UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Aplicación de Gestión y Seguimiento de Tareas "Sudle"

Trabajo Fin de Grado

Grado en Tecnologías Interactivas

AUTOR/A: Hernandez Ramirez, Jonathan Javier

Tutor/a: Bataller Mascarell, Jordi

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÉCNICA DE VALÉNCIA

ESCOLA POLITÉNICA SUPERIOR DE GANDIA

Grado en Tecnologías Interactivas

Aplicación de Gestión y Seguimiento de Tareas "Sudle"

TRABAJO DE FINAL DE GRADO

Autor:

Jonathan Javier Hernández Ramírez

Tutor/director:

Jordi Bataller Mascarell

GANDIA, 2023

RESUMEN

La forma de trabajo en empresas y oficinas actualmente es en gran medida no presencial.
Aunque esto tiene grandes ventajas para el trabajador y la empresa, aparece el problema
del seguimiento del trabajo realizado, especialmente en cuanto a tareas y tiempo invertido en ellas.
Esto es especialmente cierto en el desarrollo de software, donde es necesario un seguimiento del
progreso más detallado. Con Sudle se pretende abordar problemas como la integridad las actividades
realizadas por el personal, como facilitar la monitorización de estas.

The way of working in companies and offices nowadays is largely non-face-to-face.
Although this has great advantages for the worker and the company, there is the problem of monitoring the
work done, especially in terms of tasks and the time spent on them.
of tracking the work done, especially in terms of tasks and time spent on them.
This is especially true in software development, where more detailed progress tracking is needed.
progress needs to be tracked. Sudle is intended to address problems such as the integrity of the activities
performed by the staff, as well as to facilitate the monitoring of these activities.

Índice

ILUSTRACIONES

AGRADECIMIETOS

Deseo expresar mi más sincero agradecimiento a todas las personas que han contribuido al desarrollo de este proyecto. En primer lugar, quiero reconocer la invaluable colaboración de Jordi Bataller Mascarell, quien amablemente aceptó ser el tutor de este trabajo, brindándome no solo documentación esencial, sino también un constante soporte a lo largo de todo el proceso. Mi gratitud también se extiende a Alejandro Losa García, cuyos profundos conocimientos en el ámbito empresarial han sido de gran enriquecimiento. Finalmente, no puedo dejar de mencionar a Jose Julio Peñaranda Jara y Ausias Bañuls Mahiques, cuyo apoyo en la redacción de este documento ha sido fundamental. A todos ellos, mi más profundo agradecimiento.

1. INTRODUCCIÓN

1.1 Presentación y objetivos

En este trabajo hemos hecho el desarrollo desde cero de una aplicación web. Dicha aplicación permite gestionar equipos de proyecto y los recursos invertidos en el desempeño de tareas realizadas por estos. De esta forma los directores de equipo podrán evaluar mejor las cualidades de sus miembros de equipo y ajustar de forma más exacta las capacidades de estos logrando así un trabajo más eficaz y eficiente a la hora de realizar proyectos.

Este proyecto está dividido en tres grandes secciones. Tenemos la parte de servidor, donde se aloja la aplicación. La parte del backend encargada de atender peticiones enviadas por el usuario y donde se aloja la lógica del negocio y por último la parte del frontend que es la que hará de interfaz entre los datos de la aplicación y el usuario.

Dentro de la aplicación web hay diferentes vistas, cada una asociada a un tipo de usuario. Los administradores de proyecto y los miembros de este. Mientras que los miembros podrán ver sus proyectos y tareas asignados, los administradores verán todas las tareas que derivan el proyecto y gráficas con el rendimiento de cada equipo o miembro de equipo, más adelante se detallará en profundidad que puede ver cada usuario y que puede hacer.

Se pretende lanzar la aplicación al mercado como un servicio de subscripción mensual. Habrá un plan gratuito, en este plan estarán incluidos la cuenta del director de proyecto y 3 usuarios miembros de este.

Con nuestra aplicación pretendemos:

- Permitir registrar la jornada laboral de forma sencilla
- Monitorizar el tiempo invertido en proyectos y tareas
- Obtener el rendimiento de cada miembro del equipo
- Optimizar los recursos de los equipos

1.2 Estructura de la memoria

Esta memoria está organizada de la siguiente forma:

- Capítulo 2, Entorno y estado del arte, cómo surgió la idea y entorno en el que se va a desarrollar
- Capítulo 3, Mockups y datos de prueba
- Capítulo 4, Diseño de la aplicación
- Capítulo 5, Implementación de las diferentes partes del proyecto
- Capítulo 6, Manuales y guías de uso del proyecto
- Capítulo7, Evaluación de la aplicación y sus servicios por terceros
- Capítulo 8, Modelo comercial, como se pretende vende el proyecto
- Capítulo 9, En último capítulo, presentamos las conclusiones y proponemos ampliaciones, futuras, así como otras mejoras

2. ENTORNO Y ESTADO DEL ARTE

2.1 Modelo de negocio

Antes de empezar a hablar del entorno en el que se ha desarrollado el proyecto, consideramos que es necesario explicar cuál 2es el modelo de negocio en el que se basa este proyecto.

El modelo de negocio en el que se basa el proyecto es el modelo B2B (Business to Business), esto quiere decir que los productos y servicios que ofrece este proyecto son consumidos por empresas en lugar de consumidores. En este tipo de transacciones ambas partes son empresas, y la relación se basa en satisfacer las necesidades y objetivos mutuos.

Por ello el proyecto a desarrollar se pretende que sea versátil y que sea compatible con casi cualquier plataforma, por eso una de las partes del proyecto es la API Rest, que más adelante se hablara de que es y cómo funciona.

Otra característica de este modelo de negocio es que se quiere vender como una subscripción mensual, de esta forma las estatistas e información más valiosa serán solo para las empresas que tengan un plan contratado. No obstante, si que existirá un plan gratuito, con ciertas limitaciones. De los productos ofrecidos se hablará más adelante.

2.2 Entorno en el que se va a desarrollar

El entorno en el que se va a desarrollar la aplicación está dividido en tres secciones.

2.2.1 Servidor

Para alojar la aplicación web se quiere de un servidor capaz de ello. Estuvimos sopesando diferentes posibilidades, una de ellas era usar el servidor que ofrece la universidad para sus alumnos, sin embargo, el servidor que ofrecen está muy limitado y apenas se puede instalar o añadir nuevas funcionalidades.

Otra opción fue utilizar un sitio web que permita alojar sitios web, sin embargo, al igual que pasaba con el servidor que ofrece la universidad, este también tiene bastantes limitaciones. Finalmente se encontró una web donde se alquilan servidores, contaba.com.

Contabo es una empresa de hosting y servicios en la nube, es uno de los principales proveedores de servicios de alojamiento en Europa. Las características que hicieron que nos inclinásemos hacia esta opción fueron:

- Variedad de servicios: Ofrece una amplia gama de servicios, que incluyen alojamiento web compartido, servidores VPS, servidores dedicados, servicios de nube y más.
- Servidores dedicados y VPS: Ofrece una variedad de configuraciones que van desde servidores básicos hasta soluciones de alto rendimiento con recursos dedicados.
- Precios competitivos: Este factor a contribuido a su popularidad entre individuos, pequeñas empresas y empresas más grandes.
- Flexibilidad: Ofrece opciones flexibles en cuanto a contratos y facturación. Los clientes pueden optar por contratos mensuales, trimestrales, semestrales o anuales, lo que brinda flexibilidad en función de sus necesidades.

Las características del servidor que alquilamos son:

- 4 vCPU Cores
- 8GB de RAM
- 1 Snapshot
- 32 TB de tráfico
- Sistema operativo Ubuntu Server 18.04

2.2.2 Backend

Se entiende como backend la parte de una aplicación que trabaja detrás de la parte visible por los usuarios para que todo funcione correctamente. Esta es la parte que se encarga de procesar datos, gestionar bases de datos, realizar cálculos, etc.

Dado el modelo de negocio escogido y teniendo en cuenta que queremos que este proyecto se pueda integrar fácilmente con todo tipo de plataformas se desarrollará para gestionar las peticiones una API Rest, es decir una aplicación web que organiza mediante enlaces la forma en la que se accede a los datos alojados en esta. Se podría decir que funciona como intermediario que permite que diferentes sistemas se comuniquen entre si de manera eficiente. En lugar de presentar una interfaz visual para interactuar con la

aplicación, la API Rest permite a los programas intercambiar datos y solicitudes utilizando el protocolo HTTP.

Existen numerosos marcos de trabajo para desarrollar una API, pero al final se decidió usar el marco de trabajo Laravel, los motivos por los cuales se escogió este marco de trabajo fueron, facilidad de uso, documentación clara, rutas y controladores, Eloquent ORM, Autenticación, Middlewares, y conocimientos previos con él.

2.2.3 Frontend

Se entiende como frontend a la parte de una aplicación que los usuarios pueden ver y con la que pueden interactuar directamente. En una página web es la interfaz con la que interactúan los usuarios incluyendo el diseño, la disposición de los elementos de la web, los botones, los formularios, en definitiva, cualquier elemento que sea visible en la pantalla.

Para nuestra primera toma de contacto con otro sistema para interactuar con nuestra API Rest se ha pensado en desarrollar una aplicación web reactiva, la cual nos servirá de interfaz entre los datos del servidor y las operaciones que se desean realizar. Para ello se al igual que pasaba con el backend existen múltiples marcos de trabajo que permiten agilizar el desarrollo, en este caso hemos escogido el marco de trabajo VUE.js.

Los motivos por los que se ha escogido este marco de trabajo han sido, por sus conocimientos previos, por su fácil aprendizaje, por su versatilidad para construir aplicaciones desde cero, por su enfoque progresivo, por su reactividad y por su documentación sólida.

2.3 ¿Cómo surgió la idea?

El entorno del que surgió la idea para este trabajo fue, el desarrollo del software. Hoy en día existen una serie de pasos en común para llevar a cabo el desarrollo de software, ya sea una aplicación web, una aplicación móvil o una aplicación de escritorio, todas parten de una serie de funcionalidades (podrían verse como proyectos) las cuales se dividen en tareas.

Existen tres grandes problemas que dieron a cabo la idea de este proyecto.

El primer problema surgió de mi experiencia como alumno. En nuestra carrera GTI, existen asignaturas basadas en el trabajo en equipo, en dicha asignatura se pide a los algunos que formen un equipo de proyecto y se les encarga el desarrollo de una aplicación utilizando determinados dispositivos y tecnologías. Esto exige que haya una serie de roles en el grupo de trabajo, como mínimo debe de haber un director del proyecto (Scrum master) y el resto de los miembros. Cada cuatrimestre se cambia de proyecto y de tecnologías, incluso en ocasiones se cambia hasta de compañeros de grupo. Existen una serie de entregas y unas revisiones periódicas, en las cuales se evalúa el trabajo realizado y el trabajo pendiente, estas entregas se repiten hasta llegar a el producto mínimo viable o la presentación frente al dueño del proyecto (o producto owner).

En mi caso yo era el scrum máster de mi grupo de proyecto, y mis tareas aparte de programar y contribuir con el desarrollo del trabajo, era gestionar los recursos del equipo, y con recursos hago referencia a las capacidades de cada miembro del equipo. No todos son igual de buenos en ciertas tareas, habrá miembros que serán más buenos en el diseño de interfaz y experiencia de usuario, otros que serán mejores en la programación y desarrollos de algoritmos, etc.…
Al principio era difícil identificar que miembro era mejor en qué tipo de tareas y como no se llevaba un control de tiempo invertido, costaba llegar bien a las entregas, en muchas ocasiones los integrantes se atascaban en tareas, las cuales debían ser relevadas por otros integrantes del equipo. Los profesores propusieron varias herramientas para realizar un seguimiento de tareas y de proyecto, Worky y Trello. Sin embargo, a pesar de tener estas herramientas era complicado realizar las tareas de scrum máster. Por ello se pensó en desarrollar una aplicación que se ajustase mejor a las necesidades que un scrum máster puede tener.

El segundo problema surgió cuando entré a trabajar. Entré como programador en una empresa que se dedica al desarrollo de soluciones técnicas y telefonía tanto móvil como fija. Al igual que en los proyectos de clase en la empresa hay jornadas y proyectos, los cuales son asignados a cada empleado en función de sus habilidades y conocimientos en ciertas áreas. Este trabajo realizado por los compañeros tiene que ser seguido por el director de proyecto, y para poder hacer esto, es necesario que los empleados usen un documento Excel donde apunten las tareas que derivan del proyecto asignado y las horas empleadas en

él. Así los próximos proyectos que le lleguen al director de proyecto de ese mismo tipo, Él sabe a quién asignárselo directamente y de esta forma evitar que se pierdan horas en un proyecto que puede hacerse en menos tiempo. El problema que pude ver aquí es que es muy tedioso y engorroso apuntar las horas y las tareas realizadas en el documento, ya que la interfaz de un Excel es muy poco amigable, muchas veces pasa que un empleado se puede equivocar de casilla en la que rellena los datos y esto fastidia los cálculos y gráficas generados. Además de que cada jornada finalizada, se debe de crear un archivo totalmente nuevo usando las mismas formulas, como cada casilla lleva sus cálculos y sus nomenclaturas, en ocasiones las jornadas se crean mal o es difícil para el director del proyecto crear este documento rápidamente. Consideramos que hay formas más semillas de poder crear una jornada y poder gestionar todo esto, por ello surgió un motivo más para desarrollar este trabajo.

Pr último el problema que cada vez se está viendo más a diario es la integridad de la jornada laboral de los empleados. Desde la llegada de la pandemia, debido al confinamiento, se empezó a utilizar el teletrabajo, muchas empresas tuvieron que adaptarse a ello permitiendo a sus trabajadores trabajar desde casa. Esto fue una gran ventaja para las empresas que poseían tareas que se realizaban mediante un ordenador, ya que podían seguir funcionando sin necesidad de una oficina física, sin embargo, aquí nació un gran problema, la veracidad de las horas de trabajo empleadas en el desarrollo de tareas. Al estar en un entorno más personal, más cómodo, muchos trabajadores suelen ver mermado su rendimiento en su jornada laboral, ya sea porque se distraen en casa más fácilmente, porque las condiciones en su entrono no le permiten concentrase y trabajar eficaz o directamente que no quieran realizar sus tareas y se pongan a hacer otras cosas como ver series o películas online debido a que no tiene la supervisión de nadie. Esta mala práctica es muy perjudicial, tanto para los empleados por coger malos hábitos de trabajo, como para las empresas que puede llegar a costarles mucho dinero con personas que falsifican sus datos registrados en su jornada laboral.

A pesar de que en la actualidad ya existen varias herramientas para solucionar estos problemas, no hemos sido capaces de encontrar una aplicación que resuelva todos los problemas vistos. Trello permite la gestión de proyectos y tareas, sin embargo, desde mi punto de vista, te da tantas opciones y de tantas

formas distintas que es abrumador lo que puedes hacer y muchas veces si no se tiene un mínimo conocimiento previo es difícil de usar y aparte algunas de sus funcionalidades más útiles son de pago.

El uso de un archivo Excel es bastante común en las debido a su potencia para realizar cálculos y estadísticas con los datos que hay en él, con pocos conocimientos de ofimática y algunos cálculos se puede realizar un seguimiento de proyecto tan valido como el de Trello o incluso mejor, pero a pesar de ello registrar las horas es una tarea un poco compleja y tener que crear un archivo por jornada puede llevar más trabajo del necesario.

Por estas razones se pensó el desarrollo de este trabajo, Sudle, un proyecto que pretende cubrir estos problemas de forma eficaz y eficiente. Al ser un proyecto tan grande y con tantas funcionalidades a cubrir es muy posible que se tengan que rectificar cosas o cambiar el enfoque en algunas de las funcionalidades de este.

2.4 Funcionalidades y requerimientos

Antes de continuar con las funcionalidades que se van a definir para cada parte del proyecto vamos a hacer un pequeño esquema de las conexiones de cada parte y cómo van a estar interconectadas.

