Memoria del proyecto

Nuestro equipo se ha orientado a la realización del módulo de gestión de los recursos humanos (RRHH) del sistema ERP. A continuación se presenta la documentación técnica de todo el proceso de desarrollo, abarcando desde la planificación de este hasta la descripción técnica de todas las tecnologías usadas y una explicación con detalles del código añadido.

Equipo

A continuación, los miembros del equipo que se ha encargado del desarrollo de este módulo del ERP.

- Rosa María Rodríguez Lledó
- Ilya Slyusarchuk
- Adrián Berenguer Agulló
- Javier Rodríguez Juan

Introducción

En este apartado haremos una descripción general del sistema describiendo apartado por apartado toda la cara técnica del módulo desarrollado.

Tecnologías

Las tecnologías escogidas para el desarrollo del módulo fueron elegidas basándonos en la experiencia previa que el grupo al conjunto tenía con tecnologías de desarrollo de aplicaciones web.

En cuánto a la parte del servicio (o back-end) se escogió el uso de **PHP/Laravel v8.7**, ya que además de ser un framework de desarrollo ampliamente conocido por todos lo miembros del equipo tiene una gran comunidad y soporte para la resolución de posibles problemas que pudieran surgir durante el desarrollo. Este servicio funcionará contra una base de datos que usará el motor de gestión de bases de datos relacionales **MySql**. Por otro lado, en la parte de la aplicación (o front-end) se usó el framework de desarrollo JavaScript **Vue 2**, acompañado del framework de componentes visuales **Vuetify v2.6**. Con el objetivo de mantener los componentes de Vue lo más *stateless* posible, también usamos **Vuex 3** cómo almacén centralizado del estado de la aplicación.

Cabe destacar que para realizar las compilaciones del código JavaScript y CSS del proyecto durante el desarrollo se usó el paquete por defecto que aporta Laravel para realizar esta tarea, **Laravel Mix**.

Arquitectura

Explicaremos la arquitectura desde 2 perspectivas distintas, primero explicaremos la arquitectura general y después entraremos más internamente a explicar detalles de esta.

Arquitectura general

La arquitectura de la aplicación viene fuertemente ligada por los frameworks de desarrollo escogido. Como en la anterior subsección hemos comentado, este módulo ha sido desarrollado bajo la combinación de las tecnologías Vue + Laravel, por lo que la arquitectura de este y los patrones de diseño que definen nuestra arquitectura vienen definidos por los estándares de desarrollo en estos frameworks.

Al usar PHP/Laravel como capa de servicios adoptamos como base fundamental el patrón de diseño **MVC (Modelo-vista-controlador)** propio de este, mientras que por otro lado, la utilización de Vue junto a su *router* nos permite también adoptar una arquitectura **SPA (Single page application)** para la capa de aplicación. Como protocolo de comunicación entre la capa de aplicación y la de servicios, así como para la definición de rutas haremos uso del estándar **REST**.

El poder combinar una arquitectura MVC con una SPA es sumamente potente debido a que por un lado, preservamos la facilidad que aporta MVC al desarrollo el albergar todo el contenido del proyecto en un mismo directorio, mientras que también generamos una gran experiencia de navegación para el usuario gracias a la arquitectura SPA, que evita las recargas de página durante la navegación en el sistema.

Arquitectura interna

Cada proyecto tiene su estructura propia definida por su implementación, además de las estructuras generales que ya hemos comentado en el apartado anterior. En este apartado profundizaremos un poco más en nuestra estructura interna.

Cada proyecto basa su implementación en sus necesidades. En este caso, la estructura de servicios a desarrollar era una estructura sencilla que no necesitaba ni accesos a base de datos complejos ni procesamiento de datos posteriores. Debido a esto se ha realizado una implementación en la cuál si vemos la aplicación al completo (interfaz y servicios) como un todo, está compuesta por 3 capas: La capa de *aplicación*, la capa de los *controladores* y la capa de los *modelos*. Separaremos las capas por cliente y servicio:

- **Cliente:** Está formado por la capa de aplicación. En esta capa tenemos todas las vistas que conforman la parte gráfica de la aplicación, cabe destacar que delegamos la responsabilidad del filtrado de datos también en esta capa.

- **Servicio:** Ahora pasamos a la parte de los servicios. Bajo la capa de aplicación tenemos la capa de los controladores, donde se realiza todo lo necesario para recibir los datos de la base de datos y proveerlos a la capa de aplicación. Por tanto aquí realizamos desde las validaciones propias de esta capa (validaciones de datos de

entrada) hasta el acceso a base de datos. En esta capa no procesamos datos en ningún caso ya que no hay situaciones que lo requieran ni tampoco realizamos filtrados (esta responsabilidad la delegamos en la capa de aplicación). Por último tenemos la capa de modelos, que albergan la definición de las entidades usadas en el sistema, desde sus propiedades hasta sus relaciones con otras entidades de la aplicación.

Servicios

Todos los servicios usados en este sistema son de implementación propia, no se han usado ni APIs públicas, ni servicios desarrollados por terceras personas ajenas a nuestro equipo de desarrollo.

Servidores

Nuestra aplicación ha sido desplegada en un servidor del proveedor de servidores en la nube DigitalOcean. Este servidor cuenta con 1 núcleo virtual Intel, 1Gb de memoria RAM y 25Gb de memoria en disco.

Como podemos ver este servidor cuenta con unas especificaciones básicas pero que son suficientes para la carga computacional que vamos a recibir. Adicionalmente, cabe destacar que la tecnología usada como servidor web para el despliegue de la aplicación ha sido **Apache 2**.

Se puede acceder a la aplicación desde la url http://165.232.80.121.

Análisis inicial

En este apartado explicaremos el análisis funcional realizado previamente a la implementación.

