UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Dashboard para la gestión de la información de una webpage en su conexión a su endpoint

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Nieto Soto, Rafael

Tutor/a: Gil Pechuán, Ignacio

CURSO ACADÉMICO: 2022/2023

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Dashboard para controlar cualquier página web conectada a su endpoint

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Nieto Soto, Rafael

Tutor: Gil Pechuán, Ignacio

2022/2023

Resumen

A lo largo de este Trabajo de Fin de Grado se desarrollará el prototipo de un Dashboard Admin el cual permitirá crear, leer, modificar o eliminar datos (CRUD) de cualquier base de datos interconectada mediante endpoints. Se analizarán diferentes posibilidades en su desarrollo, estudiando cual podría ser la forma más conveniente de hacerlo.

Se presentarán dos casos de uso independientes entre sí para demostrar que se trata de una herramienta flexible con altas posibilidades.

Como primer caso de uso, se resolverá la gestión de los servicios públicos de la ciudad de Valencia utilizando tecnología actual, mejorando el tiempo del administrador de servicios al hacer su trabajo más sencillo y rápido. Además, como otro caso de uso, se demostrará que es posible conectar el dashboard a WordPress[1], en caso de que la empresa o institución lo requiera, para controlar la gamificación de su página web mediante un plugin.

El resultado de este TFG es un dashboard conectado a la base de datos de dos páginas web de muestra, los bomberos de Valencia y la policía local de Valencia, y a un plugin de WordPress mediante microservicios, de forma que el encargado del dashboard pueda realizar cualquier operación sobre dichas bases de datos en el mínimo tiempo posible.

Como conclusión de este TFG se destaca la gran repercusión que podría tener el uso de una herramienta de centralización de datos para cualquier tipo de actividad. Se podría aumentar, no solo la velocidad de las gestiones, sino también la seguridad al centralizar la forma en la que se accede a información crítica.

Palabras clave: dashboard, gestión, administración, CRUD, base de datos, microservicios, diseño web, WordPress, gamificación.

Abstract

Throughout this Thesis, the prototype of an Admin Dashboard will be developed, which will allow creating, reading, updating, and deleting (CRUD) data from any interconnected database through endpoints. Different possibilities will be analyzed in its development, studying what could be the most convenient way to do it.

Two independent use cases will be presented to demonstrate that it is a flexible tool with high possibilities.

As the first use case, the management of public services in the city of Valencia will be addressed using current technology, improving the time of the service administrator by making their work simpler and faster. Furthermore, as another use case, it will be demonstrated that it is possible to connect the dashboard to WordPress[1], in case the company or institution requires it, to control the gamification of their website through a plugin.

The result of this Thesis is a dashboard connected to the database of two sample websites, the Valencia Fire Department and the Valencia Local Police, and to a WordPress plugin through microservices, so that the dashboard manager can perform any operation on these databases in the shortest possible time.

As a conclusion of this Thesis, the significant impact of using a data centralization tool for any type of activity is highlighted. It could not only increase the speed of management processes but also enhance security by centralizing the way critical information is accessed.

Keywords: dashboard, management, administration, CRUD, database, microservices, web design, WordPress, gamification.

Tabla de contenidos

Tabla de contenido

Tabla de ilustraciones

1. Introducción

En la sociedad actual, el avance de la tecnología se ha convertido en una carrera imparable, con descubrimientos y desarrollos que ocurren a una velocidad vertiginosa. Cada día, nuevas innovaciones y soluciones tecnológicas emergen, transformando radicalmente la forma en la que vivimos, trabajamos y nos relacionamos. En este contexto, es crucial que las empresas se mantengan a la vanguardia de esta revolución tecnológica para asegurar su competitividad y éxito a largo plazo.

El uso de la tecnología se ha vuelto esencial para las empresas en diversos aspectos de su funcionamiento. En primer lugar, la implementación de soluciones tecnológicas eficientes permite automatizar tareas y procesos, ahorrando tiempo y recursos valiosos. Además, las herramientas tecnológicas mejoran la eficiencia operativa al eliminar errores humanos y agilizar la comunicación y colaboración entre los equipos.

Además, el acceso a datos y la capacidad de análisis se han vuelto cruciales en el entorno empresarial actual. Las empresas se enfrentan a volúmenes masivos de información que necesitan gestionar y aprovechar para tomar decisiones informadas y estratégicas. Aquí es donde entra en juego la importancia de utilizar una herramienta de gestión de datos, como un dashboard, tema a tratar en este TFG.

Un dashboard proporciona una visión clara y concisa de los datos clave de una empresa, presentándolos de manera visual y fácilmente comprensible. Esta visualización en tiempo real permite a los responsables de la toma de decisiones tener una panorámica integral del estado de la empresa, identificar tendencias y patrones, y evaluar el rendimiento en base a métricas relevantes.

Además, una herramienta como esta ofrece la capacidad de personalizar y adaptar los indicadores y gráficos según las necesidades específicas de cada empresa. Esto permite a los usuarios tener una vista en tiempo real de los aspectos más relevantes de su negocio y establecer alertas o notificaciones para mantenerse informados sobre cualquier desviación o cambio significativo.

1.1 Motivación

La motivación de dedicar el TFG al estudio y desarrollo de un dashboard y a sus diferentes posibilidades arquitectónicas ha sido dada por dos motivos.

En primer lugar, por tratarse de una herramienta muy flexible y de uso general con usos prácticamente ilimitados. Mientras que para una empresa podría ser interesante utilizarla únicamente para una visualización y estudio de datos, otra empresa podría querer usarla para controlar la apariencia completa de varias páginas webs. Además, es una herramienta sin fecha de caducidad, ya que, si el mercado o el rumbo general de la tecnología cambian, resultaría sencillo actualizarla, sin necesidad de desecharla completamente.

Por otro lado, es una herramienta que voy a estar utilizando a largo plazo para uso personal. Tengo intención de controlar todos mis futuros proyectos desde ella como forma de centralizar mis propios datos, sin necesidad de estar moviéndome entre proyectos para visualizar la información. A corto plazo, tengo intención de desarrollar dos aplicaciones, una de gimnasio y otra de nutrición, las cuales gestionaré desde este dashboard.

1.2 Objetivos

Para la realización de este proyecto se tendrá en cuenta en todo momento una serie de puntos como objetivo final:

- Estructura escalable: El dashboard desarrollado en este TFG es una herramienta muy flexible a la cual se le pueden dar una gran cantidad de usos diferentes. Por este motivo, se ha decidido utilizar una arquitectura basada en microservicios en lugar de una monolítica, de forma que sea rápido, sencillo y ordenado el mantenimiento y ampliación de las utilidades de la plataforma.

- Sencillez de uso: Ya que el usuario final puede no ser un perfil técnico, la interfaz de la plataforma debería ser lo más intuitiva posible. La información que se visualiza en pantalla la mayoría del tiempo debería ser mínima, y las funciones más complejas que se utilizaran la minoría del tiempo deberían no estar tan a la vista.

- Código mantenible: A nivel técnico, los estándares que se utilizaran en el desarrollo del código se centran en reducir y reutilizar el código lo máximo posible siempre y cuando no interfiera en su legibilidad.

Los componentes a desarrollar que formarán el resultado final estarán compuestos por:

- Un dashboard.
- 3 microservicios.
- 2 páginas web de ejemplo.
- Un plugin de WordPress

Una vez que todos los componentes estén desarrollados y conectados, se deberá poder modificar el aspecto de las páginas web y las tablas de WordPress desde el dashboard sin necesidad de realizar ningún cambio en el código fuente de las páginas web o en el propio panel de administración de WordPress.

1.3 Metodología

Debido a que este proyecto se ha realizado en un largo periodo de tiempo y los cambios han estado a la orden del día, se ha optado por utilizar una metodología Scrum. Se trata una metodología ágil de gestión de proyectos utilizada de forma frecuente en el desarrollo de software que se basa en ciclos de trabajo llamados "sprints", donde los desarrolladores colaboran de manera flexible y entregan resultados de forma continua. Con roles claros, reuniones diarias de seguimiento y evaluaciones al final de cada sprint, Scrum fomenta la transparencia, la adaptación y la entrega de valor, permitiendo a los equipos responder rápidamente a los cambios y necesidades del proyecto, convirtiéndola en una metodología efectiva para proyectos grandes, complejos y, sobre todo, cambiantes.

Aunque la metodología Scrum está diseñada para el trabajo en equipo, algunos principios y prácticas también pueden ser útiles para un trabajo individual. Al adoptar una mentalidad ágil y aplicar conceptos como la planificación basada en sprints, la priorización de tareas y la evaluación continua del progreso, es posible mejorar la productividad, mantener un enfoque claro en los objetivos y adaptarse de manera flexible a los cambios y desafíos que puedan surgir en el trabajo individual. Aunque puede requerir ciertas adaptaciones o tiempos extra, la esencia de Scrum, como la transparencia, la inspección y la adaptación, puede ser beneficiosa incluso para proyectos individuales.

A lo largo del desarrollo de este proyecto, el trabajo se ha dividido en sprints, centrando toda la atención en una parte determinada del proyecto en cada periodo de sprint. Para la gestión de estos sprints se ha utilizado la herramienta Jira[2], la cual facilita en gran medida este tipo de metodología.

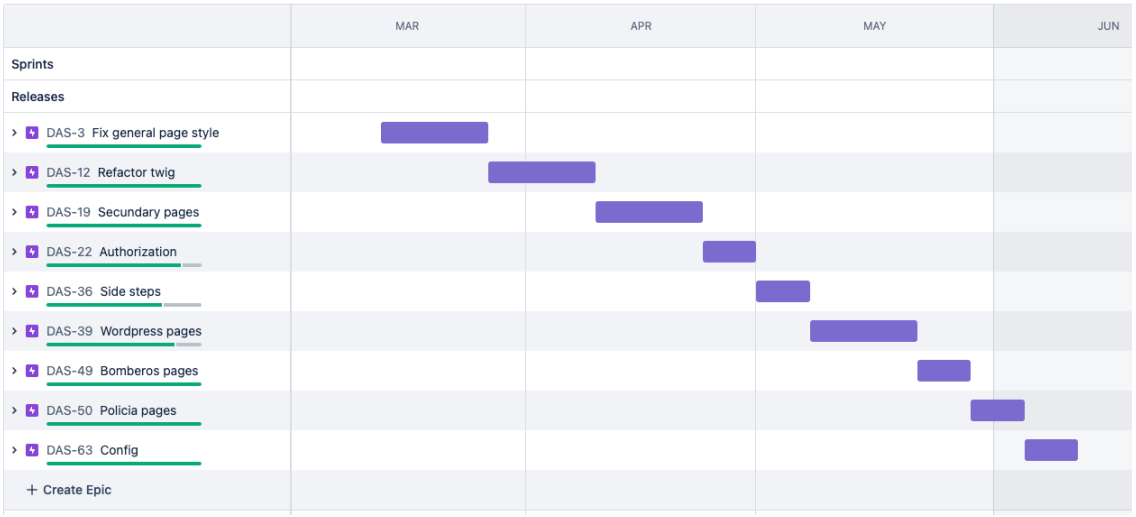


Ilustración 1 - Sprint de Jira

Cada uno de estos sprints está dividido en tareas, con la posibilidad de añadir una descripción, y modificar su estado para diferenciar las tareas pendientes de las que ya han sido terminadas:

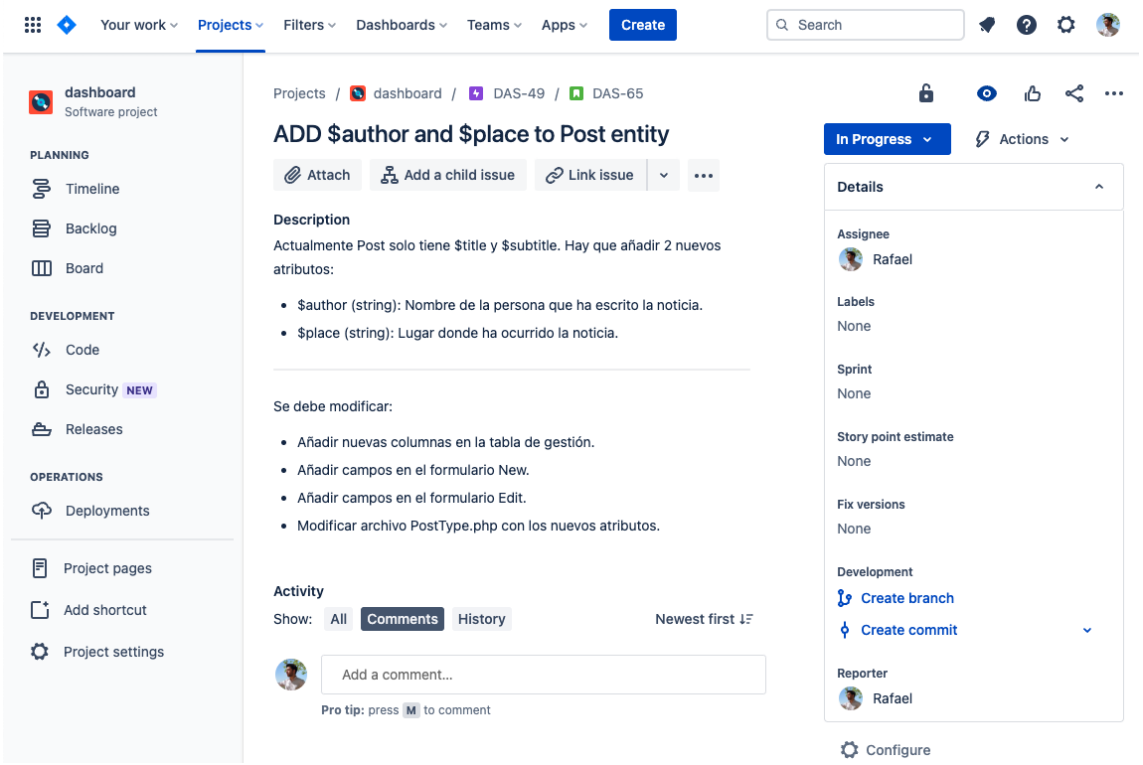


Ilustración 2 - Tarea de Jira

Una vez se ha seleccionado una tarea, se utiliza su identificador, "DAS-65" en el caso de la tarea de la imagen, para crear una nueva rama de Git en la que se desarrollará:

```
rafa@MacBook-Pro-de-Rafael dashboard % git checkout -b DAS-65
```

Ilustración 3 - Comando "git checkout"

Una vez se ha terminado de desarrollar la tarea, se procede a crear un Pull Request en Bitbucket donde se podrán visualizar todos los cambios, comprobando así que no haya errores en el código. Tras ello, se procede a fusionar la rama con el código base:

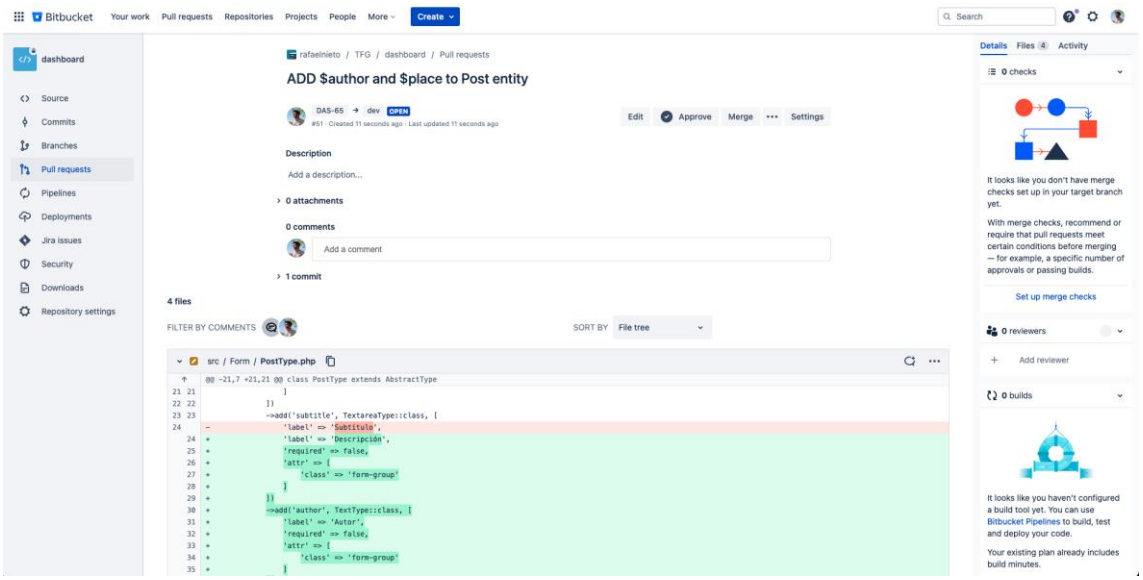
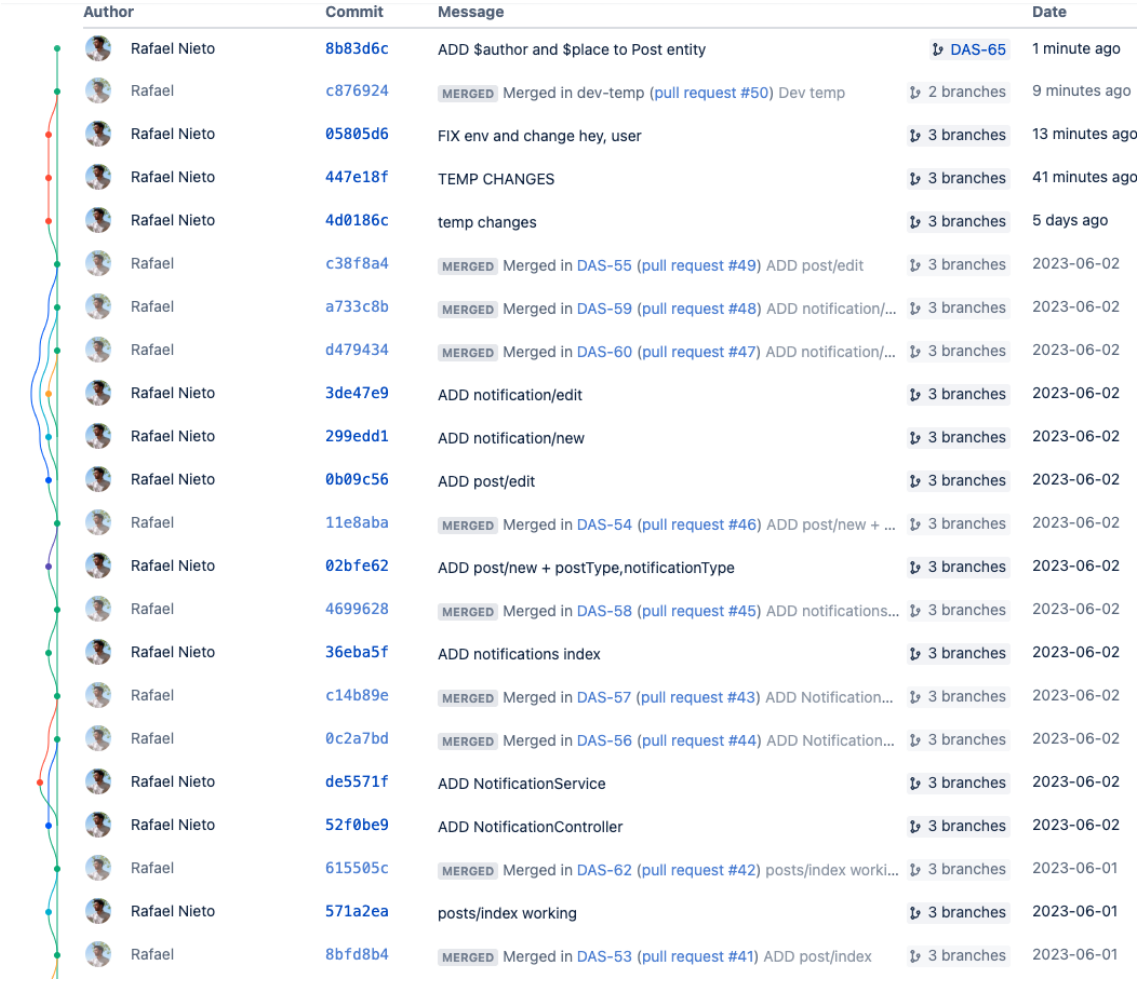


Ilustración 4 - Pull Request en Bitbucket

Bitbucket permite además visualizar todos los commits pasados, de forma que es sencillo determinar en qué momento se desarrolló algo erróneo que esté haciendo fallar alguna funcionalidad. Ya que la mayoría de las tecnologías utilizadas en este TFG han sido nuevas para mí y he tenido que enfrentarme a una gran cantidad de errores, ha sido de gran utilidad el poder detectar el commit concreto que provocaba un error y revertirlo:



Author	Commit	Message		Date
Rafael Nieto	8b83d6c	ADD $author and $place to Post entity	DAS-65	1 minute ago
Rafael	c876924	MERGED Merged in dev-temp (pull request #50) Dev temp	2 branches	9 minutes ago
Rafael Nieto	05805d6	FIX env and change hey, user	3 branches	13 minutes ago
Rafael Nieto	447e18f	TEMP CHANGES	3 branches	41 minutes ago
Rafael Nieto	4d0186c	temp changes	3 branches	5 days ago
Rafael	c38f8a4	MERGED Merged in DAS-55 (pull request #49) ADD post/edit	3 branches	2023-06-02
Rafael	a733c8b	MERGED Merged in DAS-59 (pull request #48) ADD notification/...	3 branches	2023-06-02
Rafael	d479434	MERGED Merged in DAS-60 (pull request #47) ADD notification/...	3 branches	2023-06-02
Rafael Nieto	3de47e9	ADD notification/edit	3 branches	2023-06-02
Rafael Nieto	299edd1	ADD notification/new	3 branches	2023-06-02
Rafael Nieto	0b09c56	ADD post/edit	3 branches	2023-06-02
Rafael	11e8aba	MERGED Merged in DAS-54 (pull request #46) ADD post/new + ...	3 branches	2023-06-02
Rafael Nieto	02bfe62	ADD post/new + postType,notificationType	3 branches	2023-06-02
Rafael	4699628	MERGED Merged in DAS-58 (pull request #45) ADD notifications...	3 branches	2023-06-02
Rafael Nieto	36eba5f	ADD notifications index	3 branches	2023-06-02
Rafael	c14b89e	MERGED Merged in DAS-57 (pull request #43) ADD Notification...	3 branches	2023-06-02
Rafael	0c2a7bd	MERGED Merged in DAS-56 (pull request #44) ADD Notification...	3 branches	2023-06-02
Rafael Nieto	de5571f	ADD NotificationService	3 branches	2023-06-02
Rafael Nieto	52f0be9	ADD NotificationController	3 branches	2023-06-02
Rafael	615505c	MERGED Merged in DAS-62 (pull request #42) posts/index worki...	3 branches	2023-06-01
Rafael Nieto	571a2ea	posts/index working	3 branches	2023-06-01
Rafael	8bfd8b4	MERGED Merged in DAS-53 (pull request #41) ADD post/index	3 branches	2023-06-01

Ilustración 5 - Listado de commits en Bitbucket

Dentro de lo personal, ha sido una muy buena práctica realizar el TFG bajo esta metodología, ya que me ha permitido organizarme y visualizar el trabajo que debía hacer cada semana. Los fines de semana siempre estoy fuera, y mientras estoy viajando sin acceso al ordenador he aprovechado para pensar en los siguientes pasos que debería dar el proyecto, además de estimar el tiempo que me llevaría su desarrollo. Tras ello, he ido creando el sprint de la siguiente semana o semanas, de forma que ya estuviera todo preparado para cuando volviera a tener acceso al ordenador. De esta forma, he evitado llegar a situaciones en las que no tengo nada que hacer, o en las que me he autoimpuesto unos plazos imposibles de cumplir.