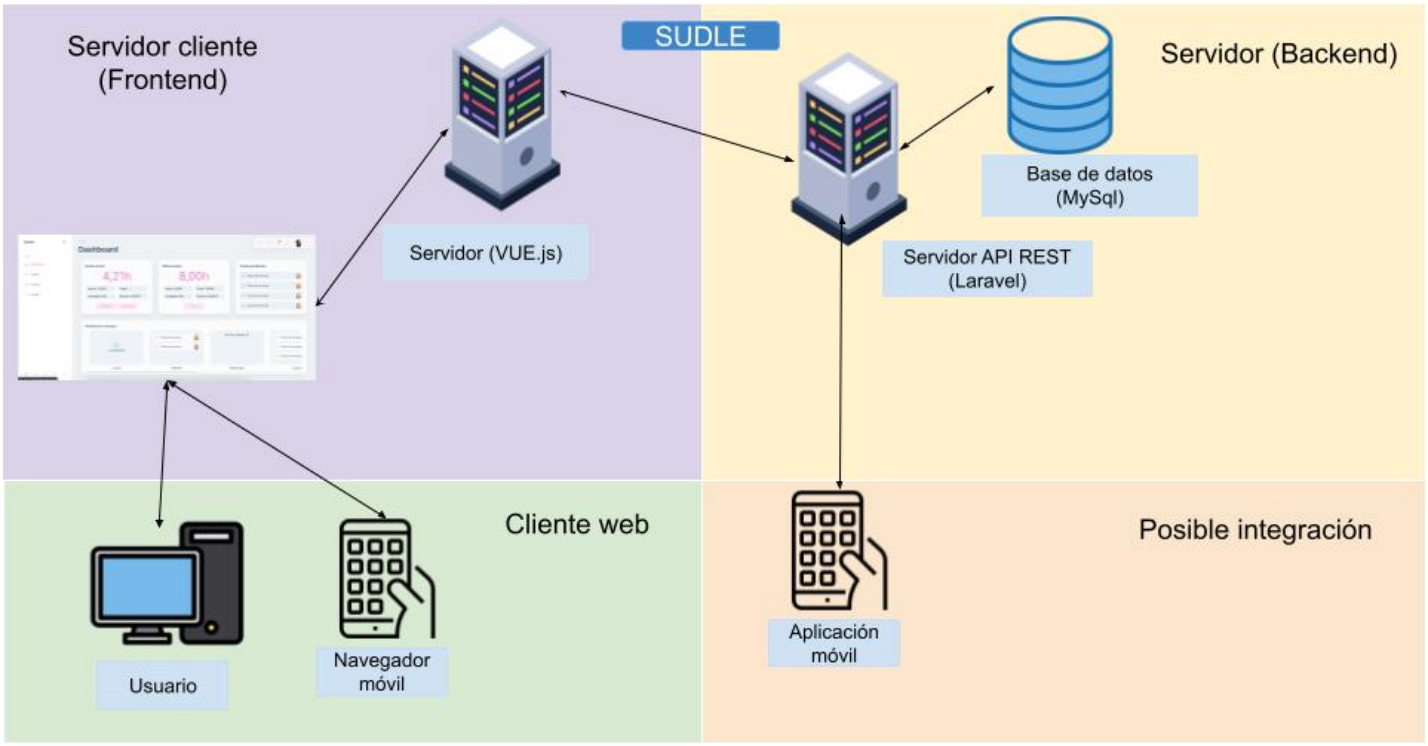


Ilustración 1 Esquema de conexiones del proyecto

Como se ha explicado antes el proyecto está dividido en tres grandes secciones, inicialmente, esto quiere decir que dada la versatilidad de nuestro backend como futura ampliación para el proyecto se podría desarrollar una Aplicación móvil

2.4.1 Funcionalidades del servidor

Necesitamos un servidor que sea capaz de alojar y gestionar nuestra aplicación web, y para ello necesitaremos instalar en nuestro servidor los siguientes servicios:

- Servicio de host, como puede ser Apache o Nginx
- Servidor DNS para la resolución de nombres
- Servidor FTP para almacenar los archivos de la aplicación
- Servidor SMTP para los envíos de correos
- Servidor de base de datos para almacenar datos de la aplicación
- CPanel para poder gestionar desde un único sitio el resto de los servicios mencionados a excepción de la base de datos
- Sistema gestor de bases de datos

Con estos servicios instalados somos capaces de poner en marcha este proyecto.

2.4.2 Funcionalidades backend

Uno de los objetivos principales de este proyecto es que sea versátil y pueda acoplarse a cualquier plataforma, para ello necesitamos una API Rest que sea capaz de:

- Atender peticiones HTTP para comunicarse con otros sistemas
- Proteger la integridad de la información mediante permisos y contraseñas
- Gestionar la información de la base de datos

2.4.3 Funcionalidades frontend

Se requiere de una interfaz gráfica para que los usuarios puedan operar en nuestra aplicación, para ello el frontend debe de tener las siguientes funcionalidades:

- Interfaz gráfica sencilla e intuitiva para agilizar las gestiones

- Registro de las jornadas
- Monitorización de la actividad de los usuarios para analizar su rendimiento
- Registro para una prueba gratuita en la aplicación
- Contratación de planes y suscripciones
- Creación de equipos
- Creación de miembros de una empresa
- Creación de proyectos
- Creación de tareas

3 MOCKUPS Y DATOS DE PRUEBA

3.1 Mockups

A continuación, se mostrarán los mockups que se han preparado para la interfaz gráfica de la aplicación

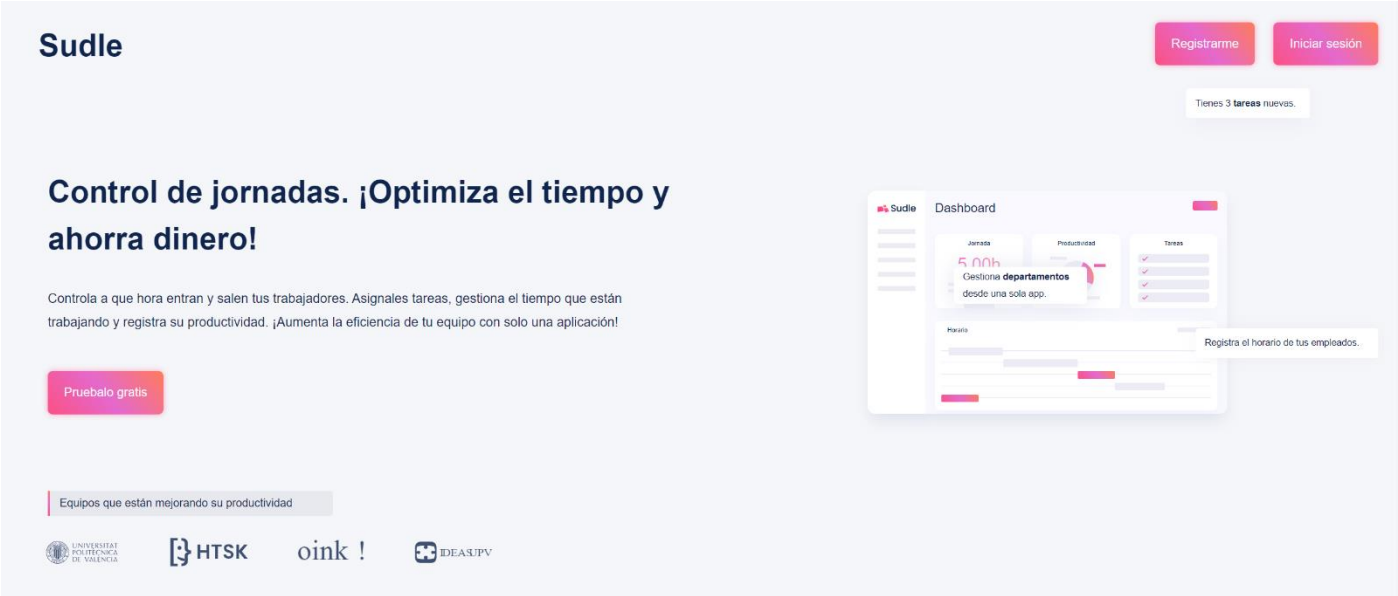


Ilustración 2 Mockup de la landing page parte 1

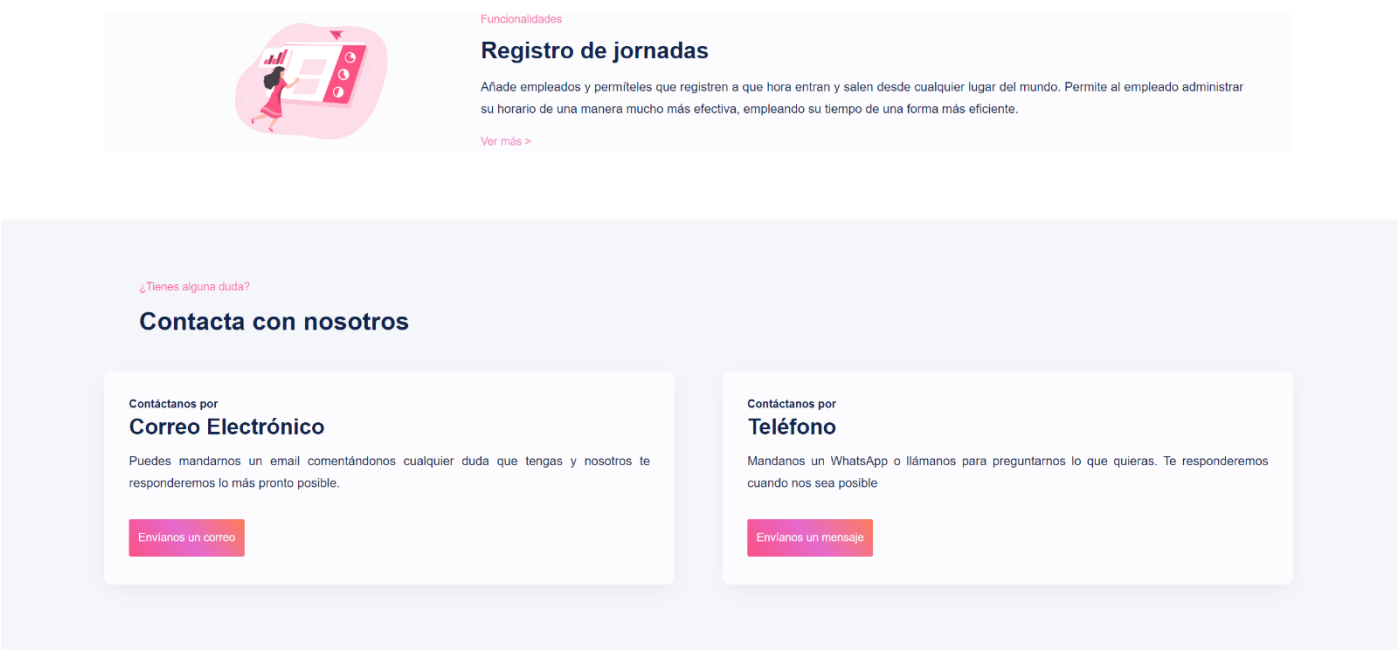


Ilustración 3 Landing page parte 2

Esta será la página que se use para la captación de potenciales clientes, ya que en ella se listarán las funcionalidades de la aplicación.

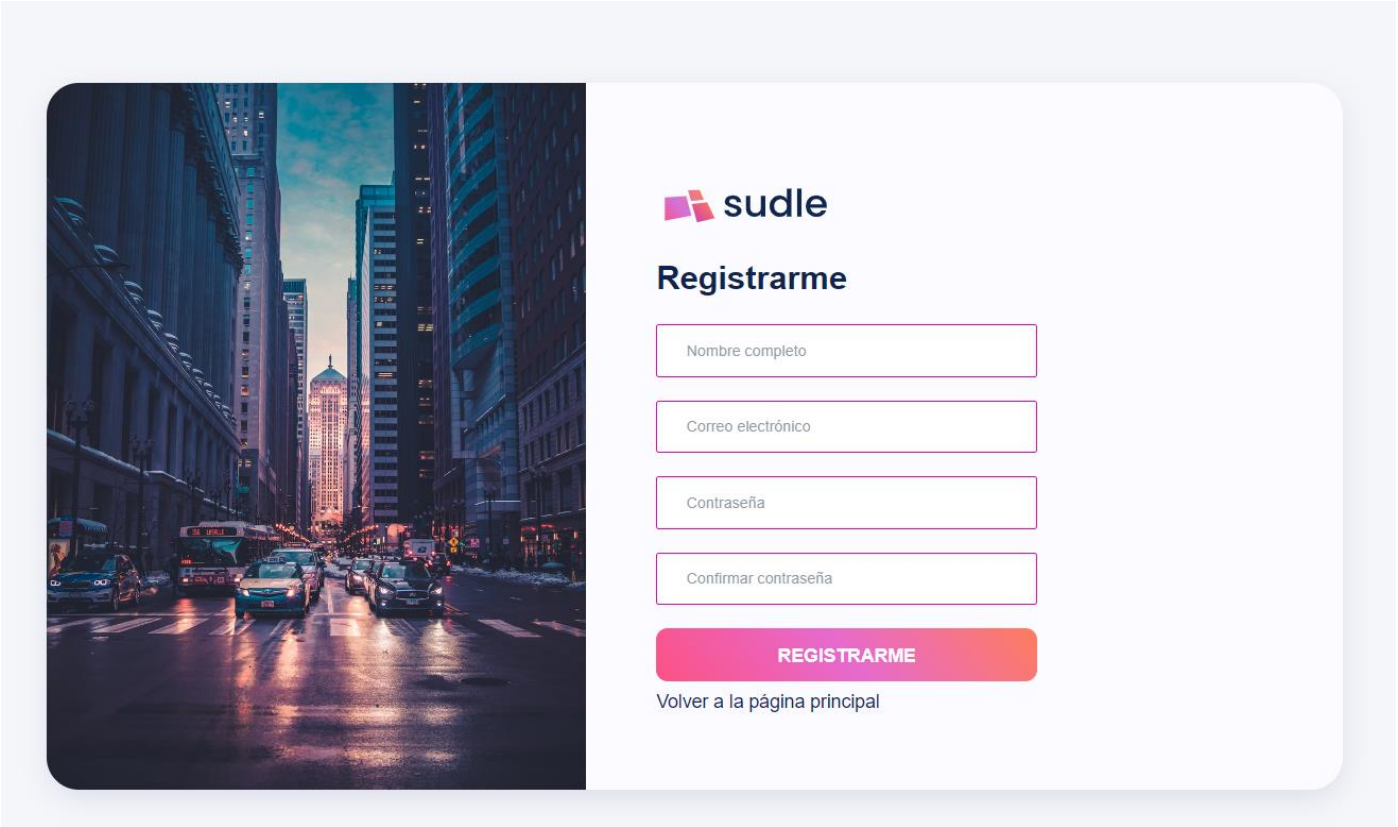


Ilustración 4 Formulario de registro

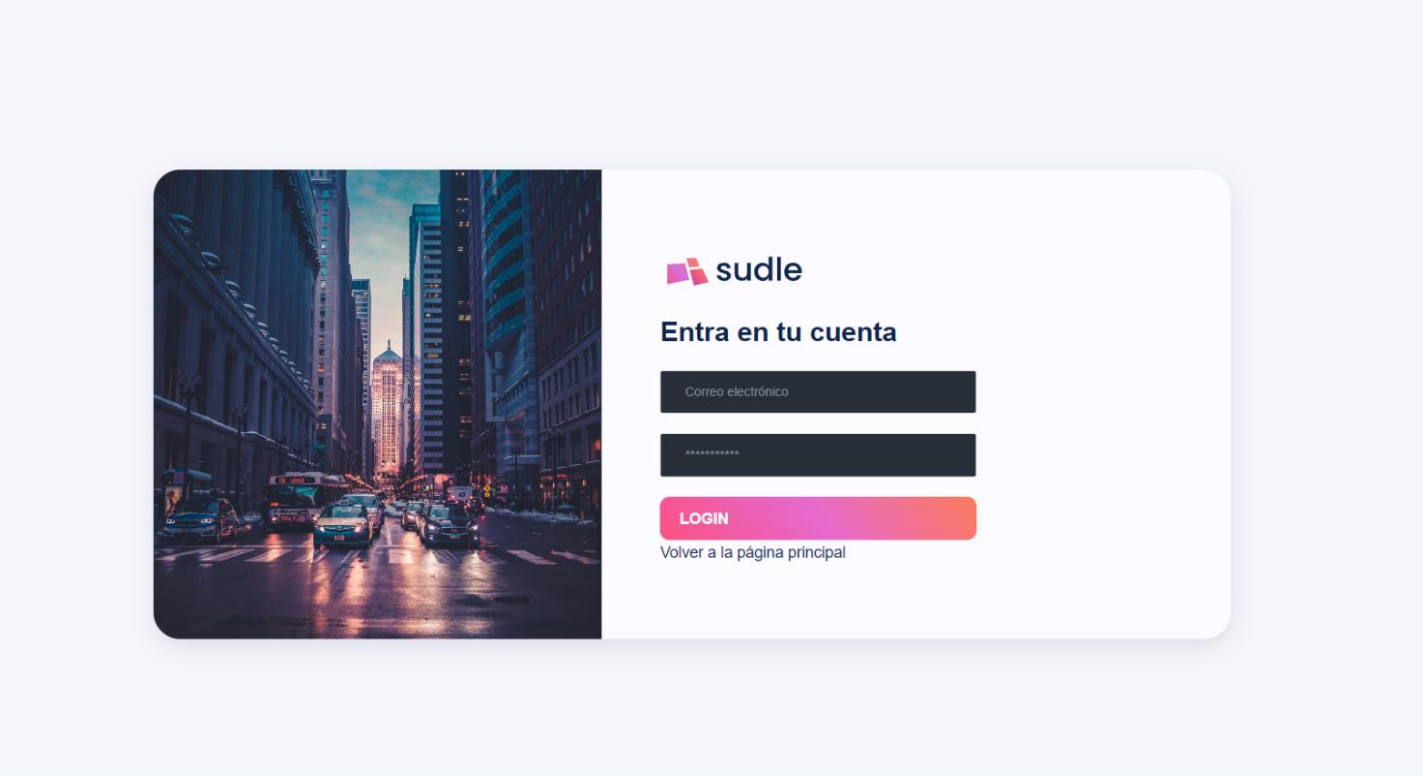


Ilustración 5 Formulario de inicio de sesión

Estos son los formularios de registro en la aplicación e inicio de sesión. En el formulario de registro no hemos requerido muchos datos ya que queremos evitar que el usuario pierda el interés al registrarse en la aplicación.