Requisitos

Dentro del apartado de requisitos podemos diferenciar 2 tipos:

Requisitos tecnológicos

En este subapartado vemos los requisitos necesarios que debe tener un usuario para poder acceder y navegar por la aplicación. Sólo mencionaremos aquí 2 puntos:

- Será una aplicación implementada como página web, por lo que se necesitará conexión a Internet.
- Se recomienda navegar por la página desde un ordenador, ya sea portátil o de sobremesa
- No hay restricciones de sistemas operativos ni de navegadores. Aún así, recomendamos utilizar Windows 7 en adelante y una versión de navegador reciente

Requisitos funcionales

Los requisitos del módulo a desarrollar abarcan la gestión de:

- Nóminas
- Usuarios
- Vacaciones
- Jornada laboral
- Control de entradas y salidas
- Bajas laborales y justificantes asociados

Como requisitos opcionales a implementar se incluye lo siguiente:

- Gestión de horas extra
- Exportación de los datos de este módulo para su uso en otros módulos del ERP

Funcionalidades

A continuación se listan las funcionalidades que deben ser implementadas de acuerdo a las especificaciones del sistema y a la información provista por el *Product Owner* del proyecto.

Teniendo en cuenta los perfiles de usuarios que habrá en el módulo y los requisitos previamente descritos obtenemos la siguiente lista de funcionalidades divididas por perfiles:

- Empleado base:
    - Fichar al principio y al final de su jornada
    - Pedir vacaciones y consultar estado de peticiones
    - Consultar días de vacaciones consumidos/disponibles
    - Pedir días de baja y subir justificantes
    - Consultar jornadas realizadas
    - Consultar sus datos básicos de usuario
    - Consultar estadísticas (número de horas trabajadas, solicitudes enviadas, nóminas recibidas y de compañeros de trabajo)
- Empleado supervisor:
    - Mismas funcionalidades que el empleado base (menos la consulta de sus estadísticas)
    - Autorizar/Rechazar peticiones de vacaciones de miembros de su equipo
    - Buscar y filtrar miembros de la empresa y de su equipo
    - Consultar datos personales e historiales de los empleados
    - Consultar presencialidad de cada miembro de la empresa
    - Consulta de vacaciones autorizadas con un calendario
- Gestor de RRHH:

- Gestionar miembros de la empresa (Consultar, añadir, eliminar y editar miembros)
- Modificar convenios de empleados
- Generar nóminas
- Consultar historial de nóminas emitidas
- Autorizar bajas laborales

Opcionalmente se implementarán también las siguientes funcionalidades:

- Empleado supervisor
    - Gestión de horas extra de los miembros de su equipo
- Usuarios de otros módulos del ERP
    - Consultar datos del módulo de RRHH actualizados

Casos de uso

A continuación y a partir del listado de funcionalidades anteriormente descritas estableceremos los siguientes casos de uso. Asumiremos que en todos los casos de uso el usuario debe loguearse para comenzar con el proceso

- Fichar al comenzar la jornada: Un empleado normal entra en el apartado “Jornada laboral” y presiona sobre el botón “Comenzar jornada”.
- Fichar al acabar la jornada: Un empleado normal entra en el apartado “Jornada laboral” y presiona sobre el botón “Acabar jornada”.
- Pedir vacaciones: A partir de 1 mes antes del comienzo de un período vacacional marcado por la empresa, un empleado normal entra en el apartado “Solicitudes”, abre la sección “Selección de vacaciones”, elige la fecha de inicio y fecha de finalización que desea en sus vacaciones respetando el máximo de días permitidos por la empresa para ese período y presiona sobre “Confirmar” para enviar su solicitud. (sin periodos, solo mes antes)
- Consultar estado de vacaciones: Un empleado normal entra en el apartado “Vacaciones” y ve los campos “Días disponibles” y “Días consumidos”.
- Consultar un registro de vacaciones: Un empleado normal que haya estado alguna vez de vacaciones entra en el apartado “Vacaciones” y en la sección “Registros vacacionales” elige “Ver más” sobre uno de los registros para ver la información completa acerca de ese registro.
- Pedir días de baja: Un empleado normal entra en el apartado “Solicitudes”, abre la sección “Pedir baja laboral”, elige la fecha de inicio y fecha de finalización y presiona sobre “Confirmar” para enviar su solicitud
- Subir justificante de baja: Un empleado normal que tenga al menos un registro de baja laboral entra en el apartado de “Solicitudes”, va a la sección “Crear justificante”, elige la opción “Baja laboral” y elige el registro de baja sobre el cual quiere aportar el justificante.
- Aceptar/Rechazar solicitud de vacaciones: Un empleado supervisor con miembros en su equipo que hayan hecho alguna solicitud entra al apartado “Solicitudes pendientes”, presiona sobre “Ver más” en la solicitud que quiera valorar y tras leer la información de la solicitud presiona sobre “Aceptar” o

"Rechazar" para enviar una notificación con su decisión al empleado. Si el supervisor acepta la solicitud, se le creará un nuevo registro de vacaciones al empleado.

- Consultar miembros: Un empleado supervisor entra en la sección "Empleados" y ve todos los miembros que trabajan en la empresa
- Consultar presencialidad de un empleado: Un empleado supervisor entra en la sección "Empleados", presiona sobre el botón "Ver más" de uno de ellos y ve en el campo "Jornada laboral" las horas diarias que realiza ese empleado.
- Añadir horas extra: Un empleado supervisor entra en la sección "Empleados", presiona sobre el botón "Ver más" de uno de ellos y si este es de su equipo, podrá añadir las horas extra que haya realizado.
- Consultar información de un empleado: Un gestor del sistema entra en la sección "Empleados" y tras presionar sobre el botón "Ver más" de uno puede ver toda la información almacenada sobre ese empleado
- Editar empleado: Un gestor del sistema entra en la sección "Empleados" y tras presionar sobre el botón "Editar" de uno de los empleados puede modificar cualquiera de los campos de información de ese empleado
- Añadir empleado: Un gestor del sistema entra en la sección "Empleados" y tras presionar sobre el botón "Añadir" rellena un formulario el cual al completar crea un nuevo empleado.
- Eliminar empleado: Un gestor del sistema entra en la sección "Empleados" y elimina a un empleado pulsando sobre el botón "Eliminar" de uno de los empleados
- Generar nómina: Un gestor del sistema entra en la sección "Nóminas", y elige el mes y el empleado sobre los cuáles quiere generar la nómina. Tras esto el empleado recibe una notificación con la emisión de su nómina.
- Aceptar/Rechazar bajas: Si algún empleado ha hecho una solicitud de baja, un gestor del sistema entra al apartado "Solicitudes pendientes", presiona sobre "Ver más" en la solicitud que quiera valorar y tras leer la información de la solicitud presiona sobre "Aceptar" o "Rechazar".