2. Estado del arte

2.1 Situación actual de la tecnología

Desde que las empresas empezaron a dar tanta importancia a trabajar con distintas herramientas de software para modernizar y agilizar su forma de operar, es fundamental que cuenten con una herramienta para centralizar la gestión de sus datos.

Disponer de una herramienta así ofrece un acceso rápido y sencillo a los datos pertinentes, lo que, al proporcionar una imagen completa de todos los datos, mejora la toma de decisiones, debido a que, de otra forma, algunos datos podrían no tenerse en cuenta. Además, mejora la integridad y calidad de la información utilizada al reducir los errores humanos y la duplicación gracias a un único repositorio de datos confiable.

La capacidad de una herramienta de centralización de datos para realizar análisis exhaustivos y producir informes precisos ofrece una base eficaz para la toma de decisiones estratégicas. Los equipos pueden realizar análisis más completos y profundos, detectar patrones, tendencias y oportunidades comerciales al tener todos los datos en un solo lugar. También pueden obtener una imagen completa del estado actual de la empresa. Hacer esto hace posible tomar decisiones críticas en el momento oportuno para modificar la estrategia general de la empresa al adquirir esta nueva información.

Además, la capacidad de la herramienta de centralización para escalar a medida que la empresa se expande y produce volúmenes de datos aún más grandes es vital para administrar de manera efectiva el crecimiento de la información. La herramienta debe ser capaz de adaptarse al crecimiento sin comprometer la funcionalidad o la accesibilidad de los datos, asegurando que la empresa pueda seguir beneficiándose plenamente de la centralización de datos a medida que crece.

En conclusión, una herramienta de centralización de datos mejora el acceso, la calidad y la seguridad de la información, optimizando la toma de decisiones, la efectividad operativa, el cumplimiento normativo y la capacidad de análisis en una empresa. Además, promueve el trabajo en equipo y la comunicación, lo que permite un crecimiento escalable y sostenible.

Algunas herramientas con estas características utilizadas por las empresas en todo el mundo son, entre otras:

- Tableau[3]: Es una reconocida plataforma de visualización y análisis de datos que permite a las organizaciones transformar datos complejos en visualizaciones interactivas y significativas. Fue fundada en 2003 y adquirida por Salesforce en 2019. Tableau se destaca por su capacidad para simplificar el proceso de análisis de datos, lo que lo convierte en una herramienta popular en diversos sectores y empresas de diferentes tamaños.

- Power BI[4]: Power BI es una plataforma de inteligencia empresarial desarrollada por Microsoft. Permite a las organizaciones analizar datos, compartir información y obtener insights significativos a través de visualizaciones interactivas y herramientas de análisis. Power BI se ha convertido en una de las herramientas líderes en el campo de la visualización de datos y el análisis empresarial. Destaca por su facilidad de uso, su capacidad de análisis avanzado y su integración con el ecosistema de Microsoft.

- Zoho Analytics[5]: Es una plataforma de análisis y visualización de datos en la nube. Permite a las empresas recopilar, procesar y analizar datos de diversas fuentes para obtener insights significativos. Con Zoho Analytics, los usuarios pueden crear informes y paneles interactivos, realizar análisis avanzados, compartir información y colaborar en tiempo real. La plataforma ofrece una amplia gama de opciones de visualización, herramientas de modelado de datos y capacidades de análisis predictivo. Además, Zoho Analytics se integra con otras aplicaciones de Zoho y ofrece opciones de personalización, seguridad y cumplimiento normativo. Es una solución popular para empresas de diferentes tamaños y sectores que buscan aprovechar al máximo sus datos para la toma de decisiones informadas.

- QlikView[6]: Es una plataforma de inteligencia empresarial y análisis de datos que permite a las organizaciones transformar datos en información significativa y visualizaciones interactivas. Con QlikView, los usuarios pueden explorar y analizar datos de diversas fuentes, descubrir relaciones ocultas y obtener insights valiosos. La plataforma utiliza un enfoque asociativo único que permite a los usuarios navegar libremente por los datos sin restricciones predefinidas, lo que facilita la identificación de patrones y tendencias. QlikView ofrece una interfaz intuitiva y flexible para la creación de dashboards personalizados, informes y cuadros de mando interactivos. Además, proporciona funcionalidades de análisis avanzado, como la capacidad de realizar análisis de "arrastre y suelte", modelado de datos flexible y capacidades de descubrimiento guiado. QlikView se ha convertido en una solución popular para empresas que buscan visualizar y analizar sus datos de manera efectiva, impulsando la toma de decisiones basada en datos y el rendimiento empresarial.

2.2 Crítica al estado del arte

El uso de herramientas de inteligencia empresarial y análisis de datos, aunque puede aportar grandes ventajas estratégicas a la empresa, también puede presentar desafíos y problemas potenciales. Algunos de estos desafíos incluyen la complejidad inicial en la implementación y configuración, la integración de datos de múltiples fuentes, los requisitos de hardware y software, la privacidad y seguridad de los datos, la dependencia de expertos en datos y los costos asociados. Estos problemas pueden dificultar la adopción y el uso efectivo de la herramienta, especialmente para aquellos que carecen de conocimientos técnicos o recursos suficientes.

Sin embargo, con una planificación adecuada, capacitación y gestión eficaz, muchos de estos desafíos pueden superarse en poco tiempo. La elección de la herramienta adecuada, la gestión de datos de calidad, la seguridad de los datos y una buena administración de recursos pueden ayudar a minimizar los problemas potenciales y permitir que las organizaciones obtengan el máximo valor de su herramienta de inteligencia empresarial.

2.3 Propuesta al estado del arte

Este TFG podría haberse diseñado de prácticamente cualquier forma, ya que este tipo de herramientas suelen tener una forma de uso muy similar. Por lo tanto, y tras haber estudiado qué puntos serían los más interesantes a seguir como estrategia de desarrollo, se ha concluido que habría que poner especial atención en las siguientes características:

1. Intuitivo: Es posible que la persona encargada de manejar esta herramienta no sea un perfil técnico, por lo que se evitarán los términos técnicos y siempre se utilizará lenguaje natural en la descripción de las funcionalidades. Además, es buena idea que las páginas mantengan una misma similitud o patrón, para que el usuario no deba memorizar una gran cantidad de apartados diferentes dentro de la herramienta.

2. Integración de datos: La integración de una nueva herramienta en la empresa podría requerir grandes cambios en la arquitectura de esta. Para este TFG se ha decidido que los datos serán compartidos utilizando endpoints, de forma que la instalación de esta herramienta requiera de unos cambios mínimos en la arquitectura una empresa real, requiriendo únicamente un endpoint de salida en la base de datos que quiera gestionarse.

3. Sobrecarga de información: Este tipo de herramientas tienen la capacidad de proporcionar una gran cantidad de información al mismo tiempo. Sin embargo, si no se seleccionan y presentan los datos de manera adecuada, existe el riesgo de sobrecargar a los usuarios con una cantidad abrumadora de información. Por este motivo, se ha decidido separar la interfaz en diferentes páginas siempre que tenga sentido, para mostrar la justa cantidad de información necesaria.

3. Análisis del problema

3.1 Análisis de la seguridad

Debido a que en este TFG, para uno de los casos de uso, se tomaran los servicios públicos de Valencia, es importante estudiar cómo se definirá la seguridad. En este caso, lo primordial es decidir quién podrá utilizar esta herramienta, ya que estará en su mano el acceso a información altamente sensible.

Por este motivo, se ha decidido que la herramienta no tendrá una interfaz de registro donde cualquier usuario pueda registrarse con su correo personal. Por el contrario, los usuarios serán creados directamente en la base de datos por los directores del sistema, quienes deberán ser los únicos con este tipo de permisos.

Por otro lado, la aplicación sí dispondrá de una pantalla de login. En este TFG el proceso de login se realizará mediante correo y contraseña, aunque podría realizarse de diferentes formas, por ejemplo, con un código de identificación o con un nombre de usuario.

En esta herramienta, el formulario de login está desarrollado mediante el componente de autentificación Auth de Symfony, el cuál es sencillo de utilizar y proporciona una base sólida de protección contra ataques como podrían ser las inyecciones SQL.

3.2 Análisis de eficiencia algorítmica

En un proyecto de software basado en microservicios, es importante tener en cuenta el rendimiento y los costos asociados a las llamadas a los microservicios. Realizar llamadas innecesarias puede impactar negativamente en el rendimiento de la aplicación y aumentar los costos, especialmente si las llamadas implican comunicación a través de la red o servicios externos.

En el diseño de la algoritmia de esta herramienta, se ha buscado la forma de reducir en todo lo posible el número de peticiones a los microservicios. Nunca debería haber una llamada dentro de un bucle o de una funcionalidad como un buscador, ya que sería extremadamente costoso e ineficiente. Para que el proceso sea correcto, primero se llamará al microservicio una sola vez, y después se aplicará la lógica necesaria sobre los datos obtenidos.

Un ejemplo claro sobre lo mencionado en esta herramienta es la barra de búsqueda, presente en las páginas de gestión de tablas. Podría haberse construido de forma que cada vez que se utilice haga una nueva llamada al microservicio aplicando un filtro, pero en su lugar, se ha programado de forma que realice una búsqueda sobre los datos obtenidos en la llamada inicial y devuelva el resultado tras aplicar el filtro de forma dinámica.

3.3 Análisis de riesgos

La funcionalidad principal de esta herramienta es obtener información de la base de datos y mostrarla en una página web. Esta acción podría conllevar riesgos si no se planifica correctamente o no se toman precauciones.

En este TFG, se han tenido en cuenta 2 formas de realizar esta lectura en la base de datos. Una forma es mediante el uso de REST API y otra mediante el envío directo de instrucciones SQL.

El envío de instrucciones SQL se acabó considerando como una mala opción, debido a que, aunque podría reducir tiempos y costes en función de la arquitectura y tamaño del proyecto final en el que se utilice, también podría traer grandes riesgos si no se ha planificado correctamente.

Las ordenes SQL más complejas no son lo suficientemente intuitivas como para asegurar que no se leerán datos inesperados, incluso tras haberlas probado en un entorno de prueba. Esto podría deberse a un error sintáctico en la orden, o a un cambio repentino en la base de datos. Además, aunque la orden esté correctamente redactada y este perfectamente planificado el resultado esperado, siempre podrían existir errores con baja probabilidad de suceder. Por ejemplo, una alta demanda de red que, junto a una o más ordenes SQL ejecutándose simultáneamente en producción, puedan conducir a que la página web sufra una caída.

Por el contrario, el uso de REST API ofrece una forma más legible, intuitiva y controlada de diseñar los endpoints a los que se conectará la página web para recibir información. Otro punto importante a tener en cuenta es que se aumenta la flexibilidad y seguridad en el proceso de comunicación al existir una capa de abstracción entre el cliente y la base de datos. Al utilizar REST API, los clientes interactúan con la API a través de solicitudes HTTP estandarizadas (GET, POST, PUT, DELETE), lo que permite que la API pueda controlar y validar las solicitudes antes de interactuar con la base de datos.

3.4 Identificación y análisis de posibles soluciones

Lo primero que se debe planificar antes de empezar a desarrollar cualquier proyecto es su arquitectura. Esto es de vital importancia para tener una visión de objetivos clara, de forma que se puedan abordar de la mejor forma posible los requisitos funcionales y no funcionales.

Antes de empezar a desarrollar este proyecto, se estudió su arquitectura general, es decir, si debería programarse todo el código en un gran bloque, formando así un monolito, o si, por el contrario, se debería separar el proyecto en microservicios.

Después de esto, y en el caso de que se tomara la segunda opción de separar el proyecto en microservicios, también debería planificarse cómo interactuarán las diferentes partes entre ellas, qué función desarrollará cada una, y de que tamaño deberían ser.

3.4.1 Microservicios vs Monolito

3.4.1.1 Microservicios

El software basado en microservicios es una arquitectura modular y escalable compuesta por múltiples servicios diferentes. Cada microservicio se enfoca en una tarea distinta, y puede ser desarrollado, probado y desplegado de forma independiente al resto.

- Ventajas:
    - Fácil escalabilidad: Los microservicios permiten escalar de forma independiente cada uno de los componentes del software, lo que permite escalar solamente la parte o partes que fueran necesarias. De esta forma, además, se facilita la gestión de cargas de trabajo.
    - Desarrollo ágil más sencillo: Al estar el software dividido en diferentes microservicios, cada parte se puede desarrollar, probar y desplegar de forma independiente por distintos desarrolladores, mejorando así el tiempo y flexibilidad de trabajo entre los desarrolladores.
    - Mantenibilidad: Al modificar cualquier servicio, es más fácil de controlar que no que afecte a otras partes del proyecto de forma descontrolada. Esto simplifica en gran medida las tareas de depuración y hace más rápida e intuitiva la resolución de problemas.
    - Resiliencia: Debido a que cada servicio se ejecuta como un proceso independiente, en el caso de que alguno falle, y suponiendo que se haya tenido en cuenta la dependencia de servicios al diseñar la arquitectura del proyecto, los demás podrán seguir funcionando con normalidad, evitándose un efecto dominó de errores.
    - Uso de diferentes tecnologías: Ya que cada servicio puede ser tratado como un proyecto individual, resulta sencillo utilizar diferentes tecnologías o lenguajes de programación para cada uno de ellos. Gracias a esto, se aumenta la flexibilidad de cómo debería enfocarse cada servicio según las tareas que deba desarrollar. Además, esto puede facilitar la integración con tecnologías de terceros que esten limitadas a un lenguaje en concreto.
- Desventajas:
    - Complejidad de gestión: Tras la decisión de montar un sistema basado en microservicios se aumenta la complejidad de cómo gestionar el sistema. El monitoreo, coordinación y despliegue de distintos microservicios requiere de una planificación cuidadosa para garantizar que no se producirán errores.

- Planificación en la comunicación entre microservicios: Debe de existir una modalidad que permita comunicarse a los microservicios con el resto del proyecto, y esto puede aumentar la latencia y el tráfico de red. Un diseño inadecuado del sistema puede llevar a un rendimiento insuficiente e incluso a problemas de escalabilidad.
- Implementación inicial más compleja: Debido a que habrá que planificar a largo plazo con una visión de futuro, el enfoque inicial del proyecto requiere de un tiempo de planificación cuidadosa.
- Mayor complejidad en las pruebas: Al probar una nueva funcionalidad, se necesitará simular y coordinar diferentes microservicios en lugar de un gran bloque de código. Para que esto pueda hacerse de forma correcta, podría ser necesario recurrir a nuevas herramientas de software, aumentando el coste del proyecto para asegurar la calidad y fiabilidad esperada.
- Asegurar la consistencia de datos: Al estar el proyecto dividido en diferentes partes que podrían funcionar de forma muy diferente, será más difícil garantizar la consistencia de los datos al ser transferidos por el proyecto. Para asegurar esta integridad y coherencia, se deberán planificar las técnicas de sincronización, aumentando así la complejidad del diseño y desarrollo.

3.4.1.2 Monolito

En el Desarrollo de software, una estructura monolítica es una estrategia en la que la aplicación se construye como una unidad indivisible. Bajo esta estrategia, todos los componentes y funcionalidades de la aplicación están conectados entre sí y se ejecutan como un único proceso.

- Ventajas:
  - Simplicidad: La estructura monolítica permite que todo el código que conforma un proyecto se encuentre en un solo lugar, lo que facilita el desarrollo inicial de cualquier proyecto, en especial los de pequeño o medio tamaño
  - Rendimiento: Al ejecutarse como una sola unidad, no se tendrán que realizar solicitudes entre diferentes servicios, lo que suele hacer el rendimiento de la aplicación más eficiente y rápido.

- Menor complejidad de implementación y despliegue: Al tratarse de una sola unidad de código, la implementación y despliegue de un proyecto monolítico es generalmente fácil y sencilla. Esto es debido a que no se requiere de una configuración compleja del sistema a la hora de trazar la estrategia del despliegue de diferentes servicios.
- Depuración sencilla: Al estar todas las partes del proyecto en un mismo lugar, encontrar el fragmento de código concreto que está provocando un fallo es más fácil que si fuera necesario buscarlo en diferentes servicios.
- Latencia inferior: Al no existir una necesidad de comunicación entre diferentes servicios, se reduce la latencia general de toda la aplicación, además de la sobrecarga de red.

- Desventajas:
  - Escalabilidad limitada: Debido a que todos los componentes se encuentran en un mismo lugar, toda la aplicación debe escalarse como un bloque compacto, incluso cuando sea un solo componente el que esté siendo más demandado y debiera ser lo único necesario a escalar.
  - Acoplamiento fuerte: Los componentes de un proyecto monolítico están fuertemente acoplados, lo que significa que existen dependencias entre todos ellos. Esto podría dificultar el reemplazo de componentes individuales, ya que estaría afectando al sistema entero.
  - Mantenimiento complejo: A medida que el código de un proyecto crece, puede ser más difícil encontrar y corregir errores, más aún tras una refactorización del código. Además, cualquier cambio en un componente individual podría suponer un cambio en la configuración global del proyecto, lo cual es otro riesgo añadido.
  - Dificultad en el desarrollo cooperativo: El desarrollo cooperativo aumenta en complejidad por la interconexión y dependencia entre componentes. Cambios y adiciones requieren de una coordinación constante entre los desarrolladores, ya que se suelen producir conflictos y demoras. La carencia de modularidad dificulta asignar tareas de forma que cada persona trabaje en una parte sin afectar al resto.

- Dificultad en la integración de nuevas tecnologías: Adoptar nuevas tecnologías puede requerir cambios extensos en todo el sistema. Esto hace que cualquier migración sea más complicada y costosa de lo que debería, lo que limita la flexibilidad del sistema y hace más difícil mantenerse al día con los avances tecnológicos.

3.5 Solución propuesta

Esta herramienta podría funcionar de las dos formas planteadas, tanto como un monolito como con una estructura basada en microservicios. Uno de los objetivos en el desarrollo de esta herramienta es que sea escalable y flexible, por lo tanto, se ha decidido que encaja mejor una estructura basada en microservicios.

Para abordar los casos de uso de este TFG se podría haber creado un microservicio para cada tipo de entidad, es decir, uno para noticias, otro para eventos, otro para anuncios, etc. Esta decisión no cumpliría con el objetivo de que la plataforma sea lo más escalable posible, ya que, si disponemos de diferentes páginas webs que compartan una misma entidad con una estructura diferente, la lógica resultante, tanto de la logarítmica como de las interconexiones, aumentaría en complejidad y disminuiría en legibilidad.

Por este motivo, se ha decidido disponer de un microservicio para cada sitio web, el cual se encargue de gestionar las llamadas de todas las entidades de únicamente dicha página web. A nivel técnico, cada página web solo estará conectada a un endpoint y no a varios.

Esta decisión resuelve también el planteamiento sobre qué tamaño deberá tener cada microservicio. Si cada microservicio controlara una entidad existente en diferentes páginas webs, podría darse una situación en la que un microservicio se utiliza el 90% del tiempo, y el resto de los microservicios se utilizan el 10% del tiempo restante. Por ejemplo, un microservicio de noticias, las cuales son comúnmente visitadas por los usuarios, tendrá mucho más tráfico que un microservicio que se encargue de gestionar unos anuncios temporales que se visualizan un día al mes. Debido a que cada microservicio se encarga de gestionar la lógica de una página web, no será necesario dividirlos ni limitar su tamaño, ya que no se darán situaciones en las que el tráfico sea tan irregular.

4. Diseño de la solución

4.1 Arquitectura del sistema

En los casos de uso de este TFG, el dashboard proporciona una interfaz visual para monitorizar y controlar la información de ambas páginas web, además del plugin de WordPress, la cual es compartida desde 3 microservicios. Cada uno de los microservicios tiene 3 conexiones. Por un lado, se encuentran conectados a la página web o plugin donde se mostrará la información, por otro lado, al dashboard de donde la recibe, y, por último, a la base de datos donde se almacena esta información, ofreciendo datos en tiempo real reactivos a cualquier cambio.

