Kit de herramientas corporativas para la productividad

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: José María Amusquívar Poppe

TUTORIZADO POR:
Agustín Rafael Trujillo Pino
Antonio José Sánchez López

31 de mayo de 2021

DIRECTORA DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

Me gustaría agradecer todo el apoyo recibido por ambos tutores, por los compañeros de trabajo de la empresa que siempre han estado dispuestos a ayudar, a todas aquellas personas que han estado relacionadas de manera directa o indirecta con este proyecto y, en especial, a mi familia que han estado presentes en todo momento proporcionando ánimos y motivación cuando más se necesitaba.

Resumen

En el proceso de desarrollo de algún proyecto empresarial, es importante disponer de herramientas que automaticen y faciliten la realización de tareas. Actualmente existen muchas opciones para satisfacer estas necesidades, sin embargo, esto significa contratar varios servicios y tener cada uno por separado. Este proyecto aprovecha esta situación proporcionando al usuario final un conjunto de herramientas cuyas características a destacar son su modularidad e independencia, pues está ideado para ser integrado en plataformas empresariales de teletrabajo con relativa facilidad. Además, se han empleado herramientas estándares para esta arquitectura, con el objetivo de seguir buenas prácticas en su desarrollo.

Finalmente, cabe destacar que cada empresa podrá solicitar el desarrollo de aplicaciones a medida consiguiendo, de este modo, aumentar el número de herramientas y, a su vez, el número de usuarios finales.

Abstract

In the process of developing a business project, it is important to have the tools that automate and facilitate the completion of the tasks. Nowadays, there are many options to satisfy these needs, however, this means hiring several services and having each one separately. This project takes advantage of this situation by providing the customer with a set of tools whose important factors are its modularity and independence, as it is designed to be integrated into business telework platforms quite easily. In addition, standard tools have been used for this architecture, with the purpose of following good practices in its development.

Finally, it should be noted that each company can request custom applications, thereby increasing the number of tools and, in turn, the number of customers.

Índice general

Índice de figuras

Índice de Algoritmos

Capítulo 1

Introducción

El mundo actual se ha visto duramente afectado a raíz de un nuevo virus denominado *coronavirus*[41], el cual surgió a finales de 2019 y es altamente contagioso. Por este motivo se tuvo que aislar y cerrar la mayoría de ambientes y locales de todo el mundo con el objetivo de controlar el número de casos. La vida diaria tuvo que cambiar para muchos, pues las clases pasaron a impartirse mediante internet, los locales se vieron obligados a repartir a domicilio y la mayoría de trabajos pasaron a adoptar un mecanismo telemático[21].

Durante esta etapa, las distintas aplicaciones de teletrabajo vivieron un auge, tal es el caso de *Zoom*[47] o *Microsoft Teams*[27] (empleado por la *ULPGC*), las cuales posibilitan las videoconferencias con un gran número de personas al mismo tiempo, sin embargo, hay aplicaciones que van más allá, como es el caso de *nubii*[30], plataforma cuya propiedad pertenece a la empresa *The Singular Factory SL*[43].

Esta plataforma de teletrabajo combina las videoconferencias con una simulación de oficina, con un estilo que podría recordar al clásico juego de *The Sims*[3]. La aplicación de *nubii* proporciona a cada usuario la posibilidad de realizar reuniones con otros compañeros de trabajos, también proporciona un chat y una serie de configuraciones del avatar, consiguiendo de este modo amenizar la jornada laboral.

Sin embargo, con el paso del tiempo han empezado a surgir nuevos competidores en este terreno, por lo que es imprescindible disponer de características que hagan única la plataforma y que sean capaces de convencer, a nuevos usuarios, de emplear esta plataforma en lugar de otras. Entonces, es en este contexto en el que se sitúa el proyecto denominado por su nombre de producto como `Toolkit`.

El objetivo de este proyecto es proporcionar a los usuarios de la plataforma de *nubii* una serie de herramientas que puedan facilitar la realización de tareas o mejorar el entorno laboral del trabajador. Todo esto incluido dentro de la plataforma de teletrabajo.

Finalmente, mi principal objetivo era ser capaz de desarrollar, en solitario, toda la estructura de una plataforma web y, si es posible, empleando herramientas adecuadas e idóneas y siguiendo buenas prácticas.

Capítulo 2

Estado actual y objetivos iniciales

Para el desarrollo de esta memoria y del proyecto, se realizó una búsqueda de aplicaciones que puedan ofrecer la misma funcionalidad que el proyecto, sin embargo, no fue posible encontrar una aplicación similar al proyecto. Por lo que se procedió a analizar cada aplicación del proyecto por separado, entre las que se encontraron tres opciones viables:

- **Fomularios de Google**[18]: Posiblemente la opción más simple y eficaz para realizar formularios y poder compartirlos. Esta aplicación permite realizar encuestas con varias preguntas, recoger sus respectivos datos e incluso conectarlo con otras aplicaciones de *Google*.
- **Fast Poll**[14]: Este software proporciona una interfaz para crear encuestas con múltiples configuraciones. Sin embargo, para poder ser usada en un entorno empresarial, es requerido realizar un pago.
- **Pixabay**[34]: Esta plataforma permite la subida de fotografías de manera gratuita, además de presentar todas sus fotos mediante un mural con distintas opciones de búsqueda. También proporciona la posibilidad de descargar dichas fotografías.
- **Hojas de cálculo de Google**[19]: Esta aplicación proporciona una interfaz para organizar datos. En la empresa ésta fue empleada para realizar las reservas de espacios en la oficina.

Del listado anterior se puede observar que existen aplicaciones que puedan reemplazar algunas funcionalidades del proyecto, pero cabe destacar que la característica más importante de este proyecto es que se trata de un conjunto unificado de aplicaciones. Pues para disponer de estas aplicaciones prescindiendo del proyecto sería necesario adquirirlas por separado, lo que puede desembocar en un gasto extra por cada una de las aplicaciones contratadas.

Finalmente, por los motivos nombrados anteriormente, se concluye que se han alcanzado los objetivo iniciales definidos, ya que se ha conseguido desarrollar un servicio web que reúne una serie de aplicaciones en un mismo lugar, proporcionando al usuario final un mayor facilidad de acceso y una gran variedad de aplicaciones

Capítulo 3

Competencias específicas y aportaciones del trabajo

3.1. Competencias

Todas las competencias descritas a continuación han sido extraídas del documento *Objetivos y Competencias del GII*[12] localizado en la página web de la *ULPGC* (Universidad de Las Palmas de Gran Canaria).

- **CII02**: *Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.*

  Esto se ha cumplimentado dado que, al ser el proyecto individual, he gestionado todo lo relacionado al proyecto así como su correspondiente despliegue y puesta en marcha. Todo ello será visto en capítulos posteriores.

- **CII08**: *Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.*

  La elección de las herramientas de desarrollo pueden confirmar esta competencia, pues se ha tratado de escoger aquellos que ofrezcan una mayor eficacia y robustez (*LAMP*), además de emplear *frameworks* que ayudan a construir un código siguiendo unas buenas prácticas.

- **CII17**: *Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas.*

  En el *front-end* se ha realizado una interfaz de usuario minimalista siguiendo patrones comunes en todo el proyecto, y se ha empleado una librería para adaptar los textos del código a más idiomas, con el objetivo de alcanzar un mayor número de usuarios.

- **TFG01**: *Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.*

  Esta competencia se ve cumplimentada al realizar este proyecto de forma totalmente independiente y empleando una gran variedad de conocimientos adquiridos en la *ULPGC*, además se refuerza con su presentación y posterior defensa.

3.2. Aportaciones

3.2.1. Aportación al sector tecnológico y social

Este proyecto será accesible por todos aquellos usuarios que dispongan de *nubii* y, entre las aplicaciones desarrolladas y las que están por venir, se tiene como objetivo fomentar y crear cultura corporativa en la oficina, posibilitando que estos usuarios puedan interactuar con otros usuarios de la plataforma. Además, se ha conseguido que el proyecto sea capaz de funcionar independientemente con su servidor y cliente, y se ha tratado de estructurar siguiendo unas buenas prácticas. Por tanto, la idea de este proyecto es que en un futuro pueda funcionar como un módulo independiente y que pueda ser añadido en otras plataformas de teletrabajo, consiguiendo así alcanzar un mayor número de usuarios.

3.2.2. Aportación personal

Tener la oportunidad de trabajar en un entorno empresarial me ha servido para poder ver de primera mano cómo es el trabajo en conjunto con otro grupo de personas, pues aunque el proyecto haya sido desarrollado de forma individual, he podido realizar varias consultas a compañeros de trabajo con más experiencia que yo buscando aprender de ellos y mejorar todo lo posible el proyecto.

Finalmente, gracias a esta experiencia laboral he conseguido ampliar gran parte de mis conocimientos en cuanto a programación web, metodologías de trabajo y despliegue y seguimiento de un proyecto.

Capítulo 4

Normativa y legislación

Se ha verificado que todas las librerías empleadas en el código de este proyecto sean de código abierto (*Open source*) es decir, que puedan ser modificadas y distribuidas comercialmente sin generar ningún tipo de conflicto con derechos de autor. Entre las licencias presentes en el proyecto cabe destacar:

- **MIT**[32]: Esta licencia permite la libre modificación y distribución del software, por lo que permite incluir el software de terceros en mi propio proyecto sin necesidad de realizar ningún tipo de trámite ni cargo monetario.
- **Apache License**[38]: Al igual que la anterior, esta licencia permite la libre modificación y distribución del software sin necesidad de proporcionar regalías, con la única exigencia que se notifique a los receptores del producto que se ha empleado una licencia de este tipo.
- **GNU General Public License**[15]: Esta licencia se corresponde a la empleada en el sistema operativo *Linux*. Posee las mismas características que las anteriores, remarcando que se debe respetar el *copyleft*[16], esto significa que todos los derivados del producto deben preservar las mismas libertades de uso. Y, algo que destacar es que las dos primeras licencias definidas son compatibles con este tipo de licencia por lo que no se genera ningún tipo de conflicto.

Capítulo 5

Herramientas de desarrollo

Para poder llevar a cabo el desarrollo de este proyecto se ha empleado una gran variedad de herramientas de tipo *software*, algunas de las cuales constituyen parte de la estructura del servidor, el cual fue montado siguiendo un diseño LAMP (*Linux, Apache, MySQL y PHP*).

5.1. Máquina LAMP

EL proyecto está compuesto por un lado del servidor denominado *back-end* y un lado del cliente denominado *front-end*. Para montar el *back-end* se ha seguido un tipo de infraestructura compuesto por *Linux*[40] como sistema operativo, *Apache*[37] como servidor web, *MySQL*[33] como tecnología para la base de datos y *PHP*[42] como intérprete para la programación del servidor.

- **Linux**: Se ha empleado una máquina con *Ubuntu* 20.04 en el entorno local, y una máquina con *Linux CLI* (*Command Line Interface*) para el despliegue en pre-producción. Un aspecto importante de este sistema operativo frente a otros es que es un software sostenido y desarrollado por una amplia comunidad, motivo por el cual es de código abierto (*Open Source*), es decir, posee una licencia libre de derechos de autor.
- **Apache**: Denominado como *The Apache HTTP Server Project* se trata de un software que proporciona un servidor web gratuito y de código abierto, el cual posee la característica de ser multiplataforma. *Apache* es el encargado de procesar todas las peticiones y respuestas realizadas mediante un protocolo *HTTP*, proporcionando de esta manera acceso a la aplicación a cualquier usuario a través de un navegador web y una *URL*, ya sea pública a través de un dominio, o privada para entornos de desarrollo local. Este software fue lanzado en 1995, por lo que posee una amplia trayectoria que le otorga una mayor robustez y experiencia frente a otros servicios webs como *Nginx*.
- **MySQL**: Esta tecnología se corresponde con la base de datos y, al igual que *Linux* y *Apache*, *MySQL* también es un software gratuito y de código abierto, el cual proporciona un sistema de control para almacenar datos a través de tablas relacionales (*RDBMS* o

relational database management system). El potencial de este software es que organiza los datos a través de tablas dentro de una base de datos relacional, es decir, una base de datos cuyas tablas pueden poseer relaciones entre sí a través de claves. Por último, el lenguaje empleado para extraer, crear y modificar la información es *SQL* (*Structured Query language*).

- **PHP**: Su nombre completo es *Hypertext Preprocessor* y se corresponde con el lenguaje de programación empleado para desarrollar la lógica del *back-end*. Y, una vez más, al igual que los anteriores, este lenguaje de programación también es gratuito y de código abierto. Entre las ventajas de usar *PHP* está presente su gran disponibilidad para trabajar con distintas tecnologías de base de datos, en este caso, con *MySQL*, además de permitir crear contenido dinámico que sea capaz de interactuar directamente con la base de datos. Junto a este lenguaje se ha instalado un gestor de dependencias denominado *Composer*, el cual proporciona acceso a una gran variedad de librerías para *PHP* (Ilustración 5.1).

```
root@3068a83424d7:/appdata/wwwroot# composer info
Xdebug: [Step Debug] Could not connect to debugging client. Tried: 172.17.0.1:9003 (through xdebug.client_host/xdebug.client_port) :-(
api-platform/core                    v2.5.10   Build a fully-featured hypermedia or GraphQL API in minutes!
aws/aws-sdk-php                      3.184.7   AWS SDK for PHP - Use Amazon Web Services in your PHP project
composer/package-versions-deprecated 1.11.99.1 Composer plugin that provides efficient querying for installed package versions (no runtime IO)
doctrine/annotations                 1.12.1    Docblock Annotations Parser
```

Ilustración 5.1: Información de librerías instaladas mediante *Composer* (servidor)

De este modo, estas 4 herramientas, son capaces de trabajar conjuntamente para proporcionar una estructura robusta en el *backend*. La manera en la que estas herramientas trabajan juntas podría ser, por ejemplo, cuando un usuario accede a la página web, ésta envía una petición al servidor *Apache*, el cual recibe dicha petición y la procesa, si esta petición tiene como objetivo un fichero *PHP* del código fuente, el servidor web transmite la petición a *PHP* que se encarga de cargar dicho fichero y ejecutar el código incluido dentro de él. Si este fichero contiene alguna consulta a la base de datos, ésta se realiza a través de *MySQL*. Finalmente, una vez ejecutado el código, *PHP* retorna el resultado al servidor *Apache*, el cual la envía al navegador, es decir, al lado del cliente. Y, como es obvio, todos estos procesos son realizados en todo momento sobre un sistema operativo Linux.

5.2. Node.js

Este software se puede definir como un entorno en tiempo de ejecución asíncrono el cual está gobernado por eventos y está diseñado para trabajar con *JavaScript*. En el proyecto, *Node.js*[31] ha sido montado en un contenedor de *Docker* para que funcione en la parte del cliente. Además, al instalar este software, también se ha añadido su gestor de librerías denominado *npm*, el cual posee una gran cantidad de paquetes en su repositorio tal como se aprecia en la ilustración 5.2, lo que otorga una gran flexibilidad a la hora de desarrollar un proyecto.

```
/appdata/wwwroot # npm list
toolkit-frontend@0.1.0 /appdata/wwwroot
+-- @babel/polyfill@7.12.1
+-- @vue/cli-plugin-babel@4.5.13
+-- @vue/cli-plugin-eslint@4.5.13
+-- @vue/cli-plugin-router@4.5.13
+-- @vue/cli-plugin-vuex@4.5.13
+-- @vue/cli-service@4.5.13
+-- axios@0.21.1
+-- babel-eslint@10.1.0
+-- core-js@3.14.0
```

Ilustración 5.2: Información de librerías instaladas mediante *npm* (cliente)

5.3. Aplicaciones y extensiones (*plugins*)

5.3.1. Docker

Docker[20] es un software gratuito y de código abierto cuyo objetivo es facilitar el desarrollo, despliegue y control de aplicaciones construidas dentro de contenedores, lo cual posibilita a los desarrolladores el empaquetado de aplicaciones, estandarizando componentes ejecutables, librerías y cualquier dependencia requerida por la aplicación. De este modo, se consigue que la aplicación pueda ser ejecutado en cualquier entorno con el único requisito de tener instalado alguna versión de *Docker*.

Para poder configurar un contenedor se puede definir un fichero *Dockerfile*, como en el algoritmo 5.1, cuyo objetivo es recoger el nombre de la imagen que se instalará junto a todas las instrucciones que serán ejecutadas al activar el contenedor.

```
FROM rabbitmq:3-management

RUN apt-get update \
    && apt-get install -y vim
```

Algoritmo 5.1: Código *Dockerfile*

Y, posteriormente, es necesario definir un fichero *docker-compose.yml*, mostrado en el algoritmo 5.2, si se trabaja con más de un contenedor a la vez, en el que se debe realizar la correspondiente referencia al fichero *Dockerfile* mediante el atributo *context*.

```
rabbit:
    container_name: toolkit-rabbitmq
    build:
      context: ./docker/rabbitmq
    environment:
      RABBITMQ_ERLANG_COOKIE: h8h5C5QpXuaKMJAaa5MFjGQ
      RABBITMQ_DEFAULT_USER: toolkit
      RABBITMQ_DEFAULT_PASS: toolkit
      RABBITMQ_DEFAULT_VHOST: toolkit
    restart: on-failure
    ports:
      - 8102:15672
    networks:
      - toolkit-network
```

Algoritmo 5.2: Código *docker-compose*

Tal como se puede apreciar en la imagen anterior, dentro de un fichero *docker-compose*, existe una gran variedad de configuraciones que serán explicadas a continuación:

- **container_name**: Un nombre identificativo para el contenedor.
- **image**: Se define la imagen a usar si no se ha especificado un fichero *Dockerfile*.
- **build/context**: Especificación de la ruta relativa del fichero *Dockerfile* en el que se encuentran la imagen y los comandos a ejecutar (si es que lo tiene).
- **environment**: Aquí se especifican algunas variables del entorno de desarrollo, configuración para el depurador (*Xdebug*), variables empleadas dentro del contenedor, entre otros.
- **ports**: Relaciona un puerto de la máquina local con un puerto establecido por el cual el contenedor podrá enviar y leer solicitudes.
- **restart**: Se especifica una regla para cuando el contenedor falle.
- **command**: Se especifica una instrucción que se ejecutará justo después de levantar el contenedor.
- **volumes**: Aquí es necesario especificar si se han de copiar ficheros locales al contenedor, por ejemplo los fichero *php.ini*, o el fichero de configuración de *Xdebug*.
- **depends_on**: Establece un orden de arranque de contenedores, pues primero se ha de levantar aquel contenedor que no tenga ninguna dependencia.
- **networks**: Posiblemente la más importante, pues establece la conexión con los otros contenedores, situando todos ellos en una misma red (*toolkit-network*).
- **entrypoint**: Se puede definir como un conjunto de *command* escritos en un fichero. Por tanto, se debe escribir la ruta de dicho fichero.

Entonces, para montar el proyecto con *Docker* se han empleado un total de seis contenedores distintos, cada uno con una utilidad determinada. De estos, cinco se han construido en

el servidor (*back-end*) y uno en el cliente (*front-end*). Y, tal como se mencionó anteriormente, para que estos contenedores, o también llamados microservicios, se conecten entre sí, es necesario especificar una red externa con un identificador único, en este caso *toolkit-network*. Además, es importante establecer un volumen con el objetivo de almacenar la información de todos estos contenedores en un único lugar, en este caso denominado *toolkit-db-data*.

A continuación se procederá a listar los contenedores del proyecto, especificando su motivo de existencia:

- **MYSQL (container_name=toolkit-mysql)**: Este contenedor se encargará del almacenamiento de datos de la aplicación, empleando para ello la tecnología *MySQL*, en su versión 8.0.21. Además, se ha habilitado la comunicación mediante el puerto 3306, al igual que se han definido las credenciales a usar, y el nombre de la base de datos *toolkit*. Finalmente, se ha establecido como lugar de almacenamiento el volumen de datos generado para el proyecto denominado *toolkit-db-data*.
- **RABBITMQ (container_name=toolkit-rabbitmq)**: Este contenedor se encarga de gestionar la cola de mensajería de la aplicación, pues cuando se necesite enviar un correo electrónico, éste debe ser enviado primero a una cola de mensajería, en el que se van encolando todos lo mensajes de la aplicación, para que, posteriormente, puedan ser consumidos mediante otro contenedor explicado más adelante. Para configurar adecuadamente este contenedor es importante establecer unas credenciales y un *virtual_host* para poder encolar los mensajes.
- **MAILER (container_name=toolkit-mailer)**: Este contenedor está relacionado con el anterior pues deben trabajar juntos. Su objetivo es consumir todos los mensajes acumulados en la cola de mensajería de *RabbitMQ*, para lo cual es necesario ejecutar un comando justo después de activar este contenedor: *bin/console messenger:consume amqp_message*. Esta instrucción está compuesta por tres componentes, el primero hace referencia a la consola existente dentro del contenedor, el segundo referencia a la librería empleada para consumir los mensajes, y el tercero hace referencia al nombre de la cola desde el cual se consumen estos mensajes.
- **PHP (container_name=toolkit-php)**: Este contenedor tiene la misma estructura y composición que el contenedor anterior, sin embargo, no pueden ser ejecutados en uno sólo debido a que el *MAILER* siempre debe estar ejecutándose y, además, separándolos se consigue automatizar la ejecución de este. Para establecer su configuración se ha empleado un fichero *Dockerfile* en el que se ha cargado la imagen de PHP versión 7.4.10, y se ha declarado todas las instrucciones necesarias para instalar las dependencias y realizar las copias de ficheros. Finalmente, para poder depurar el código del servidor en la etapa de desarrollo se ha habilitado *Xdebug*, configurando determinadas variables de entorno en el fichero *docker-compose.yml*.
- **APACHE (container_name=toolkit-web)**: Este contenedor se encarga de montar el servidor web mediante la herramienta de *Apache*, para ello se ha establecido su configuración en un fichero *Dockerfile* y en un fichero de montaje del servidor denominado *httpd.conf*. De este modo, se consigue habilitar la comunicación entre el proyecto y el exterior, por este motivo es necesario asignar un puerto de escucha del contenedor que

irá conectado al puerto 80 de la máquina. Finalmente, es necesario copiar los ficheros de configuración de *Apache* desde la máquina local al contenedor para que pueda funcionar adecuadamente.

Todos los contenedores mencionados hasta entonces son lo que se encargan de montar la estructura del servidor, motivo por el cual el último contenedor a mostrar es el que alberga toda la lógica del cliente.