Historias de usuario

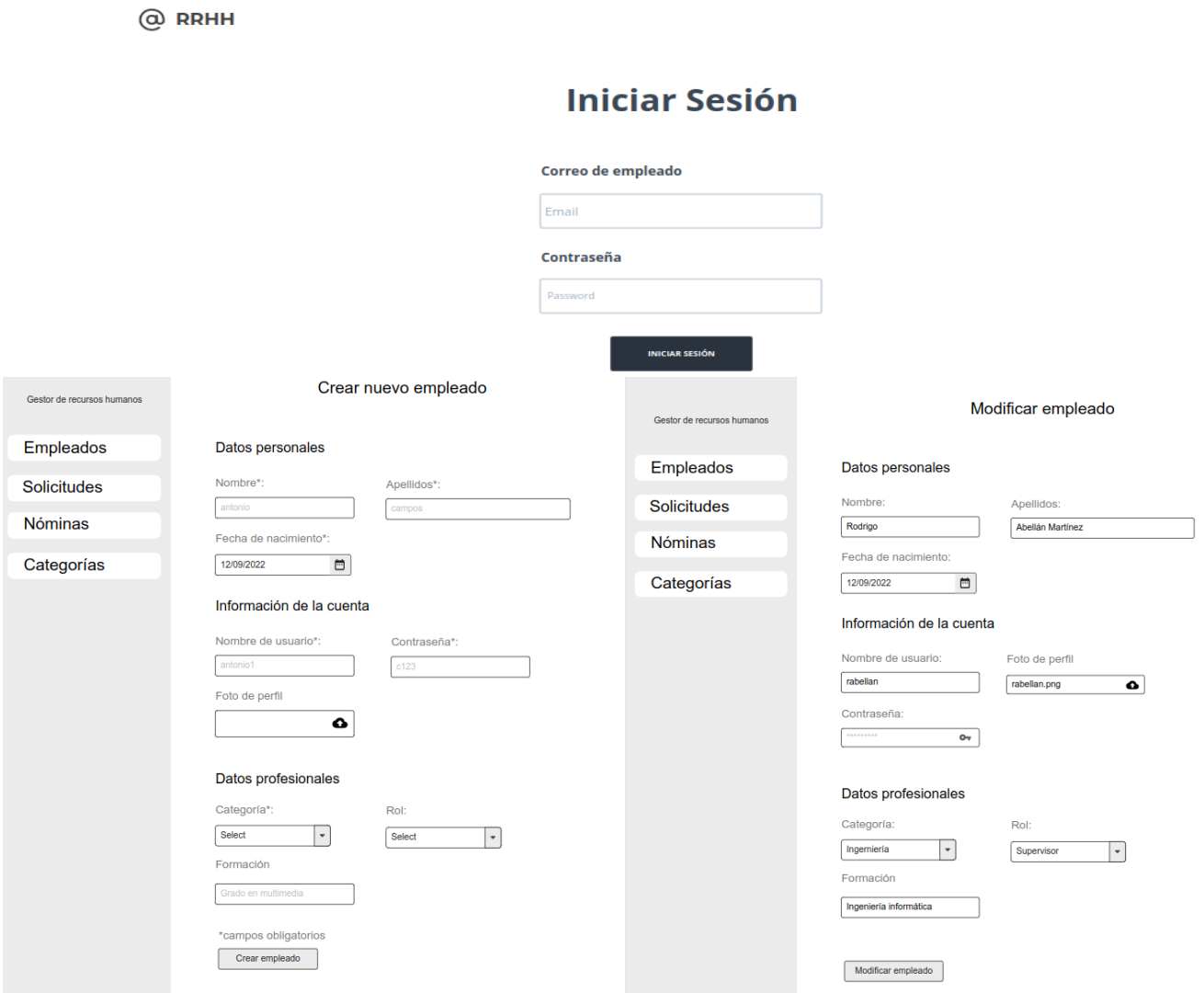
Para finalizar con el análisis previo, se presentarán las historias de usuario sobre las que el equipo de desarrollo trabajó durante la implementación de la aplicación.

- Como trabajador, quiero poder fichar para demostrar las horas trabajadas en una jornada.
- Como trabajador, quiero saber cuántos días de vacaciones tengo disponibles y poder solicitarlos.
- Como trabajador, quiero poder pedir una baja automáticamente entregando solo un justificante.
- Como supervisor, quiero saber quién está en la empresa en cada momento para poder organizar el trabajo.
- Como supervisor, quiero saber cuáles son los empleados que superviso.

- Como supervisor, quiero controlar las vacaciones de los empleados para organizar el equipo.
- Como supervisor, quiero saber cuántas horas y días trabaja cada empleado para seguir su progreso.
- Como gestor de RRHH, quiero dar de alta o de baja al resto de trabajadores para mantener actualizados los datos.
- Como gestor de RRHH, quiero guardar los contratos, convenios y demás información actualizada de los trabajadores.
- Como gestor de RRHH, quiero automatizar la generación de las nóminas y las horas extra si las hubiera.
- Como gestor de RRHH, quiero controlar las bajas automáticamente de manera sencilla.
- Como gestor de RRHH, quiero exportar todos los documentos generados fácilmente (nóminas, bajas, altas, despidos) para poder enviarlos a los trabajadores afectados.
- Como gestor de RRHH, quiero poder exportar la información de uno o muchos empleados de manera sencilla.