En el siguiente diagrama se puede apreciar el mapa completo de la arquitectura de este proyecto, en el que las flechas indican la dirección de la compartición de datos:

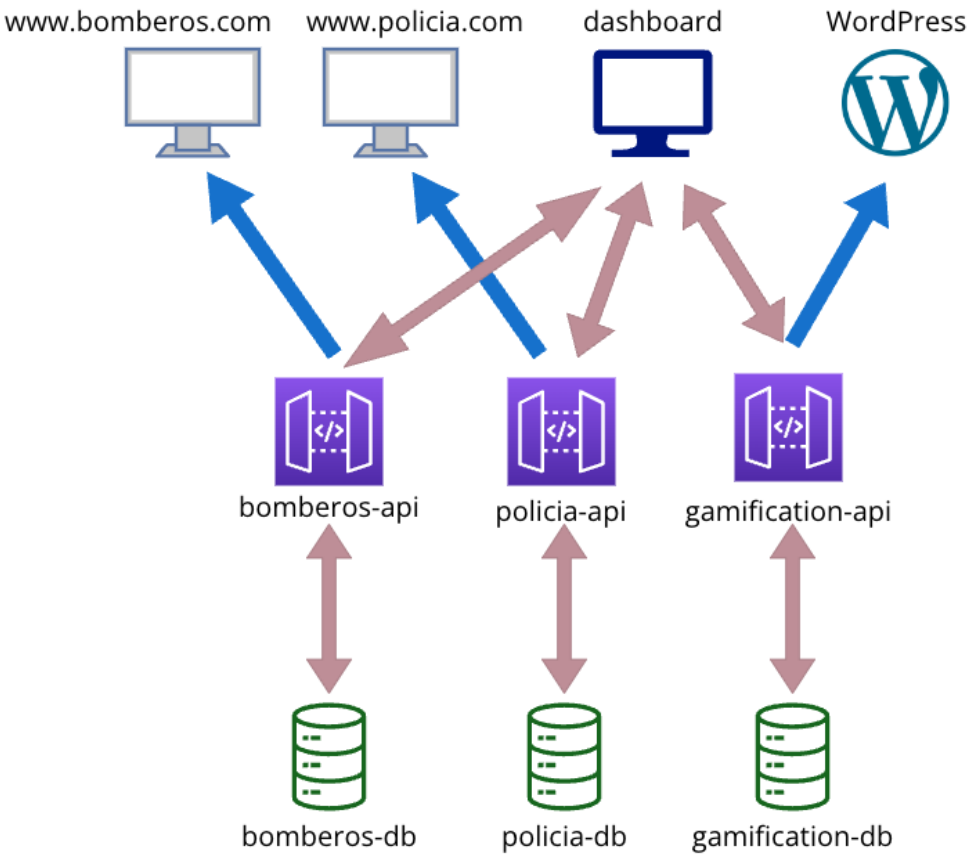


Ilustración 6 - Arquitectura completa del TFG

Sobre la anterior imagen, hay que anotar que el nombre de las bases de datos y las url de las páginas de ejemplo no son reales ya que este TFG se trata de un prototipo. En el estado actual del desarrollo del proyecto, la diferencia respecto a la arquitectura de la imagen es que se utiliza una única base de datos para almacenar todas las tablas y las páginas web y el plugin se trabajan en un entorno de prueba. Se ha decidido crear el diagrama simulando un entorno de uso real para mejorar su entendimiento.

4.2 Diseño detallado

Para desarrollar el código de este TFG, tanto el del dashboard como el de las páginas web, se ha seguido una arquitectura conocida como patrón MVC (Modelo-Vista-Controlador):

- Modelo: Su función es representar la capa de datos y la lógica de negocio de la aplicación. Su función principal es interactuar con fuentes de datos, como bases de datos o servicios externos, para realizar operaciones como consultas, actualizaciones o validaciones de datos. Además, el modelo puede contener la lógica de negocio específica de la aplicación, como cálculos, validaciones y reglas de negocio.

- Vista: Es la encargada de presentar la interfaz de usuario al usuario final. Su función principal es generar y mostrar visualmente la información de manera adecuada y comprensible. Utiliza plantillas y componentes visuales para representar los datos proporcionados por el controlador. Además, la vista puede manejar interacciones del usuario, como capturar datos o enviar formularios.

- Controlador: Se encarga de recibir las solicitudes del cliente y coordinar las acciones necesarias en el sistema. Su función principal es manejar el flujo de control y comunicarse con el modelo para obtener los datos necesarios. Además, el controlador selecciona la vista adecuada y le proporciona los datos para su presentación al usuario.

En el desarrollo de este proyecto, una de las normalizaciones para preservar el orden ha sido el nombre de los archivos y su ubicación. Se ha puesto especial atención en que los modelos tengan el nombre de la entidad con cuya base de datos interaccionan, las vistas se encuentren en carpetas con el nombre de la entidad y los controladores tengan el nombre y métodos únicamente de una entidad. El objetivo final es que un desarrollador totalmente ajeno al proyecto sea capaz de entender fácilmente su estructura en poco tiempo, además de que no resulte desordenado el añadir grandes cantidades de ficheros en el caso de que el proyecto creciera.

A continuación, se detallará mediante tablas todas las entidades que existen en el estado actual del proyecto y sus atributos, junto con un título que indica con qué partes del TFG están relacionadas:

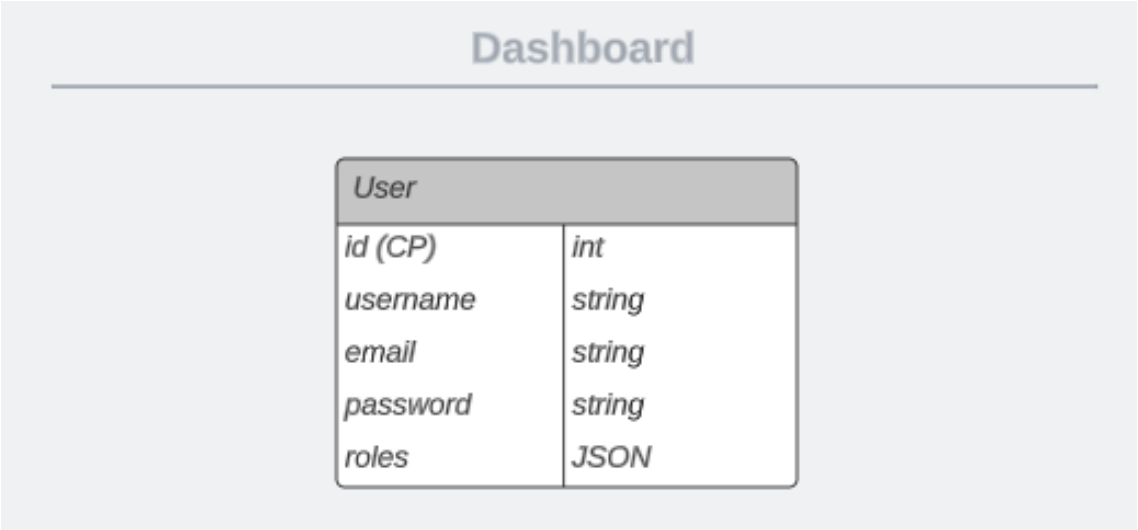


Ilustración 7 - Entidades relacionadas con el dashboard

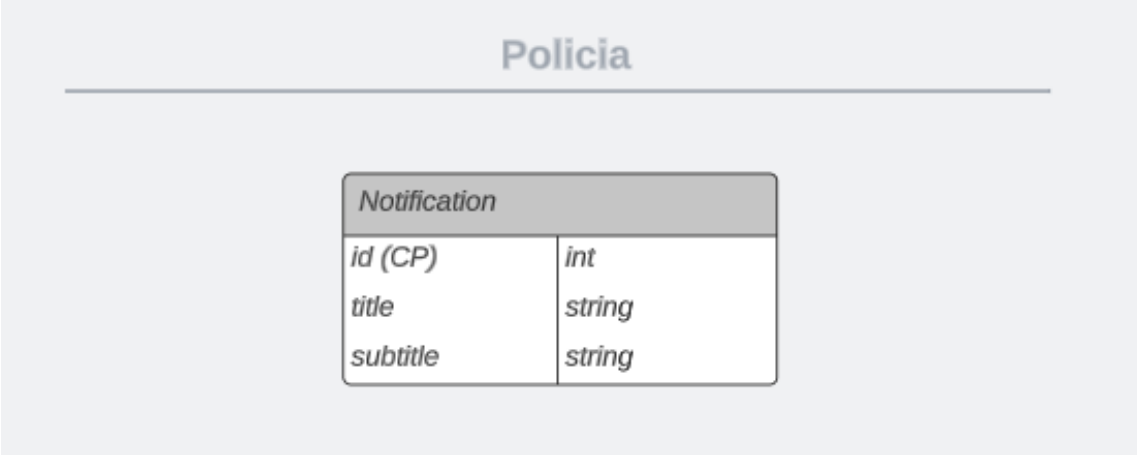


Ilustración 8 - Entidades relacionadas con la página web de la policia

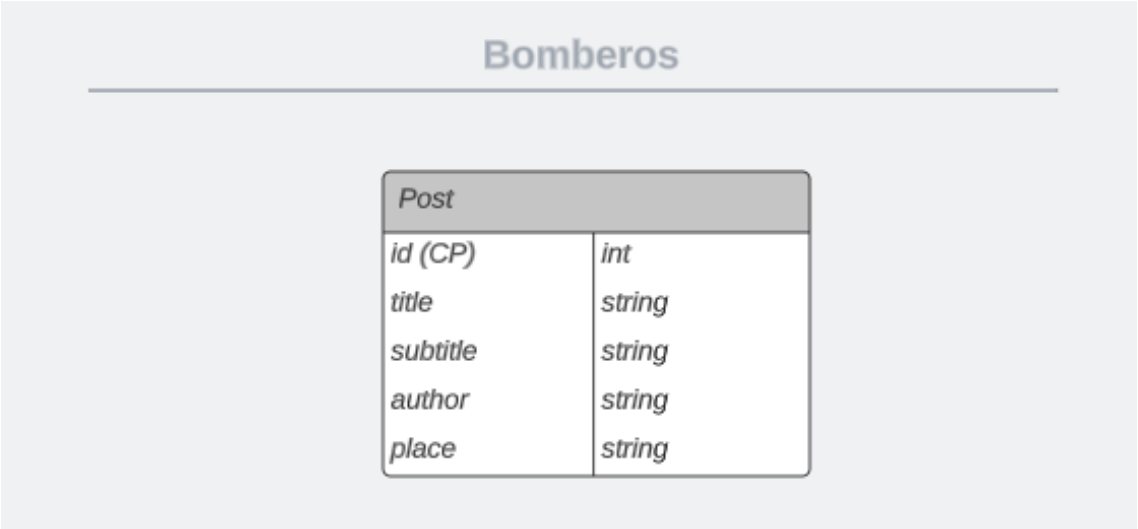


Ilustración 9 - Entidades relacionadas con la página web de los bomberos

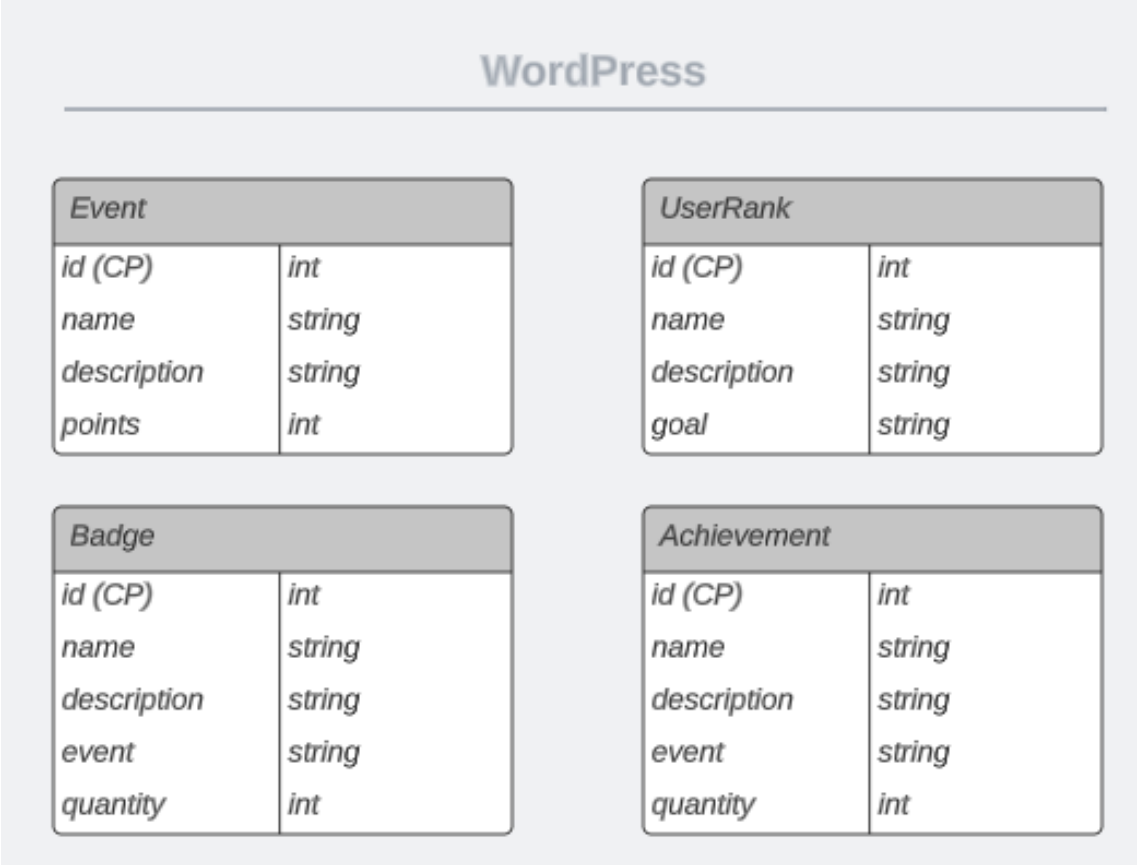


Ilustración 10 - Entidades relacionadas con el plugin de WordPress

4.3 Tecnología utilizada

Para realizar este TFG se ha utilizado una variedad de tecnologías. Algunas han servido para mejorar la gestión de tareas, mientras que otras han estado totalmente enfocadas en la parte técnica, como han sido los lenguajes de programación y framework escogidos.

A la hora de tomar la decisión de escoger el listado de tecnologías que podrían ser útiles, las posibilidades son infinitas. Como buena práctica, se han descartado varias tecnologías que, aunque podrían haber sido útiles, acabarían siendo redundantes por poder obtener el mismo valor mediante otra tecnología ya presente en el proyecto.

Ilustración 11 - Logo de Visual Studio Code

Visual Studio Code[7]

Es un entorno de desarrollo integrado (IDE) flexible y eficiente desarrollado por Microsoft. Con una interfaz intuitiva y personalizable, ofrece características como resaltado de sintaxis, autoc ompletado de código y depuración integrada. Ha sido elegido por ser un IDE de propósito general y por las grandes posibilidades de configuración que ofrece mediante el uso de extensiones.

Ilustración 12 - Logo de Bitbucket

Bitbucket[8]

Una plataforma de alojamiento y gestión de repositorios de código desarrollado por Atlassian, que ofrece características como control de versiones con Git y Mercurial, seguimiento de problemas, integración de CI/CD y herramientas de revisión de código. Bitbucket ha sido una herramienta clave a lo largo del desarrollo de este TFG permitiendo guardar código en diferentes ramas, fusionarlo o revertirlo. En varias situaciones en las que entraba en un bucle de errores me ha permitido volver a una versión anterior, ahorrándome mucho tiempo.

Ilustración 13 - Logo de Jira

Jira

Una plataforma líder de gestión de proyectos y seguimiento de problemas desarrollada por Atlassian. Proporciona un conjunto completo de herramientas para planificar, rastrear y colaborar en proyectos de software. Ha sido la herramienta clave en la planificación de tareas y metodología del proyecto. Las características más útiles han resultado ser, por un lado, la posibilidad de crear un tablero de sprints para organizar la semana, y por otro, la matriz de tareas pendientes con sus respectivas descripciones.

Ilustración 14 - Logo de Git

Git[9]

Software de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en proyectos de software. Permite a los desarrolladores trabajar en colaboración, realizar un seguimiento de los cambios, crear ramas para el desarrollo paralelo y fusionar los cambios de forma eficiente. Los comandos de Git me han permitido controlar las versiones del proyecto y ha sido la forma de interactuar con Bitbucket. En cualquier proyecto de software debería ser imperativo el uso de alguna tecnología como Git en lugar de realizar todo el desarrollo en una misma versión.

Ilustración 15 - Logo de PHP

PHP[10]

Lenguaje de programación ampliamente utilizado en el desarrollo web. Es conocido por su facilidad de uso y su capacidad para crear sitios web interactivos. Con PHP, es posible escribir código que se ejecuta en el servidor y genera contenido dinámico en una página web. PHP ha sido utilizado en el desarrollo del plugin de WordPress, uno de los casos de uso de este TFG, debido que el propio WordPress está desarrollado en PHP. Además, se trata de un lenguaje muy potente en el desarrollo web con la capacidad de permitir cualquier funcionalidad al ser combinado con alguno de sus frameworks.

Symfony[11]

Ilustración 16 - Logo de Symfony

Framework de PHP de desarrollo web altamente reconocido y utilizado en la comunidad de desarrolladores. Es conocido por su enfoque en la eficiencia y la estructura en la construcción de aplicaciones web. Permite crear proyectos web de manera organizada y escalable, gracias a su arquitectura robusta y modular. Symfony ha sido elegido para realizar la mayoría del código de este TFG por tratarse de un framework enorme con el cual es posible diseñar prácticamente cualquier funcionalidad. Además, dispone de una muy buena documentación, e incluso libros, por lo que ha sido una buena elección para un desarrollador con poca experiencia.

Api Platform[13]

Ilustración 17 - Logo de Api Platform

Framework de Symfony y potente herramienta de desarrollo y gestión de APIs que brinda a las organizaciones la capacidad de crear, documentar, publicar y controlar de manera eficiente sus interfaces de programación de aplicaciones. Ha sido elegido para diseñar los microservicios por su facilidad para implementar REST API de forma completa, incluyendo la definición de rutas, la serialización de datos y la gestión de operaciones CRUD de manera sencilla y robusta.

Composer[14]

Ilustración 18 - Logo de Composer

Herramienta de administración de dependencias para proyectos PHP. Permite definir y gestionar fácilmente las bibliotecas y paquetes externos requeridos por un proyecto a través de un archivo "composer.json". Ha sido utilizado para ejecutar todos los comandos de creación automática de archivos, además de para la instalación de dependencias. Debido a la documentación clara y completa de Composer ha sido posible resolver conflictos entre versiones de forma sencilla por disponer de una información clara sobre qué versión es compatible con cual.

WordPress

Ilustración 19 - Logo de WordPress

Sistema de gestión de contenidos (CMS) gratuito y de código abierto para crear y administrar sitios web fácilmente. Debido al desarrollo de plugins, existe una comunidad de programadores que se encuentra en contacto con WordPress, aunque no lo utilicen activamente como CMS. Ya que se trata de una herramienta muy utilizada por las empresas en la actualidad, se ha decidido incorporarla en el TFG como caso de uso.

MySQLWorkbench[15]

Ilustración 20 - Logo de MySQLWorkbench

Herramienta de modelado, desarrollo y administración de bases de datos MySQL que ofrece una interfaz intuitiva y completa para los profesionales de bases de datos. Ha sido la herramienta principal del nivel de las bases de datos de este TFG. Se ha utilizado para almacenar objetos en ella, además de para realizar pruebas provocando cambios en los objetos guardados en ella debido a su facilidad de uso y a su interfaz intuitiva. Ha sido útil también por las facilidades que ofrece al querer conectarla a un proyecto de software mediante endpoints.

5. Propuesta de solución

Para desarrollar este TFG se han seguido los estándares arquitectónicos de los proyectos desarrollados en Symfony. Además, y como buena práctica, se ha intentado utilizar siempre que sea posible comandos en la terminal de Visual Studio Code para generar los ficheros, en lugar de crearlos a mano o copiar ficheros ya existentes. Esto se ha hecho para evitar errores humanos como fallos sintácticos, además de para agilizar al máximo el proceso y evitar problemas de configuración en un futuro. La ventaja de utilizar la terminal de Visual Studio Code es que su ruta por defecto es la raíz del proyecto, de forma que muy pocas veces habrá que utilizar comandos para movernos entre rutas.

A continuación, se habla sobre cómo se ha desarrollado cada parte del TFG, utilizando para ello capturas de pantalla de algunos fragmentos de código y comandos utilizados.

5.1 Dashboard

En este apartado se explica cómo se ha desarrollado la pieza principal de este TFG, el dashboard. Otros proyectos individuales que conforman este TFG tienen una arquitectura muy similar, por lo que la explicación más detallada sobre la configuración inicial y arquitectura se hará utilizando el dashboard como ejemplo, ya que también es el proyecto más complejo. Se ha decidido hacerlo así para evitar la repetición, y en los apartados sobre otros proyectos individuales solo se hablará de los puntos más importantes y distintos a lo que ya haya sido explicado.

