TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Gestión y acceso a los servicios de una organización o empresa mediante aplicación móvil

Autor: María Yolanda López Muñoz

Grado: Ingeniería informática

Director: Pedro José Sánchez Sánchez

Fecha: 04/09/2024

Tabla de contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

Este Trabajo de Fin de Grado se centrará en el desarrollo de una aplicación multiplataforma para la gestión básica de recursos humanos y control de proyectos en el ámbito empresarial. Gestiona información sobre proyectos, empleados, clientes, control de horas, vacaciones y justificación de bajas laborales. De esta forma la información se encontrará centralizada permitiendo un mejor entendimiento de los datos y la agilización en cuanto a organización interna dentro de empresas pequeñas y medianas. Esta aplicación es una buena oportunidad para demostrar las habilidades desarrolladas en las asignaturas de programación y gestión de proyectos del grado de Ingeniería Informática de la Escuela Politécnica de Jaén en la Universidad de Jaén.

Hoy en día existen infinidad de sistemas de planificación de recursos empresariales, como son los ERP (Enterprise Resource Planning) que combina una serie de conjuntos integrados de software, utilizables para gestionar e integrar todas las funciones comerciales dentro de una organización como la automatización de procesos de finanza, gestión de recursos humanos, manufacturación, cadena de trabajo, servicios dirigidos al cliente, etc. (E.M. Shehab et al.,2004 y Boykin, 2001 y Chen, 2001 y Yen et al., 2002).

Con la introducción de la pandemia generada por el virus COVID19 se presentaron determinadas necesidades imprevistas para el entorno social. En un escenario dónde se coloca la gran mayoría de PYMEs en una desconexión física, siendo esta imprescindible para su modelo de negocio, posiciona a estas empresas en una situación de vulnerabilidad especial. En consecuencia, se precisaba con urgencia la digitalización de estas empresas con el fin de reforzar la resiliencia de estas frente a la crisis desarrollada por la pandemia.

1.1.Motivación

A causa de las pérdidas económicas producidas por la crisis del COVID19 surgieron los planes de digitalización empresariales como el Kit Digital, programa que tiene como propósito el de digitalizar en su totalidad, a través de organizaciones que actúan como agentes digitalizadores, todos negocios de pequeñas y medianas empresas, microempresas y autónomos, en el que se intenta agilizar el proceso de producción y organización con el objetivo de obtener el mayor rendimiento posible en el menor tiempo (Kit Digital, s.f.). Un objetivo deseable en el desarrollo social sería incrementar la rapidez y la comodidad en las actividades diarias, aprovechando las ventajas que ofrecen las aplicaciones conectadas a Internet para mejorar la eficiencia y optimizar el tiempo en diversas tareas.

La creación de un sitio web privado para la gestión de actividades empresariales se ha convertido en un componente esencial de la competencia actual. Aunque estos sistemas no se exhiban públicamente, su importancia radica en destacar en el mercado mediante una comunicación de calidad con los clientes y en la capacidad de satisfacer sus necesidades con la mayor adaptabilidad posible. Los sistemas ERP, a pesar de su complejidad y funcionalidad integral, suelen tener un costo mensual elevado, lo que puede resultar inaccesible para pequeñas empresas y, al mismo tiempo, limitar su capacidad para cubrir todas sus necesidades. Por esta razón, el objetivo principal de este proyecto es desarrollar una aplicación que funcione como una base versátil y modulable, capaz de adaptarse y crecer con el tiempo.

El propósito de escoger este trabajo es crear desde cero una aplicación web intuitiva, simple que cumpla con los requisititos mínimos que haría una aplicación profesional. Al perseguir este objetivo, se busca reflejar las capacidades y conocimientos adquiridos a lo largo de la carrera, demostrando la habilidad para diseñar e implementar soluciones tecnológicas efectivas que respondan a las necesidades del mercado actual.

1.2.Objetivos del Proyecto

Tras la información expuesta en los párrafos anteriores, se deduce la necesidad de crear un entorno web sencillo, para la gestión de actividades empresariales, que acompañe las actividades empresariales básicas de cualquier empresa, ya sea pequeña o grande.

Esto implica nuevos objetivos básicos:

- Gestionar un sistema que almacene usuarios y diversas actividades comunes.
- Proporcionar seguridad de los datos almacenados y de la funcionalidad interna de dicho sistema.
- Aportar una relación entre el usuario y la interfaz de la aplicación de forma interactiva, accesible, fácil e intuitiva.
- Hacer uso de la arquitectura Modelo-Vista-Controlador (Bascón, 2004).
- Asegurar la adaptabilidad de la aplicación a dispositivos móviles.
- Mantener una implementación del código limpia y concisa.
- Implementar la aplicación utilizando PHP y Laravel además de otras tecnologías relacionadas.
- Configurar y manejar un gestor de versiones como Git para gestionar el proyecto con mayor fluidez y robustez.
- Creación de una aplicación multiplataforma.
- Desarrollo de la memoria del proyecto.

1.3.Estructura del documento

- Capítulo 1 - Introducción: descripción resumida del proyecto, motivación para desarrollar el proyecto, objetivos.
- Capítulo 2 - Estado del Arte: en esta sección se presenta una revisión de las tecnologías actuales en el campo de desarrollo de aplicaciones web con el fin de proporcionar un contexto teórico y práctico.

- Capítulo 3 - Descripción general del entorno tecnológico: profundización al detalle del proyecto comparándolo con otros proyectos presentes en el mercado.
- Capítulo 4 - Diseño e implementación: especificación, estructuración y modelado de las funciones y datos de la aplicación.
- Capítulo 5 - Implementación: Breve explicación de los elementos más importantes en la implementación.
- Capítulo 6 - Evaluación y resultados: valoración de los resultados del trabajo en comparación con los objetivos propuestos además de propuestas para el futuro.
- Capítulo 7 - Conclusiones: evaluación del cumplimiento de los objetivos propuestos, aportaciones más relevantes del trabajo y mejora futura de este.

2. ESTADO DEL ARTE

La identificación y análisis soluciones disponibles en el mercado que puedan compartir características con el desarrollo del proyecto es un factor necesario para cumplir con los objetivos descritos previamente. Por lo tanto, en el próximo capítulo se abordarán los conceptos necesarios en favor de un mejor entendimiento del contexto actual con el fin de identificar las metodologías usadas en posibles soluciones del mercado.

En la actualidad, existe una amplia variedad de aplicaciones web dirigidas a distintos sectores, cuyo objetivo principal es agilizar el flujo de trabajo en numerosas empresas. Aunque cada empresa lleva a cabo una serie de actividades comunes, como la facturación, contabilidad, gestión de inventarios, comunicación con los clientes y administración de recursos humanos, son pocas las soluciones de software empresarial que integran de manera completa todas estas funcionalidades.

Los ERP o Enterprise Resource Planning (Boykin, 2001) son una buena solución adaptable a las necesidades de empresas de todos los tamaños y sectores. Algunos de los ERP más conocidos del mercado son SAP para empresas de gran tamaño, Sage para medianas y Odoo para pequeñas empresas. A lo largo de este capítulo se profundizará en Odoo como posible solución al problema que expone el proyecto al ser un proyecto de código abierto proporcionando una transparencia en el uso de sus metodologías y tecnologías que facilita la investigación de este en comparación con los demás ERP comunes del mercado.

2.1.Odoo

Odoo (The Odoo Story, s.f.) es un software de gestión empresarial de código abierto que integra una amplia variedad de aplicaciones para cubrir las necesidades de diferentes áreas de una empresa. Su desarrollo comenzó como una TinyERP (sistema de gestión empresarial de tamaño reducido) en 2005, por Fabien Pinckaers, como respuesta en contraposición a estrategias de mercado de SAP, otra solución software del mercado.

Gracias a su código abierto altamente personalizable, Odoo se sustenta en una combinación de tecnologías que le permiten ofrecer una amplia gama de funcionalidades y una gran flexibilidad. Su stack tecnológico (combinación de lenguajes de programación, frameworks, bibliotecas, herramientas y tecnologías que se utilizan para desarrollar e implementar una aplicación de software o sistema) se compone de Python como lenguaje de programación principal, PostgreSQL como base de datos relacional, XML junto con QWeb y Owl, framework de desarrollo de interfaces de usuario, junto con Werkzeug, un toolkit WSGI (Web Server Gateway Interface) usado para crear aplicaciones web en Python.

Además, usa otras tecnologías y conceptos destacables como el patrón MVC (Modelo Vista Controlador) basado en la arquitectura de modelo de tres capas, ORM (Object Relational Mapper), y el framework WSGI. Estas tecnologías serán explicadas en profundidad más adelante a lo largo del capítulo.

En los últimos años, los sistemas ERP han experimentado una evolución significativa, impulsada por avances en diversas tecnologías. Estos avances han permitido que los ERP ofrezcan funcionalidades más robustas, escalables y eficientes, adaptándose a las crecientes demandas del mercado global.

Paralelamente, el progreso tecnológico exponencial, especialmente con la expansión de Internet, ha transformado la manera en que se accede a la información, haciéndola casi instantánea y disponible desde cualquier lugar y en cualquier momento. Esta transformación ha generado nuevas necesidades para los usuarios, tales como seguridad, escalabilidad, portabilidad, accesibilidad y eficiencia, características que las aplicaciones de escritorio a menudo no pueden satisfacer plenamente, pero que sí son proporcionadas de manera efectiva por las aplicaciones web.

La web es un subconjunto de Internet, red de redes dónde reside toda la información, siendo la web la precursora del Internet. A lo largo de la historia, desde 1990, ha evolucionado en distintas versiones siendo la 2.0 (2024), la más usada por los usuarios comunes debido a su capacidad interactiva y visual. Las aplicaciones web emergen a partir de gestores de contenidos, pensados para crear y administrar los contenidos en páginas web (Latorre, 2018). Esta aplicación almacena toda la

información, tanto la interfaz como su contenido. Puede presentarse como una intranet, red privada que tiene como objetivo facilitar la comunicación interna de una organización de forma segura y rápida.

Una aplicación web necesita un dominio web, dirección única usada para identificar un sitio web a través del sistema de nombres de dominios DNS. Esta dirección se debe de registrar a través de agentes registradores de dominios que proporcionarán su respectivo nombre de dominio también de forma única. El nombre de dominio estará enlazado a una dirección IP, que se enviará a un servidor, para que este pueda procesarla.

Las aplicaciones web suelen alojarse en un servidor de almacenamientos para el procesado y publicación de sitios web. Estos pueden ser:

- Compartido, dónde se comparte el espacio del servidor con otros usuarios.
- Privados virtuales (VPS), con mejor rendimiento ya que se reservan recursos sólo para un usuario.
- Dedicado, ofrece un servidor físico en su totalidad.
- En la nube (Cloud Hosting), que utiliza varios servidores en la nube para distribuir el tráfico. (Pollock, 2003).

Además, en cuanto al procesado de peticiones, se suelen basar en una arquitectura web cliente-servidor, con las ventajas de una centralización de los datos y la seguridad que otorga el protocolo HTTP durante la conexión a Internet.

2.1.1.Servidor web

Es un estándar de Python que define cómo los servidores web deben comunicarse con las aplicaciones web escritas en Python (PEP 3333 – Python Web Server Gateway Interface v1.0.1**,** 26 de septiembre de 2010). Es esencialmente un protocolo de comunicación que permite a los servidores web entender y ejecutar aplicaciones Python.

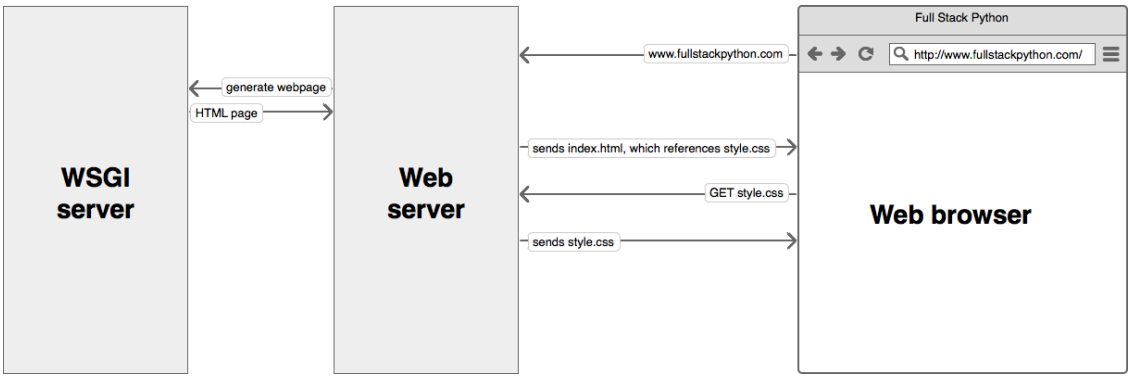

Figura 2.1. Diagrama de Cliente/Servidor. Fuente: PEP 3333 – Python Web Server Gateway Interface v1.0.1, https://peps.python.org/pep-3333/.

En la Figura 2.1 se muestra el funcionamiento de un servidor web, pero antes de entrar en su explicación es necesario aclarar algunos conceptos introductorios como la comunicación HTTP y el protocolo que la define.

El protocolo HTTP (Hyper Text Transfer Protocol) establece una conexión TCP entre un cliente y un servidor. Primero se estable una conexión TCP cada vez que el cliente envía una petición al servidor. El servidor recibe la petición, la procesa y envía al cliente una respuesta, es entonces cuando el servidor cierra la conexión TCP si la conexión es no persistente, en caso contario mantiene abierta la conexión para recibir más mensajes (ver en Figura 2.2).

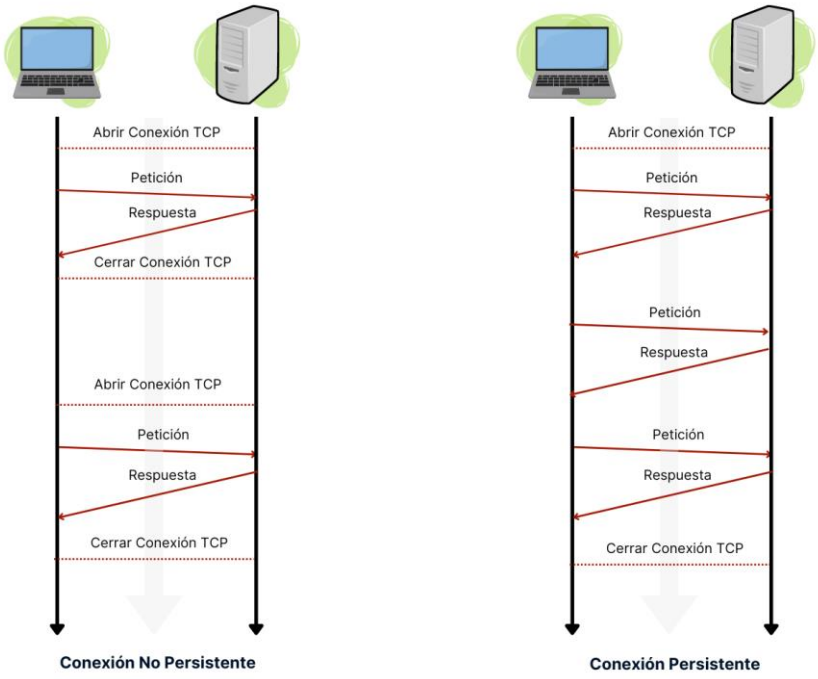

Figura 2.2. Comunicación HTTP no persistente y persistente. (Elaboración propia).

Esto toma más sentido en las aplicaciones web donde el cliente necesita enviar peticiones múltiples cada vez que interactúe con los elementos de un sitio web, reduciendo el uso de recursos del servidor como la memoria y la CPU y mejorando la respuesta de la página respecto al tiempo.

Aunque todo parecen ventajas, presentan algunos inconvenientes en cuanto a seguridad y congestión del sitio web por lo que es necesario mantener un control para que el periodo del tiempo con la conexión abierta no sea excesivo. (Rawal et al,2012).

Continuando con el servidor web, su funcionamiento consiste en recibir una solicitud HTTP, pasarla a una aplicación WSGI (una función o una clase). Esta procesa la solicitud, realiza las operaciones necesarias y genera una respuesta HTTP. La aplicación WSGI devuelve la respuesta al servidor web, que la envía al cliente y muestra la muestra por pantalla.

Para garantizar que estas operaciones se gestionen de manera eficiente y ordenada dentro de un proyecto, es fundamental contar con una estructura lógica bien definida. En este contexto, la arquitectura de tres capas resulta esencial, ya que permite separar las responsabilidades y facilita tanto el desarrollo como el mantenimiento de la aplicación. Odoo, por ejemplo, implementa esta arquitectura con el objetivo de optimizar la organización del proyecto.

2.1.2.Arquitectura de tres capas

La definición de esta arquitectura favorece que las características de la aplicación se dividan en componentes más simples de forma que permita una mayor agilidad de trabajo al poder dividirse en equipos de desarrollo por cada capa, siendo cada capa funcional por separado es fácil añadir modificaciones sin revisar toda la aplicación entera. La arquitectura de tres capas (Del Valle, 2007) es la técnica comúnmente usada para desarrollo de aplicaciones web, que suele dividirse entre el navegador/presentación, la aplicación y la base de datos (ver Figura 2.3).

- Capa de presentación, en esta capa el servidor visualiza la interfaz de usuario, todas las interacciones del usuario son enviadas a la capa

intermedia y para ser procesadas. Su función principal es la de mostrar información al usuario y recopilar datos de este. En esta capa se usan aquellas tecnologías web como HTML, CSS y JavaScript para la interfaz de usuario además de QWeb.

- Capa de aplicación, se encarga de procesar los datos recibidos de la capa de presentación. Aplica la lógica empresarial a través de reglas específicas que definen los procesos para manejar y transformar la información con el fin de enviarla a la base de datos. El nivel de aplicación también puede añadir, suprimir o modificar datos en el nivel de datos. En esta capa se usa el patrón MVC, que divide la aplicación en tres elementos modelo, vista y controlador y se explicará más adelante con mayor profundidad en el capítulo 3.
- Capa de datos, recoge los datos de la capa de aplicación para su gestión y almacenaje haciendo persistente toda la información. En esta capa se usa la base de datos PostgreeSQL y ORM para el mapeado de objetos a tablas de la base de datos.

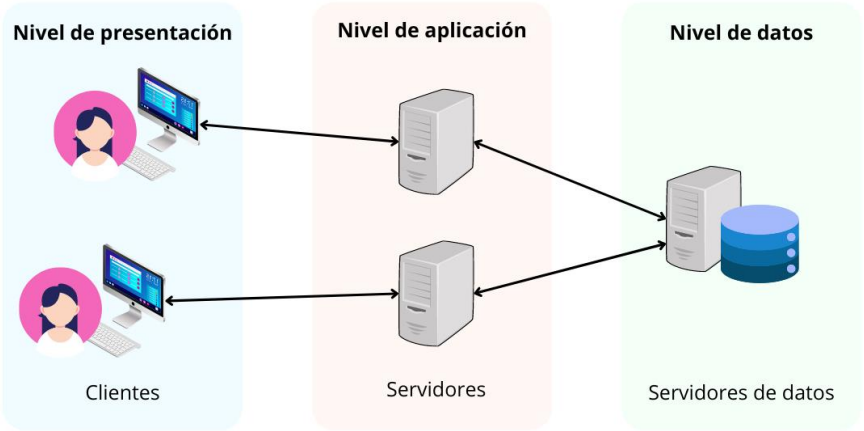

Figura 2.3. Arquitectura 3 capas. (Elaboración propia).

La arquitectura de tres capas en Odoo proporciona una estructura sólida y organizada para el desarrollo de aplicaciones empresariales. Al separar las responsabilidades en presentación, lógica de negocio y datos, se facilita la creación de aplicaciones escalables, mantenibles y personalizables.

2.1.3.Tecnologías Front de Odoo.

Odoo utiliza una combinación de tecnologías front-end para crear interfaces de usuario modernas, intuitivas y personalizables. Aunque no se basa en un framework de front-end específico, Odoo aprovecha tecnologías como:

- HTML y CSS: estructura y estiliza el contenido de las páginas web.
- JavaScript: agrega funcionalidades dinámicas a las vistas, como validaciones de formularios, animaciones y eventos.
- QWeb, motor de plantillas basado en XML que se utiliza para generar interfaces de usuario dinámicas y personalizables. QWeb permite combinar HTML, CSS y JavaScript con expresiones de Python para crear vistas complejas.
- Bootstrap: framework CSS que proporciona una estructura predefinida y componentes listos para usar. Odoo utiliza Bootstrap para acelerar el desarrollo de interfaces de usuario y garantizar una apariencia consistente en diferentes dispositivos.
- Owl: framework de componentes basado en JavaScript que se utiliza para crear interfaces de usuario modernas y eficientes. Owl proporciona una forma estructurada de organizar y reutilizar componentes, lo que facilita el desarrollo y el mantenimiento de aplicaciones complejas.

A lo largo de este capítulo se han explicado los conceptos necesarios para comprender el contexto actual del conocimiento relacionado con el tema del proyecto. Se ha presentado Odoo como una posible solución disponible en el mercado para abordar el problema planteado por el proyecto. Además, se ha analizado en profundidad el uso de las metodologías y tecnologías que emplea, con el objetivo de esclarecer la definición de una nueva solución, la cual se describirá en el capítulo siguiente.

3. DESCRIPCIÓN GENERAL DEL PROYECTO

El objetivo principal de este Trabajo Fin de Grado en Ingeniería Informática es el desarrollo de un producto software, específicamente, una intranet como aplicación web para la gestión básica empresarial que cubra las necesidades que generan las tareas de recursos humanos.

Primero, se ha escogido un nombre para la aplicación con la meta de simular un desarrollo de aplicación real, por lo que desde ahora se usará GestUJA, nombre procedente de la palabra Gestión y UJA (Universidad de Jaén) cuyo logo será el mostrado en la Figura 3.4.

Figura 3.4. Logo de la aplicación GestUJA. (Elaboración propia).

GestUJA simulará una aplicación web que sirva para gestionar todos los trámites e información correspondiente a un empleado de cualquier empresa de forma que todo empleado pueda acceder a su información personal dentro de la empresa como la adjudicación de vacaciones, la gestión de bajas, proyectos activos en los que están involucrados, etc.

Por otra parte, pretende cubrir las necesidades de los empleados de recursos humanos, que tendrán unos permisos específicos de administrador con el que podrán acceder a determinadas áreas de la plataforma que otros usuarios sin el rol de administrador. Este rol les permitirá el uso de acciones como la creación y eliminación tanto de usuarios, como de proyectos o material de la empresa además de la gestión de roles y permisos de la plataforma.

Se usará una metodología basada en tres capas al igual que Odoo, usando tecnologías web HTML, CSS, JavaScript, Blade, Bootstrap y widgets para la capa de

presentación, el framework Laravel para la capa de lógica de negocio junto y por último MariaDB y Eloquent ORM para la capa de datos.

A continuación, se expondrán todas las tecnologías que se usarán para el desarrollo del proyecto y los motivos de su elección en profundidad:

3.1.Entorno de desarrollo

En una primera instancia se usará Laragon (Documentation | Laragon, 1 de marzo de 2019), un entorno de desarrollo local diseñado para Laravel, dada la fácil instalación y puesta en marcha y su compatibilidad con variadas bases de datos.

