



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Aplicación web progresiva para la visualización de entornos inteligentes y actividades cotidianas

Autor

Marco Antonio Rodríguez Molina

Directores

Javier Medina Quero
Aurora Polo Rodríguez



Escuela Técnica Superior de Ingenierías
Informática y de Telecomunicación

Granada, Septiembre de 2024

Aplicación web progresiva para la visualización de entornos inteligentes y actividades cotidianas

Marco Antonio Rodríguez Molina

Palabras clave: Internet de las Cosas (IoT), PWA, Angular, Flask, ElasticSearch, sensores, usuarios, visualización de datos, Grafana, aplicación, web.

Resumen:

Este proyecto presenta el desarrollo de una aplicación web progresiva (PWA) encargada de la recogida y tratamiento de datos procedentes de sensores en entornos inteligentes. La solución final se centra en la integración y visualización eficiente, clara y sencilla de la información recolectada, diferenciando entre varios usuarios y distintas ubicaciones de los sensores.

Para el correcto manejo de los datos, la base de datos seleccionada ha sido Elasticsearch, dada su capacidad para gestionar grandes volúmenes de información, así como para realizar búsquedas rápidas. Además, esta elección ofrece una ventaja significativa para la visualización de los datos debido a su conexión directa con Grafana.

El backend del sistema se ha desarrollado en Flask dada su sencillez y robustez para este tipo de proyectos, puesto que facilita la creación de API RESTful para la comunicación con el frontend.

La parte del frontend del proyecto se ha construido con Angular ya que ofrece una interfaz de usuario dinámica que facilita la interacción con los datos. La visualización en tiempo real se realiza tal y como se mencionó previamente, proporcionando a los usuarios una forma clara y accesible de monitorizar los datos recopilados.

Este proyecto propone una solución lo más adaptable posible a distintos tipos de pantallas, manejando y mostrando los datos de una forma fácilmente comprensible para todo tipo de usuarios.

Progressive web Application for visualizing smart environments and everyday activities

Marco Antonio Rodríguez Molina

Keywords: Internet of Things (IoT), PWA, Angular, Flask, ElasticSearch, sensors, users, data visualization, Grafana, web application.

Abstract:

This project presents the development of a progressive web application (PWA) aimed at collecting and processing data from sensors in smart environments. The final solution focuses on the efficient, clear, and simple integration and visualization of the collected information, distinguishing between multiple users and different sensor locations.

For proper data management, ElasticSearch was chosen as the database due to its capacity to handle large volumes of information and perform quick searches. Additionally, this choice offers a significant advantage for data visualization thanks to its direct connection with Grafana.

The system's backend was developed using Flask, given its simplicity and robustness for projects of this kind, as it facilitates the creation of RESTful APIs for communication with the frontend.

The frontend of the project was built using Angular since it provides a dynamic user interface that enhances interaction with the data. Real-time visualization is implemented as mentioned earlier, offering users a clear and accessible way to monitor the collected data.

This project aims to offer a highly adaptable solution to different screen types, managing and displaying data in a way that is easily understandable for all types of users.

Yo, **Marco Antonio Rodríguez Molina**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76665202S, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Marco Antonio Rodríguez Molina

Granada a 6 de Septiembre de 2024 .

D. Javier Medina Quero, Profesor del Área del Departamento de Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

Dra. Aurora Polo Rodríguez, Profesora del Área del Departamento de Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Aplicación web progresiva para la visualización de entornos inteligentes y actividades cotidianas**, ha sido realizado bajo su supervisión por **Marco Antonio Rodríguez Molina**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 6 de Septiembre de 2024.

Los directores:

Javier Medina Quero

Aurora Polo Rodríguez

Índice general

1. Introducción	1
1.1.- ¿Qué es una PWA?	1
1.2.- ¿Por qué se ha elegido para este proyecto?.....	1
2. Fases del proyecto.....	2
2.1.- Fase de inicio del proyecto	2
2.2.- Fase de Análisis de requisitos	2
2.3.- Fase de Investigación.....	3
2.4.- Fase de Diseño	3
2.4.1.- Diseño de la Arquitectura del Sistema	4
2.4.2.- Diseño de la Interfaz de Usuario	6
2.4.3.- Diseño de la API	8
2.4.4.- Integración con Grafana	9
2.5.- Fase de implementación.....	9
2.5.1.- Desarrollo del Frontend (PWA en Angular)	9
2.5.2.- Desarrollo del Backend (API en Flask).....	10
2.5.3.- Integración de Grafana.....	10
3. Tecnologías Usadas	12
3.1.- Software empleado.....	12
3.2.- Materiales.....	14
3. 3.- Comunicación	16
3.4.- Documentación	16
4. Diagramas de la aplicación final.....	18
4.1.- Diagrama de la estructura (en carpetas) final del proyecto	18
4.2.- Diagrama de flujo de la PWA	18
4.3.- Caso de Uso de la PWA.....	19
4.4.- Diagrama de secuencia	20

5. Backend.....	21
5.1.- Configuración inicial	22
5.2.- Conexión con la BD.....	23
5.3.- Endpoints	23
5.4.- Requirements.txt.....	25
5.5.- Puerto y lanzamiento.....	25
6. Frontend	26
6.1.- Manifest.webmanifest.....	26
6.2.- Index.html y styles.css	27
6.3.- Main.ts	28
6.4.- Componentes propios de la PWA	30
6.4.1.- app.component.ts.....	30
6.5.- Archivos HTML.....	37
6.5.1.- app.component.html	37
6.5.2.- app.componentIndex.html	38
6.5.3.- app.componentNovedades.html	39
6.5.4.- app.componentListUser.html	40
6.5.5.- app.componentUser.html	41
6.5.6.- app.componentSensores.html	43
6.6.- Archivo CSS	44
6.7.- Otros archivos	47
6.7.1.- angular.json	47
6.7.2.- proxy.conf.json.....	47
6.7.3.- ngsw-worker.js	47
6.7.4.- ngsw-config.json	47
6.8.- Comandos usados y puerto	48
7. Grafana.....	49

7.1.- Instalación, primeros pasos, conexión y dashboard.....	49
7.2.- Alertas	51
8. ElasticSearch.....	53
9. Problemas surgidos	56
9.1.- Problema HTTPS	56
9.2.- Problema con implementación JWT	56
10. Conclusiones	58
11. Trabajos futuros	60
11.1.- Fases futuras a desarrollar.....	60
11.1.1.- Fase de Despliegue.....	60
11.1.2.- Fase de Pruebas	61
11.1.3.- Fase de Mantenimiento	61
11.2.- Implementaciones adicionales	62
12. Bibliografía	64
13. Anexo - Icono PWA y Capturas de la Web y PWA	70