5.1.1 Configuración inicial

El dashboard ha sido la primera pieza del TFG en ser desarrollada. El primer paso ha sido ejecutar el comando de Composer que permite crear el esqueleto de un proyecto Symfony de forma automática:

```
rafa@MacBook-Pro-de-Rafael TFG % composer create-project symfony/skeleton dashboard-web
```

Ilustración 21 - Comando para crear el esqueleto de un proyecto mediante Composer

Tras su ejecución, se habrá generado una configuración básica con todos los ficheros necesarios para empezar a desarrollar cualquier proyecto:

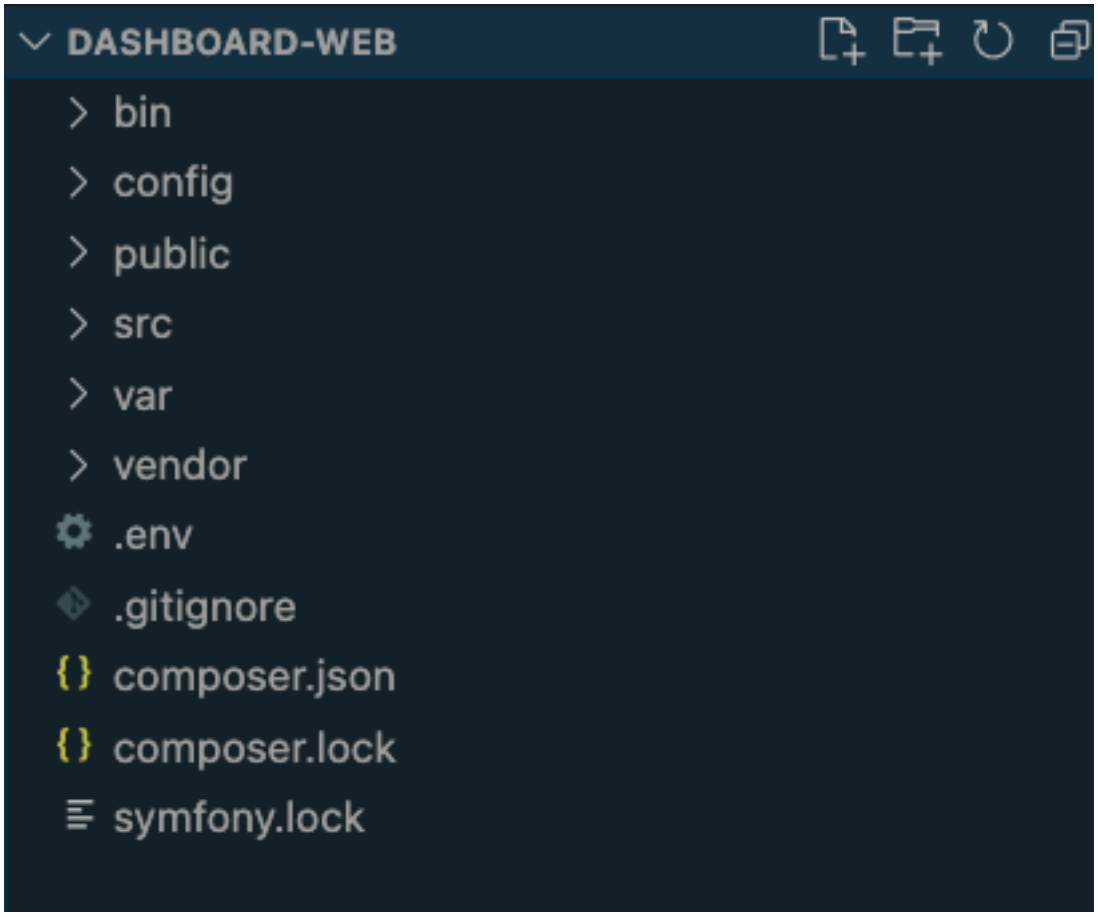


Ilustración 22 - Estructura inicial del dashboard

A continuación, se configurarán los ficheros de entorno, los cuales son ".env" para producción y ".env.local" para trabajar en local. En el desarrollo de este TFG, la única variable que se ha utilizado ha sido la ruta de cada microservicio:

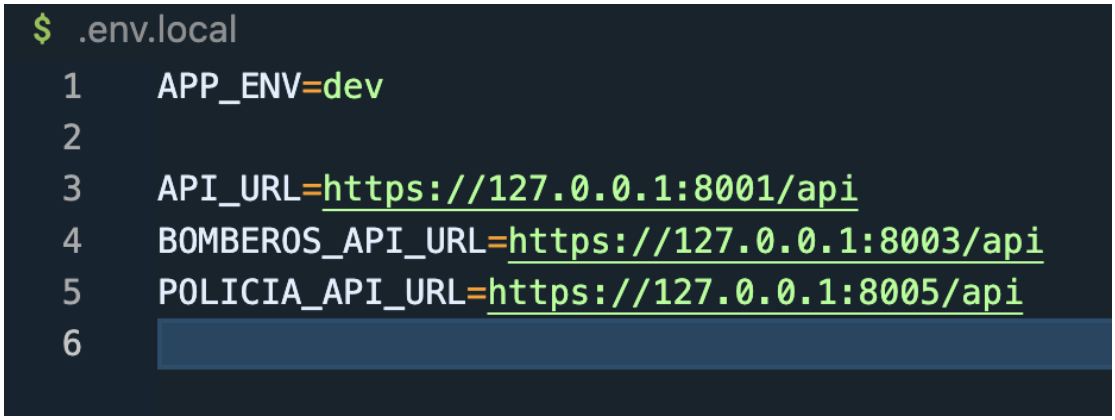


Ilustración 23 - Archivo del entorno de desarrollo del dashboard

Para poder utilizar algunas funcionalidades a lo largo del desarrollo, al igual que algunos comandos esenciales, se debe modificar el archivo composer.json. Este archivo se encuentra ubicado en la raíz del proyecto, y es utilizado en proyectos PHP como una forma de administrar dependencias entre versiones. La forma de utilizarlo es mediante comandos, los cuales añadirán bibliotecas y paquetes en su contenido, o escribiéndolas a mano. En este TFG, como ya se ha explicado, se ha decidido utilizar los comandos de la terminal siempre que sea posible. Un ejemplo de su uso es el siguiente comando “composer require (nombre del paquete) --(nombre del entorno)”:

```
rafa@MacBook-Pro-de-Rafael dashboard-web % composer require symfony/maker-bundle -dev
```

Ilustración 24 - Comando para instalar la biblioteca "symfony/maker-bundle" mediante Composer

Al ejecutarlo, se añadirá el paquete “symfony/maker-bundle” al entorno de desarrollo junto con una versión, la cual puede ser modificada en cualquier momento siempre y cuando se respeten las dependencias con el resto de paquetes. En la siguiente imagen, se puede apreciar que vamos a utilizar la versión 1.43 y sus posteriores con el icono “^”.

```
30      "require-dev": {
31          "symfony/debug-bundle": "5.4.*",
32          "symfony/maker-bundle": "^1.43",
33          "symfony/web-profiler-bundle": "^5.4"
34      },
```

Ilustración 25 - Sección de código del archivo "composer.json" del dashboard con bibliotecas de uso local

Este paquete en específico es uno de los más importantes en este proyecto, dado que permite utilizar los comandos “make:”, los cuales se verán más adelante.

Otra biblioteca importante utilizada en este proyecto es Encore[16] de Symfony, la cual se utiliza para simplificar la gestión de activos front-end y aumentar la reutilización de código. Para instalarla en el proyecto, se utilizará el siguiente comando:

```
rafa@MacBook-Pro-de-Rafael dashboard-web % composer require symfony/webpack-encore-bundle
```

Ilustración 26 - Comando para instalar la biclioteca "symfony/webpack-encore-bundle" mediante Composer

Este comando añadirá el archivo “webpack.config.js”, en el cual se especificarán los puntos de entrada de los archivos JavaScript y CSS del proyecto y podrán ser importados mediante “entrys”. A continuación, se muestra una captura de pantalla del aspecto de dicha sección dentro del archivo “webpack.config.js”:

```
.addEntry('app', './assets/app.js')
```

Ilustración 27 - Punto de entrada en el archivo webpack mediante addEntry()

Para utilizar esta Entry llamada “app”, la cual carga el contenido del archivo “app.js”, se debe importar mediante un método específico de Encore especializado en ello. A continuación se muestra una imagen donde se puede apreciar cómo se utiliza dicha entry dentro del bloque llamado “javascripts” dentro del archivo “base.html.twig”, el cual, como su nombre indica y siguiendo los estándares de Symfony, se encarga de renderizar la base del front-end del proyecto:

```
{% block javascripts %}
    {{ encore_entry_script_tags('app') }}
{% endblock %}
```

Ilustración 28 - Ejemplo de cómo importar una Entry en un bloque

Una vez el proyecto ha sido configurado agregando todas las bibliotecas y paquetes necesarios, se debe compilar para guardar la configuración. Para ello, lo primero será ejecutar el siguiente comando:

```
rafa@MacBook-Pro-de-Rafael dashboard-web % composer install
```

Ilustración 29 - Comando para instalar o reconfigurar Composer

Al hacerlo, Composer lee el archivo "composer.json" en el directorio raíz del proyecto y descarga automáticamente las dependencias especificadas. Esta nueva configuración quedará reflejada en el fichero “composer.lock”.

De igual forma, se debe ejecutar la siguiente orden:

```
rafa@MacBook-Pro-de-Rafael dashboard-web % yarn install
```

Ilustración 30 - Comando para instalar Yarn

Tras este comando, Yarn[17] leerá las dependencias de los paquetes del proyecto y generará una nueva configuración en el archivo “yarn.lock”.

Algo importante a tener en cuenta sobre Yarn es que, al realizar cualquier modificación sobre los archivos cargados por Encore, se debe realizar una compilación con el siguiente comando:

```
rafa@MacBook-Pro-de-Rafael dashboard-web % yarn encore dev
```

Ilustración 31 - Comando para compilar Yarn

Tras ejecutar este comando es cuando se verán los cambios reflejados en el front-end del proyecto.

5.1.2 Arquitectura del proyecto

Una vez la configuración del proyecto ha sido completada, este es el aspecto que tendrá su estructura:

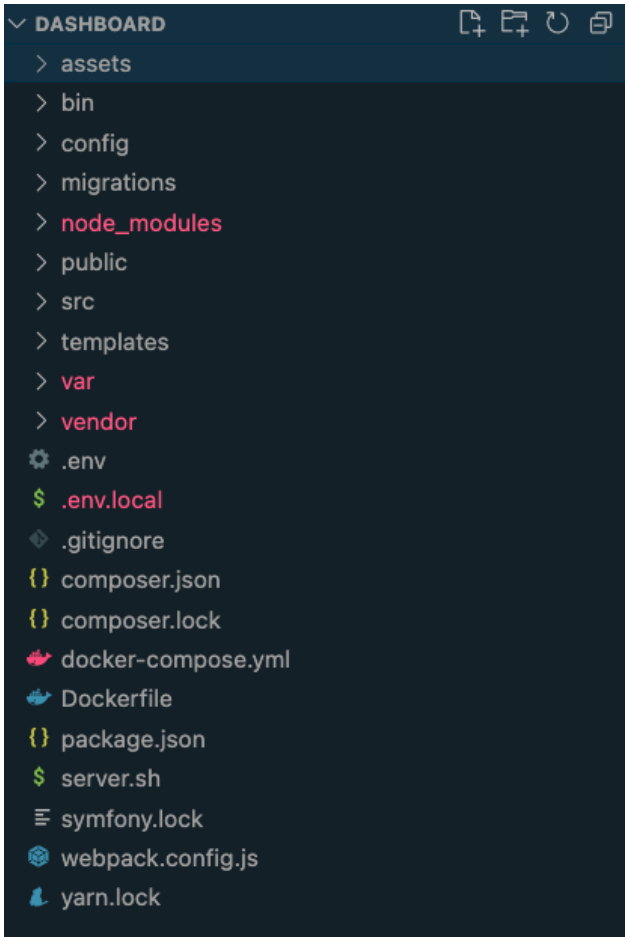


Ilustración 32 - Estructura de carpetas final del dashboard

A continuación, se verá de forma detallada la distribución de carpetas y ficheros más importantes del proyecto. En primer lugar, visitaremos la carpeta “public”, en la que se encuentra el archivo “index.php”, punto inicial del proyecto, junto con todos los archivos que se utilizaran en la interacción con internet. Entre ellos se pueden ver las imágenes utilizadas en el proyecto, además de los ficheros de estilo CSS y de lógica JavaScript:

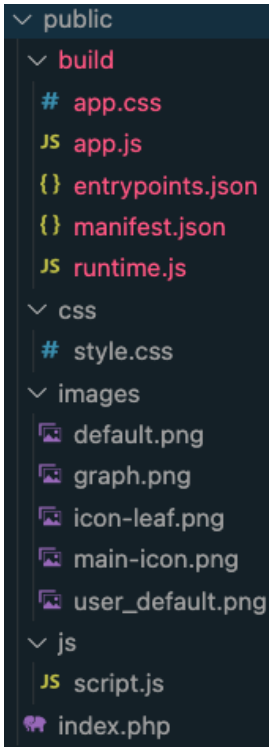


Ilustración 33 - Contenido de la carpeta "public" del dashboard

Veremos ahora el contenido de la carpeta “src”. En ella se encuentran diferentes carpetas en las que se encuentran los archivos PHP encargados de controlar la lógica del proyecto. Estos archivos serán explicados con detalle más adelante:

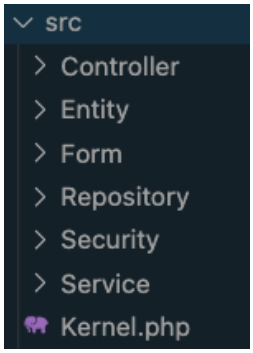


Ilustración 34 - Contenido de la carpeta "src" del dashboard

Dentro de la carpeta “templates” se encuentran los archivos twig encargados de renderizar las vistas del proyecto:

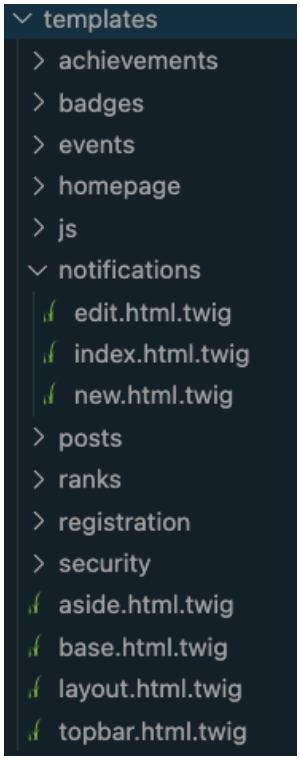


Ilustración 35 - Contenido de la carpeta "templates" del dashboard

Para preservar el orden en el proyecto, se ha creado a su vez una carpeta para cada entidad. Como se puede observar en la carpeta “notifications” desplegada en la imagen, en su interior se encuentran los 3 archivos encargados de renderizar las 3 vistas que podrá utilizar el usuario para realizar operaciones CRUD sobre la entidad Notification.

Por otro lado, se pueden observar 4 archivos que no pertenecen a ninguna entidad, sino que son comunes a toda la aplicación:

- base.html.twig: Este archivo define la estructura y los elementos comunes de todas las páginas de la aplicación, como la estructura HTML básica, el encabezado, el pie de página y otros elementos repetidos. La plantilla base se extiende en las plantillas individuales para cada página, permitiendo una estructura y diseño ordenados en todo el proyecto.

- layout.html.twig: Hace una función bastante similar a la del archivo “base.html.twig”, pero en lugar de afectar a todo el proyecto, se utiliza como plantilla base de una sección concreta del proyecto en la que existan elementos repetidos.
- aside.html.twig y topbar.html.twig: Se encargan de renderizar, respectivamente, el menú lateral y la barra superior de la aplicación. Se podrían haber escrito directamente en el archivo “layout.html.twig”, pero, como buena práctica en el desarrollo de aplicaciones Symfony, se ha decidido separarlas e incluirlas en el layout como se puede ver en la siguiente imagen, en la cual se puede ver el contenido completo del archivo “layout.html.twig”:

Ilustración 36 - Contenido del archivo "layout.html.twig" del dashboard

En la imagen se puede apreciar también como se han utilizado las capas de Symfony mediante los métodos “extend” e “include”, los cuales ofrecen facilidades en la reutilización de código.

Por último, se procederá a explicar el contenido de la carpeta “vendor”:

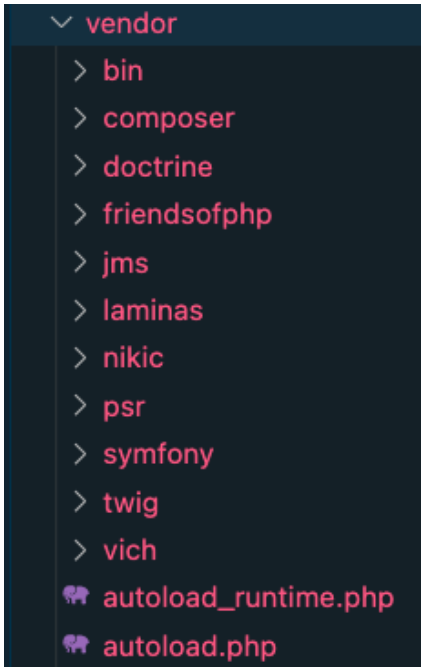


Ilustración 37 - Contenido de la carpeta "vendor" del dashboard

Su contenido es autogenerado por Composer al ejecutar el comando “composer install”. Se trata de todas las dependencias de los paquetes encontrados en el archivo “composer.json”, y se guardan automáticamente en sus propias subcarpetas.

Es importante añadir que esta carpeta no se debería incluir en el control de versiones, ya que se espera que estas dependencias sean instaladas automáticamente en cada entorno de desarrollo.

El uso de la carpeta “vendor” permite una separación clara entre el código del proyecto y las librerías externas.

5.1.3 Formularios

Todos los formularios de la aplicación están desarrollados mediante los Form[18] Types de Symfony. Se trata de clases predefinidas que definen cómo se representan y validan los campos en un formulario y brindan una manera sencilla de construir formularios flexibles y modulares.

Dentro del proyecto, se encuentran en la ruta de carpetas “src/Form”:

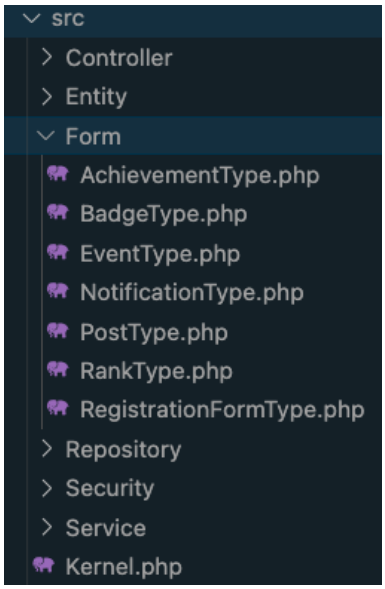


Ilustración 38 - Vista de los formularios presentes en el dashboard

A modo de ejemplo, veremos cómo se ha construido el formulario de los avisos, los cuales serán mostrados en la página web de ejemplo de la policía local de Valencia:

```
You, 2 weeks ago | 1 author (You)
class NotificationType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options): void
    {
        $builder
            ->add('title', TextType::class, [
                'label' => 'Título',
                'required' => true,
                'attr' => [
                    'class' => 'form-group'
                ]
            ])
            ->add('subtitle', TextareaType::class, [
                'label' => 'Subtítulo',
                'required' => false,
                'attr' => [
                    'class' => 'form-group'
                ]
            ])
        ;
    }
```

Ilustración 39 - Fragmento de código del formulario de la entidad Notification

Como se puede apreciar, cada campo del formulario es añadido con el método add() al atributo $builder dentro de la clase principal del archivo “NotificationType.php”. En el ejemplo, tan solo se han utilizado las 3 propiedades básicas que debería tener todo campo en este tipo de formularios:

- label: Título del campo que se mostrará en la interfaz de la web.
- required: Propiedad booleana que controla si el campo es o no obligatorio.
- attr: Conjunto de atributos que se le dará al campo una vez sea renderizado. En la imagen, el atributo “class” añadirá la clase deseada al código HTML del campo.

Una vez el archivo Type está configurado, habrá que escribir el código necesario para renderizarlo. A continuación, se presenta el fragmento de código del archivo “templates/notifications/new.html.twig” que prepara la vista del formulario:

```
{% block bodyContent %}
    <div class="main-table">
        <div class="table">
            <div class="container">
                <h2 class="mb-4">Nuevo Aviso</h2>
                <div class="row justify-content-center mt-5">
                    <div class="col-md-6">

                        {{ form_start(form) }}
                        {{ form_errors(form) }}

                        <div class="form-group">
                            {{ form_label(form.title) }}
                            {{ form_widget(form.title) }}
                        </div>/.form-group

                        <div class="form-group">
                            {{ form_label(form.subtitle) }}
                            {{ form_widget(form.subtitle) }}
                        </div>/.form-group

                        <button type="submit" class="btn btn-primary">Guardar</button>

                        {{ form_end(form) }}

                    </div>/.col-md-6
                </div>/.row.justify-content-center.mt-5
            </div>/.container
        </div>/.table
    </div>/.main-table
{% endblock %}
```

Ilustración 40 - Fragmento de código que renderiza el formulario de creación de un objeto Notification

En la imagen se pueden observar algunos métodos relacionados con el control de la información del formulario, explicados a continuación:

- form_start y form_end: Indican el inicio y el final de la lógica del formulario.
- form_errors: En el caso de que se produzca algún error al intentar enviar el formulario, mostrará un mensaje de error.
- form_label: Renderiza la label, o título, de dicho campo del formulario, el cual se ha especificado en el archivo Type.
- form_widget: Renderiza el campo del formulario, el cual el usuario podrá rellenar con la información que quiera enviar.

Una vez todo el código ha sido preparado veremos cuál es su aspecto final. En la siguiente imagen se puede ver la interfaz del formulario donde el usuario podrá crear un nuevo aviso:

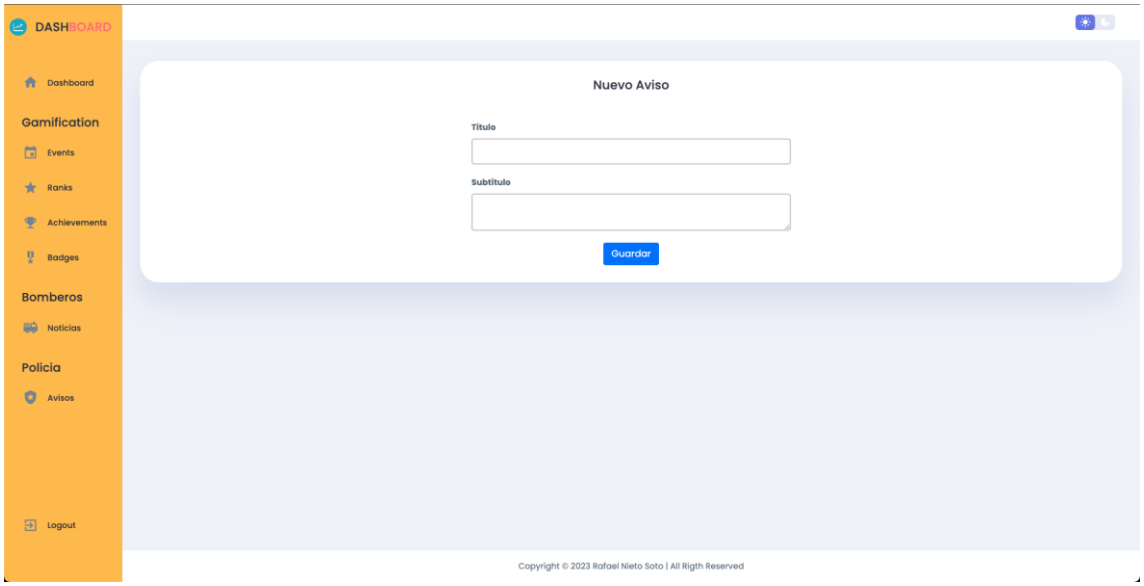


Ilustración 41 - Interfaz web del formulario de creación de un objeto Notification

Se trata de un formulario donde el usuario puede rellenar los dos campos propios de los dos atributos de un objeto Aviso y enviarlo.

5.1.4 Login

La aplicación dispone también de una página de login. Su código se encuentra en el archivo “templates/security/login.html.twig”. A continuación, se muestra el aspecto de la lógica del código:

```
{% block body %}
    <div class="background">
        <div class="shape"></div>
        <div class="shape"></div>
    </div>/.background
    <form method="post">

        {% if app.user %}
            <div class="mb-3">
                Ya has iniciado sesión como
                {{ app.user.userIdentifier }},
                <a href="{{ path('app_logout') }}">Logout</a>
            </div>/.mb-3
        {% endif %}

        <h1 class="h3 mb-3 font-weight-normal">Login de usuario</h1>

        <label for="inputEmail">Email</label>
        <input placeholder="Email" type="email" value="{{ last_username }}" name="email" id="inputEmail" class="form-control" autocomplete="email" required autofocus>
        <label for="inputPassword">Contraseña</label>
        <input placeholder="Contraseña" type="password" name="password" id="inputPassword" class="form-control" autocomplete="current-password" required>

        <input
        type="hidden" name="_csrf_token" value="{{ csrf_token('authenticate') }}">

        {% if error %}
            <div class="alert alert-danger">{{ error.messageKey|trans(error.messageData, 'security') }}</div>
        {% endif %}
        <button class="btn btn-lg btn-primary" type="submit">
            Iniar sesión
        </button>/.btn.btn-lg.btn-primary
    </form>
{% endblock %}
```

Ilustración 42 - Fragmento de código que renderiza el formulario de inicio de sesión

Como se puede apreciar, tiene dos condicionales “if” para controlar el funcionamiento de dicha página.