- **NODE (cotainer_name=toolkit-frontend)**: Este contenedor se encarga de montar toda la estructura del cliente, así que primero carga la imagen de *Node.js*, para después habilitar un puerto de escucha para todas las peticiones, 8114, el cual se conecta con el puerto 8080 de la máquina local y, finalmente, se ejecuta el algoritmo 5.3 como un *entrypoint*:

```
#!/bin/sh
npm install
npm run serve
```

Algoritmo 5.3: *Entrypoint* del contenedor del cliente

Estas instrucciones son ejecutadas cuando se activa el contenedor, con el objetivo de instalar todas las dependencias del *front-end* y, finalmente, ejecutar el servidor de *Node.js* con ayuda de *vue-cli*.

5.3.2. PhpStorm

Una de las partes más importantes en el desarrollo de un proyecto es elegir correctamente el *IDE* (*Integrated development environment*), o lo que es lo mismo el entorno de desarrollo en el que se trabajará, pues si éste dispone de variadas y eficientes funcionalidades que facilitan el desarrollo, el proyecto puede ser realizado con mayor rapidez y de una mejor manera. Por este motivo, para este proyecto se ha escogido el *IDE* denominado *PHPStorm*[23] en su versión 2021.1.4, software cuya propiedad es de la empresa checa *JetBrains*. A pesar de ser un software de pago, gracias al correo institucional de la universidad, se consiguió obtener una versión educativa funcional.

A continuación se enumeran algunas características de *PHPStorm*:

- Está preparado para trabajar con distintos marcos de trabajos o *framework*, tales como *Laravel*, *Symfony*, *Drupal*, entre otros.
- Ofrece un buen nivel de auto-completado, consiguiendo así, una mayor rapidez a la hora de escribir código, y también es capaz de trabajar con extensiones (*plugins*) y/o herramientas adicionales como es el caso de *EsLint*, que se define como un analizador de código cuyas reglas pueden ser definidas siguiendo buenas prácticas, y será explicado posteriormente.
- Puede integrarse un servidor de base de datos para poder gestionar las tablas existentes desde el propio *IDE*, disponiendo de una gran variedad de tecnologías como *MySQL*,

MariaDB, *SQLite*, entre otros.

- No sólo se limita al *back-end*, pues también dispone de otra gran variedad de utilidades para el *front-end*, como puede ser la disponibilidad de lenguajes como *CSS*, *HTML*, *JavaScript*, *TypeScript*, y, además es compatible con *frameworks* como *Vue.js* o *React*.
- Es capaz de manipular una gran cantidad de tipo de ficheros, tanto *YAML*, *Dockerfile*, *JSON*, *JS*, entre otros, y cada uno posee su respectivo auto-completado.
- Por último, es capaz de integrar depuradores, como es el caso de *Xdebug*, dentro del *IDE*, y, aún mejor, dentro del mismo contenedor de *PHP* instalado. Este depurador fue ampliamente utilizado a la hora de desarrollar el *back-end*, ya que posibilita la consulta de variables y el ciclo de ejecución en tiempo real.

Finalmente, como se aprecia en la ilustración 5.3 el proyecto se ha estructurado en dos partes dentro del *IDE*, una para el servidor y otra para el cliente.

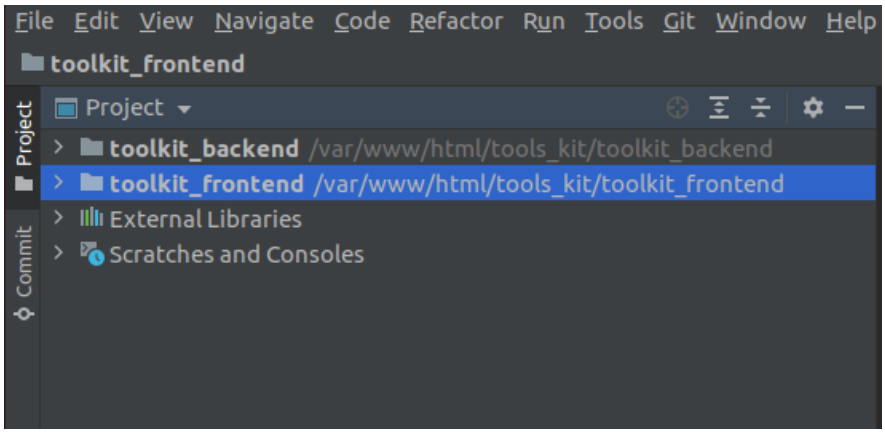


Ilustración 5.3: Proyectos creados en *PhpStorm*

5.3.3. Xdebug

Xdebug[9] es una herramienta de depuración destinada a ser usada junto a *PHP*. Este software posee distintas funcionalidades, tales como una protección contra bucles infinitos, acceso a las variables del entorno, seguimiento detallado del ciclo de ejecución, entre otras más. Configurar este depurador dentro de *PHPStorm* no conlleva mucha complejidad, pues basta con editar algunos ajustes, sin embargo, dado que se está trabajando con contenedores de *Docker*, es necesario establecer algunas configuraciones adicionales, tal como se aprecia en el algoritmo 5.4, donde se detallan las variables de entorno dentro del fichero de configuración *docker-compose.yml*, en específico dentro del contenedor de *PHP* nombrado como *toolkit-php*.

```
environment:
  PHP_IDE_CONFIG: serverName=Docker
  PHP_XDEBUG_ENABLED: 1
  XDEBUG_CONFIG: client_host=172.17.0.1 client_port=9003 # Linux users
  #       XDEBUG_CONFIG: remote_host=host.docker.internal remote_port=9005 # MacOS users
```

Algoritmo 5.4: Configuración de *Xdebug*

Tal como se observa en el algoritmo 5.4, la configuración varía según el sistema operativo en el que se encuentre el desarrollador, en este caso, se habilita únicamente la configuración para *Linux*. Una vez se ha instalado la extensión, simplemente se sitúa un punto de detención en la línea de código deseado y se habilita el modo escucha para captar la solicitud, cuya ejecución se corresponde con la ilustración 5.4.

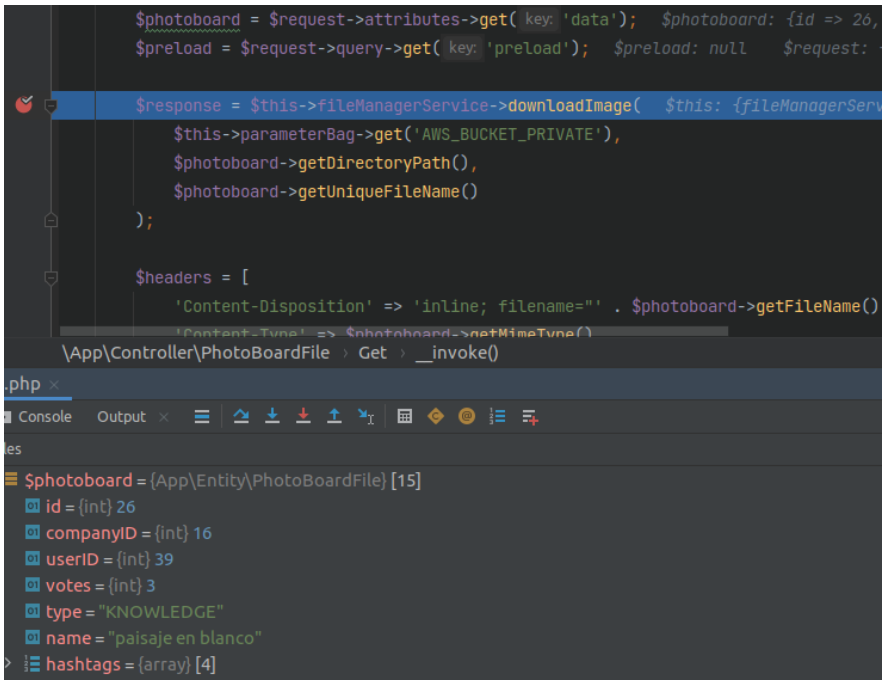


Ilustración 5.4: *Xdebug* ejecutándose y mostrando los datos del entorno

5.3.4. EsLint

EsLint[29] es otra extensión utilizada en el lado del cliente cuyo objetivo es verificar e identificar problemas de código, siguiendo para ello unas reglas y patrones definidos en su respectivo fichero de configuración *.eslintrc.js*. El *IDE* remarca estos errores con un subrayado rojo y evita que el proyecto sea compilado hasta que estos errores sean solucionados u omitidos manualmente, funcionando únicamente en la etapa de desarrollo. La guía de estilos utilizado para el desarrollo de este proyecto fue el de *Airbnb*[6], puesto que posee bastante popularidad entre los desarrolladores y, como se muestra en el algoritmo 5.5, proporciona reglas contundentes y eficaces como por ejemplo: restringir la reasignación de variables, limitar

el máximo de caracteres y atributos por línea, obligar la declaración de objetos en múltiples líneas, entre otros.

```
rules: {
'no-param-reassign': [
  'error',
  {
    props: true,
    ignorePropertyModificationsFor: [
      'state',
    ]
  }
],
'max-len': ['error', { code: 125, ignoreStrings: true }],
```

Algoritmo 5.5: Configuración de *EsLint*

En conclusión, el objetivo de emplear esta librería es el de generar un código que siga unas buenas prácticas.

5.3.5. GitHub/GitLab

Para poder implantar y gestionar un control de versiones del proyecto se ha empleado *Git*[2], en este caso, tanto *Github* para uso personal como *GitLab* para uso empresarial. Con este software es posible organizar el proyecto de buena manera, por ejemplo creando distintas ramas de trabajo, de las cuales, alguna rama podría ser utilizada para la versión de desarrollo, otra para la de pre-producción y otra para la de producción. Además, también es posible organizar el trabajo por etapas o *commits*, pudiendo volver a alguna etapa anterior si hubiese sucedido algún error. Además, una de las ventajas de emplear *Git* es que puede ser ejecutado desde la línea de comandos de Linux o, mejor aún, desde el propio *IDE* empleado (*PHPStorm*).

- **GitHub**: Se trata de una plataforma empleada para alojar proyectos, posibilitando el desarrollo cooperativo entre usuarios de la plataforma. Es de uso gratuito, aunque es necesario disponer de unas credenciales para poder alojar proyectos y colaborar en otros. Por mi parte, esta ha sido la plataforma usada para el almacenamiento de la mayoría de los proyectos realizados en la universidad.
- **GitLab**: Cumple las mismas funciones que *GitHub*, sin embargo, esta plataforma está enfocada a un uso empresarial y más profesional, razón por la cual es de pago. Una ventaja de este software es que dispone de la incorporación de una herramienta para la Integración y Despliegue continuo y automatizado (*CI/CD*), cuyo objetivo es el de automatizar la integración de nuevas características en el proyecto, y su respectivo despliegue. Algo de suma importancia para poder escalar un proyecto.

Finalmente, el código del proyecto se encuentra separado en dos repositorios conjuntos del modo que se presenta en la ilustración 5.5, uno perteneciente al *front-end* y el otro al *back-end*, de la misma manera que se estructura en el *IDE*.

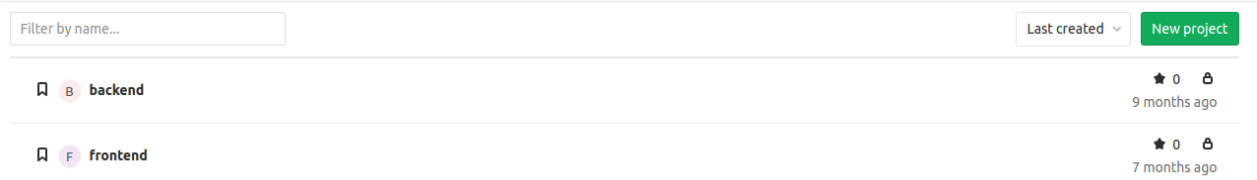


Ilustración 5.5: Repositorios creados en *GitLab*

Dentro de cada repositorio se encuentra el código estructurado por ramas, donde cada una de ellas ha sido empleada para desarrollar una determinada característica del proyecto, además de otras ramas fijas para el entorno de desarrollo (*DEV*), pre-producción (*PRE*) y producción (*PRO*). Finalmente, para añadir mayor seguridad al proyecto, todo el código subido a *GitLab* se ha realizado mediante el uso de unas claves SSH (*Secure Shell Protocol*).

5.3.6. Make

Para hacer más sencilla y ágil la gestión de los contenedores se ha empleado la herramienta de automatización *Make*. El objetivo de este software es automatizar la ejecución de código mediante el uso de reglas, las cuales realizan una determinada tarea. Para poder utilizarlo es necesario crear un fichero llamado *Makefile*, en el cual se definen una lista de reglas presentadas en el algoritmo 5.6. Para poder ejecutar estas reglas desde la línea de comandos se deben invocar mediante las instrucciones `make 'target'` o simplemente `make`, donde la primera instrucción ejecutará, solamente, la regla definida por `'target'`, mientras que la segunda las ejecutará todas.

Las reglas definidas para este proyecto son:

```
container=toolkit-php

help: ## Show this help message
    @echo 'usage: make [target]'
    @echo
    @echo 'targets:'
    @egrep '^(.+)\:\ ##\ (.+)' ${MAKEFILE_LIST} | column -t -c 2 -s ':#'

run: ## Start the containers
    docker network create toolkit-network || true
    docker-compose up -d

stop: ## Stop the containers
    docker-compose stop

restart: ## Restart the containers
    $(MAKE) stop && $(MAKE) run

build: ## Rebuilds all the containers
    docker-compose build

enter: ## Access to container -> default PHP | container=name
    docker exec -it $(container) bash
```

Algoritmo 5.6: Contenido del fichero *Makefile* del servidor

- **help**: Muestra la lista de comandos como ayuda.
- **run**: Intenta crear la red (si es que aún no se ha creado), y luego activa todos los contenedores definidos en el fichero *docker-compose.yml*.
- **stop**: Detiene todos los contenedores activos y presentes en el fichero *Docker*.
- **restart**: Es una combinación de las reglas *stop* seguido de *run*.
- **build**: Construye o reconstruye todos los contenedores del fichero *docker-compose.yml*.
- **enter**: Utilizado para acceder dentro de los contenedores que se están ejecutando. La variable *container* define el contenedor al que acceder, por defecto es al contenedor de *PHP*.

5.3.7. Ficheros *.env*

Para poder gestionar de manera eficiente y segura las distintas variables globales del proyecto se han empleado ficheros de entorno *.env*, tanto en el servidor como en el cliente. Por lo que en ambas partes existen dos ficheros de este tipo, uno denominado *.env* y otro *.env.local*. Este último fichero no se comparte en ningún momento fuera del proyecto, además que es omitido por el fichero *.gitignore*, por lo que se pueden escribir aquí todas las variables sensibles del proyecto como contraseñas, mientras que el fichero *.env* sí se comparte con el exterior, como por ejemplo *Git*, por lo que no debe contener información sensible, simplemente la estructura de las variables.

```
###> For JWT in API ###
VUE_APP_TOOLKIT=$api_location
VUE_APP_TOOLKIT_USER=$user_toolkit_account
VUE_APP_TOOLKIT_PASSWORD=$password_toolkit_account
###< For JWT in API ###
```

Algoritmo 5.7: Ejemplo de un fichero *.env*

En el algoritmo 5.7 se puede observar distintas variables globales del proyecto asignado a valores no válidos, simplemente de referencia. Por lo que el fichero *.env.local* contendrá las mismas variables pero con los valores correctos, añadiendo una mayor seguridad a los datos sensibles del proyecto. Finalmente, la ventaja de trabajar con ambos ficheros es que cualquier valor del fichero *.env* será sobrescrito por su correspondiente valor presente en el fichero local.

5.3.8. Postman

Postman[35] puede definirse como una *API* (*application programming interface*) de testeo cuyo principal objetivo es simplificar el flujo de trabajo de un proyecto y el de ayudar al desarrollador a comprobar el correcto funcionamiento de éste permitiendo la fácil y rápida realización de peticiones o solicitudes (GET, POST, DELETE, entre otros) al *back-end* del

proyecto. Además, este software proporciona entornos de trabajo, la posibilidad de crear colecciones de solicitudes, crear variables globales, entre otras funcionalidades más.

Para el desarrollo del proyecto se han creado tres colecciones (Ilustración 5.6) y dos entorno de trabajo:

- **Colecciones**:

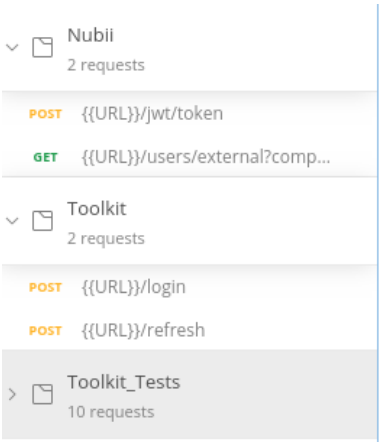


Ilustración 5.6: Colecciones de peticiones en *Postman*

  - **Nubii**: Compuesto por dos peticiones: un *POST* para enviar las credenciales de usuario al proyecto de *nubii*, y así obtener un par de *tokens* válidos; y un *GET* para obtener el listado de usuarios registrados en la plataforma, lo cual es necesario para algunas de las aplicaciones desarrolladas.
  - **Toolkit**: Compuesto también por dos peticiones, ambas *POST*: una para enviar las credenciales de usuario al proyecto y obtener un par de *tokens* válidos; y la otra es usada para actualizar este par.
  - **Toolkit_Tests**: Aquí se desglosan todas las peticiones hechas al proyecto, tanto *POST*, *GET*, *PUT*, *PATCH* o *DELETE*.
- **Entornos**:
  - **Nubii Desarrollo**: Aquí se almacenan como variables de entorno el par de *tokens* de acceso a *nubii*, además de una *URL* base a la que se solicita estos datos.
  - **Toolkit Desarrollo**: Misma estructura que en *nubii*, sólo que destinado para trabajar con el proyecto.

Finalmente, el objetivo de emplear *Postman* fue el de comprobar que todas las peticiones sean correctas, y que el flujo de trabajo también lo sea. Una alternativa a este software es la documentación del proyecto localizada en */api/v1/docs*, la cual fue generada mediante *Api Platform*, un marco de trabajo web del que se hablará más adelante.

5.3.9. ClickUp

Para poder gestionar los tiempos y tareas del proyecto se ha empleado una metodología de desarrollo ágil de tipo *Scrum*, en el que se establecieron *sprints* o jornadas de desarrollo semanales, al final de las cuales se realizaba un análisis del progreso conseguido dicha semana y se marcaban nuevos hitos para el *sprint* siguiente. Para llevar a cabo esta metodología se empleó tanto la plataforma *nubii* para realizar las reuniones semanales y/o diarias en conjunto con la plataforma web denominada *Clickup*[4]. Esta plataforma proporciona una serie de herramientas para formar equipos de desarrollo y juntos poder generar el tablón de tareas a realizar, estructurándolas en distintas etapas, como se muestra en la ilustración 5.7, tanto si éstas se encuentran en progreso, bloqueadas por otras, en revisión o listas para probarlas.

Ilustración 5.7: Algunas columnas de tareas de *ClickUp*

Por último, cada tarea se caracteriza por un código de colores, en el que amarillo significa que pertenece al presente *sprint*, azul que está en progreso, naranja que ha sido desarrollado y está listo para probarse, y gris significa que está pendiente para ser asignado a un futuro *sprint*.

5.4. Librerías destacadas

En esta sección se procederá a detallar el uso de las librerías más importantes y mayor peso en cada lado del proyecto.

5.4.1. Back-end

- **Api Platform**[25]: Se puede definir como un *framework*, el cual está construido en base a *PHP*, y es capaz de funcionar conjuntamente con el *Symfony*, consiguiendo, de este modo, una mejor estructuración del código. El objetivo de este software es acelerar y simplificar el desarrollo de código en la parte del servidor, automatizando diversas tareas de manipulación de los datos.

Este software puede ser empleado de varias maneras dentro del proyecto, pues admite especificar su configuración mediante anotaciones, ficheros *YAML*, o inclusive ficheros *XML*. En el caso del proyecto, se ha mantenido la configuración mediante anotaciones, pero con vistas a cambiarlo a ficheros *YAML*, pues este aporta una mayor independencia y menos sobrecarga a las entidades, además que contribuye a cumplir con los principios *SOLID*. Estas anotaciones se encuentran dentro de los ficheros de las entidades del proyecto y poseen una estructura específica análoga a comentarios de varias líneas de C++ o JAVA.

Las configuraciones a especificar está definido por *ApiResource*:

- **Paginación (*pagination*)**: Se puede especificar parámetros sobre la paginación, tal como desactivarla cuando sea necesario, o especificar el número de elementos que se devolverán por página, con el objetivo de aligerar la respuesta al cliente. El número de página debe ir especificado en la *query* de la petición junto a todos los otros parámetros.
- **Colecciones (*collectionOperations*)**: En este apartado se especifican los *endpoints* de la entidad.

```
*       collectionOperations={
*       "add"={
*          "method"="POST",
*          "path"="/photoboard/file/add",
*          "controller"=Add::class,
*          "deserialize"=false,
*          "denormalization_context"={"groups"={"photoBoardFile_post"}},
*          "normalization_context"={"groups"={"photoBoardFile_ok"}},
*          "openapi_context"={
*              "summary"="Add a new image",
*          },
*      },
```

Algoritmo 5.8: Colecciones empleando *Api Platform*

En el algoritmo 5.8 se muestra una petición ”POST”, cuyo objetivo es añadir una nueva instancia a la base de datos. Dentro de este *endpoint*, se puede definir una serie de atributos, como puede ser el tipo de petición, la ruta del *endpoint*, también se puede especificar si se trabajará con un controlador propio localizado en la carpeta *src/Controller*, y si se procesará la petición o no antes de ser enviada a su controlador.

Además, un aspecto importante en estas configuraciones, es que se pueden añadir grupos de *serialización* y *deserialización*, el primero engloba todas aquellas propiedades que son devueltas en la respuesta al cliente, y el segundo especifica aquellas propiedades que se espera recibir en la petición.

Cabe destacar que las colección no sólo aceptan peticiones de tipo *POST*, sino también aceptan de tipo *GET*, con el fin de obtener todas las instancias de la entidad, pudiendo procesar la respuesta con algún filtro que se explicará posteriormente,

y/o con la paginación habilitada.

Y, finalmente, también se puede añadir información adicional con *openapi_context*, para detallar y habilitar más funciones en la documentación generada y localizada en *api/v1/docs*.