Índice de figuras

Figura 1 - Modelo Cliente-Servidor.....	4
Figura 2 - Esquema Patrón MVC.....	5
Figura 3 - Boceto diseño previo de la PWA	8
Figura 4 - Logo Python	12
Figura 5 - Logo Flask.....	12
Figura 6 - Logo Angular	12
Figura 7 - Logo Angular CLI.....	13
Figura 8 - Logo TypeScrip.....	13
Figura 9 - Logo HTML5	13
Figura 10 - Logo CSS	13
Figura 11 - Logo JSON.....	14
Figura 12 - Logo ElasticSearch.....	14
Figura 13 - Logo Visual Studio Code.....	14
Figura 14 - Logo DevTools.....	15
Figura 15 - Logo Grafana	15
Figura 16 - Logo PlantUML	15
Figura 17 - Logo WhatsApp	16
Figura 18 - Logo GoogleMeet	16
Figura 19 - Logo Google Docs	17
Figura 20 - Logo Microsoft Word	17
Figura 21 - Organigrama de la estructura final	18
Figura 22 - Diagrama de flujo.....	18
Figura 23 - Caso de Uso.....	19
Figura 24 - Diagrama de secuencias	20
Figura 25 - Diagrama E/R de la BD.....	21
Figura 26 - Esquema básico Flask (Backend).....	22
Figura 27 - Línea de Código conexión con ElasticSearch	23
Figura 28 - Ejemplo de estructura de un endpoint.....	23
Figura 29 - Captura de ejemplo del contenido endpoint "sensores"	24
Figura 30 - Contenido Requirements.txt.....	25
Figura 31 - Organización de carpetas de frontend	26
Figura 32 - Ejemplo de parte del contenido del manifest	27

Figura 33 - Index.html	27
Figura 34 - styles.css.....	28
Figura 35 - Ejemplo de imports en main.ts.....	28
Figura 36 - Ejemplo de rutas en main.ts	29
Figura 37 - Definición de NgModule.....	29
Figura 38 - Registro del ServiceWorker	30
Figura 39 - Bootstrap del main.ts.....	30
Figura 40 - Diagrama de clases de los componentes del frontend.....	31
Figura 41 - Ejemplo de declaración de un componente.....	32
Figura 42 - Ejemplos de cabecera de clase	32
Figura 43 - Ejemplo declaración de variables de una clase	33
Figura 44 - Ejemplo de función dentro de una clase	33
Figura 45 - Ejemplo de constructor de clase.....	33
Figura 46 - Ejemplo de implementación de ngOnInit	33
Figura 47 - Ejemplo de ngOnInit con función externa	34
Figura 48 - Contenedor de botones de la cabecera	38
Figura 49 - Contenedor main	38
Figura 50 - Ejemplo de botón condicional: Novedades.....	39
Figura 51 - Estructura con condicional : Si no hay alertas a mostrar	40
Figura 52 - Lógica que muestra los botones de "Usuario X" en el menú	41
Figura 53 - Estructura que asegura que haya contenido a mostrar	41
Figura 54 - Menú desplegable de actividades.....	42
Figura 55 - Menú vertical (listado) de actividades	42
Figura 56 - Iframe de actividad.....	42
Figura 57 - Menú desplegable de diferentes tipos de sensor	43
Figura 58 - Mensaje de carga.....	44
Figura 59 - Ejemplo de uso media queries.....	45
Figura 60 - Display: none	45
Figure 61 - Grid de la cabecera y pie de página	45
Figura 62 - Ejemplo del estilo de los botones.....	46
Figura 63 - Grid de la estructura general	46
Figura 64 - Conexión BD en Grafana	49
Figura 65 - Información relevante en conexión con Grafana	50
Figura 66 - Error con MongoDB	50

Figura 67 - Comparativa entre archivo final JSON e inicial .tsv.....	51
Figura 68 - Conexión Webhook para alertas	52
Figura 69 - Ejemplo de consulta: Tipos de sensores.....	53
Figura 70 - Extraído del archivo de sensores	54
Figura 71 - Extraído del archivo del Usuario 1.....	55
Figura 72 - Ejemplo de inclusión de certificados en Flask	56
Figura 73 - Ejemplo de uso de JWT en Flask.....	57
Figura 74 - Icono de la aplicación.....	70
Figura 75 - Captura que muestra la instalación de la PWA	70
Figura 76 - Captura que muestra la aplicación PWA instalada en nativo	71
Figura 77 - Captura que muestra la vista desde la aplicación PWA	71
Figura 78 - Dispositivo móvil: Capturas de la página de inicio con notificación alerta y sin ella	72
Figura 79 - Dispositivo móvil: Captura de la página de novedades donde se muestra el dashboard que ha lanzado la alerta	73
Figura 80 - Dispositivo móvil: Captura de pantalla que muestran las páginas de usuarios y sensores	73
Figura 81 - Dispositivo móvil: Captura que muestra la página de sensores donde se ve la cabecera.....	74
Figura 82 - Monitor: Captura de pantalla página de inicio sin notificación	74
Figura 83 - Monitor: Captura de pantalla página de inicio con notificación	75
Figura 84 - Monitor: Captura de la página de novedades	75
Figura 85 - Monitor: Captura del menú de selección de usuarios	76
Figura 86 - Monitor: Captura de la página de actividad de Usuario 1.....	76
Figura 87 - Monitor: Captura de la página de los sensores.....	77

Índice de tablas

Tabla 1 - Requisitos funcionales y no funcionales	3
Tabla 2 - MVC de Backend y Frontend.....	6

1. Introducción

1.1.- ¿Qué es una PWA?

Una PWA (lebschool.com, 2024) recibe las siglas de su nombre en inglés, Progressive Web Application, aplicación web progresiva en español, es un tipo de aplicación que combina lo mejor de las aplicaciones web y las aplicaciones nativas. A la hora de funcionar lo hace como una página web en su núcleo, pero ofrece funcionalidades avanzadas que permiten una experiencia similar a la de una aplicación nativa, como la capacidad de trabajar sin conexión, recibir notificaciones push, y ser instalada directamente en el dispositivo sin necesidad de pasar por un gestor de aplicaciones.