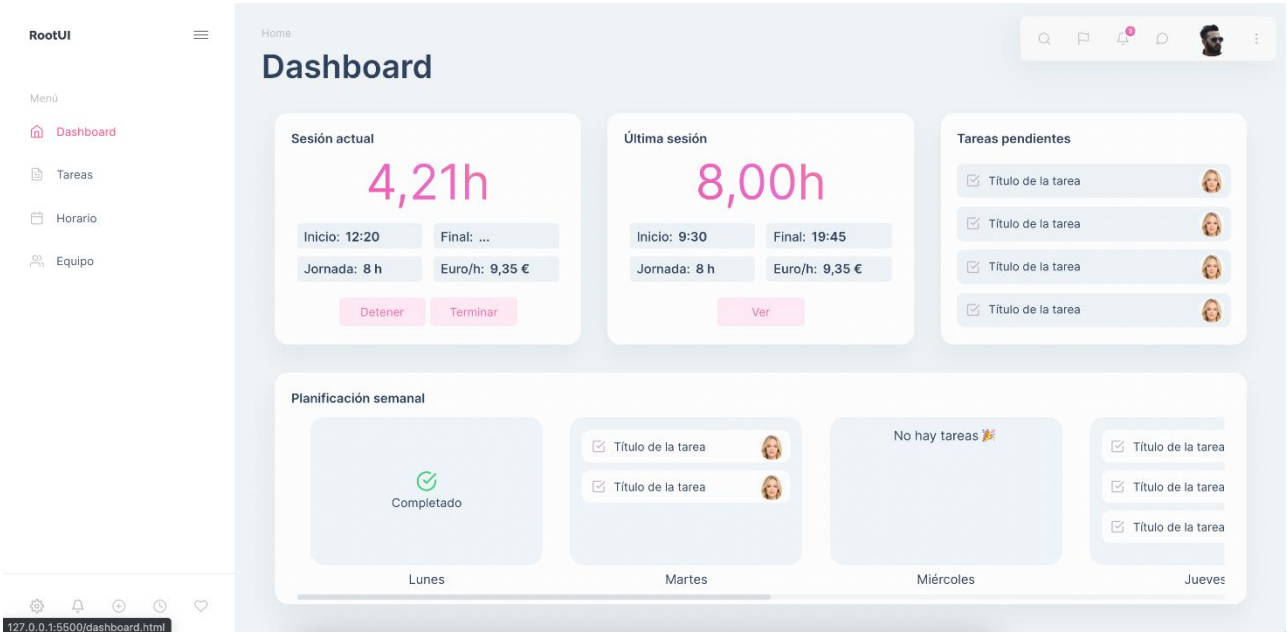


Ilustración 6 Panel de administración del Admin

Esta será la vista que verá el administrador al iniciar sesión.

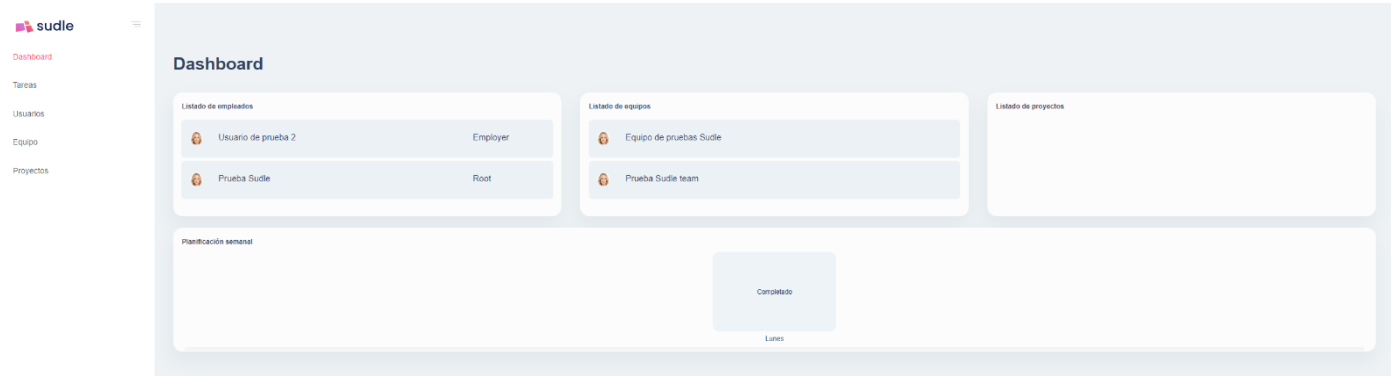


Ilustración 7 Panel de administración de un usuario

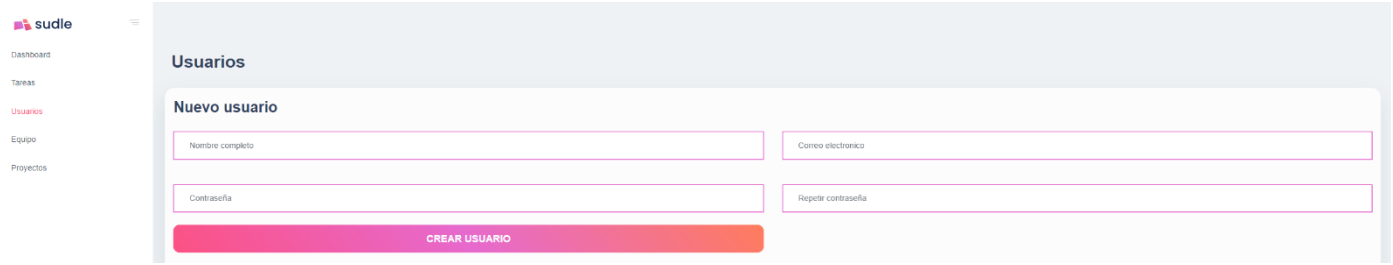


Ilustración 8 Formulario creación Usuario

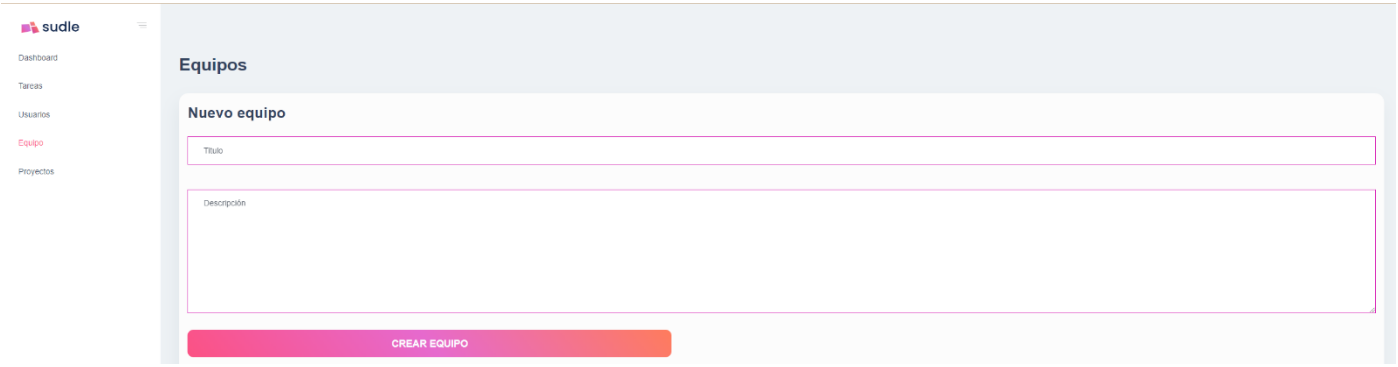


Ilustración 9 Formulario creación equipo

3.2 Datos de prueba

Para hacer las pertinentes pruebas y poder ver la aplicación en funcionamiento se pueden usar los siguientes datos de prueba:

Usuario administrador: email: sudle@sudle.com contraseña: 12345678

Usuario empleado: userprueba@sudle.com contraseña: 12345678

4. DISEÑO DE LA APLICACIÓN

En este apartado vamos a ver los diseños previos que se han realizado antes de empezar con el desarrollo del proyecto.

4.1 Diseño de la base de datos

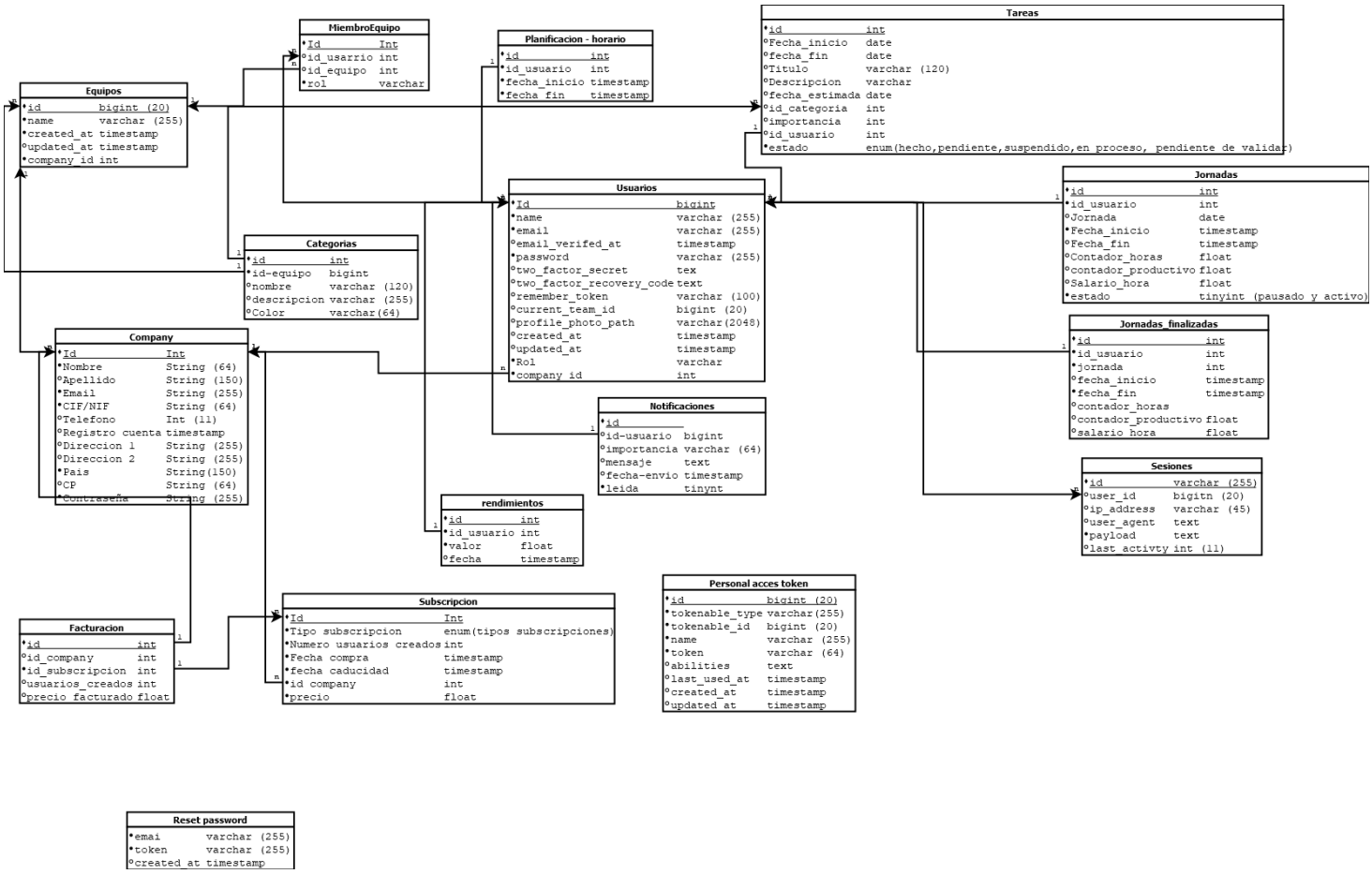


Ilustración 10 Diseño inicial de la base de datos

En un principio este era el diseño de las tablas y de las relacione que se iban a crear en la base de datos, sin embargo debido al tiempo justo que le he podido dedicar al proyecto me vi en la obligación de reducir el modelo para poder abarcar un proyecto un tanto más pequeño.