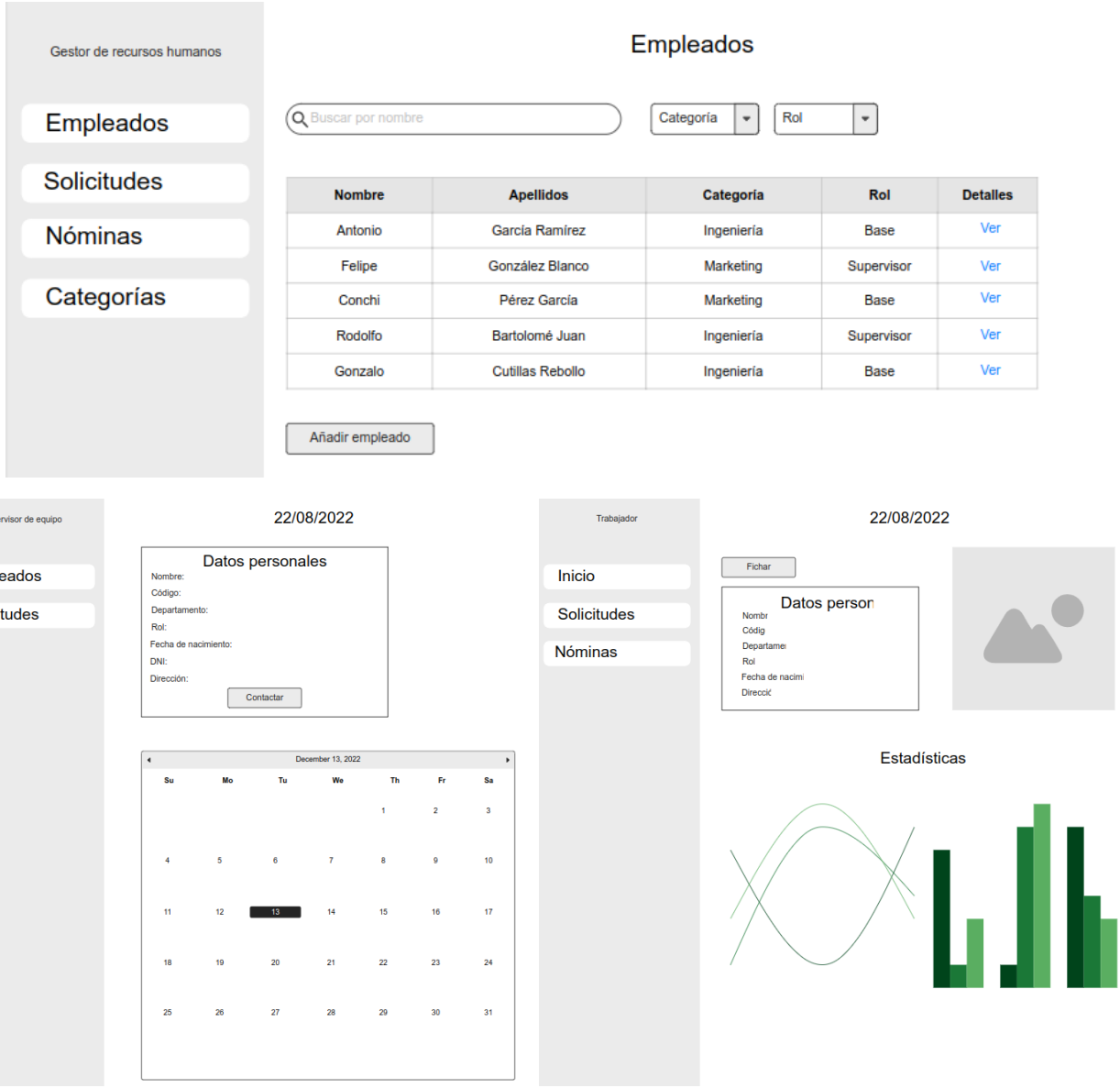
Mockups

En el apartado actual, se destacan los Mockups realizados junto a la justificación de sus patrones de diseño.



En estas tres primeras páginas, podemos observar el patrón de diseño de identificación de usuarios. Éstas son las páginas de iniciar sesión por parte de cualquier usuario y, crear y modificar los datos del usuario por parte del administrador. De esta manera se mantienen los datos del usuario y es capaz de recuperar sus datos mediante la ayuda del administrador.

Además de esto, podemos ver patrones de formulario en los formularios de creación y edición de empleados. Los campos han sido agrupados por temática de campos. Todos los campos están alineados en sus apartados. En estos formularios, se adjuntan valores por defecto inteligentes y, se especifican los campos que son obligatorios.



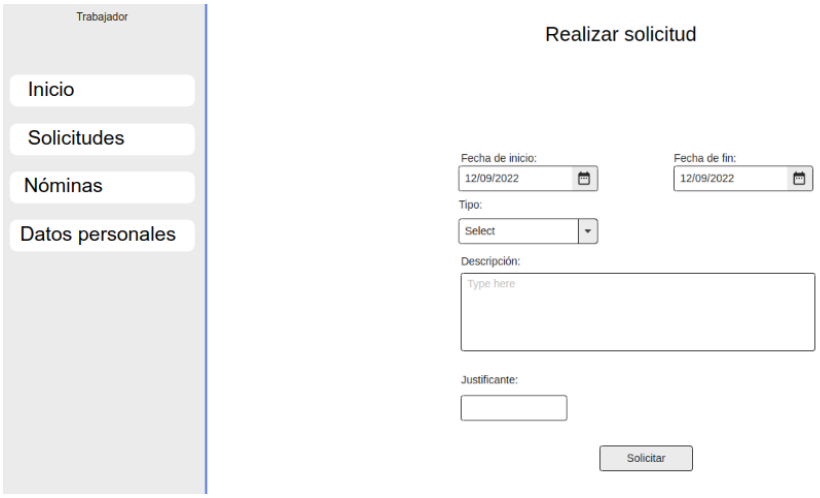
El segundo patrón que podemos observar es el de página principal. En las diferentes páginas de inicio de cada usuario, observamos el menú desde el que pueden acceder a

cada uno de los apartados de la web y, además, mostramos datos interesantes para cada uno de ellos dependiendo de su rol.

Si centramos nuestra vista en el menú horizontal, podemos destacar el patrón de navegación, mediante el cuál se puede acceder a cada una de las páginas de la web y mostramos en cada momento en qué página se encuentra el usuario.