1.2.- ¿Por qué se ha elegido para este proyecto?

Se decidió realizar este proyecto por las ventajas que proporciona.

- Fácil de usar.
- Se puede instalar en cualquier dispositivo de forma rápida.
- Funciona sin internet
- Es una opción ligera y rápida (menor tiempo de carga que apps normales)
- No requiere actualizaciones. Siempre tendrá el usuario la versión más nueva.
- Ofrece una visión integral de todos los componentes de un proyecto de informática.

Así pues, en base a esto, esta opción es perfecta para este proyecto porque los usuarios pueden ver los datos de los sensores y las actividades sin complicaciones, como si fuera una aplicación nativa, pero sin las desventajas de las apps tradicionales.

2. Fases del proyecto

2.1.- Fase de inicio del proyecto

El objetivo marcado en esta fase es consensuar con los tutores el alcance y los objetivos generales del proyecto.

Tras una reunión inicial online con el tutor:

- Se concreta que el desarrollo que se va a llevar a cabo será una PWA.
- Se debaten varias opciones sobre los lenguajes a utilizar.
- Marcamos las directrices a seguir.
- Se pone en marcha una fase previa de investigación sobre los lenguajes posibles.

2.2.- Fase de Análisis de requisitos

Definir tanto requisitos funcionales como no funcionales del sistema.

NOMBRE REQUISITO	DESCRIPCIÓN
FUNCIONALES	
F1	La aplicación debe mostrar un listado de usuarios con acceso a información detallada de cada usuario.
F2	La PWA debe permitir la visualización de los datos de sensores en tiempo real utilizando visualizaciones de Grafana.
F3	La aplicación debe permitir la selección y visualización de actividades específicas para cada usuario.

F4	El sistema debe permitir la gestión de sensores, incluyendo la selección y visualización de datos específicos de cada sensor.
F5	La PWA debe notificar a los usuarios sobre alertas o novedades relacionadas con los datos de los sensores.
F6	La PWA debe integrar un dashboard donde los datos de los sensores se muestren en tiempo real utilizando iframes de Grafana.
F7	El usuario debe poder seleccionar un sensor para ver más detalles.
NO FUNCIONALES	
NF1	La PWA debe ser compatible con los navegadores modernos (Chrome, Firefox, Safari, Edge).
NF2	La aplicación debe cargar rápidamente (igual tiempo que si se abriese en la web)
NF3	La interfaz de usuario debe ser responsive, adaptándose a diferentes tamaños de pantalla (móvil, tablet, desktop).
NF4	La base de datos en ElasticSearch debe ser capaz de manejar consultas de gran volumen de datos de forma eficiente.
NF5	La PWA debe ser fácil de usar

Tabla 1 - Requisitos funcionales y no funcionales

2.3.- Fase de Investigación

Durante la fase de investigación del proyecto, se dedicó tiempo a explorar y analizar las

mejores opciones en cuanto a tecnologías, lenguajes de programación y herramientas que podrían ser utilizadas en el desarrollo de una aplicación web progresiva (PWA) orientada a la visualización de datos de sensores y usuarios.

En primer lugar, se investigaron qué lenguajes de programación y frameworks eran más adecuados para desarrollar una PWA. Se priorizaron opciones que ofrecieran una buena integración con bases de datos NoSQL, como MongoDB o ElasticSearch, y que permitieran la creación de una interfaz de usuario moderna y responsive.

Frontend: Se exploraron diversas opciones para el desarrollo del frontend, incluyendo React, Vue.js y Angular. Finalmente, la elección fue Angular debido a su robustez, capacidad para manejar aplicaciones complejas, ya que se encargará de la lógica del cliente, gestión del estado, y comunicación con el backend a través de API REST, y sobre todo por su gran soporte para PWAs. Angular también cuenta con una gran comunidad y documentación, lo cual facilita el proceso de desarrollo y resolución de problemas.

Backend: En cuanto al backend, estaban sobre la mesa varios lenguajes como Node.js, Python y Ruby. Aunque se optó por Python utilizando el framework Flask debido a su simplicidad, flexibilidad y la facilidad con la que se puede integrar con cualquier base de datos. Flask es ligero, lo cual es ideal para proyectos que no requieren un backend demasiado complejo, pero que necesitan una estructura clara y manejable. Se encargará de manejar las solicitudes del frontend, procesar la lógica y gestionar las interacciones con la base de datos en ElasticSearch.

Base de Datos: Para la base de datos, se eligió MongoDB inicialmente, pero dada la compleja conexión que tenía con Grafana, finalmente fue ElasticSearch ya que también tiene una alta capacidad para manejar grandes volúmenes de datos y realizar búsquedas complejas de manera eficiente, gracias a que las bases de datos se basan en índices. Además, ElasticSearch es particularmente útil en proyectos que requieren la indexación y búsqueda de datos en tiempo real, como en el caso de los sensores.

Grafana: Se trata de una solución popular para la visualización de métricas que se integra bien con diversas bases de datos, incluyendo ElasticSearch (MongoDB requería de instalaciones de plugins independientes y no estaba completamente integrado). Grafana es ampliamente utilizada en proyectos de monitoreo y me proporcionó una referencia clara sobre cómo debía estructurar las visualizaciones de datos en el proyecto. Previamente no

se había tenido contacto con esta herramienta lo que supuso un estudio e investigación adicional hasta familiarizarse con ella.

En definitiva, después de analizar las opciones disponibles y las necesidades específicas del proyecto, se determinó usar Angular para el frontend, Flask con Python para el backend, y ElasticSearch como base de datos.

La combinación de estas tecnologías no solo se adapta perfectamente a las necesidades específicas de este proyecto, sino que también ofrece múltiples ventajas en términos de escalabilidad, mantenimiento y rendimiento.

- En cuanto a la escalabilidad, la arquitectura planteada permite manejar grandes volúmenes de datos y usuarios simultáneamente sin comprometer la velocidad ni la estabilidad del sistema. Sumado a esto, ElasticSearch, es ideal para escalar horizontalmente, lo que garantiza que, a medida que crezca el número de sensores o la cantidad de datos recolectados, el sistema pueda continuar funcionando de manera eficiente sin pérdidas de rendimiento.
- Desde la perspectiva del mantenimiento, las tecnologías seleccionadas, como Flask y Angular, son bien conocidas por la comunidad de desarrollo, lo que facilita su actualización y la incorporación de mejoras o nuevas funcionalidades en el futuro. Además, la modularidad de estas tecnologías simplifica la identificación y resolución de posibles problemas, permitiendo un mantenimiento más ágil y efectivo.
- Finalmente, en términos de rendimiento, el uso de una base de datos optimizada para búsquedas rápidas como ElasticSearch, junto con la capacidad de visualización en tiempo real de Grafana, asegura una experiencia de usuario fluida y receptiva, incluso bajo condiciones de carga intensiva. Esta combinación tecnológica no solo responde a las demandas actuales del proyecto, sino que también lo prepara para futuros retos y expansiones, garantizando una solución sólida, eficiente y preparada para el crecimiento a largo plazo.