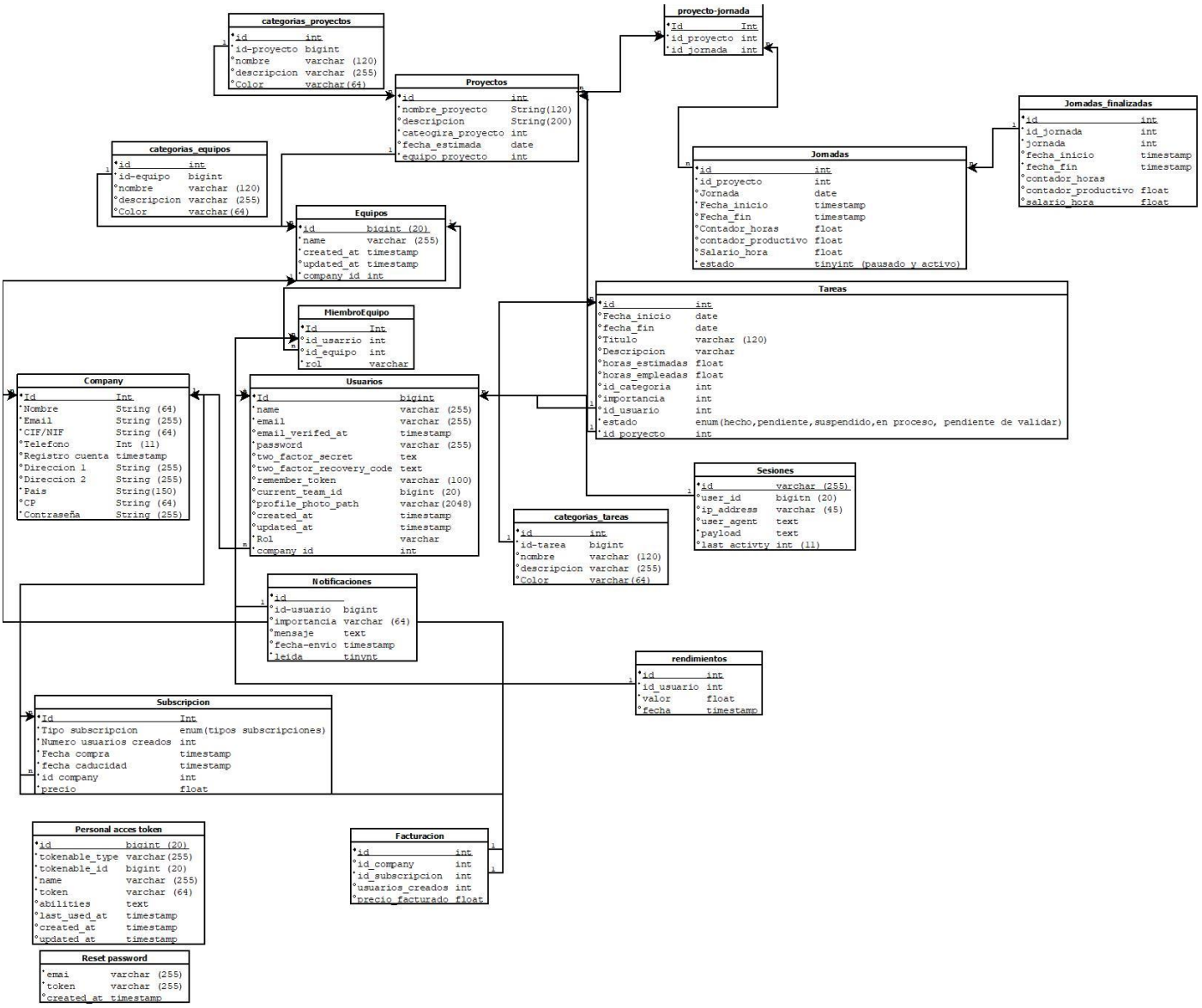


Ilustración 11 Segundo diseño de la base de datos

Una vez me puse a ver por donde podía recortar entidades e intentar optimizar relaciones obtuve un nuevo diseño, sin embargo, este no cumplía con mi expectativa de hacer una aplicación un poco más asequible para poder llegar a la entrega del proyecto, así que tuve que rehacer el diseño otra vez, y está vez si teniendo en cuenta reducir las tablas.

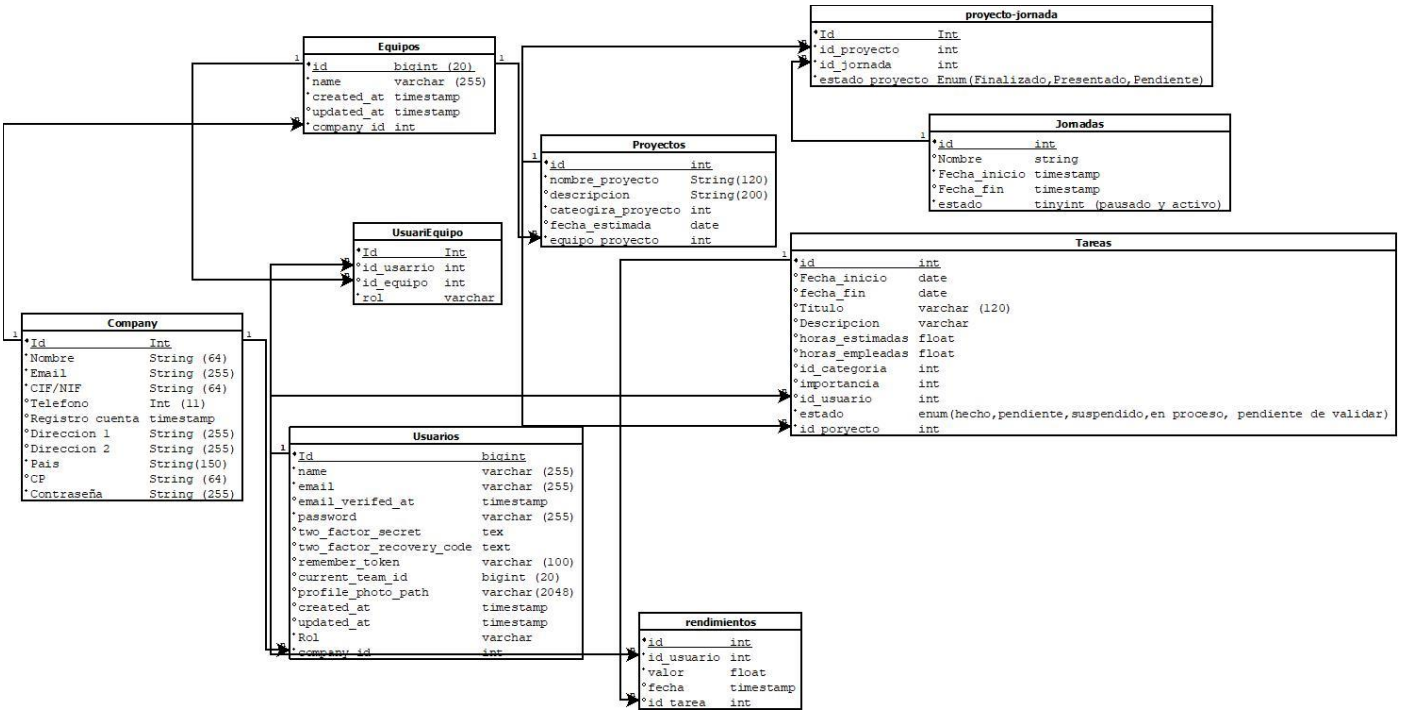


Ilustración 12 Diseño de la base de datos final

Finalmente se llego al diseño optimizado intentando reducir el número de tablas al mínimo sin que repercuta mucho en la funcionalidad principal del proyecto.

4.1.2 Entidades

A continuación, procedo a explicar las tablas, los atributos y las relaciones del diseño final de la base de datos.

El atributo ID en cada entidad es algo que se va a repetir constantemente, por lo que se explicará aquí su función para evitar redundancia. El atributo ID es un campo de tipo integer que se usa para identificar a la entidad y así facilitar su búsqueda por la base de datos, también se usa para relacionar entidades contras, como pertenecías.

Company: Esta entidad se utiliza para almacenar los datos de las empresas que se registran en la aplicación. Sus atributos son:

- Nombre: Atributo de tipo cadena de texto. Es el nombre de la empresa.
- Email: Atributo de tipo cadena de texto. El correo electrónico al que se le enviarán los enlaces de activación de la cuenta, avisos y facturación.
- CIF: Atributo de tipo cadena de texto. Identificación fiscal para la facturación

- Teléfono: Atributo de tipo bigInt. Número de contacto en caso de ser necesario contactar con la empresa cliente
- Registro cuenta: Atributo de tipo fecha. La fecha en la que se registro la empresa. Con este campo se pueden ver la antigüedad de la empresa para ofrecerle ofertas o para saber en qué época del año se registran más empresas.
- Dirección 1 y 2: Campo de tipo cadena de texto. Estos campos son para la facturación, para saber a qué dirección facturar.
- País: Atributo de tipo cadena de texto. Este campo es para la facturación para identificar el país fiscal donde se encuentra la empresa.
- Código postal: Atributo de tipo Integer. Este campo se usa para la facturación, para identificar el domicilio fiscal de la empresa.
- Contraseña: Atributo de tipo cadena de texto. Este campo se usa para proteger ciertas acciones como eliminar la cuenta de la empresa o modificar los datos de esta.

Usuarios: Esta entidad se usa para registrar a los usuarios que forman parte de la empresa. En el momento del registro se crea de forma automática un usuario, los datos de esta entidad serán los que se usen para el inicio de sesión en la aplicación. Sus atributos son:

- Name: Atributo de tipo cadena de texto. Este se usará para identificar a los empleados de la empresa, pero será más útil para los humanos.
- Email: Atributo de tipo cade de texto. Ese atributo se usará como dato de contacto y como primer dato para el inicio de sesión. Una vez se registre el usuario se le enviara un correo con un enlace para que verifique su cuenta.
- Email_verifed_at: Atributo de tipo fecha. Se usará para ver cuando verifico el usuario su email.
- Password: Atributo de tipo cadena de texto. Se usará como segunda credencial en el inicio de sesión en la aplicación, este campo en el momento de ser guardado será cifrado con el algoritmo HASH MD5, de forma que solo el usuario conocerá su contraseña.
- Two_factor_secret, two_factor_recovery_code y remember_tokken: Atributos de tipo texto. Son facilidades que ofrece laravel para la seguridad de los usuarios.

- Current_team_id: Atributo de tipo integer. Este campo sirve para identificar el equipo al que pertenece el usuario.
- Profile_photo_path: Atributo de tipo cadena de texto. Se utiliza para identificar la ruta y el archivo que sirven como foto de perfil del usuario.
- Created_At y updated_at: Atributos de tipo fecha. Son otra facilidad que ofrece Laravel para llevar un control sobre las entidades que se van añadiendo.
- Rol: Campo de tipo cadena de texto, Enum. Este atributo tendrá valores predefinidos que serán, Root, Admin y Employer. Dependiendo del rol que tenga un usuario le podrá hacer unas operaciones u otras.
- Company_id: Atributo de tipo Integer. Se usa para identificar la empresa a la que pertenece dicho usuario.

Equipo: Esta entidad se usa para clasificar los usuarios que hay en la empresa y para plasmar la división de sectores que hay en esta (Marketing, programación, sistemas, etc…)

- Name: Atributo de tipo cadena de texto. Utilizado para identificar los equipos, más útil para los humanos.
- Created_at y updated_At: Atributo de tipo fecha. Es una de las facilidades que ofrece Laravel par aun control de las entidades registradas.
- Compnay_id: Atributo de tipo integer. Utilizado para identificar la empresa a la que pertenece dicho equipo

UsuarioEquipo: Esta entidad es una entidad intermedia usada para relacionar la entidad usuario con equipo, se trata de una relación muchos a muchos, esto quiere decir que un mismo usuario puede estar en varios equipos a la vez y un equipo puede tener a varios usuarios a la vez. Al ser una tabla meramente para relaciones omitimos la explicación de algunos atributos.

- Rol: Atributo de tipo cadena de texto (Enum). Son cadenas de texto previamente definidas que sirven para identificar la función del miembro del equipo en este.

Proyectos: Entidad utilizada para el desarrollo de las actividades de la empresa, por ejemplo, si se trata de una empresa de software aquí se irían creando los productos que quieren desarrollar.

- Nombre de proyecto: Atributo de tipo cadena de texto. Este atributo se usa para identificar al proyecto, más utilizado por los humanos.
- Descripción: Atributo de tipo texto. Se usa para hacer una pequeña descripción de cual es la finalidad del proyecto y cuáles son sus requisitos.
- Fecha estimada: Atributo de tipo fecha. Se usa para saber aproximadamente cuando se debe presentar el proyecto.
- Equipo_proyecto: Atributo de tipo integer. Se usa para saber qué equipo se esta encargando del desarrollo de dicho proyecto.

Jornada (Sprint): Esta entidad se usa para dividir el trabajo de un proyecto en jornadas, estas jornadas pueden durar desde semanas hasta meses. Con esto lo que se pretende es dividir el trabajo del proyecto y ver de forma más exacta como se va progresando.

- Nombre: Atributo de tipo cadena de texto. Susa para identificar la jornada.
- Fecha de inicio: Atributo de tipo fecha. Se usa para saber cuándo empieza.
- Fecha de fin: Atributo de tipo fecha. Se usa para saber cuándo termina.
- Estado: Atributo de tipo cadena de texto (Enum). Son cadenas de texto ya predefinidas que sirven para saber cual es el estado de la jornada. Pausado o Activo.

Proyecto-jornada: Esta entidad es parecida a la de UsuarioEquipo, es una entidad de relación ya que una Jornada puede tener varios proyectos y un proyecto puede estar en varias jornadas. Los atributos que ya han sido explicados serán omitidos.

- Estado_proyecto: Atributo de tipo cadena de texto (Enum). Se utiliza para indicar el estado del proyecto con respecto a la jornada.

Tareas: Entidad que sirve para dividir el proyecto y sus funcionalidades en tareas, también puede usarse para registrar la actividad realizada en la jornada laboral.

- Fecha de inicio: Atributo de tipo fecha. Se usa para saber cuando se empezó a hacer dicha tarea.
- Fecha de fin: Atributo de tipo fecha. Se usa para saber cuando se finalizó dicha tarea.
- Titulo de la tarea: Atributo de tipo texto. Se usa para para identificar la tarea o la acción que se está haciendo.
- Descripción: Atributo de tipo texto. Se usa para especificar los detalles de la tarea.
- Horas_estimadas: Atributo de tipo double (decimales). Se usa para estimar el tiempo que debería tardarse en realizar dicha entidad.
- Horas_empleadas: Atributo de tipo double (decimales). Se usa para valorar las horas empleadas. Este atributo se calcula de forma automática al introducir la fecha de fin. Para poder modificar dicho atributo se debe realizar desde la fecha fin.
- Importancia: Atributo de tipo Integer. Se usa para darle una prioridad sobre otras tareas a esta.
- Id_usuario: Atributo de tipo Integer. Se usa para saber que usuario tiene asignada dicha tarea.
- Estado: Atributo de tipo cadena de texto (Enum). Se usa para saber el estado en el que está dicha tarea, sus estados son Hecha, pendiente, suspendida, en proceso, pendiente de validar.

Rendimientos: Esta es la entidad donde se guardan los cálculos obtenidos del rendimiento de las tareas de los usuarios, esta entidad se rellena de forma automática al finalizar una tarea, por lo que los usuarios no tienen control directo sobre ella.

- Id_usuario: Atributo de tipo Integer. Se usa para identificar el usuario al que está asignado dicho rendimiento.
- Valor: atributo de tipo double. Dependiendo de si el usuario se ha excedido en las horas asignadas a una tarea este valor saldrá en negativo o positivo. A más tareas tenga con valores positivos mejor será el rendimiento del usuario.
- Fecha: Atributo de tipo fecha. Se usar para saber el día en el que se generó dicho rendimiento.
- Id_tarea: Atributo de tipo Integer. Se usar para saber la tarea a la que está asignada dicho rendimiento.

4.1.2.1 Relaciones de las entidades

Ahora vamos a tratar de que forma se están relacionando las entidades.

- La entidad company es la entidad padre de la cual cuelgan las entidades Equipos y Usuarios, es una relación uno a uno, es decir un usuario solo puede pertenecer a una compnay a la vez y lo mismo con un equipo.
- Las entidades usuarios y equipos están relacionadas con una tabla intermedia, esto se conoce como relación muchos a muchos. Un usuario puede estar en varios equipos a la vez y un equipo puede tener varios usuarios a la vez.
- La entidad Proyecto está relacionada con la entidad Equipos, con una relación uno a uno. Un equipo solo puede tener un proyecto a la vez y un proyecto solo puede pertenecer a un equipo a la vez.
- La entidad proyecto está relacionada con la entidad Jornada, con una relación muchos a muchos, una jornada puede tener más de un proyecto y un proyecto puede estar en más de una jornada.
- La entidad tareas está relacionada con las entidades usuario y proyecto, con una relación uno a uno. Un usuario solo puede tener una tarea a la vez y una tarea solo puede pertenecer a un equipo a la vez.
- La entidad rendimientos está relacionada con la entidad tareas y usuarios.