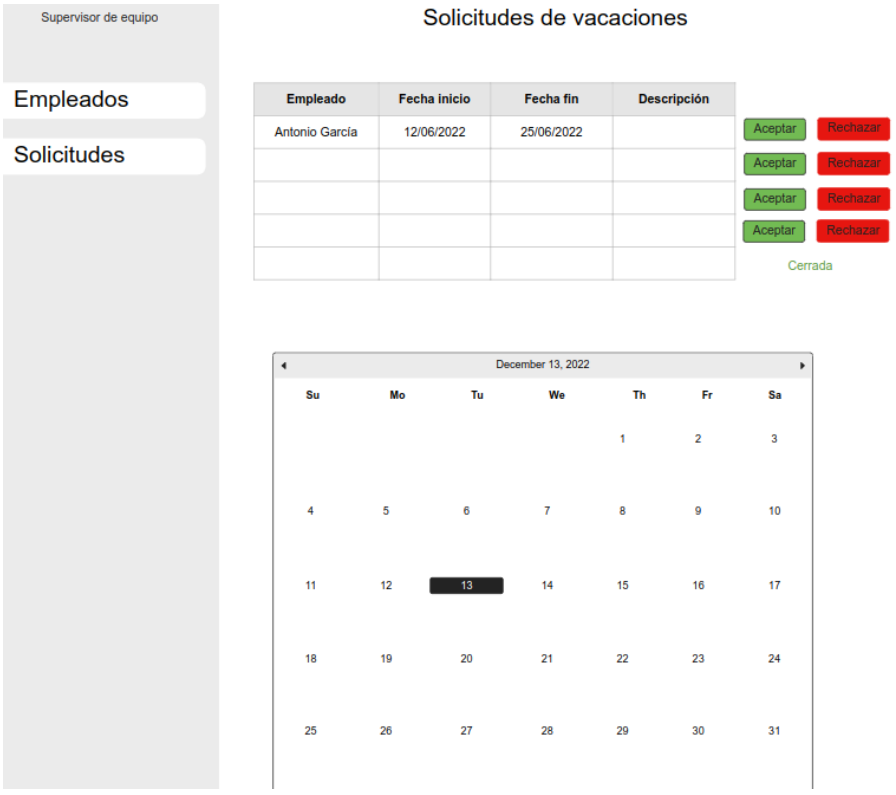
En las imágenes anteriores mostramos cómo hemos incluido el patrón de listado. En el que podemos realizar acciones y tenemos varias columnas para cada fila mostrando información. En las cuáles existen buscadores y filtros.

A continuación, tenemos otras de las vistas donde realizamos patrones ya mencionados o bien, son mockups diseñados para la aplicación sin patrones definidos.



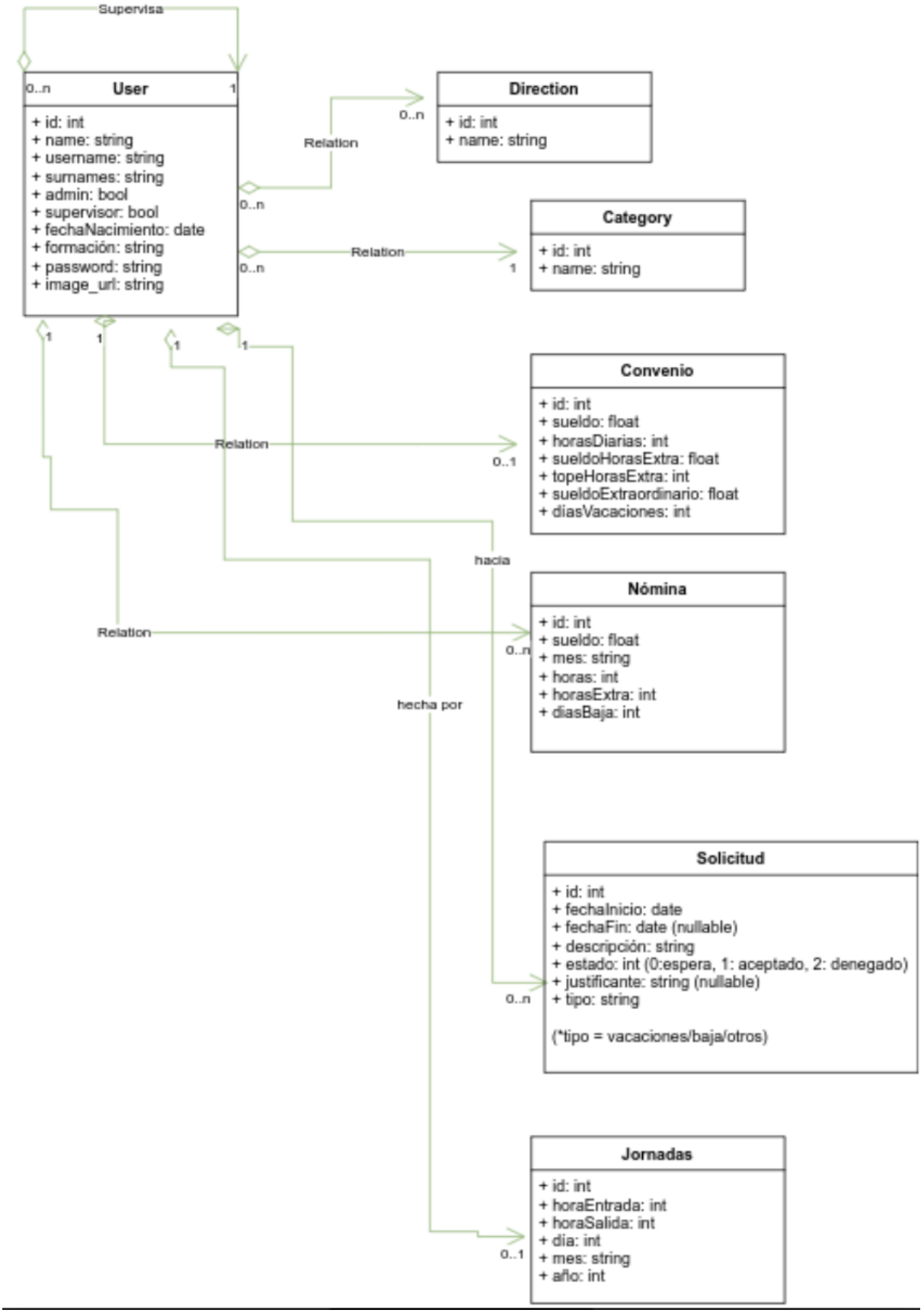
Página para realizar solicitud

Página de detalles de un empleado para el supervisor.



Página de solicitudes de vacaciones que el supervisor verá sobre sus trabajadores.

Diagramas de datos



Como se puede observar el sistema cuenta con 7 tabla y múltiples relaciones. Se podría decir que la tabla más importante es la de usuario, puesto que está relacionada con el resto de tablas.