2.4.- Fase de Diseño

La fase de diseño del proyecto fue crucial para establecer cómo se implementaría la

solución final. Durante esta fase, se definió la estructura general del sistema, se diseñaron los componentes clave, y se planificó la integración entre las diferentes tecnologías utilizadas.

Para este fin, lo primero que se hizo fue una sencilla PWA que mostrase el típico “Hola Mundo” donde la información circularía desde la BD al backend y desde este, mediante un endpoint, al frontend.

Partiendo de esta primera PWA se ha ido incrementando hasta el resultado final.

2.4.1.- Diseño de la Arquitectura del Sistema

2.4.1.1 Cliente-Servidor

El primer paso en la fase de diseño fue establecer la arquitectura general del sistema. Dado que el proyecto implica el desarrollo de una aplicación web progresiva (PWA) con un backend en Flask y una base de datos en ElasticSearch, se optó por una arquitectura cliente-servidor. Esta arquitectura permite una clara separación entre el frontend y el backend, facilitando el mantenimiento y la escalabilidad del sistema.

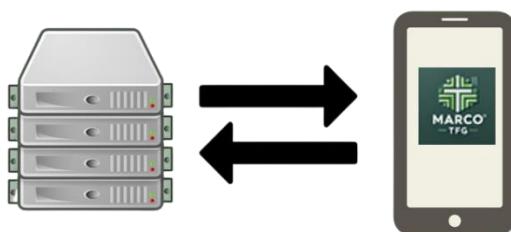


Figura 1 - Modelo Cliente-Servidor

2.4.1.2 Patrón Modelo-Vista-Controlador

Respecto a la lógica interna, el proyecto sigue el patrón Modelo-Vista-Controlador (MVC), que proporciona una estructura clara para gestionar la interacción entre las distintas partes de la aplicación. Este modelo se aprecia en el proyecto tanto en el backend como en la parte del frontend.

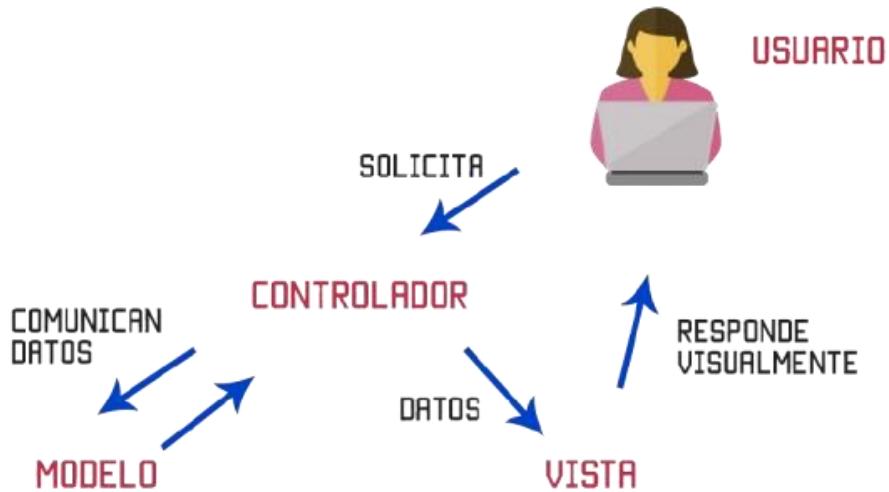


Figura 2 - Esquema Patrón MVC

MVC del Backend

Modelo: está representado por la base de datos Elasticsearch, que maneja la persistencia y almacenamiento de datos que son recuperados a través de peticiones HTTP gestionadas por Flask. Esto encapsula la lógica relacionada con los datos, haciendo que las demás capas solo interactúen con el modelo a través del controlador.

Vista: La responsabilidad de la vista está completamente delegada al frontend. Flask se limita a servir los datos que la vista en Angular utiliza para generar la interfaz.

Controlador: en Flask es el archivo app.py, que es el puente entre el frontend y el modelo (ElasticSearch). Se encarga de recibir las solicitudes, procesarlas y devolver los datos necesarios, manejando toda la lógica de negocio del lado del servidor.

MVC del Frontend

Modelo: los servicios en Angular, como por ejemplo HttpClient, cumplen el rol del modelo porque son los encargados de manejar y transformar los datos provenientes del backend. Además, dota a los componentes los datos que estos necesitan para ser presentados.

Vista: se encuentra compuesta por las plantillas HTML y CSS que definen la interfaz de usuario. Archivos con extensión .html como es el caso de app.component.html, app.componentUser.html y app.componentSensores.html entre otros, son responsables de mostrar los datos de la aplicación en la pantalla, como los detalles de los sensores o la actividad de los usuarios. Dichas vistas, que están formateadas con un estilo concreto gracias a los archivos .css, se actualizan dinámicamente e incluyen visualizaciones embebidas de Grafana.

Controlador: en el frontend el controlador está en los archivos de TypeScript que son todos aquellos con extensión .ts de los componentes, en este proyecto concretamente sólo se ha usado app.component.ts. Este archivo contiene la gestión lógica de la interfaz de usuario. Escucha eventos del usuario, realiza llamadas al backend a través de los servicios, y luego actualiza los datos que se muestran en la interfaz.

Seguidamente se muestra una tabla que recoge, a modo resumen, la información detallada anteriormente.

	BACKEND	FRONTEND
MODELO	BD en ElasticSearch	Servicios de Angular
VISTA	No maneja vista, sólo envía datos al frontend	Plantillas HTML y CSS. Visualizaciones de Grafana
CONTROLADOR	Archivo Flask, app.py	Archivo app.component.ts

Tabla 2 - MVC de Backend y Frontend

2.4.2.- Diseño de la Interfaz de Usuario

El siguiente paso fue el diseño de la interfaz de usuario. Se crearon bocetos que representaban la apariencia de la PWA. El diseño de la interfaz de usuario se enfocó en la simplicidad y la usabilidad, asegurando que la aplicación sea intuitiva y fácil de usar tanto en dispositivos móviles como en computadoras de escritorio.