Además, proporciona configuraciones opcionales como el PATH, instalación de otras bases de datos y herramientas de control de versiones como Yarn o Git, acceso a una terminal, acceso a la base de datos y a los ficheros de la web o a phpMyAdmin para la gestión de la base de datos.

Este entorno se usará junto a Apache como servidor web, el lenguaje PHP y Visual Studio Code para la implementación del código, la base de datos MariaDB además de Git para control de versiones con un repositorio remoto en Git Hub.

3.2.Tecnologías Back-end

En cuanto a la gestión de la lógica, base de datos y la interacción con el interfaz de usuario es necesario una serie de tecnologías que se encargarán de procesar las solicitudes de los usuarios y realizar las operaciones sobre los datos correspondientes a la lógica de negocio. El lenguaje de programación PHP, servidor web Apache, la base de datos MariaDB y el framework de aplicación Laravel serán elementos claves para cumplir con la funcionalidad de la aplicación.

3.2.1.Lenguaje de programación

Existen una gran variedad de lenguajes de programación dirigidos a programación web: ASP.net, Java, Ruby, Python, PHP, etc. Sin embargo, se ha decidido usar PHP Hypertext Preprocessor, al ser el lenguaje más extendido para web y con mayor comunidad de usuarios proactiva detrás.

Es usado por aplicaciones conocidas como Facebook y Wikipedia por su simplicidad para aprender y usar, su código abierto y la compatibilidad con la mayoría de bases de datos.

PHP es mucho más dinámico que la mayoría de lenguajes, proporcionando un mejor tiempo de respuesta y conexión a una base de datos grande consecuencia de ser un lenguaje al lado del servidor. Esto quiere decir que se delegará la interpretación y generación de código HTML junto con el cargado de plugins y otros servicios al servidor. El navegador recibirá el código HTML generado por el servidor y lo interpretará para visualizar la página web tal y como se puede ver en la Figura 3.5.

Figura 3.5. Funcionamiento PHP. Fuente: TUTORIAL PHP & MYSQL - 1. Introducción, https://geekytheory.com/tutorial-php-mysql-1-introduccin/.

3.2.2.Servidor Web

Para la puesta en marcha de GestUJA necesitaremos un servidor web que aloje la aplicación y la ejecute de forma que se visualice desde cualquier dispositivo conectado a Internet.

En este caso se usará Apache 2.0 (About the Apache HTTP Server Project, s.f.), su primera versión se generó a partir del servidor web más usado de aquel momento NCSA HTTPd, que procesó el primer navegador web Mosaic. El creador de este servidor Rob McCool dejó el NCSA en 1994 dando lugar a que programadores webs con experiencia tuvieran que actualizar ellos mismos el servidos. Un grupo de 8 de estos programadores web, lideraros por Brian Behlendorf and Cliff Skolnick, se pusieron de acuerdo en fusionar 8 versiones con mejoras del servidor originando la primera versión de Apache. Fue tal su impacto que a día de hoy todavía sigue siendo uno de los servidores HTTP más usados y versátiles del mundo.

Apache es sin duda la mejor opción para este tipo de aplicaciones basadas en la web por si simplicidad, seguridad, flexibilidad y multiplaforma. Se ha escogido este servidor web por preferencia personal con motivo de su amplia documentación web y la disposición de una gran comunidad involucrada en dar soporte, además de la experiencia recogida durante la carrera.

3.2.3.Base de datos

Para la base de datos se ha decantado por usar MariaDB (MariaDB in brief, s.f.). Creada a partir de MySQL por Michael Widenius, desarrollador de MySQL, en 2009. MariaDB es una base de datos relacional conocida por su velocidad y eficiencia. Presenta todas las ventajas de MySQL, es totalmente compatible con migraciones desde MySQL, y con versiones propias anteriores, aparte de incluir características de seguridad adicionales y más motores de almacenamiento que MySQL. Con el tiempo se ha convertido en una de las mejores opciones a causa de su versatilidad y alto rendimiento.

Por otro lado, ante otras alternativas como PostgreSQL se ha elegido MariaDB a causa de presentar mayor rendimiento notorio conforme sea mayor el número de peticiones (Rendimiento de MariaDB y PostgreSQL, 2020). Aunque en un inicio GestUJA se dirija como una intranet base para empresas pequeñas o medianas, se podría adaptar para empresas con mayor cantidad de empleados sin limitar el rendimiento.

3.2.4.Arquitectura web MVC

Siguiendo la línea de pensamiento de la arquitectura de tres capas existe el patrón de diseño de software Modelo-Vista-Controlador (Bascón Pantoja,2004), que también separa las responsabilidades de la aplicación web separando los datos, la interfaz de usuario y el control de estos dos:

- Clase Modelo: se desarrolla en el back end y contiene toda la lógica de los datos. Obtiene los datos de la base de datos para un mejor manejo y gestión de estos. Cada modelo almacena toda la información posible de cualquier entidad procedente de la base de datos.
- Clase Controlador: interrelaciona el modelo (los datos) con la vista (interfaz). Todas las interacciones del usuario recibidas de la interfaz se procesan en esta clase.
- Clase Vista: participa en el front end de la aplicación, se encarga de visualizar la información recogida de los modelos y proporcionada por los controladores en la interfaz buscando la interacción del usuario con estos.

La Figura 3.6 presenta un esquema que ilustra el patrón MVC, destacando las tres clases descritas previamente y sus interrelaciones. En dicho esquema, se observa cómo un usuario realiza una solicitud desde su terminal, la cual es recibida por el controlador. Este, a su vez, gestiona la petición utilizando los datos proporcionados por los modelos, y finalmente, actualiza la vista, permitiendo al usuario visualizar en su terminal los resultados de la solicitud.

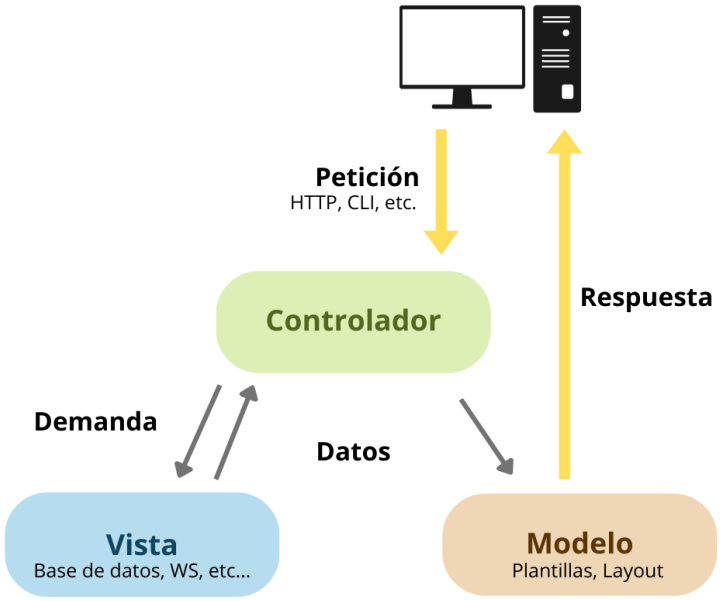

Figura 3.6. Modelo Vista Controlador. (Elaboración propia).

La división de responsabilidades en tres componentes distintos, cada uno con una funcionalidad específica, que optimiza el mantenimiento y la escalabilidad del código, facilitando la modificación o ampliación de la aplicación sin impactar otras partes del sistema. Al mantener estos componentes separados, se mejora la eficiencia en la localización y corrección de errores. Además, el modelo puede ser reutilizado en diferentes áreas del código, lo que reduce la duplicación y mejora tanto la organización como la comprensión del mismo. Esta separación también permite el desarrollo paralelo, ya que la implementación de los módulos está completamente desacoplada.

La adopción del patrón MVC conlleva una serie de ventajas que simplifican la planificación, organización e implementación del código. Este patrón ofrece una estructura altamente adaptable a cualquier tipo de proyecto, particularmente al facilitar un análisis y diseño más profundos y eficientes.

3.2.1.Laravel

Laravel (Laravel: The PHP Framework for Web Artisans, 2011), creado por Taylor Otwel en 2011, es un framework novedoso y potente de aplicaciones web en PHP que proporciona una sintaxis sencilla mediante Blade, facilitando un código ordenado e inteligible para otros desarrolladores más expertos, siendo muy adaptable para cualquier tipo de proyecto, ya sea sencillo o complejo.

Se adapta a proyectos sencillos y complejos por el uso del paradigma POO (programación orientada a objetos), su destacada estructuración y velocidad de desarrollo frente al resto de opciones: routes, closures, helpers, validación de formularios, middlewares, internacionalización, objetos ORM, motor de plantillas, etc.

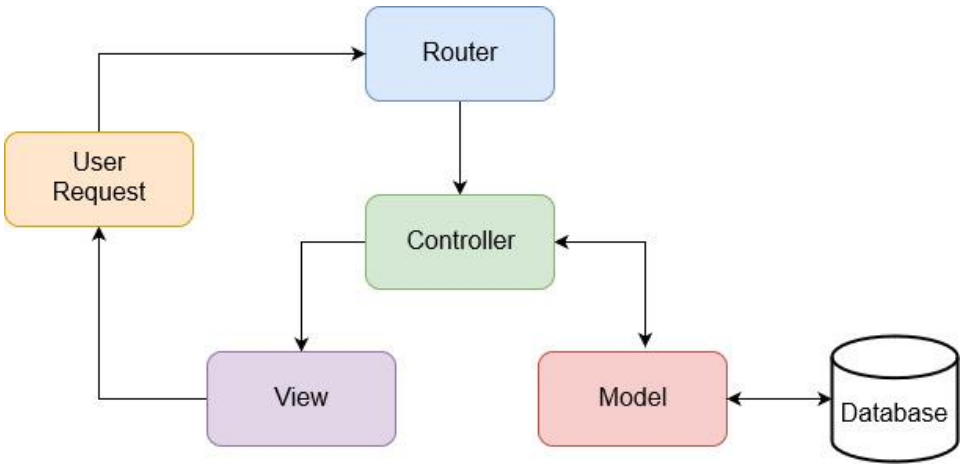

Figura 3.7. MVC de Laravel. Fuente: guia-laravel-6-paso-a-paso, https://github.com/jvadillo/guia-laravel-6-paso-a-paso/blob/master/manuscript/01-Introduccion.md.

Gracias a Eloquent ORM ilustrado en la Figura 3.7, mapeador objeto-relación, es más fácil la incorporación de la programación orientadas a objetos en PHP. Contiene herramientas como las migraciones que genera modelos de entidades automáticamente de las tablas de la base de datos, propiciando, junto a Eloquent, a seguir con mayor facilidad el patrón de diseño Modelo-Vista-Controlador. A esta arquitectura, con el añadido enrutamiento como intermediario de las peticiones procedentes de la interfaz y los controladores se obtiene una variación del MVC.

Las tecnologías back-end definen la funcionalidad de la aplicación y los datos. Laravel como framework de aplicación de PHP junto a MariaDB y el servidor web Apache,

conformarán las capacidades y características de la aplicación. Por el contrario, las tecnologías front end son responsables de la apariencia, diseño y comportamiento de la interfaz de usuario.

3.3.Tecnologías Front-end

La visualización de toda la lógica de los datos es de importancia, por ello es un requerimiento hacer uso de tecnologías que permitan la interacción del usuario con la aplicación además de una correcta disposición de elementos y efectos interactivos de forma que la experiencia del usuario sea amena, ágil y cómoda. Bootstrap y Blade serán los elementos encargados de dar forma a todo el aspecto visual de la aplicación.

3.3.1.Bootstrap

Para el diseño del front end de la aplicación necesitaremos el uso de Frameworks y plantillas con el fin de obtener una estructura base que englobe todas las funcionalidades básicas y se pueda modificar de manera sencilla en el futuro.

En el proyecto se usará Bootstrap (Bootstrap, s.f.), framework de código abierto que combina CSS y JavaScript, creado por X, conocida anteriormente como Twitter.

Usa un sistema de cuadrículas para maquetar la interfaz por columnas, estas se van adecuando a la resolución de la pantalla de cada dispositivo siendo muy buena opción para conseguir adaptabilidad responsiva en nuestra aplicación.

Hace uso de otras tecnologías HTML5, CSS3, jQuery, GitHub y JavaScript, integrándose perfectamente con las librerías de esta última.

Además, dispone de una gran documentación y soporte de la comunidad siendo compatible con los navegadores más usado debido a que es de código abierto, además, se mantiene constantemente actualizado por X.

3.3.2.Blade

Blade (Laravel, s.f.) es el motor de plantillas predeterminado de Laravel. Es una sintaxis sencilla y potente que permite crear plantillas HTML dinámicas en aplicaciones Laravel. Proporciona una forma sencilla embeber código PHP dentro de las vistas HTML, lo que facilita la creación de contenido dinámico y la separación de la lógica de presentación de la lógica de negocio.

Sus directivas existen en gran variedad y permiten controlar el flujo de ejecución, declarar condicionales y bucles entre otras funciones. Asimismo, divide las plantillas en secciones que pueden ser reutilizadas favoreciendo la comprensión del código.

A diferencia de algunos motores de plantillas PHP, Blade no te restringe de usar código PHP en plantillas. De hecho, proporciona una sintaxis para evaluar expresiones PHP. Todas las plantillas de Blade se compilan en código PHP puro y se almacenan en caché hasta que sean modificadas, mejorando el rendimiento de la aplicación.

3.4.Métodos

Para alcanzar los objetivos previamente establecidos, se procederá a crear un repositorio privado en la plataforma de alojamiento de repositorios GitHub, el cual permitirá gestionar y administrar el proyecto de manera eficiente. Seguidamente, se configurará Laragon, un entorno de desarrollo integral que soporta tecnologías como Apache, MariaDB y PHP, entre otras, donde se llevará a cabo el desarrollo de la aplicación.

Una vez instalado Laragon, se iniciará la adaptación del modelo de base de datos, lo que permitirá avanzar en el diseño de las funcionalidades esenciales que gestionarán la plataforma (back-end), incluyendo las conexiones con el servidor y la base de datos, el controlador de cada vista de la aplicación, así como el proceso de autenticación y acceso.

Con el back-end desarrollado, se procederá a trabajar en la interfaz de usuario (front-end) de la aplicación. Para ello, se empleará un framework adecuado que facilitará la creación de una plantilla común para toda la plataforma.

Finalmente, una vez estructurados el back-end y el front-end, ambos podrán comunicarse de manera efectiva con la base de datos, lo que permitirá visualizar y verificar la funcionalidad deseada hasta lograr su completo funcionamiento.

Los procedimientos anteriormente mencionados serán explicados con mayor detalle en el capítulo 5 y el Anexo I. Antes de abordar esos aspectos técnicos específicos, es fundamental entender las etapas previas de análisis y diseño.

4. ANÁLISIS Y DISEÑO

El análisis y diseño de una aplicación web son etapas cruciales en el desarrollo de una aplicación efectiva y funcional.

Por un lado, en el análisis se definen los objetivos, requisitos y modelado de los casos de uso. Es decir, se identifican los objetivos de la aplicación y las necesidades de los usuarios para posteriormente crear diagramas que representen las interacciones entre los usuarios y el sistema, simplificando la visualización de la lógica de la aplicación.

Ulteriormente, en el diseño se abstrae la información para diseñar la arquitectura e la interfaz del usuario a favor de un mejor desarrollo del back end y front end.

Estos criterios aseguran que la aplicación cumpla tanto con los requisitos técnicos como con las expectativas del usuario en referencia a su funcionalidad, seguridad, usabilidad, eficiencia y adaptabilidad.

A priori del análisis y diseño es necesario definir una determinada organización con objeto de orientar las fases que concretarán el ciclo de vida de un software.

4.1.Fases de trabajo

Para el desarrollo del proyecto se aplicará una metodología RUP (Proceso Racional Unificado). La metodología RUP (Castro Gil, 2004) consiste en una estructura de trabajo de proceso con el objetivo del producto y por tanto basada en el modelo UML (Unified Modeling Language), cuando se habla de programación orientada a objetos.

Es un marco de trabajo iterativo incremental para el desarrollo del software cuyas características esenciales son:

- Ser iterativo e incremental: el desarrollo se ejecuta en iteraciones permitiendo un mejor control de la calidad del producto.

- La utilización de casos de uso: se basa en los casos de uso para captar los requisitos funcionales de la aplicación.
- Centrarse en la arquitectura: enfatiza la importancia de una arquitectura robusta y simplificada en etapas tempranas del proyecto.
- Gestionar los riesgos: facilita la identificación y solución de riesgos durante todo el ciclo de desarrollo.

Figura 3.8. Metodología RUP. Fuente: PRESENTACIÓN RUP | PPT, https://es.slideshare.net/slideshow/presentacin-rup/54280787.

Todas estas características permiten seguir una estructura dinámica del proyecto siguiendo las prácticas mostradas en la Figura 3.8. Esta estructura se compone de las siguientes etapas:

4.1.1.Etapa Ingeniería

Esta etapa tiene como objetivos la conceptualización y diseño inicial del proyecto. Estas actividades se llevan a cabo en las fases de concepción y elaboración:

- **Fase de concepción:** dónde se conceptualiza el producto, se especifican los objetivos, el coste del proyecto y los plazos. Esto correspondería con el estudio de la aplicación web, planificación del proyecto y la obtención y análisis de requisitos.
- **Fase de elaboración:** se establecen los casos de uso a partir de un análisis de requisitos previo para definir la arquitectura base del sistema. En esta fase se establecerá el diseño de la base de datos y el diseño general de la arquitectura de la aplicación.

4.1.2.Etapa de producción

Se enfoca en la implementación, pruebas y despliegue del sistema. Estas actividades se llevan a cabo en las fases de construcción y transición:

- **Fase de construcción:** se implementa el back-end y front-end dando al completo la funcionalidad de la aplicación.
- **Fase de transición:** que corresponde a la fase de pruebas dónde el producto toma forma de prototipo o se encuentra completo. Además, se realiza la documentación de dicho producto.

Estas fases y etapas permiten un desarrollo iterativo e incremental, asegurando que el software se construya de manera eficiente y con alta calidad.

Al ingresar en la etapa de ingeniería, y tras haber conceptualizado la aplicación y definido sus objetivos, el siguiente paso es llevar a cabo el análisis de requisitos.

4.2.Análisis de requisitos

A continuación, se procederá a definir brevemente el proyecto asimismo la obtención y posterior análisis de los requisitos necesarios para implementar las diversas funcionalidades con las que el sistema podrá cumplir con los objetivos.

Para cumplir con estas funcionalidades se debe de definir detalladamente los requisitos necesarios para el correcto funcionamiento de la aplicación. Los requisitos se dividen en dos tipos:

- **Funcionales:** definen el comportamiento de la aplicación respecto a las acciones del usuario.
- **No Funcionales:** definen las propiedades de la aplicación como el rendimiento, la seguridad o la disponibilidad.

4.2.1.Actores

A continuación, se presentan los distintos actores que presenta la aplicación:

Empleado y Usuario

El actor usuario representa a todo usuario que interactúe con la aplicación. Sus funciones básicas es poder acceder y salir libremente de la aplicación mediante autentificación y poder interactuar con el contenido de esta. Los demás actores heredarán las funcionalidades de usuario al ser una extensión lógica de este.

El actor empleado identifica a toda persona que ejerza actividades dentro de la empresa. Sus funcionalidades deben de adecuarse a estas actividades que se verán reflejadas en la aplicación.

Una vez acceda a la plataforma a partir de sus credenciales, podrá ver un listado de notificaciones, su perfil de usuario y modificar algunos campos en este, acceder a la gestión de sus vacaciones dónde se le mostrarán los días que le quedan de vacaciones, sumando la información o estado de estas además de solicitarlas.

Por otra parte, será posible ver el listado de proyectos en los que está involucrado de forma activa junto con la información de estos y modificar algunos campos en el caso en el que esté implicado en el proyecto, consultar un listado de clientes con su información de contacto, y, por último, podrá acceder a un listado de material de la empresa.

Administrador

Este tipo de actor podrá ejecutar algunas acciones que no recoge el actor de empleado, como las acciones de crear, modificar y eliminar cualquier apartado del sistema como las notificaciones, los empleados, los clientes, los proyectos, el material o procesar las solicitudes de las vacaciones.

Explicados ya los tipos de requisitos y los actores se introducirá a continuación los requisitos necesarios para la correcta funcionalidad del sistema.

4.2.2.Requisitos funcionales

Los requisitos funcionales definen lo que un software debe hacer. Son como las instrucciones detalladas que le damos a un programador para que construya una aplicación que cumpla con nuestras necesidades específicas. El proyecto hará uso de los siguientes requisitos funcionales:

Sistema

RF (1).- El sistema no permite mostrar el contenido si el usuario no está registrado, así como participar y modificar otras configuraciones de la página.

RF (2).- El sistema dispone de un módulo de gestión de los usuarios.

RF (3).- El sistema acepta solamente dos tipos de usuarios: administración y empleados.

RF (4).- El sistema tiene que listar los usuarios que están registrados.

RF (5).- El sistema tiene que listar los proyectos que representen las actividades de la empresa.

RF (6).- El sistema tiene un servicio de mensajería con el fin de enviar mensajes a los usuarios a través de un correo electrónico para enviar notificaciones o sugerencias.

RF (7).- La creación y modificación de usuario debe almacenar campos de tipo fecha para saber cuándo se inicia y se acaba su contrato además de información personal de contacto como su correo personal o teléfono móvil.

RF (8).- Los proyectos de la aplicación deben especificar un atributo que indique un código único e identificativo.

RF (9).- Los proyectos tienen que permitir vincular los empleados que lo desempeñan, además del cliente.

Usuario

RF (10).- Todos los usuarios deberán identificarse para acceder a la aplicación usando un nombre de usuario y una clave. El nombre de usuario corresponderá con su DNI mientras que la clave será autogenerada por el sistema.

RF (11).- Todos los usuarios pueden ver el listado de noticias en el panel de noticias.

RF (12).- Todo usuario tendrá acceso a un perfil dónde podrá modificar datos personales como correo, nombre y contraseña a excepción de su DNI.

Administrador

RF (13).- El administrador tiene acceso a secciones privadas de la aplicación, como la edición o el listado de los usuarios, clientes, material y proyectos que hay almacenados en el sistema.

RF (14).- El administrador puede acceder a las configuraciones del sistema, como dar visibilidad a determinados apartados de la interfaz mediante permisos.

RF (15).- El administrador puede crear, modificar y eliminar noticias del panel de noticias.

RF (16).- El administrador puede crear, modificar y eliminar usuarios del sistema.

RF (17).- El administrador puede crear, modificar y eliminar proyectos del sistema.

RF (18).- El administrador puede crear, modificar y eliminar clientes del sistema.

RF (19).- El administrador puede crear, modificar y eliminar material del apartado de material de oficina.

Empleado

RF (20).- Los empleados pueden pedir vacaciones.

RF (21).- Lo empleados pueden informar de sus bajas.

RF (22).- Los empleados pueden ver y modificar proyectos a los que estén asignados.

RF (23).- Los empleados pueden ver los detalles de los clientes con su información básica: correo, teléfono de contacto, empresa, proyectos, etc.

RF (24).- Los empleados pueden ver un listado del material disponible en oficina y la información de este además de información de cada uno de ellos.