El primero de ellos muestra un mensaje informando al usuario si ya tiene un usuario iniciado sesión en ese momento, utilizando la propiedad “app.user” de Symfony para realizar la comprobación:

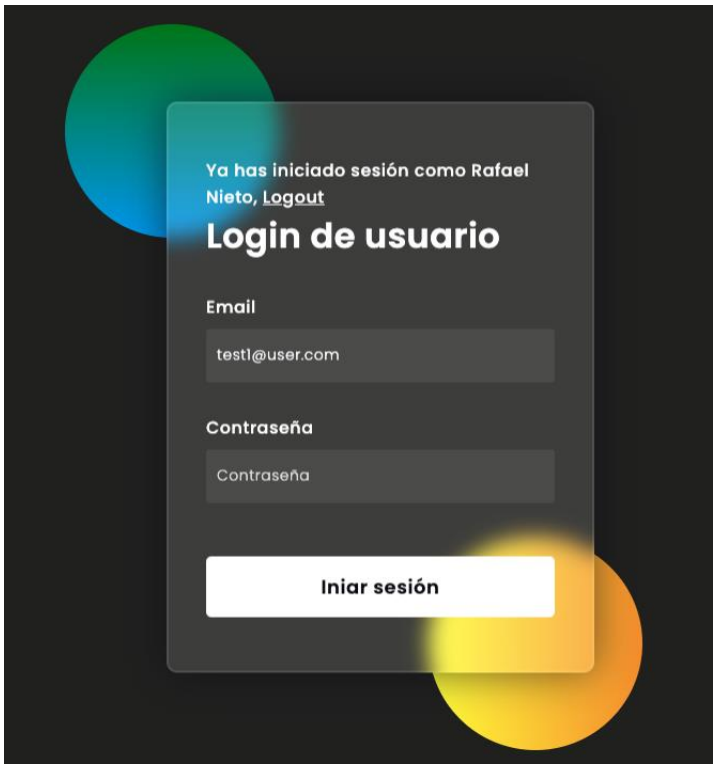


Ilustración 43 - Interfaz web del formulario de inicio de sesión con una sesión iniciada

El segundo condicional “if” comprueba si ha habido algún problema durante el inicio de sesión, mostrando un mensaje de error:

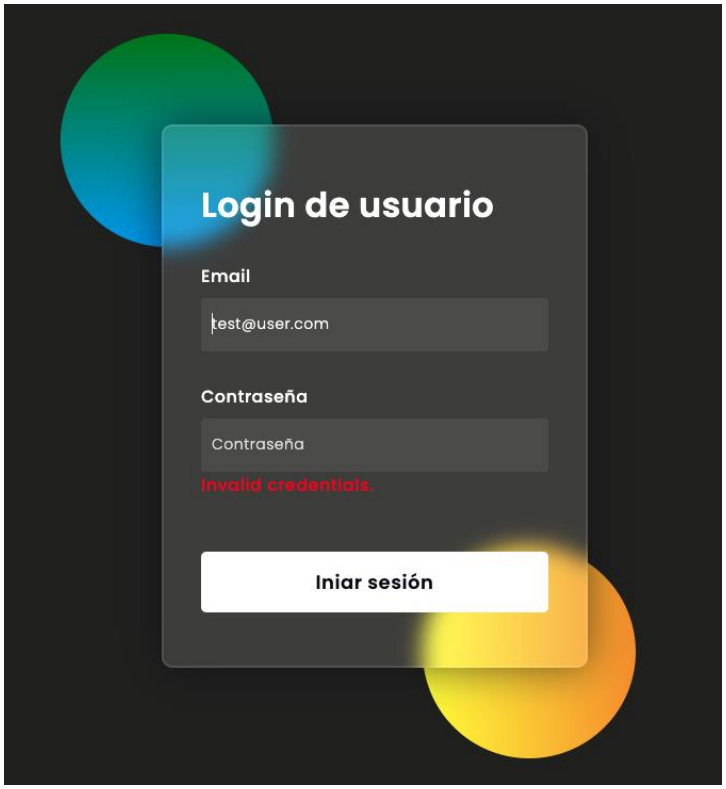


Ilustración 44 - Interfaz web del formulario de inicio de sesión con credenciales inválidas

Para realizar el front-end de esta interfaz de login se ha obtenido el código HTML y CSS de una página web de código libre[19]. A continuación, se han realizado unas ligeras modificaciones sobre su código CSS y se ha estructurado su código HTML en el archivo que construye su vista, compatibilizándolo con el lenguaje twig y la estructura de bloques de Symfony.

5.1.5 Página principal y gestión de rutas

La primera página que visualiza el usuario tras hacer login en el dashboard, es una página principal con un listado de botones, desde la cual podrá viajar a cualquier otra sección:

Ilustración 45 - Interfaz web de la página principal del dashboard

Cada uno de estos botones utiliza el método “path” de Symfony para construir la ruta a dicha página, de la siguiente forma:

```
<a href="{{ path('events_index') }}" class="btn btn-primary btn-index mb-3">Events</a>
```

Ilustración 46 - Fragmento de código utilizando el método path()

Como se puede ver, el atributo href no está asignado a una url, sino a una ruta. Dicha ruta hace referencia a la ruta del método de un controlador, en la cual comenzará el renderizado de la nueva página.

En la línea de la imagen de ejemplo, la ruta “events_index” se crea en el archivo “EventController.php”, en el método index() siguiendo los estándares de Symfony. En la siguiente imagen se puede observar cómo se crea dicha ruta mediante la etiqueta “@Route()”:

Ilustración 47 - Ruta lógica "events_index" en el dashboard

Como aclaración, en la imagen se puede observar que la ruta de este método está dividida en dos, la primera mitad en la clase principal, y la segunda mitad en el método. Se ha decidido hacerlo de esta forma como una buena práctica, debido a que, si esta clase tuviera 10 métodos, estaríamos repitiendo el contenido de la etiqueta de la clase principal 10 veces. Esto es una buena forma de cumplir con uno de los objetivos del desarrollo de este proyecto, el cual se basa en la reutilización del código.

Sobre la etiqueta “@Route()”, se pueden observar dos propiedades contenidas en ella. La primera parte, en el caso del ejemplo “events/”, creará un slug en la url de la página web. La propiedad “name” de la etiqueta, como ya se ha explicado, creará el nombre de la ruta dentro de la lógica del proyecto, la cual podrá ser invocada desde otras secciones.

5.1.6 Tabla de gestión

En el estado actual de la herramienta, la página de mayor interés es la tabla de gestión, presente en todas las entidades. Desde dicha página el usuario es capaz de realizar operaciones CRUD sobre la entidad en la que nos encontremos.

A continuación, se muestra una imagen de la tabla de gestión de la entidad Aviso, poblada con un objeto Aviso:

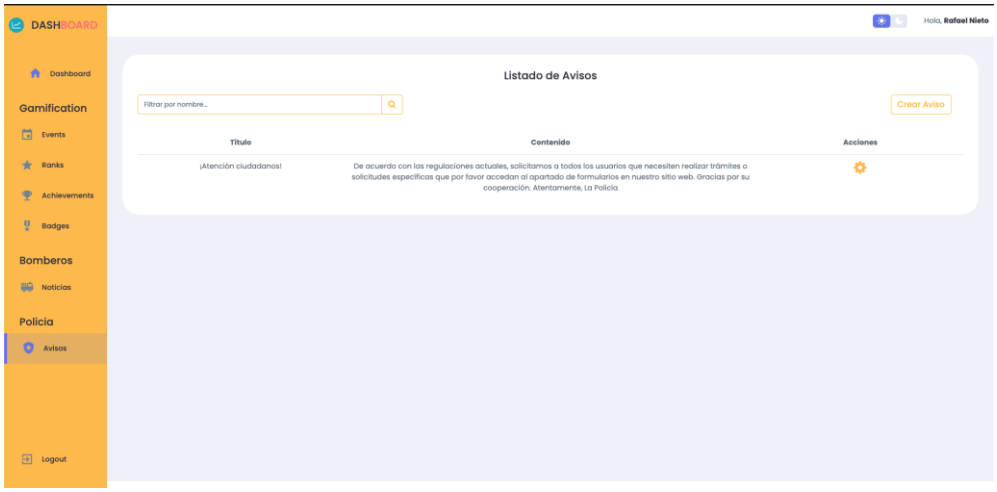


Ilustración 48 - Tabla de gestión de la entidad Notification

Se puede observar que el contenido principal es una tabla con todos los objetos leídos de la base de datos de la entidad Aviso, cuyas columnas corresponden a algunas de las propiedades de los objetos, lo cual puede ser modificado para mostrar únicamente las columnas de interés. En la parte superior derecha se encuentra un botón “Crear Aviso” cuyo propósito es llevar al usuario a un formulario en el que podrá crear un nuevo Aviso.

Al pulsar en el engranaje de opciones de una fila de la tabla, se desplegarán 2 nuevas opciones. La opción de eliminar permitirá al usuario eliminar el objeto seleccionado, por otro lado, la opción de editar llevará al usuario a un formulario en el que podrá editar las propiedades del objeto seleccionado:

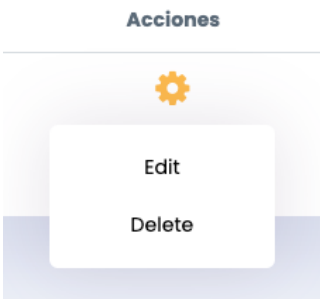


Ilustración 49 - Engranaje de opciones en la tabla de gestión

Otro elemento que puede observarse es un buscador de texto en la parte superior izquierda. La función que cumple es la que podría esperarse de un elemento como tal, la de aplicar un filtro en los objetos de la tabla para mostrar únicamente los que cumplan con el texto clave, o un mensaje “No hay resultados” en el caso de que ningún objeto cumpla con el filtro.

En la siguiente imagen se puede observar el código que permite su funcionamiento:

```
const searchInput = document.querySelector('#search-input');
const table = document.querySelector('table');
const tbody = document.querySelector('tbody');
const emptyListMessage = document.querySelector('.empty-list');

let message = document.querySelector('.no-results');

if (! message) {
message = document.createElement('p');
message.classList.add('no-results', 'text-center', 'font-weight-bold', 'mt-4', 'mb-4');
message.textContent = 'No hay resultados';
}

searchInput.addEventListener('input', () => {
const searchTerm = searchInput.value.trim().toLowerCase();
const rows = document.querySelectorAll('tbody tr');

let count = 0;
rows.forEach((row) => {
const name = row.querySelector('td:first-child').textContent.trim().toLowerCase();
const description = row.querySelector('td:nth-child(2)').textContent.trim().toLowerCase();

if (name.includes(searchTerm) || description.includes(searchTerm)) {
row.style.display = '';
count++;
} else {
row.style.display = 'none';
}
});

if (count === 0) {
tbody.style.display = 'none';
table.insertAdjacentElement('afterend', message);
emptyListMessage.style.display = 'none';
} else {
tbody.style.display = '';
message.remove();
if (emptyListMessage && rows.length === 0) {
emptyListMessage.style.display = '';
}
}
});
```

Ilustración 50 - Código JavaScript del buscador presente en la tabla de gestión

Como se puede observar, se trata de código JavaScript cuya lógica consiste en realizar una búsqueda sobre una lista de objetos ya obtenida por la llamada inicial al microservicio en lugar de realizar otra llamada con el filtro, de esta forma, es posible reducir el coste del proceso.

5.1.7 Tema claro/oscuro

La interfaz dispone de una funcionalidad común en este tipo de herramientas, a la par que útil. Se trata de un botón para alternar entre tema oscuro o claro, una opción valiosa que puede proporcionar una experiencia de usuario más cómoda, mejorar la legibilidad y permitir una personalización estética según las preferencias individuales. En la interfaz de la herramienta, se encuentra situado en la parte derecha de la barra superior:

Ilustración 51 - Selector de tema claro del dashboard

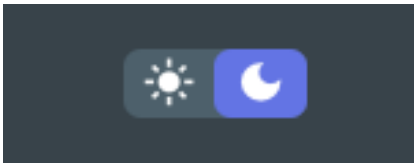


Ilustración 52 - Selector de tema oscuro del dashboard

Su código está desarrollado en JavaScript, y su funcionamiento consiste en añadir o eliminar una clase al código HTML de diversos elementos, además de modificar la apariencia del propio botón de cambiar el tema de la herramienta:

```
const themeToggler = document.querySelector(".theme-toggler");

// change theme
themeToggler.addEventListener('click', () => {
    document.body.classList.toggle('dark-theme-variables');
    themeToggler.querySelector('span:nth-child(1)').classList.toggle('active');
    themeToggler.querySelector('span:nth-child(2)').classList.toggle('active');
})
```

Ilustración 53 - Código JavaScript del selector de tema del dashboard

La sección de código CSS en la que se seleccionan los colores ha sido preparada para una facilidad de mantenimiento e integración con esta funcionalidad:

```css
/* ROOT VARIABLES */
:root {
    --color-primary: #7380ec;
    --color-danger: #ff7782;
    --color-success: #41f1b6;
    --color-warning: #ffbb55;
    --color-white: #fff;
    --color-info-dark: #7d8da1;
    --color-info-light: #dce1eb;
    --color-dark: #363949;
    --color-light: rgba(132, 139, 200, 0.18);
    --color-primary-variant: #111e88;
    --color-dark-variant: #677483;
    --color-background: #f1f3f9;
    --color-main-yellow: #f8c146;

    --card-border-radius: 2rem;
    --border-radius-1: 0.4rem;
    --border-radius-2: 0.8rem;
    --border-radius-3: 1.2rem;

    --card-padding: 1.8rem;
    --padding-1: 1.2rem;

    --box-shadow: 0 2rem 3rem var(--color-light)
}

/* DARK THEME VARIABLES */
.dark-theme-variables {
    --color-background: #D6DBE2;
    --color-main-yellow: #E2AC3E;
    --color-white: #414C54;
    --color-dark: #edeffd;
    --color-dark-variant: #a3bdcc;
    --color-light: #596B77;
    --box-shadow: 0 2rem 3rem var(--color-light)
}
```

Ilustración 54 - Código CSS del tema claro/oscuro del dashboard

```css
h3 {
    font-size: 0.87rem;
    color: var(--color-dark);
}

h4 {
    font-size: 0.8rem;
    color: var(--color-dark);
}
```

Ilustración 55 - Ejemplo de uso del método var() en CSS

Como se puede ver, el método var() de CSS es de gran utilidad para diseñar este tipo de funcionalidades, y es por ello por lo que se ha decidido utilizarlo para permitir el cambio de tema en la herramienta.

5.2 Microservicios

A continuación, se explica cómo se han desarrollado los microservicios presentes en este TFG. En esta presentación se tomará el microservicio "bomberos-api" como ejemplo a mostrar debido a que los 3 disponen de configuraciones similares a excepción de las entidades presentes en cada uno.

El primer paso, al igual que con cualquier otro proyecto, es crear el esqueleto del proyecto, usando para ello Composer:

```
rafa@MacBook-Pro-de-Rafael dashboard % composer create-project symfony/skeleton bomberos-api
```

Ilustración 56 - Comando para crear el esqueleto de bomberos-api mediante Composer

A continuación, se configurarán los ficheros de entorno. Este paso es similar al que se hizo para configurar el dashboard, con la única diferencia de que los microservicios conectan directamente con la base de datos:

```
$ .env.local
1   APP_ENV=dev
2
3   DATABASE_URL=mysql://root:password@localhost:3306/bomberos-api
```

Ilustración 57 - Archivo de entorno de desarrollo de bomberos-api

Tras esto, se deberá cambiar la configuración del fichero "composer.json" con todos los paquetes necesarios. El comando para instalar el paquete principal del microservicio es el siguiente:

```
rafa@MacBook-Pro-de-Rafael bomberos-api % composer require api-platform/core
```

Ilustración 58 - Comando para instalar Api Platform

Tras instalar este paquete, podemos decir que tenemos un proyecto Api Platform. Este paquete proporciona una serie de herramientas y componentes para desarrollar y exponer microservicios de una manera rápida y sencilla, y es lo que diferenciará el funcionamiento de este microservicio de cualquier otro desarrollado en PHP o Symfony.

En los microservicios también se utilizan otros dos paquetes importantes, los cuales permiten cumplir con el objetivo del proyecto de generar archivos mediante comandos siempre que sea posible. Se trata de "symfony/maker-bundle" y "orm", los cuales permitirán crear entidades desde la terminal.

Una vez el microservicio ha sido configurado, podremos empezar a crear entidades. Ya que se está tomando de ejemplo el microservicio de los bomberos, también tomaremos de ejemplo una entidad propia de ellos, como puede ser Noticias (en inglés Post):

```
rafa@MacBook-Pro-de-Rafael policia-api % php bin/console make:entity

 Class name of the entity to create or update (e.g. AgreeablePuppy):
 > Post

 Mark this class as an API Platform resource (expose a CRUD API for it) (yes/no) [no]:
 > yes
```

Ilustración 59 - Comando para crear una entidad en Api Platform

Tras este comando, se creará un nuevo fichero llamado “Post.php”, con todas las propiedades que queramos darle a la nueva entidad, y sus respectivos setters y getters:

```
src > Entity > Post.php > ...
    You, 56 minutes ago | 1 author (You)
 1  <?php
 2
 3  namespace App\Entity;
 4
 5  use ApiPlatform\Core\Annotation\ApiResource;
 6  use App\Repository\PostRepository;
 7  use Doctrine\ORM\Mapping as ORM;
 8
    You, 56 minutes ago | 1 author (You)
 9  /**
10   * @ApiResource()
11   * @ORM\Entity(repositoryClass=PostRepository::class)
12   */
13  class Post
14  {
15      /**...
20      private $id;
21
22      /**...
25      private $title;
26
27      /**...
30      private $subtitle;
31
32      public function getId(): ?int
33      {...
35      }
36
37      public function getTitle(): ?string
38      {...
40      }
41
42      public function setTitle(string $title): self
43      {...
47      }
48
49      public function getSubtitle(): ?string
50      {...
52      }
53
54      public function setSubtitle(string $subtitle): self
55      {...
59      }
60  }
```

Ilustración 60 - Código de la entidad Post

Se puede observar que la entidad Post tiene 3 atributos en su estado actual, lo cual aumentará según avance el proyecto. El atributo $id, el cual actúa como identificador o clave primaria, es autogenerado y un usuario corriente no debería tener acceso a él. Por otro lado, $title y $subtitltle serán los atributos que el usuario sí podrá alterar y cuyos cambios serán reflejados en la página web.

Para que esta nueva entidad se vea reflejada en la base de datos como una nueva tabla, se deberán ejecutar 2 comandos. A continuación se muestra el primero:

```
rafa@MacBook-Pro-de-Rafael bomberos-api % php bin/console make:migration
```

Ilustración 61 - Comando para crear un archivo de migración Version

Tras ejecutarlo, se creará un archivo “Version.php” con la instrucción SQL que se ejecutará para realizar este cambio:

```php
<?php

declare(strict_types=1);

namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

You, 1 second ago | 1 author (You)
/**
 * Auto-generated Migration: Please modify to your needs!
 */
final class Version20230601110142 extends AbstractMigration
{
    public function getDescription(): string
    {
        return '';
    }

    public function up(Schema $schema): void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->addSql('CREATE TABLE post (id INT AUTO_INCREMENT NOT NULL, title VARCHAR(255) NOT NULL,
        subtitle VARCHAR(255) NOT NULL, image VARBINARY(255) NOT NULL, PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
    }

    public function down(Schema $schema): void
    {
        // this down() migration is auto-generated, please modify it to your needs
        $this->addSql('DROP TABLE post');
    }
}
```

Ilustración 62 - Código de un archivo Version

Como se puede ver, en la clase principal Version se encuentran los métodos up() y down(), los cuales aplicarán y revertirán los cambios respectivamente. Para ejecutar el método up() de esta Version, bastará con ejecutar el siguiente comando:

```
rafa@MacBook-Pro-de-Rafael bomberos-api % php bin/console doctrine:migrations:migrate
```

Ilustración 63 - Código para aplicar las instrucciones del archivo Version en la base de datos

En el caso de que no se haya producido ningún error, se podrá comprobar que los cambios han tenido efecto inmediato visitando la base de datos:

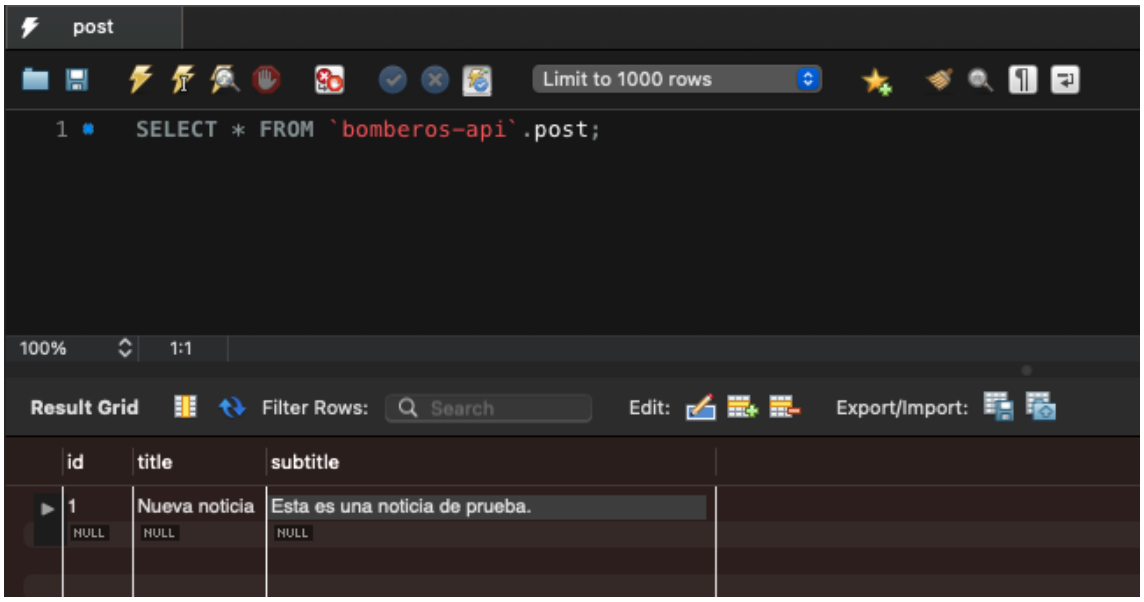


Ilustración 64 - Base de datos de Post

En la imagen se puede observar que es posible almacenar objetos de tipo Noticia en la base de datos, los cuales podrán ser leídos desde cualquier lugar del proyecto que disponga de la conexión correcta.