- Navegación: Se diseñó una estructura de navegación clara, con un menú principal

que permite a los usuarios acceder fácilmente a las secciones de usuarios, sensores y alertas.

- Visualización de Datos: Se diseñaron componentes para mostrar los datos de los sensores y usuarios utilizando gráficos integrados mediante iframes de Grafana. Esto permite a los usuarios visualizar datos en tiempo real directamente desde la PWA.
- Responsividad: La interfaz fue diseñada para ser responsive, adaptándose a diferentes tamaños de pantalla y dispositivos. Para ello, además se ha contado con hojas de estilo css, para hacer la experiencia del usuario totalmente adaptativa.

2.4.2.1 Iframe

Cabe resaltar, antes de continuar, el concepto iframe. Ya que se ha hecho mención anteriormente en la visualización de datos y será un concepto recurrente a lo largo del documento.

Iframe es la abreviatura de “inline frame” y se trata de un elemento HTML que permite incrustar contenido de otras fuentes dentro de la página actual. Es decir, como si fuese una ventana dentro de una página web.

Tiene varias ventajas como:

- Incorporación de contenido externo sin cargas en servidor.
- Aísla el contenido del resto de la página.
- Permite cargar contenido de diferentes orígenes.

Pero también presenta inconvenientes tales como:

- Problemas de seguridad, ya que si su uso no es correcto pueden hacer vulnerables a las webs.
- Es totalmente dependiente de la fuente externa.
- Compatibilidad limitada, puesto que algunos navegadores los bloquean.
- Presenta problemas de adaptación a pantallas pequeñas

Para el caso concreto de la PWA, su uso nos supone más ventajas que inconvenientes, puesto que posibilita la visualización e interacción de los paneles de Grafana como si se estuviesen viendo desde la fuente original. Además, los problemas de adaptación responsive no ha supuesto un gran problema y se ha solucionado con un gran resultado, como se verá a lo largo del documento.

2.4.2.2 Bocetos previos

Las imágenes que se muestran debajo muestran un boceto inicial de la forma en que se vería la aplicación.



Figura 3 - Boceto diseño previo de la PWA

2.4.3.- Diseño de la API

La API del backend en Flask se diseñó para manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) necesarias para gestionar los datos de usuarios y sensores. Además, se definieron endpoints específicos para integrar los datos de ElasticSearch con la visualización en Grafana.

- Endpoints RESTful: Se diseñaron endpoints RESTful para las operaciones básicas de la API. Cada endpoint fue documentado para asegurar que el frontend pudiera interactuar con el backend de manera eficiente y coherente.

- Seguridad: Se deberían haber implementado medidas de seguridad, como la autenticación de usuarios, para proteger el acceso a la API y asegurar que los datos sensibles estén resguardados.

2.4.4.- Integración con Grafana

Finalmente, se planificó la integración con Grafana para la visualización de datos en tiempo real. Grafana se configuró para conectarse con ElasticSearch, permitiendo crear dashboards que se pueden embeber en la PWA mediante iframes. Esto asegura que los usuarios puedan acceder a visualizaciones dinámicas y actualizadas de los datos de sensores directamente desde la aplicación.

La fase de diseño estableció una base sólida para el desarrollo del proyecto. Se definieron claramente los componentes y la interacción entre ellos, asegurando que el sistema final sea cohesivo, eficiente y fácil de mantener. Con un diseño bien planificado, el proyecto estaba preparado para pasar a la fase de implementación, donde las ideas y planes detallados durante esta etapa serán llevados a cabo.

2.5.- Fase de implementación

En esta etapa, se desarrollaron los diferentes componentes del sistema, siguiendo el diseño establecido en la fase anterior. El objetivo principal de esta fase fue construir un sistema que cumpliera con todos los requisitos funcionales y no funcionales definidos, asegurando que todas las partes trabajaran juntas de manera efectiva.

2.5.1.- Desarrollo del Frontend (PWA en Angular)

El primer paso en la implementación fue desarrollar el frontend de la aplicación utilizando Angular. Siguiendo lo establecido en la fase de diseño, se construyeron los componentes de la PWA.

- Estructura de Componentes: Se implementaron los componentes clave de la PWA, como la lista de usuarios, la visualización de datos de sensores, y las notificaciones de alertas. Cada componente fue diseñado para ser modular y reutilizable, facilitando la gestión del código.

- Rutas y Navegación: Se configuró el enrutamiento en Angular para manejar la navegación entre las diferentes vistas de la aplicación. Esto incluyó la implementación de rutas protegidas para garantizar que solo los usuarios autenticados pudieran acceder a ciertas partes de la PWA.
- Interfaz de Usuario: Se implementaron los estilos CSS y se añadieron interacciones para asegurar que la interfaz de usuario fuera responsiva y se adaptara a diferentes tamaños de pantalla. También se aseguraron las características PWA, como la capacidad de operar offline y la instalación en dispositivos móviles.

2.5.2.- Desarrollo del Backend (API en Flask)

Simultáneamente, se desarrolló el backend utilizando Flask, siguiendo el diseño de la API creada en la fase anterior.

- Configuración de Flask: Se configuró el entorno de Flask, incluyendo la instalación de dependencias y la configuración inicial del proyecto. Se crearon los archivos principales del proyecto, como app.py, donde se implementó la lógica de la aplicación.
- Implementación de Endpoints: Se desarrollaron los endpoints RESTful para manejar las operaciones CRUD sobre los datos de usuarios y sensores. Estos endpoints permitieron que el frontend se comunicara con el backend para realizar acciones como mostrar listas de usuarios, agregar nuevos sensores, o actualizar datos existentes.
- Integración con ElasticSearch: Se implementó la conexión entre Flask y ElasticSearch, permitiendo al backend realizar búsquedas y manipular los datos de los sensores de manera eficiente. Se realizaron pruebas para asegurar que los datos se almacenaran y recuperaran correctamente desde ElasticSearch.

2.5.3.- Integración de Grafana

Una parte fundamental de la implementación fue la integración con Grafana para la visualización de los datos de los sensores.