- **ItemOperations**: En este apartado se especifican los *endpoints* que afectan a una única instancia, identificada mediante un identificador único. Las operaciones realizadas en el proyecto fueron de tipo *GET* para obtener el objeto, tipo *PUT/-PATCH* para modificar el objeto o *DELETE* para eliminar la instancia. Al igual que las colecciones y como se presenta en el algoritmo 5.9, estas operaciones aceptan las mismas configuraciones, sin embargo, todas estas peticiones deben llevar el identificador de la instancia en la *query*.

```
*     itemOperations={
*      "get"={
*          "path"="/photoboard/file/get/{id}",
*          "controller"=Get::class,
*          "openapi_context"={
*              "summary"="Get an image resource",
*          },
*      },
*      "delete"={
*          "path"="/photoboard/file/delete/{id}",
*          "openapi_context"={
*              "summary"="Delete an image resource",
*          },
*      },
```

Algoritmo 5.9: Operaciones simple empleando *Api Platform*

Dado que se ha estado trabajando con anotaciones, los distintos grupos de *serialización* y *deserialización* se incluyen en este mismo fichero, declarando la pertenencia de cada propiedad a un grupo mediante anotaciones por encima de dicha propiedad del mismo modo que se presenta en el algoritmo 5.10. Por lo que, una propiedad puede pertenecer a los grupos que sean necesarios, simplemente se requiere añadirlos separándolos por comas.

```
/*
 * @var int
 * @Groups({"photoBoardFile_post", "photoBoardFile_get_all"})
 */
private int $userID;
```

Algoritmo 5.10: Operaciones simple empleando *Api Platform*

La última funcionalidad a explicar de *Api Platform* es la posibilidad de aplicar una serie de filtros y ordenación a las respuestas que contengan varias instancias. Pues como se muestra en el algoritmo 5.11, este software permite filtrar los datos mediante alguna de sus propiedades, especificando si se trata de un filtro numérico, booleano, de texto o, inclusive, de fecha. Todos esto filtros permiten añadir parámetros adicionales para

especificar el tipo de búsqueda a realizar, ya sea por un rango de fechas, diferencia de valores o por búsqueda de términos. Estos filtros deben ser pasados en la *query* de la petición.

```
* @ApiFilter(SearchFilter::class, properties={"type": "exact", "name": "partial"})
* @ApiFilter(OrderFilter::class, properties={"votes"})
```

Algoritmo 5.11: Filtrado y ordenación con *Api Platform*

Finalmente, en la ilustración 5.8 se muestra un ejemplo combinado de estas opciones de filtrado, ordenación y paginación en una misma petición,

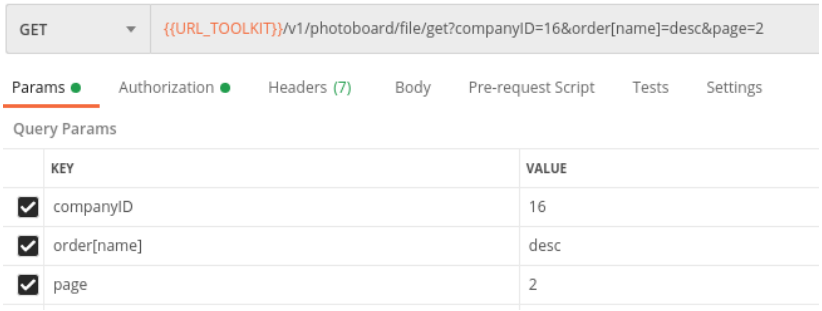


Ilustración 5.8: Ejemplo de petición con *Api Platform*

- **Doctrine**[10]: Esta librería puede definirse como un kit de librerías destinadas, en su mayoría, al almacenamiento de datos y manipulación de objetos. Para el desarrollo de este proyecto se ha empleado *Doctrine ORM* (*object-relational mapper*) cuyo objetivo es proporcionar una persistencia transparente para los objetos, y su principal ventaja es que realiza un mapeo de los objetos separando así la lógica de los objetos de su respectiva persistencia en la base de datos.

  Antes de seguir es importante definir que son las entidades, ya que este término es ampliamente usado en el proyecto. Una entidad es básicamente un objeto *PHP* identificado por una clave o identificador único. Éstas están definidas en ficheros *PHP* como clases, dentro de la ruta *src/Entities*.

```
/**
 * @ORM\Entity(repositoryClass=PhotoBoardFileRepository::class)
 * @ORM\HasLifecycleCallbacks()
```

Algoritmo 5.12: Definición de *Doctrine* en una entidad

Y, puesto que se está trabajando mediante anotaciones y como se puede observar en el algoritmo 5.12, lo primero y requerido a definir dentro de la entidad es que cada una de ellas disponga de un repositorio, con determinadas acciones de búsqueda como *findBy*, *findAll*, entre otros. Lo siguiente a definir es si la entidad posee algún ciclo de vida, es decir, alguna función que se ejecuta cada vez que se manipula la entidad, en este caso, un caso evidente de uso sería poseer una función para actualizar la fecha y hora de modificación de la entidad.

Una vez se ha definido estos parámetros, es hora de especificar las propiedades que contendrá la tabla de datos, para ello es importante añadir las correspondientes anotaciones sobre las propiedades de la entidad. Algunas de las configuraciones que aceptan estas anotaciones son:

- **ID**: Establece la propiedad como clave primaria, cuyo valor puede ser generado automáticamente.
- **NULO**: Es posible especificar que una propiedad puede o no ser nula.
- **TIPO**: Especifica el tipo de datos, ya sea entero, *string*, *array*, *Blob*, entre otros.
- **CHOICE**: Establece que el valor recibido debe coincidir con alguno de una lista establecida.
- **FILE**: Acepta un fichero, siempre y cuando éste cumpla las restricciones especificadas en su definición (algoritmo 5.13).

```
/**
* @Assert\File(
*     maxSize = "7M",
*     mimeTypes = {
*          "image/jpg",
*          "image/jpeg",
*          "image/png",
*      },
*     groups={"photoBoardFile_post"},
* )
```

Algoritmo 5.13: Restricción para la subida de ficheros

Algo de suma importancia en una base de datos es poder establecer relaciones entre tablas, para conseguir esto, también es posible especificarlo mediante anotaciones, sin importar si la relación es de uno a muchos, muchos a uno, entre otros. Junto a esta relación, es importante especificar a que tabla y columna referencia como clave foránea, y establecer el tipo de borrado, ya que si se elimina un elemento de la asociación, es probable que se dese eliminar el otro o colocar a nula esa relación.

```
/**
 * @ORM\OneToMany(targetEntity="App\Entity\OptionPoll", mappedBy="poll")
 */
private Collection $options; // Propiedad de la encuesta (Poll)

/**
 * @ORM\ManyToOne(targetEntity="App\Entity\Poll", inversedBy="options")
 * @ORM\JoinColumn(name="poll_id", referencedColumnName="id", onDelete="CASCADE")
 *
 * @var Poll
 * @Groups("option_add")
 */
private Poll $poll; // Propiedad de la opcion (OptionPoll)
```

Algoritmo 5.14: Relaciones entre tablas

En el algoritmo 5.14 se aprecia que, dentro de la entidad *OptionPoll*, existe una propiedad *Poll*", cuya relación es muchos a uno, es decir, una encuesta puede tener muchas opciones pero una opción sólo debe pertenecer a una encuesta. También se observa que esta propiedad es la inversa de la colección de opciones presente en cada encuesta, por lo que es necesario establecer una clave foránea (*poll_id*).

Y, finalmente, también es importante establecer que, si la encuesta es eliminada, todas sus opciones también lo serán, esto se especifica mediante la configuración *onDelete=CASCADE*.

Finalmente, en la entidad *Encuesta* existe una propiedad llamada *$options*, la cual es una colección de opciones *OptionPoll*, y está mapeada por la propiedad *$poll* presente en cada entidad opción.

- **JWT**: JSON web token[24] es un estándar (RFC 7519) el cual define una manera compacta e independiente de intercambio de información entre dos afectados mediante un objeto *JSON*, que puede estar firmado otorgándole un punto más de seguridad. Este mecanismo de transmisión puede ser empleado para servicios de autenticación, autorización o, simplemente, el intercambio de información. En el caso del proyecto, se ha empleado *JWT* para el cometido de autorización, evitando que sólo determinados usuarios puedan acceder al proyecto.

  Dado que el proyecto no dispone de un sistema de autenticación directa, ya que hereda los datos de usuario desde *nubii*, se ha optado por habilitar un sistema de autorización, mediante el cual sólo podrán acceder al proyecto aquellos usuarios que dispongan de un *JWT* válido y, siempre y cuando, se encuentren dentro de la plataforma de *nubii*, en caso contrario, se deshabilitan todas las aplicaciones y no se obtiene ninguna autorización. Y, además, dado que el proyecto necesita obtener datos desde el servidor de *nubii*, como es el listado de usuarios de la plataforma, se requiere disponer de otro par de autorizaciones exclusivo para funcionar con la petición realizada dicho servidor. Entonces, como se aprecia en la ilustración 5.9, el proyecto dispondrá de cuatro *JWT* almacenados en la memoria del navegador, dos son *token* y los otros dos son *refresh_token*.

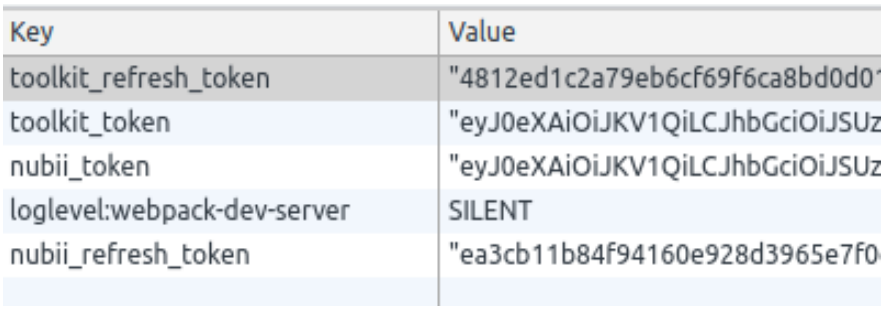


Ilustración 5.9: *JWT* almacenados en el navegador

Para poder utilizar esta tecnología en el servidor se empleó dos librerías, la primera denominada *LexikJWTAuthenticationBundle* que proporciona la lógica para generar y validar *tokens*, y la segunda librería fue *JWTRefreshTokenBundle* que se encarga de los *refresh_tokens*.

Una vez instaladas ambas librerías, el primer paso es generar una clave pública y una privada que serán las que usará la librería para autorizar usuarios. Además, es necesario establecer unas variables de entorno, entre las que destaca el tiempo de vida de los datos de autorización. Así pues, buscando seguir buenas prácticas y considerando que el proyecto sólo dispone de un mecanismo de autorización dentro de la plataforma de *nubii*, se ha establecido el tiempo de vida del *token* a 1 día, y el del *refresh_token* se ha establecido a 3 meses.

Para poder realizar este proceso de autorización adecuadamente, se ha generado un usuario y una contraseña para acceder a los datos del servidor, tanto para el *JWT* de *nubii* como para el del proyecto. Estas credenciales son almacenadas en el cliente como datos sensibles en el fichero de entorno local.

Finalmente, cuando la aplicación es ejecutada, lo primero que se comprueba es si existen *JWT* almacenados en la memoria del navegador, en caso negativo se realiza una petición al servidor con las credenciales, y en caso afirmativo, como se aprecia en el algoritmo 5.15, se procede a comprobar su validez, empleando el *refresh_token* para actualizar los *JWT* en caso de estar vencida.

```
const expiredJWT = (token) => {
  const expiration = JSON.parse(atob(token.split('.')[1])).exp;
  return Date.now() >= expiration * 1000;
};
```

Algoritmo 5.15: Comprobación de validez de *JWT*

- **AWS**: *Amazon Web Service*[1] es una plataforma de computación en la nube diseñado y ofrecido por *Amazon*, el cual incluye una gran variedad de productos y servicios, entre los cuales se encuentra *Amazon Simple Storage Service* (*Amazon S3*), utilizado en este proyecto. Este servicio proporciona una plataforma de almacenamiento ilimitado y seguro, en el cual se puede albergar cualquier tipo de fichero, páginas webs, aplicaciones móviles, entre otra gran amplitud de variedades.

  Este servicio se ha empleado específicamente para almacenar las imágenes de la aplicación *Fotos de empresa*, dado que guardarlos en la base de datos era inviable por las dimensiones de las imágenes y porque sería poco eficaz, y malgastaría recursos.

  Amazon S3 está compuesto de cubos o *buckets*, dentro los cuales se puede organizar los datos como se desee. Dado que la aplicación de *Fotos de empresa* debe ser segura, ya que son fotos privadas, se ha establecido que el cubo sea privado, por lo que para poder acceder a estos datos es necesario hacerlo con unas credenciales específicas desde un entorno adecuado. Dentro de este cubo no es posible declarar directorios como tal, sino que se generan los nombres de archivos con la ruta completa, es decir, para una imagen de la compañía con identificador 16, y con tipo de imagen *MATERIAL*, se crearía el siguiente nombre de fichero: `photoboard/16/MATERIAL/filename.jpg`, de este modo, como se muestra en la ilustración 5.10, se consigue estructurar todas las imágenes como si estuviesen organizadas por carpetas.

Ilustración 5.10: Estructura de un cubo en *Amazon S3*

Finalmente, poder conectar este servicio con el proyecto, se ha tenido que instalar la librería `aws/aws-sdk-php` en el servidor, la cual proporciona una serie de métodos para manipular los datos del cubo, eso sí, siempre accediendo con las credenciales adecuadas.

- **Imagick**[7]: Se puede definir como una extensión de *PHP* cuyo objetivo es posibilitar la manipulación de imágenes empleando para ello la librería *ImageMagick*. Esta librería ha sido y será empleada para generar imágenes de poca resolución y tamaño para utilizarlas como imágenes de esperar o *thumbnails* mientras una mejor imagen está siendo cargada.

  Su principal uso ha sido para aplicación *Fotos de Empresa*, en la cual se muestra un listado de varias fotos a la vez, hacer esto representaba ocupar una gran cantidad de memoria al mismo tiempo puesto que cada imagen puede alcanzar los 7 MB (*MegaBytes*). Para evitar que la memoria se sature, se ha optado por cargar imágenes de una resolución menor de forma asíncrona. Estas imágenes son visualmente parecidas a la original pero ocupando menos memoria y, únicamente, se obtendrá la imagen original cuando el usuario seleccione una imagen del listado para verla en tamaño completo o si desea descargar dicha imagen.

5.4.2. Front-end

- **Axios**[39]: Se puede definir como una librería presente en el lado del cliente, el cual proporciona funcionalidades para realizar peticiones a un servidor mediante el uso de *endpoints*. Este servidor puede ser tanto uno externo como el mismo *back-end* del proyecto. Y, puesto que realiza peticiones, esta librería trabaja sobre el protocolo *HTTP*, empleando un mecanismo de promesas[28]. Estas promesas se definen como objetos que representan la respuesta, tanto exitosa como errónea, de una petición asíncrona.

  En el proyecto, sea ha empleado *Axios* para realizar todas y cada una de las peticiones desde el *front-end*, tanto al servidor del proyecto como al servidor de *nubii*, empleando peticiones de tipo *GET*, *PUT*, *PATCH*, *POST* o *DELETE*. Todas estas peticiones realizadas deben emplear un mecanismo asíncrono, evitando así que se produzca un cuello de botella o una espera demasiado larga sin ninguna acción para el usuario y,

también deben llevar en su cabecera (*headers*) el *token* de la sesión, para que pueda ser autorizado en el servidor.

Todas las peticiones realizadas poseen una estructura parecida compuesta por una *URL* de tipo *endpoint* y una cabecera (*header*) que contiene el *token* de la sesión y alguna configuración adicional. Además, según el tipo de petición, puede añadirse un objeto a enviar o añadir una *query* a al *URL*.

En el proyecto se han empleado las siguientes peticiones:

- **GET**: Esta petición obtiene un único recurso de la base de datos, empleando para ello un identificador único. Un ejemplo de este uso, presentado en el algoritmo 5.16, es la petición que obtiene una imagen desde el servidor pasándole el identificador de imagen y una *query* adicional que especifica si se desea una imagen de resolución menor o la imagen original. Además, dado que se espera recibir una imagen con datos crudos, es importante remarcar que la respuesta esperada es de tipo *blob*.

```
const getImage = (id, query) => axios.get(
  `${process.env.VUE_APP_PHOTOBOARD}get/${id}${query}`,
  {
    responseType: 'blob',
    headers: authHeader(),
  },
);
```

Algoritmo 5.16: Petición *GET* al servidor

  Este tipo de petición funciona tanto si se desea obtener un único objeto o un listado de ellos, para este segundo caso se debe proporcionar algún parámetro para controlar los datos que se obtienen, como puede ser una paginación, un filtrado u ordenación de los datos.

- **POST**: Esta petición es empleada para añadir nuevas instancias en la base de datos, para ello es importante adjuntar el objeto que será añadido, el cual, como se muestra en el algoritmo 5.17, suele estar estructurado como un *JWT* o un *FormData*.

```
{
    "userID": 3,
    "username": "Jose Maria",
    "timeIn": "2021-04-16",
    "description": "Primera reserva",
    "space": "/api/v1/spaces/1",
    "companyID": 16,
    "parking": null
}
```

Algoritmo 5.17: Ejemplo de objeto enviado a la base de datos con un *POST*

- **PUT/PATCH**: Ambas peticiones son empleadas par actualizar el objeto en la base de datos, sin embargo, el primero requiere poseer todo el objeto para actualizarlo, mientras que *PATCH* sólo requiere la propiedad que va a actualizar sin necesidad de consultar ni requerir de otros parámetros.

  Tal como se cita en el estándar *RFC5789*: "The difference between the PUT and PATCH requests is reflected in the way the server processes the enclosed entity to modify the resource identified by the Request-URI. In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced". — Dusseault & Snell [11].

  En el proyecto, una petición *PUT* podría ser empleada para actualizar un objeto de tipo *SPACE*, pues este objeto sólo posee dos propiedades editables, por lo que sea cual sea la que ha editado el usuario, se procede a actualizar la entidad entera.

  En cambo, un uso para un petición *PATCH* podría ser la actualización del número de corazones (*Me gusta*) de una imagen, pues cuando el usuario pulsa sobre el respectivo ícono, sólo debe actualizarse esta propiedad en la base de datos.

  Finalmente, cabe destacar que para poder emplear este tipo de petición es necesario especificar una nueva cabecera: `application/merge-patch+json`.

- **DELETE**: Esta petición es la más simple, en cuanto a lógica, de todas las anteriores, pues simplemente se requiere enviar el identificador del objeto en la base de datos, para que, mediante el uso de *Api Platform* en el servidor, la instancia sea eliminada de su correspondiente tabla.

Dentro del proyecto, la lógica de esta librería se ha desarrollado y acoplado de forma conjunta con la lógica de la tienda de *Vuex*. Por ello es que todas las peticiones de cada aplicación se encuentran aisladas en unos ficheros específicos, localizados en `src/Services/APP_NAME/_api`.

Finalmente„ la ventaja de emplear llamadas asíncronas con *axios* es que el cliente puede seguir su flujo de funcionamiento mientras espera que la petición se complete, esto es algo importante, pues en la aplicación de *Fotografías de Empresa*, se obtiene cada imagen individualmente de forma asíncrona, y se van cargando y mostrando cada una por pantalla a medida que se completa su correspondiente petición.

- **Vuex**[8]: Se trata de una librería para *Vue.js* que funciona en base al patrón de diseño denominado Manejo de Estado, el cual busca proporcionar un espacio centralizado donde se puedan almacenar datos de la aplicación de manera global, es decir, que todos los componentes de la aplicación puedan disponer y editar estas variables de modo que el cambio afecte a todos aquellos que estén empleando dichas variables. Este espacio se denomina tienda o, como se le llama en inglés, *store*.

  Para llevar a cabo esta funcionalidad, *Vuex* proporciona una estructura específica, compuesto de cuatro tipos de eventos:

  - **State**: Se corresponde con un objeto que contiene todas las propiedades de la

tienda, lo que certifica que esta librería hace uso de la práctica denominada *Single source of truth* (Fuente única de la información), dado que los datos que se almacenan en este objeto no existen en ningún otro lugar, por lo que para poder acceder o modificarlos es necesario seguir unos pasos específicos empleando para ello los siguiente tres eventos existentes en *Vuex*.

- **Actions**: Son métodos que pueden ser asíncronos, los cuales realizan una determinada acción para que, al finalizar, se realice alguna mutación sobre alguna propiedad de la tienda, empleando para ello el evento *commit* proporcionado por *Vuex*. Es en estos métodos donde se invoca al servicio que se encarga de realizar las peticiones al servidor con *axios*.
- **Mutations**: Esta es la única manera de alterar una propiedad del estado, empleando para ello la función *commit* de *Vuex*. Cada propiedad del estado puede disponer de alguna mutación, cuya función es actualizar alguna propiedad del estado por otro valor que se la ha pasado como parámetro.
- **Getters**: Tal como su nombre lo especifica, los *getters* hacen referencia a un conjunto de funciones que devuelven cada una de las propiedades del estado.

Una ventaja de *Vuex* es su capacidad para trabajar adecuadamente con propiedades computadas de *Vue.js*, éstas, tal como su nombre lo indica, son variables dinámicas que funcionan dentro de los componentes. Al poseer un valor dinámico, estas variables deben disponer de una función *get()* y otra *set()*, tal como se observa en el algoritmo 5.18, con el objetivo de poder editarlas y consultarlas. Por este motivo, es posible crear una propiedad computada que haga referencia a una propiedad del estado de la tienda, haciendo uso de un *getter()* para el *get()*, y un *commit()* para el *set()* de la propiedad, de este modo se consigue simplificar la lógica de actualización y visualización de la tienda. Además, dado que estas propiedades son dinámicas, cada vez que la tienda sea actualizada por otro componente, esta propiedad computada, también será actualizada.

```
mode: {
  get() {
    return this.$store.getters['PhotoBoardStore/getMode'];
  },
  set(newMode) {
    this.$store.commit('PhotoBoardStore/setMode', newMode);
  },
},
```

Algoritmo 5.18: Ejemplo de propiedad computada con *Vuex*

Por otro lado, dado que el proyecto está compuesto por varias aplicaciones sin relación entre ellas, se ha generado una tienda para cada una de las aplicaciones, además de otras adicionales para almacenar datos generales que afectan a todas las aplicaciones. Las tiendas creadas se encuentran dentro de la carpeta *services*, y reciben los nombres presentados en la ilustración 5.11.

