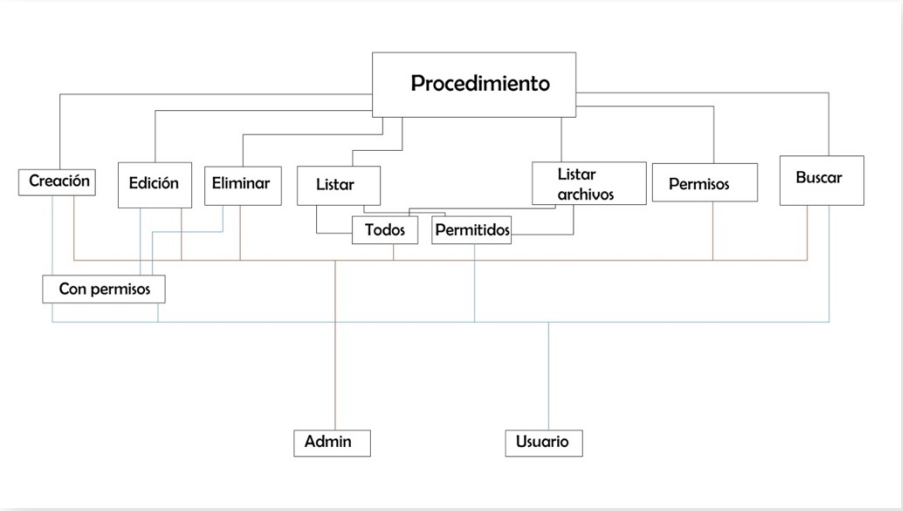


Figura 22. Funciones asociadas al objeto "procedimiento"

Archivo. El usuario corriente listará aquellos archivos que tenga asignados mediante el uso de permisos, salvo que, en este caso, si no se han modificado previamente por un administrador, los usuarios dispondrán de los mismos permisos sobre el archivo de los que disponían sobre el procedimiento al que pertenecen. El usuario podrá realizar acciones de visualización y edición/creado en aquellos sobre los que tenga asignados los

permisos pertinentes. De nuevo, podrá además realizar búsquedas de dichos procedimientos desde la vista de usuario. El administrador podrá aparte actualizar los permisos asociados a dicho archivo.

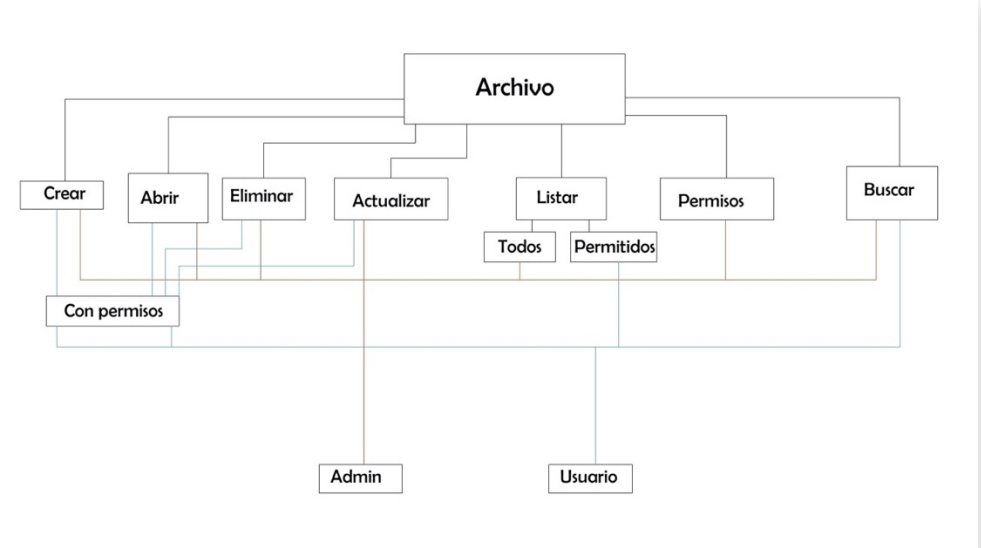


Figura 23. Funciones asociadas al objeto "archivo"

De nuevo, y al tratarse de la fase de planteamiento, el esquema presenta un planteamiento de funciones básicas requeridas, a las que posteriormente podrán sumárseles nuevas funciones implementadas con el objetivo de cubrir aspectos no tenidos en cuenta o que puedan ser sugeridos a modificación durante el proceso de planteamiento y desarrollo por los propios profesionales que finalmente harán uso de la aplicación. No obstante, con lo disponible podremos proceder con la siguiente tarea, en la que se perfilarán las vistas que presentarán finalmente la información requerida al usuario.

4.2.3 Tarea V. Esquema general de vistas

Consiste en el diseño de un boceto de vistas generales que constituyan el front-end de la aplicación y sean el entorno final con el que el usuario interactuará. Para ello, basándonos en los esquemas previos, procedemos con el diseño de las distintas vistas, tanto para el usuario común como para el administrador, que serán diseñadas de forma que puedan ser compartidas, mostrando a cada tipo de usuario (rol) la información permitida. A continuación, se listan las vistas implementadas:

- Vistas de usuario:
  - Vista de menú clientes y distintas posibles visualizaciones y configuraciones.
  - Vista de menú procedimientos y distintas posibles visualizaciones y configuraciones.

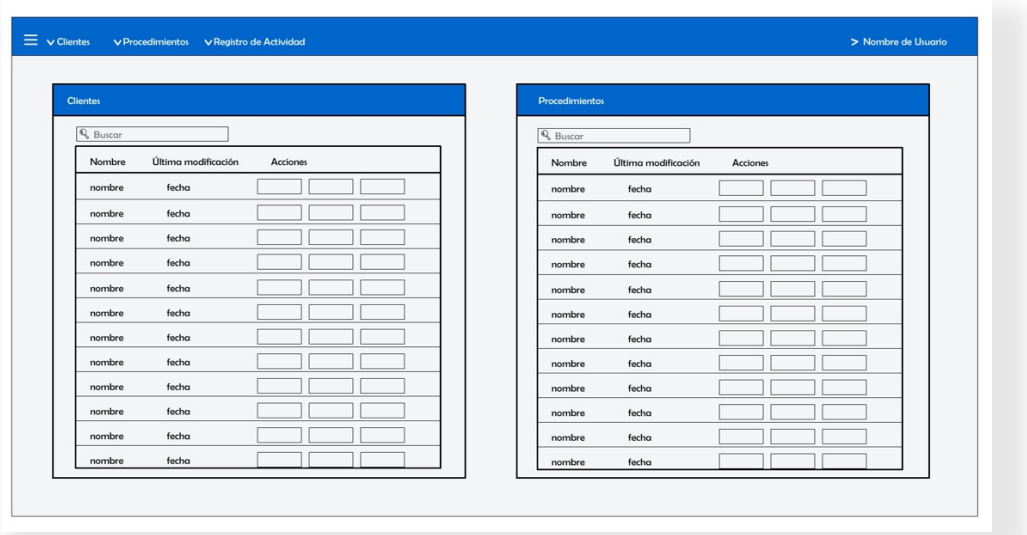


Figura 24. Vista de clientes del panel de usuario

Vistas de administrador:

- Vista de menú clientes y distintas posibles visualizaciones y configuraciones.
- Vista de menú procedimientos y distintas posibles visualizaciones y configuraciones.
- Vista de menú usuarios y distintas posibles visualizaciones y configuraciones.

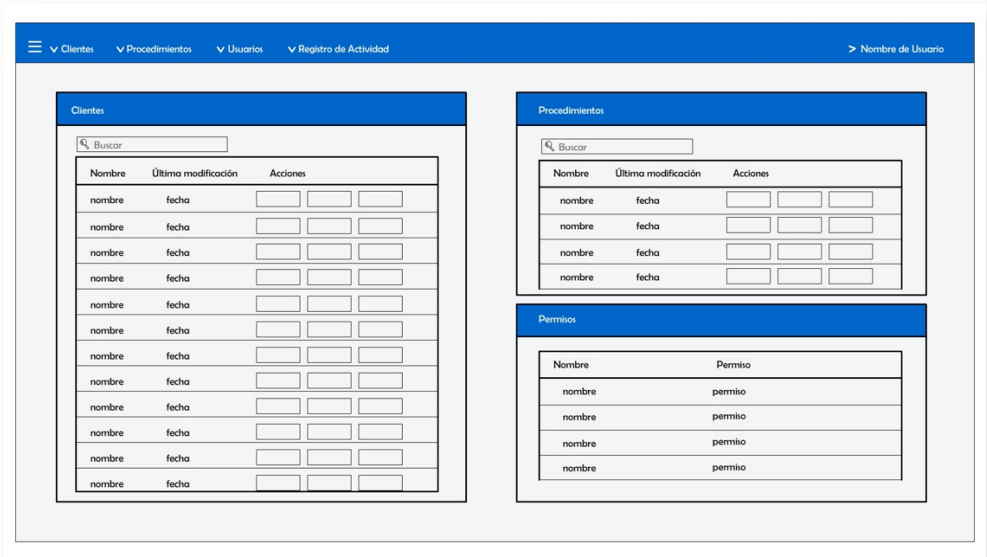


Figura 25. Vista de clientes del panel de administrador

- Vistas compartidas
  - Vista de inicio de sesión.
  - Vista de registro de actividad.
  - Vista de edición de perfil de usuario.

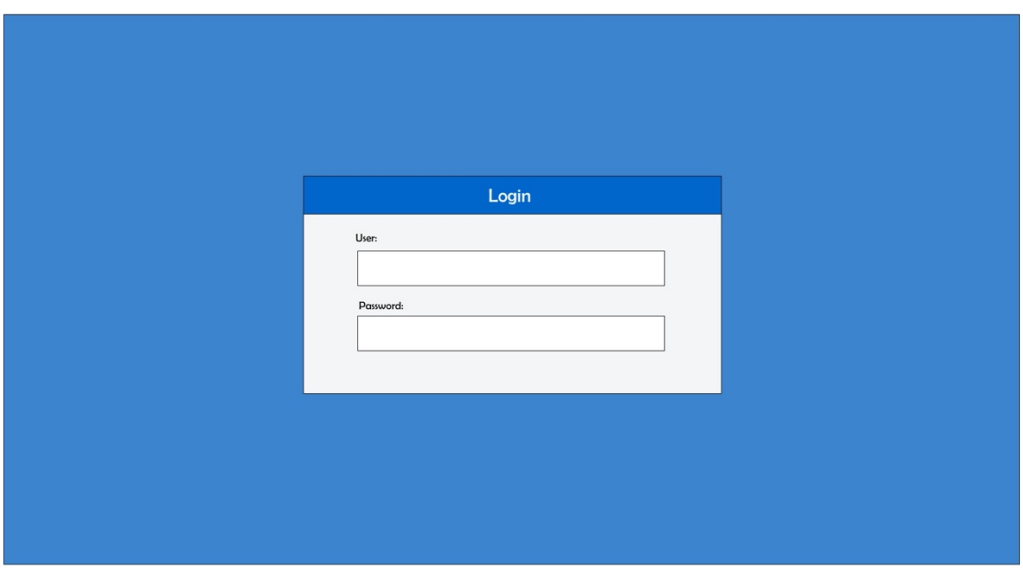


Figura 26. Ventana de inicio de sesión

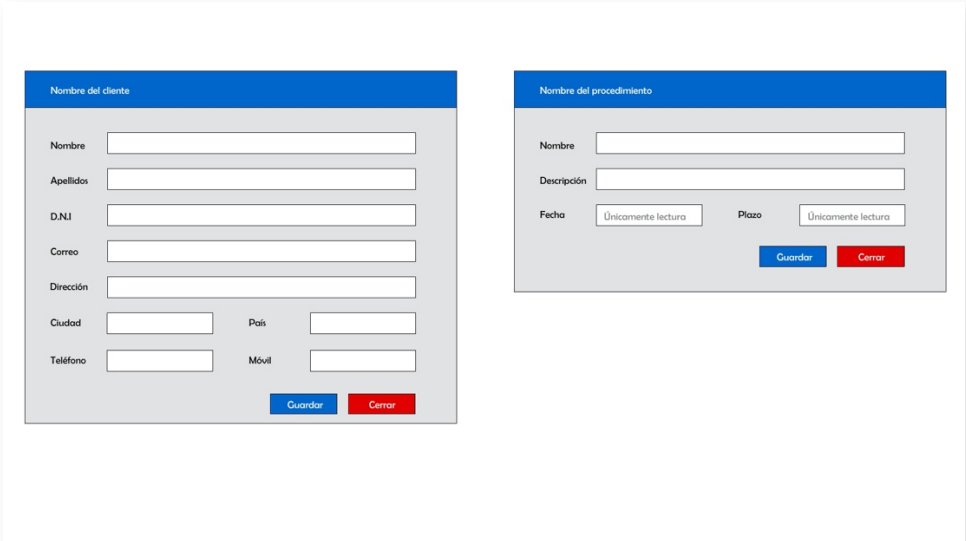


Figura 27. Ventanas de edición de cliente y procedimiento

- Menú de usuario

Figura 28. Menú de navegación de usuario

- Menú de administrador

Figura 29. Menú de navegación de administrador.

Al tratarse de bocetos pertenecientes a la fase de planteamiento, estas vistas podrían verse modificadas en un futuro, pero nos servirán de orientación en los primeros pasos de creación del front-end para alcanzar un global satisfactorio, ya que cumplen con los requisitos previamente establecidos y que consideramos básicos, entre ellos el de presentar un entorno amigable e intuitivo.

4.2.4 Tarea VI. Esquema general de tareas programadas

Dada la necesidad de envío de ciertas notificaciones de manera periódica por parte de la aplicación, se programan tareas en el sistema operativo que va a contener la aplicación en un futuro, en este caso Windows Server, a fin de que se ejecuten los comandos relacionados con dicho envío, implementados en la aplicación previamente. Se establecen dos tareas principales, las cuales serán:

- Envió diario del registro de actividad a los administradores y aquellos usuarios que los administradores hayan establecido. Esta tarea podrá ser llevada a cabo también por un administrador desde el menú de administración.
- Envío diario de la fecha de vencimiento de plazo de los procedimientos que un usuario tenga asignado. Este comando únicamente será ejecutado si la diferencia entre la fecha actual y la fecha de vencimiento de plazo es menor o igual a siete días.

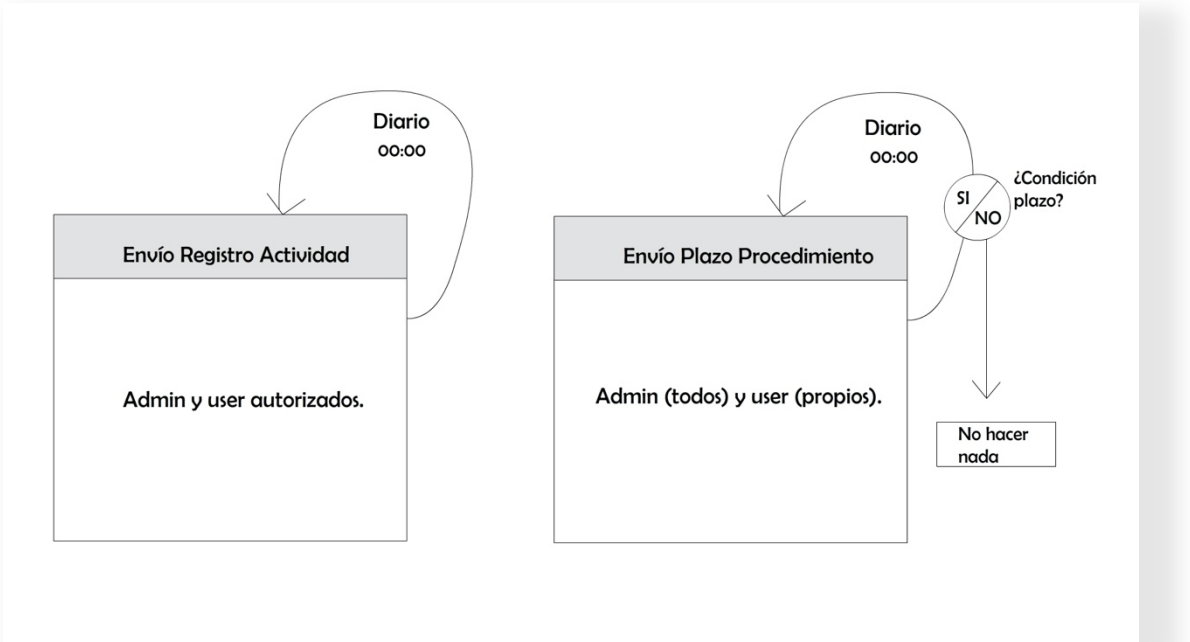


Figura 30. Tareas programadas.