- Configuración de Dashboards: Se configuraron dashboards en Grafana conectados a la base de datos de ElasticSearch. Estos dashboards fueron diseñados para mostrar visualizaciones claras y concisas de los datos en tiempo real.
- Embebido de Dashboards: Se implementaron iframes en la PWA para embeber los dashboards de Grafana, permitiendo a los usuarios ver las visualizaciones directamente desde la aplicación. Se realizaron ajustes para asegurar que las visualizaciones fueran responsivas y se integraran bien con el diseño de la PWA.

La fase de implementación culminó en la creación de una PWA funcional que cumple con los requisitos del proyecto. El sistema fue construido de manera modular y escalable, permitiendo futuras extensiones o modificaciones con facilidad. Sin embargo, encontramos diferencias importantes entre el diseño previo y la implementación final debido a problemas ajenas a la implementación, pero sí relacionados con el proyecto. En un apartado concreto se detallarán los imprevistos surgidos.

3. Tecnologías Usadas

3.1.- Software empleado

- **Python** (Carhuapoma, 2024): Es el lenguaje de programación utilizado para desarrollar el backend con Flask. Es conocido por su simplicidad y facilidad de uso, lo que lo hace ideal para desarrollar APIs y manejar la lógica del servidor.



Figura 4 - Logo Python

- **Flask** (J.D Muñoz, 2017): Es el framework utilizado en el backend. Es un microframework de Python que permite desarrollar aplicaciones web de manera rápida y sencilla. Flask gestiona las peticiones HTTP y se conecta a la base de datos para proporcionar la lógica de negocio de la aplicación.



Figura 5 - Logo Flask

- **Angular** (M.J Gonçalves, 2021): Es el framework principal utilizado para desarrollar la aplicación web progresiva (PWA). Angular proporciona la arquitectura MVC (Modelo-Vista-Controlador) y herramientas como enrutamiento, servicios, y directivas que facilitan la construcción de aplicaciones web robustas.



Figura 6 - Logo Angular

- **Angular CLI** (A. Sánchez, 2023): Es la interfaz de línea de comandos que facilita la creación, desarrollo, y despliegue de proyectos Angular.



Figura 7 - Logo Angular CLI

- **TypeScript** (AlejoDev, 2023): Es el lenguaje de programación utilizado en Angular. Es un superconjunto de JavaScript que añade tipos estáticos y otras características avanzadas.



Figura 8 - Logo TypeScript

- **HTML5** (J.D.P. Jiménez, 2019): Lenguaje de marcado utilizado para estructurar y presentar el contenido en la web. Define la estructura de las vistas y componentes en los archivos .html de Angular.



Figura 9 - Logo HTML5

- **CSS** (D. Santos, 2023): Lenguaje utilizado para describir la presentación de un documento HTML. Define el estilo y la disposición de los elementos en los archivos .css de Angular.



Figura 10 - Logo CSS

- **JSON** (Formato JSON, 2024): Utilizado para almacenar datos en ElasticSearch. JSON es el formato principal para representar y transmitir la estructura de los datos que se almacenan en la base de datos, facilitando el manejo de datos estructurados y semiestructurados.



Figura 11 - Logo JSON

- **ElasticSearch** (E. de Imagina, 2024): Sistema de base de datos empleado, que permite búsquedas y análisis en tiempo real de los datos de los sensores y usuarios.



Figura 12 - Logo Elasticsearch

3.2.- Materiales

- **Visual Studio Code** (F. Flores, 2022): Es el editor de código fuente empleado para desarrollar tanto el frontend en Angular como el backend en Flask. Visual Studio Code, conocido por su versatilidad y extensibilidad, ofrece soporte para múltiples lenguajes de programación y está enriquecido con características como autocompletado inteligente, depuración integrada, y una amplia gama de extensiones que facilitan el desarrollo. Es una herramienta esencial para la edición de código, la gestión de proyectos, y el despliegue de aplicaciones.



Figura 13 - Logo Visual Studio Code

- **DevTools Chrome** (IONOS, 2023): Son las herramientas de desarrollo proporcionadas por el navegador Google Chrome. DevTools se utiliza para inspeccionar y

depurar el frontend de la aplicación Angular. Ofrecen funcionalidades como la inspección del DOM, la depuración de JavaScript, la monitorización del rendimiento, la edición de estilos CSS en tiempo real, y la simulación de diferentes dispositivos. Estas herramientas son fundamentales para garantizar la compatibilidad y el rendimiento óptimo de la aplicación en distintos entornos.



Figura 14 - Logo DevTools

- **Grafana** (RedHat, 2024): Es la plataforma de visualización de datos utilizada para crear dashboards interactivos que muestran los datos de los sensores almacenados en ElasticSearch. Grafana permite integrar y visualizar métricas de diversas fuentes de datos, ofreciendo gráficos dinámicos y alertas en tiempo real. Se integra en el frontend de la PWA mediante iframes, permitiendo a los usuarios monitorear y analizar los datos de sensores y usuarios de manera efectiva.



Figura 15 - Logo Grafana

- **PlantUML** (Plantuml, 2024): es una herramienta de código abierto la cual se ha usado para la construcción de los diferentes diagramas del proyecto.



Figura 16 - Logo PlantUML

3. 3.- Comunicación

- **Whatsapp** (Wikipedia, s.f): Es la aplicación de mensajería instantánea utilizada para la comunicación rápida y eficiente con el tutor. A través de WhatsApp, se han intercambiado mensajes de texto, archivos y documentos relevantes para el proyecto, permitiendo resolver dudas y recibir feedback de manera inmediata. Este canal ha sido esencial para mantener una comunicación fluida y continua a lo largo del desarrollo del proyecto.



Figura 17 - Logo WhatsApp

- **Google Meet** (D. Terreros, 2022): Es la plataforma de videoconferencias empleada para realizar reuniones virtuales con el tutor. A través de Google Meet, se han llevado a cabo reuniones para abordar aspectos técnicos y conceptuales de manera más efectiva.



Figura 18 - Logo GoogleMeet

3.4.- Documentación

- **Google Docs** (B. de Sousa, 2023): Es la herramienta de procesamiento de texto basada en la nube utilizada para la toma de notas durante la realización del proyecto para la posterior memoria. Google Docs permite redactar, editar y compartir documentos en tiempo real. Además, su integración con Google Drive asegura un almacenamiento seguro y accesible desde cualquier dispositivo, permitiendo trabajar en la documentación de manera flexible y eficiente.