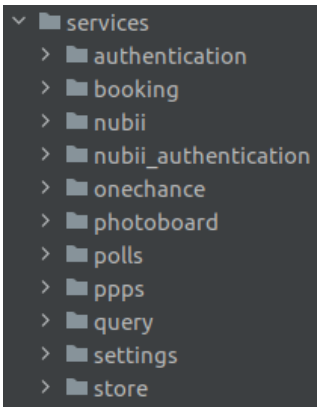


Ilustración 5.11: Estructura de las tiendas del proyecto con *Vuex*

De la imagen anterior, cabe destacar las tiendas:

- **authentication**: Existen dos tiendas de este tipo y están destinadas al tratamiento del *JWT* como autorización, tanto para el proyecto como para *nubii*.
- **nubii**: Se ha generado esta tienda para almacenar toda la información obtenida desde el servidor de *nubii*, pues estos datos son empleados en más de una de las aplicaciones.
- **settings**: Esta tienda almacena y trata las configuraciones generales del proyecto, tales como el día que se realiza el informe de la aplicación *ppps* o la normativa de la empresa empleada en la aplicación de reservas, entre otros datos.
- **store**: Esta carpeta contiene la lógica para construir la tienda general del proyecto la cual engloba todas las demás tiendas, por lo que es necesario importar y añadir todas las tiendas existentes para que puedan ser accesibles desde cualquier sector del proyecto.
- **resto**: El resto de tiendas se corresponde con cada una de las aplicaciones desarrolladas, según su nombre.

Cada tienda generada en la aplicación posee una estructura similar que se corresponde con la presentada en la ilustración 5.12.

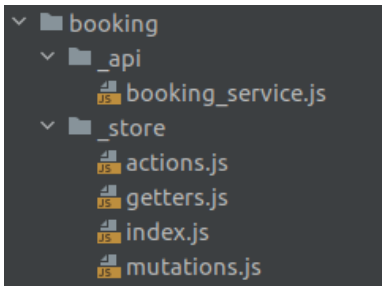


Ilustración 5.12: Estructura de una tienda del proyecto

Finalmente, la estructura de una tienda está compuesta por dos carpetas, una denominada *_api*, y otra denominada *_store*, la primera contiene todas las peticiones realizadas de la tienda mediante la librería *axios*, mientras que la segunda posee toda la estructura de la tienda, donde el fichero *index.js* se corresponde con la definición del estado, y el resto con los otros tres eventos existentes de *Vuex*. Por lo tanto, tal como se muestra en el algoritmo 5.19, para poder añadir una nueva *store* a la tienda general, se debe importar la carpeta *_store*.

```
import BookingStore from '@/services/booking/_store';
import PhotoBoardStore from '@/services/photoboard/_store';

Vue.use(Vuex);

export default new Vuex.Store({
  modules: {
    AuthStore,
    SettingsStore,
```

Algoritmo 5.19: Importación de tiendas en *Vuex*

- **router**[13]: Esta librería cumple el papel de enrutador o rúter del proyecto, siendo el software oficial diseñado para *Vue.js*. Por tanto, aprovechando este factor, *Vue Router* es capaz de proporcionar una multitud de ventajas y utilidades como rutas anidadas, la posibilidad de transferir datos ya sea en una *query* o como parámetros, una navegación fluida y, lo más importante, dispone de un historial de rutas, lo que le proporciona una mayor facilidad para navegar entre distintas vistas de una aplicación.

  Para emplear esta librería dentro del proyecto se ha definido un fichero de rutas, localizado en *src/router*. Este fichero importa todas las vistas disponibles de la aplicación y las estructura en una colección de objetos, donde cada uno se corresponde con una ruta y, como se observa en el algoritmo 5.20, posee una gran variedad de parámetros.

```
{
    path: '/u/ppps',
    name: 'PPPS',
    component: PPPS,
    redirect: {
      name: 'FillForm',
    },
    children: [
      {
        path: 'fill',
        name: 'FillForm',
        component: Fill,
      },
```

Algoritmo 5.20: Objeto que representa una ruta en *Vue Router*

En la imagen anterior se aprecia un fragmento de código de las rutas de la aplicación *ppps*, compuesta por una ruta general (padre) y una hija. Todas las rutas de las distintas aplicaciones del proyecto siguen esta misma estructura, y poseen tres parámetros

importantes: el nombre de la ruta (*name*), la ruta (*path*) y la vista que se carga en dicha ruta (*component*). Además, todas las rutas padres poseen un parámetro adicional que es una redirección a la ruta hija principal, siempre y cuando la ruta a la que se haya intentado acceder no exista.

Una ventaja de esta librería es que proporciona un mecanismo para analizar los datos antes de acceder a cada ruta mediante la función *router.beforeEach()*. El objetivo de emplear esto es poder controlar si se ha accedido al proyecto desde *nubii*, generando un error en caso contrario.

Anteriormente se mencionó que cada ruta carga una vista determinada, las cuales están localizadas en *src/views*, dentro de la cual, tal y como se aprecia en la ilustración 5.13, se han generado tantas carpetas como aplicaciones tiene el proyecto, y dentro de cada una de estas subcarpetas se ha generado un fichero *.vue* del mismo nombre, que será la vista padre sobre la cual todas las vistas hijas serán cargadas.

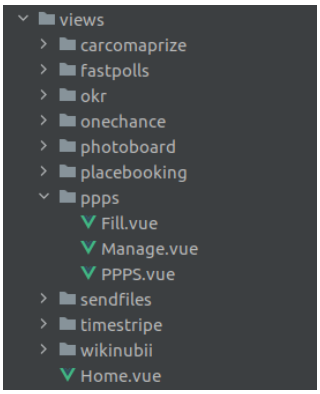


Ilustración 5.13: Estructura de vistas de usuario mediante *Vue Router*

Todas las vistas padres poseen una estructura común, en las que se comprueba primero si se ha recogido correctamente los datos del usuario en la tienda *QueryStore*. Si estos datos están correctos entonces se carga la vista solicitada mediante la etiqueta *router-view*. En la ilustración 5.14 se muestra una situación en la que se ha intentado acceder de manera ilícita o no se han cargado bien los datos del usuario, generando un mensaje de advertencia sin cargar ninguna de las vistas hijas.

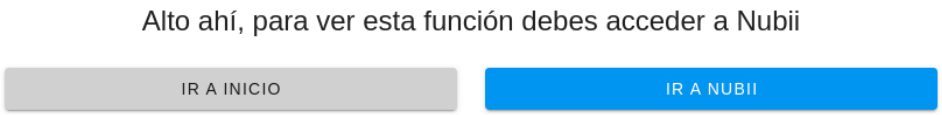


Ilustración 5.14: Error de enrutamiento con *Vue Router*

- **toastr**[5]: Esta librería está diseñada para funcionar con *JavaScript*, y su cometido es proporcionar un mecanismo para generar notificaciones no bloqueantes, configuradas con *HTML5* y *jQuery*. Se ha preferido emplear esta librería respecto a otras debido a su sencillez de uso, configuraciones disponibles y que es puramente *JavaScript*.

  Para generar una notificación se debe invocar la librería con *this.$toastr* seguido del tipo de notificación a mostrar, ya que si se selecciona una *s*, la notificación será de color verde, si se emplea *w*, ésta será de color amarillo, y una *e* para generar una notificación de error de color roja (ilustración 5.15). El parámetro dentro de esta función debe ser una cadena de texto.

Ilustración 5.15: Notificación de error con *Toastr*

- **i18n**[17]: Esta palabra es un modo de referirse a la frase en inglés, *internationalization and localization*, la cual se define como el proceso de preparar y adaptar un proyecto a varios idiomas del mundo consiguiendo, de este modo, que la aplicación sea más accesible para un mayor número de personas.

  Para llevar este objetivo al proyecto, se ha empleado una librería del mismo nombre (*i18n*), la cual, por medio de ficheros *JSON* y una configuración previa para detectar el lenguaje del navegador (algoritmo 5.21), es capaz de seleccionar los textos adecuados según lenguaje local. De momento, el proyecto sólo dispone de dos lenguajes, español (es) e inglés (en). Por lo que si se detecta que el lenguaje del navegador es español, se procede a usar las correspondientes traducciones, en cualquier otro caso se emplean las traducciones en inglés.

```
const locale = navigator.language.substring(0, 2) === 'es' ? 'es' : 'en';
```

Algoritmo 5.21: Detección del idioma del navegador

  Estos ficheros de traducciones reciben el nombre de la abreviatura de su lenguaje, `es` para español y `en` para inglés y todos estos ficheros de traducciones están estructurados en formato *JSON*, donde ambos deben poseer las mismas claves.

  Finalmente, en el algoritmo 5.22 se presenta el modo de invocar a esta librería empleando para ello la función *this.$i18n.t()*, cuyo parámetro es el nombre completo de la etiqueta definido en el fichero, es decir, su ruta desde el primer nivel hasta llegar al nivel de la etiqueta.

```
this.$toastr.s(this.$i18n.t('ppps.fillForm.form.message.delete.ok'));
```

Algoritmo 5.22: Ejemplo de uso del traductor *i18n*

Capítulo 6

Análisis

6.1. Análisis del problema

La plataforma de teletrabajo *nubii* proporciona diversas herramientas para permitir a trabajadores interactuar entre ellos, tales como videollamadas, un chat, estados del personaje, entra otros. Sin embargo, esta plataforma requiere disponer de características propias que la definan, o como se le suele nombrar en la empresa, cultura corporativa. Por ejemplo, con la situación actual de restricciones a raíz de la pandemia, para poder asistir a la oficina física, en la empresa se empleaba una hoja de cálculo en el que los trabajadores tenían que apuntarse; algo parecido sucedía cuando se deseaba desarrollar encuestas simples, pues para hacerlas se empleaba formularios de *Google*[18] o mediante el chat de *Mattermost*[26], lo que conlleva a tener dependencias adicionales de aplicaciones de terceros y puede que algunas de ellas representen un gasto adicional.

Pero este no es el único caso, pues hoy en día, debido a al situación actual, las plataformas de teletrabajo se comienzan a usar en mayor medida y puede que sea algo que conviva con el trabajo presencial. Entonces, estas nuevas plataformas pueden sufrir de este tipo de carencias por lo que tendrían que recurrir a desarrollarlas por sí mismos, contratar aplicaciones por separado o, simplemente, adquirir este servicio, además de todas sus aplicaciones y las que el usuario desee, como un complemento adicional a su plataforma.

6.2. Requisitos y casos de uso

Para poder diseñar y estructurar de buena manera las aplicaciones, se realizó un análisis de requisitos y casos de uso con el tutor de la empresa, además de recoger propuestas de diseño. Por ejemplo, en la ilustración 6.1 se muestra el plan realizado para la aplicación de `fotos de empresa`.

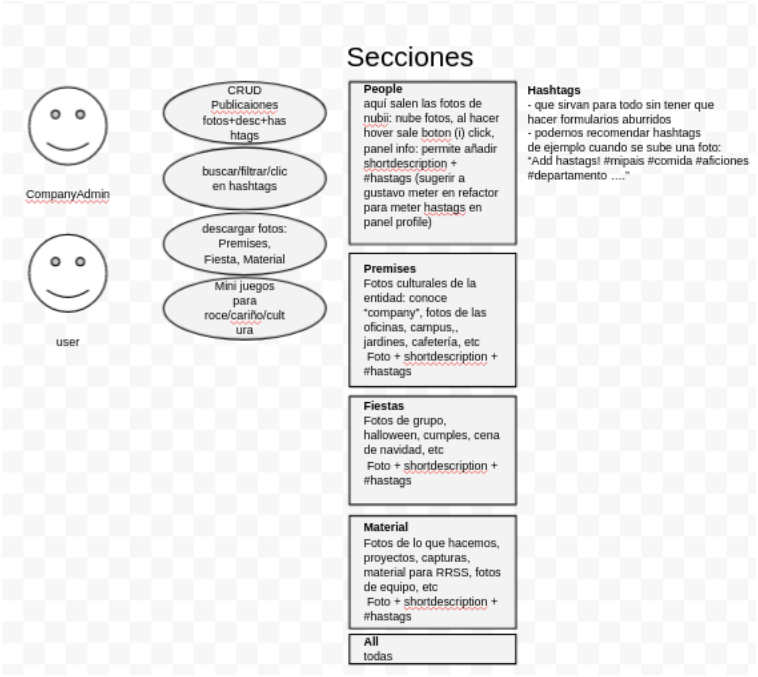
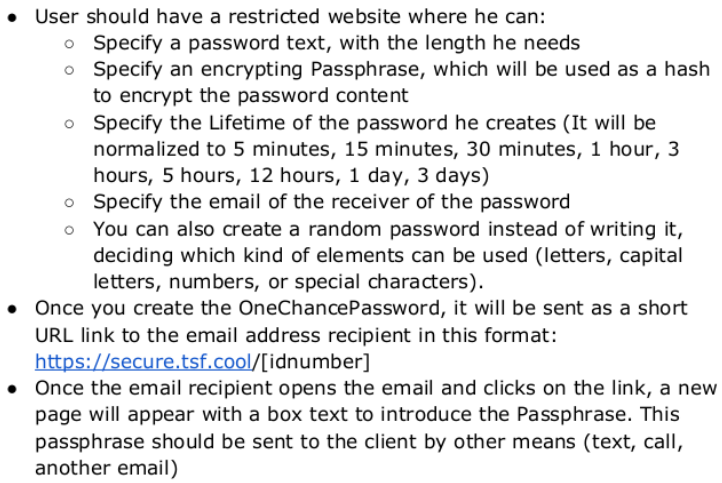


Ilustración 6.1: Casos de uso y estructura de la aplicación *fotos de empresa*

Donde se puede observar que existen dos roles: un administrador y un usuario, además se definen también los casos de uso de los usuarios, en este caso comunes para ambos. Finalmente, también se definió cómo se organizarían estas fotografías según los tipos declarados en las secciones.

Otro ejemplo de este análisis es el caso de la aplicación *One Chance Password*, en la cual, como se aprecia en la ilustración 6.2, se desarrolló un fichero con todos los casos de uso y requisitos del usuario.



- User should have a restricted website where he can:
  - Specify a password text, with the length he needs
  - Specify an encrypting Passphrase, which will be used as a hash to encrypt the password content
  - Specify the Lifetime of the password he creates (It will be normalized to 5 minutes, 15 minutes, 30 minutes, 1 hour, 3 hours, 5 hours, 12 hours, 1 day, 3 days)
  - Specify the email of the receiver of the password
  - You can also create a random password instead of writing it, deciding which kind of elements can be used (letters, capital letters, numbers, or special characters).
- Once you create the OneChancePassword, it will be sent as a short URL link to the email address recipient in this format: https://secure.tsf.cool/[idnumber]
- Once the email recipient opens the email and clicks on the link, a new page will appear with a box text to introduce the Passphrase. This passphrase should be sent to the client by other means (text, call, another email)

Ilustración 6.2: Casos de uso de la aplicación *One Chance Password*

6.3. Plataforma de teletrabajo (*nubii*)

Esta plataforma es propiedad de la empresa *The Singular Factory SL*[43], fundada en 2011 por `Gustavo Medina`, la cual posee varias oficinas y trabajadores repartidos por el mundo como Miami, Bogotá, Ciudad de México, Madrid, islas Canarias, entre algunas más, motivo por el cual gran parte de sus trabajadores y de sus proyectos son desarrollados de manera remota a través de una plataforma de teletrabajo desarrollado por esta misma, denominada nubii[30]. Esta plataforma, cuya interfaz de usuario se corresponde con la ilustración 6.3, permite a compañeros de trabajo poder realizar videollamadas o acceder a un chat para poder comunicarse entre ellos, además de permitir al usuario la capacidad de personalizar un avatar, y controlarlo como si de un videojuego se tratara, todo ello con el objetivo de hacer más atractivo la plataforma de cara al usuario.

Ilustración 6.3: Interfaz de usuario de *nubii*

Puesto que *nubii* es un producto de la empresa, se busca hacerlo más atractivo y añadirle más funcionalidades para atraer un mayor número de usuarios. Es por este motivo que se ha pensado desarrollar un kit de herramientas accesibles dentro de la misma plataforma que otorgue más herramientas al usuario, pues es aquí donde este proyecto encaja. Sin embargo, la idea de desarrollar este proyecto dentro del proyecto de *nubii* era algo improbable pues terminaría ralentizando la plataforma al tener más código del necesario en todo momento. Por este motivo, se desarrolló el proyecto de manera independiente a *nubii*. Pero, a raíz de esta decisión, surgió el problema de cómo heredar la sesión de la plataforma en el proyecto.

La solución encontrada para este problema fue la de transferir los datos más importantes del usuario mediante una *query*. En el algoritmo 6.1 se presenta la lógica que se tuvo que añadir al proyecto de *nubii* para poder adjuntar estos datos en dicho mecanismo.

```
const iWant = ['name', 'email', 'roles', 'id'];
const userData = this.$store.getters['UserStore/getUser'];

iWant.forEach((item) => {
obj.push(`${encodeURIComponent(item)}=${encodeURIComponent(userData[item])}`);
});

obj.push(`${encodeURIComponent('companyID')}=${encodeURIComponent(userData.company.id)}`);
```

Algoritmo 6.1: Lógica para transferir datos desde *nubii*

Finalmente, como se observa en el algoritmo 6.1, los datos transferidos al proyecto son cinco: el nombre del usuario, su correo electrónico, el rol que posee, su identificador y el identificador de la empresa a la que pertenece.

Capítulo 7

Diseño

7.1. Interfaz gráfica de usuario

Para facilitar el desarrollo de la interfaz de usuario del proyecto, además de poder agilizarlo, se ha empleado *Vuetify*[45], que se define como un *framework* de código abierto ideado para *Vue.js*. Esta marco de trabajo facilita el diseño de interfaces de usuario (*UI*) empleando para ello especificaciones de, como su nombre en inglés lo especifica, *Material Design*, el cual es, básicamente, un sistema de diseño de interfaces desarrollado por *Google*.

A diferencia de otras herramientas como *Bootstrap*, *Vuetify* aplica en la totalidad de sus componente un estilo *Material*, por lo que respeta un mismo patrón de diseño. Para poder emplear este *framework* en el proyecto, es necesario cambiar la etiqueta principal *div* presente en el fichero *App.vue*, por una etiqueta propia de este *framework* denominada *v-app*, de este modo se consigue que los estilos aplicados sean generales en todo el proyecto. Además, como se muestra en el algoritmo 7.1, es posible modificar algunas de sus configuraciones mediante su fichero de configuración localizado en *src/plugings/vuetify.js*.

```
    themes: {
      light: {
        primary: '#2196F3',
        secondary: '#424242',
        accent: '#82B1FF',
        error: '#FF5252',
        info: '#2196F3',
        success: '#4CAF50',
        warning: '#FFC107',
      },
    },
  },
  icons: {
    iconfont: 'md',
  },
```

Algoritmo 7.1: Fichero de configuración de *Vuetify*

Una de las ventajas de emplear esta herramientas es que proporciona desde elementos pequeños como una cabecera, un botón o un cuadro de texto hasta elementos más complejos como un tarjeta denominada *v-card* o, inclusive, un calendario *v-calendar*. Todos estos componentes poseen un gran abanico de configuraciones, desde el color de fondo, la elevación del elemento, las reglas de validación para formularios, entre otros.

Por ejemplo, para poder adjuntar una imagen es posible hacerlo mediante la etiqueta *v-file-input*, la cual acepta un *v-model* para recoger el estado del componente, y otro gran cantidad de configuraciones como el poder especificar el tipo de fichero que acepta, o si se desea mostrar el tamaño del fichero, o adjuntar una etiquetas de texto y, lo más importante en este tipo de elementos que recogen datos y como se aprecia en el algoritmo 7.2, es la disponibilidad de añadir determinadas reglas de validación mediante la opción de *rules*.

```
ruleFile: [
    (v) => !!(v) || this.$i18n.t('photoboard.imageCard.empty'),
    (v) => !(v) || v.size < 7000000 || this.$i18n.t('photoboard.imageCard.size'),
  ],
```

Algoritmo 7.2: Validación de formularios con *Vuetify*

En el algoritmo 7.2 se pueden apreciar dos reglas para una entrada de ficheros: la primera comprueba que se haya subido algún fichero, mientras que la segunda comprueba que el fichero no exceda el tamaño fijado. De este mismo modo, se pueden establecer reglas para todo tipo de elementos que recojan datos tales como entradas de texto, selectores, botones, entre otros. Si alguna de las reglas no se cumple, el *v-model* perteneciente al formulario no se valida por lo que no se acepta el envío de éste hasta que el usuario introduzca unos datos válidos.

Ilustración 7.1: Ejemplo de validación de formularios en *Vuetify*

Tal como se observa en la ilustración 7.1, el *framework* colorea de verde la entrada si ésta ha sido correcta (en este caso, no se requiere un texto), o de rojo si ha habido algún fallo, como por ejemplo que no se ha elegido un valor. Mientras el formulario no sea válido, o lo que es lo mismo, mientras las reglas de los distintos componentes no se cumplan, el botón de `enviar` seguirá deshabilitado hasta que se verifiquen todas las reglas.

Si se intentara alterar el código *HTML* desde la sección de inspección del navegador y, por consiguiente, activar el botón de *enviar*, se ha añadido una segunda verificación por *JavaScript* ejecutada justo antes de enviar la petición, evitando de este modo que se envíen datos erróneos al servidor.

A continuación, en la ilustración 7.2, se enseña el uso de *Vuetify* para mostrar imágenes mediante la etiqueta *v-img*, la cual dispone de una configuración para establecer una imagen de espera mientras se obtiene y se carga la imagen original empleando el atributo *lazy-src*, de este modo se consigue amenizar la espera de la carga de la imagen.