Finalizada la sexta tarea se da por finalizada la fase de planteamiento de la que, como se ha comentado, es posible que haya que corregir alguno o varios aspectos según se vayan implementando y testeando los esquemas y modelos propuestos. Es hora pues de proceder con la fase de desarrollo y sus distintas tareas.

4.3 Fase de desarrollo

La fase de planteamiento ha finalizado y es, por tanto, hora de pasar al desarrollo e implementación de la base lógica de la aplicación. Como se ha citado previamente, para el desarrollo de la aplicación se empleará el framework Laravel, que correrá bajo XAMPP. Laravel ofrece un enfoque para acceso de datos en una base de datos en el que cada tabla tiene una clase o modelo correspondiente que se utiliza para interactuar con dicha tabla. A parte del uso de modelos haremos uso de otra característica de la que dispone Laravel que son las migraciones. Las migraciones son un control de versiones para la base de datos que nos permite definir y compartir la definición del esquema de la base de datos de nuestra aplicación, lo que nos hacer olvidarnos de tener que estar modificando "manualmente" las tablas de la base de datos cada vez que se produzca un cambio. Laravel proporciona soporte independiente de la base de datos para crear y manipular tablas en todos los sistemas de bases de datos compatibles. Según se entre en siguientes tareas se expondrá lo requerido del framework en cada una de ellas.

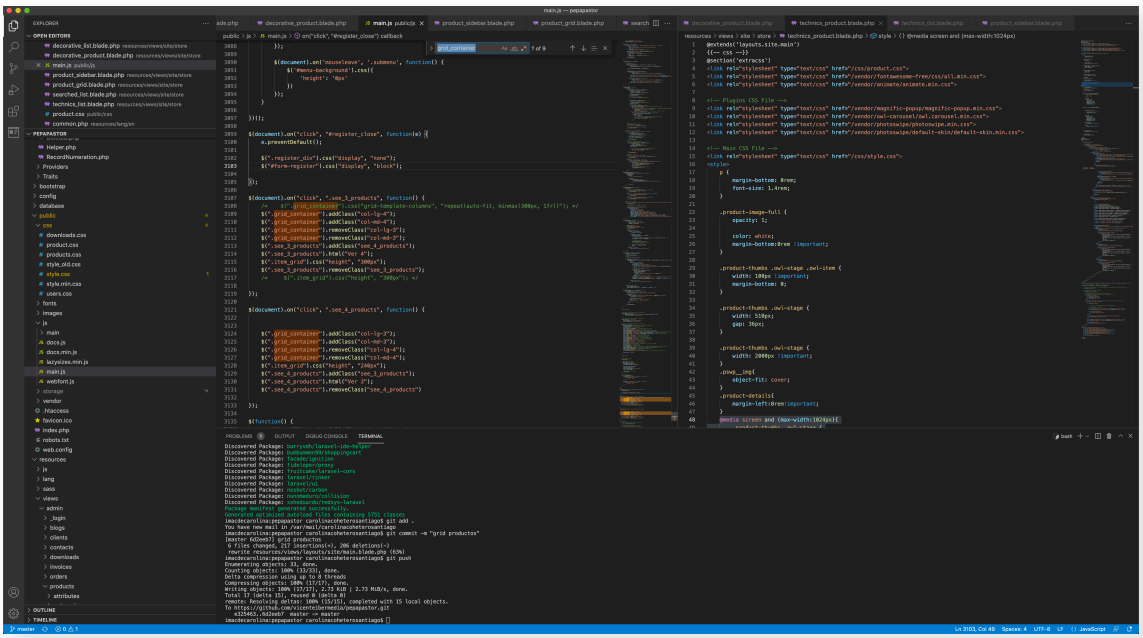


Figura 31. Vista de Visual Studio Code

4.3.1 Tarea VII. Implementación de la base de datos

La base de datos se creará desde PHPMyAdmin. Para acceder a PHPMyAdmin, una vez inciados los servicios de XAMPP, basta con introducir la correspondiente url en el navegador (http://localhost/phpmyadmin).

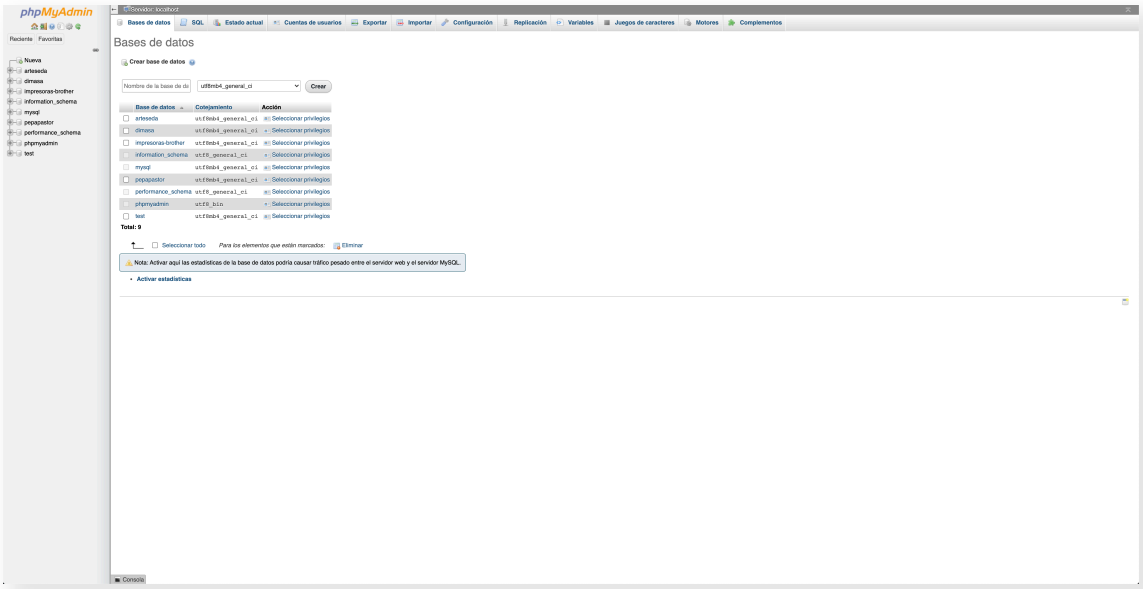


Figura 32. PHPMyAdmin

La base de datos esta creada pero esta vacía ya que no hemos especificado ninguna tabla, para comenzar a disponer su estructura se debe modificar el archivo *.env* para que Laravel conozca los datos de base de datos con la que debe conectar al ejecutar nuestra aplicación. Los campos a modificar son los siguientes:

- DB_DATABASE: indica el nombre de la base de datos.
- DB_USERNAME: nombre de usuario de acceso a la base de datos.
- DB_PASSWORD: contraseña de acceso.

El siguiente paso será generar las migraciones que nos permitan crear las tablas en la base de datos, atendiendo a los esquemas que previamente hemos fijado. Una característica de editores de código como el seleccionado es la posibilidad de tener abierta la consola en la misma pantalla del editor, desde la cual podemos introducir los comandos que nos permitan generar clases y otros componentes de nuestra aplicación en desarrollo.

Accediendo (desde el explorador de Virtual Studio) a database/migrations encontraremos las clases de migraciones generadas.

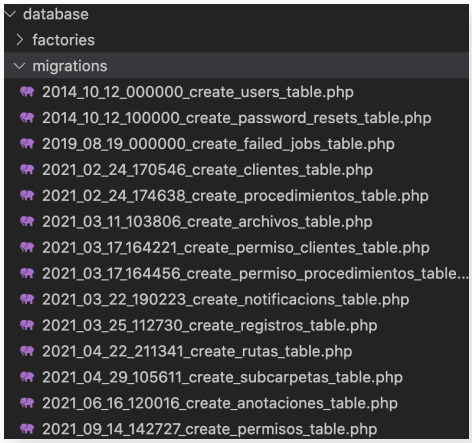


Figura 33. Directorio de migraciones.

Una clase de migración contiene dos métodos: up y down. El método up se utiliza para agregar tablas nuevas, columnas o índices a la base de datos, mientras que el método down se utiliza para revertir las operaciones realizadas por el método up. Dentro de estos dos métodos se puede emplear el generador de esquemas (Schema::create()) de Laravel para crear y modificar tablas de forma expresiva. A continuación se comenta las configuraciones de las distintas clases de migración creadas:

- **Migración de usuarios**. Dado que un usuario necesita tener un rol de usuario asignado, deberemos generar la tabla de roles de usuario e introducir valores predeterminados de los tipos de rol que serán asignables, en este caso: administrador y usuario. A la hora de crear la tabla de usuarios, declaramos las relaciones e insertamos, de nuevo, ciertos valores predeterminados que nos permitan testear las funciones que serán implementadas posteriormente, así como acceder a la aplicación en sí. Laravel nos ofrece la posibilidad de incluir tokens para evitar ataques del tipo "Remember Me cookie" actualizando el valor después del login y el logout y evitando que un individuo malicioso que se haya hecho con la cookie haga uso de ella, y campos del tipo timestamp, que nos permiten registrar fechas de creación y actualización de los registros contenidos en las tablas.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('tipos_users', function (Blueprint $table) {
            $table->id();
            $table->char('nombre');
        });

        DB::table('tipos_users')->insert([
            ['id' => 1, 'nombre' => 'Administrador'],
            ['id' => 2, 'nombre' => 'Usuario'],

        ]);



        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('avatar')->default('default.png');
            $table->foreignID('tipo_id')
                ->constrained('tipos_users')
                ->onUpdate('cascade')
                ->onDelete('cascade');
            $table->boolean('activo')->default(1);
            $table->string('conexion');
            $table->boolean('conectado')->default(0);
            $table->rememberToken();
            $table->timestamp('created_at')->default(\DB::raw('CURRENT_TIMESTAMP'));
            $table->timestamp('updated_at')->default(\DB::raw('CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP'));
        });
        $hash = Hash::make('password');
        DB::table('users')->insert([
            ['id' => 1, 'name' => 'Victor', 'email' => 'victor.baeza.d@gmail.com', 'password' => $hash, 'tipo_id' => 1],
            ['id' => 2, 'name' => 'Caro', 'email' => 'coheterocarolina@gmail.com', 'password' => $hash, 'tipo_id' => 2],
            ['id' => 3, 'name' => 'Dr. Slump', 'email' => 'slump@gmail.com', 'password' => $hash, 'tipo_id' => 2],
            ['id' => 4, 'name' => 'Chicho Terremoto', 'email' => 'terremoto@gmail.com', 'password' => $hash, 'tipo_id' => 2],
        ]);
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('tipos_users');
    }
}
```

Figura 34. Clase de migración de Laravel.

- **Migración de clientes.** En la migración de la tabla de clientes se hace uso de otra posibilidad ofrecida por Laravel, que es el estilo de borrado *softDeletes(),* que marca los elementos indicando que ya no están disponibles en el sistema, ocultando esos registros de las operaciones del usuario. Haciendo uso de *softDeletes()* podremos presentar toda la información, aun habiendo sido borrada.
- **Migración de procedimientos.** Se establecen los campos necesarios, junto con los de control de borrado y actualización.
- **Migración de archivos.** Se establecen los campos necesarios, junto con los de control de borrado y actualización
- **Migración de permisos.** Se crea la tabla y se rellena con valores predeterminados que corresponden a los distintos permisos que un usuario puede tener asignados sobre un cliente o procedimiento.
- **Migración de permiso_cliente.** Se crean los campos necesarios para establecer las relaciones entre el usuario y el tipo de permiso asignado sobre un cliente.
- **Migración de permiso_procedimiento.** Se crean los campos necesarios para establecer las relaciones entre el usuario y el tipo de permiso asignado sobre un procedimiento.
- **Migración de registros.** Esta tabla contendrá los campos necesarios para almacenar la información requerida en cuanto a modificación de un usuario, archivo, procedimiento o cliente.
- **Migración de subcarpetas**. Dado que una subcarpeta puede pertenecer a un cliente, procedimiento o a otra subcarpeta, se implementan los campos necesarios para generar las distintas relaciones. Estos campos se establecen como *nullable(),* lo que indica que no es un valor requerido a la hora de rellenar la tabla y, si no existe dicho valor, no obtendremos ningún error.

Teniendo todas las clases de migración listas, el siguiente paso es realizar la migración que generará las tablas en nuestra base de datos. Esto lo haremos desde la línea de comandos.
Si al ejecutar el comando oportuno no obtenemos ningún error nuestra base de datos se habrá rellenado con las tablas que hemos creado.

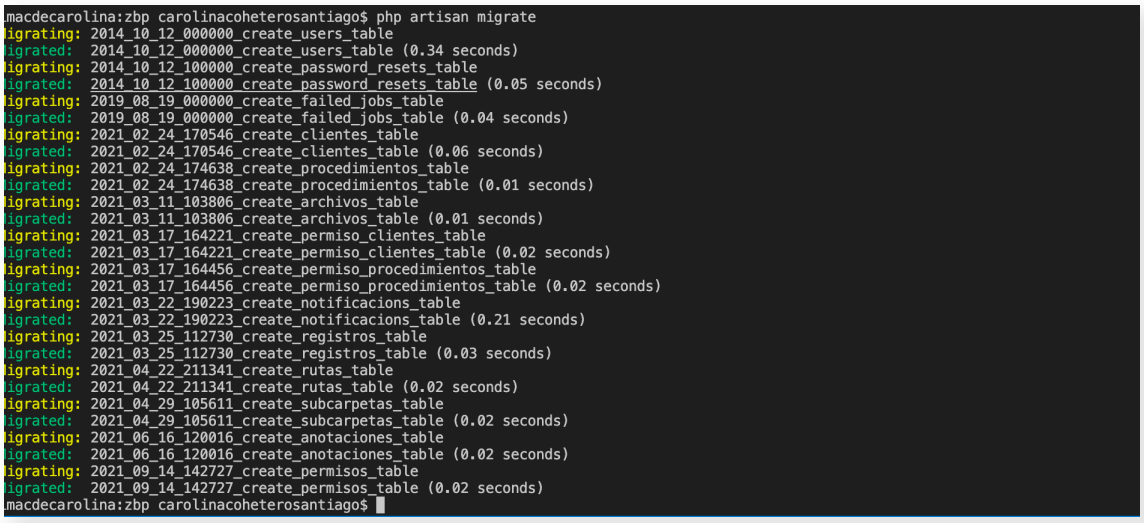


Figura 35. Terminal.

Este proceso podríamos haberlo realizado también de forma individual por cada clase de migración, añadiéndose cada tabla a las ya creadas previamente en la base de datos, así que si posteriormente nos encontramos con la necesidad de añadir una nueva tabla no nos veremos obligados a regenerar de nuevo todas y cada una de ellas. Mediante línea de comandos, con la

interfaz Artisan, disponemos de otros comandos que nos pueden ser muy útiles a la hora de manejarnos con nuestra base de datos, entre ellos cabe destacar dos en concreto: rollback y refresh. Rollback nos permite revertir la ultima acción de migrado efectuada sobre la base de datos, mientras que refresh revertirá todas las migraciones y ejecutará a continuación la migración de todas las clases de migración generadas previamente.
Una vez realizada la migración volvemos a acceder a phpMyAdmin desde el navegador para comprobar que las tablas de nuestra base de datos han sido creadas y se encuentran disponibles para empezar a almacenar información.

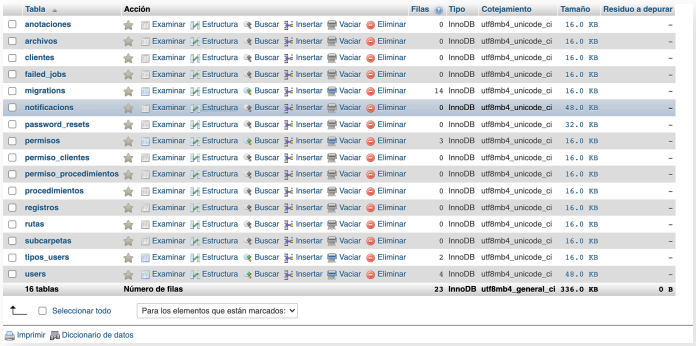


Figura 36. Tablas de la base de datos.

Como se observa junto a las tablas comentadas se han generado otras que puedan llegar a ser usadas en una posterior versión de la aplicación, mas no en la presente.

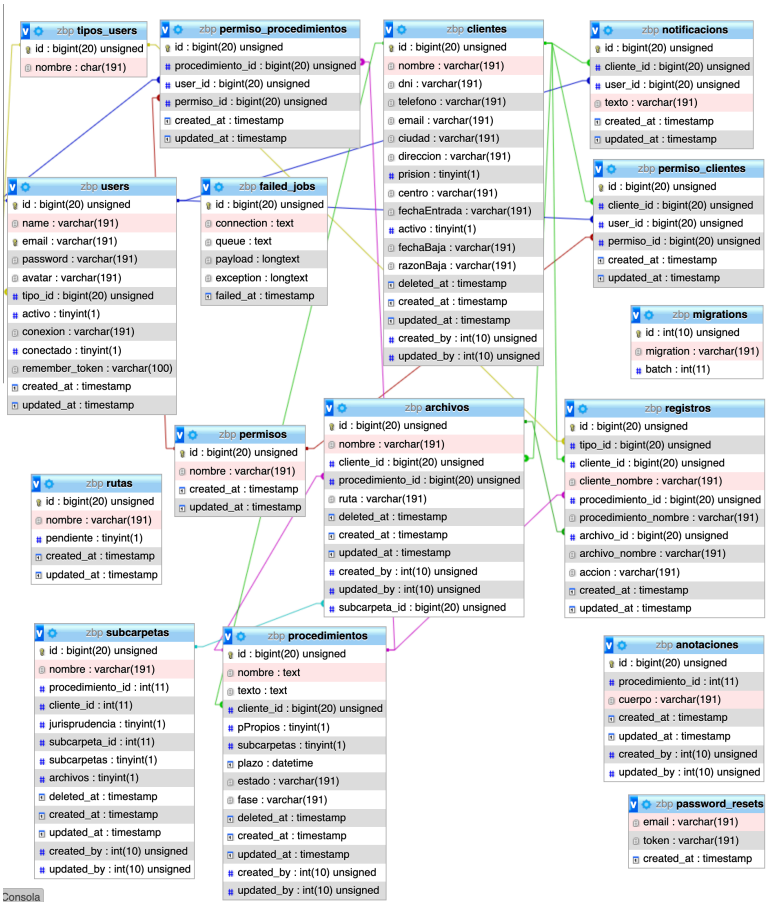


Figura 37. Tablas de la base de datos.

4.3.2 Tarea VIII. Implementación de funciones

Continuando con la fase de desarrollo y disponiendo de la base de datos, con sus tablas, lista para funcionar, debemos implementar los modelos que estarán asociados a cada tabla con tal de poder generar las relaciones y métodos propios de cada modelo. Como hemos visto, cuando se usa Eloquent, cada tabla de la base de datos tiene un "Modelo" correspondiente que se usa para interactuar con esa tabla. Además de recuperar registros de la tabla de la base de datos, los modelos Eloquent permiten también insertar, actualizar y eliminar registros de la tabla.
Una vez creado el modelo podemos acceder a el desde el navegador del editor de código. En cada modelo declararemos, entre otras cosas, las relaciones existentes entre modelos, que reflejan las existentes entre tablas. A fin de establecerlas, deberemos comprender el léxico que Laravel emplea con este propósito:

- *hasOne():* indica que la relación generada entre tablas es de 1:1. Esto quiere decir que cada registro contenido en la tabla/modelo tendrá un único registro asociado en la tabla/modelo con la que tenga dicha relación.
- *hasMany():* indica que la relación entre tablas es de 1: ∞. Esto significa que el registro de la tabla/modelo que tenga implementada dicha relación tendrá varios registros asociados en la tabla/modelo con la que tenga dicha relación.
- *belongsTo():* igual que la anterior, indica que la relación entre tablas es de 1: ∞, salvo que en este caso la tabla/modelo tiene varios registros asociados a un registro de la tabla/modelo con la que mantiene la relación.
- *belongsToMany():* indica que la relación es ∞:∞, lo que significa que ambas tablas contienen varios registros asociados a otros varios de la tabla/modelo con la que mantienen la relación.

En app/models encontraremos las clases de modelos generadas:

- **Modelo de usuario**. Veremos este modelo a tipo de ejemplo. El modelo de usuario es creado de forma automática por Laravel y contiene, entre otras cosas, las variables necesarias para efectuar el control de autenticación del usuario (*$hidden* y *$casts*). La propiedad *$fillable* es necesaria, y hace que todos los modelos de Eloquent estén protegidos contra vulnerabilidades del tipo asignación masiva de forma predeterminada. Se ha generado la relación (1:1) con el modelo TipoUser, que establece el tipo de usuario (admin o user), la relación (1: ∞) con el modelo Registro, que relacionará al usuario con la información asociada a este en el registro de actividad, y la relación (1: ∞) con el modelo Cliente, que relacionará a un usuario con sus clientes asignados.