Figura 19 - Logo Google Docs

- **Microsoft Word** (C. García, s.f): Es el software de procesamiento de texto utilizado para la redacción formal y estructurada de la documentación del proyecto. Microsoft Word ofrece un amplio conjunto de herramientas para el formato, la inserción de gráficos, tablas, y la creación de contenidos profesionales. Es especialmente útil para preparar informes finales y documentos oficiales con un alto nivel de detalle y precisión en el formato.



Figura 20 - Logo Microsoft Word

4. Diagramas de la aplicación final

4.1.- Diagrama de la estructura (en carpetas) final del proyecto

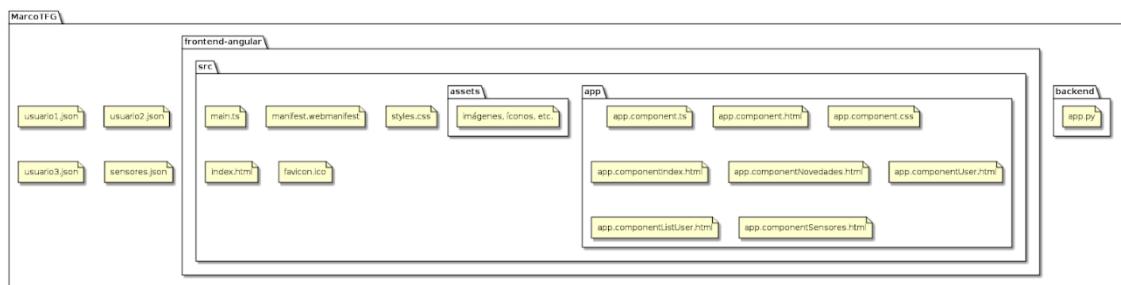


Figura 21 - Organigrama de la estructura final

4.2.- Diagrama de flujo de la PWA

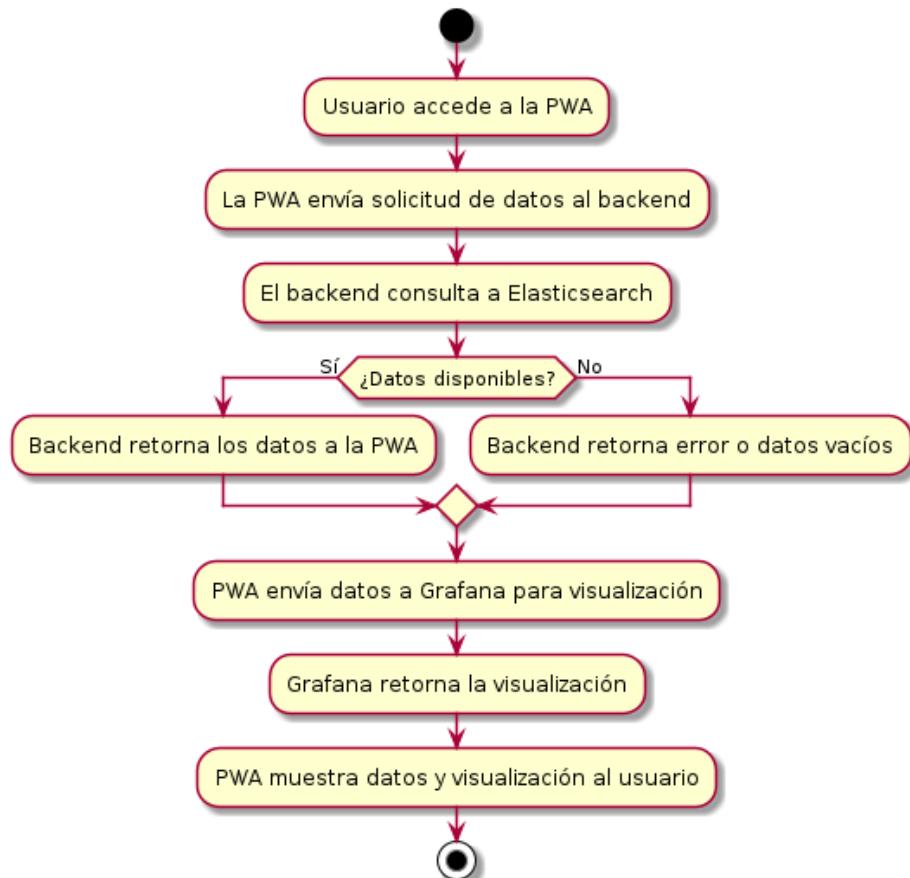


Figura 22 - Diagrama de flujo

Este diagrama ilustra el flujo de interacción entre el usuario y la PWA. El proceso

comienza con el acceso del usuario a la PWA, seguido de una solicitud de datos al backend. Esta consulta a la BD de Elasticsearch para determinar si los datos están disponibles. Si los datos existen, se devuelven a la PWA; si no, se muestra un error o datos vacíos. Paralelamente, se pasan los datos a Grafana para generar visualizaciones que se muestran al usuario.