4.3.Requisitos no funcionales

Los requisitos no funcionales se enfocan en las cualidades del software, tales como su rendimiento, seguridad, usabilidad y mantenibilidad. En resumen, establecen las características cualitativas del sistema. Los requisitos no funcionales de la aplicación serán los siguientes:

Interfaz y usabilidad

RNF (1).- La interfaz gráfica será sencilla, atractiva y fácil de manejar, basado en usuarios con unos conocimientos mínimos en Internet.

Mantenimiento y portabilidad

RNF (2).- El administrador tendrá la opción de poder mantener el sistema con actualizaciones de versiones con el fin de corregir errores y mejorar los servicios.

RNF (3).- La aplicación será privada, permitiendo al administrador el acceso al servidor.

Recursos

RNF (4).- La información de todos los usuarios de la aplicación se almacenará en una base de datos y se dispondrá de un módulo de gestión de empleados y administradores.

RNF (5).- La información de todos los proyectos y clientes de la aplicación se almacenará en una base de datos y se dispondrá de un módulo de gestión de proyectos y clientes.

Verificación y fiabilidad

RNF (6).- Para registrar un usuario, habrá que realizar un cuestionario con información obligatoria como el nombre de usuario, contraseña, correo electrónico, teléfono y tipo de contrato.

Rendimiento

RNF (7).- La aplicación permitirá el acceso a un usuario por dispositivo de forma concurrente.

RNF (8).- El tiempo de respuesta de la aplicación no deberá de exceder de treinta segundos.

Requisitos tecnológicos

RNF (9).- La aplicación usará diseño responsivo y será compatible con dispositivos móviles con conexión a Internet.

Habiendo definido los requisitos funcionales y no funcionales de la aplicación, es posible establecer los casos de uso que ilustran cómo estos requisitos se implementan en situaciones prácticas. A continuación, se detallan los casos de uso que ejemplifican las interacciones clave entre los usuarios y el sistema.

4.4.Casos de Uso

Un caso de uso es una descripción detallada de cómo un usuario interactúa con un sistema para realizar una tarea específica. Son una herramienta visual que ayuda a representar estas interacciones de manera clara y comprensible.

Se expondrán posteriormente los casos de uso más irrelevantes:

4.4.1.Casos de Uso Usuario

El siguiente caso de uso será universal para todo usuario registrado en la plataforma independientemente de los permisos o roles adjudicados a este.

Cada usuario deberá identificarse en el sistema para acceder a la página principal de la intranet. Una vez dentro podrá ver las notificaciones publicadas por los administradores. Además, podrá acceder a su perfil de usuario dónde se mostrará información relacionada con su usuario con la posibilidad de modificación.

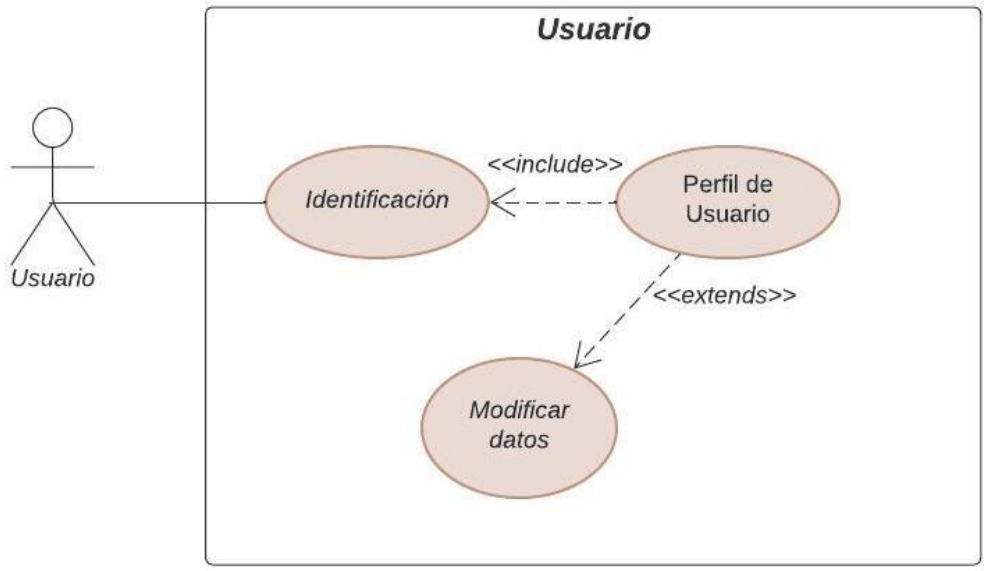

Figura 4.9. Diagrama de Casos de uso de Usuario. (Elaboración propia).

RF010. IDENTIFICACIÓN DE USUARIO			
ACTORES		Usuario	
OBJETIVO		Autentificar a un usuario	
DESCRIPCIÓN		El usuario se autentifica usando un nombre de usuario y una contraseña.	
PRECONDICIONES			
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Introduce su usuario y contraseña para acceder a la plataforma	**1**	Comprueba sus credenciales y los valida.
2		**2**	Redirecciona a la página principal.
DECISIONES ALTERNATIVAS			
Línea 1	Si el sistema invalida al usuario, notifica con un mensaje de error y no permite continuar a la página principal.		

Tabla 4.1. Descripción de acción identificación de usuario.

Escenario Previo 1: usuario accede a la plataforma de sesión desde un ordenador.

Un escenario posible para el caso de uso de identificación de usuario en la Tabla 1 es un usuario que acceda a la página de identificación de usuario desde un ordenador y rellene todos los campos con su usuario y contraseña. El resultado esperado es un inicio de sesión correcto y la redirección a la página de inicio. El sistema en caso de autentificación fallida mostrará al usuario una notificación de error indicando que las credenciales no son correctas.

RF011. VER NOTICIAS			
ACTORES	Usuario, Administrador		
OBJETIVO	Ver notificaciones		
DESCRIPCIÓN	El usuario dispondrá de un panel de noticias creadas por Administradores.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de noticias.
2		**2**	
DECISIONES ALTERNATIVAS			

Tabla 4.2. Descripción de la acción notificaciones.

RF012. PERFIL DE USUARIO			
ACTORES	Usuario		
OBJETIVO	Tener y ver perfil de usuario		
DESCRIPCIÓN	El usuario podrá acceder a un panel de perfil de usuario dónde podrá consultar información personal de contacto y poder modificarlos a excepción del DNI.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede a su perfil.	**2**	Muestra el perfil del usuario.
DECISIONES ALTERNATIVAS			
Línea 1	El usuario modifica los datos de contacto como el nombre, los apellidos, el correo, el teléfono y confirma los confirma los cambios y el sistema los guarda.		

Tabla 4.3. Descripción de la acción perfil de usuario.

Escenario Previo 1: el usuario desea cambiar la foto de su perfil de usuario.

El usuario con un perfil previamente creado ha accedido a la plataforma usando sus credenciales con intención de añadir una foto a su usuario. Desde el menú de navegación accede a su perfil opción ubicada en el nombre de su usuario. El sistema muestra una página con la información relacionada del usuario, como su nombre, su cargo, roles, etc. El usuario quiere cambiar su foto de perfil por lo que selecciona la opción de “Editar” dentro del perfil, es sistema posteriormente activará los campos que mostraban la información para que el usuario pueda modificarlos. El usuario cambiará la foto de perfil, guardará los cambios provocando que el sistema valide los datos y los almacene en la base de datos. El resultado sería el cambio de foto de usuario ubicada en el menú de navegación.

4.4.2.Casos de uso de empleado

El empleado podrá solicitar y ver información del estado de las solicitudes de sus vacaciones, informar de sus bajas además de ver un listado con la información de clientes, material y proyectos teniendo la posibilidad de modificar el estado de estos últimos.

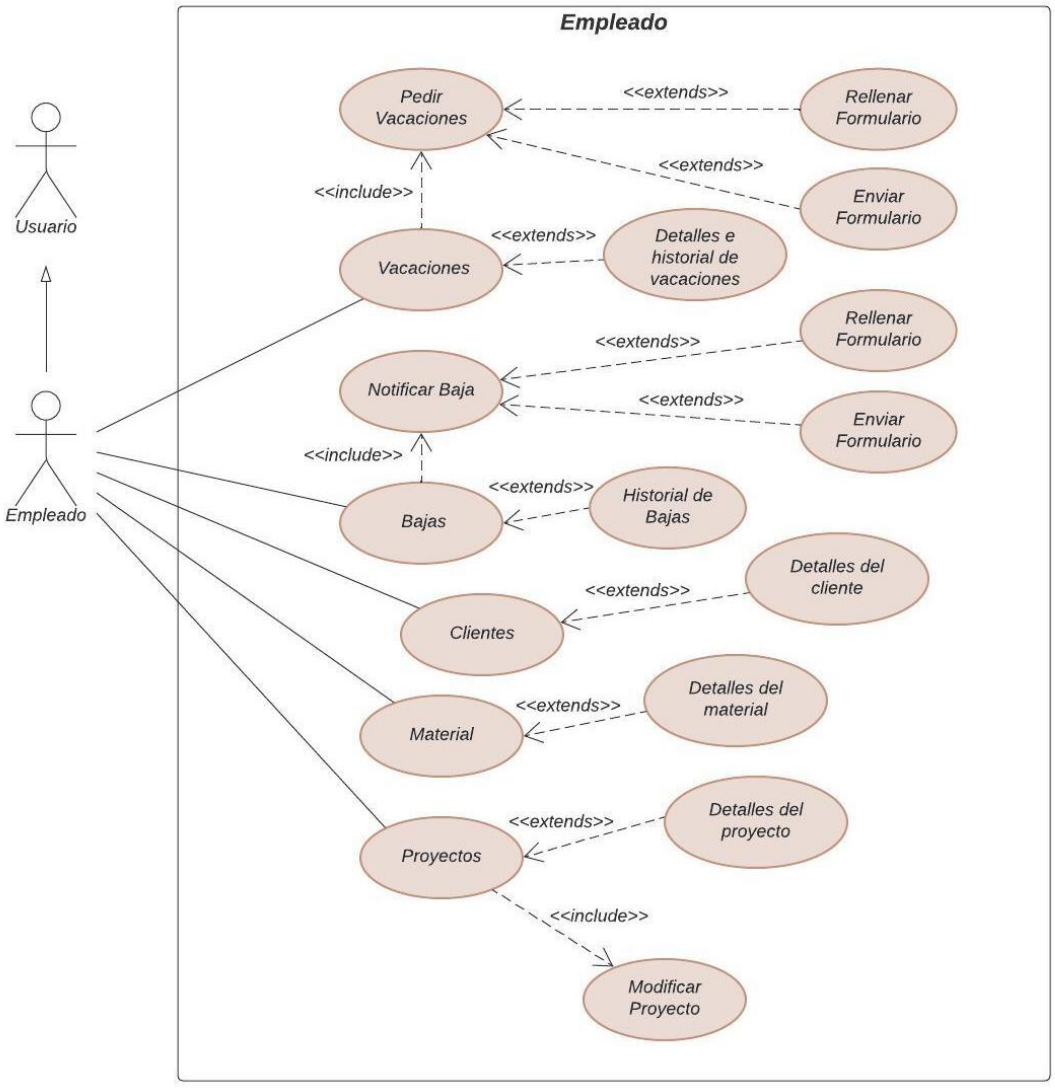

Figura 4.10. Diagrama de Casos de Uso de Empleado. (Elaboración propia).

RF021. PEDIR VACACIONES			
ACTORES	Empleado		
OBJETIVO	Solicitar vacaciones		
DESCRIPCIÓN	El empleado podrá solicitar vacaciones en el panel de vacaciones a espera de su validación.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al panel de vacaciones.	**2**	Muestra el panel de vacaciones.
3	Envía una solicitud de vacaciones rellenando la fecha deseada mediante un formulario.	**3**	
DECISIONES ALTERNATIVAS			
Línea 1	Si el administrador rechaza las vacaciones se enviará una notificación al empleado.		

Tabla 4.4. Descripción de la acción pedir vacaciones.

Escenario Previo 1: empleado solicita vacaciones.

Se aproxima las vacaciones de verano y los empleados han sido notificados para solicitar sus vacaciones con el fin de una mejor organización. El empleado verifica sus días libres restantes y evalúa su carga de trabajo para escoger el rango de días de vacaciones. Tras tomar la decisión, solicita las vacaciones en la opción “Nueva solicitud” dentro del módulo de vacaciones. El sistema le redirigirá a un formulario para rellenar las fechas del periodo vacacional, una vez el empleado guarde los campos, el sistema almacenará los datos y mandará una notificación al usuario con rol de supervisor.

Este escenario previo asegura que el proceso de solicitud de vacaciones se realiza de manera ordenada y conforme a las políticas de la empresa. Permite a los empleados planificar su tiempo libre de manera eficiente, mientras que los supervisores pueden gestionar la disponibilidad del equipo y asegurar que las operaciones no se vean interrumpidas.

RF022. INFORMAR DE LAS BAJAS			
ACTORES	Empleado		
OBJETIVO	Informar de las bajas		
DESCRIPCIÓN	El empleado podrá informar de sus bajas, incluyendo la causa y la fecha de duración de esta.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al panel de bajas.	**2**	Muestra el panel de bajas.
3	Envía un formulario con información de la baja como la causa y el tiempo de recuperación.	**3**	Recoge los datos del formulario y los almacena.
DECISIONES ALTERNATIVAS			

Tabla 4.5. Descripción de la acción informar de las bajas.

RF023. DETALLES DEL PROYECTO			
ACTORES	Empleado		
OBJETIVO	Ver los detalles de un proyecto		
DESCRIPCIÓN	El empleado podrá ver la información de los proyectos en los que esté involucrado. Además, podrá modificar algunos campos como el del estado de este o la fecha de finalización.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al listado de proyectos.	**2**	Muestra el listado de proyectos
3	Accede al proyecto deseado.	**3**	Muestra un panel con la información detallada del proyecto.
DECISIONES ALTERNATIVAS			
Línea 1	El empleado modifica los datos de contacto como el estado o fechas que indican etapas del proyecto, confirma los confirma los cambios y el sistema los guarda.		

Tabla 4.6. Descripción de la acción detalles del proyecto.

Escenario Previo 1: usuario busca información sobre un proyecto específico.

Un usuario pudiendo ser empleado o administrador, ha iniciado sesión en el sistema y necesita consultar información detallada sobre un material específico, como el cliente del proyecto o para verificar la información. El usuario accede al módulo de gestión de proyecto desde el menú principal del sistema y ubica el proyecto de interés para seleccionarlo. El resultado esperado es que el sistema redirija una página con

toda la información del proyecto como el nombre, la descripción, el presupuesto o las fechas en las que se llevaran a cabo el proyecto.

Escenario Previo 2: usuario sin permisos intenta acceder a los detalles del proyecto.

Un usuario con permisos limitados, ha iniciado sesión en el sistema intenta acceder a los detalles de un proyecto, pero no tiene permiso para ver información de los proyectos. Accede a la sección de proyectos desde el menú de navegación y el resultado esperado es que el sistema verifique los permisos y al detectar la falta de autorización muestre una notificación al usuario de la falta de permisos, denegando el acceso a los detalles del proyecto.

Estos escenarios previos son fundamentales para garantizar que el sistema maneje adecuadamente las situaciones que pueden ocurrir antes de mostrar los detalles de un proyecto.

RF024. DETALLES DEL CLIENTE			
ACTORES	Empleado		
OBJETIVO	Ver los detalles de un cliente		
DESCRIPCIÓN	El empleado podrá ver la información de los clientes.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al listado de clientes.	**2**	Muestra el listado de clientes
3	Accede al cliente deseado.	**3**	Muestra un panel con la información detallada del cliente.
DECISIONES ALTERNATIVAS			

Tabla 4.7. Descripción de la acción detalles del cliente.

RF025. DETALLES DEL MATERIAL			
ACTORES	Empleado		
OBJETIVO	Ver los detalles de un material		
DESCRIPCIÓN	El empleado podrá ver la información de los materiales.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al listado de materiales.	**2**	Muestra el listado de material
3	Accede al material deseado.	**3**	Muestra un panel con la información detallada del material.
DECISIONES ALTERNATIVAS			

Tabla 4.8. Detalles del material.

4.4.3.Casos de uso de Administrador

El administrador tendrá la opción de realizar todas las acciones del empleado junto con la facultad de publicar y eliminar notificaciones de la página principal, conceder y quitar permisos a cualquier usuario del sistema, aceptar o rechazar las vacaciones solicitadas por los usuarios y gestionar (crear, modificar y eliminar) usuarios, proyectos, materiales y clientes.

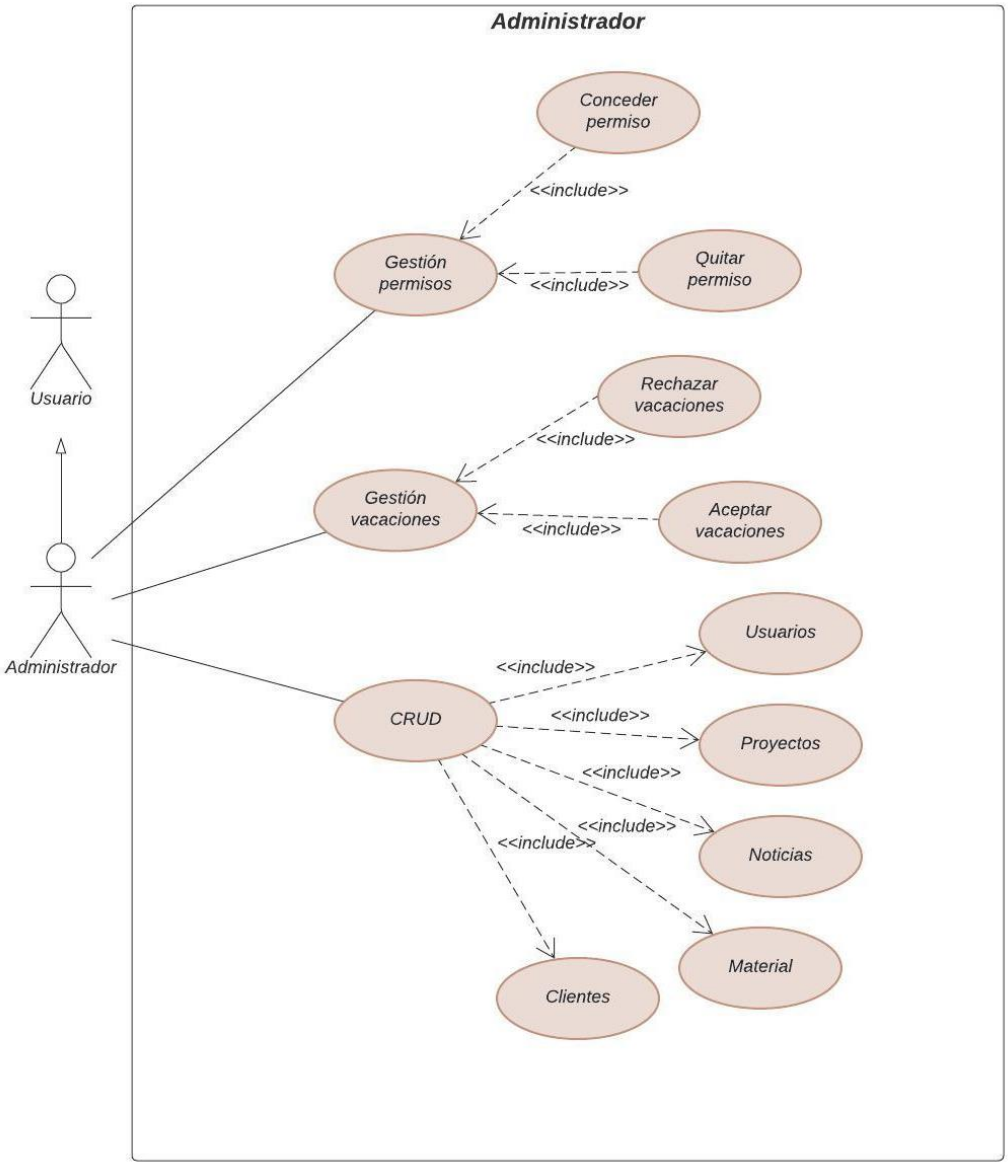

Figura 4.11. Diagramas de Casos de Uso de Administrador. (Elaboración propia).

RF015. CONCEDER PERMISOS			
ACTORES	Administrador		
OBJETIVO	Dar permisos a un usuario.		
DESCRIPCIÓN	El administrador podrá conceder uno o más permisos de acceso o acciones a otros usuarios fuera de los permisos del rol de este.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al panel de permisos.	**2**	Muestra el panel de permisos.
3	Rellena el formulario con los permisos a conceder y se confirman los cambios.	**3**	Guarda los cambios de permisos.
DECISIONES ALTERNATIVAS			

Tabla 4.9. Descripción de la acción conceder permisos

Escenario Previo 1: un nuevo empleado necesita acceso a funciones específicas del sistema.

Un nuevo empleado le han asignado tareas dónde necesita crear proyectos. El rol asignado para el empleado es de gestor de proyectos que da acceso al módulo de proyectos, pero no a la creación de estos. Al iniciar sesión por primera vez, intenta crear, pero no encuentra opciones para ello. Por lo que a través de comunicación directa solicita al administrador del sistema dichos permisos. El administrador en el panel de administrador selecciona el rol gestor de proyectos y activa el permiso para crear proyectos y mostrar sus detalles. El empleado verifica el acceso y comienza el desarrollo de sus tareas (ver Tabla 4.9).

Este escenario previo asegura que los empleados tienen los permisos adecuados para realizar su trabajo desde el primer día, facilitando su integración en la empresa y asegurando que el sistema se utiliza de manera segura y eficiente.

RF016. CRUD DE NOTICIAS			
ACTORES	Administrador		
OBJETIVO	Poder crear, modificar y eliminar notificaciones.		
DESCRIPCIÓN	El administrador podrá crear, modificar y eliminar notificaciones del panel de notificaciones de la página principal.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal con el panel de notificaciones.
2	Accede al panel de notificaciones y	**2**	Muestra el panel de notificaciones con las diversas opciones de crear, modificar o eliminar.
3	Escoge una opción entre crear, modificar o eliminar una notificación y pide la ejecución de esta.	**3**	Ejecuta la opción escogida y muestra los cambios.
DECISIONES ALTERNATIVAS			

Tabla 4.10. Descripción de la acción crear, mostrar, actualizar y eliminar notificaciones.

Escenario Previo 1: administrador crea una noticia.

Como condición inicial tenemos al administrador, un usuario autorizado que ha iniciado sesión en el sistema con permisos para crear y gestionar noticias. El administrador ha accedido al panel de noticias de la aplicación, dónde se gestionan las noticias. El administrador escoge crear noticias y seleccione la opción “Crear noticia” desde el mismo apartado. El sistema redirigirá a un formulario con campos obligatorios como título, cuerpo del texto entre otras opciones. El resultado esperado es que el redactor rellene los campos y los publique. El sistema deberá redirigir al panel de noticias y mostrar la noticia creada.

Escenario Previo 2: administrador necesita actualizar una noticia antigua.

El administrador ha encontrado una errata en una noticia ya publicada y necesita modificar la noticia para corregirla por lo que accede al panel de noticias y localiza la noticia a actualizar, y selecciona “Editar”. El sistema redirigirá a un formulario con los campos ya rellenos con la información de la noticia seleccionada, seguidamente el administrador editará la errata en el campo correspondiente y los guardará. El resultado esperado es que el sistema almacene los datos, redirija al usuario al panel de noticias y muestre la edición correctamente.