Ya que en este proyecto las conexiones entre componentes se realizan mediante endpoints, debemos conocer la ruta de estos endpoints. Api Platform proporciona una documentación por tablas con toda la información que necesitamos, la cual se encuentra en la url “(dominio-del-microservicio)/api”.

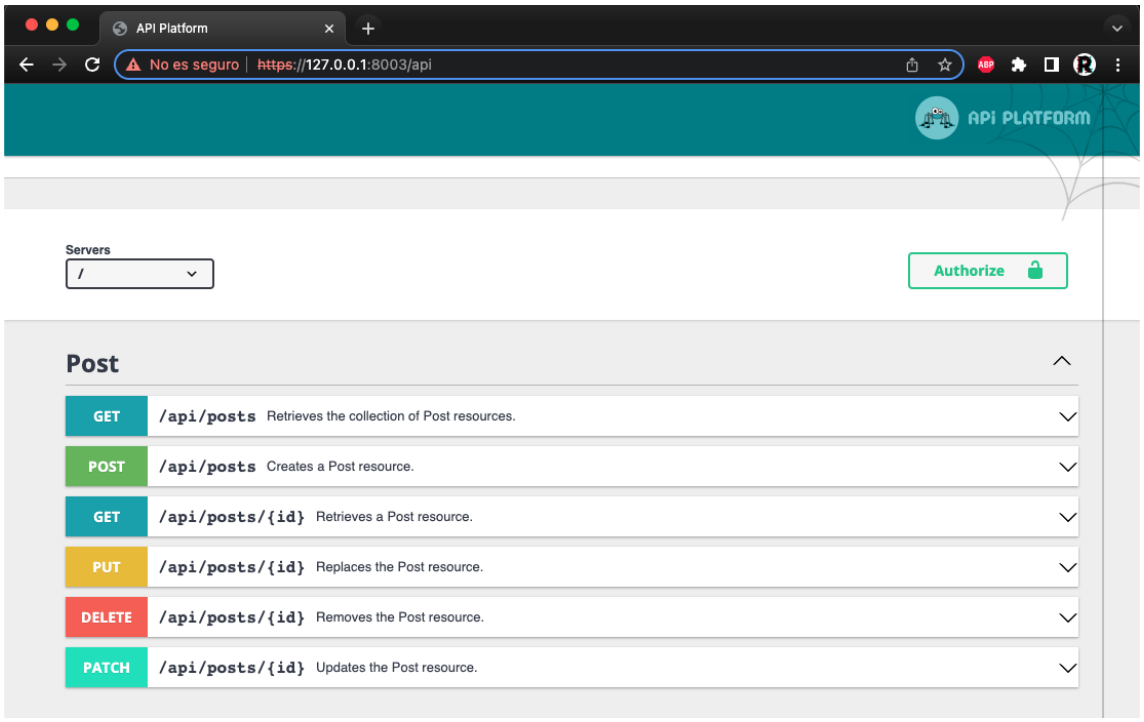


Ilustración 65 - Documentación web de Api Platform de la entidad Post

En la imagen se pueden apreciar todos los métodos que conforman las operaciones CRUD de la entidad Post, junto con todos sus endpoints abiertos.

5.3 Páginas web de los servicios públicos

En este apartado se explica cómo se han desarrollado las 2 páginas web de ejemplo de los bomberos y de la policía local de Valencia, con un diseño muy sencillo cuyo propósito es demostrar que el sistema es funcional. Ambas páginas son muy similares, a excepción del front-end. Debido a que no existen diferencias interesantes presentes en únicamente una de ellas, solo se mostrará, a modo de ejemplo, el desarrollo de la página web de los bomberos.

Tras haber preparado la configuración inicial del proyecto, se deberá construir una página web con una interfaz principal. Para ello, se seguirá el patrón de diseño MVC. Lo primero será programar el controlador, llamado “HomepageController.php”, con un método que renderice una vista. En la siguiente imagen se puede ver el resultado:

```php
<?php

namespace App\Controller;

use App\Service\PostService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Exception;

You, now | 1 author (You)
/**
 * @Route("", name="homepage_")
*/
class HomepageController extends AbstractController
{
    private $postService;

    public function __construct(PostService $postService) {
        $this->postService = $postService;
    }

    /**
     * @Route("/", name="index")
     */
    public function index(): Response
    {
        $posts = $this->postService->getPostsList();

        return $this->render('homepage/index.html.twig', [
            'items' => $posts,
        ]);
    }
}
```

Ilustración 66 - Controlador principal de la página web de los bomberos

En la imagen se puede observar un método index(), el cual será invocado al acceder a la url principal “/”. Su único cometido es, mediante el método render(), cargar el contenido del fichero “homepage/index.html.twig” pasándole como argumento un array de noticias obtenidas desde un endpoint en el método getPostsList():

```php
<?php

namespace App\Service;

use Symfony\Component\HttpClient\HttpClient;
use Exception;

You, 2 minutes ago | 1 author (You)
class PostService
{
    private $apiUrl;
    private $httpClient;

    public function __construct(string $apiUrl)
    {
        $this->apiUrl = $apiUrl;
        $this->httpClient = HttpClient::create([
            'verify_peer' => false,
            'verify_host' => false,
        ]);
    }

    public function getPostsList()      You, 2 weeks ago • dynamic index working
    {
        try {
            $response = $this->httpClient->request('GET', $this->apiUrl . '/posts');
            $data = $response->toArray();

            return $data['hydra:member'];
        } catch (\Exception $exception) {
            throw new Exception($exception->getMessage());
        }
    }
}
```

Ilustración 67 - Servicio conectado al endpoint de la entidad Post

Una vez se dispone de la conexión con el microservicio funcionando correctamente, será posible enviar objetos de la base de datos al front-end, veamos cómo se ha preparado el código HTML de la página web:

```twig
{% for item in items %}

    <div class="news-item">
        <div class="news-content">
            <h3 class="news-title">{{ item.title }}</h3>
            <p class="news-subtitle">{{ item.subtitle }}</p>
            <p class="news-place">Lugar:{{ item.place }}</p>
            <p class="news-author">Autor:{{ item.author }}</p>
        </div>/.news-content
    </div>/.news-item

{% endfor %}
```

Ilustración 68 - Fragmento de código que renderiza de forma dinámica los objetos Post

Mediante el código de la imagen, contenido en un bucle "for" de twig, es posible mostrar información en la página web de forma dinámica. De esta forma, si se añadiera una nueva noticia a la base de datos, se vería reflejado de forma inmediata en la interfaz web.

Algo importante que se debe tener en cuenta siempre que se sigan este tipo de patrones, es controlar todas las situaciones posibles. En el caso de que se produzca un fallo en el microservicio, o no existan objetos en la base de datos, el sistema debería ser capaz de adaptarse, mostrando algún mensaje de forma controlada sin detener el funcionamiento de toda la página web.

El resultado final de las páginas web de la policía y los bomberos se puede ver a continuación:

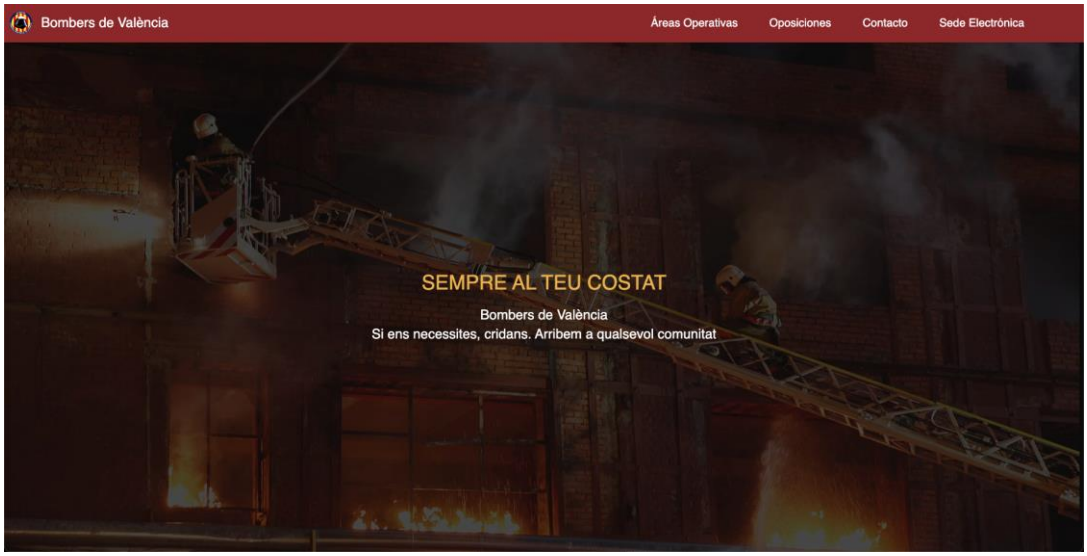


Ilustración 69 - Página web de los bomberos

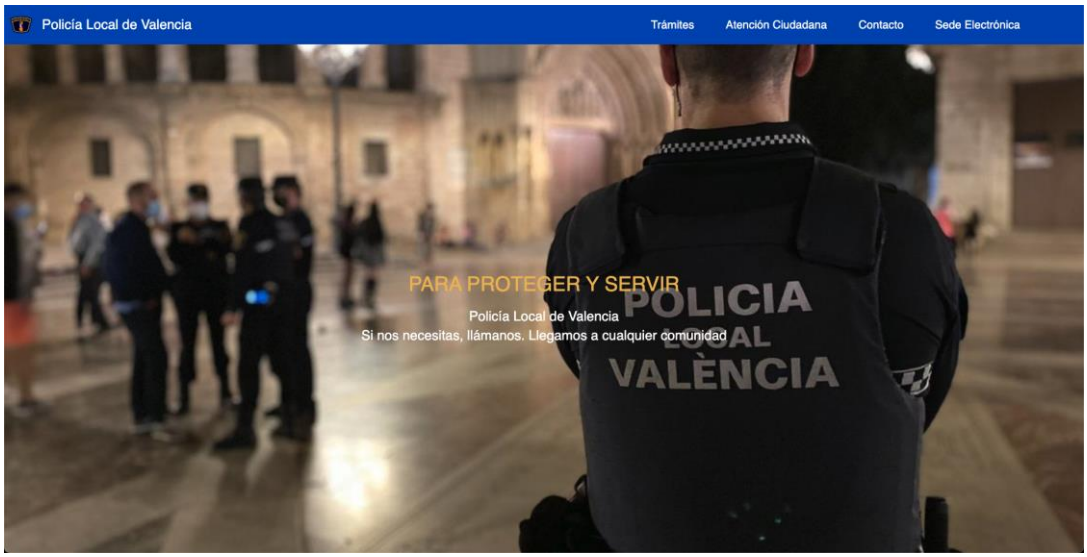


Ilustración 70 - Página web de la policia

5.4 WordPress

En este apartado se explica cómo se ha desarrollado el plugin de WordPress utilizado en uno de los casos de uso de este TFG. Ya que WordPress es un CMS de código abierto, es posible instalarlo y realizar modificaciones en un entorno de desarrollo para su desarrollo. Esta es una práctica común para desarrollar plugins de WordPress, los cuales se podrían subir a la propia plataforma y monetizarlos.

Para explicar cómo se ha realizado el código, se tomará de ejemplo la entidad Event, la cual es leída desde el dashboard, quedando presente en WordPress sin necesidad de utilizar las opciones del CMS.

5.4.1 Configuración

Para que las funcionalidades desarrolladas tengan efecto, es necesario instalar el plugin, al cual se le ha llamado “LevelUp Pro” por ser un plugin cuya temática es el control de la gamificación de la página web a la que esté conectado. La forma de hacerlo es la misma para instalar cualquier otro plugin presente en el CMS. Desde el apartado Plugins de la barra lateral de WordPress se hará click en el botón de “Añadir nuevo”, y en el listado de plugins se buscará el plugin desarrollado. Tras esto, aparecerá en el listado de plugins instalados. Otro plugin fundamental para que pueda funcionar este caso de uso, es el llamado Advanced Custom Fields (ACF), el cual permite crear en WordPress tablas de objetos con columnas personalizadas. Estas columnas deberán ser similares a las de la base de datos en la que se encuentra almacenada la información leída:

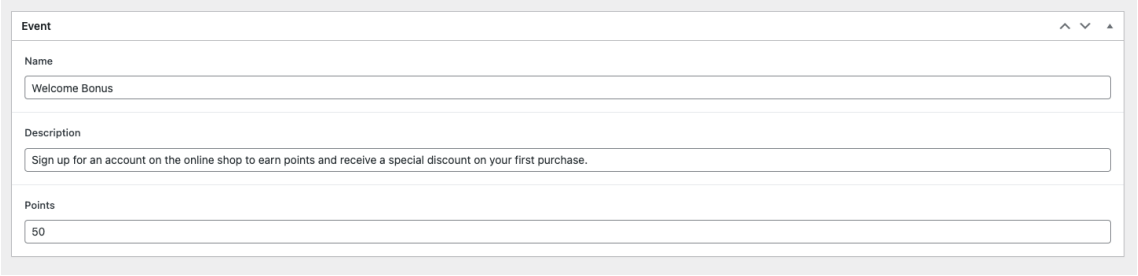


Ilustración 71 - Creación de una tabla con columnas personalizadas mediante ACF

A continuación, se puede observar el apartado de plugins con el listado de plugins instalados:

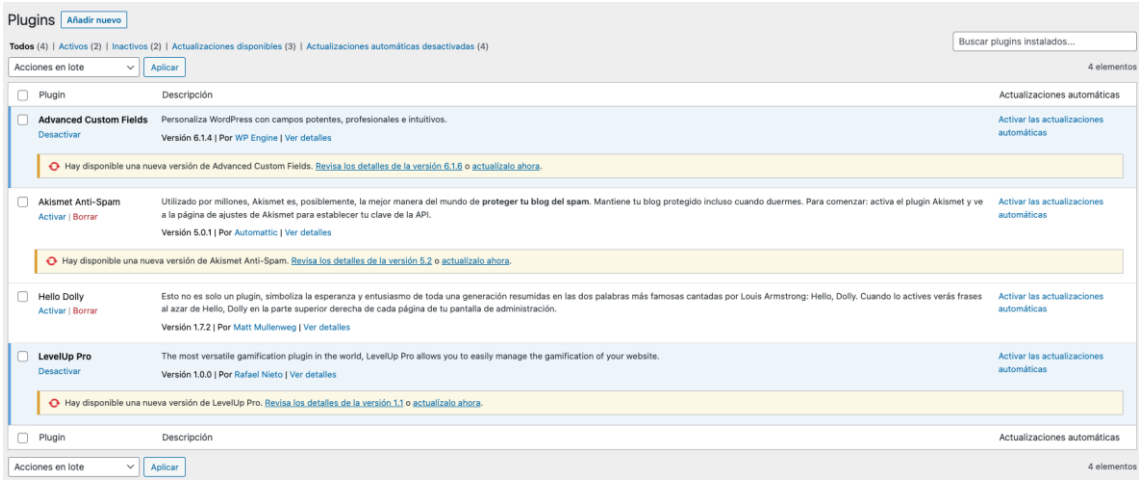


Ilustración 72 - Plugins de WordPress instalados en el proyecto

Tras disponer de un portal de WordPress preparado para el desarrollo de un plugin, podemos decir que es el momento de empezar con el desarrollo del código.

5.4.2 Desarrollo

Los plugins, al igual que WordPress, están desarrollados en PHP y disponen de métodos propios de WordPress, por lo que será el lenguaje del que se hablará a lo largo de este apartado. Todos los archivos que conforman un plugin deben estar ubicados en la carpeta “plugins” de WordPress:

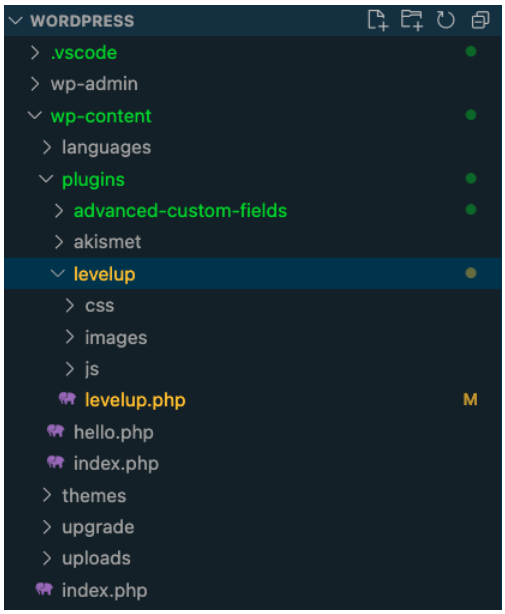


Ilustración 73 - Estructura de carpetas de WordPress

Lo primero que debe hacerse al desarrollar un plugin es seguir un estándar de WordPress en el que se deben especificar ciertas características como la versión, el autor y una descripción del plugin. Esto se debe escribir justo al inicio del fichero principal, con una sintaxis propia de los comentarios de PHP. Para este plugin se han especificado las siguientes características:

```
/**
 * Plugin Name: LevelUp Pro
 * Description: The most versatile gamification plugin in the world, LevelUp Pro allows you to easily manage the gamification of your website.
 * Author: Rafael Nieto
 * Author URI: https://rafaelnieto.com
 * Version: 1.0.0
 */
```

Ilustración 74 - Configuración principal del plugin de WordPress

Lo siguiente que se ha añadido al plugin, y que siempre debería ser tenido en cuenta al desarrollar un plugin, es el siguiente fragmento de código, justo a continuación de las características:

```
if (!defined('ABSPATH')) {
    echo 'Nothing to see here';
    exit;
}
```

Ilustración 75 - Capa de seguridad en el plugin de WordPress

Debido a que es posible visualizar el contenido de cualquier fichero del proyecto desde la url del sitio web, cabe la posibilidad de comprometer datos privados por error. La función del código visto en la imagen anterior es la de mostrar un mensaje “Nothing to see here” en lugar del contenido del fichero, añadiendo así una capa de seguridad.

Tras esto, y siguiendo el ejemplo de leer la entidad Event de la base de datos, se deberá crear un nuevo apartado en la barra lateral del portal de WordPress. Esto se ha hecho utilizando el método add_action() dentro la función constructora del fichero, de forma que la función que añade este nuevo apartado es llamada tan pronto como se carga WordPress:

```
public function __construct()
{
    // Create events page
    add_action('init', array($this, 'listEvents'));
```

Ilustración 76 - Inicialización de un nuevo apartado en el portal de WordPress

```php
public function listEvents()
{
    $args = array(
        'label' => 'Events',
        'public' => true,
        'capability_type' => 'post'
    );

    register_post_type('events', $args);
}
```

Ilustración 77 - Método con los atributos de un apartado personalizado en WordPress

Siendo el resultado el siguiente en la barra lateral de WordPress:

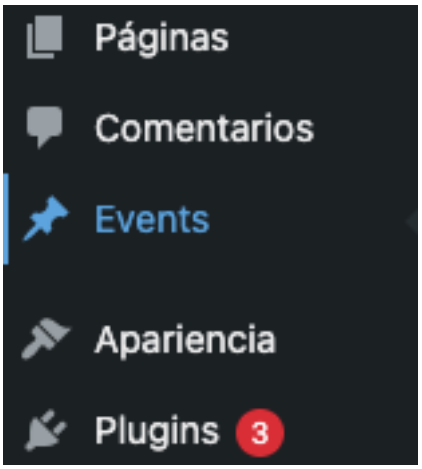


Ilustración 78 - Vista de un apartado personalizado en el portal de WordPress

Lo siguiente que se explicará, siendo lo más importante de este plugin, es cómo se recibe la información desde el microservicio “gamification-api”.

Debido a que nos encontramos ante un proyecto desarrollado en PHP, una forma común de abordar este tipo de labor es mediante cURL (Client for URLs), una biblioteca que permite enviar solicitudes HTTP a servidores web y recibir respuestas:

```php
public function getData()
{
    $curl = curl_init();

    curl_setopt_array($curl, array(
        CURLOPT_URL => 'http://localhost:8001/api/events/',
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_ENCODING => '',
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 0,
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
        CURLOPT_CUSTOMREQUEST => 'GET',
        CURLOPT_SSL_VERIFYPEER => false,
        CURLOPT_SSL_VERIFYHOST => false
    ));

    $response = curl_exec($curl);

    curl_close($curl);

    $data = json_decode($response);

    return $data;
}
```

Ilustración 79 - Función que recibe información desde el microservicio hasta el plugin

Se puede ver en la imagen una función getData(), cuyo propósito es, utilizando los métodos propios de cURL, recibir información del endpoint de la entidad Event en formato JSON y devolverla en forma de un array de objetos. Este array será utilizado para poblar la tabla de la entidad Event, en el nuevo apartado de la barra lateral de WordPress ya mencionado.

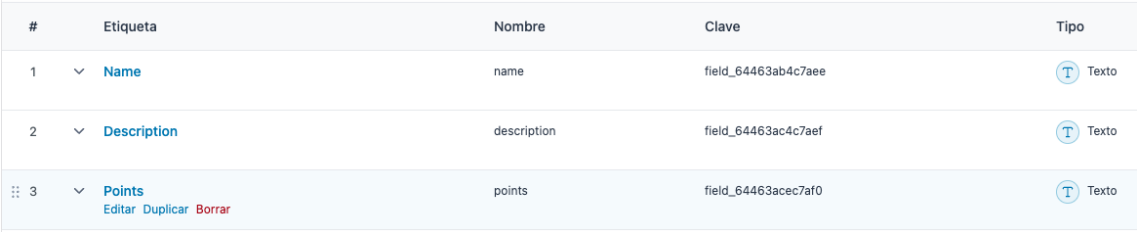
A continuación, se pasa a explicar el método más importante del plugin, el cual es el encargado de controlar la lógica de la tabla del apartado Event.