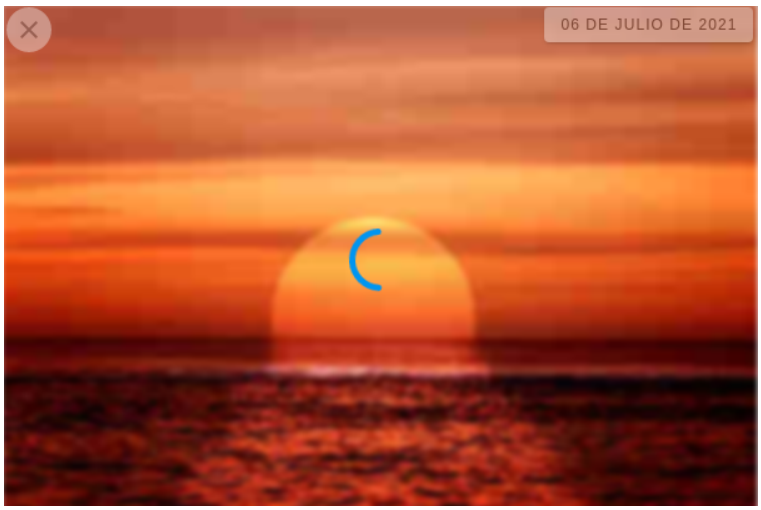


Ilustración 7.2: Imagen de carga mientras se espera la original (*Vuetify*)

Para concluir, destacar que el uso de *Vuetify* en el proyecto ha sido de suma importancia, pues gracias a éste se ha construido una interfaz de usuario minimalista y con un diseño atractivo. Además, este *framework* proporciona herramientas adicionales como la verificación de formularios o la carga de imágenes de espera, con lo que se consigue emplear menos librerías de terceros y agregándole menos dependencias.

7.2. Servidor

Para estructurar y desarrollar el código del servidor se ha empleado el *framework Symfony*[36]. Se ha seleccionado este marco de trabajo debido a que era el utilizado en la empresa debido a que posee una gran reputación, además de proporcionar una estructura basada en el patrón de diseño Modelo Vista Controlador (*MVC*), que le otorga una mayor robustez al proyecto.

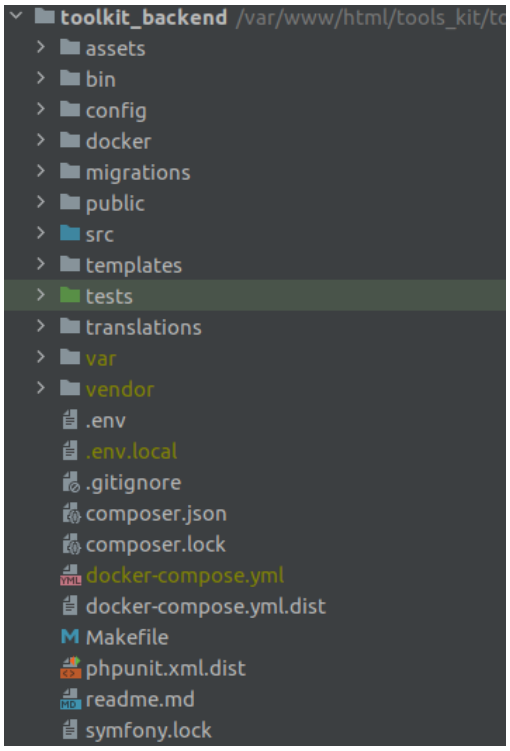


Ilustración 7.3: Estructura de directorios de *Symfony*

Tal como se observa en la ilustración 7.3, este *framework* posee una estructura determinada de directorios y ficheros, de los cuales se procederá a explicar los más relevantes.

- **Directorio *config***: Aquí se almacenan todos los ficheros de configuración del proyecto (ilustración 7.4).

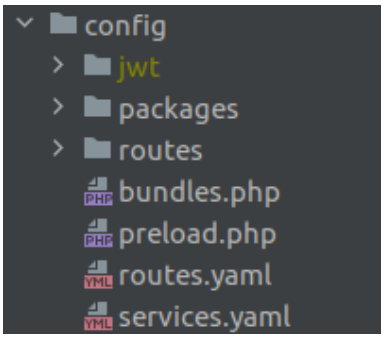


Ilustración 7.4: Ficheros de configuración de *Symfony*

Donde la carpeta *jwt* almacena las claves públicas y privadas del *JWT* usado para temas de autorización en el proyecto, que será explicado más adelante. La carpeta *packages* contiene los ficheros de configuración de todos los paquetes instalados en el proyecto tales como la configuración del software *Api Platform*, la configuración de *JWT* para controlar la seguridad del proyecto, o la configuración del software empleado para enviar correos electrónicos.

Entre todos estos ficheros, uno de los más importante es el llamado *security.yaml*, que se muestra en el algoritmo 7.3 y se encarga de restringir el acceso al proyecto mediante el uso de reglas.

```
    docs:
        pattern: ^/api/v1/docs$
        methods: [ GET ]
        security: false
    api_token:
        pattern: ^/api/login
        methods: [ POST ]
        stateless: true
        anonymous: true
        json_login:
            check_path: /api/login_check
            success_handler: lexik_jwt_authentication.handler.authentication_success
            failure_handler: lexik_jwt_authentication.handler.authentication_failure
    refresh:
        pattern: ^/api/refresh
        stateless: true
```

Algoritmo 7.3: Fragmento de código del fichero *security.yaml*

Como se puede apreciar en el algoritmo 7.3 se han definido una serie de reglas:

- **docs**: Esta regla referencia a la documentación del proyecto, cuya ruta no está protegida para que cualquier desarrollador pueda consultar los distintos *endpoints* disponibles.
- **api_token**: Esta regla está relacionada con la autorización mediante *JWT*, ya que para poder acceder a esta ruta es necesario pasar la autenticación de usuario. Si este proceso es exitoso, entonces se devuelve la pareja de *JWT* de autorización, en caso contrario se genera un mensaje de error informando que no se tiene los permisos correspondientes.
- **refresh**: Esta regla también tiene relación con la autorización mediante *JWT*, pues se trata de un *endpoint* que recibe peticiones para refrescar los datos de autorización mediante el uso de un *refresh_token* válido.

El resto reglas restringen el acceso al resto de *enpoints* si el usuario no se ha autenticado mediante las credenciales correspondientes. Para finalizar este apartado, en el algoritmo 7.4 se muestran unas reglas para el control de acceso mediante roles de la aplicación, denegando el acceso a determinados *endpoints* según el rol del usuario.

```
    access_control:
        - { path: ^/admin, roles: ROLE_ADMIN }
        - { path: ^/api/login_check, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/api/refresh, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/api/v1/messages, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/api/v1/docs, roles: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/api/v1, roles: IS_AUTHENTICATED_FULLY }
```

Algoritmo 7.4: Autorización por roles a la aplicación

Dentro del directorio *config* también se define la configuración del mapeo de clases y configuraciones adicionales de *Api Platform*, como por ejemplo, el de añadir un sufijo a todos los *endpoints* del proyecto, en este caso se añadió el sufijo *v1*.

En los ficheros restantes (*routes.yaml*) se definen las rutas existentes en el proyecto, las cuales se corresponden con aquellas empleadas para obtener la pareja de *JWT* de autorización o para refrescarla. Y, finalmente, en el fichero *services.yaml* se definen todas las inyecciones del proyecto, tanto si se trata de variables como de los controladores.

- **Directorio *docker***: Este directorio alberga toda la información de los distintos contenedores del *back-end*, estructurados de la siguiente manera:

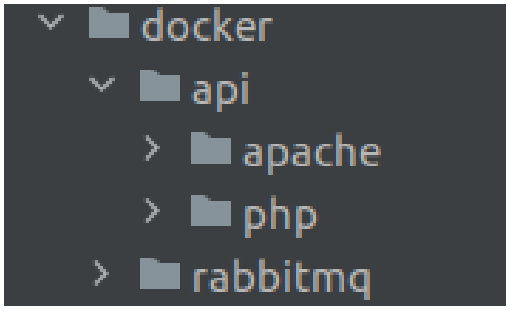


Ilustración 7.5: Estructura del directorio *docker*

De la ilustración 7.5 se concluye que la carpeta *apache* contiene el fichero de configuración *Dockerfile* y otros ficheros adicionales para montar el servidor web, la carpeta *php* contiene también su correspondiente *Dockerfile* acompañado de otros ficheros de configuración como es el fichero *php.ini*, y, finalmente, la carpeta *rabbitmq* posee la misma estructura que la carpeta *php*.

- **Directorio *migrations***: Aquí se almacenan todos los ficheros *PHP* generados con cada migración de la base de datos realizada, donde cada fichero posee un nombre con la fecha de dicha migración.
- **Directorio src**: Este directorio es, probablemente, el más importante del proyecto, pues contiene toda su lógica y su estructura se corresponde con la presentada en la ilustración 7.6, agrupando las clases por categorías y funcionalidades.

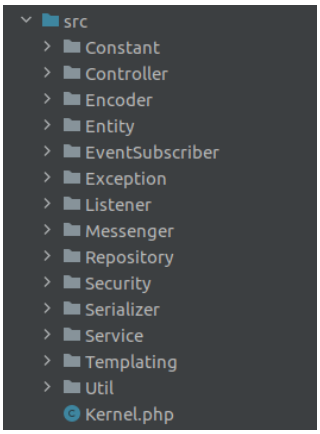


Ilustración 7.6: Estructura del directorio *src*

- **Constant**: Almacena todas las variables inmutables del proyecto separadas por entidad o funcionalidad.
- **Controller**: Esta carpeta se estructura en otros subdirectorios cuyo nombre se corresponde con el nombre de la entidad para la que funcionan. Es en este directorio donde se almacenan todos aquellos controladores que no puedan ser ejecutados por *Api Platform*, ya que se debe realizar un tratamiento adicional. Un ejemplo de ello es el caso de la aplicación *Fotos de empresa*, donde cada imagen que es añadida por un usuario debe ser subida al servicio en la nube de *Amazon Web Services*, razón por la cual esta tarea debe ser realizada manualmente. Finalmente, cada controlador puede estar conectado a un servicio de la carpeta *src/Service*, puesto que los controladores sólo se encargan de recibir la petición, procesar que ésta sea adecuada para, posteriormente, delegar la acción al servicio correspondiente.
- **Encoder**: Esta carpeta contiene una clase que se encarga de codificar el mensaje de la aplicación *One Chance Password*.
- **Entity**: Esta carpeta contiene todas las entidades del proyecto, es decir, todos aquellos objetos presentes como tablas en la base de datos (ilustración 7.7).

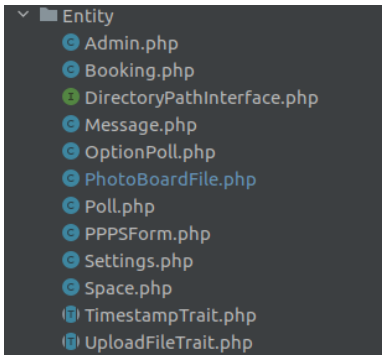


Ilustración 7.7: Estructura del directorio *src/Entity*

En la imagen anterior se aprecian todas las entidades existentes en el proyecto, además de tres clases auxiliares que complementan estas entidades. Una de ellas es una interfaz empleada para el uso de directorios en la aplicación de *Fotografías de Empresa*, mientras que las otras dos funcionan como *Traits*, es decir, son una serie de propiedades y métodos comunes que pueden ser utilizadas en más de una entidad.

- **EventSubscriber**: En este directorio se añaden clases que funcionan como eventos de entidades, es decir, eventos que son ejecutados tanto antes de validar como antes y después de escribir en la entidad. Esto se ha empleado también para la aplicación *Fotos de Empresa*, pues era necesario procesar la imagen y subirla a la nube de *Amazon* antes de podes persistir el objeto en la base de datos.
- **Exception y Listener**: La primera reúne clases específicas que se ejecutan cuando sucede alguna excepción en el proyecto, mientras que la segunda se encarga de escuchar estas excepciones y procesarlas según se requiera.
- **Messenger**: Esta carpeta es empleada para generar un nuevo objeto que represente un mensaje en la aplicación de *One Chance Password*.
- **Repository**: En este directorio se encuentran todos los repositorios de las entidades, identificados por el nombre de la entidad a la que pertenecen.
- **Security**: Aquí se encuentra la lógica para conceder o denegar acceso al listado de mensajes de la aplicación *One Chance Password*.
- **Serializer**: En esta carpeta se estructuran las respuestas que se devuelven al cliente, organizándolas como sea conveniente.
- **Templating**: Aquí se almacenan variables constantes que se relacionan con platillas de *Twig*.
- **Util**: En este directorio se almacenan todas aquellas funciones que puedan ser empleadas por varias clases, evitando de este modo tener que generar código repetido.

Finalmente, a parte de todos estos directorios, también existen algunos ficheros de configuración relevantes, entre ellos está presente los ficheros *composer* que se encargan de la gestión de paquetes del proyecto y el fichero *docker-compose.yml* el cual se encarga de montar todos los contenedores empleando los ficheros de configuración localizados en la carpeta *docker*.

7.3. Cliente

Para facilitar el desarrollo de la parte del cliente del proyecto se ha empleado *Vue.js*[44], el cual se define como un *framework* progresivo *JavaScript* de código abierto especializado en aplicaciones de tipo *single-page*, es decir, páginas webs que sobreescriben dinámicamente una

única página en lugar de cargarla desde cero, consiguiendo así una mayor fluidez y sencillez de cara al usuario.

Se ha empleado este marco de trabajo, en principio, por ser parte del equipamiento o *stack* usado por la empresa, sin embargo, también se ha tomado en cuenta el gran nivel de desacoplamiento que posee y la capacidad de juntar todos los componentes deseados en uno y conseguir que estos sean capaces de trabajar juntos.

El elemento básico de este *framework* es el fichero *.vue*, cuya estructura está compuesta por tres partes, una denominada *template*, otra *script* y una última denominada *style*:

- **template**: Esta parte se corresponde con la plantilla *HTML*, representada en el algoritmo 7.5, la cual siempre está compuesta por una etiqueta que engloba todo el contenido de éste, por lo general suele ser una etiqueta de tipo *div*. La ventaja que proporciona este *Vue.js* es que es posible añadir una variable mediante *v-model* a distintos elementos *HTML*, consiguiendo que éstos respondan y modifiquen esta variable, la cual puede ser procesada en el código *JavaScript* (*script*) del componente.

  Pero, además de esta característica, los distintos elementos poseen una gran variedad de configuraciones como *v-text*, *v-if*, *v-for*, entre otros propios de *Vue.js*.

  Y, finalmente, *Vue.js* también proporciona eventos para cada elemento haciendo que sea más fácil la detección de cambios en él, entre los más utilizados en el proyecto se encuentra *@click*, el cual se encarga de detectar cuando el usuario ha realizado un clic sobre dicho elemento pudiendo invocar un método cada vez que este evento sea recogido.

```
<template>
  <div>
    <v-dialog
      v-model="deleteDialog"
      max-width="600"
    >
      <v-card rounded="xl" class="pa-2">
        <v-card-title class="headline justify-center">
            {{ $i18n.t('bookings.forms.delete') }}</v-card-title>
        <v-card-actions>
          <v-btn color="info" @click="deleteForm" width="150">
            {{ $i18n.t('bookings.forms.ok') }}
          </v-btn>
        </v-card-actions>
      </v-card>
    </v-dialog>
  </div>
</template>
```

Algoritmo 7.5: Sección *template* de un fichero *Vue*

- **script**: Esta sección de un fichero *Vue* recoge toda la lógica y datos del componente, cuya estructura podría ser la presentada en el algoritmo 7.6, la cual está compuesta por determinadas secciones, algunas de las cuales se procede a explicar.

```
<script>
    export default {
      name: 'DeleteItem',
      methods: {
        async deleteForm() {
          await this.$store.dispatch(`${this.store}`,
            {
              query: this.id,
              toastr: this.$toastr,
            });
          this.close();
        },
        close() {
          this.$emit('close-dialog');
        },
      },
      props: {
        deleteDialog: Boolean,
        id: Number,
        store: String,
      },
    };
</script>
```

Algoritmo 7.6: Sección *script* de un fichero *Vue*

- **Nombre (*name*)**: Empleado para referenciar el componente desde otra sección del proyecto.
- **Datos (*data*)**: Aquí se recogen todas las variables del componente almacenados en un objeto.
- **Métodos (*methods*)**: En esta sección se definen todas las funciones del componente.
- **Datos heredados (*props*)**: Esta sección es importante para poder compartir información de un elemento padre a un elemento hijo.

Además, todo componente *Vue* tiene un ciclo de vida con distintas etapas que se corresponden la las presentadas en la ilustración 7.8, las cuales pueden ser invocadas mediante determinados métodos.

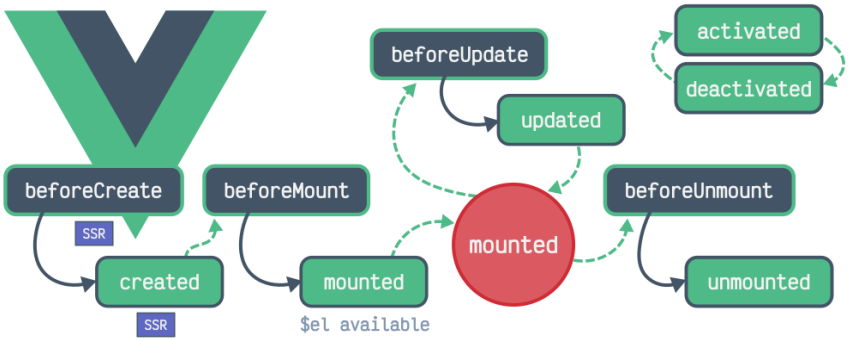


Ilustración 7.8: Ciclo de vida en *Vue.js* [extraído de *lenguajejs.com*[22]]

Tal como se aprecia en la ilustración 7.8, todo componente *Vue* pasa por una serie de etapas durante toda su existencia, en los que se diferencian dos elementos del componente, por una lado se encuentra el objeto que representa el componente, y por el otro los datos de éste.
Entonces, una vez se ha creado el componente (*created*), los datos del componente ya son accesibles, pero el objeto aún no.
Cuando el componente alcanza la etapa de montaje (*mounted*), es entonces cuando tanto los datos como el objeto ya son accesibles, es decir, el componente está listo para su uso. Y, mientras permanezca en este nivel, se generará una etapa cíclica entre *mounted* y *updated*, pues al realizar cualquier cambio sobre el componente, éste se ve actualizado y posteriormente montado.
Finalmente, el componente es eliminado, pudiendo acceder a esta etapa mediante el método *destroyed*, después del cual, el componente dejará de existir y liberará su memoria.

- **style**: En esta sección, la cual se muestra en el algoritmo 7.7, se especifican todos los estilos del fichero *Vue*, tanto si es una clase, un identificador, algún evento, o lo que el desarrollador vea conveniente.

```
<style scoped>
    .hover__button {
      background-color: whitesmoke;
      opacity: 40%;
    }
    .hover__button:hover {
      transition: 0.3s;
      opacity: 100%;
    }
    .position__button {
      position: fixed;
      z-index: 10;
      margin-top: 3px;
    }
</style>
```

Algoritmo 7.7: Sección *style* de un fichero *Vue*

Continuando con la explicación de este *framework*, se proseguirá a especificar la estructura adoptada, pues un proyecto realizado con *Vue.js* sigue el patrón de directorios y ficheros mostrado en la ilustración 7.9.

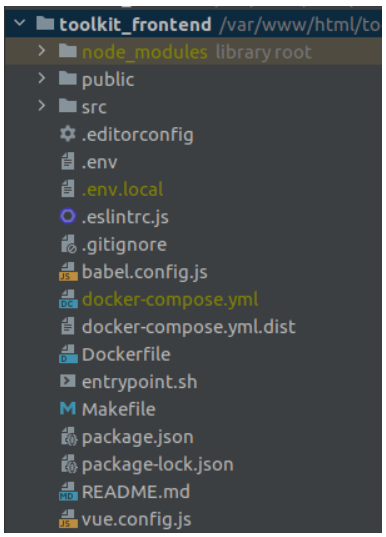


Ilustración 7.9: Estructura de un proyecto *Vue.js*

Tal como se aprecia en la imagen anterior, la estructura del *fronte-end* se compone en tres carpetas y una serie de ficheros, de los cuales se procederá a explicar los más relevantes:

- **Módulos (*node_modules*)**: Esta carpeta alberga los datos de todas las librerías instaladas en el proyecto, por lo que es generada automáticamente al ejecutar por primera vez el proyecto, y se van añadiendo o quitando los paquetes según las necesidades del desarrollador.
- **Inicio (*public*)**: Este directorio almacena el documento *HTML* principal denominado *index.html*, y en cuyo cuerpo (*body*) se sobreescribirán todas las vistas existentes.
- **Recursos (*src*)**: Al igual que en el servidor, esta carpeta contiene toda la lógica del cliente, estructurado en una serie de carpetas (ilustración 7.10).

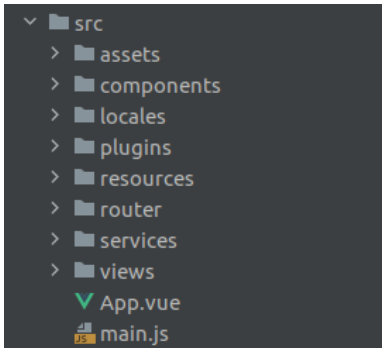


Ilustración 7.10: Estructura de la lógica del cliente

Dentro de este directorio se encuentran dos ficheros, el primero es una fichero *JavaScript* denominado *main.js*, el cual se encarga de inicializar todas librerías del proyecto,

además de manipular los *JWT* de autorización y arrancar el proyecto realizando un montaje sobre el componente *App.vue* (el segundo fichero de esta carpeta), cuyo objetivo es servir de plantilla base para cargar todas las vistas del proyecto.

Ahora se procederá a explicar la existencia de cada una de las carpetas de *src*:

- **assets**: Contiene los iconos y multimedia que se vaya a emplear en la interfaz del proyecto.
- **components**: Esta carpeta contiene todos los módulos de código que serán añadidos a las vistas. Por lo tanto, se ha estructurado en otras subcarpetas en la que cada una se corresponde con una de las aplicaciones del proyecto. Además, existen algunos componentes comunes y reutilizables para todos como por ejemplo el componente de carga *Loading* u otro de confirmación para eliminar un ítem.
- **locales**: Esta carpeta contiene los ficheros de traducción del proyectos, definidos con la librería *i18n*, que será explicada más adelante.
- **plugins**: En este directorio se han definido las configuraciones del *framework* usado para los estilos del proyecto (*Vuetify*).
- **resources**: Esta carpeta contiene todos aquellos objetos y funciones comunes o que puedan ser reutilizables, en este caso denominados *Mixins*.
- **router**: En esta carpeta se ha especificado la configuración del enrutador del proyecto, además de la respectiva importación de todas las vistas de éste.
- **services**: Esta carpeta ha sido empleada para definir lo relacionado a la tienda del proyecto implementada con *Vuex*.
- **views**: Aquí se han definido todas las vistas del proyecto que serán cargadas mediante el enrutador.