4.2 UML de las clases utilizadas en Laravel

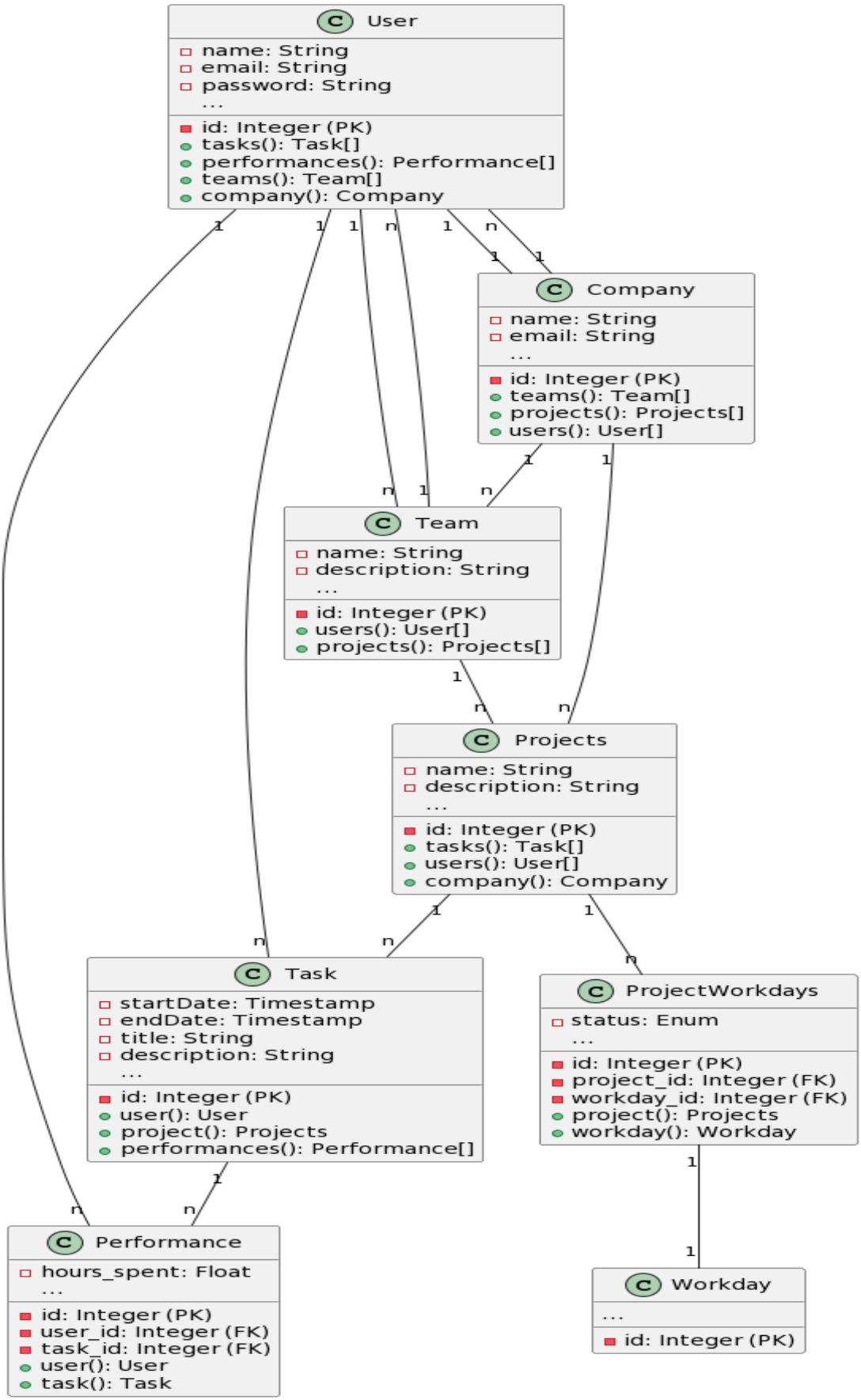


Ilustración 13 Sudle UML

5. IMPLEMENTACIÓN

En este capítulo veremos cómo se ha implementado cada una de las partes de la aplicación.

5.1 Metodología de trabajo y herramientas de desarrollo

Dadas las tres partes de este proyecto se va a dividir este punto en sus pertinentes partes y se explicará cómo se ha hecho.

5.1.1 Despliegue del servidor

Recapitulando los requisitos que se habían definido para el servidor, eran los siguientes:

- Servicio de host, como puede ser Apache o Nginx
- Servidor DNS para la resolución de nombres
- Servidor FTP para almacenar los archivos de la aplicación
- Servidor SMTP para los envíos de correos
- Servidor de base de datos para almacenar datos de la aplicación
- CPanel para poder gestionar desde un único sitio el resto de los servicios mencionados a excepción de la base de datos
- Sistema gestor de bases de datos

5.1.2 Instalación Nginx

Así que vamos a empezar por la implementación del servicio Nginx. Antes que nada, vamos a explicar porque se ha escogido Nginx y no apache. En un principio se pensó en utilizar Apache ya que era un servicio visto en clases y sobre el cual ya se tenía conocimientos. Sin embargo, la configuración del servidor no era compatible con Apache y no se conseguía ver correctamente la web. Se busco otra solución y se encontró Nginx, que es un servicio muy parecido a Apache.

Al tener como sistema operativo Ubuntu server, este no posee interfaz gráfica por lo que la instalación de servicios se hace mediante comandos por el terminal.

Sus pasos fueron:

1. Actualizar las librerías y programas instalados con
    a. Sudo apt update

2. Luego se instala Ngnix
    a. Sudo apt install Nginx
3. Luego se toca configurar el firewall
    a. Sudo ufw app list
4. Se configura una lista de perfiles de aplicaciones como el siguiente
    a. Output

       Available applications:

       Nginx Full

       Nginx HTTP

       Nginx HTTPS

       OpenSSH
5. Luego para habilitar dichos perfiles se hizo con
    a. Sudo ufw allow ‘Nginx HTTP’

Una vez cargada dicha configuración ya se podían alojar sitios web en nuestro servidor

5.1.3 Instalación servicio DNS Bind9

Para poder hacer la resolución de nombres tanto ordinaria como inversa de nuestro servidor se tenía que instalar un servidor DNS, y para ello se instaló Bind9:

1. Para su instalación
2. sudo apt-get install bind9 bind9utils bind9-doc
3. Luego hay que configurar la IP para versión 4
    a. sudo nano /etc/default/bind9
    b. y al final del archivo añadir la línea ‘OPTIONS=”-u bind -4’
4. Luego toca reiniciar el servicio
    a. Sudo systemctl restart bind9
5. Después toca configurar el archivo local
    a. Sudo nano /etc/bind/named.conf.local

zone " http://vmi621282.contaboserver.net/" {

```
    type master;
    file "/etc/bind/zones/db. http://vmi621282.contaboserver.net/"; # zone file path
    allow-transfer { 173.212.205.170; };          # ns2 private IP address - secondary
};
```

Por recomendación de compañeros y profesores se ha decidido evitar dar detalles tan extensos de la instalación, para ver los pasos seguidos se puede visitar los enlaces de la bibliografía.

5.1.4 Instalar servidor FTP

Para la transferencia de archivos entre el equipo de desarrollo y el servidor donde se aloja la aplicación es necesario instalar un servidor FTP. Para ello se han seguido los siguientes pasos:

1. Instalación vsftpd
    a. Sudo apt-get install vsftpd
2. Luego hay que habilitar el tráifco en los puertos 20 y 21 para ello
    a. Sudo ufw allow 20/tcp
    b. Sudo ufw allow 21/tcp
3. Luego hay que configurar los accesos

    Como a partir de aquí es tocar archivos de texto y reiniciar el servidor para que cargue dicha configuración no se omiten estos pasos.

5.1.5 Instalar servidor SMPT POSTFIX

Para el envío de correos de verificación y notificaciones de la aplicación es necesario tener instalado un servidor SMTP, para ello se ha instalado el servidor Postfix.

1. Para instala postfix
    a. Es necesario instalar el paquete mailutils para agrupar postfix con programas complementarios
        i. Sudo apt install mailutils
2. Luego hay que ejecutar el comando dpkg para lanzar la configuración realizada con el anterior comando
    a. Sudo dpkg-reconfigure postfix

El resto de los pasos son configuraciones de archivos de texto por lo que se han omitido.

5.1.6 Instalación CPanel

Un Cpanel es una aplicación que permite centraliza los servicios de un equipo y poder gestionarlos desde una cómoda interfaz gráfica, para nuestra apliación se decidió instalar Vesta Cpanel. Para ello se siguieron los siguientes pasos:

1. Hay que ejecutar el siguiente comando para instalar vesta
    a. Wget http://vestacp.com/pub/vst-install.sh
2. Una vez descargado hay que cambiar los permisos del archivo
    a. Chmod 755 vst-install.sh
3. Luego se ejecuta la instalación
    a. Bash vst-install.sh
4. A través del terminal se piden un par de datos básicos pero una vez los rellenemos se habrá instalado y configurado nuestro CPanel

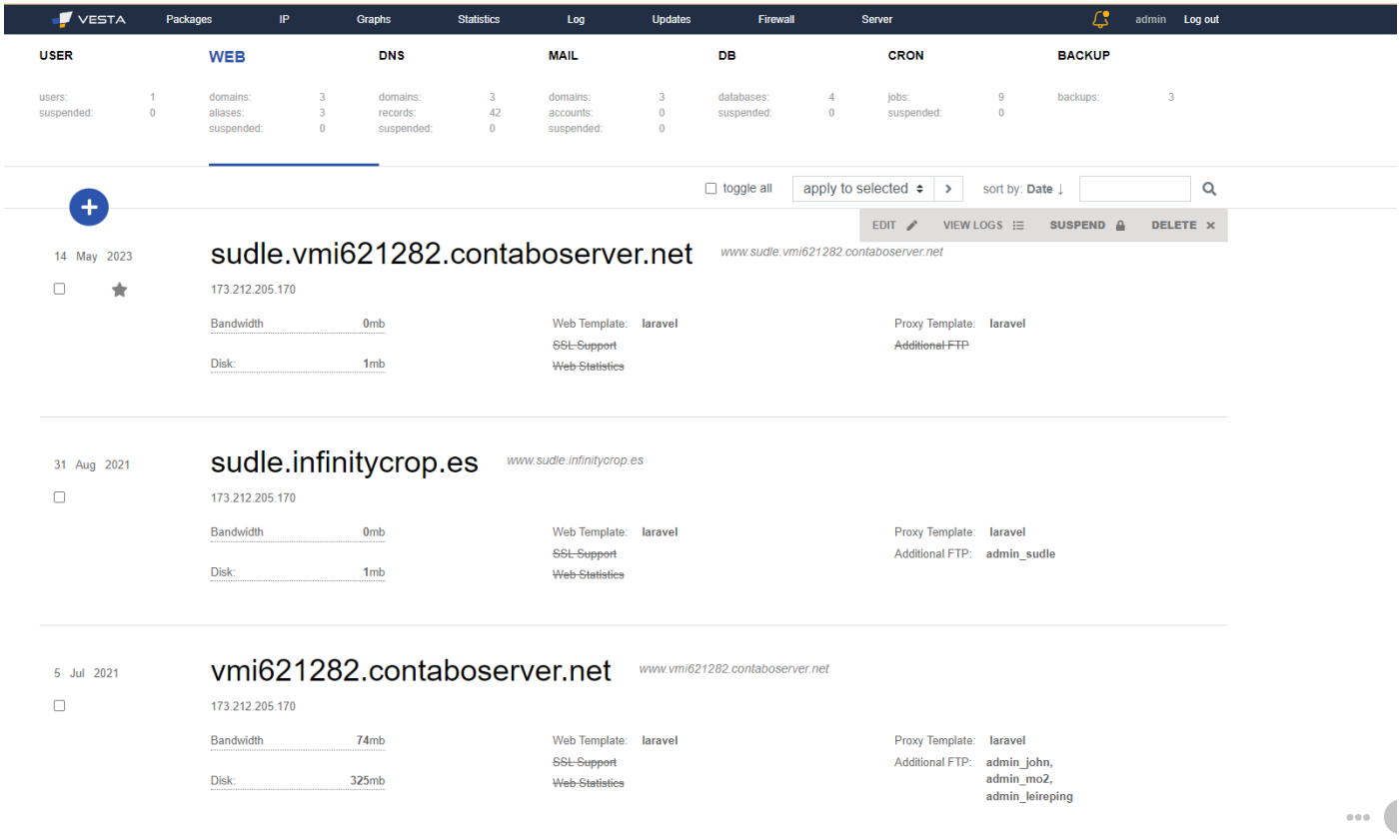


Ilustración 14 Vesta panel de control

Una de las ventajas de tener este servicio instalado es que servicios como bases de datos o backups ya vienen integrados y pueden ser gestionados desde aquí, por lo que con esto quedaría abordado el resto de los requisitos que se esperaba en el servidor.

5.1.2 Implementación del backend

Para el desarrollo de la API Res hay que instalar Laravel en el servidor, para ello se ejecutaron los siguientes comandos:

1. Primero se instala PHP ya que es el lenguaje con el que trabaja Laravel
    a. sudo apt install php7.2-common php7.2-cli php7.2-gd php7.2-mysql php7.2-curl php7.2-intl php7.2-mbstring php7.2-bcmath php7.2-imap php7.2-xml php7.2-zip
2. Luego hay que instalar el gestor de paquetes composer
    a. curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=composer
3. Luego ya se puede crear un proyecto de laravel con el siguiente comando
    a. composer create-project --prefer-dist laravel/sudle

Una vez instalado Laravel y creado nuestro proyecto ya se puede empezar con el desarrollo de la API Rest

5.1.3 Implementación del frontend

Para desarrollar nuestra aplicación web vamos a instalar VUE.js, sin embargo, al igual que pasaba con Laravel hay que instalar previamente otros servicios.

1. Se empieza instalando NODE.js
    a. Sudo apt install nodejs
2. Luego para que se actualice
    a. Sudo apt install npm
3. Luego instalamos vue
    a. Npm install vue
4. Cremos la aplicación en veu
    a. Vue créate Sudle

5. Para poner en marcha y compilar los componentes
    a. Npm run serve

5.2 Estructura de la aplicación

En este apartado se verán los directorios de cada aplicación, como están estructurados y porque, sin embargo, antes de seguir con esto vamos a hablar del concepto MVC en los que se basan los marcos de trabajo Laravel y VUE.

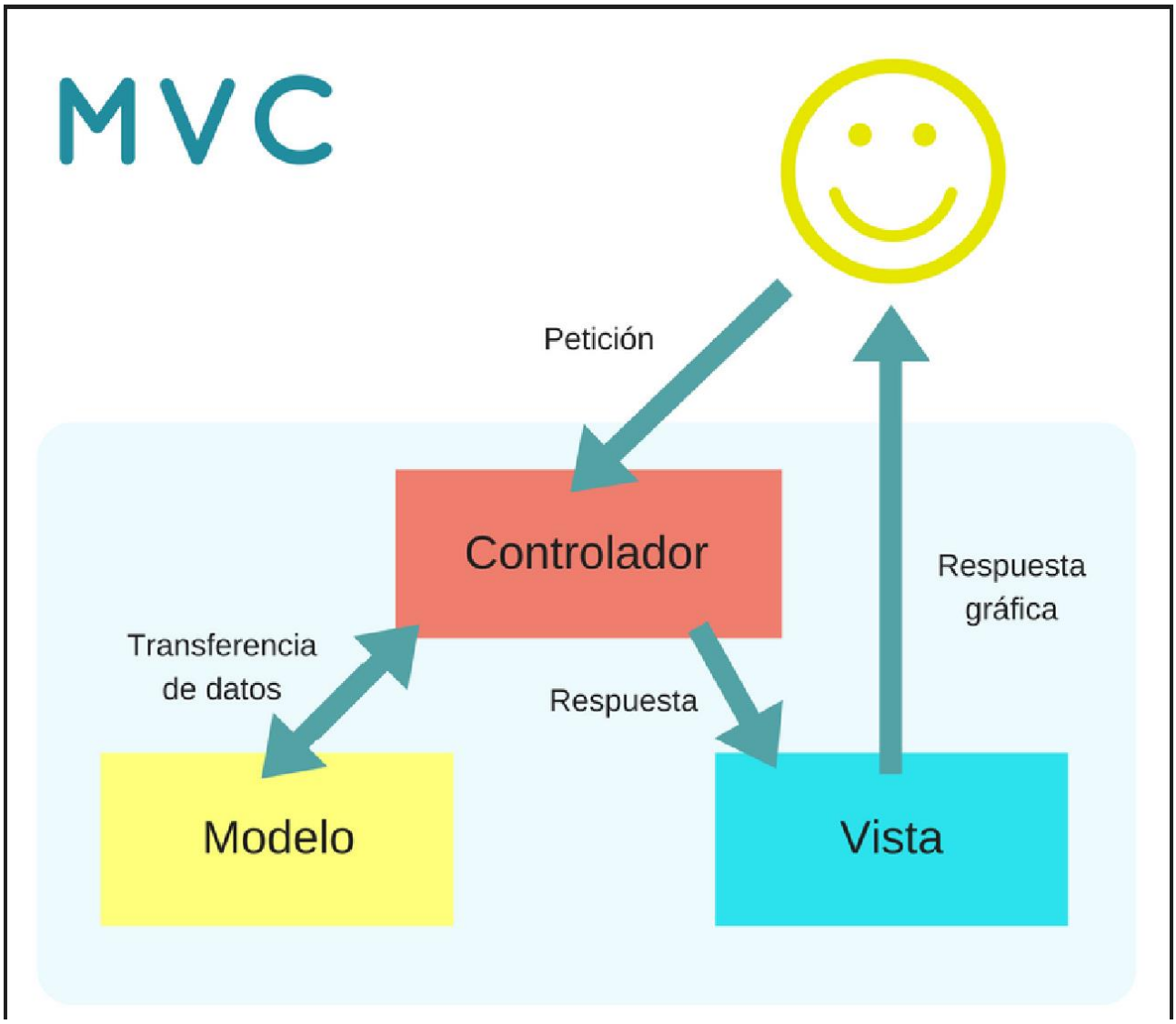


Ilustración 15 Modelo MVC

El modelo MVC es un enfoque de diseño utilizado en la programación de aplicaciones para separar diferentes componentes y responsabilidades de una manera organizada. Se compone de tres partes principales: Modelo (Model), Vista (View) y Controlador (Controller). Laravel y Vue.js son dos tecnologías que trabajan juntas muy bien siguiendo este modelo.