```php
<?php

namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

use Cliente;
use TiposUser;

class User extends Authenticatable
{
    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password','avatar', 'tipo_id',
    ];

    protected $table='users';
    public $timestamps = true;
    protected $primaryKey = 'id';

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function Tipo(){

        return $this->hasOne(TiposUser::class, 'id', 'tipo_id');

    }



    public function registros(){

        return $this->hasMany(Registro::class,'user_id','id');

    }

    public function clientes(){

        return $this->belongsToMany(Cliente::class)->orderby('updated_at','DESC');


    }
}
```

Figura 38. Clase de modelo de Laravel

- **Modelo de tipos de usuario.**
- **Modelo de cliente.**
- **Modelo de procedimiento.**
- **Modelo de archivo.**
- **Modelo de subcarpeta.**
- **Modelo de anotación.**

Disponiendo de los modelos implementados podemos dar paso a la implementación de los controladores. Los controladores pueden agrupar solicitudes relacionadas manejando la lógica en una sola clase, por ejemplo, una clase de controlador UserController (controlador de usuario) manejará todas las solicitudes entrantes relacionadas con los usuarios, lo que incluye mostrar, crear, actualizar y eliminar usuarios. De forma predeterminada, los controladores los almacenaremos en el directorio app/Http/Controllers de nuestra aplicación.

Artisan nos permite también generar todo lo visto hasta ahora (migración, modelo y controlador) mediante un único comando, para ello deberíamos introducir lo siguiente:

```
php artisan make:model Nombre –mc
```

En la línea anterior se está generando la clase de modelo y se indica que se quiere generar también la clase de migración y la clase de controlador mediante -m (migración) y -c (controlador).
Como ya sabemos, los controladores deberán contener los métodos que nos permitan manejar toda la información asociada a estos, de manera que sea accesible tanto desde el backend como desde el frontend de nuestra aplicación. Procedemos pues a implementar los distintos métodos esquematizados previamente para cada uno de los modelos.

Como se comentaba en el planteamiento incial del diagrama temporal, la tarea de implementación de modelos y controladores y la tarea de implementación de vistas se realizó prácticamente de manera conjunta dado que se pretendían probar todas las funcionalidades de la aplicación sobre la vista de usuario final, con el fin de poder replantear todo aquello que pudiese haber quedado esquematizado previamente de manera poco provechosa o conveniente. De todos modos, en este documento ambas tareas se presentan de forma independiente con la meta de proporcionar una mas fácil lectura y comprensión de los distintos procesos realizados.

- **Controlador de usuario.** Se generan los métodos que nos permiten crear, obtener, almacenar, listar, buscar, y paginar usuarios, además de los métodos relacionados con el acceso al perfil del usuario, el cambio y envío de contraseña. Se crean también los métodos involucrados en el proceso de importación de un usuario y en el seguimiento y actualización de actividad por parte del administrador. Para los distintos flujos de manejo de la información emplearemos las funciones y clases necesarias a las que atiende Laravel. En cuanto a los controladores relacionados con la autenticación del usuario, Laravel y, en concreto, el paquete laravel/ui proporcionan una forma rápida de estructurar todas las rutas y vistas que se necesitan para la autenticación. El paquete laravel/ui generará también varios controladores de autenticación prediseñados, que podemos encontrar en App/Http/Controllers/Auth, entre ellos: RegisterController y LoginController. A continuación se comentan algunas de las funciones/métodos implementadas finalmente en el controlador de usuario:
    - **Listar usuarios.** Nos permite obtener un listado de todos los usuarios contenidos en la base de datos, ordenados por fecha de actualización descendente y con un máximo de 15 usuarios por página.
    - **Obtener tabla de usuarios.** Devuelve la colección de usuarios almacenada en la base de datos a una tabla que será mostrada en la vista de usuario. La información puede devolverse en su totalidad o filtrada.
    - **Almacenar usuario.** Crea un nuevo usuario que es almacenado en la base de datos con los datos introducidos desde la vista de usuario. Se genera también el registro correspondiente que indica que dicho usuario ha sido creado y quién lo ha creado. Una vez el usuario ha sido creado se devuelve la vista que lista los usuarios actuales.
    - **Restablecer contraseña.** Se restablece la contraseña de usuario mediante envio de un código a la dirección del usuario.
    - **Obtener directorios de usuarios actuales.** Se obtienen los directorios de la ruta especificada previamente en la vista de usuario (en este caso administrador), y se genera un array de nombres de usuario a partir del nombre de cada fichero contenido en dicha ruta. Este proceso se basa en la jerarquía existente en el actual directorio de ficheros del servidor de la firma de abogados. Una vez obtenidos los usuarios, se devuelve a la vista de importación junto con el listado de los usuarios a importar.
    - **Importar usuarios.** Importa el usuario o usuarios seleccionados desde la vista de importación. Se realizan previos ajustes a los nombres de usuario con tal de no obtener caracteres no validos para la base de datos o que puedan llegar a

generar cierto tipo de inconvenientes. Una vez tratado el nombre se genera el usuario, al que se le asigna una contraseña y correo electrónico genéricos que podrá modificar a posteriori.

- **Buscar un usuario.** Función que nos permitirá indexar los usuarios desde la vista final de usuario (administrador). Un usuario puede buscarse por nombre, email o por estado de actividad, obteniéndose siempre ordenados por fecha de modificación descendente.
- **Actualizar actividad de usuario.** Permite a un usuario administrados activar o desactivar a otro usuario. Se genera un registro que indicará la acción ejecutada y el ejecutor.

- **Controlador de cliente.** Se generan, además de las funciones que nos permitan crear, obtener, actualizar y eliminar un cliente, aquellas que nos permitan listarlo, paginarlo e indexarlo. Entre las funciones mas destacables se encuentra la que nos permite importar clientes y todo lo relacionado con un cliente desde el directorio actual de trabajo de la firma de abogados. Se comentan a continuación los mas interesantes:
  - **Listar clientes.** Si la petición es realizada por parte de un usuario devolverá el listado de clientes sobre los que dicho usuario tenga asignado algún tipo de permiso. Si la petición es realizada por un administrador devolverá todos los clientes contenidos en la base de datos. Si el usuario no se encuentra activo (porque un administrador lo haya desactivado previamente) no se obtendrán los clientes y se devolverá la vista que indica al usuario que su cuenta se encuentra en estado de inactividad. Por otra parte, si el usuario no se ha autenticado se le redirigirá a la vista de login. De nuevo el listado se realiza ordenado por fecha de modificación descendente, mostrando 15 elementos por página.
  - **Actualizar cliente.** Actualiza el cliente indicado desde la vista de usuario con los datos introducidos por el usuario. Se genera el registro que indica que el cliente ha sido actualizado y quien lo ha actualizado.
  - **Almacenar cliente.** Crea un nuevo cliente en la base de datos con los datos introducidos desde la vista de usuario junto con los permisos asignados a los usuarios que tendrán acceso a la información de dicho cliente. Dado que cada tipo de permiso dispondrá, aparte de los propios, de los privilegios de los permisos que se encuentren en un nivel inferior, se restringe el tipo de permiso a uno por usuario. Con el fin de evitar errores por parte de los usuarios que puedan desembocar en errores de búsqueda y filtrado de la información, se procesa la información introducida y se elimina la posibilidad de que existan usuarios con varios permisos sobre el mismo cliente. Una vez procesada la información se crean el cliente, los permisos, y el registro que indica la acción realizada y qué usuario la ha llevado a cabo. Además, se notifica por medio de correo electrónico a los usuarios indicados previamente de que un nuevo cliente les ha sido asignado. Esta notificación se realizara mediante la clase *ClientNotificationMail()*, que será comentada posteriormente.
  - **Eliminar cliente.** Elimina el cliente solicitado junto con los procedimientos, subcarpetas y archivos que este pudiera tener asociadas. De nuevo se genera el registro correspondiente al borrado de un cliente.
  - **Recuperar cliente.** Recupera el cliente previamente eliminado junto con toda la información que este pudiese haber tenido asociada en el momento de su desactivación y genera el correspondiente registro.
  - **Importar cliente.** Importa los clientes de un usuario y toda la información contenida de estos desde la jerarquía de directorios del directorio de trabajo de la firma de abogados, realizando también la asignación de permisos a los usuarios por parte del administrador. Con este esquema se crean los clientes y los procedimientos, subcarpetas y archivos asociados a estos en la base de datos, y

se almacenan todos los archivos en el directorio de almacenamiento de la aplicación, generándose la ruta de acceso de cada cliente. Se genera de manera simultanea un duplicado de seguridad, reordenamiento y reestructuración de la información contenida en el directorio de trabajo actual, reflejándose el esquema previamente establecido.Se procesa la información separándola por niveles:

- **Primer nivel:** nombre de cliente.
- **Segundo nivel:**
  - Si es directorio: nombre de procedimiento del cliente.
  - Si es archivo: nombre de archivo del cliente.
- **Tercer nivel:**
  - Si es directorio: nombre de subcarpeta de procedimiento del cliente.
  - Si es archivo: nombre de archivo de procedimiento del cliente.
- **Cuarto nivel:**
  - Nombre de subcarpeta de subcarpeta de procedimiento de cliente.

- **fusion.** Como ya se ha comentado, fruto de la desorganización que puede haber existido en cuanto al almacenamiento de información por parte de los profesionales de la firma de abogados, puede darse el caso de que existan clientes duplicados. Esta función permite fusionar dos clientes, seleccionados previamente por el usuario, de manera que toda la información sea asignada a solo uno de ellos, mientras que el otro será eliminado.

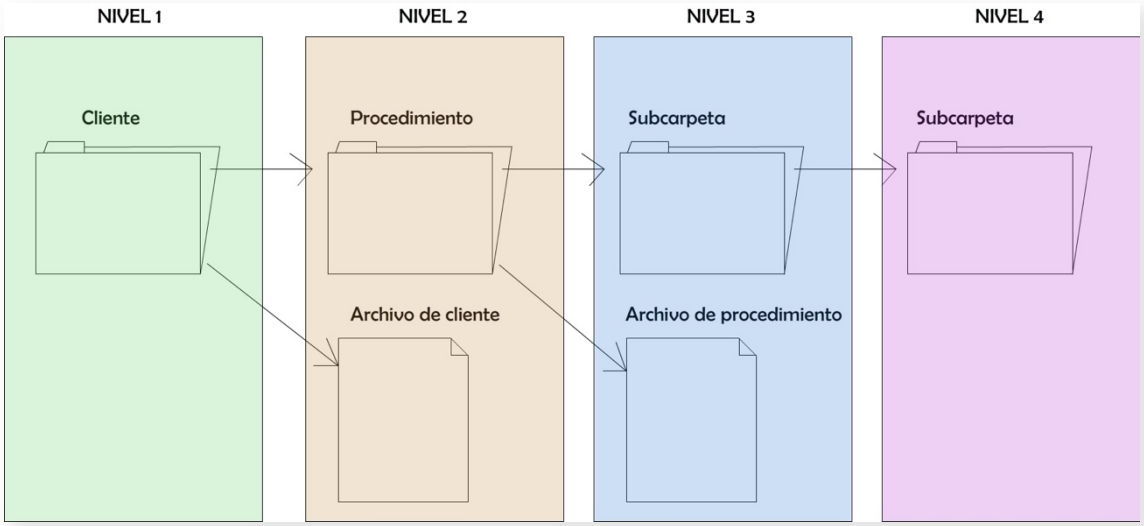


Figura 39. Jerarquía establecida para importación de clientes.

- **Controlador de procedimiento.** Una vez mas el controlador incluirá las funciones que nos permitan crear, obtener, actualizar, eliminar y recuperar un procedimiento, además de aquellas que nos permitan buscar, listar e indexar procedimientos desde la vista final de usuario. También se implementarán las funciones que permitan asignar y actualizar un tipo de permiso sobre un determinado procedimiento a un usuario.
  - **index.** Obtiene todos los procedimientos sobre los que el usuario que ha generado la petición tiene asignado un tipo de permiso. Si la petición la ha generado un administrador devolverá el listado completo de los procedimientos almacenados en la base de datos. De nuevo, si el usuario ha sido desactivado no se le mostrará

ningún listado, mientras que si no se ha autenticado será redirigido a la vista correspondiente para que se identifique como usuario autenticado.

- **permisosProcedimiento.** Retorna los usuarios y tipos de permisos asociados a un cierto procedimiento. Permite al administrador visualizar dicha información desde la vista de administrador.
- **getPermisoUser.** Obtiene el tipo de permiso que el usuario tenga asignado sobre un procedimiento. Este método es empleado para filtrar la información y establecer las posibles acciones a realizar sobre dicho procedimiento desde la vista de usuario.
- **procedimientosClienteUser.** Devuelve el listado de los procedimientos de un cliente sobre los que el usuario que ha generado la petición posee algún tipo de permiso. Podrá realizarse una búsqueda por campo y orden, o búsqueda por indexado.

- **Controlador de subcarpeta.** Dado que los usuarios tendrán asignados sobre las subcarpetas los mismos permisos que tuviesen sobre el procedimiento o subcarpeta que la contenga, únicamente se implementan las funciones que permitan:
  - **Crear una nueva subcarpeta.** Se crea una nueva subcarpeta y, según la petición generada desde la vista de usuario, la asocia a un procedimiento o subcarpeta existente en la base de datos.
  - **Eliminar una subcarpeta.** Se elimina la subcarpeta seleccionada junto con todos los archivos y subcarpetas que esta pudiese contener.

- **Controlador de archivo.** Se han implementado las funciones que permiten:
  - **Descargar un archivo.** Se habilita la descarga del archivo seleccionado desde la vista final de usuario y se añade el registro pertinente al registro de actividad.
  - **Abrir un archivo.** Se busca el archivo seleccionado y se abre en modo visualización. Los tipos de archivos que se han permitido en la aplicación son: .docx, .pdf y archivos de imagen (jpg, jpeg, png,…). Los pdf y las imágenes son abiertas de forma automática por el explorador al devolverlos desde el controlador, mas no ocurre así con los archivos .docx, que deben ser tratados previamente a retornarlos para que sean visualizados por el usuario. Para ello emplearemos la función loadXML() de php, que carga un documento XML desde una cadena (string). Con el fin de que el documento XMl generado se pueda visualizar, deberemos eliminar las etiquetas “w” (Word) que han sido introducidas en la creación del archivo. Finalmente se retorna el archivo para su visualización y se almacena el registro que indica qué archivo ha sido visualizado y por quien.
  - **Actualizar un archivo.** Se actualiza el archivo indicado. Dicha actualización se realiza en dos pasos: actualización de la información contenida del archivo en la base de datos y actualización del archivo contenido en el almacenamiento interno de la aplicación.
  - **Notificar.** Se notifica a todos los usuarios asociados a un determinado cliente que cierta acción ha sido realizada sobre un archivo. Dado que el archivo puede pertenecer directamente al cliente, a un procedimiento del cliente, o a una subcarpeta del cliente, se obtiene dicha información y se adjunta a la notificación que el usuario recibirá por correo electrónico.
  - **Subir un archivo.** Se sube el archivo seleccionado desde la vista de usuario y se almacena tanto en la base de datos (información asociada a el archivo) como en el almacenamiento interno de la aplicación. Se establecen ciertas condiciones que nos permitan saber, previamente al almacenado, si el archivo corresponde

directamente a un cliente, a un procedimiento o a una subcarpeta, a fin de almacenarlo de manera coherente.

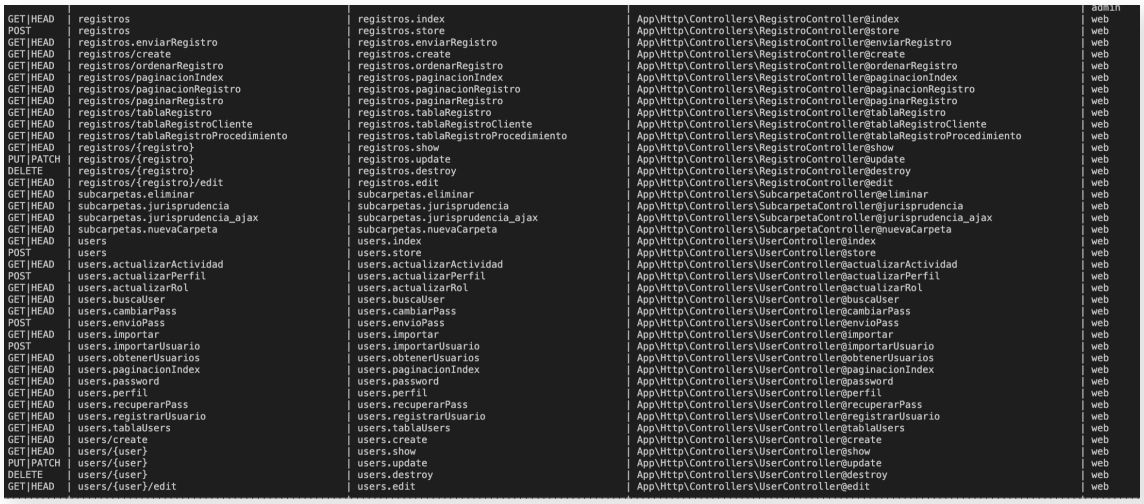
- **Controlador de registro.** Como se ha comentado, los registros almacenados en la tabla del registro de actividad se generan desde los distintos controladores involucrados en el manejo de información referida a clientes y procedimientos. Al tratarse del registro de actividad, no tendría sentido que se pudiesen eliminar o modificar dichos registros, así que las funciones implementadas en este caso se ciñen al listado, filtrado y envío del registro de actividad.
    - **index.** Obtiene el listado de registros contenidos en el registro de actividad asociados a clientes sobre los que el usuario solicitante tenga asignado algún tipo de permiso. Si el solicitante es el administrador devuelve la lista completa de registros.
    - **ordenarRegistro.** Devuelve el listado del registro de actividad de manera ordenada según el orden y campo indicados desde la vista de usuario.
    - **enviarRegistro.** Envía un correo electrónico con el registro de actividad a los usuarios administradores. Esta función, a parte de ejectuarse de manera automatica periódicamente, podrá ser ejecutada por parte del administrador desde la vista de usuario (administrador). El envío se realiza mediante la clase *LogMail(),* que posteriormente comentaremos.

- **Envío de notificaciones por correo electrónico.** Una vez vistas las funciones de mas interés de los principales controladores, proseguimos comentando las clases involucradas en las distintas funciones de notificación mediante correo electrónico comentadas.
    - **ClientNotificationMail.** Esta función recibe como parámetros el cliente y el tipo de permiso y los retorna como propios a la vista en la que se ha establecido la estructura del email a enviar.
    - **LogMail.** Recibe el listado de registros y lo retorna a la vista que contiene la estructura final del email que recibirá el administrador.

- **Archivo de rutas.** Los controladores y los modelos están implementados, así que ya disponemos de la lógica que determinará qué información y de qué manera va a ser tratada en cada momento. Pero los controladores, y las funciones declaradas en ellos, no serán accesibles si no les asignamos una ruta que Laravel pueda interpretar. Para introducir dichas rutas deberemos dirigirnos al archivo *routes/web.php.* Todos los archivos que se encuentran en el directorio routes son cargados automáticamente por la aplicación. Definiremos las rutas atendiendo al verbo HTTP correspondiente (según si la petición sea de tipo GET o tipo POST), e indicando URI, controlador y función o vista asociada, y nombre de la ruta establecida. Como ejemplo procedemos a comentar la siguiente ruta:

```
Route::get('registros.enviarRegistro',
'RegistroController@enviarRegistro')->name('registros.enviarRegistro');
```

En la que se indica que la petición es de tipo GET, con identificador 'registros.enviarRegistro', que tiene asociada la función *enviarRegistro()* del controlador de registro (RegistroController), y a la cual se le ha asignado el nombre "registros.enviarRegistro".



GET|HEAD	registros	registros.index	App\Http\Controllers\RegistroController@index	web
POST	registros	registros.store	App\Http\Controllers\RegistroController@store	web
GET|HEAD	registros.enviarRegistro	registros.enviarRegistro	App\Http\Controllers\RegistroController@enviarRegistro	web
GET|HEAD	registros/create	registros.create	App\Http\Controllers\RegistroController@create	web
GET|HEAD	registros/ordenarRegistro	registros.ordenarRegistro	App\Http\Controllers\RegistroController@ordenarRegistro	web
GET|HEAD	registros/paginacionIndex	registros.paginacionIndex	App\Http\Controllers\RegistroController@paginacionIndex	web
GET|HEAD	registros/paginacionRegistro	registros.paginacionRegistro	App\Http\Controllers\RegistroController@paginacionRegistro	web
GET|HEAD	registros/paginarRegistro	registros.paginarRegistro	App\Http\Controllers\RegistroController@paginarRegistro	web
GET|HEAD	registros/tablaRegistro	registros.tablaRegistro	App\Http\Controllers\RegistroController@tablaRegistro	web
GET|HEAD	registros/tablaRegistroCliente	registros.tablaRegistroCliente	App\Http\Controllers\RegistroController@tablaRegistroCliente	web
GET|HEAD	registros/tablaRegistroProcedimiento	registros.tablaRegistroProcedimiento	App\Http\Controllers\RegistroController@tablaRegistroProcedimiento	web
GET|HEAD	registros/{registro}	registros.show	App\Http\Controllers\RegistroController@show	web
PUT|PATCH	registros/{registro}	registros.update	App\Http\Controllers\RegistroController@update	web
DELETE	registros/{registro}	registros.destroy	App\Http\Controllers\RegistroController@destroy	web
GET|HEAD	registros/{registro}/edit	registros.edit	App\Http\Controllers\RegistroController@edit	web
GET|HEAD	subcarpetas.eliminar	subcarpetas.eliminar	App\Http\Controllers\SubcarpetaController@eliminar	web
GET|HEAD	subcarpetas.jurisprudencia	subcarpetas.jurisprudencia	App\Http\Controllers\SubcarpetaController@jurisprudencia	web
GET|HEAD	subcarpetas.jurisprudencia_ajax	subcarpetas.jurisprudencia_ajax	App\Http\Controllers\SubcarpetaController@jurisprudencia_ajax	web
GET|HEAD	subcarpetas.nuevaCarpeta	subcarpetas.nuevaCarpeta	App\Http\Controllers\SubcarpetaController@nuevaCarpeta	web
GET|HEAD	users	users.index	App\Http\Controllers\UserController@index	web
POST	users	users.store	App\Http\Controllers\UserController@store	web
GET|HEAD	users.actualizarActividad	users.actualizarActividad	App\Http\Controllers\UserController@actualizarActividad	web
POST	users.actualizarPerfil	users.actualizarPerfil	App\Http\Controllers\UserController@actualizarPerfil	web
GET|HEAD	users.actualizarRol	users.actualizarRol	App\Http\Controllers\UserController@actualizarRol	web
GET|HEAD	users.buscaUser	users.buscaUser	App\Http\Controllers\UserController@buscaUser	web
GET|HEAD	users.cambiarPass	users.cambiarPass	App\Http\Controllers\UserController@cambiarPass	web
POST	users.envioPass	users.envioPass	App\Http\Controllers\UserController@envioPass	web
GET|HEAD	users.importar	users.importar	App\Http\Controllers\UserController@importar	web
POST	users.importarUsuario	users.importarUsuario	App\Http\Controllers\UserController@importarUsuario	web
GET|HEAD	users.obtenerUsuarios	users.obtenerUsuarios	App\Http\Controllers\UserController@obtenerUsuarios	web
GET|HEAD	users.paginacionIndex	users.paginacionIndex	App\Http\Controllers\UserController@paginacionIndex	web
GET|HEAD	users.password	users.password	App\Http\Controllers\UserController@password	web
GET|HEAD	users.perfil	users.perfil	App\Http\Controllers\UserController@perfil	web
GET|HEAD	users.recuperarPass	users.recuperarPass	App\Http\Controllers\UserController@recuperarPass	web
GET|HEAD	users.registrarUsuario	users.registrarUsuario	App\Http\Controllers\UserController@registrarUsuario	web
GET|HEAD	users.tablaUsers	users.tablaUsers	App\Http\Controllers\UserController@tablaUsers	web
GET|HEAD	users/create	users.create	App\Http\Controllers\UserController@create	web
GET|HEAD	users/{user}	users.show	App\Http\Controllers\UserController@show	web
PUT|PATCH	users/{user}	users.update	App\Http\Controllers\UserController@update	web
DELETE	users/{user}	users.destroy	App\Http\Controllers\UserController@destroy	web
GET|HEAD	users/{user}/edit	users.edit	App\Http\Controllers\UserController@edit	web

Figura 40. Lista de rutas generadas

4.3.3 Tarea IV. Implementación de vistas

Todo el trabajo del backend, salvo todo aquello que pueda surgir fruto de errores de calculo o esquematización, está realizado, así que es hora de pasar a comentar el trabajo realizado en el frontend de la aplicación, o sea en aquello que el usuario podrá visualizar. Basándonos en los esquemas y diseños previamente realizados podemos proceder a crear todo lo relacionado al aspecto visual de nuestra aplicación, que habíamos diferenciado entre vistas de usuario y administrador.
Laravel nos facilita de nuevo la tarea de implementación de las vistas de nuestra aplicación mediante Blade, un motor de plantillas simple pero poderoso. A diferencia de algunos motores de plantillas PHP, Blade nos permite usar código PHP simple en nuestras vistas, de hecho todas las plantillas Blade son compiladas en código PHP simple y se almacenan en el caché hasta que se modifican. Las plantillas Blade son, básicamente, hojas HTML que permiten el uso de variables php, junto con la sección de las hojas de estilos (CSS) y la sección de scripts (Javascript o JQuery) clásicas. Los archivos de plantilla Blade usan la extensión .blade.php y se almacenan generalmente en *resources/views*.

Las vistas de usuario han sido implementadas de manera que sean plenamente responsivas, esto quiere decir que responderán de manera adecuada al amplio abanico de resoluciones que presentan hoy día los equipos/dispositivos electrónicos.

Se respetan los colores fijados en la fase de planteamiento, texto blanco sobre fondo azul, que ofrece una visibilidad suficientemente nítida, pero se decide cambiar la barra de navegación superior diseñada por una barra lateral que otorgue un aire mas fresco y dinámico a la aplicación.

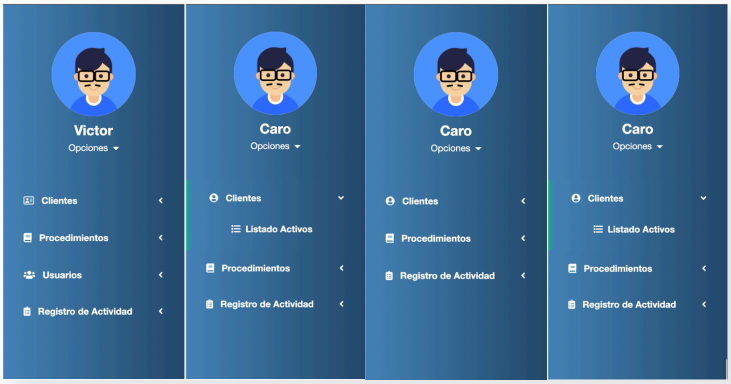


Figura 41. Barras de navegación de usuario y administrador.

Tras la barra de navegación implementamos las distintas vistas pertenecientes a cada uno de los modelos basándonos en la estructura de tablas establecida en la fase de planteamiento. Inicialmente se mostrará una tabla con la información solicitada, ya sean clientes, procedimientos o usuarios, junto a ella, y según la opción u opciones seleccionadas, se mostrarán hasta dos tablas mas (una encima de la otra) que mostrarán otra información. Con el propósito de que dichas tablas contengan cierta información que nos permitan visualizar el funcionamiento del diseño, Laravel nos ofrece la posibilidad de sembrar de datos nuestra base de datos de manera rápida y fácil. Esto se consigue mediante el uso de las clases Seeder y Factory. A su vez, y mediante la librería Faker de PHP, podemos generar ágilmente datos ficticios que serán los que utilizará la clase Factory, llamada desde la clase Seeder.

Una vez sembrada nuestra base de datos, podremos visualizar las vistas y tablas generadas y continuar con ciertos aspectos de su diseño. A continuación, se comentan las diferentes vistas implementadas (únicamente se comentan las vistas de administrador, ya que el administrador tiene acceso a todas las vistas de la aplicación y en las que comparte con el usuario dispone de mas información):

- ***Vistas de cliente.*** Son aquellas desde las que el administrador y el usuario pueden visualizar y realizar acciones sobre los clientes.
    - **Nuevo cliente.** Consta de un formulario para introducción de los campos requeridos del cliente junto con los selectores de usuarios para cada tipo de permiso asignable sobre el cliente.
    - **Listado/Inactivos.** Ambas muestran un listado de los clientes disponibles, salvo que, como es obvio, la segunda vista muestra únicamente aquellos que se encuentren desactivados. Al acceder a la vista se mostrará la tabla con los clientes y las acciones realizables sobre estos. Si se seleccionan las opciones de archivos o permisos, se abrirán nuevas tablas a la derecha mostrando dicha información, mientras que si se hace click sobre el cliente se obtendrán todos los procedimientos de que este disponga y se listarán en la tabla de la izquierda, permitiendo al usuario navegar entre tabla de clientes y procedimientos, asi como entre las de archivos y permisos respectivas. Si se selecciona la opción ver, se abrirá un modal con un formulario que muestra de manera predeterminada la información actual del cliente o procedimiento y, si el usuario dispone de los permisos necesarios, se le permitirá modificar dicha información, actualizándose automáticamente el listado. En la parte superior de todas las tablas disponemos de un buscador y todas ellas nos permiten ordenar la información de manera ascendente o descendente a partir de cada uno de sus campos, esto es así en todas las vistas. Todas las peticiones relacionadas con la obtención y visualización de

la información en las distintas vistas se han realizado utilizando la función AJAX de JQuery, que nos permite intercambiar datos con el servidor y actualizar partes de nuestra página sin recargarla. Para ello ya se han implementado previamente los métodos, de forma que, al realizarse cierto tipo de solicitud, el controlador devuelva la información solicitada a cierta vista (que en nuestro caso contienen las distintas tablas) que será cargada e introducida como html en el código de la vista actual. Como ejemplo, podemos observar el siguiente script, en el que, mediante AJAX, se realiza una consulta a una función del controlador de procedimientos, a la que se le adjunta una id de cliente. Dicha función devuelve la paginación de los procedimientos actuales y la carga en el contenedor html "#pagProcs".