Escenario Previo 3: administrador necesita eliminar una noticia.

El administrador ha recibido la tarea de eliminar una noticia ya publicada por lo que localiza la noticia, y selecciona “Eliminar”. El sistema mostrará un aviso al administrador para que confirme la eliminación de la noticia, el administrados confirma la eliminación. El resultado esperado sería el sistema eliminado la noticia del panel de noticias.

Estos escenarios previos son fundamentales para garantizar que el sistema maneje adecuadamente diversas situaciones que pueden ocurrir antes de la creación de una noticia (ver Tabla 4.10).

RF017. CRUD DE USUARIOS			
ACTORES	Administrador		
OBJETIVO	Poder crear, modificar y eliminar usuarios.		
DESCRIPCIÓN	El administrador podrá crear, modificar y eliminar usuarios desde el listado de usuarios.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal.
2	Accede al listado de usuarios.	**2**	Muestra el listado de usuarios con las diversas opciones de crear, modificar o eliminar.
3	Escoge una opción entre crear, modificar o eliminar un usuario y solicita la ejecución de esta.	**3**	Ejecuta la opción escogida y muestra los cambios.
DECISIONES ALTERNATIVAS			

Tabla 4.11. Descripción de la acción crear, mostrar, actualizar y eliminar usuarios.

RF018. CRUD DE CLIENTES			
ACTORES	Administrador		
OBJETIVO	Poder crear, modificar y eliminar clientes.		
DESCRIPCIÓN	El administrador podrá crear, modificar y eliminar clientes desde el listado de clientes.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal.
2	Accede al listado de clientes.	**2**	Muestra el listado de clientes con las diversas opciones de crear, modificar o eliminar.
3	Escoge una opción entre crear, modificar o eliminar un cliente y solicita la ejecución de esta.	**3**	Ejecuta la opción escogida y muestra los cambios.
DECISIONES ALTERNATIVAS			

Tabla 4.12. Descripción de la acción crear, mostrar, actualizar y eliminar clientes.

RF019. CRUD DE PROYECTOS			
ACTORES	Administrador		
OBJETIVO	Poder crear, modificar y eliminar proyectos.		
DESCRIPCIÓN	El administrador podrá crear, modificar y eliminar proyectos desde el listado de proyectos.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal.
2	Accede al listado de proyectos.	**2**	Muestra el listado de proyectos con las diversas opciones de crear, modificar o eliminar.
3	Escoge una opción entre crear, modificar o eliminar un proyecto y solicita la ejecución de esta.	**3**	Ejecuta la opción escogida y muestra los cambios.
DECISIONES ALTERNATIVAS			

Tabla 4.13. Descripción de la acción crear, mostrar, actualizar y eliminar proyectos.

RF020. CRUD DE MATERIAL			
ACTORES	Administrador		
OBJETIVO	Poder crear, modificar y eliminar materiales.		
DESCRIPCIÓN	El administrador podrá crear, modificar y eliminar materiales desde el listado de materiales.		
PRECONDICIONES	Haberse identificado en el sistema con éxito.		
POSTCONDICIONES			
FLUJO BÁSICO			
ACCIONES USUARIO		**ACCIONES SISTEMA**	
1	Inicia sesión en la plataforma.	**1**	Muestra la página principal.
2	Accede al listado de materiales.	**2**	Muestra el listado de materiales con las diversas opciones de crear, modificar o eliminar.
3	Escoge una opción entre crear, modificar o eliminar un proyecto y solicita la ejecución de esta.	**3**	Ejecuta la opción escogida y muestra los cambios.
DECISIONES ALTERNATIVAS			

Tabla 4.14. Descripción de la acción crear, mostrar, actualizar y eliminar materiales.

4.5.Diseño lógico de datos

Al igual que para construir un edificio se necesitan planos que contengan todo lo ineludible en la construcción, el diseño e implementación de cualquier aplicación es equiparable. Definir la estructura de la base de datos es una tarea imprescindible a la hora de diseñar una aplicación para garantizar la integridad y eficiencia de los datos.

A continuación, se indagará en el diseño de las tablas que definen las entidades junto con sus características y relaciones que conformarán el modelo de datos del sistema.

4.5.1.Tablas base de datos

Para representar los elementos que conforma la aplicación es necesario diseñar la estructura de forma que se representen los datos y la interacción entre ellos.

En el diagrama de clases expuesto en la Figura 4.12, más adelante, se presentan las distintas clases y relaciones:

Usuario

Representa a la persona física que interactúa con la aplicación. Dispondrá de varios atributos identificativos dónde id tiene como motivo el diferenciar cada objeto de esta clase de forma única en la base de datos, usuario junto con una password para identificar dentro de la aplicación además del inicio de sesión, nombre real del usuario y por último el correo con el fin de notificar al usuario a este desde la aplicación.

Proyecto

La clase proyecto establecerá los proyectos que representan las actividades de la empresa. Para el proyecto es necesario un id y un nombre con fin identificativo, una descripción de este, el presupuesto para representar su valor económico además de fechas de inicio y fin para la planificación del proyecto. Serán necesarias algunas

relaciones con otras clases como usuario para mostrar el empleado responsable del proyecto y otra con cliente para el cliente al que se le proporcionará el proyecto.

Cliente

Simboliza los clientes cuyo interés es buscar soluciones en las actividades que ejerce la empresa. Un cliente requiere de un id identificativo, la empresa que representan, un correo para notificaciones, teléfono y dirección de contacto junto con un CIF y un DNI para gestiones de facturación y contabilidad fuera de la aplicación.

Noticias

Esta clase expondrá una sección de noticias con motivo de informar a todos los usuarios. Serán necesarios un título y una descripción.

Vacaciones y Bajas

Las dos clases serán los elementos que permitirán al usuario solicitar vacaciones o informar de las bajas a través de la aplicación.

Las bajas sólo necesitarán de las fechas de inicio y fin en las que se ha dado de baja el empleado y las vacaciones las fechas de inicio y fin en las que el empleado usará sus días libres. Particularmente las vacaciones harán uso de un estado en el que se muestre al empleado si se han aceptado o denegado por parte de recursos humanos su solicitud.

Tanto vacaciones como bajas requerirán de una relación con usuario para que el usuario que las gestione ubique a los empleados.

Material

El material hará referencia a toda herramienta o dispositivo que se use durante la actividad empresarial. Cada material necesitará de un nombre y descripción, la cantidad existente en la empresa, además de un estado que representará su disponibilidad de este.

Por otro lado, se hará uso de una relación con la tabla usuario para reflejar el empleado que esté haciendo uso de dicho material.

Permiso y rol

Las entidades permiso y rol se usarán para el manejo de restricciones de acceso a áreas y algunas acciones que irán destinadas sólo para el rol de administrador.

Para ello permiso necesitará un nombre con el que identificar el permiso y un slug para facilitar su implementación. Por la parte de rol sólo hará falta un nombre identificativo.

Permiso_Rol, Usuario_Rol, Menu_Rol

Permiso_rol, usuario_rol y menu_rol serán clases destinadas a crear un puente entre las dos clases al que hacen mención, necesitando ambas solo un atributo que haga de relación entre ambas entidades.

Menu

Finalmente, la clase menú junto con la entidad de apoyo menu_rol irá dirigida a gestionar parte del front con el fin de invalidar el acceso a menús con acciones dirigidas sólo a administrador. Para identificarlo se usará un id y un nombre, una URL para el enrutamiento de cada apartado del menú, orden que definirá el orden de aparición de cada apartado en menú y un submenú, si existe, junto con un icono representativo del apartado que se visualizará en dicho menú.

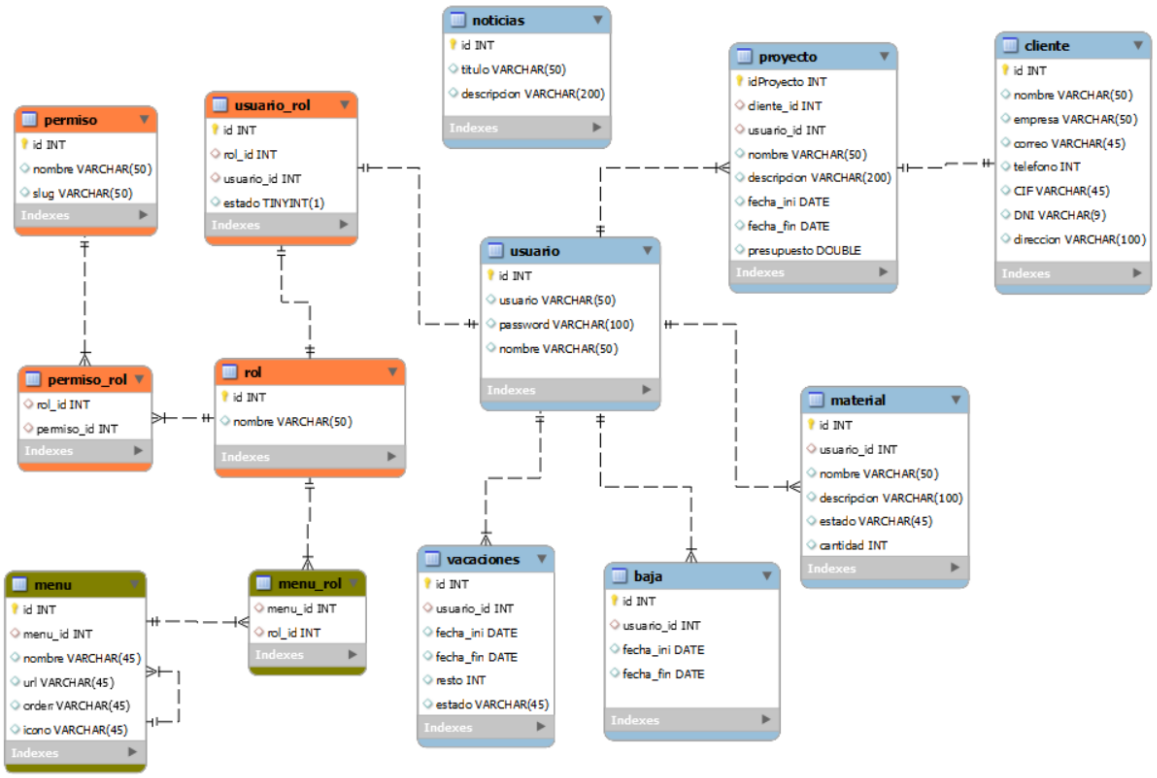

Figura 4.12. Modelo de datos. (Elaboración propia).

Como se ve en la Figura 4.12, las tablas representan cada una de las entidades del mundo real con un papel importante en las actividades de la empresa.

Cada una de las tablas contiene campos con valores que se necesitan almacenar además de una clave primaria, representada por una llave, que identifican a la entidad como única. Las claves foráneas se representan con un rombo rojo e indican las relaciones entre las entidades. Muchas de las relaciones son de uno a uno como usuario con usuario_rol y otras de uno a muchos como usuario a vacaciones.

Un modelo de datos lógico es una herramienta rentable ya que simplifica el traslado de la información a la base de datos debido a que su representación gráfica es muy similar a las tablas de un servidor SQL.

4.5.2.Clases de Datos

Una vez definidas las clases, sus atributos y relaciones en el apartado anterior, el siguiente objetivo concuerda con la representación de las clases de datos correspondiendo con los modelos de entidades del ORM de Laravel, que interactuarán con las tablas de la base de datos.

Los modelos definen los objetos de cada una de las entidades de la aplicación junto con sus funcionalidades básica. Algunas operaciones como `getInfo()` son destinadas a obtener datos sobre los atributos, otras como `getVacaciones()` del modelo Usuario manejan datos sobre la relación de *Usuario* con *Vacaciones* devolviendo información respectiva a las vacaciones de este usuario y por último funciones como `calculaResto()` de *Vacaciones* cuyo provecho persiste en la gestión de datos circunstancial.

El la Figura 4.13 se muestran los modelos más importantes, cada modelo tiene la responsabilidad de acceder a los datos y modificarlos manteniendo la lógica de negocio.

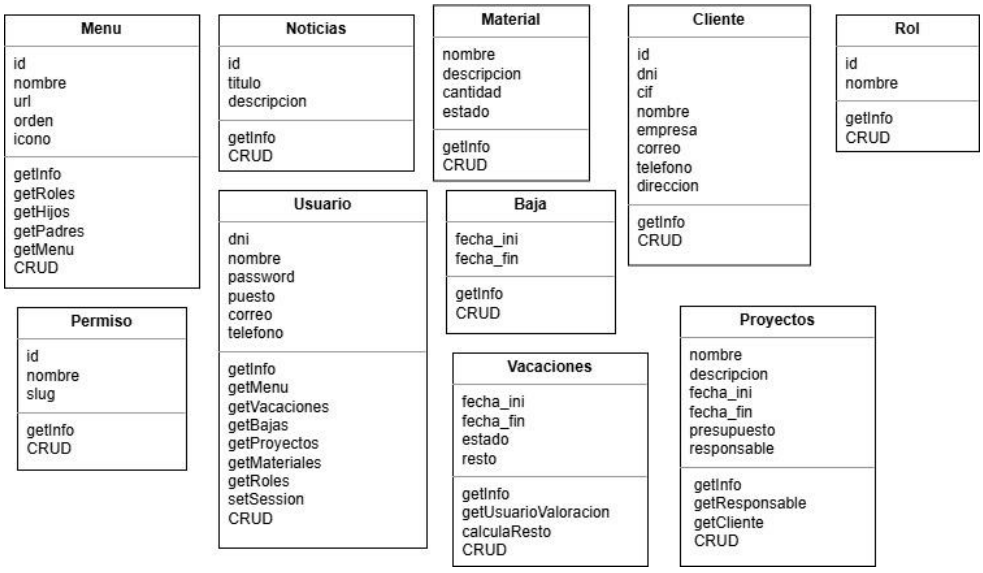

Figura 4.13. Clases de datos. (Elaboración propia).

En Laravel el ORM que controla los modelos es Eloquent, esto quiere decir que los modelos se implementan como datos en una base de datos siendo un modelo asociado a una entidad y por ende a una tabla de la base de datos.

Con los objetos y sus funciones representados el siguiente paso es manejar las solicitudes que llegan desde la interfaz por parte de los usuarios como son las peticiones CRUD (Create Read Update Delete) que permiten crear, mostrar, actualizar y eliminar registros desde los controladores.

4.5.3.Clases del Controlador

Los controladores se encargarán de manejar las peticiones producidas desde la interfaz de usuario usando los modelos además de enviar datos desde estos para su correspondiente visualización en la aplicación. Es por esta causa por la que cada modelo definido necesita un controlador propio.

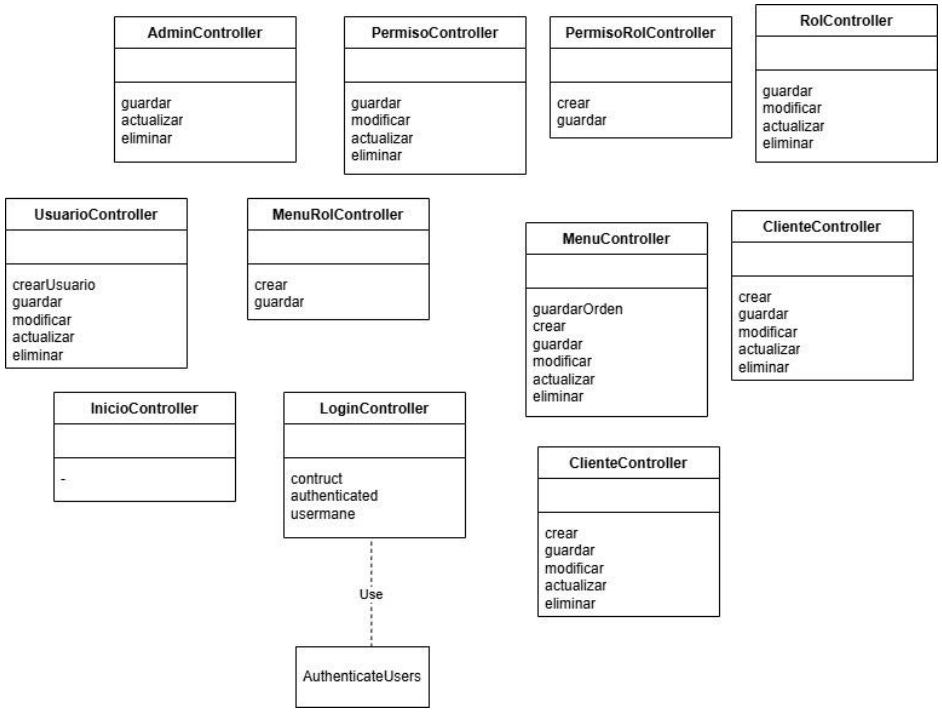

Figura 4.14. Controladores. (Elaboración propia).

Aunque se puede ver en la figura 18 que cada controlador contiene las operaciones de CRUD existen algunos como *PermisoRolController* y *MenuRolController* que sólo incluyen crear y guardar a causa de únicamente gestionar las relaciones entre dos clases.

Todos los controladores coinciden con sus modelos respectivos en la Figura 4.14, exceptuando algunas como el *LoginController* que hace uso de la clase *AuthenticateUsers* para gestionar los datos en la Autentificación del usuario a la hora de iniciar sesión en el sistema o el *InicioController* para mostrar la vista principal de la aplicación.

En el patrón MVC, el modelo y el controlador conforman la lógica de la aplicación y las vistas, la lógica de representación. Mientras que los controladores reciben solicitudes y las gestionan accediendo a los datos de los modelos, que obtienen desde la base de datos, las vistas se encargan de sintetizar el código HTML sin que los controladores y modelos lo contengan, centrándose únicamente en la representación de los datos.

4.5.4.Clases de Servicios

Las vistas son las encargadas de visualizar todos los cambios lógicos producidos en la base de datos desde los modelos. Los controladores son los que proveen a las vistas de la información de los datos a visualizar.

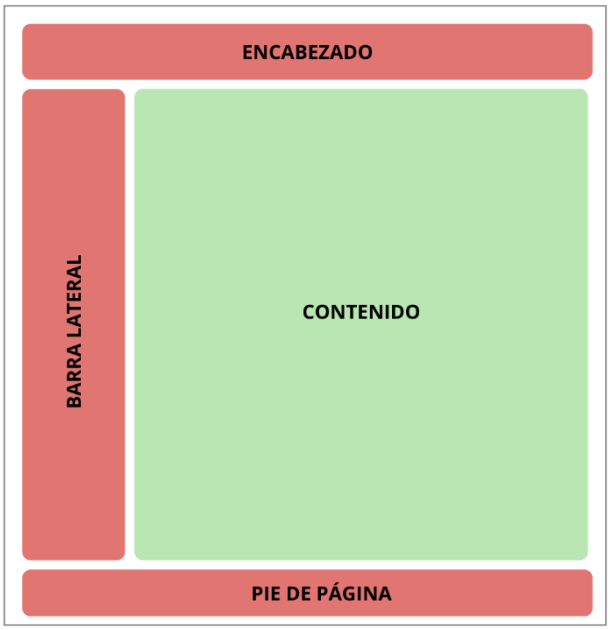

Figura 4.15. Estructura web. (Elaboración propia).

Toda página web debe seguir una estructuración que reflejará el posicionamiento de los elementos que compongan la interfaz de la aplicación. El conjunto de vistas *Theme*, mostrará todos los elementos necesarios fijos durante toda la navegación web; encabezado, barra lateral y pie de página. De este modo define la disposición de todas las páginas tal cual se muestra en la Figura 4.15.

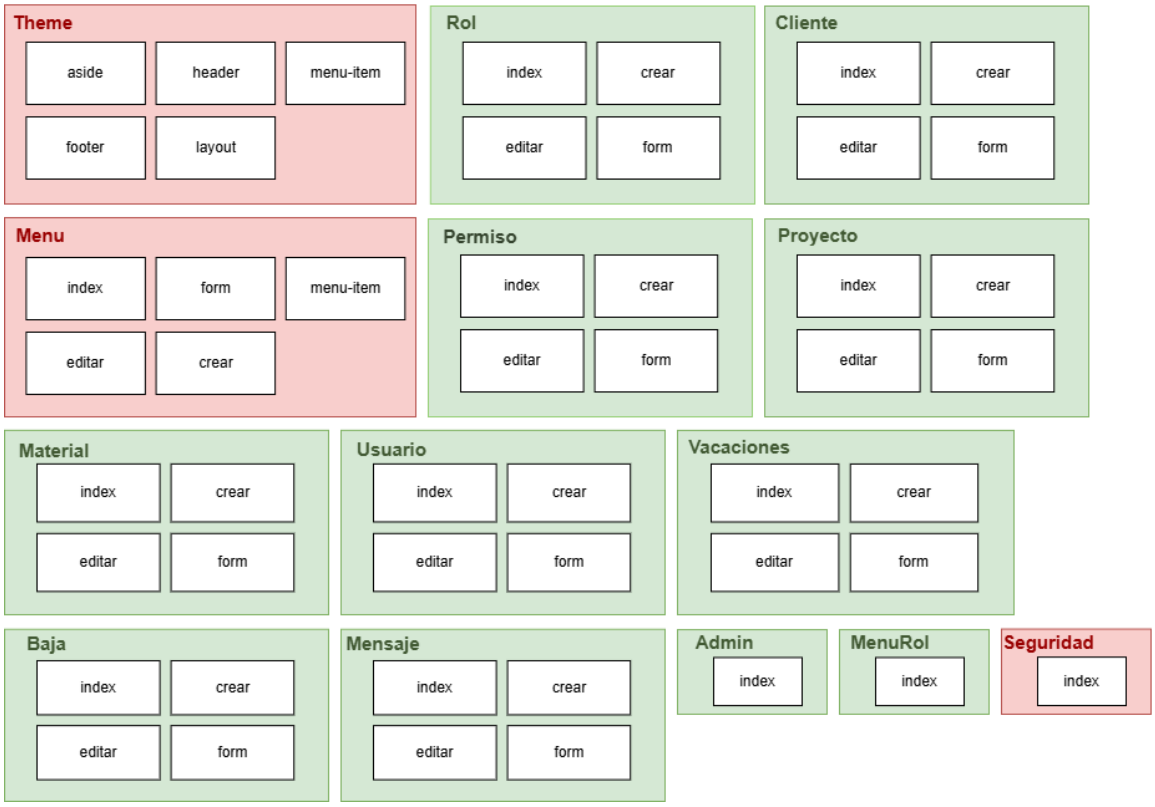

Figura 4.16. Vistas. (Elaboración propia).

En la Figura 4.16, se encuentran todas las vistas de la aplicación. Las vistas index son las vistas principales de cada entidad, algunas visualizan un resumen de los atributos de todas las entidades existentes de un mismo modelo almacenadas en la base de datos de forma organizada en una tabla como *Material*, *Proyectos*, *Cliente*, *Vacaciones*, *Baja*, *Usuario*, *Permiso* y *Rol* componiendo el contenido de una página web.

Las vistas form serán auxiliares que recogerán los formularios usados en las vistas crear y editar. Estas vistas tendrán como función disponer los formularios necesarios para que el usuario pueda introducir datos con el fin de crear una nueva entidad o editar una ya existente.