Modelo (Model): Representa los datos y la lógica de negocio.

Manipula la información de la base de datos y se asegura de que los datos sean consistentes y precisos.

En Laravel, los modelos se definen como clases que interactúan con la base de datos a través de Eloquent ORM.

En Vue.js, el modelo se refiere a los datos que se utilizan para representar y manipular la interfaz de usuario.

Vista (View):

Presenta los datos al usuario y maneja la interfaz de usuario.

Muestra la información al usuario de manera legible y atractiva.

En Laravel, las vistas son archivos de plantillas que se combinan con datos para generar el contenido que se muestra al usuario.

En Vue.js, las vistas se crean con componentes que definen cómo se ve y se comporta la interfaz de usuario.

Controlador (Controller):

Maneja la lógica y la interacción entre el modelo y la vista.

Recibe las solicitudes del usuario, interactúa con el modelo para obtener los datos necesarios y luego devuelve la respuesta a la vista.

En Laravel, los controladores son clases que definen las acciones que se deben realizar en respuesta a las solicitudes del usuario.

En Vue.js, el controlador se refiere a los métodos y la lógica que se ejecutan en los componentes para manejar eventos y actualizar datos.

Laravel y Vue.js implementan el modelo MVC de la siguiente manera:

Laravel:

El modelo se implementa utilizando Eloquent ORM y se encarga de las operaciones de la base de datos. Las vistas se crean usando plantillas de Blade y se llenan con datos antes de ser enviadas al navegador. Los controladores manejan las rutas y las solicitudes del usuario, interactuando con el modelo y preparando los datos para las vistas.

Vue.js: El modelo se representa mediante los datos que se usan en los componentes Vue para mostrar y manipular la interfaz de usuario.

Las vistas se crean utilizando componentes Vue que definen cómo se muestra la información y cómo responde a las interacciones del usuario.

Los controladores en Vue.js son los métodos definidos en los componentes que manejan eventos, actualizan datos y realizan acciones en respuesta a la interacción del usuario.

5.2.1 Estructura backend

Una vez se ha visto en que consiste el modelo MCV podemos ver y entender la estructura de los directorios de un proyecto en Laravel.

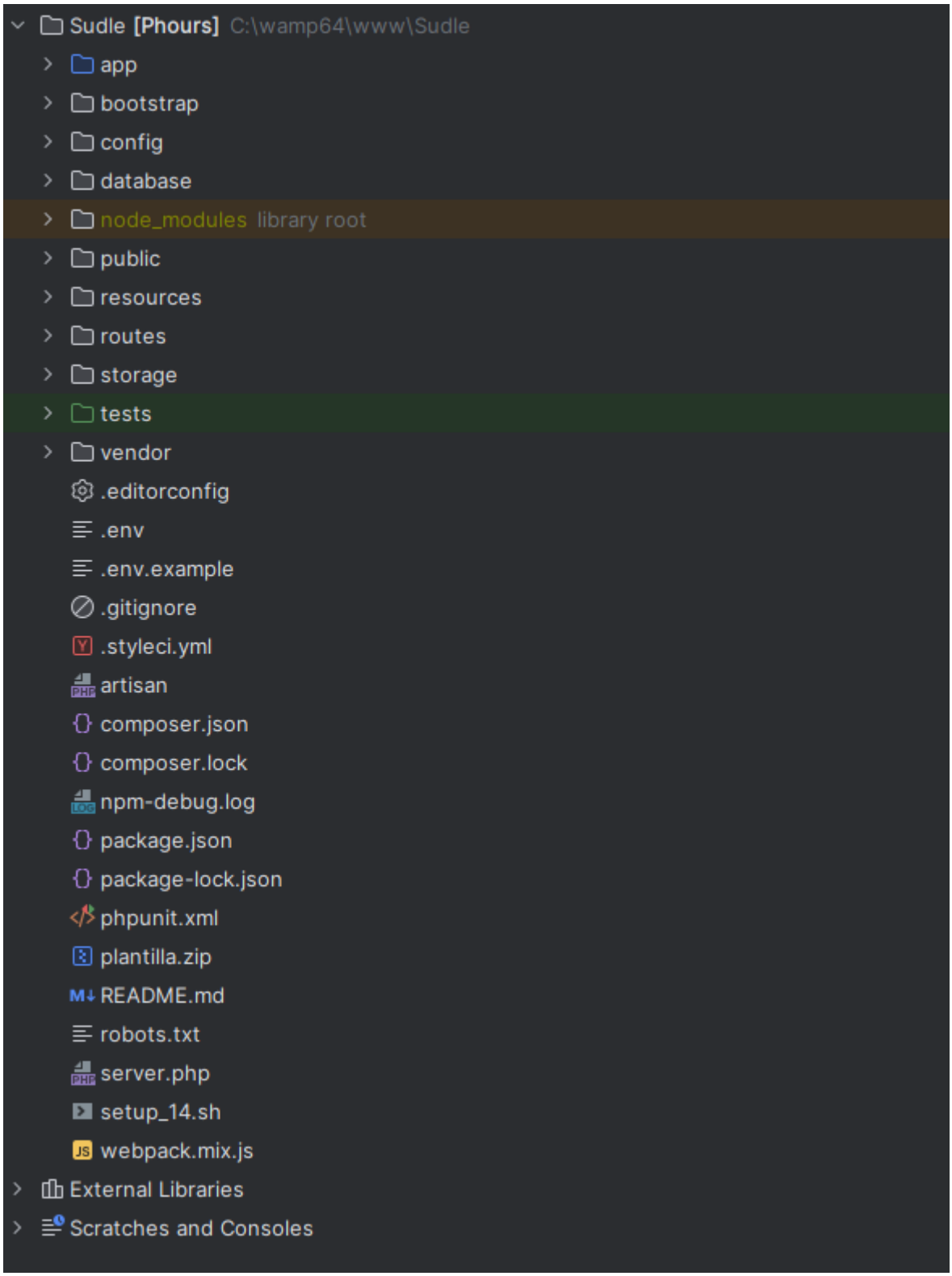


Ilustración 16 Estructura de directorios Laravel

Este es el listado de directorios. Ahora se explicará el contenido que hay dentro de ellos, solo de los que son relevantes para él proyecto.

Dentro del directorio App es donde se guardan, los controladores, los modelos y los midleware, etc.

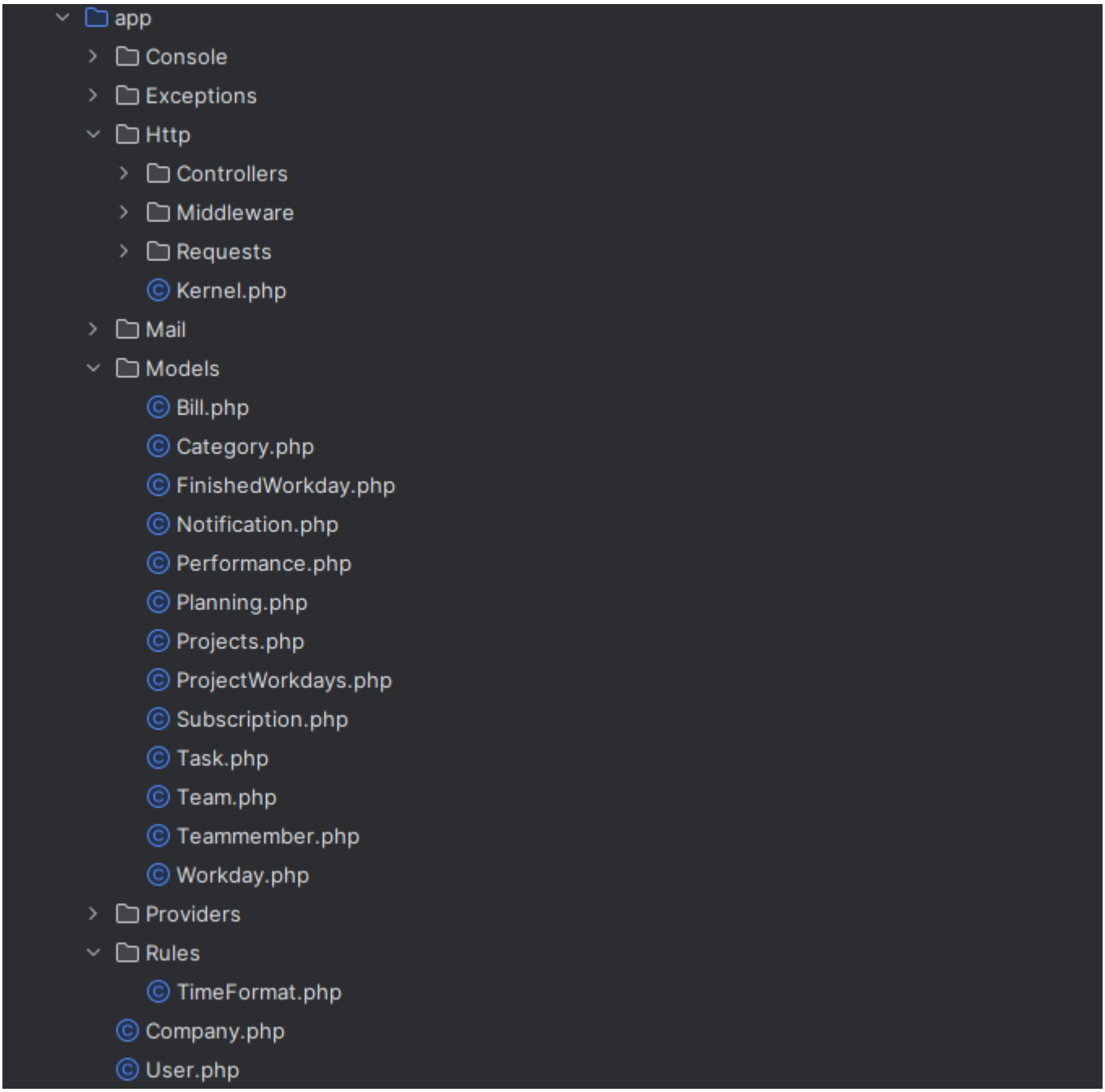


Ilustración 17 Estructura de archivos de la carpeta App de Laravel

Dentro de la carpeta Models están los modelos, que serían las clases que contienen las lógicas de extracción y procesado de datos. Las carpetas Mail, Providers y Rules contienen lógicas para la interacción de los controladores con los modelos. Por ejemplo, el envío de un email cuando se ha registrado un usuario, o la validación de los datos de un formulario.

Los Middlewares son filtros o capas intermedias que pueden ser aplicados en las peticiones HTTP antes de que lleguen a las rutas solicitadas.

Si se abre la carpeta Conrollers

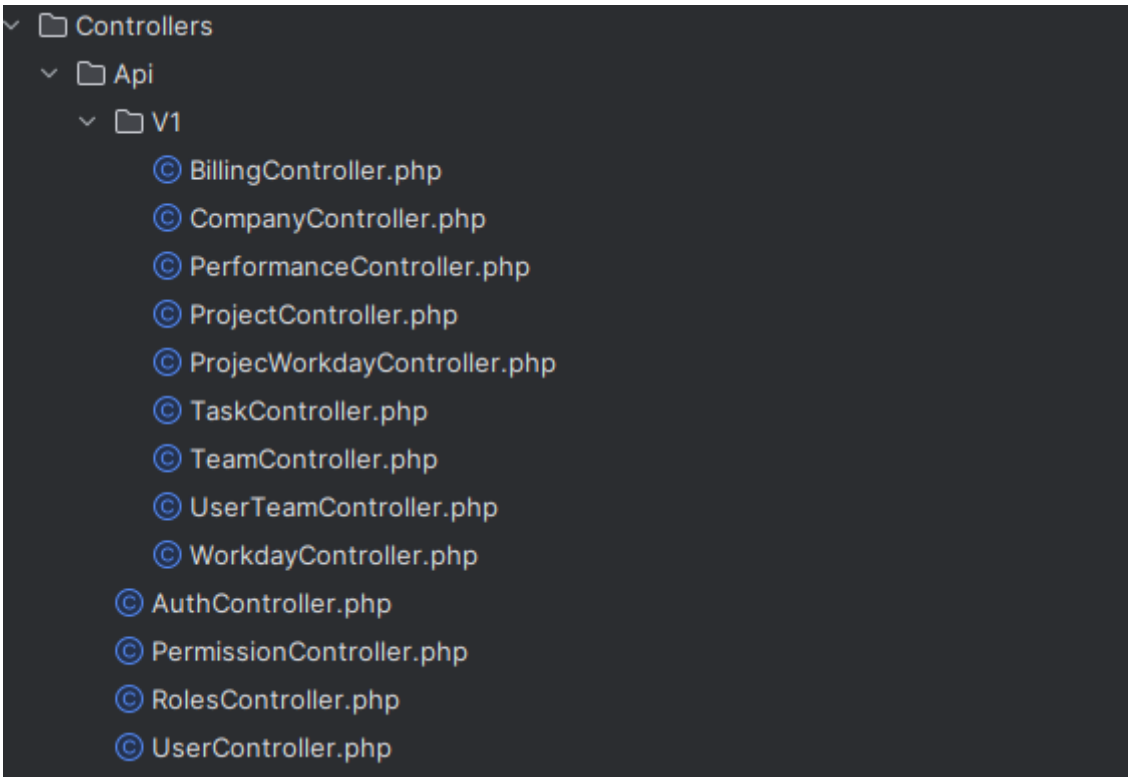


Ilustración 18 Estructura de archivos de la carpeta Controllers de Laravel

Tenemos el directorio API, en el es donde se van a ir metiendo todos los controladores para atender las peticiones recibidas por la aplicación web.