Metodología y planificación

Seguiremos la metodología **Scrum**. Inicialmente los sprints serán semanales. Los martes en clase son los días dedicados a planificar nuevas tareas y hacer un control del trabajo de la semana anterior. Usaremos **Trello** para definir las historias de usuario y el tablero de **Github** para cuestiones técnicas.
Además, definimos una rama de desarrollo de git para cada integrante del equipo, cuyos cambios se van integrando periódicamente en la rama main.

Descripción de la implementación

A continuación se entrará en detalle en cada una de las funcionalidades añadidas al sistema. En los casos donde la explicación sea más compleja también se aportarán capturas de partes del código para mejorar el entendimiento de estas.

Autenticación y protección de rutas

Para comenzar, explicaremos los mecanismos de autenticación y de protección de rutas implementados. Primero comentaremos el sistema de autenticación y seguridad de acceso según rol implementado en la parte de servicios y luego veremos el mecanismo de protección de rutas implementado en la parte cliente.

Autenticación

Para proteger la API de usuarios que puedan estar mandando peticiones a esta de forma externa sin estar registrados se ha implementado un sistema de autenticación por token JWT. Para ello, hemos usado el paquete de PHP "*php-open-source-saver/jwt-auth*".

Para hacer que este sea usado por defecto en el sistema tendremos que hacer uso de los "*guards"* de Laravel. Añadiremos un nuevo guard (llamado "*api"* en nuestro caso) que usará el driver "jwt" y lo pondremos como "*guard*" por defecto (toda esta configuración está disponible en el archivo "*auth.php*")

```
'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],
        'api' => [
            'driver' => 'jwt',
```

```
            'provider' => 'users',
        ],
    ],
```

```
'defaults' => [
        'guard' => 'api',
        'passwords' => 'users',
 ],
```

Todas las llamadas que tengan que ver con autenticación las almacenaremos en "*AuthController.php*". En este archivo podemos encontrar los métodos que usamos para iniciar sesión y para cerrar sesión. De aquí podemos destacamos la siguiente línea:

```
 $token = Auth::attempt($credentials);
```

En esta es donde realizamos la generación del token si las credenciales son válidas durante el proceso de inicio de sesión. Después, devolvemos este token en la respuesta de la llamada y almacenaremos este junto a la información del usuario devuelta por la API en el *store* de Vuex para poder acceder a la información desde cualquier lugar de la aplicación. Cabe destacar que este token sólo será válido durante la sesión del usuario logueado, una vez este cierre sesión este token ya no será válido.

Una vez el sistema de tokens JWT está listo debemos hacer que nuestra API solo acepte peticiones si en la cabecera de estas existe un token JWT válido. Para ello haremos uso de un *middleware* que nos proporciona el paquete que hemos instalado. Aplicaremos este *middleware* a todas las rutas de la API menos al login. A continuación cómo se aplica este *middleware* a las rutas:

```
Route::group(['prefix' => 'api', 'middleware' => 'auth:api'], function
() {

. . . .

}
```

Dentro de ese grupo de rutas tendremos toda nuestra API de forma que con sólo esa invocación aplicaremos la autenticación en todas las rutas.

Cabe destacar que almacenaremos tanto el token JWT como la información del usuario en el *localStorage* del navegador para evitar que se cierre la sesión del usuario al realizar una recarga de la página. Para esto inicializamos las variables de estado del *store* al contenido del *localStorage*.

```
var store = new Vuex.Store({
 state() {
   return {
```

```
    user: JSON.parse(localStorage.getItem('user')),
    _token: localStorage.getItem('_token'),
. . . . .
}
```

Protección de rutas públicas

En el apartado anterior hemos visto cómo aplicamos el mecanismo de autenticación en el lado del servicio, ahora nos centraremos en el sistema de protección de rutas públicas, a las cuáles el usuario podrá acceder o no según si ha iniciado sesión y según su rol de usuario.

Para esto haremos uso de un método especial que nos proporciona el *router* de Vue, este es el método *beforeEach()*. Este método nos permite realizar acciones antes de cada navegación realizada por el usuario. Además de esto, usaremos la propiedad *meta* que puede ser añadida a las rutas de Vue para dar información acerca de estas.

```
{
  path: '/superole/dashboard',
  name: 'superole-dashboard',
  component: () => import('@/views/dashboard/DashboardSuper.vue'),
  meta: {
    auth: true,
    super: true
  },
},
```

Dentro de la propiedad *meta* usaremos 4 distintivos, el atributo *auth*, *super*, *base* y *admin*. El distintivo *auth* será añadido cuándo se necesite autenticación para poder entrar a la ruta, el *base* cuándo la ruta sea accesible para un usuario base, el *super* cuándo se necesiten permisos de supervisor o administrador y el *admin* cuándo se necesiten permisos de administrador (*auth* siempre irá acompañado de *base, super* o *admin*).

Usando estos atributos *meta* y los booleanos “admin” y “supervisor” del modelo *User* realizaremos todas las comprobaciones que necesitamos para filtrar las navegaciones a rutas desde el método *beforeEach()* del *router*.

```
router.beforeEach((to, from, next) => {
 if (to.matched.some(record => record.meta.auth) && store.state.user
=== null && to.name !== 'login') {
  next({
    name: "login"
  });
 }
```

```
 if (to.matched.some(record => record.meta.base) && store.state.user
!== null && (store.state.user.admin || store.state.user.supervisor)) {
   next({
     name: "error-404"
   })
 }

. . . . .

 else {
   next();
 }
}
```

Cómo podemos observar en la anterior captura, accedemos a la propiedad *meta* de la ruta a dónde vamos a navegar y en función de sus distintivos permitimos la navegación (llamando a *next()*), la rechazamos (mostrando la página de error 404) o redirigimos al usuario a la pantalla de iniciar sesión.

Exportación de datos

Para que otros módulos de nuestro proyecto ERP puedan tener acceso a los datos de los empleados en cualquier momento, se han habilitado 2 *endpoints* protegidos por *API KEY* con la cuál podrán realizar peticiones a estos.