Menu, además de tener vistas como las anteriores descritas, contiene menu-item que representará las secciones accesibles de este dentro de la barra lateral izquierda de la web.

Por último, *Seguridad* sólo dispondrá de index para presentar la vista de inicio de sesión.

4.5.5.Diagrama UML

Una vez definido el modelo lógico de datos, es hora de especificar a un escenario dirigido a la implementación. La representación visual de la Figura 4.17 presenta una especificación más completa de los diagramas antes vistos, desvelando algunas funciones de la implementación e información detallada de las relaciones entre clases.

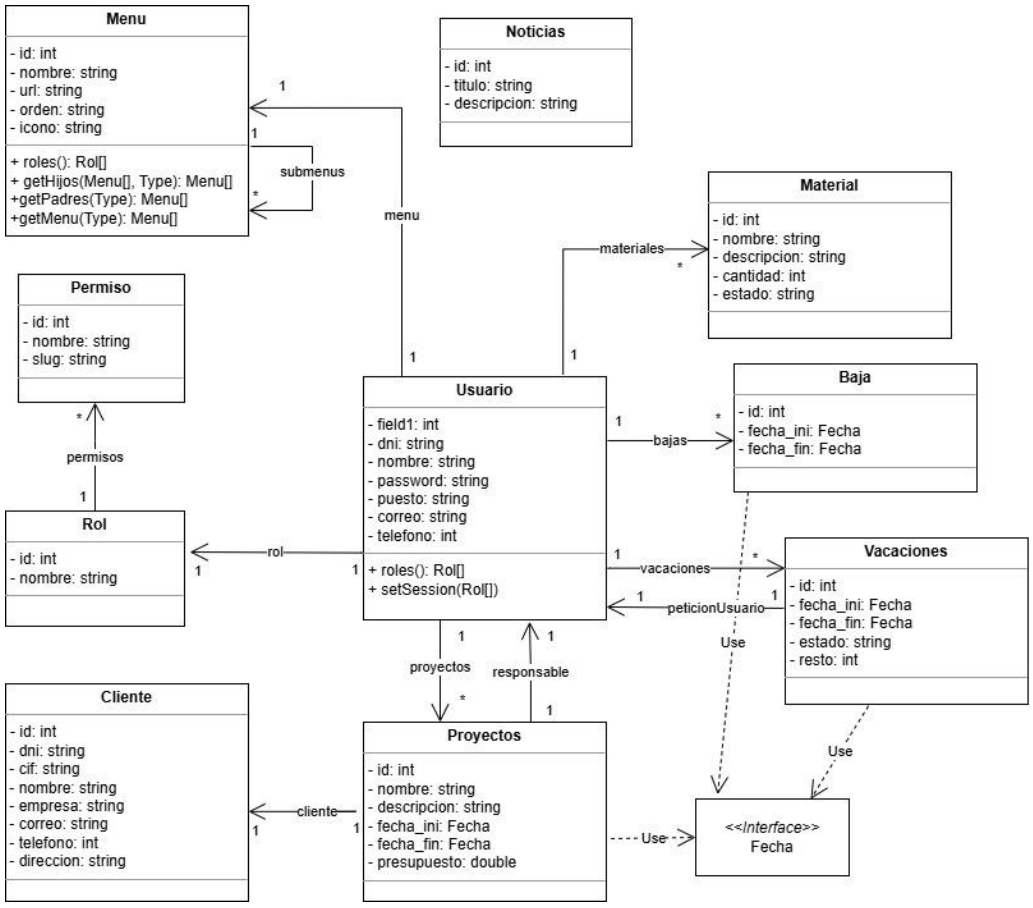

Figura 4.17. UML. (Elaboración propia).

A continuación, una breve explicación de cada componente del diagrama, sus atributos, métodos y relaciones:

Clase Menú

Se visualizará en la barra lateral de la web. Cada menú puede contener a su vez otro menú siendo submenú de forma recursiva. Esta relación se implementará haciendo uso de las funciones `getHijos()` que devolverá los submenús y `getPadres()` los menús principales. El método `getMenu()` recogerá los menús y submenús para pasarlos al controlador y mostrarlos en la vista.

Clase Noticias

Almacena las noticias que se podrán crear desde la aplicación. Las noticias podrán crearse, ser editadas o eliminadas únicamente por aquellos usuarios que tengan el rol de gestor y mostradas para todos los usuarios.

Clase Material

Cada material sólo podrá ser creado, editado y eliminado por el rol de gestor. El atributo de cantidad indicará cuantos materiales hay existentes y el estado la disponibilidad de estos como en uso o disponible. Si el material se encuentra en uso debe de tener un usuario asociado mediante la relación de materiales que se mostrará en la vista.

Clase Baja y Vacaciones

La clase Baja contendrá las fechas de inicio y de fin para mostrarlas en la vista correspondiente con objetivo informativo.

Vacaciones contendrá los mismos atributos que *Baja* con la extensión del atributo estado debido a que las vacaciones se pedirán como una solicitud por parte del usuario desde la aplicación, cuando se hace una solicitud el estado pasa a "*En revisión*". Las solicitudes mandadas serán notificadas al gestor de la aplicación que deberá responder cambiando el estado de cada solicitud a "*Rechazadas*" en caso de rechazarlas o "*Aceptadas*". El usuario sin el rol de gestor sólo podrá crear y ver las solicitudes, pero no podrá modificarlas ni eliminarlas.

Si las solicitudes se aceptan, la función `calculaResto()` se encargará de calcular los días libres restantes que le queda al usuario solicitante a partir de la diferencia de fechas solicitadas y el número de días libres anuales.

Clase Proyectos

Los proyectos representan las peticiones de los clientes a la empresa aceptadas bajo presupuesto. Crear, modificar y eliminar un proyecto será una acción limitada a los gestores. Se mostrarán unas fechas designadas y el presupuesto como valor económico de este. Al ser una petición de un cliente, también deberá mostrar el nombre de este y el nombre del trabajador que se encargará de ejecutar el proyecto.

Clase Cliente

La clase cliente almacena todos los datos significativos de los clientes con motivo de agilizar la actividad empresarial que los involucre. Es por esto que deberá conglomerar todos los atributos identificativos y de contacto como el DNI, CIF, empresa, nombre, correo, teléfono y dirección física de la empresa.

Clase Permiso y Rol

El usuario estándar no tendrá consciencia de la existencia de los permisos y los roles debido a que toda operación CRUD sobre estos estará reservada para el rol de administrador.

Cada rol podrá ser asignado a cualquier usuario, dependiendo del rol el usuario podrá acceder o no a las secciones o contenido de la web. Por ejemplo, a un usuario con rol de gestor no se le mostrarán algunas secciones del menú como las vistas de *Permiso-Rol* o la de *Usuarios*. En el caso que se quiera acceder mediante la ruta en el navegador la aplicación notificará un aviso de acceso restringido.

Por otro lado, cada rol tendrá asociado una serie de permisos que posibilitarán ciertas acciones como modificar campos de vistas de *Proyecto* o *Material*.

Clase Usuario

La clase usuario representará a todo empleado de la empresa que interactúe con la aplicación. Para acceder a la aplicación deberá iniciar sesión con su usuario y contraseña. El DNI y nombre son identificativos, correo y teléfono atributos de contacto y el puesto será el cargo del empleado dentro de la empresa. Todos los atributos de todos los usuarios podrán mostrarse desde la sección de usuarios para aquellos que tengan el rol administrador, con la posibilidad para este último de crear, modificar y eliminar cualquier usuario.

En otro orden de ideas, cada usuario podrá ver su información desde una vista perfil en la que podrán modificar los campos de contacto.

Finalmente, la función roles devolverá todos los roles asignados a usuario y `setSession()` introducirá todos los datos de los roles asociados de este a la sesión, para gestionar los accesos a áreas restringidas.

Es de suma importancia comprender los propósitos principales de la aplicación y como resolverlos, considerar qué arquetipo de usuarios finales y las funciones requeridas que convergen en la necesidad de analizar los requisitos funcionales y no funcionales de esta con el fin de cumplir con los criterios de usabilidad, rendimiento y seguridad.

5. IMPLEMENTACIÓN

La fase de implementación es una etapa crucial en el desarrollo de cualquier proyecto de software, ya que en ella se materializan las ideas y diseños previamente elaborados. En este apartado, se detalla el proceso de conversión de los requisitos y modelos conceptuales en un sistema funcional, haciendo uso de las tecnologías y herramientas seleccionadas durante las etapas iniciales del proyecto.

Además, se explicará cómo cada módulo importante del sistema ha sido desarrollado e integrado, poniendo especial énfasis en la calidad del software, la modularidad y la capacidad de mantenimiento a largo plazo.

5.1.Requisitos

Para la implementación del proyecto se necesitan las siguientes tecnologías:

- Laragon o cualquier entorno de desarrollo rápido.
- Servidor web como Apache.
- PHP versión 8.1 con los paquetes curl,cli, fpm, -mysql, mbstring y sus dependencias.
- Firewall.
- Composer.
- Git.
- MariaDB.
- MySQLSecure.
- ModReWrite Apache.
- Laravel 11.
- Bootstrap 5.
- HTML 5, CSS, JavaScript y AJAX.

5.2.Estructura del proyecto

En este apartado se ha descrito brevemente la funcionalidad de los directorios (ver Tabla 5.16) más esenciales de Laravel.

Directorio	Descripción
App/Models	Contiene todos los modelos creados.
App/Http/Controllers	Contiene todos los controladores creados.
App/Providers	Archivos que proveen servicios o funcionalidades adicionales al framework
App/Rules	Contiene los archivos que especifican instrucciones para filtrar los datos de los formularios.
Database/migrations	Archivos que carga la base de datos con datos predefinidos automáticamente.
Database/seeders	Archivos que definen una serie de entidades para establecerlas directamente en la base de datos.
Resources/views	Contiene todas las vistas creadas.
Public/assets	Recursos necesarios para la funcionalidad y diseño de la aplicación como archivos que usan AJAX, JavaScript, JQuery y CSS.
Routes	Archivos dónde se definen las rutas de navegación internas de Laravel que serán llamadas en las vistas mediante Eloquent.
vendor	Archivos dependencia necesarios para la ejecución de Laravel.

Tabla 5.15. Directorios de Laravel.

Laravel organiza el código fuente de la aplicación en una estructura de directorios clara y bien definida, cada uno con una responsabilidad específica. Esta organización no solo facilita la navegación y el mantenimiento del proyecto, sino que también promueve el uso de buenas prácticas de desarrollo al segregar las distintas funcionalidades de la aplicación.

5.3.Migraciones

Dentro de esta estructura organizada, el directorio *database/migrations* juega un papel crucial en la gestión de la base de datos. Las migraciones en Laravel funcionan como un sistema de control de versiones para la base de datos, permitiendo definir y compartir de manera eficiente la estructura del esquema de la base de datos de la aplicación. En lugar de redactar manualmente las consultas SQL para crear, modificar o eliminar tablas y columnas, las migraciones permiten describir estos cambios mediante archivos de código PHP, lo que facilita el seguimiento de las modificaciones y la colaboración en equipo.

Las migraciones se generan utilizando la herramienta de línea de comandos Artisan a través del comando **php artisan make:migration create_rol_table**, y se almacenan en el directorio *database/migrations* (ver Figura 5.18).

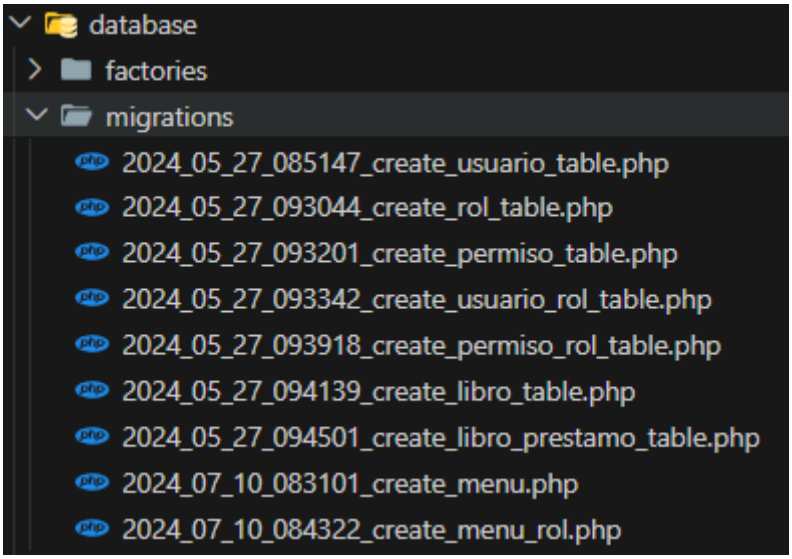

Figura 5.18. Directorio database/migrations. (Elaboración propia).

Dentro del archivo, se usan las clases Schema y Blueprint para definir la estructura de la base de datos. A través de ellos se pueden definir tablas, columnas, índices, relaciones y demás aspectos de esta. El método `create()` de Schema recoge el nombre de la tabla de la base de datos a manipular y otro objeto de la clase Blueprint que contendrá los métodos que definirán las columnas de la tabla.

Una vez definida la tabla se podrá ejecutar la migración usando el comando **php artisan migrate**, que ejecutará todas las migraciones realizadas y pendientes de ejecutar y crearán las tablas en la base de datos.

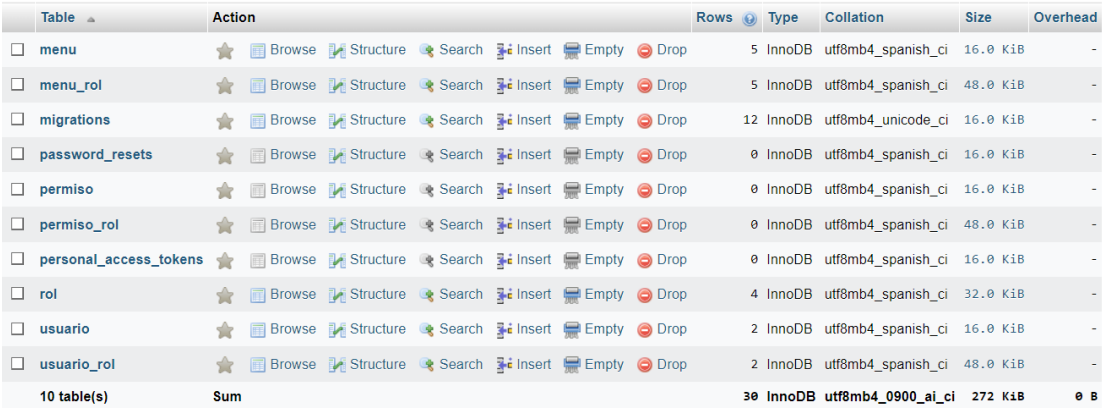

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	menu	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_spanish_ci	16.0 KiB	-
☐	menu_rol	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_spanish_ci	48.0 KiB	-
☐	migrations	Browse Structure Search Insert Empty Drop	12	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
☐	password_resets	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_spanish_ci	16.0 KiB	-
☐	permiso	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_spanish_ci	16.0 KiB	-
☐	permiso_rol	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_spanish_ci	48.0 KiB	-
☐	personal_access_tokens	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_spanish_ci	16.0 KiB	-
☐	rol	Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_spanish_ci	32.0 KiB	-
☐	usuario	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_spanish_ci	16.0 KiB	-
☐	usuario_rol	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_spanish_ci	48.0 KiB	-
	10 table(s)	Sum	30	InnoDB	utf8mb4_0900_ai_ci	272 KiB	0 B

Figura 5.19. Tablas de las bases de datos. (Elaboración propia).

En la Figura 5.19 se presentan todas las tablas que han sido creadas mediante migraciones para el proyecto. Con la base de datos configurada, el siguiente paso es poblar estas columnas. Laravel proporciona la capacidad de rellenarlas automáticamente utilizando los seeders.

5.4.Seeders

Los seeders en Laravel son clases diseñadas para poblar la base de datos con datos iniciales. Estas clases resultan especialmente útiles para precargar la aplicación con información de ejemplo, datos de prueba o incluso datos de producción.

Para crear un nuevo seeder, utiliza el comando Artisan: **php artisan make:seeder EjemploSeeder**, el cual generará el archivo EjemploSeeder en el directorio *database/seeders* (ver Figura 5.20).

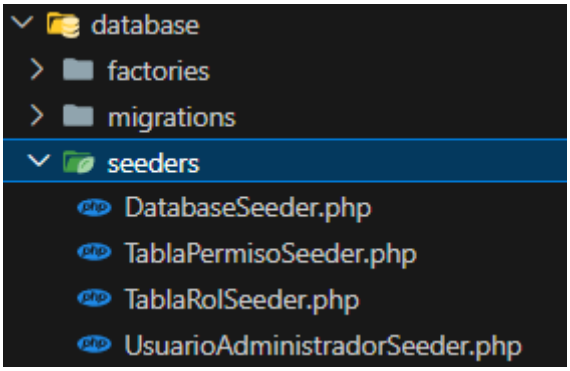

Figura 5.20. Directorio database/seeders. (Elaboración propia).

Cada seeder contiene un método `run()`, que es ejecutado a través del comando de Artisan **php artisan db:seed**. En este método, se definen las columnas de las tablas de la base de datos utilizando constructores de consultas a través de la clase DB, un facade clase de Laravel que facilita la ejecución de consultas a la base de datos mediante una sintaxis simplificada.

Una vez ejecutadas las migraciones y posteriormente los seeders, las tablas de la base de datos estarán creadas y pobladas. El siguiente paso consistirá en desarrollar los modelos, controladores y vistas necesarios para gestionar estos datos.

5.5.Modelos

Son la representación de las tablas de tu base de datos dentro de tu aplicación. Son la columna vertebral de la interacción entre tu código PHP y la información almacenada en la base de datos.

Para crear de forma automática la estructura de todos los modelos se ha usado el Artisan CLI **php artisan make:model** *Rol* en el directorio *app/Models* (ver Figura 5.21).

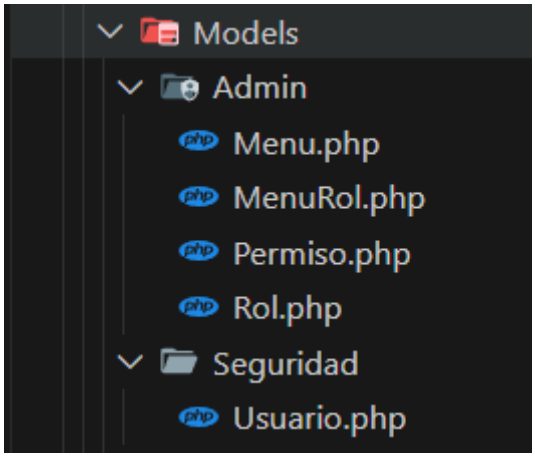

Figura 5.21. Directorio app/Models. (Elaboración propia).

Cada modelo corresponde a una tabla en tu base de datos y cada instancia de un modelo representa una fila de esa tabla.

Los modelos pueden crear, leer, actualizar y eliminar registros, realizar consultas complejas o definir relaciones con otros modelos.

A continuación, se muestra una lista con todos los modelos del proyecto:

- Menu.
- MenuRol.
- Permiso.
- Rol.
- Usuario.
- Baja.
- Cliente.

- Material.
- Noticia.
- Proyecto.
- Vacaciones.

Los modelos en Laravel representan la capa de datos de la aplicación, gestionando la interacción con la base de datos a través de Eloquent ORM.Cada modelo se asocia a una tabla específica de la base de datos y permite realizar operaciones como la creación, lectura, actualización y eliminación de registros (CRUD) de manera sencilla y orientada a objetos. Además, los modelos pueden definir relaciones entre diferentes tablas, permitiendo una manipulación eficiente y coherente de los datos.

El uso de modelos facilita la abstracción de la lógica de la base de datos, encapsulando las operaciones de datos en una estructura clara y coherente. Esto no solo mejora la organización del código, sino que también permite reutilizar la lógica de datos en diferentes partes de la aplicación, asegurando consistencia y mantenimiento simplificado.

5.6.Controladores

Mientras que los modelos gestionan la lógica y manipulación de los datos, los controladores actúan como intermediarios entre los modelos y las vistas, orquestando la interacción del usuario con la aplicación.

Los controladores en Laravel actúan como los intermediarios entre las solicitudes HTTP entrantes y las respuestas que se envían al navegador del usuario. Son la pieza central de la lógica de la aplicación, determinando qué acciones se llevarán a cabo en respuesta a una determinada petición.

Laravel utiliza un enfoque convencional para organizar los controladores en la carpeta *app/Http/Controllers* (ver Figura 5.22). La creación de un nuevo controlador se hace usando el Artisan CLI: **php artisan make:controller UserController.**

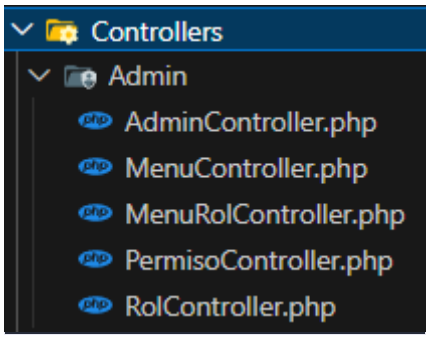

Figura 5.22. Directorio app/Http/Controlers. (Elaboración propia).

Un controlador captura las peticiones del usuario (GET, POST, PUT, DELETE, etc.) y extrae los datos relevantes. Seguidamente, procesa los datos, interactúa con los modelos para acceder a la base de datos, realiza cálculos, valida información, etc. Finalmente genera una vista (HTML), redirige a otra página, o devuelve una respuesta en formato JSON, XML o cualquier otro formato necesario.

Para el proyecto se han creado los siguientes controladores:

- MenuController: controlador del objeto menú.
- MenuRolController: controlador de la relación entre menú y rol.
- PermisoController: controlador del objeto permiso.
- PermisoRolController: controlador de la relación entre permiso y rol.
- RolController: controlador de rol.
- UsuarioController: controlador del objeto usuario.
- LoginController: controlador de inicio de sesión.
- BajaController: controlador del objeto baja.
- ClienteController: controlador del objeto cliente.
- MaterialController: controlador del objeto material.
- NoticiaController: controlador del objeto noticia.
- ProyectoController: controlador del objeto proyecto.
- VacacionesController: controlador del objeto vacaciones.

Los controladores en Laravel actúan como intermediarios entre el modelo, que maneja la lógica de datos, y la vista, que gestiona la presentación al usuario. Los controladores son responsables de procesar las solicitudes entrantes, interactuar con

los modelos para recuperar o manipular datos, y devolver la respuesta adecuada, que generalmente es una vista o una redirección.

Cada acción del usuario en la aplicación puede desencadenar una solicitud que será gestionada por un método específico dentro de un controlador. Estos métodos, a su vez, determinan la lógica de negocio que se debe ejecutar y los datos que se deben enviar a la vista correspondiente.

5.7.Enrutamiento

Para que las solicitudes del usuario lleguen a los métodos apropiados en los controladores, Laravel utiliza un sistema de enrutamiento. El enrutamiento define cómo las solicitudes HTTP son dirigidas a los métodos específicos dentro de los controladores. Este sistema permite mapear URLs a funciones del controlador, asegurando que cada interacción del usuario se maneje de manera correcta y eficiente.