- **Ficheros**: Todos y cada uno de estos ficheros tienen una funcionalidad específica, así que se explicarán los más relevantes:
  - **docker-compose y Docker**: Ambos son empleados por *Docker*, para poder activar el contenedor.
  - **eslintrc.js**: Este fichero alberga la configuración del analizador léxico.
  - **env**: Representan a los ficheros de entorno.
  - **editorconfig**: Este fichero establece unas reglas léxicas y sintácticas para el *IDE*.
  - **babel.config.js**: Es un fichero de configuración para el software que se encarga de compilar el código para convertirlo en un estándar.
  - **package.json**: En este fichero se recogen las configuraciones y librerías del proyecto (algoritmo 7.8).

```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build",
  "lint": "vue-cli-service lint"
},
"dependencies": {
  "@babel/polyfill": "^7.4.4",
  "@ssthouse/vue-tree-chart": "^0.5.3",
```

Algoritmo 7.8: Fragmento de configuración del fichero *package.json*

En esta imagen se aprecia la definición de *scripts*, los cuales son empleados dentro del contenedor *Docker* para ejecutar el proyecto, empleando para ello la librería *@vue-cli*, cuyo propósito es facilitar la creación y manipulación de proyectos creados con *Vue.js*.

Finalmente, la ventaja de emplear *vue-cli* es que automatiza la configuración de *webpack*, *babel* y todos los *plugins* y configuraciones necesarias para que pueda funcionar adecuadamente el proyecto. *Webpack* y *babel* son requeridos para poder compilar y transcribir el código de *Vue* a un lenguaje estándar y entendible por los navegadores.

Capítulo 8

Desarrollo

Este proyecto se denomina *Kit de herramientas corporativas para la productividad*, cuyo nombre de producto es, simplemente, *Toolkit* o *Kit de herramientas*. Este producto ha sido desarrollado para ser integrado dentro de la plataforma de teletrabajo *nubii*. Pues este proyecto está ideado para disponer de una gran cantidad y variedad de aplicaciones, conforme sean solicitadas por las distintas empresas que emplean *nubii*. En la ilustración 8.1 se muestran las cinco aplicaciones desarrolladas en esta primera ronda.

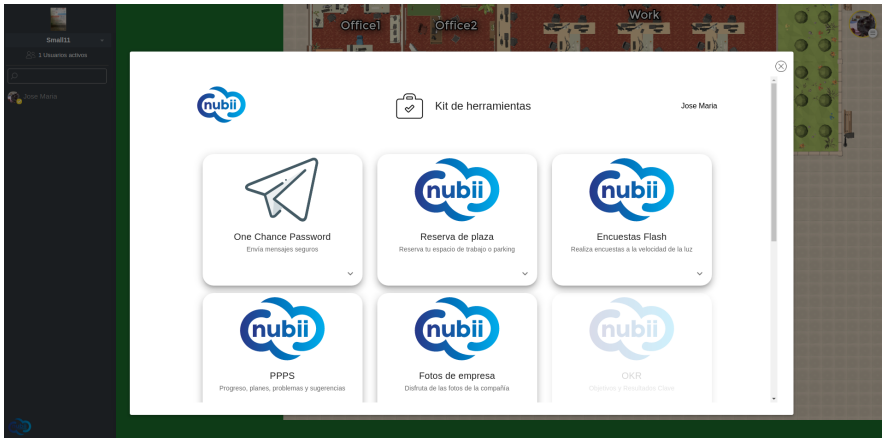


Ilustración 8.1: Aplicaciones desarrolladas para el proyecto

- **One Chance Password**: Esta aplicación permite enviar contraseñas cifradas seguras, que disponen de un tiempo de expiración, a un remitente mediante un correo electrónico.
- **Reserva de plazas**: Su objetivo es posibilitar la reserva de espacios, ya sean de trabajo o aparcamientos, físicos, ayudando a controlar el aforo máximo de cada empresa.
- **Encuestas flash**: Aplicación en la que se pueden crear encuestas personalizadas de una sola pregunta con la finalidad de recoger el porcentaje de votos para cada opción.
- **PPPS**: Formularios semanales para controlar el desarrollo de cada trabajador realizándole cuatro preguntas sobre su actual y futura semana laboral.

- **Fotos de empresa**: Esta aplicación permite subir un ilimitado número de fotografías para cada compañía, mostrándolas en un mural y permitiendo descargarlas y votar por cada una.

El resto de aplicaciones como por ejemplo el envío de ficheros de gran dimensiones, una aplicación para gratificar las ideas u otra para implantar la metodología de trabajo *OKR* (*Objectives and Key Results*) serán desarrolladas en una siguiente etapa, a parte de todas aquellas aplicaciones que sean solicitadas por los usuarios de *nubii*.

Antes de empezar a explicar cada una de estas aplicaciones, es recomendable visualizar la estructura de la base de datos de la aplicación la cual se corresponde con la ilustración 8.2.

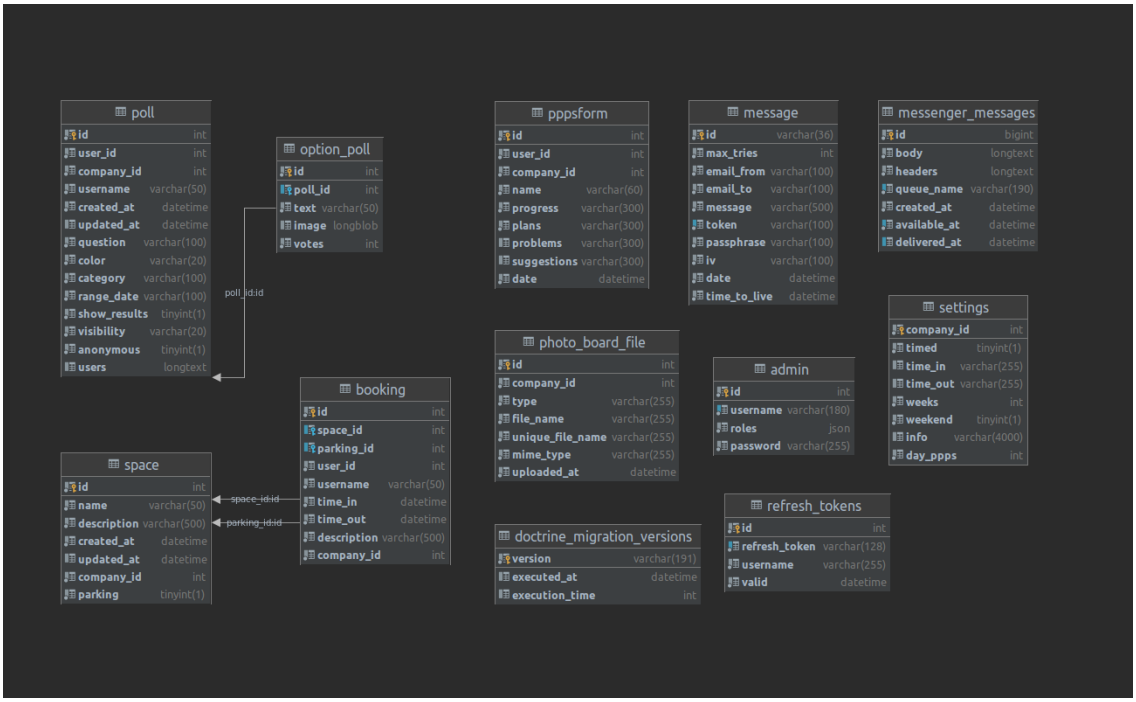


Ilustración 8.2: Estructura de la base de datos del proyecto

Tal como se aprecia en la ilustración 8.2, se han creado doce tablas para el proyecto:

- **Encuestas flash**: Posee dos tablas, *poll* y *option_poll*, enlazadas por una relación de uno a muchos.
- **Reservas de plaza**: Posee dos tablas, *space* y *booking*, ambas enlazadas por dos relaciones de uno a muchos.
- **One Chance Password**: Posee dos tablas, *message* y *messenger_messages*, empleadas para los mensajes generados y enviados.
- **Fotos de empresa y PPPS**: Cada una posee una única tabla de datos, *photo_board_file* y *pppsform*.
- **Globales**: Las cuatros tablas restantes son empleadas para configuración del proyecto.

8.1. One Chance Password

El objetivo de esta aplicación es facilitar la compartición de contraseñas o textos sensibles entre compañeros de trabajo, proporcionando al remitente la opción de establecer un tiempo de vida para el mensaje y una frase de desbloqueo generada aleatoriamente o proporcionada por este mismo, cuya interfaz de usuario se corresponde con la ilustración 8.3.

Ilustración 8.3: Interfaz de usuario de *One Chance Password*

En este formulario de envío se requiere proporcionar el nombre del remitente, el correo electrónico del destinatario, el texto a enviar y el tiempo de expiración del mensaje, la frase de contraseña no es requerida pues se puede generar aleatoriamente. Estos cuatro campos están adecuadamente validados para que el objeto a guardar en la base de datos sea correcto.

El ciclo de ejecución de esta aplicación comienza cuando el usuario envía el formulario con los datos del mensaje. En el servidor, este objeto será desestructurado (*denormalization*), añadiendo cada campo del objeto como un campo de la tabla creada para esta aplicación. Entre los cuales cabe destacar:

- **Tiempo de expiración**: Este campo del objeto almacena el tiempo de vida del mensaje en segundos, por lo que es obtenido sumando este tiempo a la hora de creación del mensaje.
- **Frase de contraseña**: Este campo puede ser nulo o vacío generando una frase aleatoria o una frase proporcionada por el remitente, y es empleada para codificar el mensaje. Se ha añadido una restricción, pues una frase de contraseña no puede contener únicamente espacios.
- **Email destinatario**: Este valor es comprobado con una función específica de *PHP* para validación de correos electrónicos, generando una excepción si no es válido.

- **Mensaje**: Este campo es el que recibe más tratamiento, pues primero se ha de comprobar si cumple con las restricciones de longitud para que, posteriormente, pueda ser codificado usando un algoritmo de cifrado disponible en el listado de la *OpenSSL*, en este caso se emplea por defecto el algoritmo de cifrado `AES-256-CBC` (*Advanced Encryption Standard*).
- **Máximo de intentos**: Se establece a 3 el número máximo de intentos para desbloquear un mensaje, si se sobrepasa dicho umbral el mensaje será eliminado totalmente.

En el algoritmo 8.1 se presenta un mecanismo para evitar almacenar mensajes cuya hora de expiración haya vencido sin que éstos fuesen leídos, para lo cual se ha creado un evento (*Scheduler*) en la base de datos que se ejecuta cada 10 segundos y elimina de ésta todos aquellos mensajes cuya hora de expiración haya sobrepasado la hora actual.

```
$this->addSql('CREATE EVENT removeMessage
    ON SCHEDULE EVERY 10 SECOND
    DO
      DELETE FROM message
      WHERE `time_to_live` < NOW()');
```

Algoritmo 8.1: Definición del *Scheduler*

Una vez realizado el proceso de persistencia del mensaje en la base de datos, se envía este objeto a la cola de mensajería de *RabbitMQ* para que, después, el consumidor que está siendo ejecutado en un contenedor de *Docker* capture dicho mensaje y lo procese, obteniendo los datos del mensaje, cargando la plantilla específica y generando, al final, un nuevo correo electrónico, cuyo estilo inicial se corresponde con la ilustración 8.4.

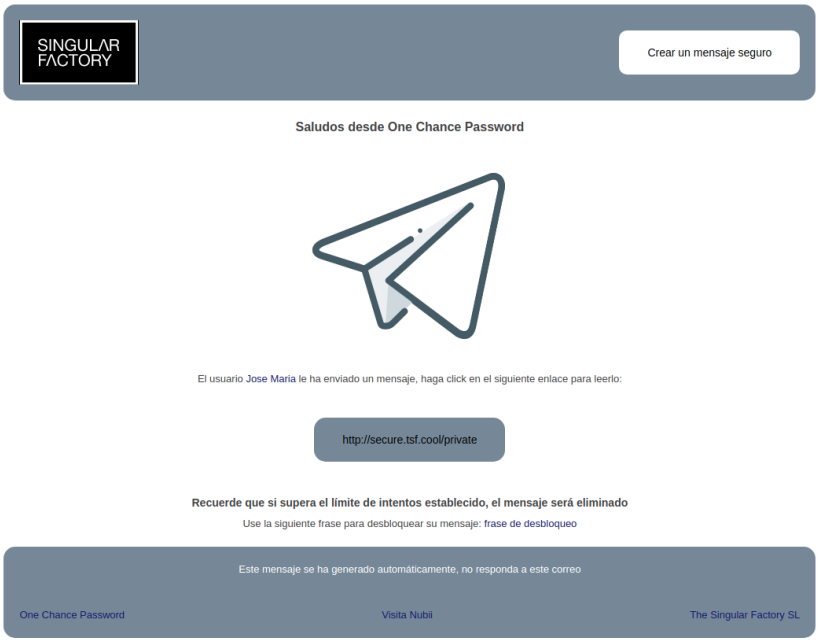


Ilustración 8.4: Correo electrónico enviado por *One Chance Password*

Este correo electrónico posee varios datos como es el nombre del usuario que envío dicho mensaje, una *URL* ficticia para poder ver el mensaje y, finalmente, la frase de contraseña empleada para desbloquear el mensaje. Es importante destacar que este modelo de correo electrónico es temporal, puesto que no es seguro enviar la frase de desbloqueo junto al correo del mensaje, por lo que se ha pensado, en un futuro, enviar dicha frase de desbloqueo por un medio diferente como un mensaje de texto móvil o mediante el mismo chat que existe dentro de *nubii*.

Siguiendo con el flujo de ejecución, cuando el usuario presione sobre el enlace del correo, éste le redirigirá a una vista del proyecto en el que se le preguntará por la frase de desbloqueo enviada, tal y como se presenta en la ilustración 8.5, además se le informará del número restantes de intentos para desbloquear el mensaje.

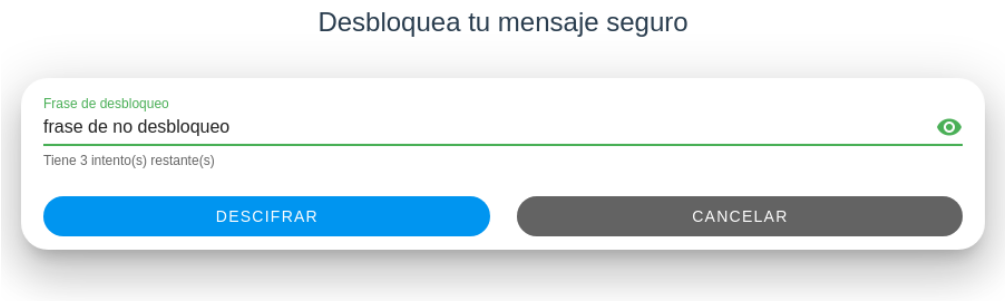


Ilustración 8.5: Vista de desbloqueo del mensaje en *One Chance Password*

Si la frase de contraseña es incorrecta el número de intentos irá disminuyendo hasta que éste llegue a cero, entonces se mostrará una notificación de error, se eliminará el mensaje de la base de datos y se le redirigirá a la vista de inicio, quedando totalmente inservible la *URL* del correo electrónico. Por otro lado, si la frase de contraseña es correcta, entonces se generará una nueva vista donde podrá leer el mensaje y se le mostrará un recordatorio informándole que si recarga la página el mensaje no volverá a ser accesible.

Finalmente, si el usuario vuelve a intentar acceder al mismo enlace habiendo leído correctamente el mensaje, este le redirigirá a la página de inicio informándole que dicho mensaje ya no existe en la base de datos.

8.2. Reservas de plaza

Esta aplicación fue desarrollada con el objetivo de permitir controlar el acceso a un espacio físico así como el aforo máximo, algo de suma importancia hoy en día debido a la pandemia en la que el mundo se encuentra. Un usuario podrá seleccionar el día deseado, y permitido, de la semana para reservar un espacio de trabajo o un aparcamiento de la oficina física, esta última opción fue añadida por solicitud de una compañía a la que le gustaría poder gestionar las plazas de aparcamiento disponibles en su oficina, tal como si se tratara de un espacio de trabajo.

Esta aplicación posee dos roles, uno de administrador y otro de usuario normal, ambos comparten las mismas características, sin embargo, el administrador dispone de una panel adicional donde puede gestionar los espacios existentes, así como las reservas realizadas por los usuarios.

En la ilustración 8.6 se presenta la interfaz de usuario que se encontrará tanto un usuario como un administrador cuando accedan a esta aplicación.

Ilustración 8.6: Interfaz de usuario de *Reservas de plaza*

En esta interfaz se presenta un calendario mensual por defecto con una cabecera para seleccionar el día de hoy, alternar entre meses o cambiar el tipo de vista (mensual, semanal o diario). Los días de la semana estarán remarcados por un código de colores, en el que el color gris significa que el día está deshabilitado para reservas además de mostrar la reserva de otro usuario, el color azul significa que el día contiene una reserva hecha por el usuario actual, el color rojo aparece cuando ya no se admiten más reservas debido a que ya no quedan plazas disponibles y el color verde representa todos aquellos días en los que se puede realizar una reserva. El botón flotante de la parte izquierda inferior se emplea para mostrar las normativas de la empresa respecto a las reservas mediante un cuadro de diálogo.

Todo usuario tiene la posibilidad de cambiar el tipo de vista a semanal o diaria, según las necesidades, además de consultar las reservas de otros usuarios. Una reserva sólo es posible realizarla si el usuario se encuentra en modo diario, en el cual se le habilitará el botón de reserva superior, siempre y cuando no tenga un reserva ya hecha dicho día, en tal caso, solamente podrá editar o eliminar su reserva. El diseño de este formulario se corresponde con la ilustración 8.7.

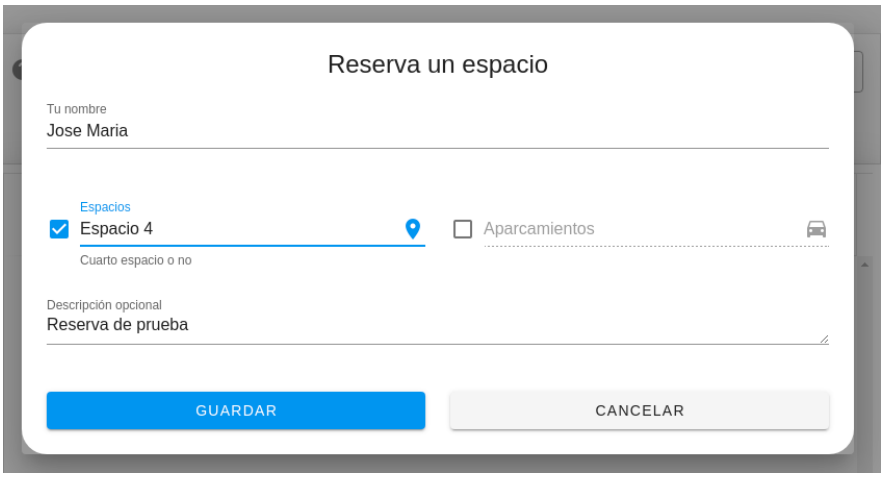


Ilustración 8.7: Formulario de reservas en *Reservas de plaza*

En el formulario anterior es obligatorio especificar un nombre de usuario (se asigna por defecto el nombre de *nubii*), y seleccionar, al menos, un espacio de trabajo, un aparcamiento, o ambos; la descripción de la reserva es opcional. Una vez se realiza una reserva exitosamente, se mostrará una notificación de color verde informando que se ha reservado correctamente o de color rojo en caso contrario. Si un usuario intenta realizar una reserva un día en el que ya existen otras reservas, sólo se le mostrará aquellos espacios que no hayan sido reservados aún.

Y, aparte del rol de usuario, está disponible el rol de administrador, el cual, además de visualizar el calendario también tiene disponible un botón para gestionar los espacios y reservas, cuyo diseño se corresponde con la ilustración 8.8.

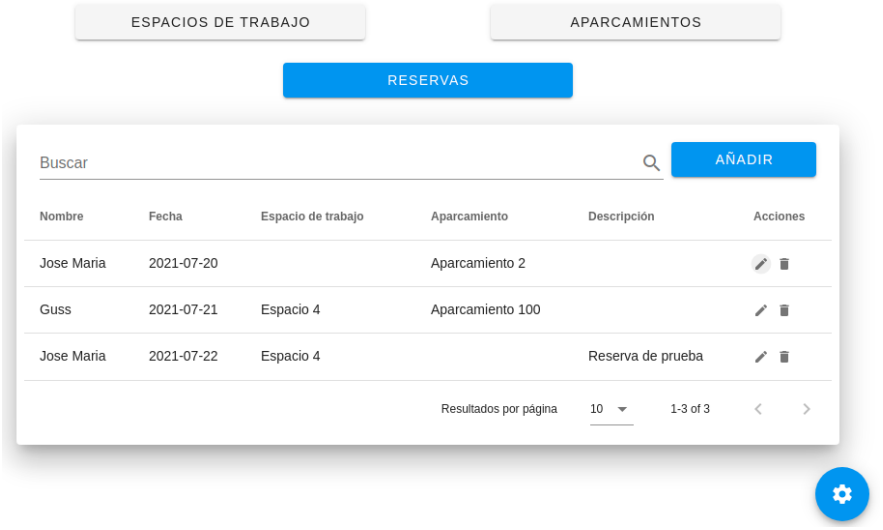


Ilustración 8.8: Vista de administrador en *Reservas de plaza*

Aquí se presentan tres apartados, dos correspondientes a los espacios, ya sea uno de trabajo o un aparcamiento, y un tercero relacionado a las reservas. Un administrador podrá

crear, editar, visualizar y eliminar (CRUD) espacios y reservas para todos los usuarios de su compañía. Además, podrá filtrar por los campos de cada objeto o realizar una búsqueda por valor. Por este motivo, es el administrador el que define el máximo de aforo de la empresa, pues este valor viene determinado por el número de espacios creados. La ilustración 8.9 representa el diseño de estos formularios.