```
function paginarProcedimientos(id) {
    $.ajax({
        delay: 250,
        method: "GET",
        url: "{{route('procedimientos.paginacionProcs')}}",
        data: {
            id: id
        },
        success: function(data) {
            $("#pagProcs").html(data);
        }
    });
}
```

Figura 42. Función Ajax.

- **Fusionar clientes.** Presenta un formulario con dos selectores desde los cuales seleccionar los dos clientes que se pretende fusionar.
- **Importar clientes.** Formulario con un selector de clientes a importar. Una vez los clientes sean importados se mostrará el correspondiente aviso al administrador. Para poder importar clientes deben haberse importado previamente usuarios.

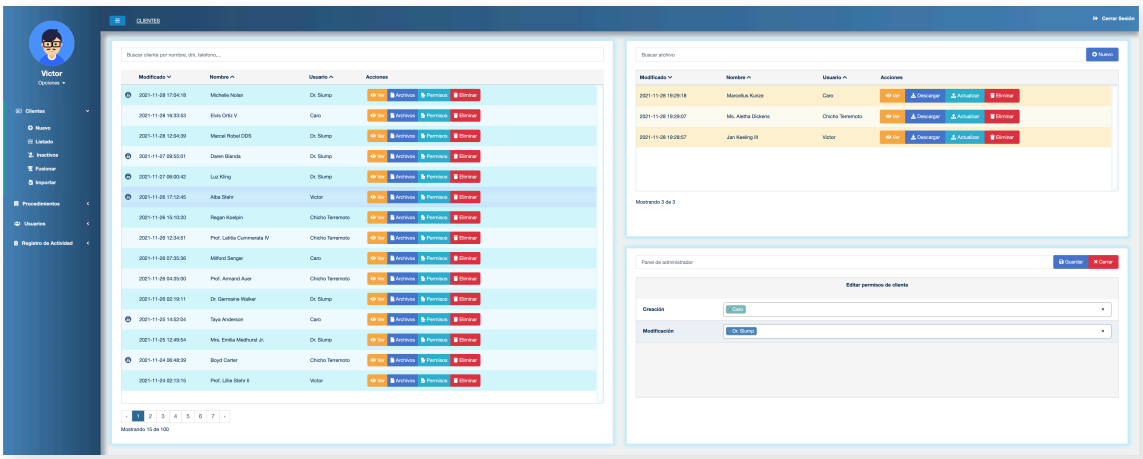


Figura 43. Vista de menú clientes.

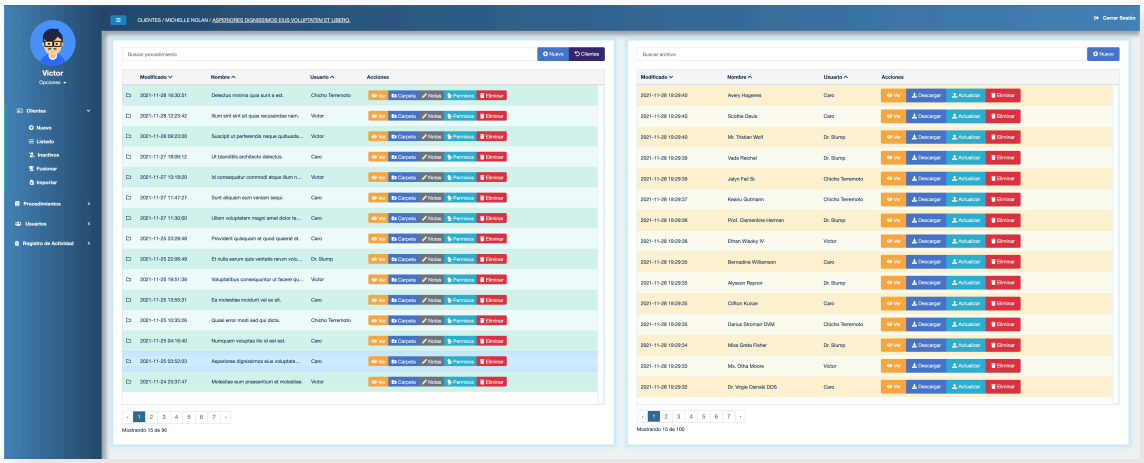


Figura 44. Vista de procedimientos de cliente.

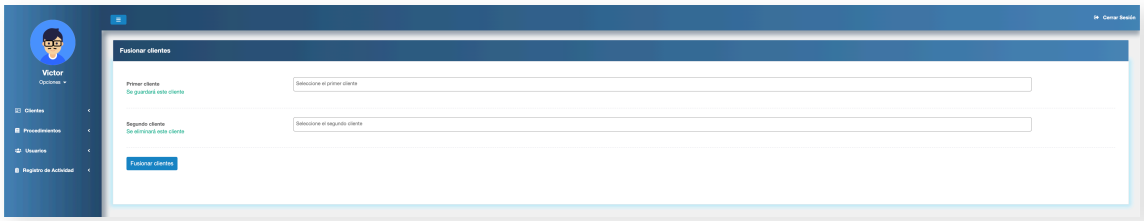


Figura 45. Vista de fusión de clientes.

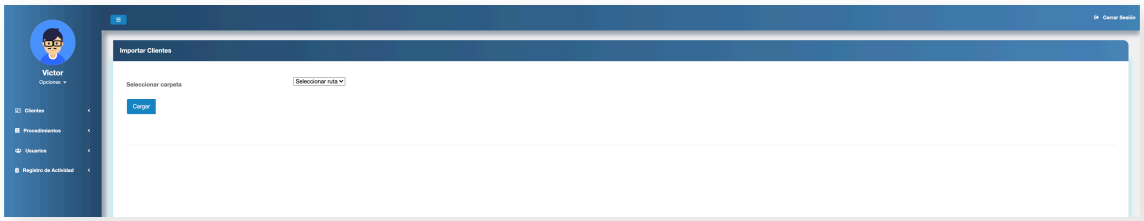


Figura 46. Vista de importado de clientes.

- **Vistas de procedimiento.** Son aquellas otras desde las que el usuario y el administrador pueden visualizar además de realizar acciones sobre los procedimientos.
    - **Nuevo procedimiento.** Muestra un formulario mediante el cual introducir la información requerida del procedimiento a la hora de su creación. Dispone de un selector de cliente al que será asociado el procedimiento y dos selectores de permisos.
    - **Listado.** Su funcionamiento es idéntico al del listado de clientes, salvo que en este caso el usuario accederá a la totalidad de los procedimientos sobre los que tenga asignado algún tipo de permiso, y no a los de determinado cliente. Desde la vista de listado de procedimientos el usuario podrá realizar las mismas acciones sobre el procedimiento que desde la vista de clientes.

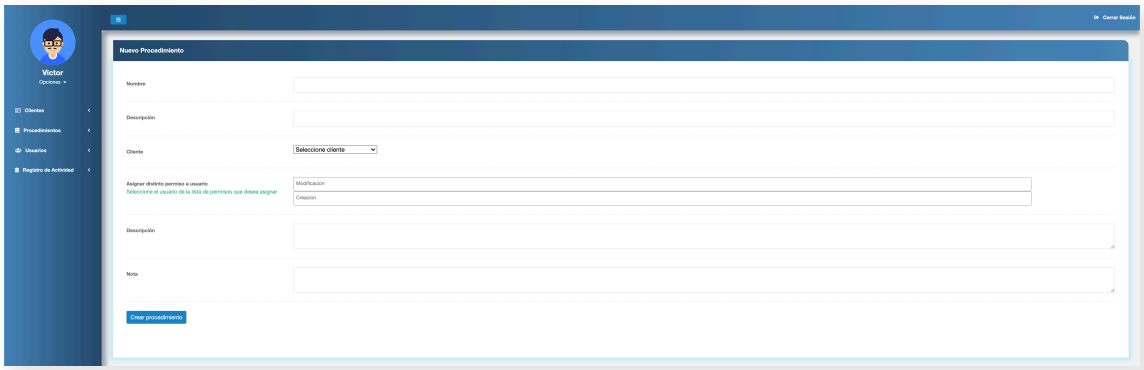


Figura 47. Vista de nuevo procedimiento.

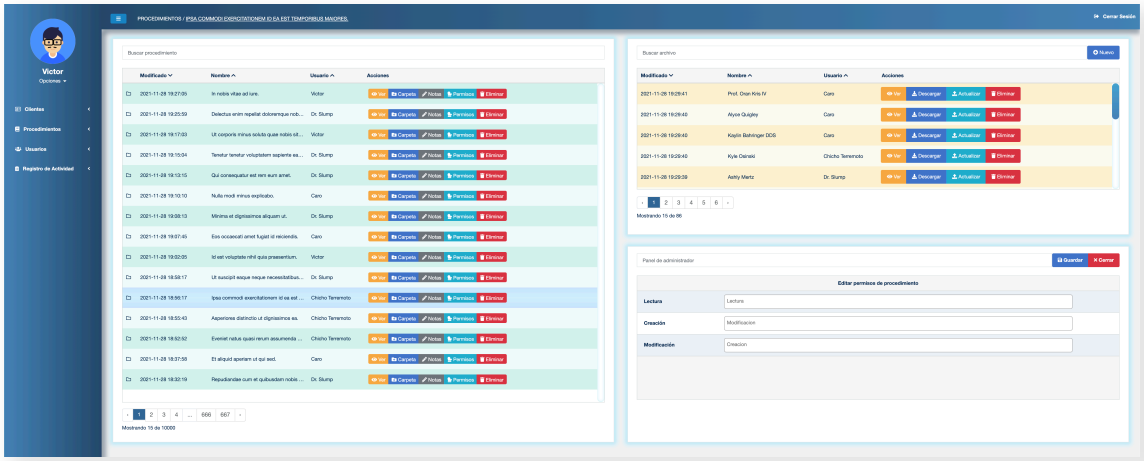


Figura 48. Vista de procedimientos de un usuario

- **Vistas de usuarios.** Permiten únicamente al administrador visualizar y realizar acciones sobre los usuarios.
    - **Nuevo usuario.** Presenta un formulario para la creación de un nuevo usuario y requiere, entre otros campos, de un tipo de rol asignado a dicho usuario de entre los ya sabidos (usuario o administrador).
    - **Listado de usuarios.** Muestra un listado de los usuarios y la información asociada a estos, además del tipo de rol del usuario y el estado de actividad. El administrador podrá desactivar y asignar un nuevo rol a un usuario.
    - **Importar usuarios.** Formulario que nos permite la introducción de un directorio contenido en el equipo desde el cual importar los usuarios. Una vez el directorio es seleccionado, se mostrará un listado con los distintos directorios contenidos por este, los que sean seleccionados serán importados y añadidos al selector que previamente veíamos en la vista de importación de clientes, que permitirá seleccionar el usuario del que se pretende importar la información (clientes y procedimientos y archivos asociados).

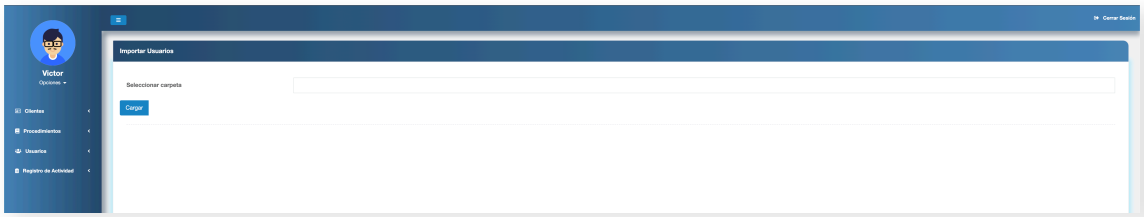


Figura 49. Vista de importado de usuarios.

- **Vista de registro.** Muestra el registro de actividad hasta la fecha, completo para el administrador y para el usuario tan solo los registros en los que él esté relacionado de alguna manera. Con el fin de facilitar la búsqueda de cierto registro, además del correspondiente ordenamiento ofrecido por la tabla por campos, se ha decidido que al pasar el ratón por encima de cada registro, este se resaltará en un determinado color, que estará asociado a un tipo de acción realizada. Por ejemplo, el color azul significa que ha habido una actualización o creación, el amarillo que ha habido una visualización, el celeste que se han actualizado permisos, el rojo que se ha eliminado un elemento, etc,…).

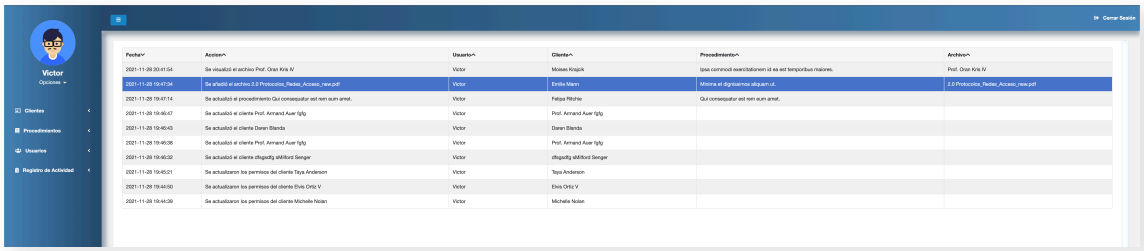


Figura 50. Vista de registro de actividad.

4.3.4 Tarea X. Implementación de tareas

Consiste en la configuración de las tareas previamente establecidas en el servidor que contendrá la aplicación. Dicha configuración se realiza desde el panel de programación de tareas del sistema operativo. Para ello se establece un periodo y se asocia un archivo .bat, previamente generado, que contenga las instrucciones que ejecuten el comando especificado.