Una ruta es una asociación entre una URL y una acción a ejecutar. Esta acción puede ser una función anónima, un método de un controlador o cualquier otra lógica que se desee ejecutar.

Las rutas deben de escribir en el archivo web.php dentro del directorio *resources/routes* (ver Figura 5.23).

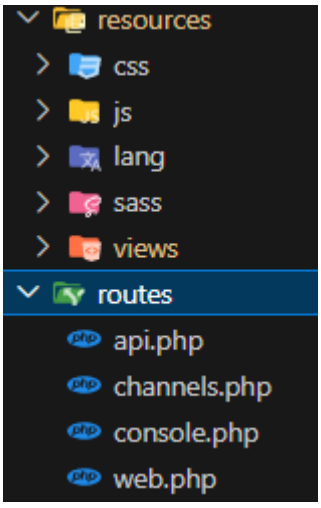

Figura 5.23. Directorio resource/routes. (Elaboración propia).

Laravel utiliza la clase *Route* tal y como se puede ver en la figura 5.27 para definir rutas indicando el tipo de ruta llamando al método correspondiente, indicando en el método la URL a la que se desea acceder, el controlador asociado junto con el método del controlador que gestionará la ruta. De forma opcional se puede nombrar la ruta de con intención identificativa para usar este nombre en vez de la ruta completa desde las vistas usando `route(´nombre_ruta´).`

```
12  Route::get( uri: '/', action: [InicioController::class,'index'])->name( name: 'inicio2');
13  Route::get( uri: 'seguridad/login', action: [LoginController::class,'index'])->name( name: 'login');
14  Route::post( uri: 'seguridad/login', action: [LoginController::class,'login'])->name( name: 'login_post');
15  Route::get( uri: 'seguridad/logout', action: [LoginController::class,'logout'])->name( name: 'logout');
```

Figura 5.24.Definición de rutas. (Elaboración propia).

Las rutas pueden diferenciarse según el tipo de acción deseada por parte del usuario:

- **Rutas GET:** Se utilizan para solicitar datos del servidor.
- **Rutas POST:** Se utilizan para enviar datos al servidor, desde los formularios.
- **Rutas PUT:** Se utilizan para actualizar recursos existentes.
- **Rutas DELETE:** Se utilizan para eliminar recursos.

Método	URL	Acción	Nombre de Ruta
GET	`/usuario/{id}/crear`	create	crear_usuario
POST	`/usuario`	store	guardar_usuario
GET	`/usuario`	show	perfil
GET	`/usuario/{id}/ /editar`	edit	editar_usuario
PUT	`/usuario/{id}/`	update	actualizar_usuario
DELETE	`/usuario/{id}`	destroy	eliminar_usuario

Tabla 5.16.Rutas de Laravel.

En la Tabla 5.17 se ilustra un ejemplo de diferentes tipos de rutas según su funcionalidad.

El enrutamiento actúa como el puente entre las solicitudes del usuario y las respuestas proporcionadas por la aplicación, asegurando que cada solicitud se gestione de manera eficiente y segura. Al definir las rutas, también se determinan las vistas que deben ser renderizadas en función de la lógica de negocio aplicada por los controladores.

5.8.Vistas con Blade

Una vez definido el enrutamiento, el siguiente paso en la estructura de la aplicación es la gestión de las vistas, que representan la capa de presentación del sistema.

Las vistas en Laravel son los archivos que generan la interfaz de usuario de tu aplicación. Son responsables de presentar los datos al usuario de forma visualmente atractiva y funcional.

Se establecen como plantillas, archivos HTML con código PHP embebido que se utilizan para generar páginas dinámicas cuyos componentes pueden reutilizarse en la interfaz de usuario que encapsulan la lógica de presentación.

Tiene como particularidad el uso del motor de plantillas Blade que proporciona una sintaxis sencilla y potente para crear vistas. Un ejemplo es el mostrado en la Figura 5.25 uso `@extends` para introducir código HTML de otra vista y `@section` que tiene como utilidad definir secciones que podrán sobrescribir otras zonas en las vistas definidas con `@yield`.

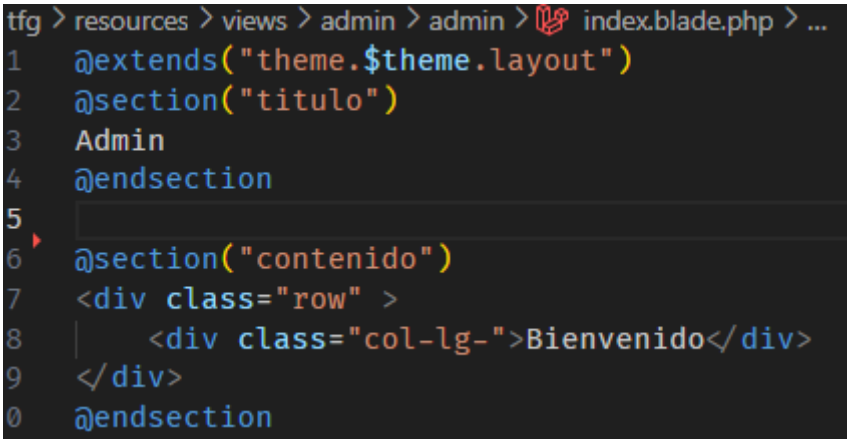

Figura 5.25. Ejemplo de sintaxis Blade en una vista. (Elaboración propia).

Aunque para modelos y controladores si existen Artisan CLI para generarlos automáticamente, las vistas se suelen crear de forma manual en el directorio *resources/views* (ver Figura 5.26) con un nombre descriptivo seguido de .blade.php. En este caso se ha decidido estructurar las vistas en subdirectorios que representen la entidad relacionada.

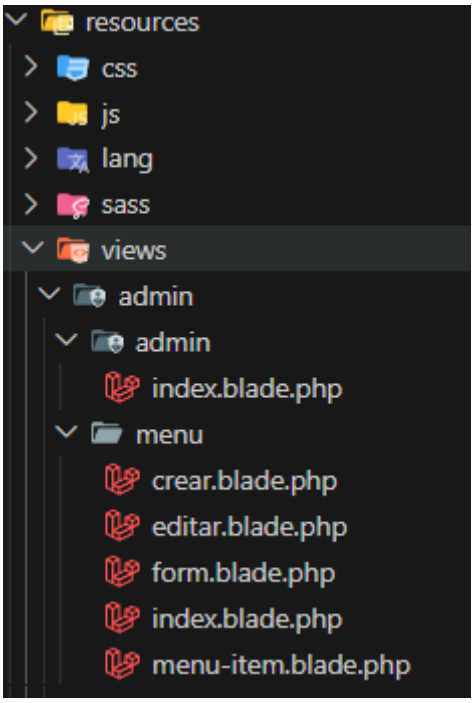

Figura 5.26. Directorio resources/views. (Elaboración propia).

El uso de Blade proporciona una variedad de directivas para controlar el flujo de ejecución, incluir archivos, y realizar otras tareas. Permite evaluar expresiones PHP dentro de las vistas además de usar secciones de estas, que pueden ser reutilizadas o anuladas, y crear componentes reutilizables para estructurar las vistas.

A continuación, se mostrará el listado de vistas que se ha usado en el proyecto y una breve explicación de estas:

- home.blade.php: vista de inicio, se muestra después de la autentificación de usuario.
- **Menu-rol y permiso-rol.**
    - index.blade.php: vista para mostrar información de las relaciones.
- **Menu, permiso, rol, baja, vacaciones, usuario, cliente, material, noticia y proyecto.**
    - crear.blade.php: contiene el formulario para almacenar nuevos registros.

  - editar.blade.php: contiene el formulario para modificar registros creados.
  - form.blade.php: plantilla formulario usado en vistas crear y editar.
  - index.blade.php: muestra un listado de todos los registros creados.

- **Perfil**

  - perfil.blade.php: muestra la información del usuario.
  - form.blade.php: plantilla formulario usado en perfil.

- **Theme**

  - aside.blade.php: barra lateral izquierda que muestra el logo, nombre e imagen de usuario y el menú de navegación.
  - footer.blade.php: pie de página de toda la aplicación web.
  - header.blade.php: cabecera de toda la aplicación web.
  - layout.blade.php: plantilla que contiene los elementos visibles durante toda la navegación en la aplicación web.
  - form.blade.php: plantilla con elementos de formulario usados en toda la plataforma.
  - menu-item.blade.php: plantilla de código HTML usada en la vista aside para el menú.

- **Includes**

  - boton-form-crear.blade.php: botones de crear.
  - boton-form-editar.blade.php: botones de editar.
  - form-error.blade.php: mensaje de error en los formularios.
  - mensaje.blade.php: mensajes informativos para la aplicación.

- **Layouts**

  - app.blade.php: vista predeterminada de AdminLTE, una plantilla HTML.

- **Seguridad**
    - index.blade.php: vista para el inicio de sesión.

Es fundamental subrayar la importancia de implementar reglas y validaciones para garantizar el adecuado funcionamiento de los formularios. Laravel ofrece un sistema de validación robusto y flexible, que permite definir normas para los datos recibidos a través de los formularios. La utilización de estas herramientas no solo protege contra inyecciones de código malicioso, sino que también garantiza la integridad de los datos y mejora la experiencia del usuario.

Para implementar las validaciones, es necesario crear una clase Request, que representa una solicitud HTTP y extiende de FormRequest, una clase personalizada utilizada para definir reglas de validación dirigidas a los datos enviados desde un formulario específico. Esta clase se puede generar directamente mediante el comando CLI de Artisan **php artisan make:request NombreRequest**, y se ubicará en el directorio *app/Http/Requests*. Dentro del archivo Request, se invoca al método `rules()` proporcionado por FormRequest, donde se definen las reglas de validación para los campos del formulario.

Hasta este punto, se ha proporcionado una explicación general sobre los elementos fundamentales que conforman la estructura del proyecto. Las migraciones y los seeders se encargan de crear y poblar las tablas en la base de datos, mientras que los modelos, controladores y vistas gestionan los datos que se mostrarán en el contenido de las páginas web. En esta etapa, es pertinente implementar la autenticación de la aplicación con el fin de controlar el acceso a la misma.

5.9.Módulo Usuario

El módulo de usuario es una de las piezas fundamentales en la implementación de cualquier sistema de gestión, ya que se encarga de la administración de los perfiles de los usuarios que interactúan con la aplicación. En este proyecto, el módulo de usuario ha sido diseñado para gestionar de manera eficiente la información personal, los permisos y las funcionalidades asociadas a los distintos roles dentro del sistema.

Este apartado describirá el proceso de implementación del módulo de usuario, incluyendo las decisiones de diseño adoptadas, las tecnologías empleadas y los desafíos encontrados. También se detallarán las funcionalidades clave del módulo, tales como la gestión de perfiles, la asignación de roles y permisos, y la seguridad en el acceso a la aplicación.

5.9.1.Control de acceso

Laravel concede la posibilidad de configurar de manera sencilla una autentificación de usuarios mediante las librerías guards y providers. Un guard, define la autentificación del usuario en cada solicitud mientras que un provider especifica cómo se recuperan los usuarios desde una base de datos usando Eloquent.

En la aplicación, se usará un guard para definir el método de autentificación basada en sesiones, estas se originarán en el momento en el que un usuario acceda a la aplicación a través de un inicio de sesión usando un nombre de usuario y una contraseña.

Los controladores implicados vienen predeterminados por Laravel en el directorio *app/Http/Controllers/auth* dónde se dispone de diferentes controladores que gestionarán distintas funcionalidades como la verificación de las credenciales, el restablecimiento de la contraseña, registro o inicio de sesión. En este caso se usarán los controladores más básicos en los que se omiten la recuperación de la contraseña o el registro por parte del usuario con motivo de que al estar bajo un contexto de gestión interna empresarial, el acceso mediante credenciales tenga un mayor control por parte de la empresa para evitar riesgos de seguridad. Por lo tanto, *LoginController*

será el único controlador que gestionará el inicio de sesión, este recogerá los datos de la petición HTTP, validará los datos interactuando con el modelo de usuario para su verificación y en caso positivo, generará un token de acceso para identificar al usuario durante la sesión de este. Por último, redirigirá a la página de inicio.

Para configurar la autenticación en Laravel, se ha editado el archivo *config/auth.php*. En este archivo, se ha especificado el modelo que usará el provider de autenticación.

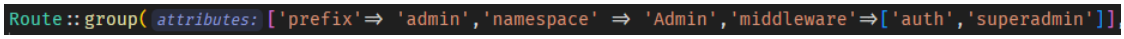

Figura 5.27. Ejemplo de middleware. (Elaboración propia).

En el proyecto existen rutas específicas que necesitan restringirse sólo para el uso del rol administrador y el inicio de sesión, por lo que es útil hacer uso de un middleware, que proveen de mecanismos para filtrar solicitudes HTTP. Un ejemplo de uso de un middleware es el mostrado den la Figura 5.27, dónde se usan dos middlewares, el superadmin, que llama al middleware *PermisoAdministrador*, creado para filtrar la ruta para el rol de administrador y otro, predeterminado de Laravel, auth, para que sólo accedan a las rutas los usuarios que estén autentificados.

En resumen, cuando un usuario accede a la plataforma e introduce sus credenciales en la vista de inicio de sesión, los controladores reciben los datos, y comprueba su existencia en la base de datos, en el caso que no corresponda con la información proporcionada por esta, la sesión es rechazada por la aplicación y se notifica al usuario. Si, por el contrario, ha sido exitosa, el usuario podrá acceder al contenido de la aplicación, el cual se hallará restringido bajo un sistema de permisos y roles.

5.9.2.Gestión de usuario

Como se ha mencionado previamente, el acceso a la aplicación estará restringido mediante el uso de roles y permisos, el cual será gestionado a través de la interfaz por usuarios con rol de administrador. Un administrador tendrá acceso a vistas relacionadas con la gestión de menús, roles, permisos y usuarios, lo que le permitirá realizar cualquier acción básica, como visualizar, editar, crear y eliminar registros.

Los usuarios se muestran en la vista index.blade.php, ubicada en el directorio *resources/views/admin/usuario*. Para acceder a esta vista, primero se invoca al controlador correspondiente mediante la ruta seleccionada en el menú, *admin/usuario*, que dirige al método `index()` del controlador de usuario a través de una solicitud GET. En este método, se utiliza la sintaxis de Eloquent para recuperar todos los registros de usuarios de la base de datos junto con sus roles asociados, y se redirige a la vista de índice de usuario para mostrarlos en una tabla.

Las acciones de editar y eliminar un registro de usuario se presentan en forma de botones dentro de una columna de la tabla, mientras que la opción para crear un nuevo usuario se encuentra en un botón ubicado fuera de la tabla.

Para editar un registro, se accede a la ruta *admin/usuario/id* a través del botón de edición, donde el identificador del usuario correspondiente a la fila seleccionada de la tabla y se pasa como parámetro en la URL. Esto es posible porque, previamente, todos los registros de usuarios se han pasado a la vista como una colección a través del controlador. Al acceder a esta ruta, los datos se envían nuevamente al controlador, pero esta vez al método `edit()`, que realiza dos consultas a la base de datos.

La primera consulta obtiene todos los roles del sistema utilizando los métodos `orderby()`, al que se le pasa el identificador del usuario, `pluck()`, para extraer los atributos nombre e id tomando este último como clave, y `toArray()`, para convertir los datos en un array. La segunda consulta recupera el objeto usuario asociado y sus roles utilizando `with()`, para devolver los roles, y `findOrFail()`, que busca el usuario en la base de datos a partir de su identificador.

Una vez realizadas las consultas, se redirige a la vista editar.blade.php de usuario, donde los datos del usuario seleccionado se pueden modificar mediante campos pre-rellenados con los datos actuales del usuario utilizando el método `old()`. Estos campos están dentro de un formulario con restricciones definidas por las validaciones y la clase Request. Si los datos ingresados no cumplen con los formatos válidos, el sistema no permitirá el envío del formulario y notificará al usuario de los errores. Si los datos cumplen con las reglas definidas en las validaciones, se enviarán a la ruta usuario/id especificada dentro del formulario utilizando el método PUT. Para asegurarse de que el formulario envíe los datos al método correcto y no a otra ruta que comparta la misma URL, es necesario incluir `@method("put")` dentro del formulario.

La ruta invocará el método `update()` del controlador, que primero realizará una consulta para obtener el usuario a modificar. Luego, filtrará los datos recibidos del formulario mediante `array_filter()`, evitando así modificar campos que no hayan sido rellenados, y actualizará al usuario utilizando el método `update()`. Además, se actualizará la relación del usuario con sus roles mediante el método `sync()`, que elimina todas las relaciones existentes y las reemplaza con las nuevas proporcionadas en el formulario. Una vez que la base de datos se ha actualizado correctamente, el sistema redirigirá a la vista inicial con todos los usuarios, mostrando el mensaje "Perfil actualizado con éxito".

El proceso para crear un nuevo usuario es similar al de editar. Al hacer clic en el botón de creación, se accede a la ruta usuario/crear, donde se presenta un formulario con campos similares a los de la edición, aplicando las mismas validaciones. En este caso, no es necesario especificar el método en el formulario, ya que por defecto se utilizará el método POST. Una vez que los datos se envían a través del formulario, se dirigen a la ruta usuario, la cual invoca el método `store()` del controlador. Este método se encargará de crear un nuevo usuario utilizando `create()`, pasando los datos del formulario, y asignará los roles correspondientes mediante el método `sync()`. Finalmente, el usuario será redirigido a la vista de índice con un mensaje de éxito.

En el caso de eliminar un usuario, se debe utilizar un formulario que incluya un botón con la indicación `@method("delete")`, ya que el método invocado en la ruta será DELETE. Al hacer clic en el botón de eliminación, se mostrará un aviso al usuario mediante AJAX para confirmar la acción. Si el usuario acepta, la solicitud se enviará como una petición AJAX. Cuando la ruta invoque el método `destroy()` del controlador, este verificará si se trata de una solicitud AJAX, buscará al usuario mediante una consulta, eliminará las relaciones existentes con `detach()`, y procederá a eliminar al usuario con el método `delete()`. La eliminación del usuario se reflejará en la vista mediante AJAX, que removerá la fila correspondiente al usuario eliminado y mostrará un mensaje en pantalla confirmando el éxito de la operación.

5.9.3.Perfil

El usuario también dispondrá de la vista perfil.blade.php, ubicada en *resources/admin/usuario/perfil*, a la cual se podrá acceder a través de la ruta perfil haciendo clic en el nombre de usuario ubicado en la barra lateral izquierda. En esta vista, el usuario podrá visualizar la información relacionada con su cuenta a través de un formulario cuyos campos estarán pre-rellenados con los datos del usuario. Inicialmente, estos campos no podrán ser modificados, ya que estarán configurados con el atributo disabled.

Para permitir la modificación de los campos, se ha implementado un botón que, mediante JavaScript y el uso del método `onclick()`, activa la funcionalidad de edición en el formulario. Además, se mostrará un botón previamente oculto, que permitirá enviar los datos modificados al controlador a través de la ruta actualizar_usuario, la cual invocará el método `update()` del controlador. Dado que este método ya se estaba utilizando en la vista de edición, su contenido fue ajustado para verificar el origen de los datos utilizando `url()->previous()`. Si los datos provienen de la vista de edición, el sistema redirige a la vista de índice de usuarios; en caso contrario, redirige a la vista de perfil, mostrando los resultados de la modificación junto con un mensaje informativo.

En el módulo de usuario se ha mostrado el comportamiento de los datos y detalles técnicos respecto a la implementación de las operaciones CRUD, presentes en todos los demás elementos de la aplicación. A continuación, se explicará el módulo de proyecto dónde se hará énfasis en el manejo de las relaciones.

5.10.Módulo Proyecto

La gestión de proyectos se lleva a cabo desde la sección dedicada a proyectos, la cual presenta una tabla que lista todos los registros existentes en el sistema, junto con información relevante como el título, el presupuesto y el responsable, además de opciones para editar, eliminar, visualizar el proyecto y crear uno nuevo, mediante botones.

La sección de proyectos implementa la misma funcionalidad CRUD que la sección de usuarios; sin embargo, en este contexto, se abordará desde una perspectiva de las relaciones entre clases. Cada proyecto está asociado con un usuario y un cliente: el usuario representa al trabajador encargado del proyecto, y el cliente es la entidad que ha solicitado el presupuesto. Ambas relaciones son de tipo uno a uno, es decir, un cliente y un usuario por cada proyecto. Este tipo de relación se denota como *One to One*.

Laravel, mediante Eloquent ORM, soporta una variedad de relaciones entre modelos. A continuación, se presenta un resumen de las relaciones utilizadas en este proyecto:

- belongsTo: Indica que un modelo pertenece a otro. Se utiliza en relaciones *One to One*. Ejemplo: un proyecto pertenece a un usuario.
- belongsToMany: Indica que un modelo pertenece a muchos otros modelos y viceversa. Se utiliza en relaciones *Many to Many*. Ejemplo: un permiso puede estar asociado a muchos roles, y un rol puede tener múltiples permisos.

El primer paso para crear el módulo de proyectos es definir la estructura de la base de datos utilizando migraciones. En este caso, dado que existen relaciones entre

tablas, estas se definirán utilizando el método foreignId(), pasando como parámetro la clave primaria del modelo a vincular, y el método constrained(), que define la clave foránea y establece una restricción de integridad referencial entre dos tablas, pasando el nombre de la tabla del modelo a relacionar. Además, se emplean los métodos `onDelete('restrict')` para evitar la eliminación de registros relacionados, es decir, para que al eliminar un proyecto no se eliminen también los usuarios asociados a este, y `onUpdate('restrict')` para prevenir la modificación de la clave primaria del usuario en la tabla de proyectos.

Una vez definida la estructura de la base de datos, el siguiente paso es crear el modelo y reflejar las relaciones en este. Es necesario crear un método que genere una consulta a la base de datos para establecer la relación entre modelos, teniendo en cuenta el tipo de relación. En el caso de los proyectos, al tratarse de una relación *One to One*, se utilizará el método `belongsTo()`, al cual se le pasa el modelo y la clave primaria de la relación. Cabe destacar las convenciones de nomenclatura en Laravel, donde el nombre del método que devuelve los resultados de la consulta se ajusta a la singularidad o pluralidad de los nombres de los modelos. En este caso, los métodos se nombrarían `usuario()` y `cliente()`. Si la relación fuera de tipo *Many to Many*, se utilizarían los nombres en plural, como `usuarios()` y `clientes()`.

En el desarrollo de controladores, se presentarán situaciones en las que sea necesario pasar las relaciones del modelo a una vista. El método `with()` de Eloquent permite cargar automáticamente las relaciones junto con los modelos principales en una sola consulta a la base de datos por cada relación. Este método se utiliza, por ejemplo, en la función `index()` del controlador, donde la consulta devuelve el objeto proyecto junto con el identificador y el nombre del usuario asociado, para luego ser enviado a la vista index como una colección de datos.

Dentro de la vista index, la relación se muestra, por ejemplo, en una tabla, utilizando un bucle que recorre todos los objetos de la colección y accediendo a los atributos de cada elemento de la colección para mostrar el nombre del usuario.