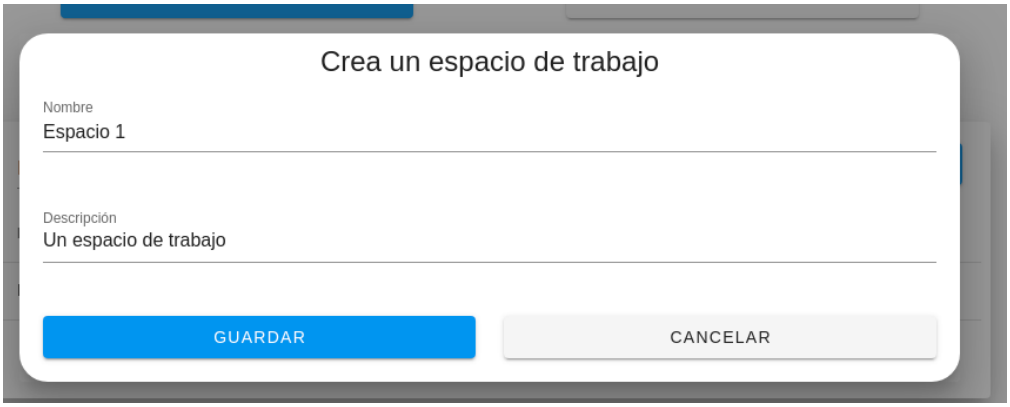


Ilustración 8.9: Formulario para crear espacios en *Reservas de plaza*

Tal como se aprecia, para crear un espacio sólo es suficiente proporcionar un nombre y una descripción. Como una mejora se tiene pensado añadir una opción para adjuntar la imagen del espacio.

Finalmente, la ilustración 8.10 muestra los ajustes que puede editar un administrador.

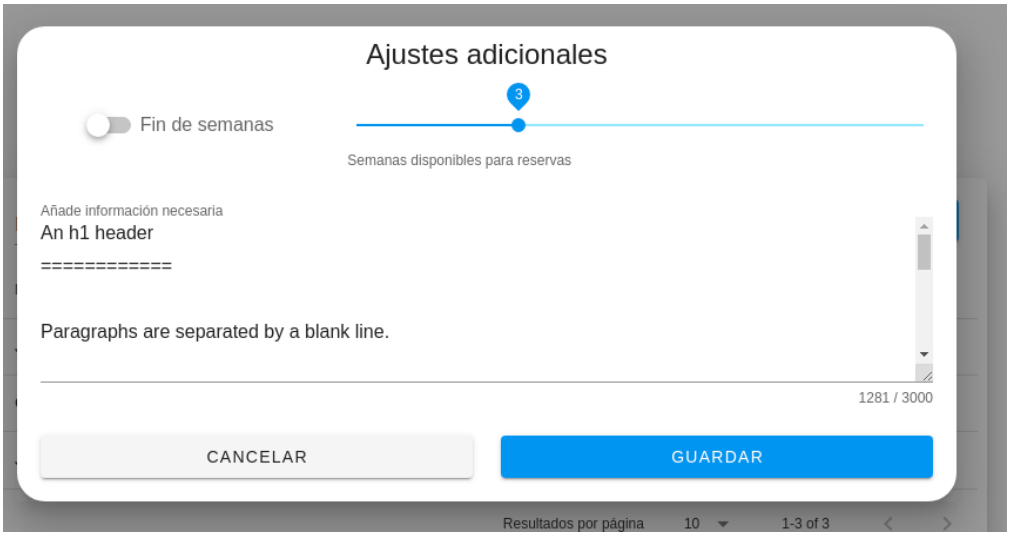


Ilustración 8.10: Formulario de configuración en *Reservas de plaza*

- **Fin de semanas**: Este ajuste permite seleccionar si se habilita o no las reservas de espacios los fines de semana.
- **Semanas disponibles**: Este deslizador representa el número de semanas disponibles para realizar una reserva, contando desde la semana actual hasta el domingo de la

última semana elegida, por lo que, al inicio de cada semana se actualiza el calendario permitiendo siempre reservar el número de semanas establecido.

- **Normativa**: En este cuadro de texto se tiene pensado que los administradores redacten las normativas de su oficina para realizar una reserva. Algo que destacar es que este editor admite instrucciones de estilo mediante el lenguaje de marcado *Markdown*.

8.3. Encuestas flash

Esta aplicación llamada `Fast polls` o `Encuestas flash` permite a los usuarios de una compañía desarrollar encuestas de una sola pregunta y múltiples opciones con el objetivo de recolectar el porcentaje de votos para cada una de las opciones disponibles. Además, cada opción de la encuesta acepta la inclusión de una imagen que represente la opción escrita. La ilustración 8.11 representa la vista principal de esta aplicación en la que se incluye un listado de encuestas, un botón para crear y una barra de configuración.

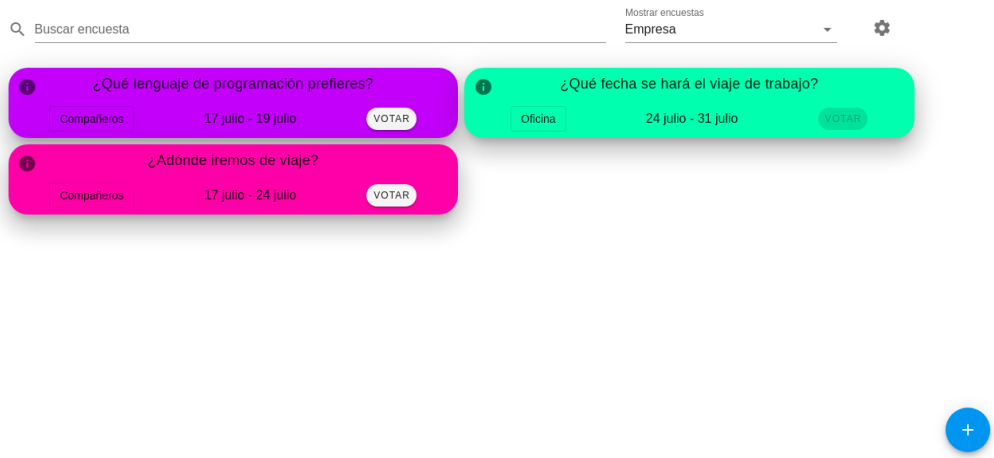


Ilustración 8.11: Interfaz de usuario de la aplicación *Encuestas flash*

La barra de configuración permite al usuario buscar una encuesta por el nombre de ésta, cambiar el modo de visibilidad de las encuestas o filtrar por categorías, usuario y/o fecha de creación.
Cambiar el modo de visibilidad significa poder visualizar tres tipos de encuestas: privadas, de compañía o públicas. La primera se corresponde con aquellas encuestas propias del usuario que aún no han sido publicadas por lo que sólo el usuario puede visualizarlas; el tipo compañía se refiere a todas aquellas encuestas que pertenezcan a la misma compañía del usuario; y la tercera muestra las encuestas que están disponibles para todas las compañías, es decir, accesibles para cualquier usuario de *nubii*.

El botón de crear situado en la parte derecha inferior genera una nueva vista con el diseño presentado en la ilustración 8.12, en la que el usuario puede crear una encuesta compuesta de una pregunta y un máximo de 5 opciones además, cada opción puede tener una imagen asociada.

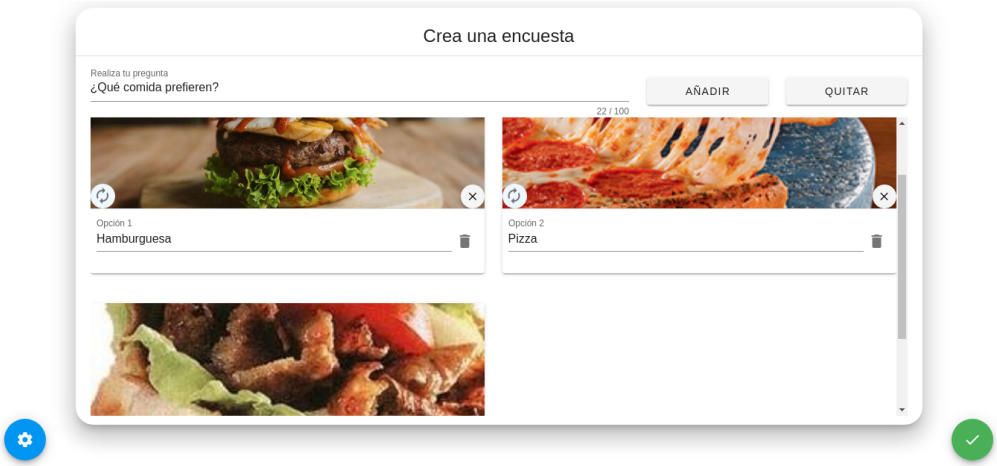


Ilustración 8.12: Creación de una encuesta en *Encuestas flash*

Dentro de la vista de creación también está disponible una vista de configuración de la encuesta (ilustración 8.13), en la que se puede editar el rango de fechas en las que la encuesta estará activa, las categorías de ésta, la visibilidad de la encuesta, un color característico y, dos opciones adicionales, la primera especifica si se muestran los resultados una vez se ha votado y la segunda si se muestra los usuarios que votaron.

Ilustración 8.13: Configuración de una encuesta en *Encuestas flash*

Para persistir un objeto encuesta en la base de datos se emplean dos tablas distintas, una de ellas almacena el título de la encuesta, todas sus configuraciones y una colección de opciones que se relacionan con la segunda tabla. Esta última almacena una fila por cada opción creada, donde los campos de este objeto son un identificador de opción, una imagen guardada en formato *blob* y una referencia a la encuesta a la que pertenecen. Por tanto, existe una relación uno a muchos entre estas dos tablas, pues una encuesta puede tener muchas opciones pero una opción sólo puede pertenecer a una encuesta. Razón por la cual se ha habilitado la eliminación en cascada, consiguiendo así que, si una encuesta es eliminada, todas sus opciones también lo sean.

Cuando usuario que quiera participar en una encuesta, simplemente debe dirigirse al botón de `votar` de la encuesta deseada, este botón le redirigirá a una nueva vista, cuyo diseño se corresponde con la ilustración 8.14, en esta vista el usuario puede consultar las opciones disponibles, ver las imágenes de cada una y votar por la que más le guste. Es importante destacar que si la encuesta es anónima, no se mostrará ningún botón para consultar los usuarios que hayan participado.

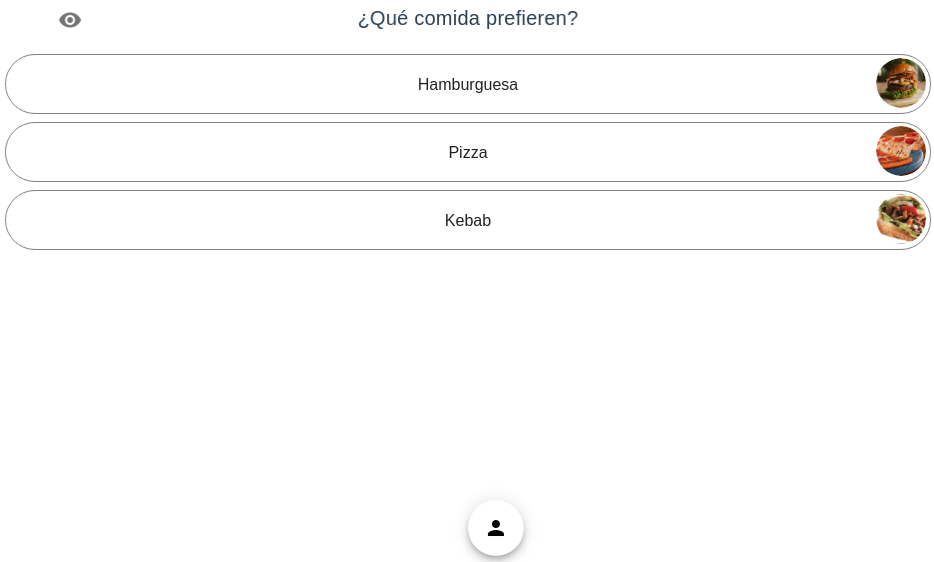


Ilustración 8.14: Participación en una encuesta en *Encuestas flash*

Una vez ha seleccionado y votado en la encuesta, ésta mostrará los resultados si la opción está habilitada (ilustración 8.15), en caso contrario informará al usuario que para ver los resultados debe volver el día después de la fecha de finalización de la encuesta.

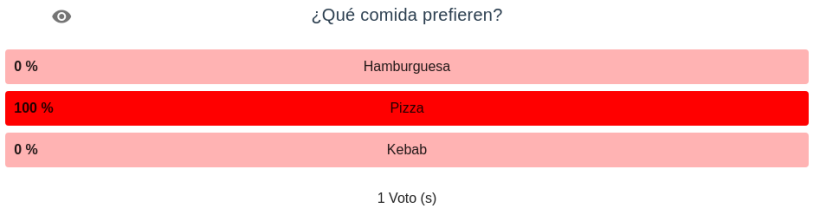


Ilustración 8.15: Resultados de una encuesta en *Encuestas flash*

Finalmente, para evitar que un usuario pueda votar dos veces en una misma encuesta, se registra el ID del usuario junto a la opción seleccionada en un listado almacenado en la base de datos, en el cual se comprueba si el ID existe cada vez que se intenta participar en una encuesta.

8.4. PPPS

Dentro de una empresa es importante disponer de un mecanismo para evaluar el desarrollo recurrente de los trabajadores, tanto si se desea conocer los avances que han tenido como los problemas a los que se han enfrentado, por este motivo se ha decidido incluir esta aplicación, la cual es una variante de *PPP*[46] (*problems, plans and progress*). Este formulario se desarrolla a nivel empresa, por lo que son los administradores de cada compañía los que deciden qué día se realiza el formulario, además de encargarse de revisarlos y obtener un resumen que pueda servir para entender la situación del equipo de trabajadores.

La variante desarrollada contiene una cuarta categoría que pregunta por las sugerencias (suggestions) que pueda tener el equipo. De estas cuatro categorías, dos son obligatorias y las otras dos no, esto se debe a que durante el periodo puede no haber surgido ningún problema o ninguna sugerencia, lo que sí tiene que especificar es la explicación de los progresos que haya realizado y los planes que tenga para el próximo periodo. El diseño de esta aplicación se corresponde con la presentada en la ilustración 8.16.

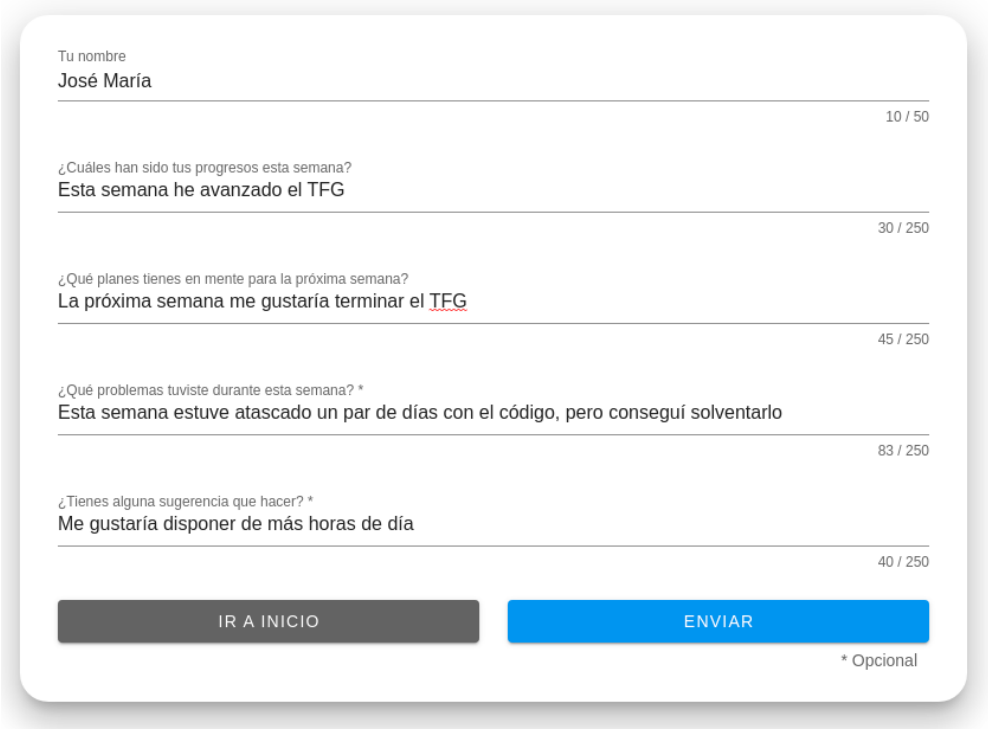


Ilustración 8.16: Interfaz de usuario de la aplicación *PPPS*

En esta primera fase del proyecto, este formulario únicamente puede ser realizado una vez a la semana, pues escoger el día es tarea de los administradores de cada compañía. Si un usuario que ya ha contestado el formulario semanal o el día actual es distinto al día del formulario, intenta realizar este formulario, entonces se le indicará, tal y como se aprecia en la ilustración 8.17, de dicho problema mediante unos cuadros de diálogo informativos.

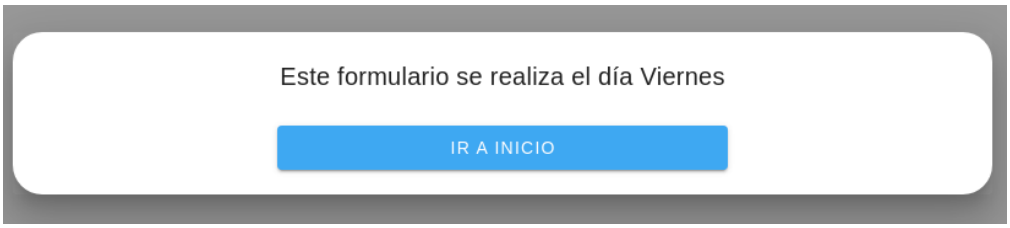


Ilustración 8.17: Control de usuarios en la aplicación *PPPS*

El apartado más importante de esta aplicación es que se puede consultar los formularios y filtrarlos para recoger datos relevantes para la empresa. Esta tarea sólo lo puede realizar un administrador, por tanto, si un usuario posee este rol, dispondrá de un botón que le redirigirá a una vista de administración de formularios con el siguiente diseño mostrado en la ilustración 8.18.

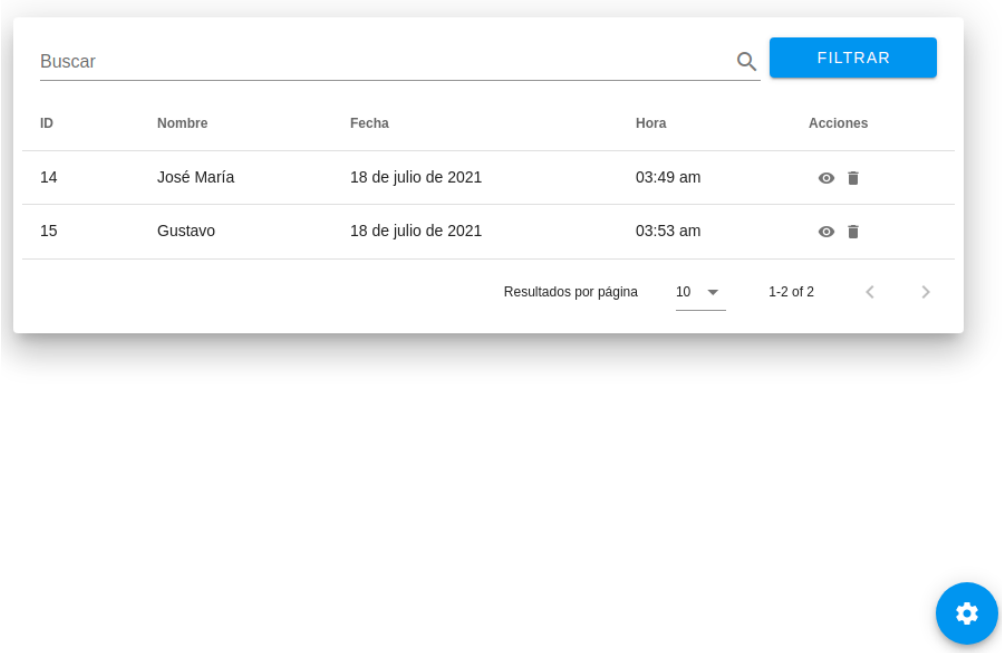


Ilustración 8.18: Vista del administrador en *PPPS*

Para cada formulario realizado, el administrador puede visualizarlo o eliminarlo mediante los iconos de acción situados en la parte derecha. En la ilustración 8.19 se presenta la vista del diálogo de consulta de formularios.

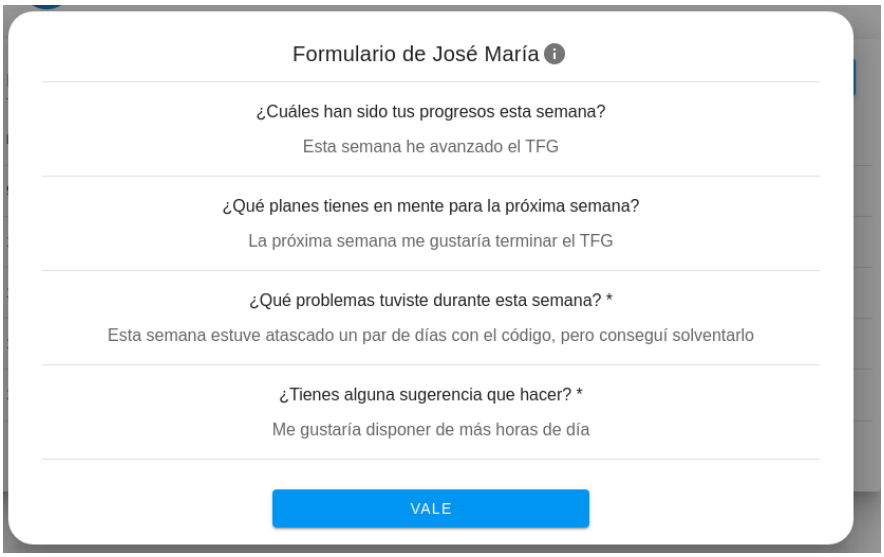


Ilustración 8.19: Consulta de un formulario en *PPPS*

Finalmente, el botón de ajustes de la parte inferior derecha abre un cuadro de diálogo para poder seleccionar el día de la semana que se desea realizar el formulario. Entre las mejoras futuras está presente la posibilidad de poder establecer este formulario de manera quincenal, mensual o como el usuario crea conveniente.

8.5. Fotografías de empresa

Ésta es la última aplicación desarrollada e incluida en el proyecto, cuyo nombre es `Fotos de empresa`. Su objetivo es proporcionar a los usuarios de *nubii* un modo de almacenar y compartir fotografías de la empresa, ya sea de sus trabajadores, los proyectos que han realizado y eventos o fiestas. Al igual que las aplicaciones anteriores, ésta se estructura a nivel de empresa, por lo que cada una posee su espacio de almacenamiento fijo y privado. Para que esta aplicación funcione se ha empleado un servicio de *Amazon Web Service (AWS)* denominado *Amazon Simple Storage Service* o*Amazon S3*.