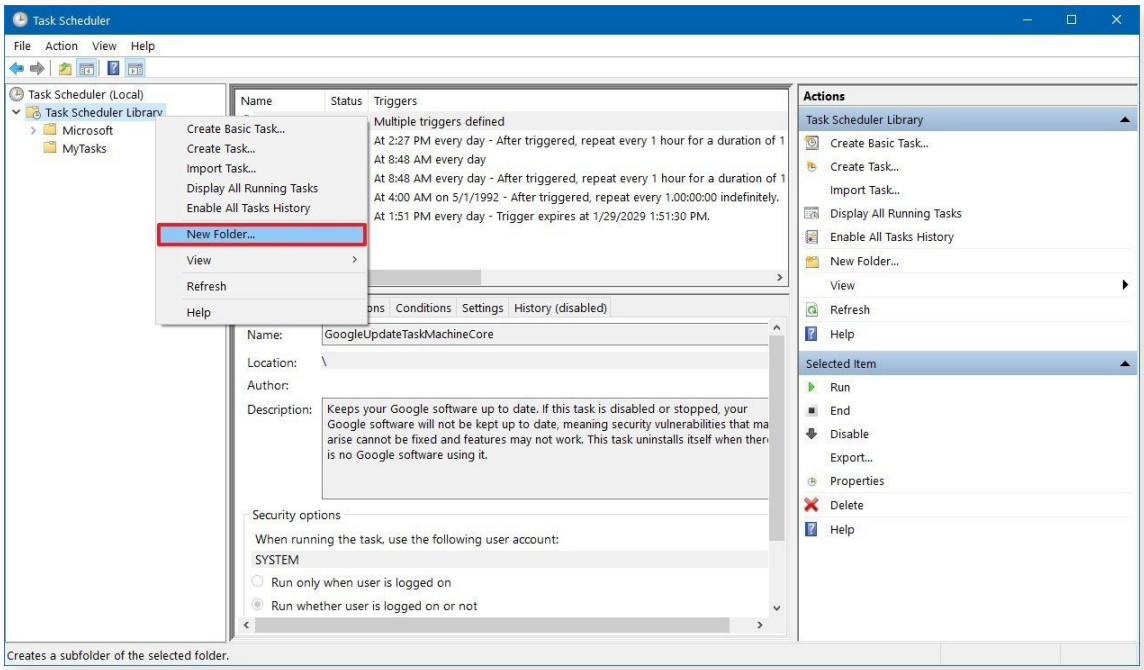


Figura 51. Programación de tareas en Windows Server.

4.3.5 Tarea XI. Fase de testeo

Dado que la tarea de implementación de funciones y la tarea de implementación de vistas han sido llevadas a cabo de forma prácticamente simultanea, aunque si que es cierto que se necesitan ciertas implementaciones iniciales por parte de las funciones para poder implementar las vistas, estamos seguros de que lo implementado funciona y que, en la medida en que lo hemos probado, funciona correctamente. Esto no implica que no puedan aparecer errores o bugs, ya que hasta las mas grandes aplicaciones son sometidas a multitud de actualizaciones y parcheados una vez son lanzadas al mercado, pues se torna una tarea mas bien complicada la tarea de realizar un testeado al 100%.

Con la aplicación lista y plenamente funcional se programa una nueva reunión con los profesionales y administradores de la firma de abogados en la que se realizará una presentación de las funcionalidades de la aplicación. Con el fin de poder mostrar la aplicación con la información ya importada desde el directorio de trabajo de la firma de abogados, procederemos a migrar el proyecto a su servidor y realizar desde ahí la importación de usuarios. Para ello emplearemos la herramienta de escritorio compartido Anydesk y el universalmente conocido GitHub, que nos permite alojar proyectos utilizando el sistema de control de versiones Git visto en clase, y mediante el cual instalaremos la aplicación.

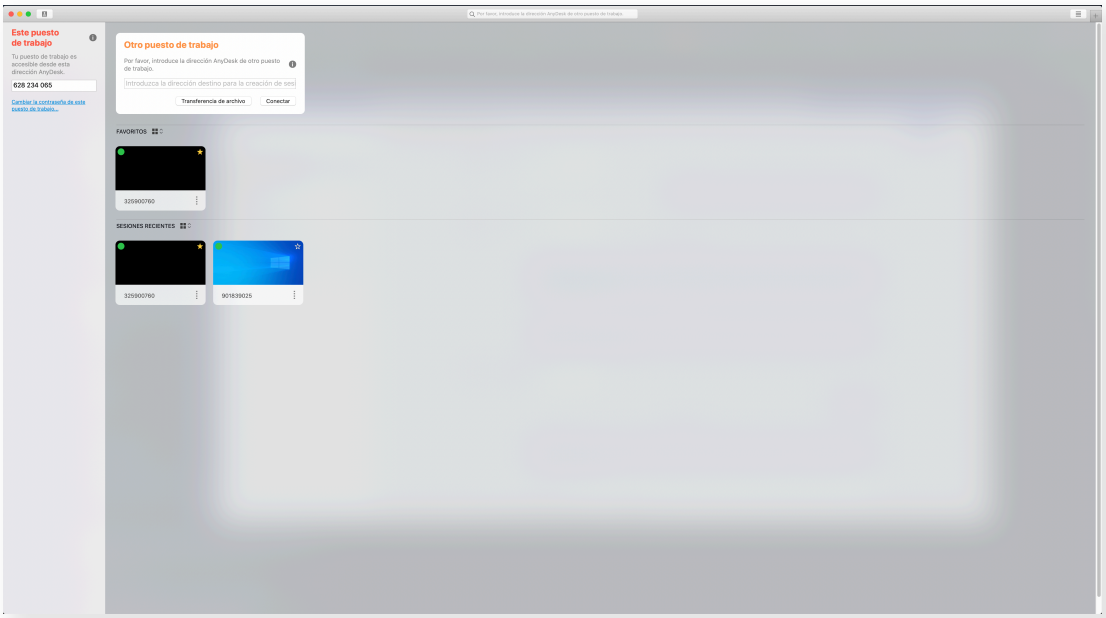


Figura 52. Anydesk.

Tras la reunión disponemos de ciertas mejoras o añadidos sugeridos por los profesionales de la firma de abogados, tales como:

- Un procedimiento deberá contener, de manera predefinida, una estructura de subcarpetas que estará relacionada con las distintas fases por las que un procedimiento puede pasar, a saber:
    - Instrucción
    - Juicio oral
    - Ejecutoria
    - Recurso T.S.J
    - Recurso Casación
    - Recurso Amparo
- Un procedimiento contendrá, también de manera predeterminada, una subcarpeta “Gestión” que estará únicamente disponible para administradores.
- Al generarse un nuevo cliente o procedimiento, todos los usuarios dispondrán de permisos de lectura sobre este. Los permisos de creación y modificación deberán seguir siendo asignados por parte de un administrador.
- Un cliente podrá ser activado o desactivado. Si es desactivado deberá poder introducirse la razón.
- Deberá indicarse si un cliente se encuentra en un centro penitenciario, junto con la fecha de entrada al mismo.
- Deberá añadirse una nueva opción a la barra de navegación llamado “Jurisprudencia”. Dicha opción será únicamente mostrada a los administradores y contendrá un sistema de subcarpetas como el previamente implementado.

4.3.6 Tarea XII. Fase de depuración

Se planifica la fase de depuración con los añadidos propuestos:

- Fases de procedimiento
- Subcarpeta Gestión
- Permiso de lectura sobre todos los clientes.

- Activación/desactivación de un cliente
- Centro penitenciario
- Jurisprudencia

Base de datos	Back-end	Front-end
Fases de procedimiento		
Subcarpeta Gestión		
	Lectura sobre clientes	
	Activación/Desactivación cliente	
Centro Penitenciario		
Jurisprudencia		

Tabla 7. Esquematización de los añadidos.

Añadidos	Duración
Fases de procedimiento	8 horas
Subcarpeta Gestion	2 horas
Lectura sobre clientes	1 horas
Activación/desactivación cliente	2 horas
Centro penitenciario	2 horas
Jurisprudencia	16 horas

Tabla 8. Planificación de los añadidos.

Se realizan y testean los reajustes y se procede a programar una nueva reunión con la firma de abogados en la que mostrar las mejoras implementadas.

4.4 Resultados

La aplicación final ofrece todo aquello que en un principio se consideró como imprescindible, mas el añadido de los ajustes realizados tras las propuestas de los profesionales. El resultado es una aplicación a medida para una firma de abogados que cumple con las funciones necesarias para facilitar las labores realizadas, clasificar y administrar la información de una manera óptima y agruparla en un único sistema global administrable, protegido y restringible.

Por otra parte, aunque la aplicación se desarrolló expresamente para una empresa, dado que cubre las necesidades organizacionales y laborales de un determinado ámbito profesional, puede ser ofertada a otras muchas mas empresas de dicho ámbito.