Los *endpoints* habilitados son los siguientes:

- http://165.232.80.121/api/users → Devuelve todos los empleados
- http://165.232.80.121/api/users/{id} → Devuelve los detalles del empleado con el identificador especificado

Para la protección de estas rutas hemos empleado un mecanismo similar al visto anteriormente para la autenticación de usuarios. Esta protección se basa en el uso de una *API KEY* única que almacenamos en nuestro proyecto como variable de entorno, y en un *middleware* que será el encargado de comprobar si la *API KEY* marcada por el usuario que realiza la petición es válida. En caso de que esta sea inválida le enviaremos al usuario un error *HTTP 401*.

Usuario administrador

Panel de inicio

El panel de inicio del administrador del sistema está compuesto de una tabla de búsqueda y filtrado de empleados. Este tratamiento de datos lo hemos llevado a cabo directamente en la capa de aplicación para realizar un procesamiento de los datos más rápido.

```
filterTeam() {
     if (this.supervisor) {
       this.employees = this.employees.filter(employee =>
employee.supervisado === this.$store.state.user.id)
     }
   }
```

Adicionalmente, se ha hecho uso del evento *keyUp*, para que los resultados de búsqueda por correo electrónico se actualicen cada vez que se escriba un nuevo carácter.

Los datos de estas tablas serán almacenados también en el *store* de Vuex para evitar realizar llamadas extra cuándo se tengan que recuperar esos datos en cualquier otra parte de la aplicación.

Formulario de creación de empleados

Para la confección de este formulario se han usado los componentes visuales *v-text-field* de Vuetify cómo *inputs* para el usuario. Mediante la propiedad *rules* de estos componentes se han añadido validaciones de los campos para enviar al servicio solo información correcta. Cada campo tiene una validación distinta según el *input* que se esté pidiendo en este.

Cabe destacar que para aceptar la inserción de archivos (la foto de perfil) en el formulario se ha usado en la capa de aplicación el elemento *v-file-input* también de Vuetify, y se guarda en nuestro servicio haciendo uso de la interfaz *Storage* de Laravel.

En este formulario también se añaden los datos relativos al convenio.

Página de detalles de empleado

Esta página es accesible desde la lista de empleados. De cada empleado se pueden ver su foto de perfil, sus datos personales, sus datos profesionales y los datos laborales. Asimismo también se puede contactar con el usuario gracias a la propiedad *mailto* de html, editar los datos accediendo al formulario de edición, o borrar al empleado. También se puede acceder al historial de solicitudes.

Cabe destacar que por una cuestión de seguridad, no se puede borrar al empleado administrador. Tampoco se puede acceder al historial de solicitudes porque entendemos que esta cuenta pertenece a un super usuario, no a un trabajador regular.

Antes de borrar, utilizamos el componente *v-dialog* para recordarle al usuario que los cambios son irreversibles y los datos se perderán si borra al usuario.

Formulario de edición de empleados

Este formulario mantiene la estructura del formulario de creación de empleado. La única diferencia es que los datos vienen preestablecidos y con el botón *restablecer* se puede volver a ellos. Hemos decidido separar el formulario en cuatro partes diferentes (datos personales, profesionales, convenio, datos de la cuenta) para facilitar la validación. Desde esta página también se puede borrar al empleado, no sin antes recibir una alerta recordando que se perderán todos los datos.

Historial de solicitudes individuales

Como extra para aumentar la comodidad de los usuarios administradores y supervisores, hemos creado una página para ver el historial de solicitudes individuales de un trabajador. Por coherencia, quien acceda a esta página solo puede ver los datos presentes, y no aceptar ni denegar las solicitudes que sean pendientes puesto que los administradores y los supervisores aceptan solicitudes diferentes, como veremos en el apartado de *solicitudes recibidas*.

CRUD de categorías

Otro aspecto importante en la gestión de los administradores son las CRUD de las categorías. Las categorías sirven para clasificar a los empleados y en un principio iban a permitir asignar un convenio a una categoría concreta.
El administrador puede acceder a la lista de categorías desde el panel de navegación lateral. Estas categorías se pueden buscar por nombre y cada una de ellas se puede editar. El formulario de edición y de creación son muy similares puesto que hay un solo *v-text-field,* con las mismas reglas de validación (menos de 50 caracteres).
Hemos decidido que no se puedan borrar categorías porque los usuarios deberían tener una categoría en todo momento. Si se quisiera cambiar la categoría de algún usuario en concreto, siempre es posible crear una nueva o editar las existentes y añadírsela al usuario en el formulario de edición de usuario.

Lista de solicitudes recibidas

Como veremos próximamente, el usuario supervisor solo gestiona las solicitudes de vacaciones de los usuarios que supervisa. Por tanto, el resto de solicitudes posibles las gestiona el usuario. Estas serían: solicitudes de baja, solicitudes de tipo ‘otros’ y cualquier solicitud de vacaciones de un usuario que no tenga supervisor.
El administrador, igual que el supervisor, puede ver quién envió la solicitud, la descripción y las fechas, el estado de la solicitud, el tipo y los archivos adjuntos. También puede aceptar o denegar las solicitudes pendientes.

Generación y gestión de nóminas

El administrador puede listar todas las nóminas que hayan disponibles en la base de datos. Estas nóminas son generadas automáticamente para el mes anterior si es que no se han generado en otro momento. Esto se consigue haciendo una llamada a un *endpoint* que nos confirma si ya se han generado o no las nóminas para el mes anterior (generamos las del

mes anterior porque las del mes actual no están completas del todo), si ya se han generado simplemente las guardamos en el *store* de **Vuex**. En el caso de no haberse generado, llama a otro endpoint que recorre todos y cada uno de los convenios, jornadas para ese mes y solicitudes de baja de los usuarios para generar la nómina correspondiente para cada uno.

Si entramos un poco más en detalle podemos decir que para el cálculo de la nómina se ha hecho un recorrido por las jornadas laborales del mes en el que se quiere calcular la nómina del usuario. Antes de hacer esto se calcula la cantidad de días laborales que hay en ese mes, para después compararlo con la cantidad de días trabajados del empleado.

En el caso de que haya menos días trabajados que jornadas laborales ese mes, entonces se le realizará un cómputo para reducirle en un porcentaje el sueldo total. Si tiene horas de más alguna jornada, se le sumará al sueldo total teniendo en cuenta que las horas extra se cobran a un precio mejor.