6. EVALUACIÓN Y RESULTADOS

El capítulo de Evaluación y Resultados se centra en analizar y presentar los resultados obtenidos tras la implementación de la aplicación software desarrollada. En esta sección, se evaluarán aspectos como la eficiencia, usabilidad y estabilidad de la aplicación, utilizando métricas y pruebas que permitan determinar su efectividad. Los resultados obtenidos no solo reflejarán el éxito del desarrollo, sino que también proporcionarán una base para futuras mejoras y optimizaciones del software.

6.1.Pruebas de unidad

Es una técnica fundamental en el desarrollo de software que consiste en aislar y probar cada unidad de código de forma independiente. Una unidad de código puede ser una función, un método o una clase. Las pruebas unitarias aseguran que cada componente funciona correctamente por sí solo, lo que facilita la integración de diferentes partes del sistema.

Para realizar este tipo de pruebas se ha usado PHPUnit, un framework de pruebas unitarias ampliamente utilizado en el desarrollo de aplicaciones PHP. Proporciona una forma estructurada y eficiente de escribir y ejecutar pruebas para asegurar la calidad y la corrección de tu código.

Se ha creado una clase de prueba AplicacionTest de PHPUnit en la que se han simulado diferentes funciones que representan el comportamiento de la aplicación.

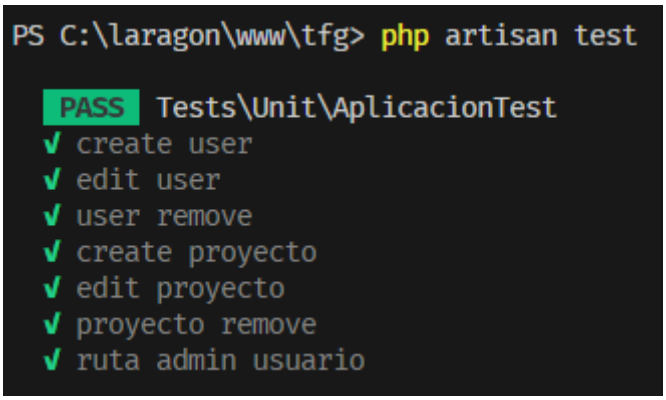

Figura 6.28. Test de PHPUnit

En la Figura 6.28, se muestra un ejemplo de los test que han simulado funciones como las básicas de CRUD en los modelos de *Usuario* y *Proyecto* junto con la comprobación de acceso a la *ruta /admin/usuario* después de ejecutar todos los test usando el comando **php artisan test**. Si el test resulta favorable se mostrará en la terminal 'PASS' en caso contrario devuelve un error indicado en el test y el motivo de este.

Con las pruebas se ha concluido una buena funcionalidad separando los métodos más irrelevantes, como los CRUD. Con PHP es posible identificar errores dentro de métodos con mayor facilidad aislando el código.

6.2.Pruebas de integración

Se verifica cómo interactúan diferentes componentes o módulos de un sistema. En otras palabras, se busca garantizar que las distintas partes del software trabajen juntas de manera correcta y coherente.

Para estas pruebas se ha usado Debugbar Laravel, extensión popular para el framework PHP Laravel que proporciona una barra de herramientas interactiva en la parte inferior de las páginas web que muestra información útil sobre el rendimiento, las consultas de la base de datos, las variables, las rutas, los mensajes de depuración, etc.

A continuación, se presentará los resultados obtenidos del uso de esta herramienta con el módulo de usuario y el de proyecto.

Antes de pasar con los módulos, se ha podido comprobar el correcto funcionamiento del inicio de sesión observando en barra de herramientas la información obtenida de la sesión dónde se puede ver como recibe las URLs, si la autentificación ha sido correcta e información asociada al usuario como su rol y nombre de usuario (ver Figura 6.29).

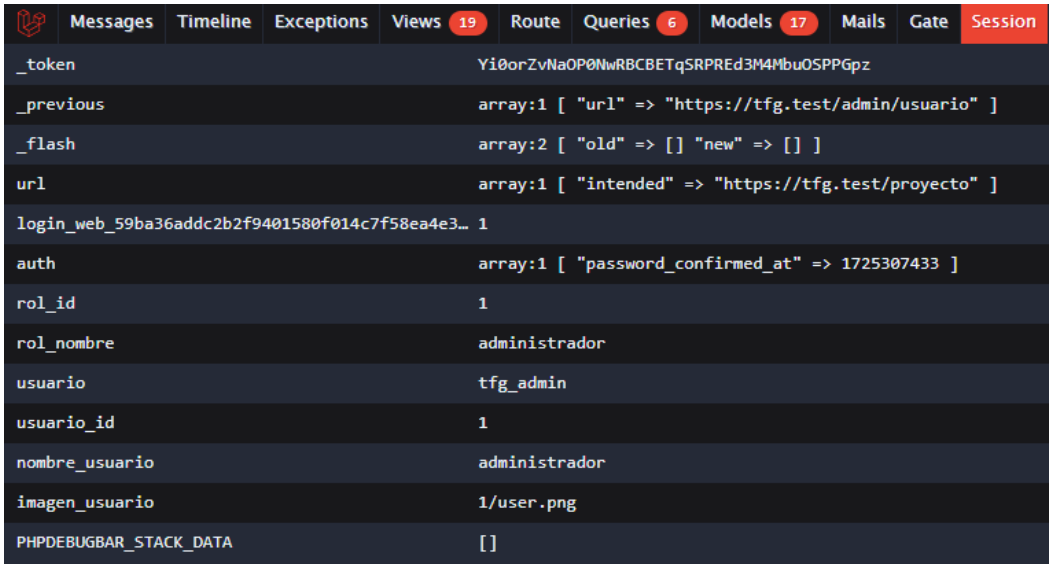

Figura 6.29. Información de sesión. (Elaboración propia).

Por otro lado, se ha podido verificar la carga tanto de vistas como de modelos.

En la Figura 6.30 y Figura 6.31, se muestran en la barra de herramientas todas las vistas cargadas en la página web del listado de usuario dónde se aprecia las vistas correspondientes al layout, header, footer y aside (barra lateral izquierda) además del index para mostrar el listado de usuarios y algunos elementos usados en todas las páginas. De estas vistas se espera que carguen en todas las páginas dentro del proyecto, con el uso de esta herramienta se ha podido comprobar que cumplen con esta funcionalidad. También se puede ver como carga la vista index en sus respectivas páginas.

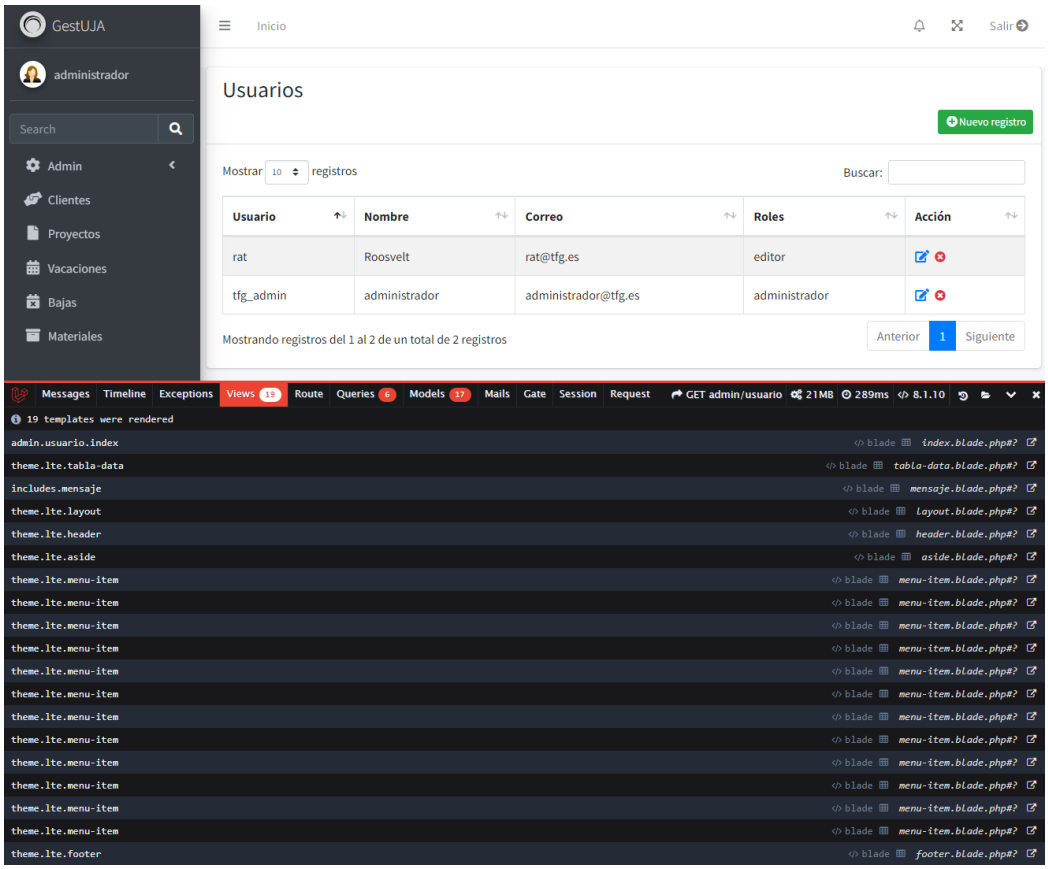

Figura 6.30.Carga de vistas en la página usuarios. (Elaboración propia).

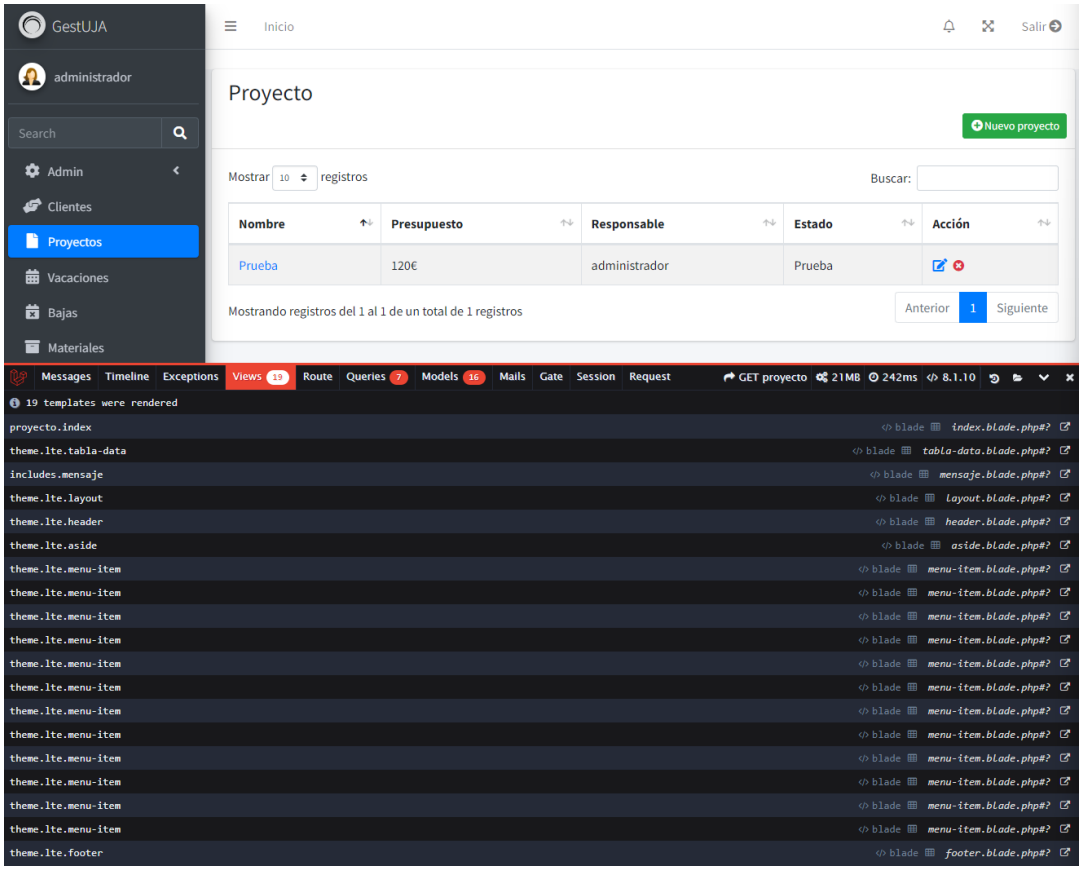

Figura 6.31. Carga de vistas en la página proyectos. (Elaboración propia).

Siguiendo con la sección de usuarios se puede comprobar que modelos necesita cargar cada vista para su plena funcionalidad. En la Figura 6.32 se puede apreciar que el modelo *Menu* se carga 12 veces que representan las secciones accesibles desde el menú de la barra lateral izquierda en la interfaz. En el caso del modelo usuario se carga tres modelos, el modelo de usuario correspondiente a la sesión, y los otros dos que se muestran en el listado junto con dos modelos *Rol* para los roles de los usuarios en el listado.

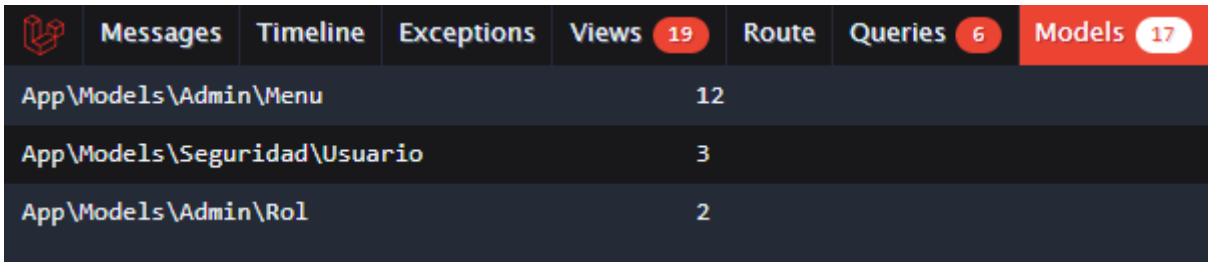

Figura 6.32. Funcionalidad modelos en listado usuarios. (Elaboración propia).

Comparando con el listado de proyectos, en la Figura 6.33 se vuelven a repetir los 12 modelos de *Menu*, comprobando así que el menú se encuentra presente desde las distintas secciones. Usuario se carga dos, una para sesión y otra para mostrar el responsable en la lista, un cliente por relación con proyecto y un solo modelo de proyecto al haber creado sólo uno.

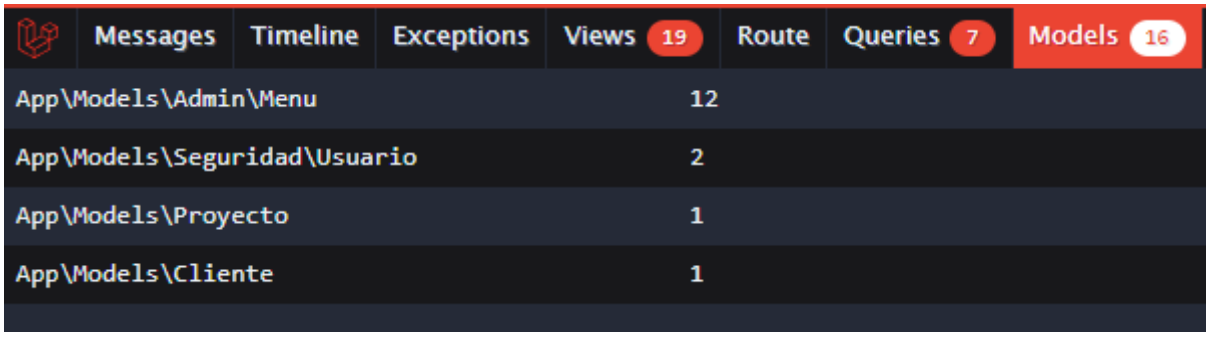

Figura 6.33. Funcionalidad modelos en listado proyecto. (Elaboración propia).

Una vez comprobado los modelos y vistas, el siguiente paso es comprobar las consultas que se han hecho en los módulos, de forma que tenga cohesión con lo presentado en las vistas.

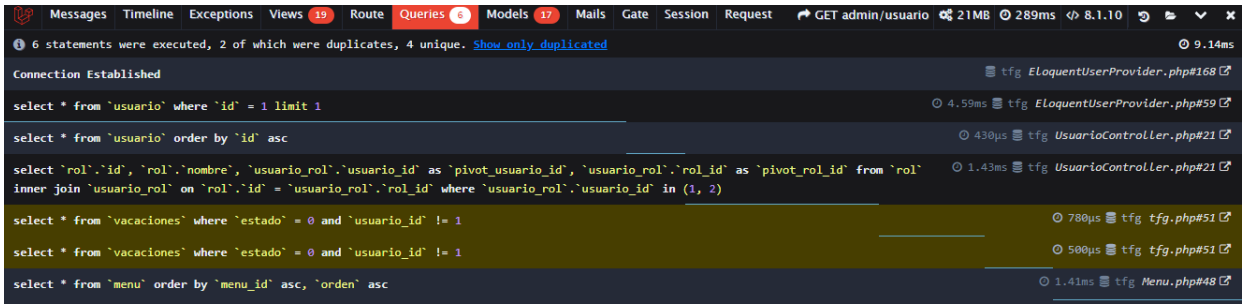

Figura 6.34. Consultas de usuario. (Elaboración propia).

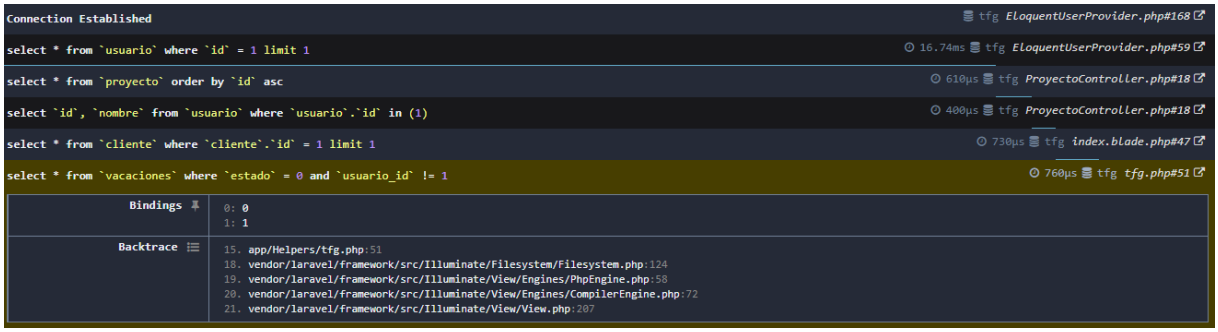

Figura 6.35. Consultas de proyecto. (Elaboración propia).

Lo esperado en el módulo de usuario es que para el listado se haga una consulta que obtenga todos los usuarios de la base de datos para mostrarlos en la lista y otra para los roles para posteriormente enviarla a vistas como crear, editar y mostrar. Estas consultas se deben de hacer desde el controlador *UsuarioController*. Como se muestra en la Figura 6.34, nos muestran las consultas necesarias mencionadas anteriormente junto con el controlador llamado esperado. En la Figura 6.35 se puede comprobar que para proyecto se hacen las consultas correctas necesarias para el listado de este. Además, se puede observar como la herramienta avisa de que existen consultas duplicadas las cuales se producen en la línea archivo tfg.php.

Por último, se ha verificado el comportamiento de las rutas:

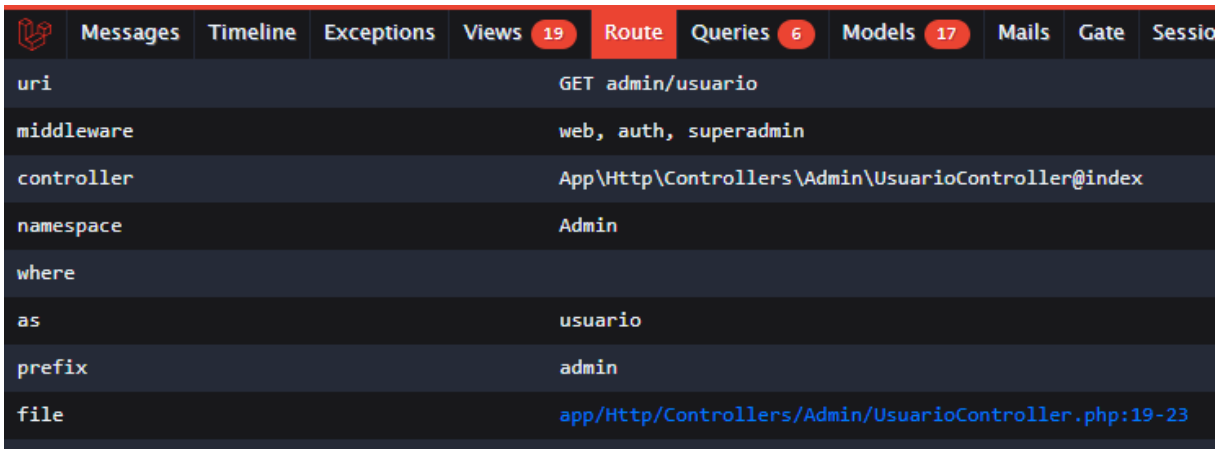

Figura 6.36. Ruta de admin/usuario

Dentro del apartado “Route”, en la barra de herramientas se muestra información de la ruta como su dirección URL junto con el método, el middleware que usa, si está agrupado en un grupo de rutas bajo un prefijo y desde dónde se ha hecho la llamada a la ruta (ver Figura 6.36).

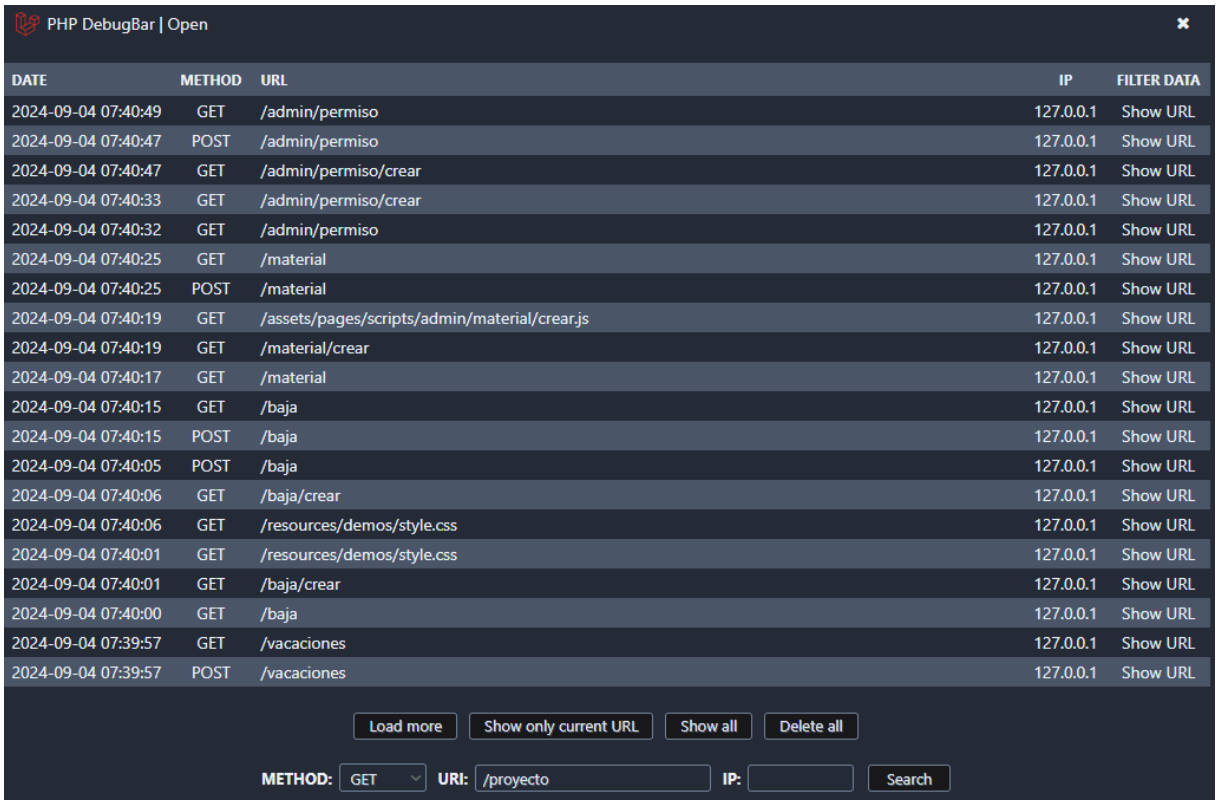

Figura 6.37. Historial de rutas.