A continuación, como muestra de resultados, se presentan dos ejemplos de interacción por parte de los dos tipos de usuarios. Las capturas se han realizado bajo resoluciones típicas de dispositivos (tablet, móvil y distintos tamaños de monitor) a modo de muestra de la responsividad de la aplicación.

- **Interaccion de administrador.**
  - Autenticación.
  - Creación de clientes.
  - Creación de procedimientos para estos clientes.
  - Modificación de permisos.
  - Listado de usuarios y modificación de rol de usuario.
- **Interacción de usuario**
  - Autenticación.
  - Listado de clientes y procedimientos, antes y después de los cambios de permisos y de rol de usuario.

- Envío registro.

Tras autenticarse, el administrador se dirige al área de clientes, al apartado de nuevo cliente e introduce la información necesaria en el formulario que se le presenta. Para el primer cliente, llamémoslo "Chicho Terremoto", no se asigna ningún tipo de permiso, lo que quiere decir que todos los usuarios únicamente podrán realizar acciones de visualización.

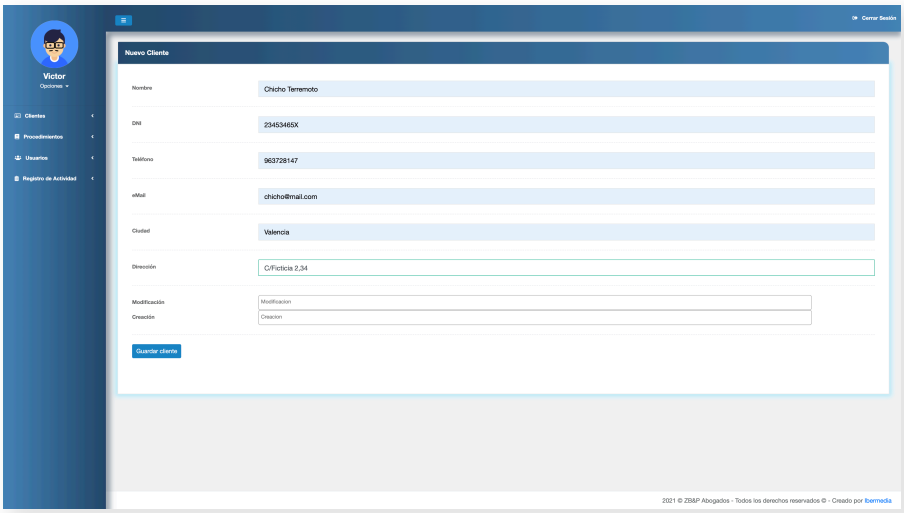


Figura 53. Creación de cliente Chicho Terremoto.

Para el segundo cliente, llamémoslo "Musculman", se asignan permisos de creación a la usuaria Caro.

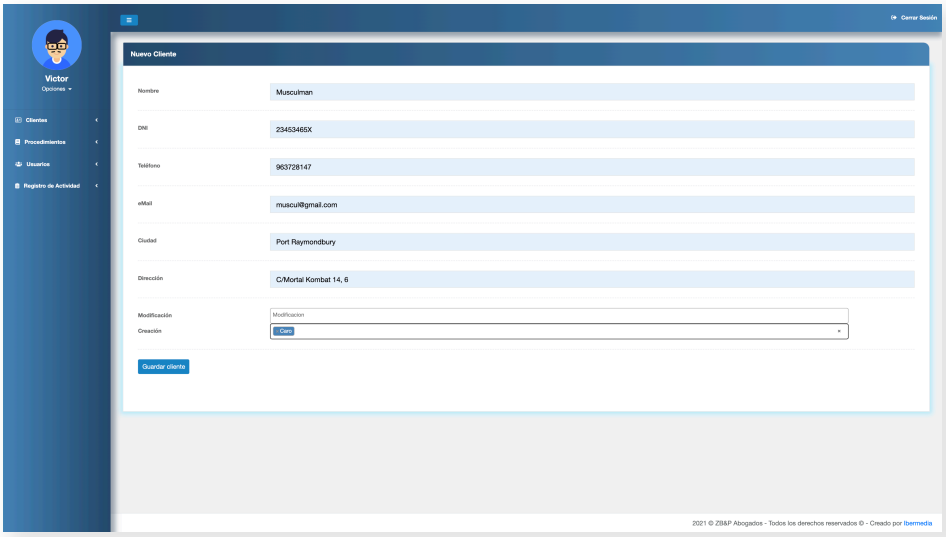


Figura 54. Creación de cliente Musculman.

Al crear clientes, la aplicación enviará una notificación por correo electrónico a los administradores y usuarios asignados a dicho cliente.

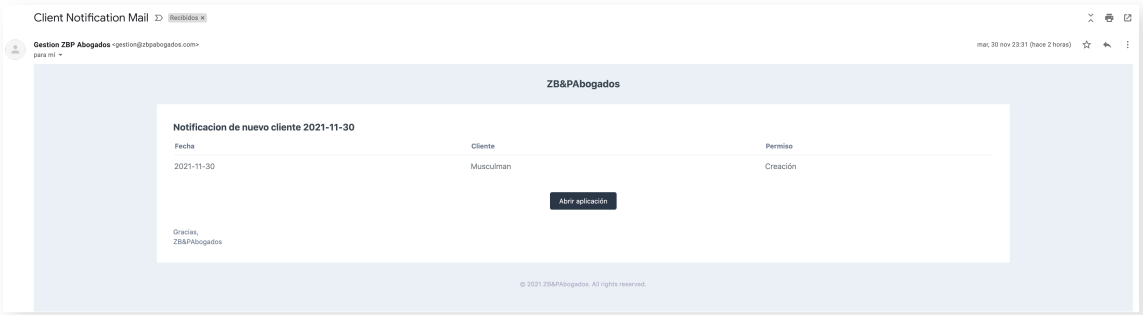


Figura 55. Notificación recibida por correo electrónico.

Como se observa, para “Chicho Terremoto”, Caro no tiene asignado ningún tipo de permiso.

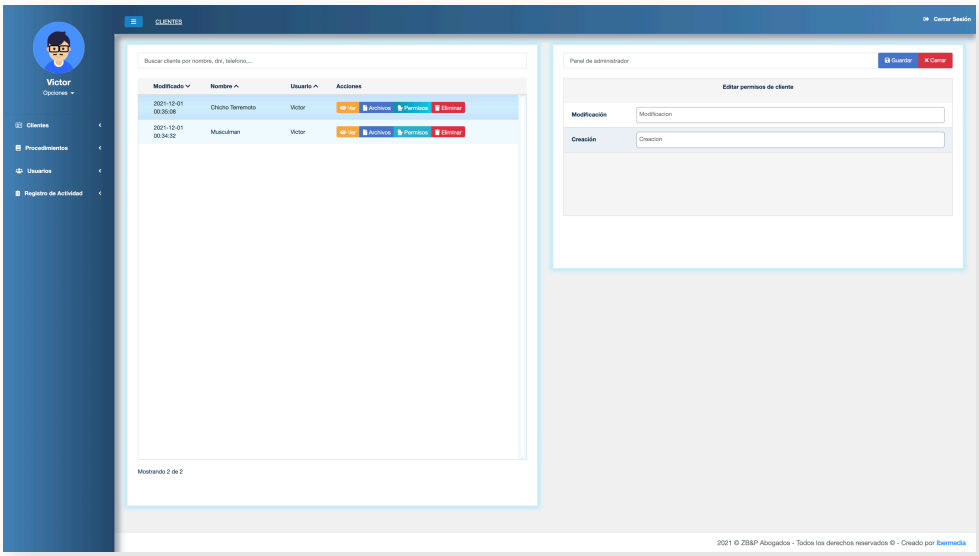


Figura 56. Permisos sobre el cliente Chicho Terremoto.

Mientras que para “Musculman” tiene asignado el permiso que se le ha asignado.

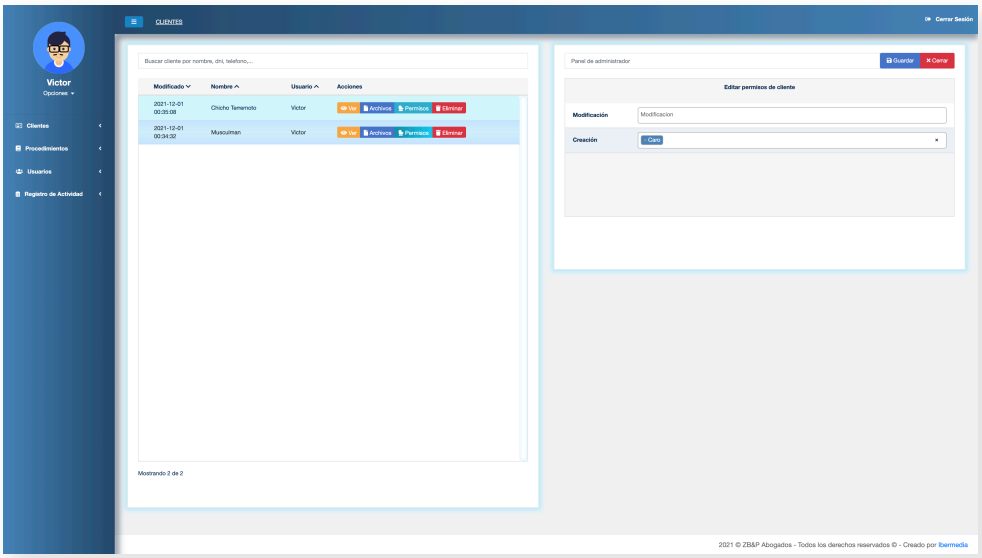


Figura 57. Permisos sobre el cliente Musculman.

El administrador crea un procedimiento, “proc 1”, asociado a Musculman en el que se le asignan permisos de modificación a Caro.

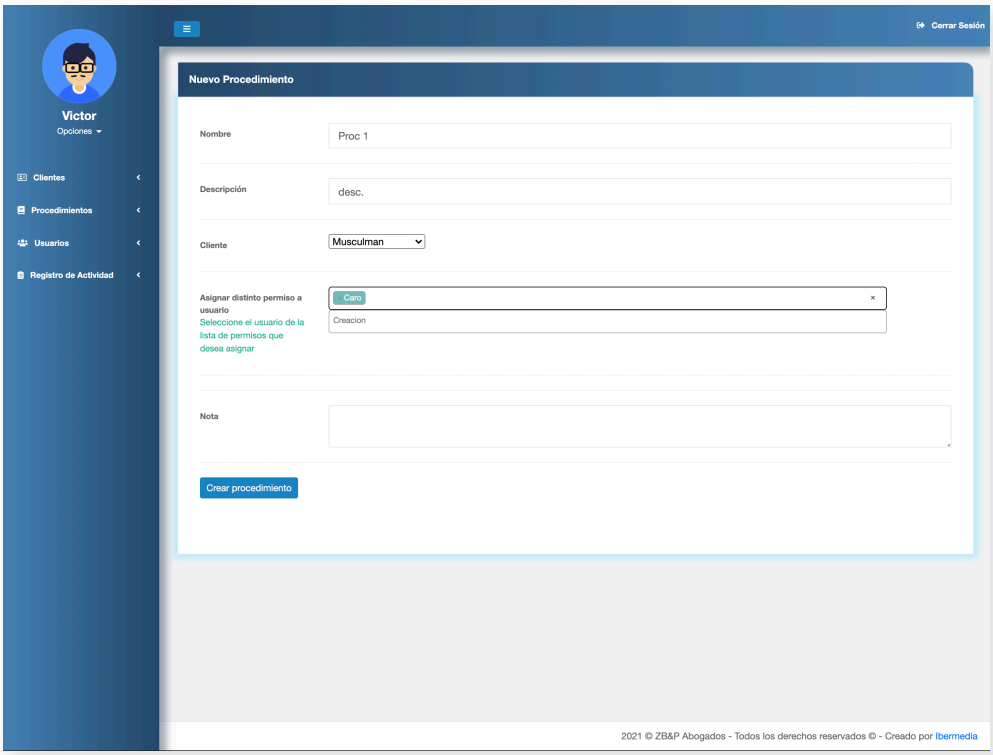


Figura 58. Creación de un nuevo procedimiento.

Crea un segundo procedimiento, “proc 2”, asociado a Musculman, pero no asigna permisos a ningún usuario.

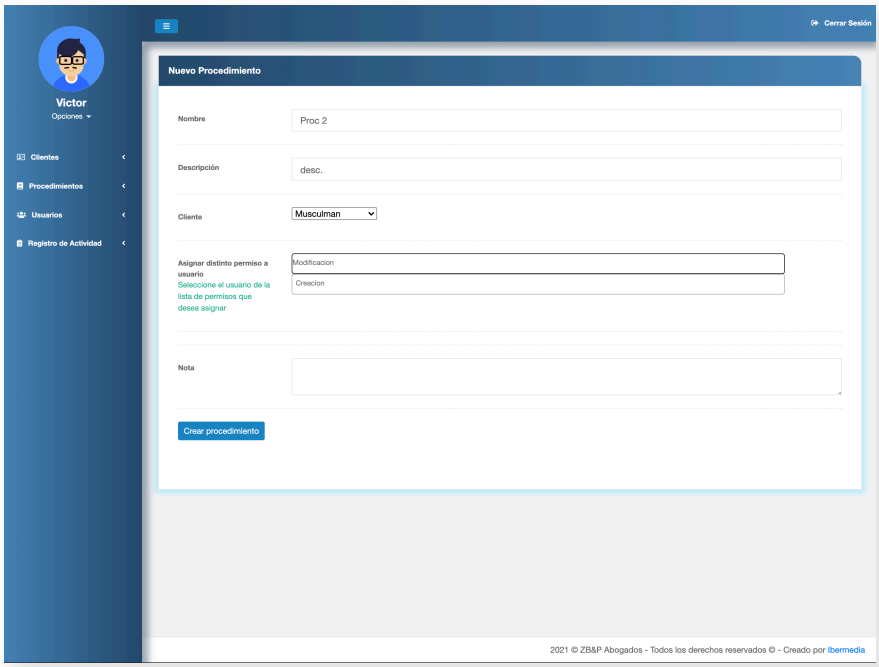


Figura 59. Creación de procedimiento "proc 2" de Musculman.

Se comprueba que, para el procedimiento “proc 1”, Caro tiene asignado el permiso de modificación.

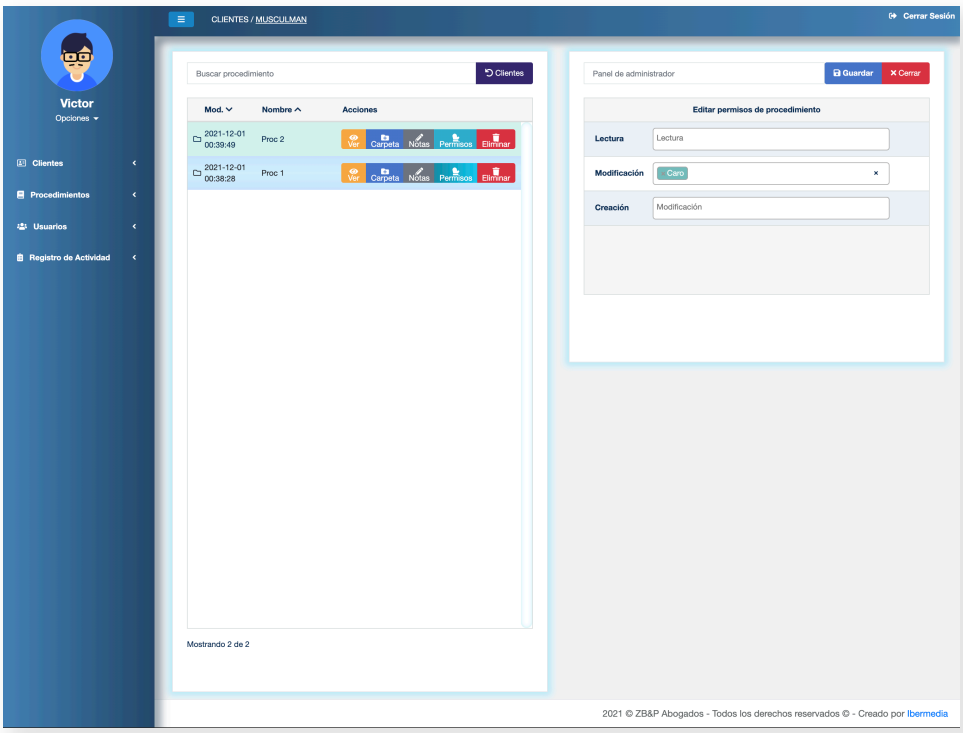


Figura 60. Permisos sobre el procedimiento "proc1".

Ahora la usuario Caro inicia sesión y se dirige al área de clientes. Como se observa puede visualizar los dos clientes, ya que por defecto se asigna permiso de lectura sobre todos los clientes y, además, sobre el cliente Musculman se le ha asignado el permiso de creación.

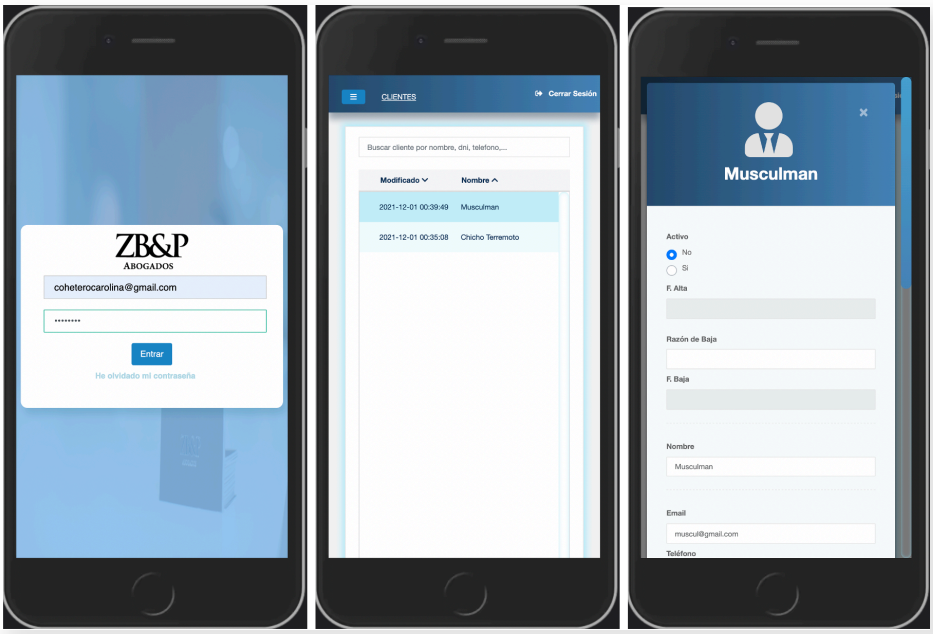


Figura 61. Login, vista de clientes y edición de cliente.

Al acceder a los procedimientos del cliente Musculman, Caro únicamente puede acceder a aquel sobre el que se le han asignado permisos, en este caso "proc 1", que podrá modificar. Además, dado que Caro posee permisos de creación sobre Musculman, podrá crear nuevos procedimientos y subir archivos asociados a este cliente.

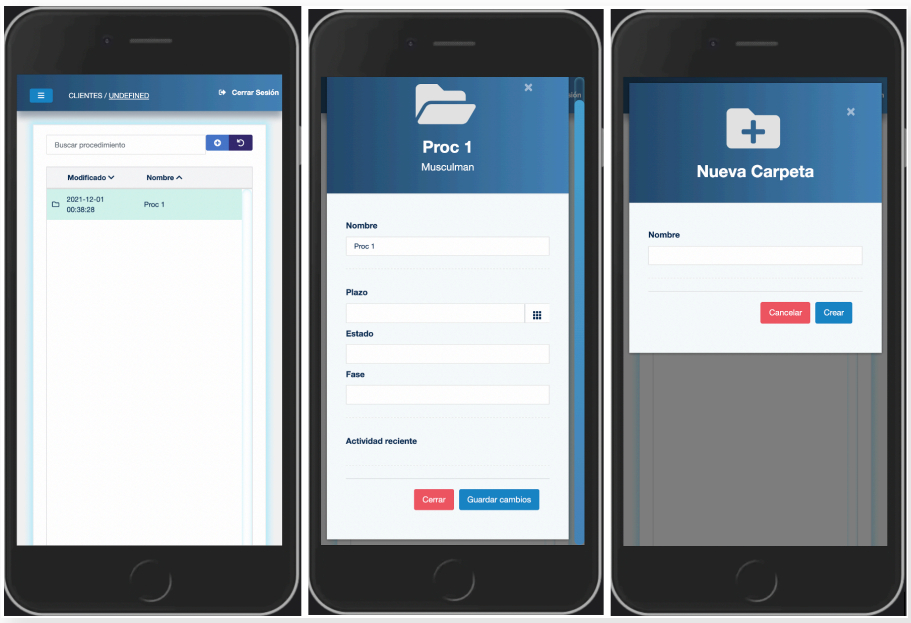


Figura 62. Procedimientos de cliente, edición y ventana de nueva carpeta.

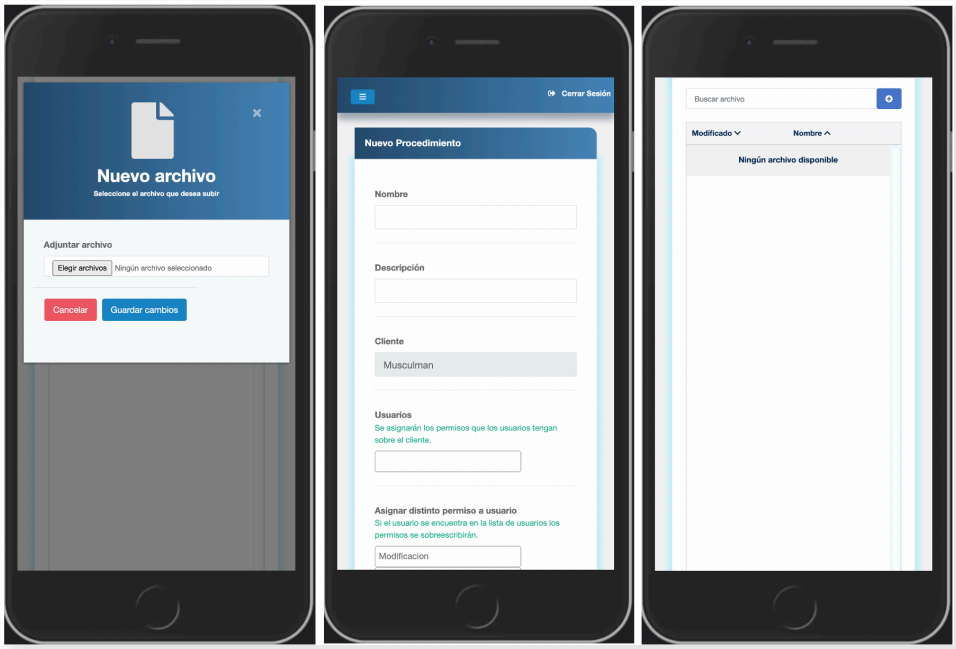


Figura 63. Ventana de nuevo archivo, creación de nuevo procedimiento y vista de archivos de procedimiento.

El administrador elimina los permisos asignados sobre “proc 1” a Caro desde el panel de clientes del administrador.

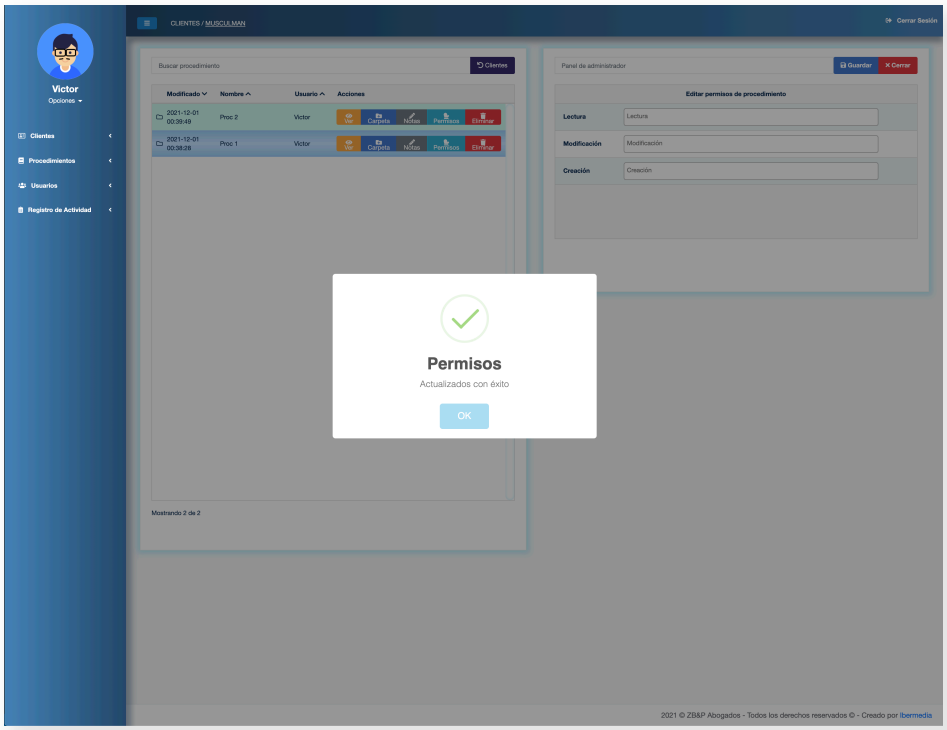


Figura 64. Modificación de permisos sobre un procedimiento.

Caro intenta ahora acceder al procedimiento “proc 1” de Musculman, pero ningún procedimiento le es listado.

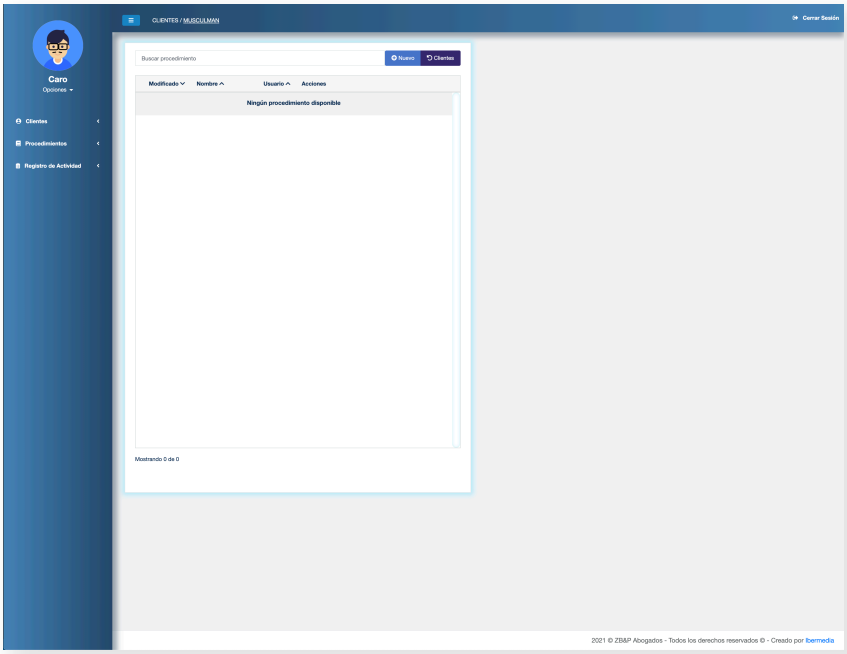


Figura 65. Listado de clientes.

El administrador accede ahora al listado de usuarios y cambia el rol de usuario de Caro, de usuario a administrador.

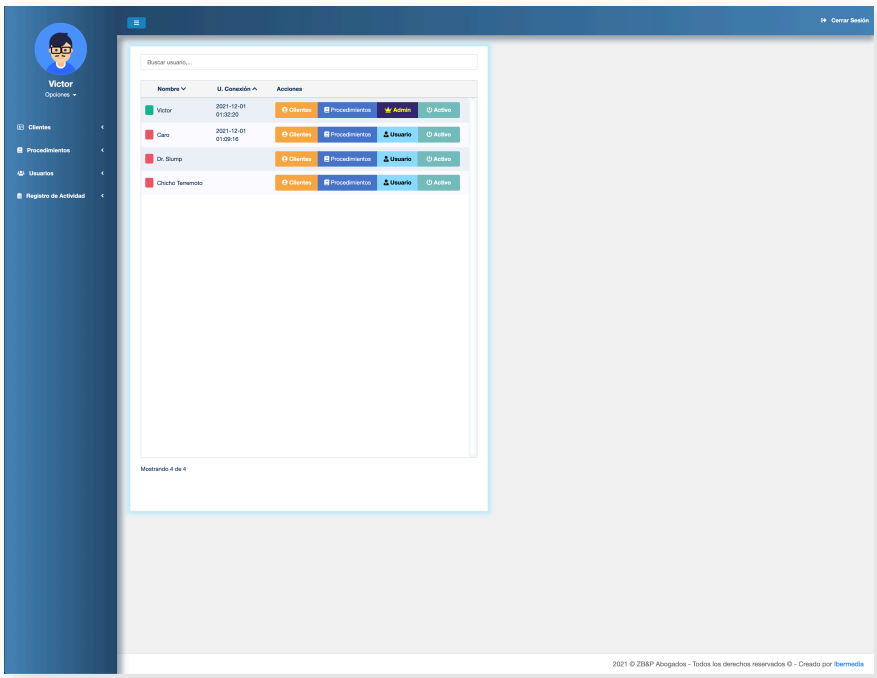


Figura 66. Listado de usuarios.

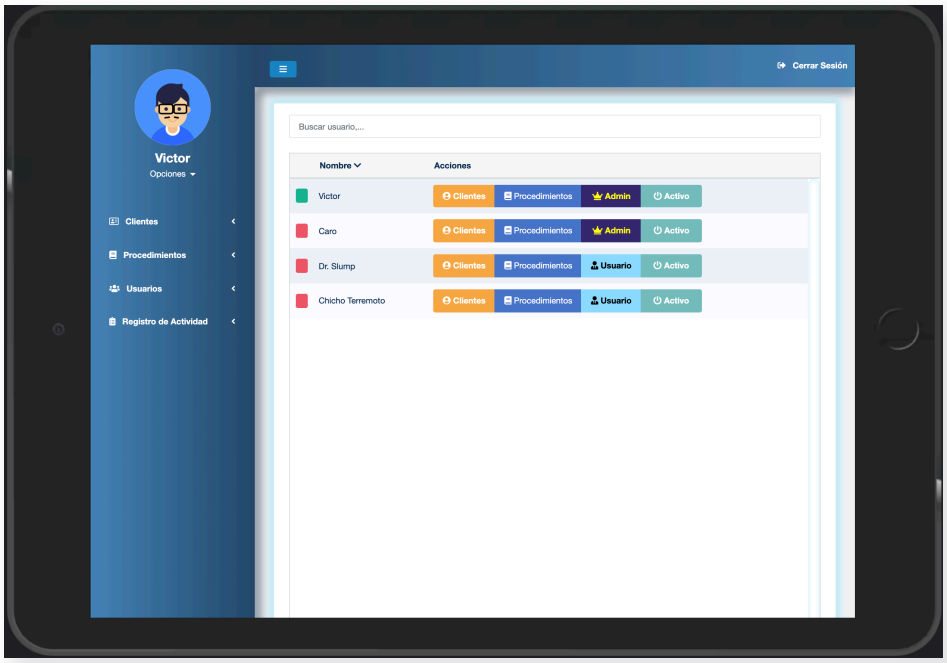


Figura 67. Listado de usuarios.

Caro puede acceder ahora a la totalidad de la información contenida en el sistema, además de a los menus de administración.

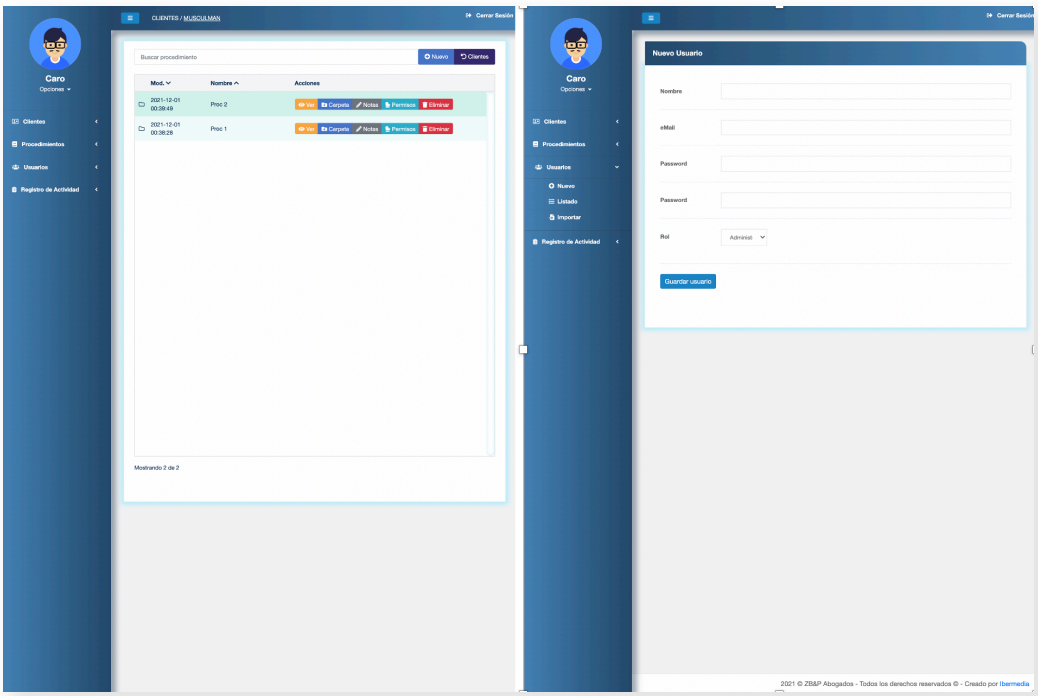


Figura 68. Vista de procedimientos y creación de usuario.