De este modo, cuando un usuario accede a la aplicación, éste se encontrará con un mural de fotografías diseñado empleando la técnica de estructuras denominada *Masonry*, la cual considera cada objeto como una unidad, apilando dicho objeto en el hueco que exista sin tener en cuenta el resto, este diseño se presenta en la ilustración 8.20. Por este motivo, para llevar a cabo este diseño dentro del proyecto se ha empleado la librería *vue-masonry*. Pero, además del mural de fotografías, también se presenta una barra de búsqueda y filtrado y un botón para agregar una nueva imagen al mural, tal como se aprecia en la imagen siguiente.

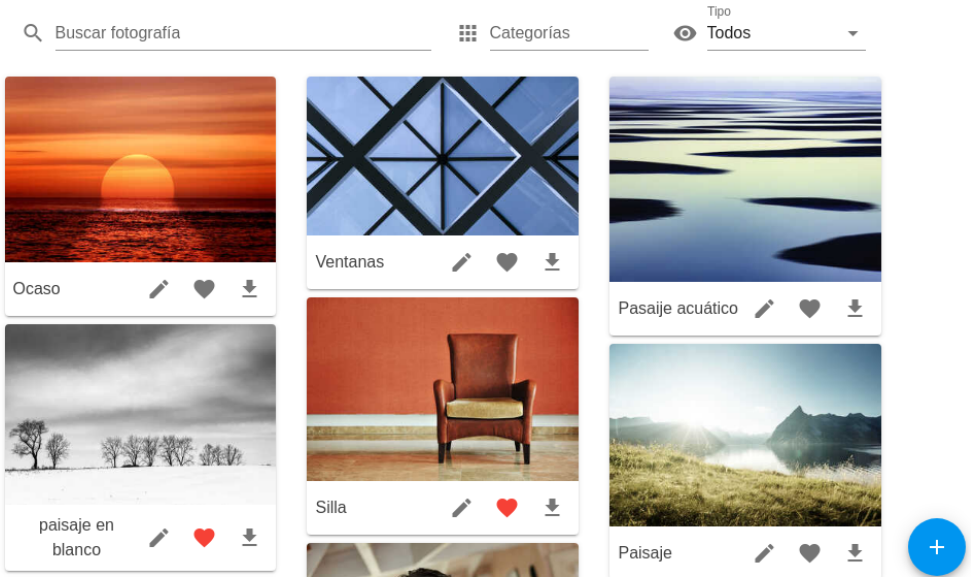


Ilustración 8.20: Interfaz de usuario de la aplicación *fotografías de empresa*

La barra superior proporciona un campo de búsqueda por título de imagen, un filtrado por categorías representados en formato de *chips*, y opciones para ordenar las fotografías por fecha, nombre o número de corazones. Y, también se presenta un selector de tipos, pues cada fotografía subida debe pertenecer obligatoriamente a alguno de los cuatro tipos presentes: Conocer, Personas, Eventos o Material.

Para agregar una nueva fotografía, se debe presionar sobre el botón azul situado en la parte inferior derecha, la cual redirigirá al usuario a una nueva vista, tal y como se muestra en la ilustración 8.21, con las opciones necesarias para añadir una nueva imagen.

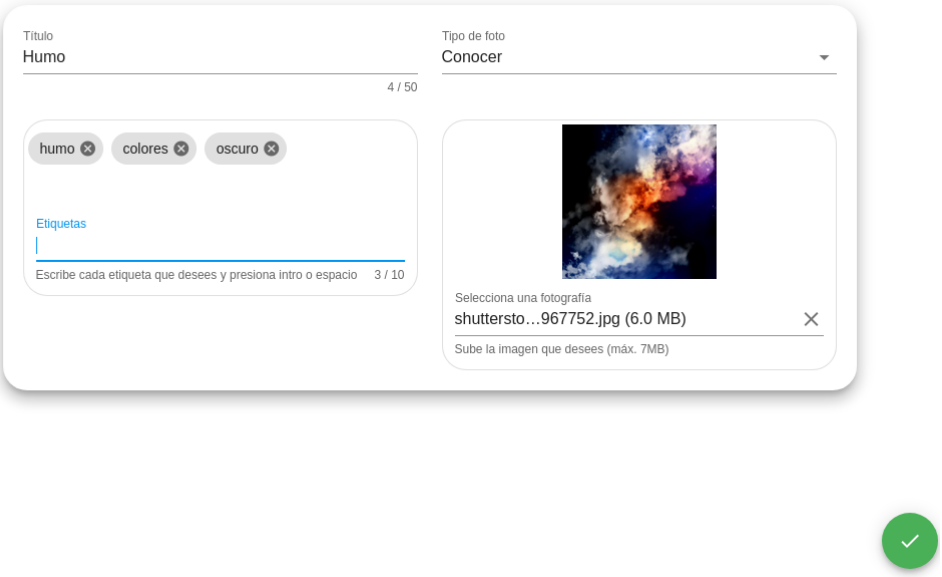


Ilustración 8.21: Vista de creación de imágenes en *fotografías de empresa*

En la ilustración 8.21 se puede apreciar que toda fotografía debe disponer de un título, un tipo de fotografía y una imagen que no sobrepase los 7 MegaBytes. Las etiquetas son opcionales.

El problema principal de esta aplicación fue que cargar una gran cantidad de imágenes de gran tamaño al mismo tiempo en el navegador desembocaba en la saturación de memoria, por lo que se ha desarrollado una paginación reactiva, en la que se van cargando imágenes independientemente cada vez que el usuario supere un umbral a la hora de realizar un *scroll* hacia abajo con el ratón del ordenador. Otro mecanismo para evitar la saturación fue reducir la resolución y tamaño de la imagen mostrada en el mural y, solamente, cuando el usuario desee descargar o ver la imagen en un tamaño más grande (ilustración 8.22), la imagen original alojada en el cubo de *Amazon S3* es obtenida sin modificaciones. Para llevar acabo esto se empleó la librería de manipulación de imágenes para *PHP* denominada *Imagick*.

Ilustración 8.22: Modal para la visualización de imágenes en *fotografías de empresa*

Cada fotografía que es subida sigue un flujo de procesamiento específico, pues lo primero es comprobar que el objeto transferido contiene todos los campos necesarios para que, posteriormente, pueda ser creado un objeto de tipo *PhotoBoardFile*, el cual será obtenido en los eventos de persistencia de la entidad *PreValidate* y *PreWrite*, en el primero se genera un nombre único para la imagen, además de obtener el tipo de imagen y generar un icono de ésta que pueda servir como imagen de espera mientras se está cargando la original, es decir, que pueda servir como *lazy-image*; en el segundo evento (*PreWrite*) se procede a cargar la imagen en el cubo *S3*, mediante las funciones del *SDK* de *AWS*.

Cualquier usuario podrá realizar un *CRUD* de su imagen, mientras que un usuario que posea un rol de administrador podrá realizar estas acciones sobre todas las imágenes de su compañía.

Para terminar, cabe destacar que se ha establecido una variable en la *query* de petición de una imagen, mediante la cual se especifica si se desea obtener dicha imagen alterada (`preload=1`) o no (`preload=0`). Y, como futuras mejoras, se tiene pensado habilitar una opción para compartir las imágenes por redes sociales o, incluso, poder descargarlas al teléfono móvil mediante un código *QR*.

Capítulo 9

Resultados y pruebas

Para poder realizar pruebas, la empresa me proporcionó una máquina dentro de sus servidores en la que pude descargar el código del proyecto mediante línea de comandos desde el repositorio de *GitLab*. Esta máquina disponía del servicio de *Docker* por lo que, una vez descargado el código, simplemente tuve que activar ambas partes del proyecto empleando para ello las instrucciones definidas en los ficheros *Makefile*. Esta máquina posee una dirección *URL* perteneciente al dominio de la empresa, facilitando, de este modo, su inclusión dentro del proyecto de *nubii* en los entorno de preproducción y producción.

Todas las pruebas del proyecto fueron realizadas en el entorno de preproducción, en un principio con un grupo de tres personas que se encargaban de probar la aplicación y recoger los errores y mejoras a realizar. Durante esta etapa la retroalimentación de estas personas fue muy positiva pues les agradaba el diseño de la aplicación, su estructuración y funcionalidades, sobretodo, la aplicación de fotografías, y los errores y mejoras recogidos fueron solucionados y realizados. Por este motivo, se puede concluir que los primeros resultados del proyecto fueron prometedores, pues funcionaba adecuadamente y recibió una buena retroalimentación.

Recientemente, el proyecto ha sido habilitado a todos los usuarios de la empresa, es decir, a todos los trabajadores de *The Singular Factory SL*, por lo que ahora mismo se encuentra en una etapa de prueba masiva en la que se probará la simultaneidad de usuarios, así como la subida de fotografías, envío de correos, entre otras pruebas. De este modo, una vez superada esta prueba, se procederá a ampliar el número de usuarios con acceso, incluyendo a todos los usuarios de *nubii*. Así que, finalmente, el proyecto será transferido a un entorno de producción.

Capítulo 10

Conclusiones y trabajo futuro

Habiendo concluido este proyecto, se puede echar la mirada hacia atrás y analizar cada parte, cada decisión, cada problema surgido durante su desarrollo, con el objetivo de aprender de todas estas situaciones y mejorar para proyectos futuros. Pues hubo momentos en los que demoraba varios días o semanas para progresar, ya que existían muchas nuevas tecnologías que aprender a usar. Un ejemplo de ello fue la primera vez que me enfrenté al *framework* de *Symfony* el cual posee una curva de aprendizaje muy alta, por lo que me llevó varias semanas poder entender toda su estructura, y pasó lo mismo con el *framework* del cliente. Sin embargo, una vez que se aprende a usarlos el desarrollo del proyecto avanza de mejor manera y siguiendo una buena estructuración. Pues esto se ve reflejado en los tiempos que se tardó en realizar la primera de las aplicaciones y la última de ellas, existiendo, fácilmente, una diferencia de, al menos, un mes.

Observando todos mis conocimientos actuales, estoy seguro que ahora podría realizar, nuevamente, este proyecto y tenerlo listo en un menor tiempo posible siguiendo buenas prácticas desde un comienzo. Sin embargo, cabe remarcar que, a pesar de haber aprendido a usar estas herramientas, el tiempo de aprendizaje nunca termina pues siempre existirán y se desarrollarán nuevas herramientas. Por todo lo mencionado anteriormente, puedo concluir que estoy bastante satisfecho con el trabajo entregado pues representa el producto de varios meses y semanas de arduo trabajo, así como la integración de muchos conocimientos adquiridos durante la carrera y la estadía en la empresa.

Para finalizar, este proyecto posee un gran margen de mejora, dado que cada una de las aplicaciones posee una lista de mejoras que se realizarán en etapas posteriores de desarrollo, tal es el caso de la aplicación de mensajería en la cual se requiere enviar la frase de contraseña por un medio distinto, o la aplicación de fotos de empresa en la cual se tiene pensado añadir la posibilidad de compartir imágenes por redes sociales o descargarlas en el teléfono móvil mediante un código *QR*, entre otras mejoras individuales. Y, también recordar que este proyecto no dejará de crecer puesto que se tiene pensado seguir añadiendo más funcionalidades a medida que éstas sean requeridas por los usuarios. Por este motivo, seguiré desarrollando este producto en un futuro cercano, con el objetivo de seguir mejorando el proyecto, y aprendiendo y adquiriendo más experiencia para mi futuro laboral.

Capítulo 11

Manual de usuario y software

Para poder acceder a la aplicación es necesario seguir una serie de pasos, además de disponer de ciertos permisos y credenciales en la plataforma de *nubii*.

- **Acceso a *nubii***: Es indispensable disponer de una credenciales válidas en esta plataforma, pues el proyecto se encuentra en su interior. Por lo que, disponiendo de los correspondientes datos, se procede a acceder a la plataforma a través de este formulario de acceso de *nubii*[30] presentado en la ilustración 11.1.

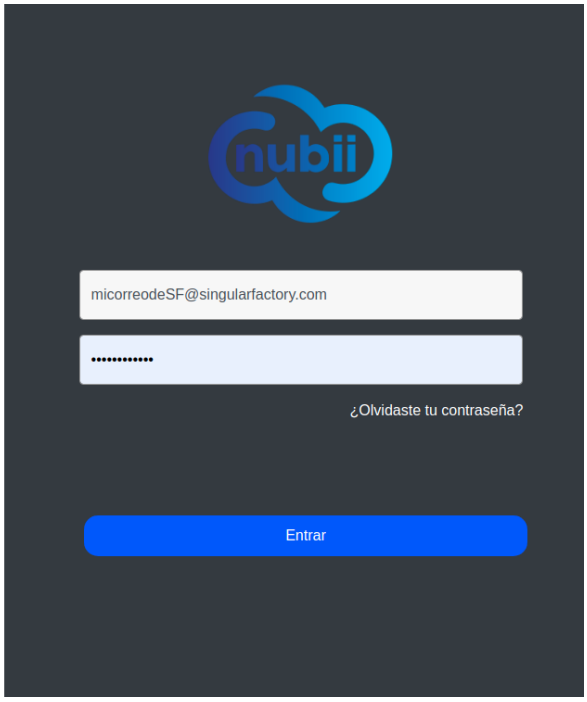


Ilustración 11.1: Formulario de acceso a *nubii*

- **Interfaz y acceso al proyecto**: Una vez accedido a la plataforma, se presentará una interfaz con avatares y una oficina, tal como se explicó en capítulos anteriores. Dentro de esta oficina es necesario situar el cursor sobre algún *redpoint*, tal y como se observa en la ilustración 11.2, el cual debe haber sido previamente configurado por un administrador con la dirección *URL* del proyecto.

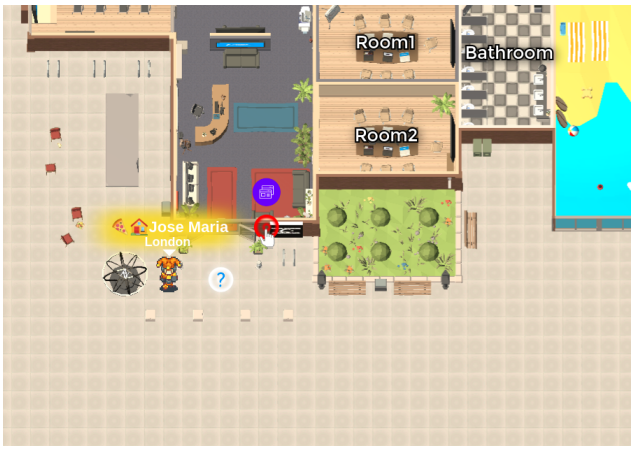


Ilustración 11.2: Acceso al proyecto a través de un *redpoint*

- **Proyecto Toolkit**: Después de hacer clic sobre el *redpoint*, se abrirá una pestaña embebida dentro de la misma plataforma (ilustración 11.3). Desde este punto, el usuario puede acceder a cualquiera de las aplicaciones que desee, además de cerrar la pestaña embebida y volverla a abrir sin problema alguno.

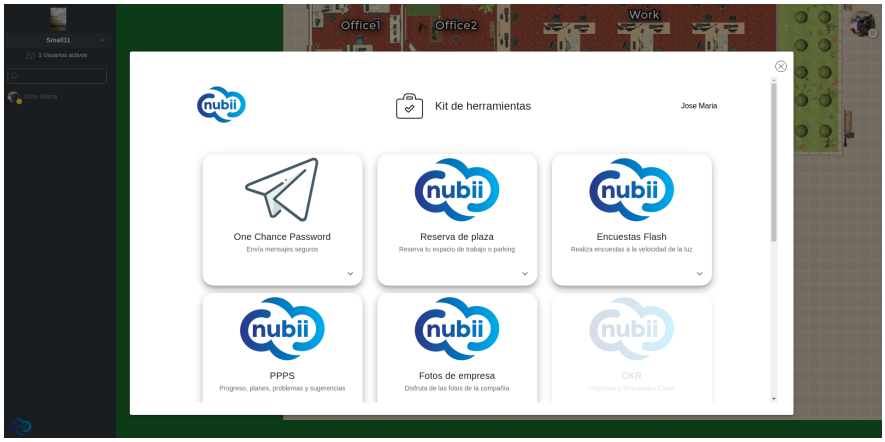


Ilustración 11.3: Acceso al proyecto a través de un *redpoint*

Bibliografía

[1] Amazon Web Services, Inc. (10 de Julio de 2021). What is aws. https://aws.amazon.com/what-is-aws/.

[2] Atlassian (28 de Junio de 2021). What is git. https://www.atlassian.com/git/tutorials/what-is-git.

[3] Britannica (19 de Julio de 2021). The sims — electronic game. https://www.britannica.com/topic/The-Sims.

[4] ClickUp (01 de Julio de 2021). Clickup - one app to replace them all. https://app.clickup.com/.

[5] CodeSeven (12 de Julio de 2021). Simple javascript toast notifications. https://codeseven.github.io/toastr/.

[6] Cristian Moreno (28 de Junio de 2021). *ESLint* usando la guía de estilo de *Airbnb*. https://eslint.org/.

[7] Dan Ackroyd (10 de Julio de 2021). Imagick. https://github.com/Imagick/imagick.

[8] Daniel Kelly (12 de Julio de 2021). What is a store in vue.js. https://vueschool.io/articles/vuejs-tutorials/what-is-a-store-in-vue-js/.

[9] Derick Rethans (28 de Junio de 2021). *Xdebug*. https://xdebug.org/.

[10] Doctrine team (10 de Julio de 2021). Doctrine: Php open source project. https://www.doctrine-project.org/.

[11] Dusseault & Snell (12 de Julio de 2021). Patch method for http. https://datatracker.ietf.org/doc/html/rfc5789.

[12] Escuela de Ingeniería Informática (15 de Julio de 2021). Objetivos y competencias del gii. https://www.eii.ulpgc.es/tb_university_ex/?q=objetivos-y-competencias-del-gii#Segundo_Enlace.

[13] Evan You, Eduardo San Martin Morote (12 de Julio de 2021). Vue router. https://router.vuejs.org/.

[14] Fast Poll team (17 de Julio de 2021). Fast poll - create instant, real-time polls. https://fast-poll.com/.

[15] FSF & GNU (15 de Julio de 2021a). Gnu general public license. https://www.gnu.org/licenses/gpl-3.0.en.html.

[16] FSF & GNU (15 de Julio de 2021b). What is copyleft. https://www.gnu.org/licenses/copyleft.en.html.

[17] Google (12 de Julio de 2021c). Localizing your app. https://angular.io/guide/i18n.

[18] Google (17 de Julio de 2021a). Formularios de google. https://www.google.com/forms/about/.

[19] Google (17 de Julio de 2021b). Hojas de cálculo de google. https://www.google.com/sheets/about/.

[20] International Business Machines Corporation (IBM) (28 de Junio de 2021). What is docker. https://www.ibm.com/cloud/learn/docker.

[21] International Labour Organization (ILO) (19 de Julio de 2021). At least 23 million people have transitioned to teleworking. https://www.ilo.org/caribbean/newsroom/WCMS_811296/lang--en/index.htm.

[22] J. Román (Manz) (05 de Julio de 2021). Ciclo de vida de vue.js. https://lenguajejs.com/vuejs/componentes/ciclo-vida/.

[23] JetBrains (28 de Julio de 2021). *PhpStorm*. https://www.jetbrains.com/phpstorm/.

[24] Jones, et al (10 de Julio de 2021). Json web token (jwt). https://datatracker.ietf.org/doc/html/rfc7519.

[25] Kévin Dunglas (10 de Julio de 2021). Api platform: Rest and graphql framework. https://api-platform.com/.

[26] Mattermost, Inc. (17 de Julio de 2021). Mattermost: Open-source. https://mattermost.com/.

[27] Microsoft (19 de Julio de 2021). Video conferencing, meetings, calling. https://www.microsoft.com/en-ww/microsoft-teams/group-chat-software.

[28] Mozilla and individual contributors (12 de Julio de 2021). Promise - javascript — mdn. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[29] Nicholas C. Zakas (28 de Junio de 2021). *EsLint*. https://eslint.org/.

[30] nubii team (15 de Julio de 2021). Trabaja en remoto. https://nubii.us/es/.

[31] OpenJS Foundation (01 de Julio de 2021). About node.js. https://nodejs.org/en/about/.

[32] Opensource.org (15 de Julio de 2021). The mit license. https://opensource.org/licenses/MIT.

[33] Oracle Corporation (01 de Julio de 2021). Mysql documentation. https://dev.mysql.com/doc/.

[34] Pixabay team (17 de Julio de 2021). Más de 1 millón de imágenes. https://pixabay.com/es/.

[35] Postman, Inc (01 de Julio de 2021). Postman learning center. https://learning.postman.com/docs/getting-started/introduction/.

[36] Symfony SAS (05 de Julio de 2021). What is symfony? https://symfony.com/what-is-symfony.

[37] The Apache Software Foundation (01 de Julio de 2021b). Welcome!-the apache http server project. https://httpd.apache.org/.

[38] The Apache Software Foundation (15 de Julio de 2021a). Apache license. https://www.apache.org/licenses/LICENSE-2.0.

[39] The Axios Project (12 de Julio de 2021). Getting started — axios docs. https://axios-http.com/docs/intro.

[40] The Linux Foundation (01 de Julio de 2021). What is linux? https://www.linux.com/what-is-linux/.

[41] The New York Times (19 de Julio de 2021). A timeline of the coronavirus pandemic. https://www.nytimes.com/article/coronavirus-timeline.html.

[42] The PHP Group (01 de Julio de 2021). Php: Documentation. https://www.php.net/docs.php.

[43] The Singular Factory SL (15 de Julio de 2021). Singular factory. https://www.singularfactory.com/.

[44] Vue.js team (05 de Julio de 2021). Introduction -vue.js. https://vuejs.org/v2/guide/.

[45] Vuetify Team (05 de Julio de 2021). Vuetify - a material design framework. https://vuetifyjs.com/en/.

[46] Weekdone (15 de Julio de 2021). Ppp: Progress, plans, problems. https://weekdone.com/resources/plans-progress-problems.

[47] Zoom Video Communications, Inc (19 de Julio de 2021). Videoconferencia, teléfono nube, seminario web. https://zoom.us/.