En la Figura 6.37 se muestra un historial con todas las rutas ejecutas y sus métodos después de realizar varias acciones en la plataforma como la creación de

material, vacaciones y bajas. Se puede ver como las rutas se llaman correctamente para cada acción realizada.

6.3.Pruebas no automatizadas

Se ha de mencionar el uso de pruebas manuales implican la ejecución de casos de prueba por parte de un tester humano sin el uso de herramientas automatizadas.

Una vez terminado el desarrollo se han hecho pruebas para asegurar la ejecución y tiempo de respuesta por parte de la interfaz además de comprobar la adaptabilidad en móviles. Las pruebas se han centrado en sistema de autentificación y sistema de permisos y roles en el inicio de sesión del usuario usando diferentes roles para comprobar el acceso a distintas secciones, además de la creación, modificación y eliminación de modelos dentro de las listas comprobando las respuestas AJAX y la representación de dichas acciones en la interfaz.

A lo largo de estas pruebas se han ido encontrando errores de interfaz y paso de parámetros en formularios en crear proyecto y usuarios y se han corregido con el apoyo del sistema de mensajes de Debugbar, que actúa como el helper dd de Laravel (función auxiliar que puede usarse en toda la aplicación para mostrar variables) que permite mostrar el contenido de variables sin ocultar la interfaz de la aplicación.

Tras la realización de las pruebas y evaluaciones del software, se puede concluir que la aplicación desarrollada cumple con los requisitos funcionales y no funcionales establecidos en las etapas iniciales del proyecto. Las pruebas de integración y de unidad demostraron que los módulos individuales funcionan correctamente tanto de manera aislada como en conjunto, garantizando la integridad y coherencia de los datos a lo largo de los diferentes componentes del sistema.

Las pruebas de rendimiento indicaron que la aplicación es capaz de responder con rapidez a las distintas operaciones sin experimentar degradaciones significativas en la velocidad o eficiencia. Además, las pruebas de seguridad confirmaron que las medidas implementadas para proteger los datos de los usuarios y prevenir ataques cibernéticos son efectivas.

Sin embargo, se identificaron áreas que podrían beneficiarse de mejoras en versiones futuras, tales como la optimización de la interfaz de usuario para una mayor usabilidad en móvil. Estas mejoras, aunque no afectan el cumplimiento de los requisitos actuales, representan oportunidades para aumentar la robustez y escalabilidad del software en el futuro.

En resumen, el software ha superado con éxito todas las fases de prueba y evaluación, lo que valida su preparación para ser implementado en un entorno de producción.

7. CONCLUSIONES

En este capítulo se presentan las ideas recopiladas a lo largo del proyecto, junto con una reflexión sobre ellas, manteniendo siempre la perspectiva de mejora futura para la aplicación.

El objetivo principal de este proyecto fue el análisis, diseño e implementación de una aplicación web que permitiera gestionar de forma privada las actividades empresariales básicas relacionadas con la organización interna, asegurando la accesibilidad tanto desde un ordenador como desde un dispositivo móvil.

Se optó por utilizar un modelo de arquitectura en tres capas, diferenciando entre la capa de presentación, la capa de aplicación y la capa de datos, con el fin de simplificar el desarrollo de cada una de estas sin afectar significativamente a las demás.

Para llevar a cabo el proyecto, se eligió el lenguaje de programación PHP, utilizando el framework Laravel, que ofrece una amplia gama de características integradas, tales como autenticación, autorización, enrutamiento, ORM (Eloquent) y plantillas (Blade). Estas herramientas facilitaron la configuración de diversos elementos de la aplicación. El uso del patrón MVC contribuyó a la modulación del proyecto, permitiendo la adición de nuevas funcionalidades y un mejor mantenimiento postproducción.

Además, se emplearon tecnologías como el framework Bootstrap, el motor de plantillas Blade, así como JavaScript y AJAX, para desarrollar una interfaz intuitiva y fácil de usar, con una adaptabilidad adecuada tanto para ordenadores como para dispositivos móviles.

Al finalizar, se logró desarrollar una aplicación que cumplió con los objetivos planteados al inicio del trabajo. El proyecto incluye funcionalidades básicas de gestión de empleados y proyectos, proporcionando opciones para contener y reflejar información importante sobre clientes, proyectos, ausencias y material de la empresa, además de ofrecer herramientas para la gestión de días libres.

En cuanto al trabajo futuro, el proyecto está diseñado para facilitar la adición de módulos funcionales adicionales según se requiera. Existen numerosas ideas beneficiosas para futuras expansiones, como la incorporación de un sistema de fichaje para el personal, un planificador de horarios rotativos o la generación automática y gestión de facturas. En términos de mejoras a las funcionalidades existentes, se podría fortalecer la sección de proyectos con información más detallada y un filtrado de tablas más avanzado. En cuanto al rendimiento, sería posible implementar AJAX para mostrar una cantidad limitada de elementos en las tablas. Estos son solo algunos ejemplos de las mejoras que podrían realizarse, gracias a la escalabilidad que ofrecen las tecnologías utilizadas en el proyecto.

Bibliografía

Apache. (s.f.). *About the Apache HTTP Server Project*. Recuperado el 03 de septiembre de 2024 de https://httpd.apache.org/ABOUT_APACHE.html

Bascón Pantoja, E. (2004). *El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swin*g. Acta Nova, 2(4), 493-507.

Bootstrap. (s.f.). *Documentación oficial*. Recuperado el 04 de septiembre de 2024 de https://getbootstrap.com/docs/5.3/getting-started/introduction/

Boykin, R.F. (2001). *Enterprise resource-planning software: a solution to the return material authorization problem*, *Computers in Industry*, Vol. 45, pp. 99-109.

Castro Gil, R. (2004). *Estructura básica del proceso unificado de desarrollo de software*.

Cowburn, P. (2024) *Historia de PHP y Proyectos Relacionados (2024)*. PHP.net. Recuperado el día 25 de Julio de 2024 de https://www.php.net/manual/es/history.php.php

Del Valle, R. V., & Granados, J. M. (2007). *Programación en capas*. Di Mare, Costa Rica.

Git. (s.f.). *Una breve historia de Git*. Recuperado el 25 de julio de 2024 de https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git

Huang, M; Wang, J y Wang, H. (2010). *Research for Real Time Information Transfer Scheme Based on HTTP Persisten Connection*.

Kit Digital (2021). *Kit Digital | Red.es*. Recuperado el día 25 de julio de 2024 de https://www.red.es/es/iniciativas/proyectos/kit-digital

Laragon. (s.f.). *Documentation | Laragon*. Recuperado el día 04 de septiembre de 2024 de https://laragon.org/docs/

Laravel. (s.f.). *Documentación oficial*. Recuperado el día 04 de septiembre de 2024 de https://laravel.com/docs

Laravel. (s.f.). *Documentación oficial de Blade.* Recuperado el día 04 de septiembre de 2024 de https://laravel.com/docs/9.x/blade

MariaDB. (s.f.). *MariaDB in brief*. Recuperado el día 03 de septiembre de 2024 de https://mariadb.org/en/

Ministerio de asuntos económicos y transformación digital (2021, diciembre 30). *Bases Reguladoras Kit Digital*.https://www.acelerapyme.gob.es/sites/acelerapyme/files/2021-12/BOE-A-2021-21873.pdf

Moreno Sánchez, J y Ortiz Rojas, O y Vargas Espinosa, W. (2024). *Metodología de gestión de proyectos de software para la empresa COLNEX SI SAS*.

Latorre Ariño, M. (2018*) Historia de la web, 1.0,2.0,3.0 y 4.0*.

Pinckaers, F.(s.f.). *The Odoo story*. Recuperado el día 03 de septiembre de 2024 de https://www.odoo.com/es_ES/blog/odoo-news-5/the-odoo-story-56

Pollock, P. (2013). *Web Hosting for dummies*. John Wiley & Sons.

Rawal, B. S., Karne, R. K., & Wijesinha, A. L. (2012, July). *Split protocol client/server architecture. In 2012 IEEE Symposium on Computers and Communications (ISCC) (pp. 000348-000353). IEEE.*

Shehab, E.M., y Sharp, M.W., y Supramaniam, L. y Spedding, T.A. (2004), "*Enterprise resource planning: An integrative review", Business Process Management Journal*, Vol. 10 No. 4, pp. 359-386.

Smith, J. (2022). *Choosing the right ERP system: A practical guide.* TechCrunch.

Solano-Fernández, E y Porras-Alfaro, D. (2020) *El modelo iterativo e incremental para el desarrollo de la aplicación de realidad aumentada Amon_RA.*

Taylor Otwell. (2011). *Laravel: The PHP Framework for Web Artisans. Laravel.com.* https://laravel.com

Yen, D.C., Chou, D.C. y Chang, J. (2002), "*A synergic analysis for Web-based enterprise resources-planning systems", Computer Standards & Interfaces,* Vol. 24 No. 4, pp. 337-46.

Glosario

- **AJAX -** Asynchronous JavaScript XML.
- **API -** Application Programming Interface.
- **BD -** Base de Datos.
- **CSS -** Cascading Style Sheets.
- **HTML -** Hypertext Markup Language.
- **HTTP -** Hypertext Transfer Protocol.
- **MVC -** Modelo Vista Controlador.
- **MySQL** – My Structured Query Language.
- **PHP –** Hypertexr Preprocessor.
- **PUDs -** Proceso Unificado de Desarrollo Software.
- **URL -** Uniform Resource Locator.
- **TCP -** Protocolo de Control de Transmisión.
- **WSCI -** Web Server Gateway Interface.

Anexos

Anexo I – Despliegue del proyecto

El despliegue del proyecto es una fase crítica que implica la transferencia del software desarrollado desde el entorno de desarrollo al entorno de producción, donde estará disponible para los usuarios finales. Esta etapa no solo requiere la instalación del software en servidores o plataformas adecuadas, sino también la configuración de todos los componentes necesarios para garantizar que el sistema funcione correctamente y de manera eficiente.

En esta sección, se detallará el proceso de despliegue seguido para la aplicación desarrollada, diferenciando entre el despliegue para el desarrollo en local y para el entorno de producción en servidor web.

Para el correcto despliegue de la aplicación serán necesarios una serie de aplicaciones y herramientas:

En Local:

- Tener previamente instalado Laragon.
- Cualquier entorno de desarrollo.

En servidor web:

- Servidor web accesible por SSH. En este caso se ha usado una máquina virtual con Debian en la versión 12.
- Control de versiones Git instalado en el servidor.
- Base de datos MariaDB instalado en el servidor.
- PHP versión 8 instalado en el servidor con los paquetes curl,cli, fpm, -mysql, mbstring y sus dependencias.
- Gestor de dependencias para PHP Composer.
- Firewall

Local

Para crear el proyecto con clic derecho del ratón se mostrará un menú contextual que permite especificar a Laragon la configuración y puesta en marcha automática de un nuevo proyecto Laravel (ver Figura 38).

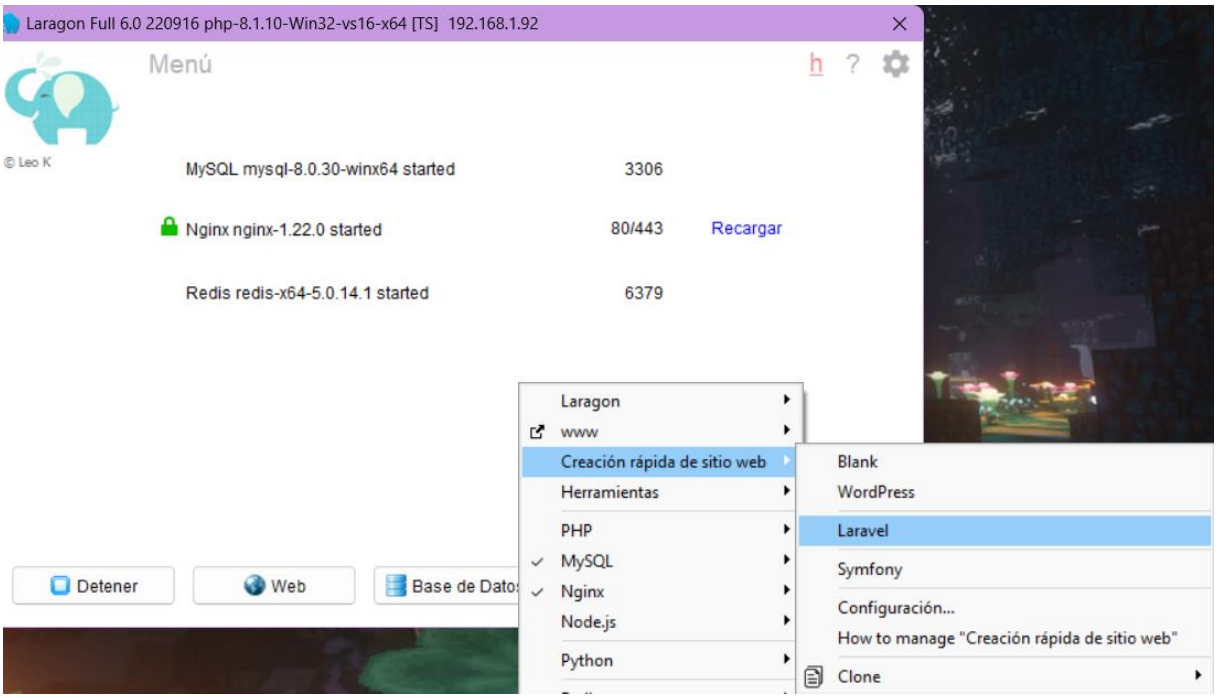

Figura 38. Configuración de Laragon. (Elaboración propia).

Este nuevo proyecto se reflejará en la carpeta www de Laragon, tal y como se muestra en la Figura 39 con example.

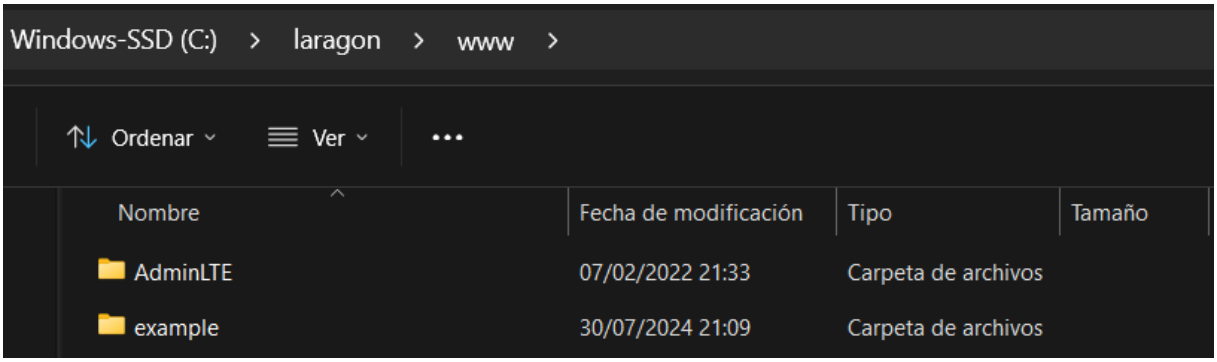

Figura 39. Directorio www de Laragon. (Elaboración propia).

Ya sólo queda iniciar los servicios de web pulsando el botón Iniciar todo en Laragon para poder ver el nuevo proyecto desde cualquier navegador accediendo

mediante la siguiente sintaxis de URL, dónde example es el nombre del proyecto generado anteriormente: example.test (ver Figura 40).

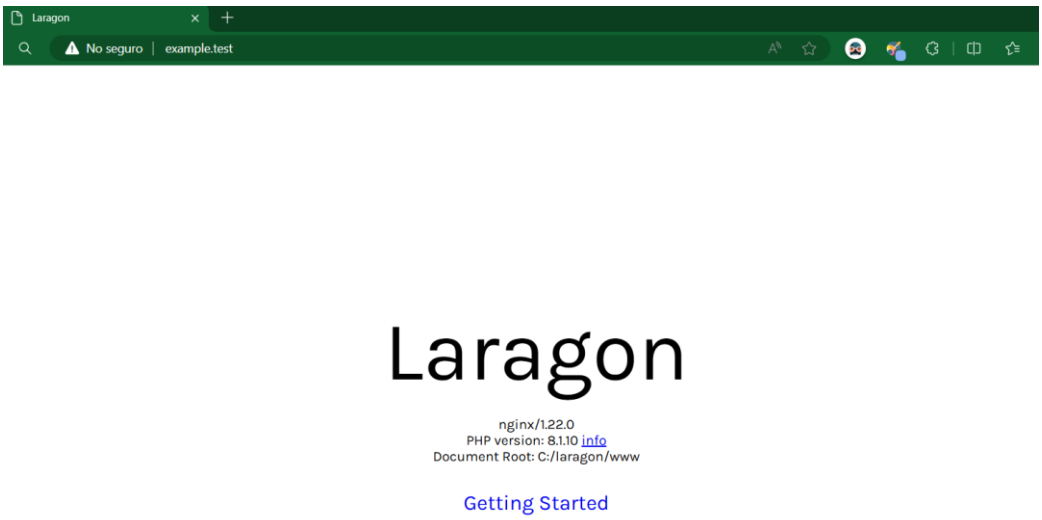

Figura 40. Visualización proyecto usando Laragon. (Elaboración propia).

Con esta prueba, el proyecto estará listo para comenzar su construcción desde un entorno de desarrollo.

En servidor web

Primero, para que el proyecto se ejecute con el nuevo server es necesario modificar el archivo .env-example con los nuevos datos para conectar con la base de datos del servidor web mostrados en la Figura 41:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gestujadb
DB_USERNAME=gestuja_admin
DB_PASSWORD=Klov_26lies;
```

Figura 41. Configuración base de datos. (Elaboración propia).

A continuación, en el directorio /var/www/html se activa el archivo de configuración de Laravel laravel.conf y se desactiva el de por defecto de Apache:

```
ylm00004@MV-TFG:/var/www/html$ sudo 2ensite laravel.conf
```

```
ylm00004@MV-TFG:/var/www/html$ sudo 2ensite 00-default.conf
```

El siguiente paso es habilitar el nuevo módulo ReWrite, necesario para manipular las URL de la aplicación de forma que sea más legibles:

```
ylm00004@MV-TFG:/var/www/html$ sudo 2aenmod rewrite
```

Se da permiso de lectura, escritura y modificación al grupo por defecto de apache www-data al directorio /var/www/html de forma recursiva.

```
ylm00004@MV-TFG:/var/www/html$ chmod -R 775 /var/www/html
```

Por último, se necesita reiniciar Apache para afianzar los cambios anteriores:

```
ylm00004@MV-TFG:/var/www/html$ sudo service apache2 restart
```

Con el servidor web ya configurado el siguiente paso es transferir con la herramienta git previamente instalada en el servidor web los archivos que componen la aplicación en GitHub.

Para esto se requiere el uso de un token que se usará como credencial para acceder al repositorio y clonarlo. Este token se genera dentro de la configuración del perfil de Git Hub (ver Figura 42).

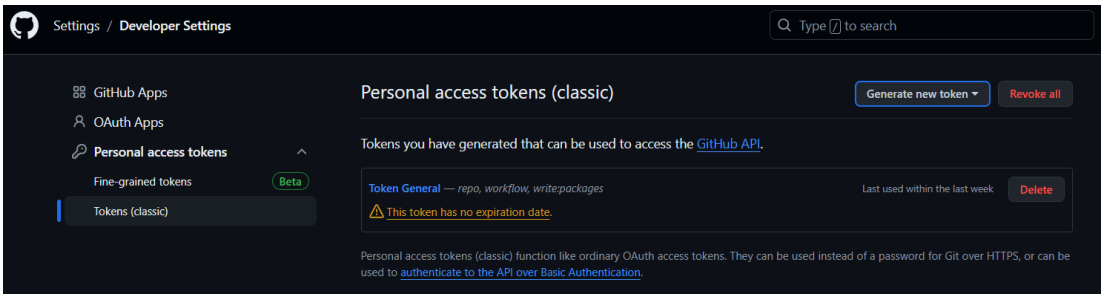

Figura 42. Token del repositorio de Git Hub. (Elaboración propia).

Con el comando de git pull llamamos al repositorio para clonar los archivos en el directorio dónde alojamos la aplicación web. Este nos pedirá el token como usuario y contraseña en cada ocasión que se haga un pull (ver Figura 43).

```
ylm00004@MV-TFG:/var/www/html$ sudo git pull
Username for 'https://github.com': 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
Password for 'https://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@github.com':
```

Figura 43. Bajar repositorio con git. (Elaboración propia).

Una vez que se tengan el proyecto alojado en git subido al web server se han de seguir los siguientes pasos para terminar de configurar Laravel:

1. `ylm00004@MV-TFG:/var/www/html$ sudo composer install`

Obtiene las dependencias del software necesarias para el funcionamiento de Laravel procedentes del archivo composer.json y las instala en directorio vendor.

2. `ylm00004@MV-TFG:/var/www/html$ cp .env.example .env`

Copia el contenido del archivo de configuración .env.example, previamente modificado con los nuevos datos, dentro de .env.

3. `ylm00004@MV-TFG:/var/www/html$ sudo php artisan key:generate`

Genera una clave única para identificar el proyecto, usada para encriptar los datos con el objetivo de preservar la seguridad del proyecto. Dicha clave se guarda en el archivo .env, dónde se definen parámetros de como los datos necesarios para conectarse a la base de datos o la configuración de la caché.

4. `ylm00004@MV-TFG:/var/www/html$ sudo php artisan storage:link`

Comando para habilitar el acceso a los recursos alojados en la carpeta public.

5. `ylm00004@MV-TFG:/var/www/html$ sudo php artisan migrate`

Ejecuta las migraciones de la aplicación.

6. `ylm00004@MV-TFG:/var/www/html$ sudo php artisan db:seed`

Ejecuta los seeders.

7. `ylm00004@MV-TFG:/var/www/html$ sudo php artisan optimize`

Revisa los recursos de la aplicación para optimizar el rendimiento de esta.

Con esto ya se encontraría el proyecto correctamente desplegado, listo para visualizar accediendo a la URL www.sueliman.ujaen.es