La aplicación ha guardado un registro de toda la actividad realizada que puede ser visualizado y enviado por Caro por correo electrónico, dado que ahora posee privilegios de administrador.

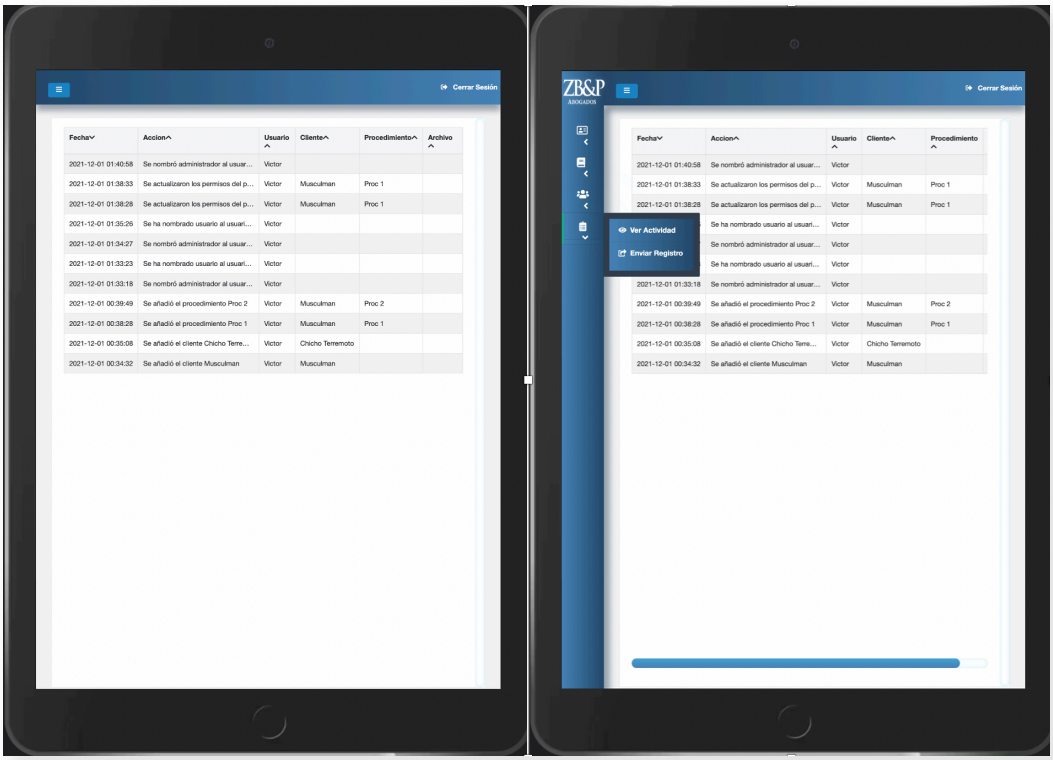


Figura 69. Vista del registro de actividad y envío del registro.

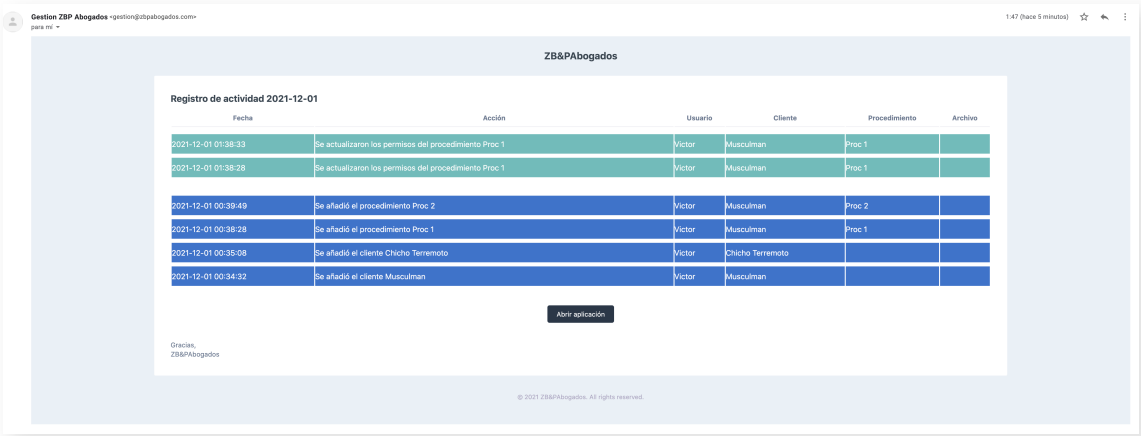


Figura 70. Registro de actividad recibido por correo electrónico.

Capítulo 5. Pliego de condiciones

5.1 Presupuesto personal

El coste del proyecto está dividido por departamentos, y no se incluye el tiempo dedicado al aprendizaje. La siguiente tabla muestra una estimación de los costes:

Departamento	Campo	Coste/tiempo	Horas	Coste
Diseño	Arte	10€/h	60	600€
Programación	Programas	15€/h	200	3000€
Documentación	Otros	10€/h	40	400€
			Total:	4000€

Tabla 9. Presupuesto personal.

5.2 Presupuesto de licencias de software

Software		Coste
HTML, CSS y librerías JavaScript		Gratis
JQuery		Gratis
VisualCode Studio		Gratis
XAMPP		Gratis
Laravel		Gratis
Anydesk		Gratis
MySQL		Gratis
	Total:	Gratis

Tabla 10. Presupuesto material.

5.3 Hosting

Ya que la aplicación se alojará en el servidor de la firma de abogados no se requerirá contratar un hosting.

Capítulo 6. Conclusiones y propuestas de trabajo futuro

Las conclusiones extraidas del proyecto realizado abarcan todo lo relacionado con llegar a comprender el desempeño de un desarrollador de aplicaciones, desde el software y lenguajes de programación utilizados, pasando por la esquematización de la información, la lógica que administrará la aplicación y el diseño de la vista final de usuario.

En cuanto a software y lenguajes de programación utilizados, la conclusión es que, aunque desarrollar una aplicación es un arduo y largo proceso de largos días de trabajo, los softwares como editores de código, paquetes de servidores web y frameworks han facilitado mucho la labor de desarrollo hoy dia. Este conjunto de aplicaciones nos permiten, entre otras cosas, librarnos del tedioso código espagueti, permitiéndonos mantener un código limpio y ordenado en el que sea fácil movernos. También nos permiten ejecutar nuestras aplicaciones de manera local, de manera rápida, sin necesidad de configuración de cada uno de los archivos. Los editores de código añaden hoy día multiples funcionalidades, actualizadas casi a diario, que nos facilitan también la introducción de código, añadiendo autocompletado de funciones o variables frecuentemente empleadas, además de proporcionar magníficas funciones de indexado y filtrado de texto, así como de sustitución.

Referido a la parte de esquematización, lógica y diseño, la conclusión es que son tareas complicadas que deben realizarse de manera ordenada y coherente. Aunque no ha habido gran problema en cuanto a este tema, son aspectos que pueden llegar a tornarse críticos ya que, al ser todos dependientes de los otros, un cambio en uno de ellos suele tener sus repercusiones en los demás.

La conclusión final es que, aunque el mundo está plenamente informatizado hoy en día, no hacemos un uso correcto de la información. Con esto me refiero a que el que la debe almacenar no la almacena como corresponde, el que tiene mucha a su alcance no suele hacer uso mas que de la inútil y el que la crea, muchas veces, miente. Pero por suerte aún queda información servible y de mucha utilidad al alcance de todo aquel que quiera perder el tiempo buscándola, y muestra de ello es el proyecto realizado, que se ha desarrollado utilizando un conjunto de herramientas y lenguajes de programación aprendidas mediante documentación encontrada en internet, algo que hace no tanto tiempo era algo imposible.

Capítulo 7. Bibliografía

[1] Visual Studio Code. https://code.visualstudio.com

[2] HTML/CSS. https://www.w3.org/standards/webdesign/htmlcss

[3] JavaScript. https://developer.mozilla.org/es/docs/Web/JavaScript

[4] JQuery. https://jquery.com/

[5] Ajax. https://www.w3schools.com/xml/ajax_intro.asp

[6] PHP https://www.php.net/manual/en/intro-whatis.php

[7] Laravel. https://laravel.com/docs/7.x

[8] XAMPP. https://www.apachefriends.org/es/index.html

[9] W3schools. https://www.w3schools.com

[10] Bootstrap https://getbootstrap.com/docs/5.1/getting-started/introduction

[11] Faker Library https://github.com/fzaninotto/Faker

[12] MySQL. https://www.mysql.com

[13] PHP & MySQL. https://www.php.net/manual/es/book.mysql.php

[14] FreeFrontend. https://freefrontend.com