Algo importante que se debe tener en cuenta al poblar la tabla, es de qué forma diferenciar los objetos entre ellos en el caso de que tengan un nombre similar. La forma en la que se ha decidido resolver es, en lugar de insertar el nombre del objeto, insertar un nuevo string formado por el nombre del objeto, un guion y el id del objeto mediante el método sanitize_title():

```php
$event_slug = sanitize_title($event->name . '-' . $event->id);
```

Ilustración 80 - Creación de un identificador personalizado para los objetos en el plugin

Otro aspecto a tener en cuenta sobre este proceso de inserción de datos en la tabla es cómo saber en qué columna se debe insertar cada atributo del objeto. Debido a que se utiliza el plugin ACF para crear la tabla, también es posible acceder a unas claves que proporciona este plugin para identificar cada columna:



#	Etiqueta	Nombre	Clave	Tipo
1	Name	name	field_64463ab4c7aee	Texto
2	Description	description	field_64463ac4c7aef	Texto
3	Points Editar Duplicar Borrar	points	field_64463acec7af0	Texto

Ilustración 81 - Identificador de las columnas de la tabla del plugin

Estas claves se han utilizado en el código del plugin para formar un array con todas las columnas de la tabla, en las que se insertará cada atributo:

```
$fillable = [
    'field_64463ab4c7aee' => 'name',
    'field_64463ac4c7aef' => 'description',
    'field_64463acec7af0' => 'points'
];

foreach ($fillable as $key => $name) {
    update_field($key, $event->$name, $inserted_event);
}
```

Ilustración 82 - Fragmento de código sobre el uso de los identificadores de cada columna de la tabla personalizada del plugin

Más adelante se mostrará el código completo para entender en su totalidad el proceso, pero hasta este momento es interesante entender cómo se utilizan las claves de ACF dentro de un bucle “foreach”.

Se ha añadido también un control a la función para evitar objetos repetidos. Uno de los problemas al desarrollar la lógica de esta tabla fue que los objetos recibidos eran sumados a los ya existentes, y se debería de poder evitar de alguna forma. Se decidió entonces añadir un bucle “foreach” que, mediante el método propio de WordPress wp_delete_post(), vacíe la tabla antes de poblarla con la nueva información:

```php
$existing_events = get_posts(array(
    'post_type' => 'events',
    'numberposts' => -1,
    'post_status' => 'any'
));

foreach ($existing_events as $existing_event) {
    wp_delete_post($existing_event->ID, true);
}
```

Ilustración 83 - Fragmento de código que limpia los objetos presentes en la tabla del plugin antes de añadir otros nuevos

Como último control dentro de la función, es de vital importancia que no se detenga el proceso entero en caso de que se produzca un error en la lectura. La forma en la que se ha decidido controlar esto es ignorando la información malformada, saltando al siguiente objeto a leer en caso de que se produzca un fallo. En la siguiente imagen se puede ver como se ha conseguido mediante el método propio de WordPress is_wp_error():

```php
if (is_wp_error($inserted_event)) {
    continue;
}
```

Ilustración 84 - Capa de control en caso de fallo al cargar datos en el plugin

Tras haber explicado los puntos más importantes de la carga de datos de este plugin, se procede a mostrar el aspecto de la función completa:

```php
public function load_events_table()
{
    $existing_events = get_posts(array(
        'post_type' => 'events',
        'numberposts' => -1,
        'post_status' => 'any'
    ));

    foreach ($existing_events as $existing_event) {
        wp_delete_post($existing_event->ID, true);
    }

    $data = $this->getData();

    // Loop through the events from the API
    foreach ($data->{"hydra:member"} as $event) {
        $event_slug = sanitize_title($event->name . '-' . $event->id);

            $inserted_event = wp_insert_post([
                'post_name' => $event_slug,
                'post_title' => $event_slug,
                'post_type' => 'events',
                'post_status' => 'publish',
            ]);

            if (is_wp_error($inserted_event)) {
                continue;
            }

            $fillable = [
                'field_64463ab4c7aee' => 'name',
                'field_64463ac4c7aef' => 'description',
                'field_64463acec7af0' => 'points'
            ];

            foreach ($fillable as $key => $name) {
                update_field($key, $event->$name, $inserted_event);
            }
    }
}
```

Ilustración 85 - Método encargado de la lógica de la tabla del plugin

5.5 Configuración del entorno de desarrollo

Al comenzar con el desarrollo de un proyecto de software, es posible realizar los primeros pasos de formas diferentes. En el caso de este TFG, se ha realizado en mi ordenador personal, mediante el uso de servidores web. Esta es una técnica conveniente, debido a que puedo probar y depurar el código de una forma muy sencilla, y en un proyecto como este, en el que he utilizado un gran número de técnicas que desconocía, los cambios y errores serían frecuentes.

Symfony proporciona un comando con el cuál es posible iniciar uno de estos servidores web preparado para el desarrollo en local:

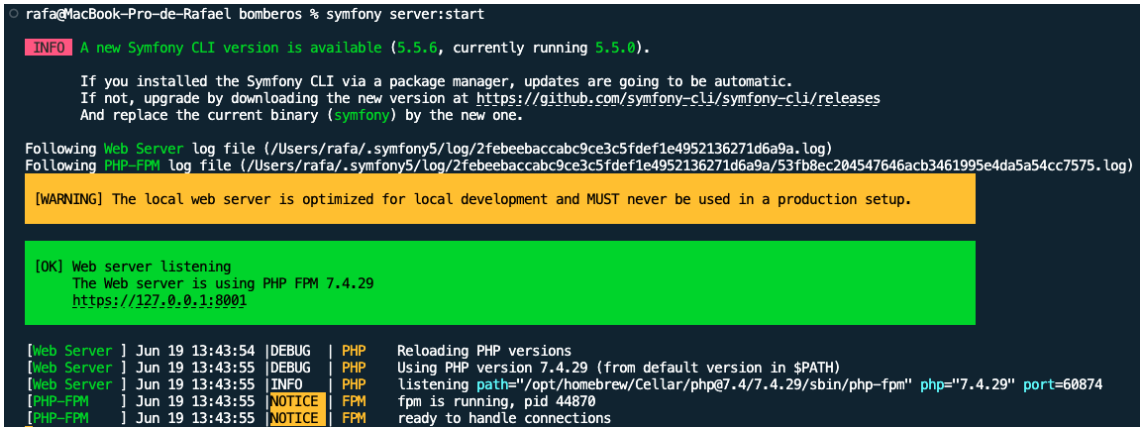


Ilustración 86 - Comando para iniciar un servidor web de Symfony

Tras ejecutarlo, un nuevo puerto se abrirá, comenzando por el 8000 a no ser que se indique lo contrario, y estará disponible para su uso. Debido a que en este TFG se requieren diferentes puertos, se han tomado dos decisiones para hacer el proceso más rápido y menos complicado.

La primera decisión ha sido crear un archivo ejecutable llamado “server.sh” en cada proyecto, de forma que su contenido pueda ser ejecutado desde la terminal mediante la orden “sh server.sh”. Su contenido, tomando el proyecto de la página web de los bomberos como ejemplo, es el siguiente:

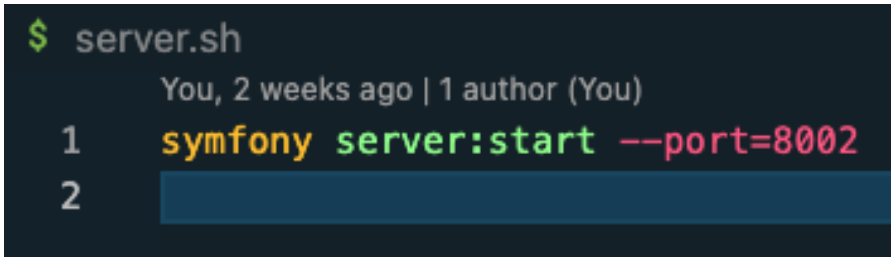


Ilustración 87 - Contenido del archivo "server.sh"

Se trata de un fichero con una única línea de código, el cual ejecutará el comando especificado en la línea de comandos. Se puede también apreciar que se ha utilizado la etiqueta "--port" para forzar a que el puerto abierto sea el indicado, lo cual ha sido la segunda decisión tomada.

En el estado actual de este TFG, existen 7 proyectos independientes que necesitan un puerto abierto para funcionar, y este puerto debería ser siempre el mismo para no provocar fallos en las llamadas a los endpoints. Para lograrlo, se ha asignado un puerto a cada uno de los proyectos independientes como sigue:

Nombre del proyecto	Puerto
dashboard	8000
gamification-api	8001
bomberos	8002
bomberos-api	8003
policia	8004
policia-api	8005
wordpress	8080

5.6 Conexión mediante endpoints

Todas las interconexiones en todos los proyectos individuales de este TFG, a excepción del plugin de WordPress ya explicado, están desarrolladas de una forma similar. Para ilustrar cómo se ha estructurado y desarrollado su funcionamiento, se tomará de ejemplo la lectura de la entidad Noticias (Post) de la base de datos desde la interfaz del dashboard.

A lo largo de este apartado, se modificará ligeramente parte del código mostrado en las capturas de pantalla, de modo que la información mostrada sea lo más resumida y concreta posible.

El primer paso es, en el código del dashboard, inicializar la variable a la cual se le asignará el endpoint para realizar la llamada al microservicio "bomberos-api". Para ello, y siguiendo con los estándares de Symfony, se ha añadido una nueva variable en el fichero de entorno de desarrollo, la cual apuntará a la url generada por Api Platform:

Ilustración 88 - Inicialización de la variable con la ruta del microservicio bomberos-api en el archivo de entorno de desarrollo del dashboard

Una vez la variable "BOMBEROS_API_URL" ha sido inicializada, se deberá importar dicha variable en el archivo "PostService.php", el cual será el encargado de realizar llamadas sobre la entidad Noticia. Esto puede hacerse en el fichero "services.yaml" de la siguiente forma:

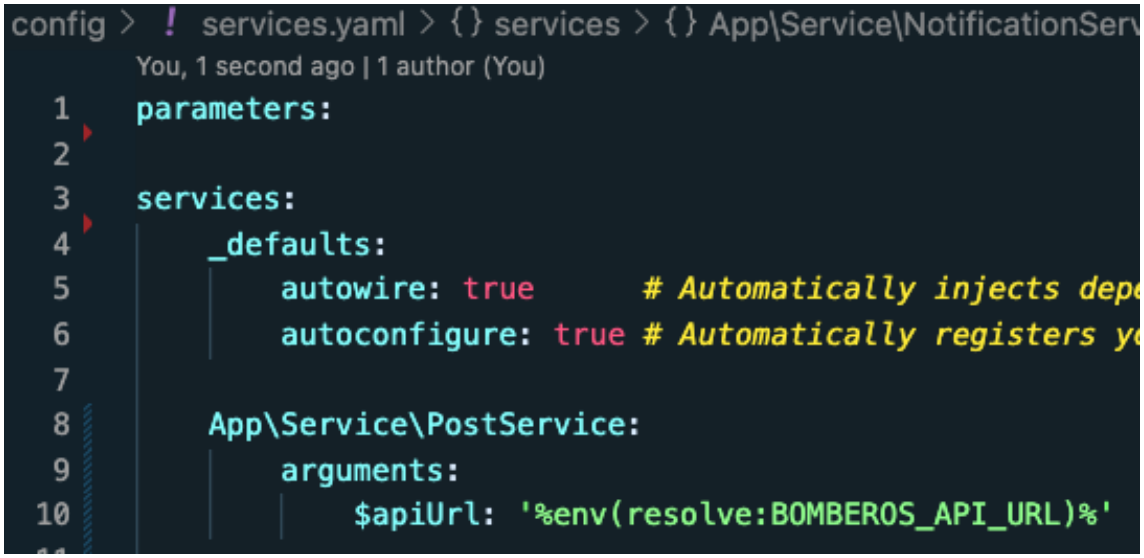


Ilustración 89 - Fragmento de código de una inyección de dependencias en el dashboard

Como se puede apreciar, a partir de la línea 8 se proporciona un argumento $apiUrl con el contenido de la variable "BOMBEROS_API_URL" a través de una inyección de dependencias. A partir de este momento, se podrá leer dicha variable desde el archivo "PostService.php", la cual puede ser modificada desde el fichero de entorno.

A continuación, se muestra una captura de pantalla del archivo "PostService.php", la cual procederemos a estudiar:

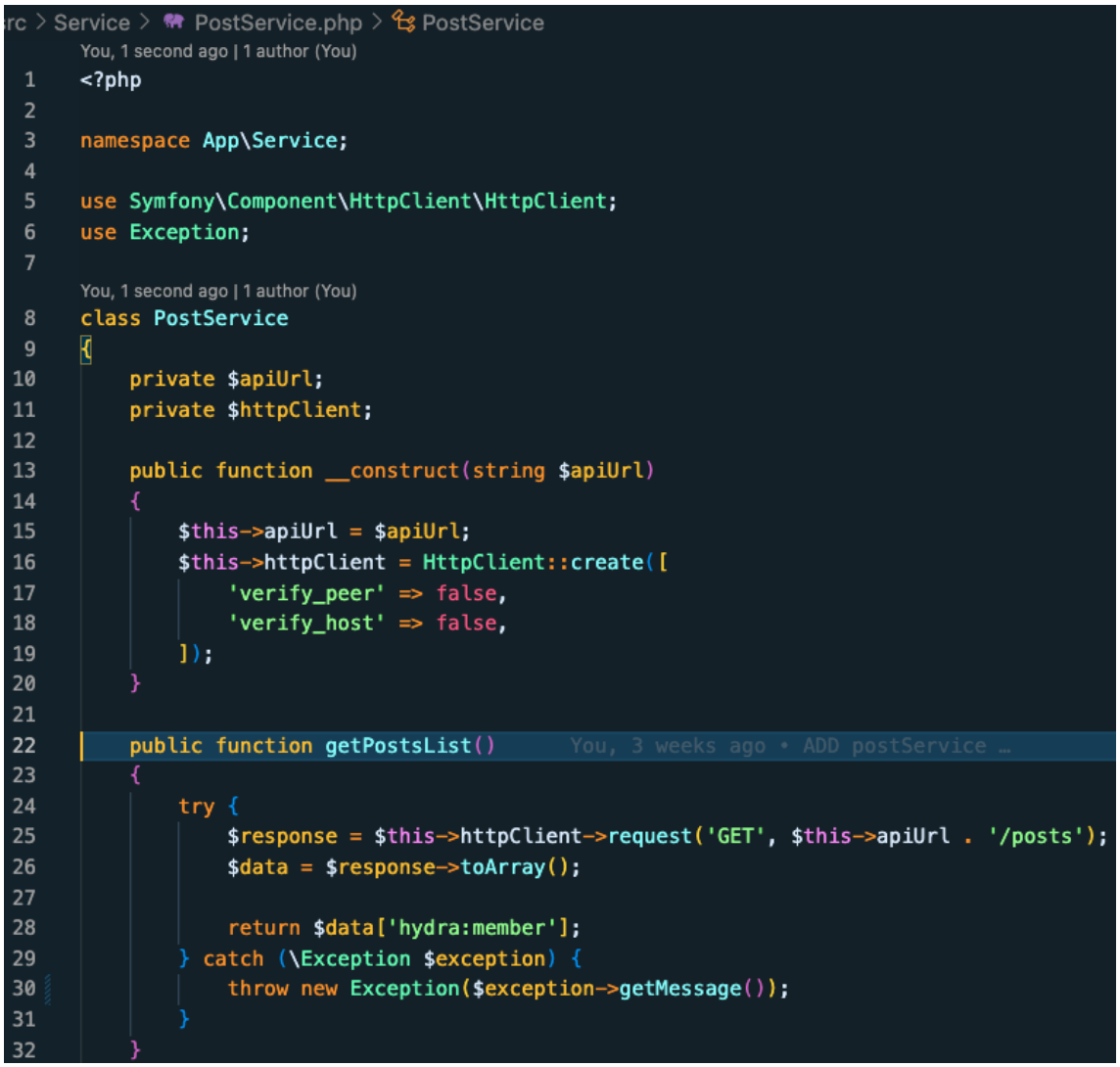


Ilustración 90 - Fragmento de código del archivo "PostService.php" con la inicialización de una variable de ruta de microservicio

En la línea 15 se puede observar cómo se asigna el contenido de la variable “BOMBEROS_API_URL” a la variable $this->apiUrl. Esta variable será utilizada en todos los métodos de las llamadas al microservicio.

Tras disponer de un endpoint preparado para su uso, debemos entender cómo utilizarlo. En la línea 25, se realiza una llamada de tipo GET a la url del endpoint abierto mediante el método request(), pasándole como argumentos el tipo de la llamada deseada y la url a la cual se hará dicha llamada.

Para terminar de entender cómo realizar estas llamadas, se verá ahora de qué forma podría realizarse una llamada para eliminar un objeto determinado. El tipo de la llamada pasará a ser “DELETE”, y en el argumento de la url del microservicio se deberá añadir el atributo $id del objeto elegido:

```php
    public function delete($id)
    {
        try {
            $response = $this->httpClient->request('DELETE', $this->apiUrl . '/posts/' . $id);
            $data = $response->toArray();

            return $data;
        } catch (\Exception $exception) {
            throw new Exception($exception->getMessage());
        }
    }
```

Ilustración 91 - Método encargado de realizar operaciones 'DELETE' sobre la entidad Post en el dashboard

De esta forma, es posible disponer de un sistema que sea escalable y legible a nivel técnico, a la par que intuitivo para un usuario que quiera realizar métodos propios de CRUD.

6. Implantación y pruebas

Una vez han sido desarrollados y conectados todos los proyectos individuales que conforman este TFG, se debe comprobar el correcto funcionamiento del sistema, tema a tratar en este apartado.

Para ello, hay dos puntos principales que deben probarse. Uno de ellos es el inicio y cierre de sesión de un usuario del dashboard, y el otro será realizar una operación CRUD, función principal del dashboard.

Primero se comprobará que un usuario pueda iniciar y cerrar su sesión, y qué cambios se pueden apreciar tras hacerlo.

Empezando en la pantalla de login del dashboard, el usuario ingresa un nombre de usuario y contraseña válidos. Estas credenciales han sido creadas directamente en la base de datos, como ya se explicó previamente. Tras ello, el usuario es redirigidos a la página principal del dashboard, viendo en la esquina superior derecha el nombre de usuario:

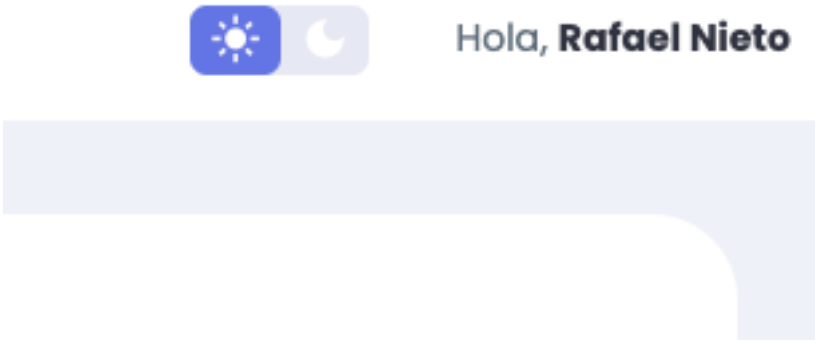


Ilustración 92 - Nombre de usuario en el dashboard

A continuación, el usuario pulsa en el botón “Logout”, en la parte inferior del menú izquierdo. Al hacerlo, será dirigido de vuelta a la página de login:

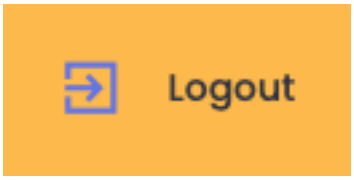


Ilustración 93 - Botón de logout

Tras haber comprobado su correcto funcionamiento, se procede a realizar la siguiente prueba.

Hay que asegurar que todas las conexiones entre los diferentes proyectos individuales estas correctamente configuradas, siendo posible modificar elementos de una página web desde el dashboard. Se intentará crear un objeto Noticia desde el dashboard y a comprobar que su aparición queda reflejada en la página web de los bomberos.

En primer lugar, nos situamos en la tabla de gestión del dashboard, y nos dirigiremos al formulario de creación de objetos:

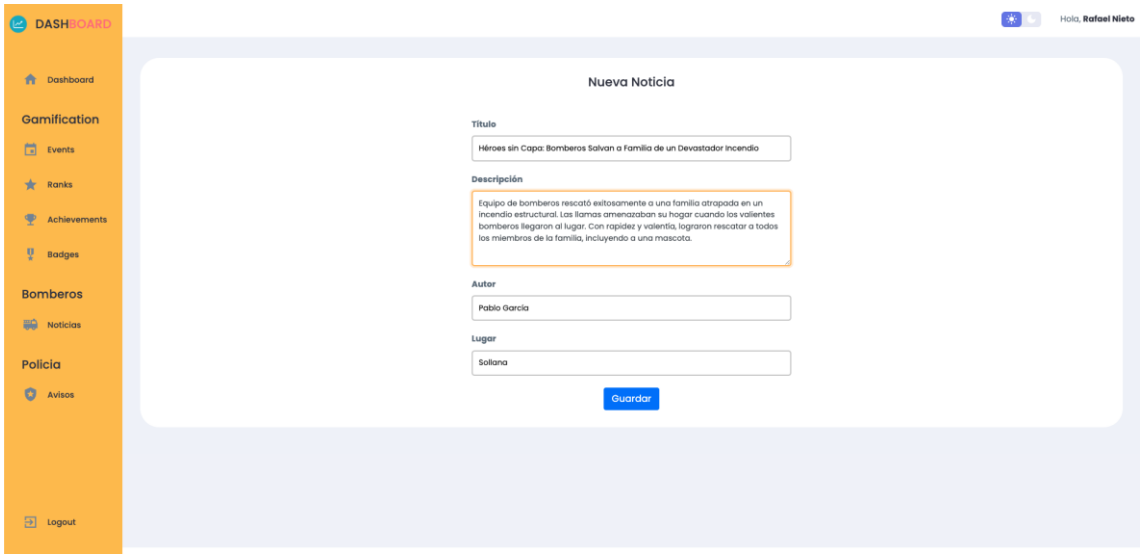


Ilustración 94 - Formulario para crear un objeto Post

Al hacer click en el botón de guardar, la información es enviada al microservicio “bomberos-api”, y posteriormente a la base de datos:

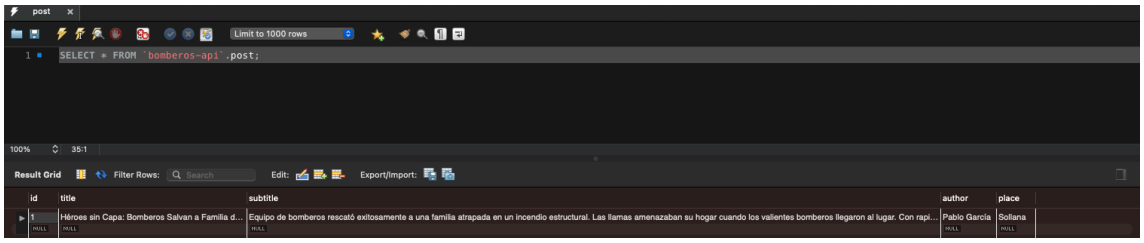


Ilustración 95 - Base de datos de la entidad Post con un objeto creado

A continuación, esta información es leída, de nuevo, por el microservicio “bomberos-api”, y mostrada tanto en la página web de los bomberos como en la tabla de gestión del dashboard:

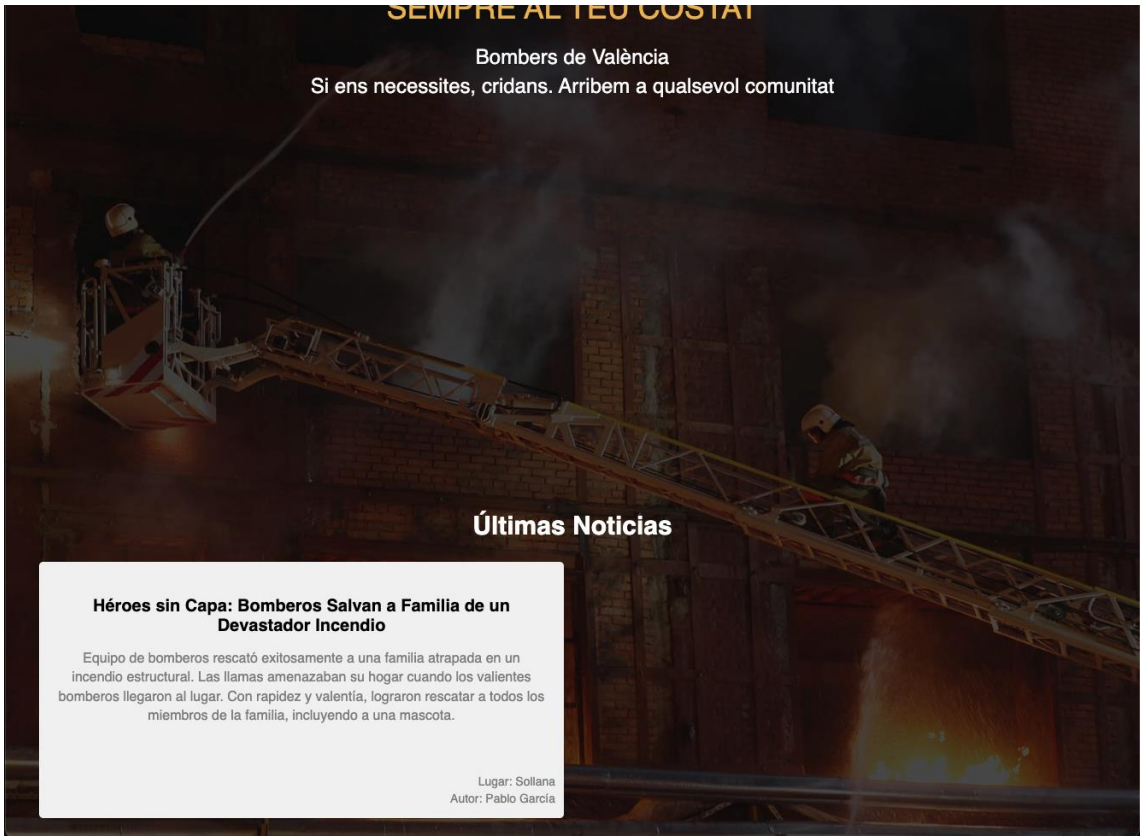


Ilustración 96 - Objeto Post creado desde el dashboard reflejado en la página web de los bomberos

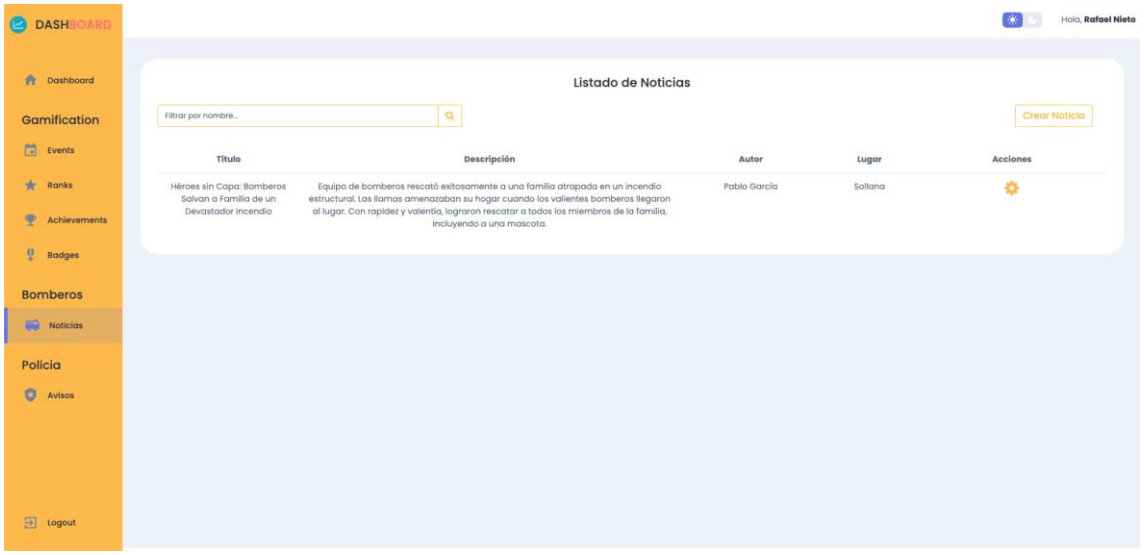


Ilustración 97 - Tabla de gestión de la entidad Post

En la página web de la policía es posible utilizar el dashboard de la misma forma para crear avisos desde el apartado Avisos:

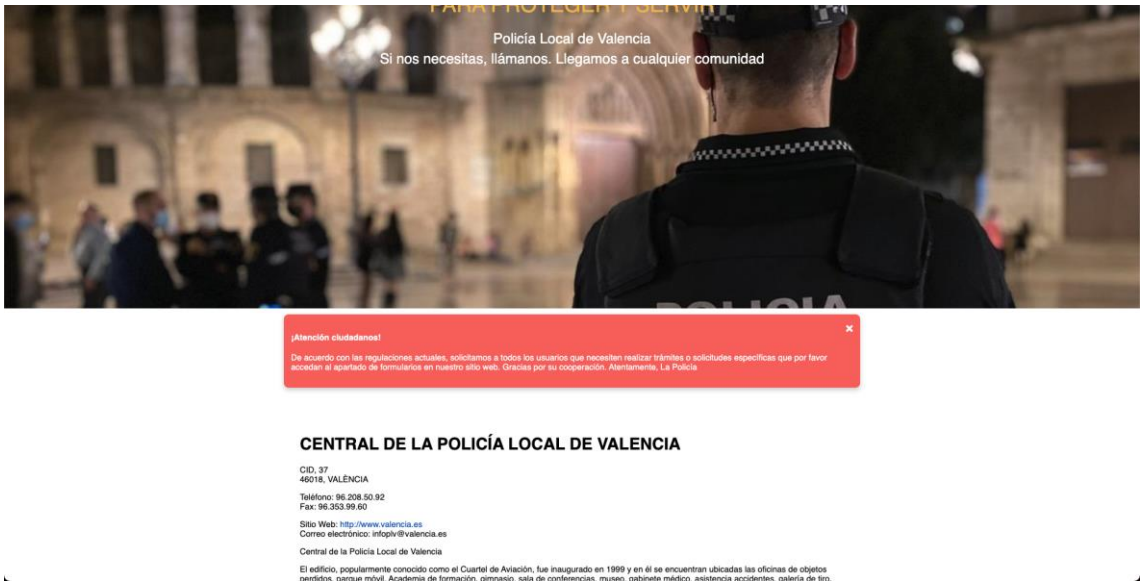


Ilustración 98 - Objeto Post creado desde el dashboard reflejado en la página web de los bomberos

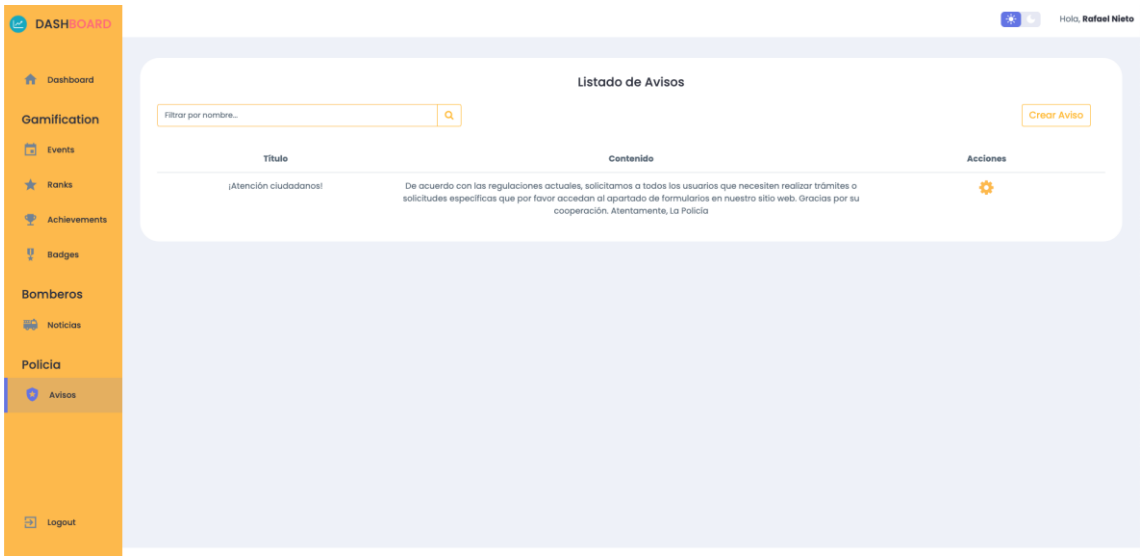


Ilustración 99 - Tabla de gestión de la entidad Notification

Siguiendo un proceso similar, se puede apreciar que al crear un objeto Event, entidad propia del plugin de WordPress, los cambios quedan reflejados en la tabla del portal de WordPress:

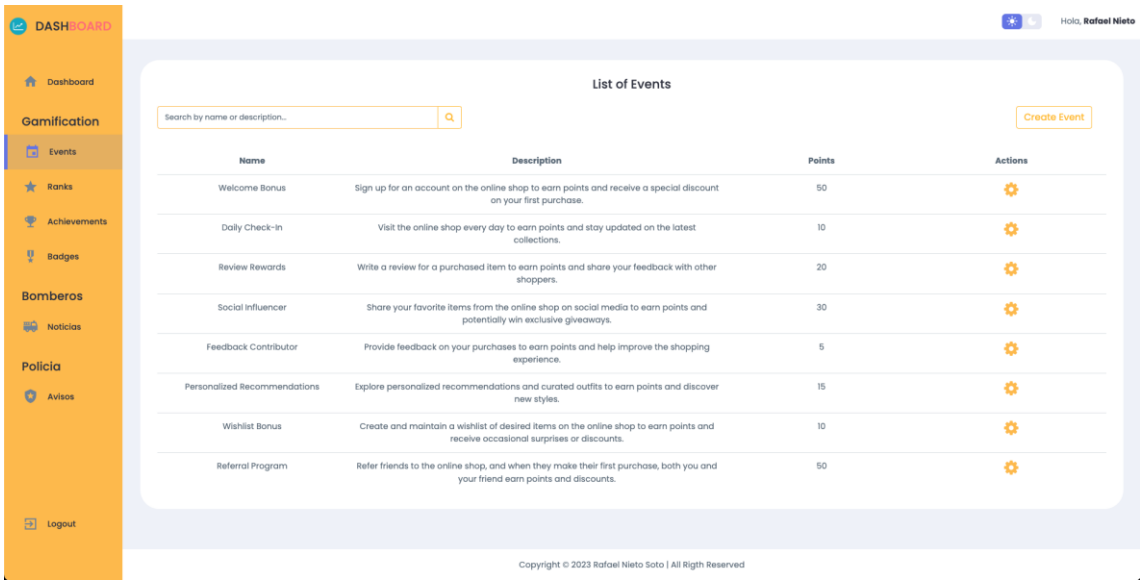


Ilustración 100 - Tabla de gestión poblada de la entidad Event en el dashboard

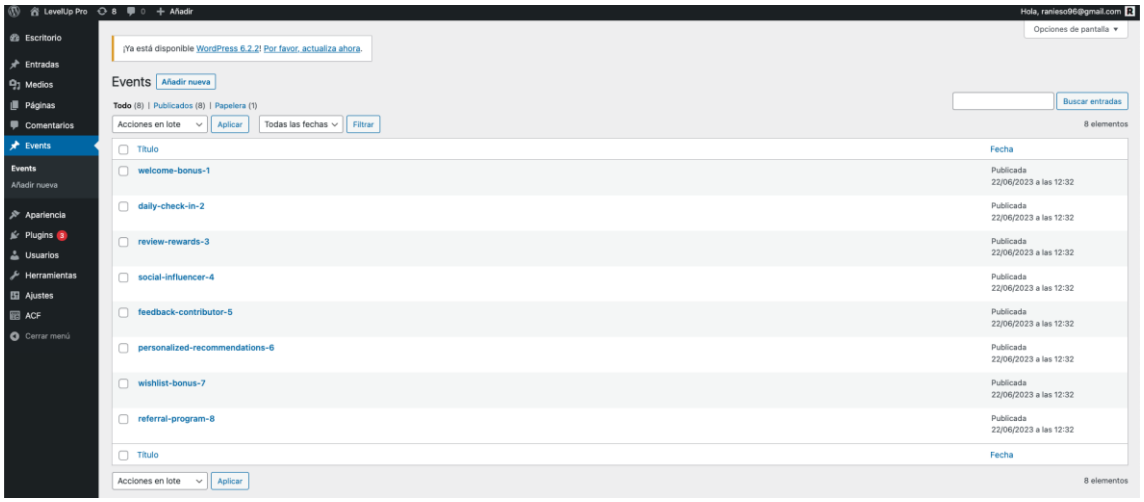


Ilustración 101 - Tabla de gestión poblada de la entidad Event en el plugin

Efectivamente hemos comprobado que los cambios han quedado reflejados entre diferentes sitios webs. De igual forma que se comprobado la creación y lectura de datos, también se ha comprobado que los datos se puedan eliminar sin dejar rastros en la base de datos. Esto significa que, al crear un objeto tras haber borrado otro, el nuevo objeto puede heredar el identificador $id del objeto eliminado, en lugar de generarse uno nuevo.

7. Conclusiones

El desarrollo desde cero de esta herramienta ha sido una forma de adquirir conocimientos que de otra forma nunca hubiera aprendido. Ha sido una muy buena práctica el analizar cómo funcionan herramientas similares, y de conocer qué hay detrás de cada tecnología, en lugar de utilizarlas a ciegas sin más. Considero que la práctica de dedicar un tiempo inicial al análisis y estudio de la tecnología antes de proceder al desarrollo debería hacerse siempre, puesto que, en caso de que algo falle por un error de planificación, el tiempo perdido en rectificarlo podría ser mayor que el invertido en la propia planificación.

Ya que este TFG ha sido mi primer gran proyecto desarrollado desde cero, sin contar pequeñas aplicaciones que he ido desarrollando a modo de hobbie, han surgido bloqueos y dificultades que han alargado el proceso, aunque esto es algo que siempre se supone que pueda pasar. La mayoría de estos bloqueos surgieron cuando, tras realizar algún cambio en la configuración del proyecto, los endpoints dejaban de funcionar y debía dedicar un tiempo para resolver algo que anteriormente ya estaba funcionando. Otra parte problemática del proyecto ha sido cómo utilizar la base de datos. En un principio intenté utilizar extensiones de VSCode, pero acababa con una base de datos vacía por motivos que desconozco. Tras ver que no se mostraba información en la interfaz del dashboard, pensé que el problema estaría relacionado, una vez más, con los endpoints, teniendo así que invertir más tiempo en buscar la causa del problema.

Por otro lado, al principio estuve dudando en si me estaba siendo efectivo invertir un tiempo extra en controlar las versiones del código mediante Bitbucket y Git. A mediados del desarrollo del proyecto, cambié la configuración radicalmente y entré en un bucle de errores el cuál no conseguí arreglar, y fue entonces cuando poder volver a una versión anterior me fue realmente útil.

Sobre la metodología Scrum empleada, ha sido una decisión más que acertada. El poder planificarme durante los viajes en los fines de semana el trabajo que debería realizar la semana próxima entre semana, ha encajado perfectamente con mi rutina semanal. Me ha permitido, además de trabajar de una forma más organizada al dividir el trabajo en tareas, mejorar mi gestión del tiempo, evitando tener días con demasiadas cosas que hacer y otros días con prácticamente nada.

En definitiva, tras haber desarrollado este proyecto, puedo decir que han aumentado en gran medida mis habilidades en la programación y en la ingeniería de sotfware, y lo considero un muy buen primer proyecto personal por ello.

7.1 Relación del trabajo desarrollado con los estudios cursados

A lo largo de la carrera, el lenguaje que más se ha utilizado ha sido Java, convirtiéndose mi primer lenguaje de programación. Aunque no he utilizado dicho lenguaje, me ha resultado sencillo aprender otras tecnologías, ya que Java proporciona una base sólida en conceptos fundamentales de programación, como la orientación a objetos, la gestión de memoria y la estructura de control. Estos conceptos son aplicables en muchos otros lenguajes, por lo que resulta muy sencillo la transición. Además, tiene una sintaxis muy verbosa, lo que permite entender y usar de forma efectiva tecnologías que permiten una sintaxis mucho más resumida.

Al haber cursado la rama de Sistemas de la Información, uno de los aspectos más estudiados ha sido las bases de datos, cómo deberían estar construidas y conectadas, y formas de hacer sus llamadas más eficientes. Esto ha sido útil para tener una idea sobre estos aspectos, como claves primarias y ajenas, o por qué una tabla debería o no tener según qué atributos.

Por otro lado, asignaturas como Redes y Sistemas distribuidos me han ayudado a tener una base en cómo deberían funcionar las REST APIs. Este aspecto de la programación es muy general y se puede hacer de muchas formas diferentes, aun así, ha sido importante conocer lo básico sobre este tema antes de empezar a analizarlo.

En conclusión, los estudios cursados me han aportado una base general para ser capaz de enfrentar cualquier tipo de proyecto en un futuro, tanto al desarrollar el código como en la fase inicial de planificación y organización.

8. Trabajos futuros

A lo largo de esta memoria se ha presentado el análisis y desarrollo inicial de esta herramienta, además de dos casos de uso basados en la resolución de la gestión de servicios públicos de Valencia y en un plugin de WordPress. Esto no significa que haya consistido en una herramienta con el único propósito de exponerla ante la universidad, sino que, como ya se ha explicado en otros apartados, desde un primer momento se ha tomado la decisión de hacerla lo más escalable y flexible posible con la idea de ir modificándola a lo largo del tiempo.

Lo que pretendo hacer con esta herramienta es centralizar todos mis proyectos personales de forma que pueda controlarlos desde el dashboard, ahorrándome así el tener que ir de aplicación en aplicación realizando cambios.

A modo de ejemplo de lo dicho, y como afición personal, soy un gran amante del mundo del fitness y le doy gran importancia al nutricionismo, lo cual, sumado a mis conocimientos y pasión por la programación, me han llevado a querer desarrollar dos aplicaciones de uso personal las cuales utilizaré en dichos campos. Suponiendo que utilice la aplicación de gimnasio para controlar mi rutina de cada día, lo que pretendo conseguir es poder modificar los ejercicios de cada día junto con los tiempos y series desde el dashboard. De igual forma, debería poder mostrar en la aplicación de nutricionismo dietas personalizadas, calculando y mostrando las cantidades desde el dashboard.

9. Bibliografía

[1] Web oficial de WordPress: https://wordpress.com/ [Online, Junio 2023]

[2] Web oficial de Jira: https://www.atlassian.com/es/software/jira [Online, Junio 2023]

[3] Web oficial de Tableau: https://www.tableau.com/ [Online, Junio 2023]

[4] Web oficial de Power BI: https://powerbi.microsoft.com/ [Online, Junio 2023]

[5] Web oficial de Zoho Analytics: https://www.zoho.com/analytics/ [Online, Junio 2023]

[6] Web oficial de QlikView: https://www.qlik.com/es-es/products/qlikview [Online, Junio 2023]

[7] Web oficial de Visual Studio Code: https://code.visualstudio.com/ [Online, Junio 2023]

[8] Web oficial de Bitbucket: https://bitbucket.org/ [Online, Junio 2023]

[9] Web oficial de Git: https://git-scm.com/ [Online, Junio 2023]

[10] Documentación oficial de PHP: https://www.php.net/docs.php [Online, Junio 2023]

[11] Documentación oficial de Symfony: https://symfony.com/doc/current/index.html [Online, Junio 2023]

[12] Documentación oficial de Twig: https://twig.symfony.com/doc/ [Online, Junio 2023]

[13] Documentación oficial de Api Platform: https://api-platform.com/docs/ [Online, Junio 2023]

[14] Documentación oficial de Composer: https://getcomposer.org/doc/ [Online, Junio 2023]

[15] Web oficial de MySQLWorkbench: https://www.mysql.com/products/workbench/ [Online, Junio 2023]

[16] Documentación oficial de Encore: https://symfony.com/doc/current/frontend/encore/simple-example.html [Online, Junio 2023]

[17] Documentación oficial de Yarn: https://yarnpkg.com/getting-started [Online, Junio 2023]

[18] Documentación oficial de Symfony Form: https://symfony.com/doc/current/forms.html [Online, Junio 2023]

[19] Web de código libre de la interfaz de login: https://freefrontend.com/css-login-forms/ [Online, Junio 2023]

10. Anéxo

Objetivos de desarrollo sostenible

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades.**				**X**
ODS 11. **Ciudades y comunidades sostenibles.**			**X**	
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres**				**X**
ODS 16. **Paz, justicia e instituciones sólidas**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

La herramienta desarrollada a lo largo de esta memoria podría ser utilizada de diversas formas diferentes, cumpliendo con unos ODS u otros en función de qué empresa esté al cargo. En el estado actual del proyecto, existe relación directa con 2 ODS:

- **ODS 9 - Industria, Innovación e Infraestructura:** Dentro del contexto de este ODS, este TFG se relaciona con la promoción de la industria ya que busca mejorar la infraestructura tecnológica, fomentar la innovación y facilitar la gestión en el contexto de las páginas web. Estos aspectos son fundamentales para el desarrollo de una industria sólida y competitiva en el ámbito digital.
- **ODS 11 – Ciudades y comunidades sostenibles:** Este proyecto se relaciona con la creación de comunidades más sostenibles al mejorar la eficiencia y calidad de los servicios públicos ofrecidos a través de estas páginas web en el caso de uso de este TFG. Al desarrollar una herramienta que permita gestionar servicios públicos en una ciudad, como los servicios de bomberos y la policía local, la herramienta podría tener un impacto directo en el bienestar de la comunidad y en la calidad de vida de sus habitantes.