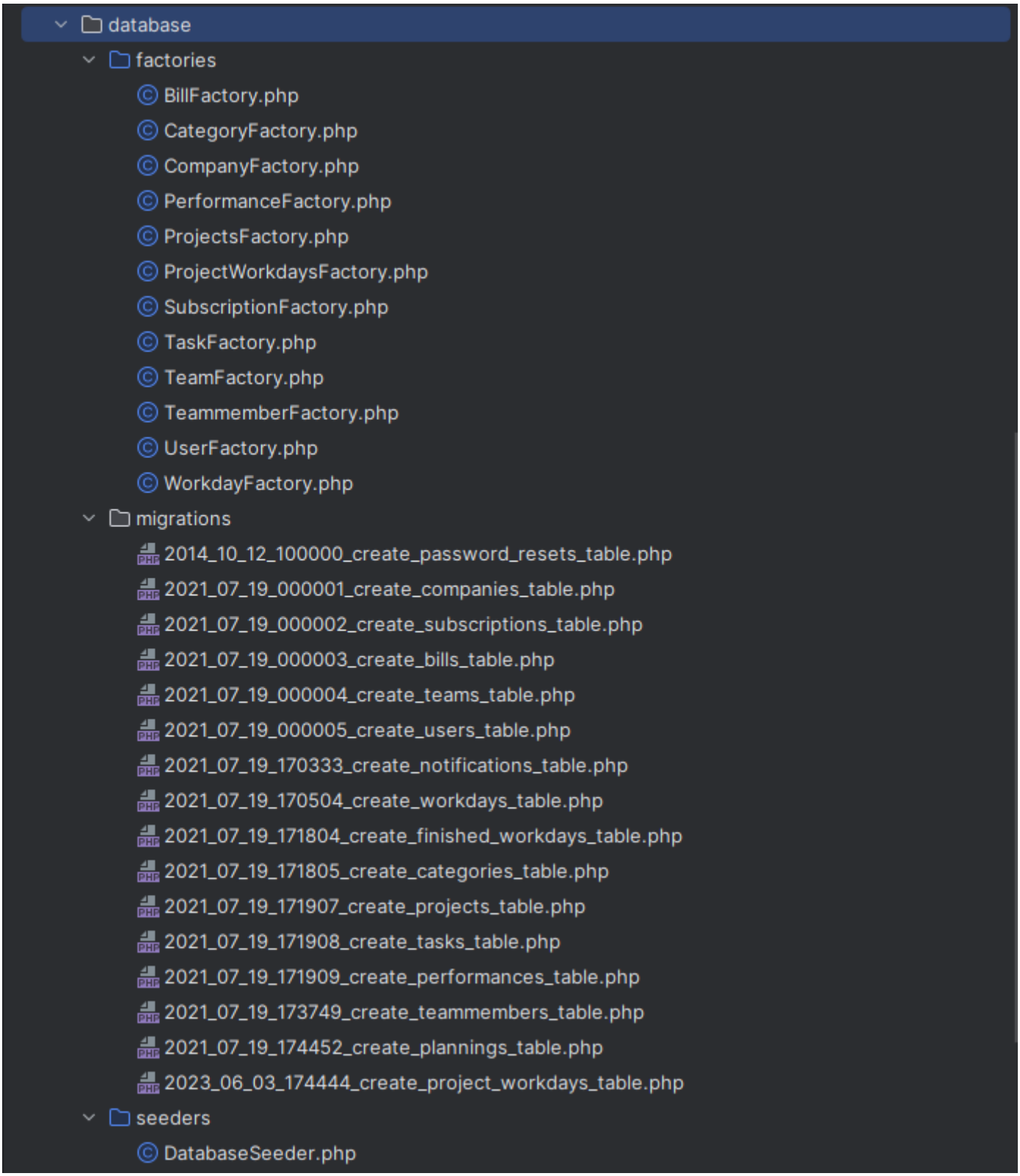


Ilustración 19 Estructura de directorios database Laravel

La carpeta database es donde se almacenan todos los archivos que tienen que ver con la base de datos. Las migraciones son archivos que están compuestos por sentencias de código que al ejecutarse las convierte en sentencias SQL.

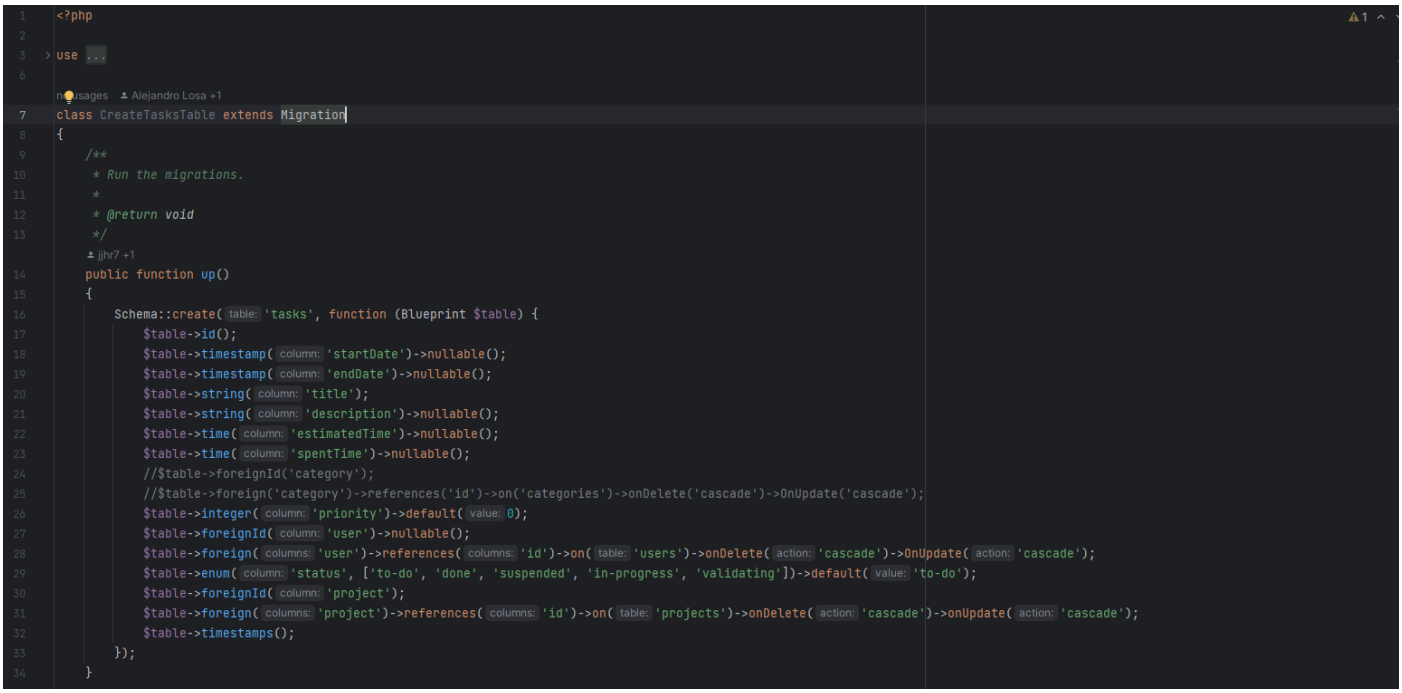


Ilustración 20 Ejemplo de migración

En la imagen se puede ver un ejemplo de migración, más en concreto de las sentencias que hay en el para luego acabar teniendo una base de datos.

También están los archivos facotries, que son los encargados de generar datos de prueba. En ellos mediante sentencias con ciertos parámetros son capaces de insertar datos de forma automática una vez se monte la base de datos.

Y por último están los archivos sedes, que son archivos que al igual que las factories generan datos, sin embargo, estos no son datos aleatorios, estos son datos proporcionados por nosotros.

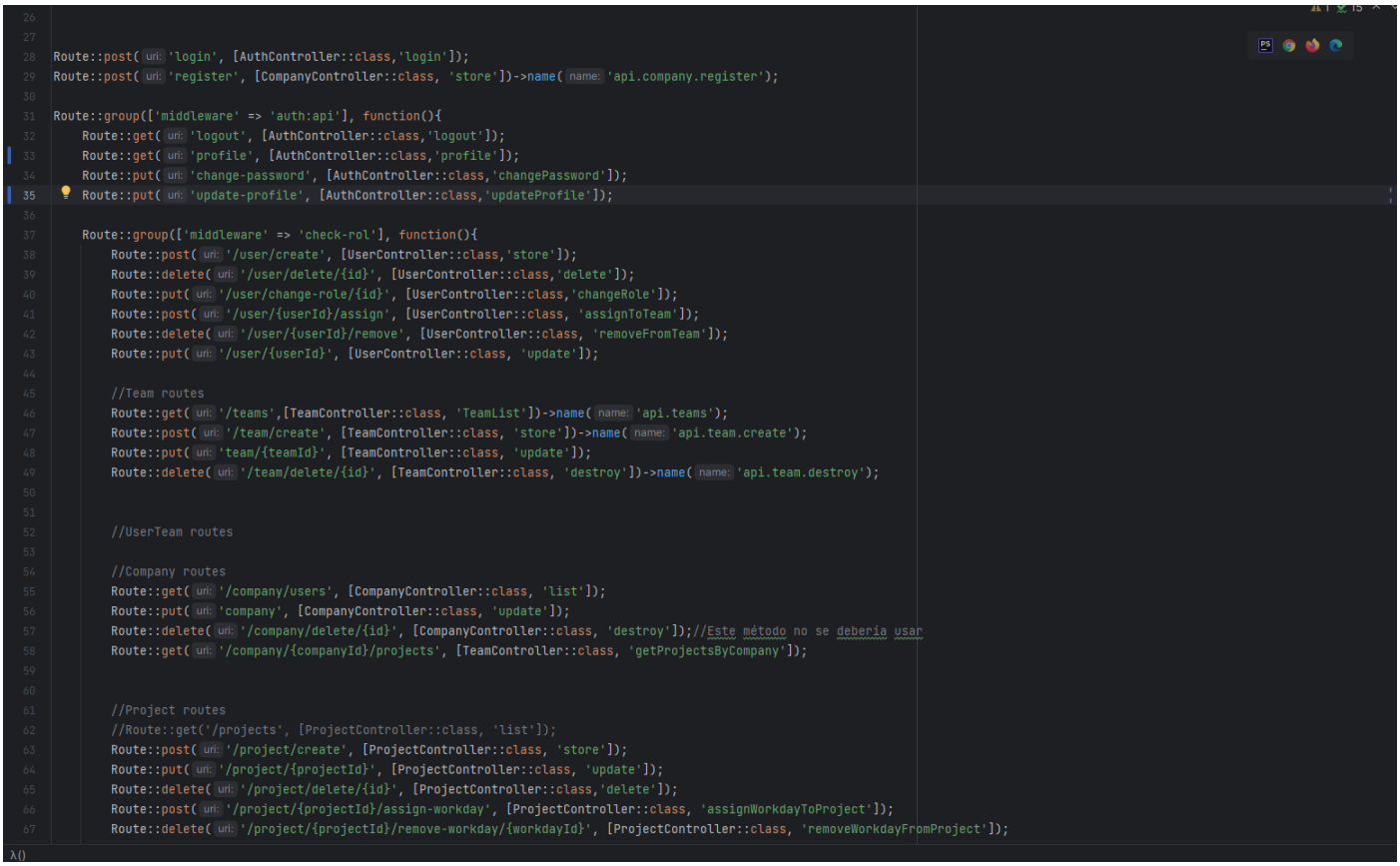
Mediante estos dos tipos de archivos podemos generar información en nuestra base de datos sin necesidad de tener que ir rellenando formularios e ir insertando entidades una a una.

Ilustración 21 Directorio public de Laravel

Dentro de la carpeta public, dentro de views es donde se guardarían las vistas de nuestra aplicación, sin embargo, como hemos delegado toda la parte visual a otro marco de trabajo aquí solo tenemos las vistas nativas que nos ofrece el Laravel al crear el proyecto.

Ilustración 22 Directorio Routes

Para finalizar veremos uno de los directorios más importantes de Laravel, el directorio de rutas. Existen diferentes archivos, si se hubiera montado toda la aplicación aquí usaríamos el archivo web.php para las rutas de web y este ya haría todas las llamadas a los controladores y modelos, sin embargo como se ha decidido hacer una API Rest, solo se usa el archivo api.php. La funcionalidad es la misma sin embargo aquí las rutas no apuntan a ninguna vista.



```php
Route::post( uri: 'login', [AuthController::class,'login']);
Route::post( uri: 'register', [CompanyController::class, 'store'])->name( name: 'api.company.register');

Route::group(['middleware' => 'auth:api'], function(){
    Route::get( uri: 'logout', [AuthController::class,'logout']);
    Route::get( uri: 'profile', [AuthController::class,'profile']);
    Route::put( uri: 'change-password', [AuthController::class,'changePassword']);
    Route::put( uri: 'update-profile', [AuthController::class,'updateProfile']);

    Route::group(['middleware' => 'check-rol'], function(){
        Route::post( uri: '/user/create', [UserController::class,'store']);
        Route::delete( uri: '/user/delete/{id}', [UserController::class,'delete']);
        Route::put( uri: '/user/change-role/{id}', [UserController::class,'changeRole']);
        Route::post( uri: '/user/{userId}/assign', [UserController::class, 'assignToTeam']);
        Route::delete( uri: '/user/{userId}/remove', [UserController::class, 'removeFromTeam']);
        Route::put( uri: '/user/{userId}', [UserController::class, 'update']);

        //Team routes
        Route::get( uri: '/teams',[TeamController::class, 'TeamList'])->name( name: 'api.teams');
        Route::post( uri: '/team/create', [TeamController::class, 'store'])->name( name: 'api.team.create');
        Route::put( uri: 'team/{teamId}', [TeamController::class, 'update']);
        Route::delete( uri: '/team/delete/{id}', [TeamController::class, 'destroy'])->name( name: 'api.team.destroy');


        //UserTeam routes

        //Company routes
        Route::get( uri: '/company/users', [CompanyController::class, 'list']);
        Route::put( uri: 'company', [CompanyController::class, 'update']);
        Route::delete( uri: '/company/delete/{id}', [CompanyController::class, 'destroy']);//Este método no se debería usar
        Route::get( uri: '/company/{companyId}/projects', [TeamController::class, 'getProjectsByCompany']);


        //Project routes
        //Route::get('/projects', [ProjectController::class, 'list']);
        Route::post( uri: '/project/create', [ProjectController::class, 'store']);
        Route::put( uri: '/project/{projectId}', [ProjectController::class, 'update']);
        Route::delete( uri: '/project/delete/{id}', [ProjectController::class,'delete']);
        Route::post( uri: '/project/{projectId}/assign-workday', [ProjectController::class, 'assignWorkdayToProject']);
        Route::delete( uri: '/project/{projectId}/remove-workday/{workdayId}', [ProjectController::class, 'removeWorkdayFromProject']);
```

Ilustración 23 Archivo api.php

Este es el contenido que más o menos tiene nuestro archivo api.php. Más adelante se verán todas las peticiones que la API es capaz de atender.

Y por último, pero no menos importante el archivo. env que fue listado en la primera imagen, allí es donde se guarda toda la información sensible que pueda comprometerse, contraseñas, direcciones ip, tokens de seguridad, etc.

5.2.2 Estructura frontend

Ahora vamos a ver la estructura de archivos que tiene VUE.js

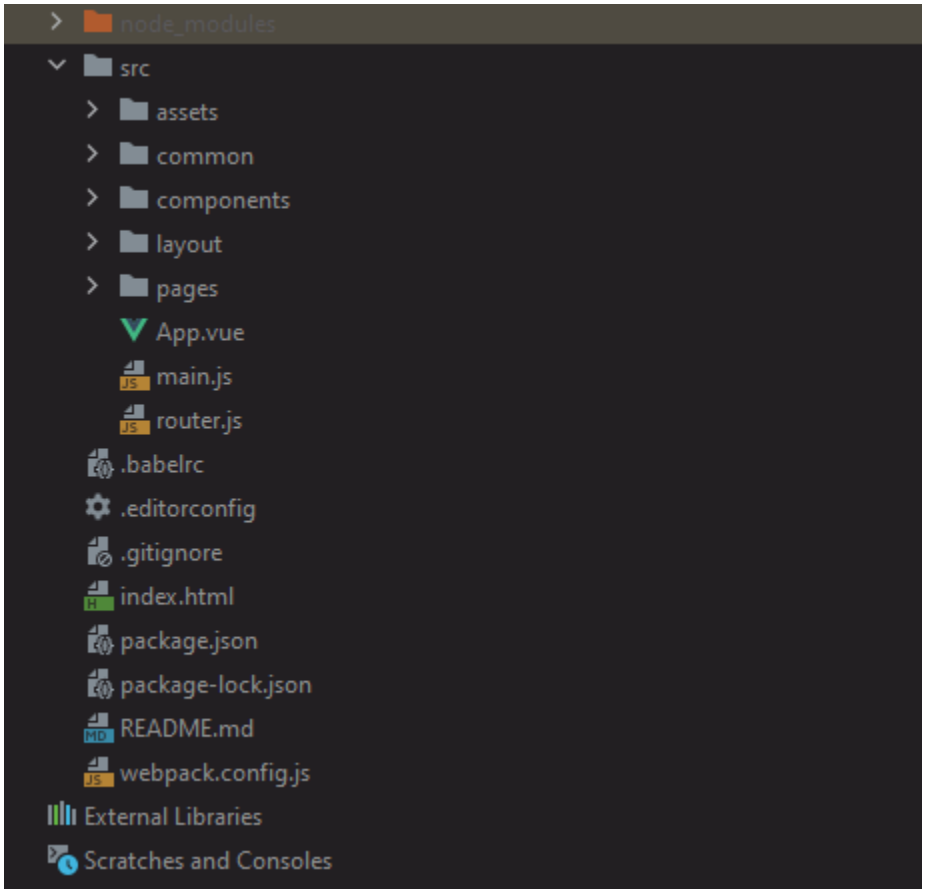


Ilustración 24 Estructura de archivos VUE.js

Dentro de la carpeta assets es donde se guarda el css, librerías de js como puede ser chartis,js para generar gráficas o librerías de terceros para alguna funcionalidad en concreto que modifique la vista de forma estética.