Por último, cuando el administrador pague la nómina, le dará la opción al empleado y a sí mismo de descargar esa nómina en formato PDF para poder visualizarla mejor y más en detalle.

Usuario supervisor

Panel de inicio

El panel de inicio de un supervisor es muy similar al de un administrador, por ello se han implementado ambos dentro de un mismo componente, reutilizando así este y siendo más eficiente con la gestión de componentes en la aplicación. El control de lo que se le debe mostrar a uno y al otro no se realiza mediante la directiva *v-if* de Vue.

La única diferencia que tiene el panel de inicio de un supervisor con la de un administrador es que el primero verá además de la tabla de todos los empleados que ve el administrador, otra tabla de sus misma características donde solo aparecen los empleados que están bajo su supervisión. El componente tabla también se reutiliza para la implementación de ambas tablas, la única diferencia entre estas es que en la tabla de empleados bajo supervisión, antes de mostrar los datos se hace un filtrado de los empleados según su supervisor.

Página de solicitudes de trabajadores

Las solicitudes de los trabajadores pertenecientes al equipo de un supervisor, le aparecen en una lista al mismo. Este supervisor será el que gestionará estas solicitudes en base al calendario (hecho con componente *v-calendar* de **Vuetify** y **JS** para gestión de los eventos y navegación por el calendario) mostrado bajo la lista de solicitudes. Este calendario sirve más que nada para poder visualizar un poco mejor el periodo de vacaciones y como solapan con ciertas fechas o entre empleados. Deberá gestionar correctamente las vacaciones de sus trabajadores para tener disponibilidad de ellos en diferentes fechas según las necesidades de la empresa. El supervisor puede filtrar por las solicitudes mediante cualquiera de sus campos.

Usuario trabajador

Panel de inicio

En el panel de inicio del usuario trabajador, se puede observar información diversa con respecto al usuario. En primer lugar, el usuario puede comenzar su jornada laboral mediante un botón, esto hace que de manera muy sencilla y rápida se pueda comenzar y terminar la jornada laboral. Después, observamos datos como las horas trabajadas en la empresa, diversos datos personales, compañeros de equipo, etc.

```
async fichar() {
  var ultima = this.$store.state.jornadas[0].fecha
  var hoy = new Date()
  if(hoy.getMonth() + 1 < 10) {
    var mes = '0' + (hoy.getMonth() + 1)
  } else {
    var mes = hoy.getMonth() + 1
  }
  var fecha = hoy.getFullYear() + '-' + mes + '-' + hoy.getDate()
  if(ultima == fecha) {
    this.negativo = true;
  }
  else {
    try {
      const response = await axios.
        post('/api/startJornada/' + this.user.id,{
          _token: this.$store.state.csrfToken
        }, {
          headers: {
            'Authorization': 'Bearer ' + store.state._token
          }
        }
        ).then(response => {
          this.$store.dispatch('fetchCompletada')
        });
    } catch (error) {
      console.log(error);
    }
  }
},
```

Para que el usuario pueda fichar, debe no haber fichado hoy con anterioridad debido a que el cálculo de las nóminas y las horas trabajadas, se realiza en base a las jornadas de cada mes, porque si no es así podría haber un desfalco en jornadas por cada usuario.

Página de solicitudes realizadas (trabajador y supervisor)

En esta página, el usuario puede revisar las solicitudes que ha realizado, filtrar por cualquiera de los campos e incluso ordenar los campos de la lista. Las solicitudes que ha realizado tienen su estado para que el usuario lo sepa en todo momento. También tendrá la posibilidad de borrar las solicitudes que todavía estén pendientes si quisiera. En cada solicitud aparece un botón para descargar el justificante asociado a la misma.

Por otro lado, aparece un botón para realizar una nueva solicitud arriba de la lista que redirige directamente al formulario para crear una nueva solicitud.

Formulario de nueva solicitud (trabajador y supervisor)

El formulario para una nueva solicitud, se realiza mediante 5 campos. El usuario puede elegir el tipo de solicitud que quiere enviar, una pequeña descripción, las fechas de inicio y fin y, por último, puede añadir un documento que sirva como justificante o similares. Este documento es opcional. Esta solicitud será enviada al administrador o a su supervisor según el tipo de la misma. Las solicitudes de vacaciones se enviarán a su supervisor, mientras que las otras, al administrador.

Problemas encontrados y solución

El problema más llamativo era el cálculo de nóminas por la complejidad del asunto. Había que tener en cuenta muchas variables propias del convenio (sueldo base, extra, horas extra…) además de las horas reales trabajadas por el usuario. Se resolvió el problema creando una tabla *jornadas* que guarda las horas y días trabajados por cada usuario.
Un problema común en los trabajos de clase es el problema de la coordinación del equipo: al contrario que en un proyecto real, los programadores nos encontramos cada uno en su casa y no nos reunimos diariamente. Eso causa problemas de coordinación, errores que no se comunican, mejoras que se quedan sin hacer… La única solución a esto es la completa implicación de todos los miembros del equipo y un amplio conocimiento del estado de la aplicación.

Mejoras y ampliaciones

Una mejora que teníamos en mente desde el principio del proyecto era poder subir y bajar documentos en formato .pdf para gestionar datos. Esto ha resultado siendo bastante complicado y en algunas partes de la página estamos insatisfechos por este tema. Por ejemplo, una posible mejora sería poder subir el currículum de un trabajador cuando se crea, y poder acceder a él más tarde. Por otro lado, es posible bajarse justificantes de las solicitudes, y también generar un pdf con la nómina.
Otros detalles que se podrían añadir es la gestión de accesos para que el supervisor pudiera ver en tiempo real quién se encuentra trabajando en ese momento. También, la creación de un cronómetro que mostrara las horas y minutos trabajados.
Otra posible mejora es que el usuario pueda ver más detalles cuando recibe un

Referencias

A continuación se listan los *links* a los espacios de trabajo usados para el desarrollo del sistema, como son el sistema de control de versiones y el tablero de organización de tareas. Adicionalmente se ha vuelto a mencionar el link a seguir para visitar la aplicación web.

- **GitHub**: https://github.com/berengueradrian/IWEB-ERP
- **Tablero Kanban de Trello**: https://trello.com/b/7CCXLxRA/iweb-rrhh
- **Web:** http://165.232.80.121