En comon tenemos un script encargado de enviar peticiones y tratar las respuestas. Se podría decir que es la lógica de las peticiones.

En components es donde se almacenan los componentes de la interfaz web. Es muy útil dividir la interfaz en componentes ya que, si se hace de forma eficaz, estos pueden ser reutilizado en diferentes sitios, además estos permiten parametrizaciones, lo que hace que la información que hay en ellos sea diferente dependiendo de los parámetros de entrada.

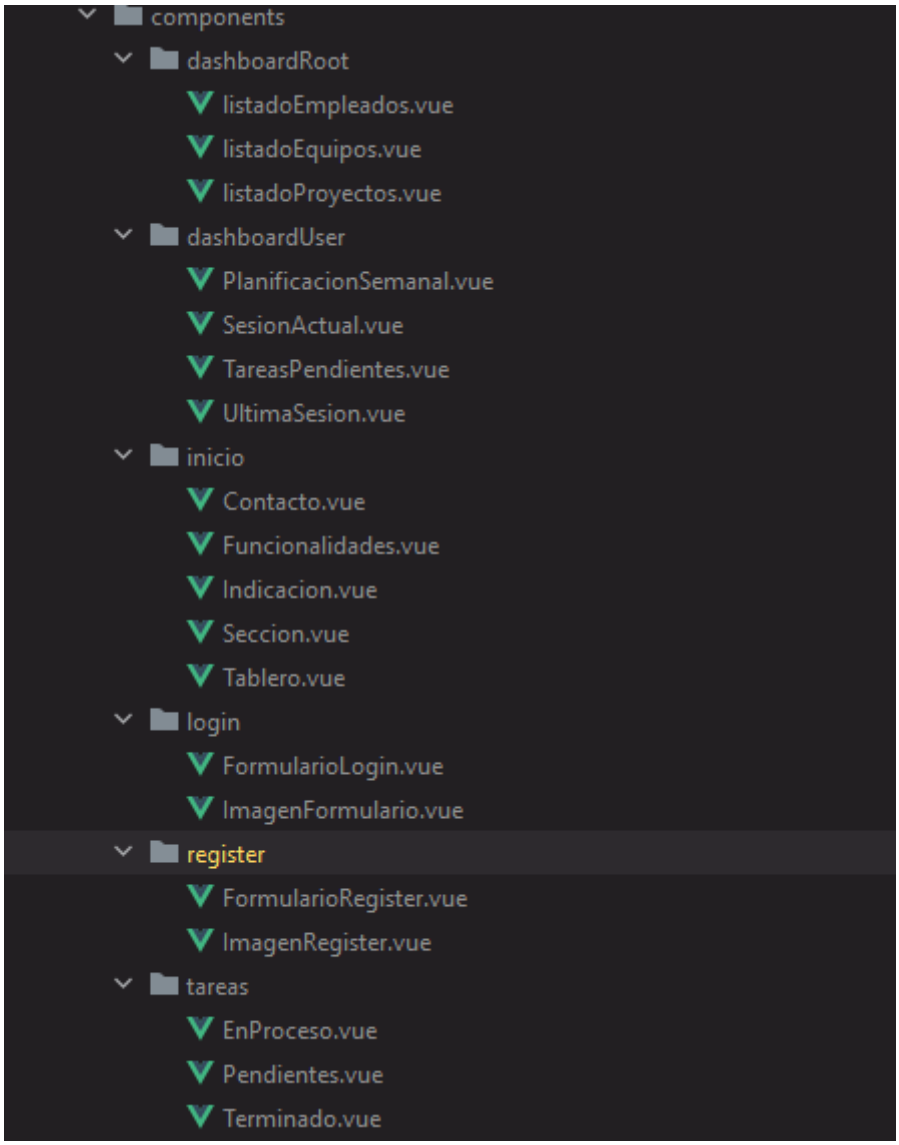


Ilustración 25 Directorio Components de VUE.js

Hay que destacar que estos componentes son elementos como formularios, menús de navegación, tablas de listados, etc.

Luego tenemos la carpeta Layouts, aquí se tiene guardan los componentes que le dan estructura a la página web, como contenedores div.

Y finalmente tenemos el directorio pages, aquí es donde se guardan todas las páginas que dispondrá el sitio web, desde aquí es donde se van llamando al resto de componentes Vue para ir montando una página con sus elementos.

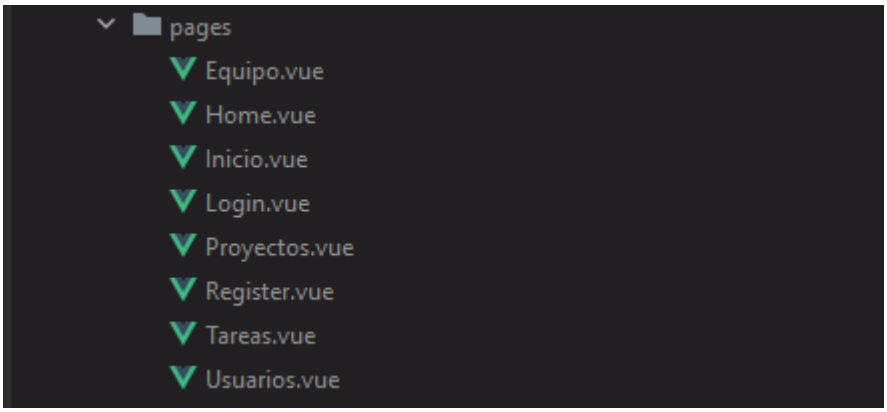


Ilustración 26 Directorio pages de VUE.js

Fuera de los directorios listados tenemos dos archivos muy importantes, main.js que es donde se ejecuta la lógica más importante al llamar a las rutas del proyecto y routes.js que al igual que el api.php es el encargado de definir que peticiones es capaz de atender la aplicación web.

5.3 Problemas de implementación resueltos

Durante el desarrollo de esta aplicación surgieron muchos problemas algunos se llegaron a solucionar, otros tuvieron que ser aislados o cambiado la lógica a otra perspectiva.

A continuación, mostraré una lista de los problemas que surgieron y como los solucione:

- La instalación de apache en mi servidor fue un desastre, fácilmente me tire un mes intentando instalarlo y hacerlo funcionar y no lo conseguía. Mi solución fue instalar Nginx
- Como yo quería un servidor propio ya sea para mi portfolio como profesional como para poder trabajar de forma remota desde diferentes equipos, necesitaba un nombre de dominio y un servidor en la nube. Por ello obtén en alquilar el servidor en Contabo.com, ya que ellos te proporcionan un servidor con un nombre de dominio.
- A la hora de instalar Laravel en el servidor la versión de php que se instalo inicialmente no era compatible con Laravel, por lo que tuve que actualizar la versión de php del servidor. Esto me retraso 1 semana.
- La creación de datos ficticios para rellenar la base de datos son funciones que se han quedado obsoletas en las nuevas versiones de Laravel, por lo que tuve que buscar una nueva librería para poder generar estos datos de prueba.

- Al haber ocupado ya el dominio que ofrecía el servidor no se puede alojar más de dos sitios web en un mismo dominio, o al menos a mí no me dejo. Por ello me decante por subir el frontend a Netlify, un portal web que te permite subir proyectos web como si fuese un servidor de forma gratuita.
- El calculo para obtener los rendimientos de los usuarios fallaba constantemente, las funciones devolvían valores muy anormales. Para solucionar esto se tuvo que restructurar algunas entidades y funciones.
- La versión que se ha usado de VUE.js ha cambiado la forma en la que se comunican los componentes, por lo que tocó revisar la documentación y emplear la nueva forma.
- Envío de correos de confirmación y verificación, por desgracia este problema no se pudo solucionar a pesar de haber visitado muchos sitios en internet ninguno parecía tener una solución. Se ha pensado que una de la causa que cause este error es que el proxy de contaba no este dejando salir los correos que no están emitidos con certificado SSL. De esta forma evitan que se creen bots de correos de Spam.
- Desarrollo asíncrono, dada la envergadura de este proyecto y los posibles errores que pueden surgir en cada parte de este, era muy difícil llevar todo el desarrollo al mismo tiempo, por ello la parte más avanzada y preparada para salir a producción es la API Res y el frontend se ha quedado un poco atrás.
- Durante los últimos días ha habido un problema con el repositorio GIT, no me deja subir los últimos cambios a él.

6. MANUALES

En este apartado se va a explicar como instalar los proyectos caso de haber una migración de servidor (Nivel técnico) y una pequeña guía de cómo usar la aplicación.

6.1 Manual de instalación

6.1.1 Instalación proyecto Laravel Para hacer una instalación del proyecto Laravel dese cero.

- Primero habría que mover todos los directorios y archivos al servidor que se quiere migrar.
- Introducir las credenciales y tokens de acceso necesarios en el archivo. env
- Montar la base de datos
    - php artisan migrate:refresh –seed

Lo que hace este comando es eliminar todas las tablas que tengan una migración en el directorio databases/migrations y volver a crearlas, el parámetro –seed es para que mediante las factories y los seeders se generen los datos automáticamente despues de la creación de las tablas.

6.1.2 Instalación proyecto VUE.js Para hacer la instalación del proyecto de VUE, al igual que con Laravel, lo primero es mover los archivos de sitio, y una vez se encuentren en el directorio hay que iniciar el servidor node.js y luego ejecutar el comando “npm run dev” una vez realizado esto nuestro proyecto de vue ya debería funcionar correctamente, siempre y cuando el servidor este accesible a través de internet.

6.2 Guía de uso

A continuación, se hará una pequeña guía de uso sencilla para realizar acciones básicas en Sudle:

- **Registro:** Para registrarse y tener acceso a una prueba gratuita, basta con entrar en el enlace de la parte superior derecha “Registrarse” rellenas los campos y automáticamente será redirigido al formulario de inicio de sesión.
- **Inicio de sesión:** Para iniciar sesión en la aplicación hay que ir al botón de la parte superior derecha y hacer click en “Iniciar sesión”
- **Como administrador crear entidades:** Una vez se haya iniciado sesión el administrador tendrá en la parte izquierda un menú lateral donde cada opción que hay allí te lleva directamente a un formulario de creación.
    - Lo que puede crear es equipos, proyectos, jornadas y tareas.

    - Podrá asignar proyectos a equipos
    - Jornadas a proyectos
    - Tareas a usuarios
- **Como usuario:** Una vez se haya iniciado sesión el usuario tendrá un menú lateral diferente al del administrado. Este solo podrá crear tareas y como mucho editarlas para ponerlas en estado pendiente o finalizada
- **Visualizar las gráficas de rendimiento:** basta con entrar en el panel de control, esa es la vista que se le muestra al administrador al iniciar sesión.

7. EVALUACIÓN

Este proyecto se ha presentado a varios conocidos, se les ha hablado sobre el y de como funciona, incluso algunos han llegado a probar tanto la API como la aplicación web.

En general la evaluación que me han dado ha sido bastante positiva, de hecho, alguno de ellos (Un profesor de la EPSG) me sugirió que este proyecto podría complementarse con una red neuronal para crear modelos de trabajadores, de está forma poder crear como estándares y con ello ajustar mostrar al administrador de los equipos cual debería ser el rendimiento de un tipo de trabajador.

Otro caso fue que se presento este proyecto a una Psicóloga (fue de casualidad) y nos comento la idea de que no solo serviría para evaluar si un empleado esta haciendo su trabajo o no, si no también para ver cuando un empleado se siente exhausto o padece estrés por demasiada responsabilidad y de esta forma los recursos humanos pueden intervenir antes de que el empleado decida marcharse de la empresa. Aunque para poder llegar a este punto habría que hacer bastantes cambios en la base de datos y diseñar lógicas nuevas para la captación de nuevas actividades que ayuden a saber el estado del trabajador.

8. MODELO COMERCIAL

Volviendo a hablar de el modelo de negocio, este es un proyecto pensado para ser B2B con una subscripción mensual.

Los planes que se han pensado son:

- **Plan gratuito:**
    - Límite de usuarios que se pueden crear: 3 (sin contar el usuario que se crea al crear la cuenta en Sudle)
    - Límite de equipos que se pueden crear: 1
    - Coste mensual: 0€
- **Plan básico:**
    - Limite de usuarios que se pueden crear: 15
    - Limite de equipos que se pueden crear: 5
    - Coste mensual: 7.99€
- **Plan premium:**
    - Limite de usuarios que se pueden crear:45
    - Limite de equipos que se pueden crear: 10
    - Coste mensual: 19,99€
- **Plan Bussines:**
    - Limite de usuarios que se pueden crear: Ilimitados
    - Limite de equipos que se puede crear: ilimitados
    - Coste mensual: 1€/usuario creado y activo

9. CONCLUSIONES

El proyecto ha sufrido un poco en la parte del frontend ya que mi experiencia y conocimientos en ese campo son algo escaso, sin embargo, ello me ha motivado a querer aprender más sobre el frontend y al UX/UI (Experiencia de usuario / Interfaz de Usuario).

Mi conclusión sobre este proyecto es que he escogido un proyecto demasiado grande para intentar desarrollarlo yo solo, y más aún trabajando y estudiando a la vez. Sin embargo, a pesar de ello me ha enriquecido profesionalmente bastante, ya que me ha aportado una visión diferente de como se despliega una aplicación web real. He aprendido de primera mano las tecnologías que más me gustan en mi campo que es el desarrollo web. He configurado mi primer servidor para alojar sitios web.

En las evaluaciones de mis conocidos me quedó claro que como posible mejor se le añadirá una red neuronal al proyecto, ya sea con AWS o alguna herramienta similar para crear los modelos de trabajadores, de esta forma se podrá estimar de mejor forma cual debería ser el rendimiento de este. Quien sabe incluso se podrían hacer como fichas de trabajadores con estadísticas como si de un videojuego se tratase.

10. BIBLIOGRAFÍA

- Kathleen Juell and Justin Ellingwood. *Cómo instalar Nginx en Ubuntu 18.04.* 26/04/2019 URL: https://www.digitalocean.com/community/tutorials/como-instalar-nginx-en-ubuntu-18-04-es
- Mitchell Anicas and Justin Ellingwood. *Cómo configurar BIND como servidor DNS de red privada en Ubuntu 18.04.* 22/01/2020 URL: https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-a-private-network-dns-server-on-ubuntu-18-04-es
- Marcos Rodríguez*. INSTALAR SERVIDOR FTP EN UBUNTU 18.04 PASO A PASO.* 11/04/2020 URL: https://vivaubuntu.com/instalar-servidor-ftp-en-ubuntu-paso-a-paso/
- Savic and finid. *Cómo instalar y configurar Postfix como servidor SMTP de solo envío en Ubuntu 18.04.* 19/052020 URL: https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-postfix-as-a-send-only-smtp-server-on-ubuntu-18-04-es
- Projektfarm GmbH. *Cómo instalar VestaCP en Ubuntu 18.04 LTS. 06/12/2021.* URL: https://howtoforge.es/como-instalar-vestacp-en-ubuntu-18-04-lts/
- Piero Noviello. *Cómo instalar Laravel en Ubuntu 18.04 LTS.* 01/03/2021 URL: https://noviello.it/es/como-instalar-laravel-en-ubuntu-18-04-lts/?expand_article=1
- Evan You. *Instalación.* URL: https://es.vuejs.org/v2/guide/installation
- Brian Boucheron *Cómo instalar Node.js en Ubuntu 20.04.* 11/06/2020 URL: https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-ubuntu-20-04-es
- Silvia Mazzetta. *Cómo crear un aplicación Vue.js en 5 minutos.* 23/05/2020 URL: https://www.ma-no.org/es/programacion/javascript/como-crear-un-aplicacion-vue-js-en-5-minutos
- Taylor Otwell. *Eloquent: Getting Started.* URL: https://laravel.com/docs/10.x/eloquent