4.3.- Caso de Uso de la PWA

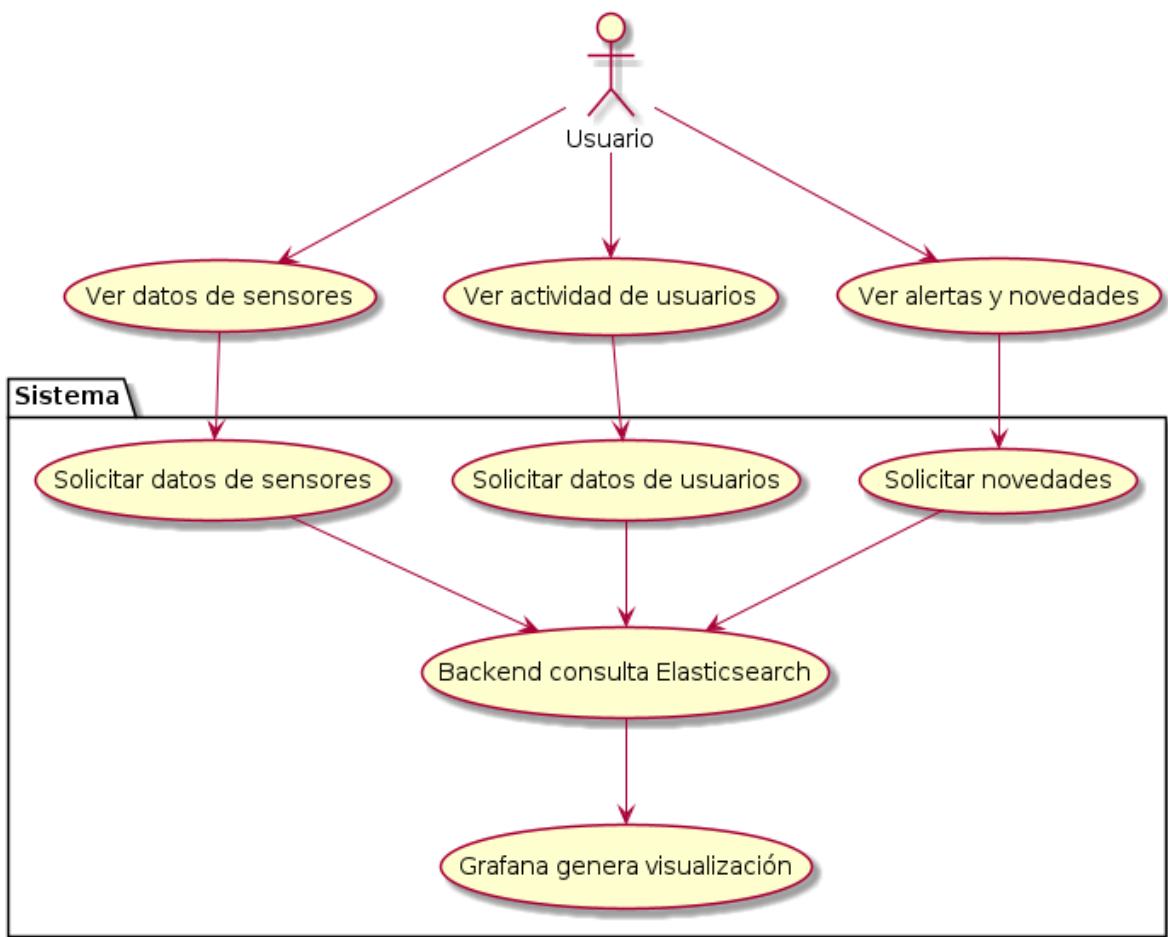


Figura 23 - Caso de Uso

los distintos casos de uso que el usuario puede realizar dentro del sistema. Como se aprecia puede ver datos de sensores, actividad de otros usuarios o alertas y novedades. En cada uno de estos casos, el sistema solicita datos específicos al backend, que a su vez consulta a Elasticsearch. Posteriormente, Grafana genera las visualizaciones de estos datos.

4.4.- Diagrama de secuencia

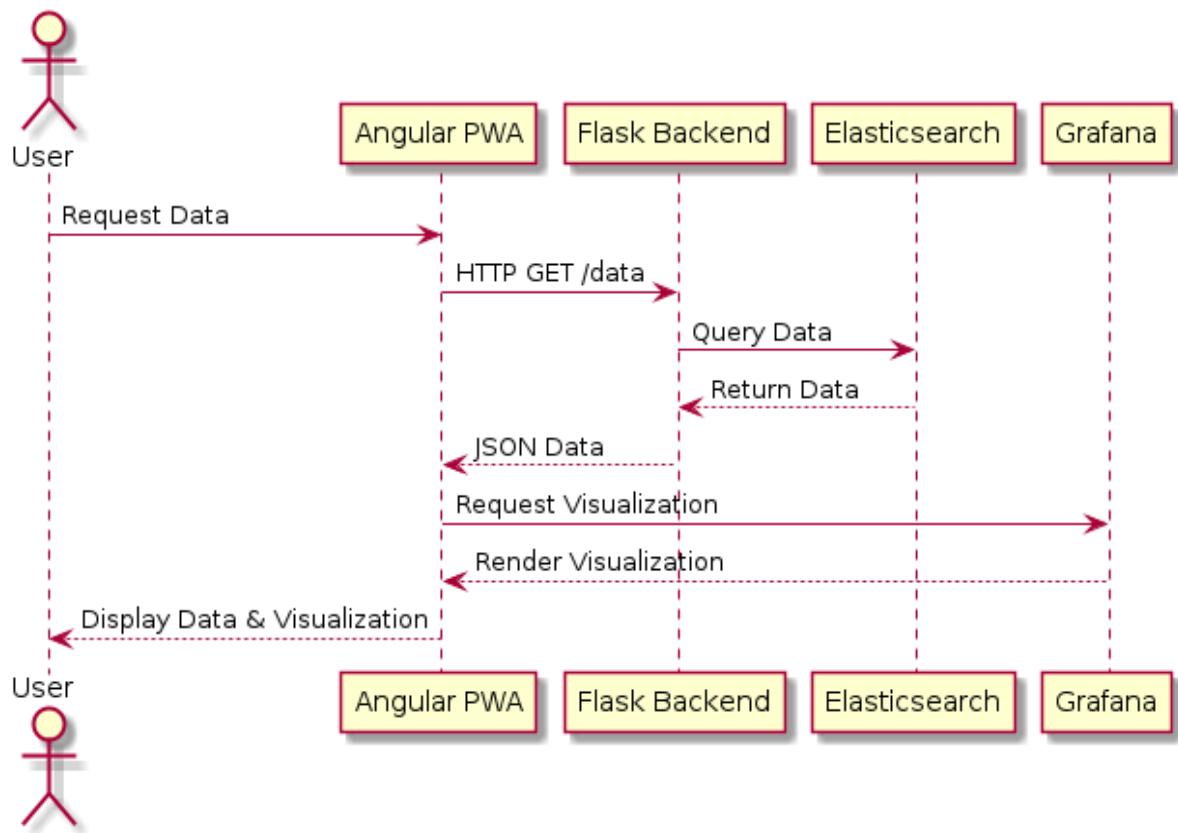


Figura 24 - Diagrama de secuencias

Aquí, en este diagrama de secuencia se presenta una secuencia temporal de las interacciones entre los componentes del sistema. El usuario solicita datos desde la PWA Angular, que envía una petición HTTP al backend en Flask, que consulta a Elasticsearch para obtener los datos solicitados y los devuelve.

Se realiza otra solicitud para generar visualizaciones en Grafana, que a su vez retorna la visualización renderizada para mostrársela al usuario.

5. Backend

El backend está desarrollado utilizando Flask, un microframework de Python que es ideal para la definición de rutas (endpoints) que manejan diferentes tipos de solicitudes, desde la obtención de datos hasta la gestión de alertas y usuarios. El archivo correspondiente se denomina app.py.

En lo que respecta al diseño de la base de datos, a continuación se muestra un diagrama Entidad-Relación.

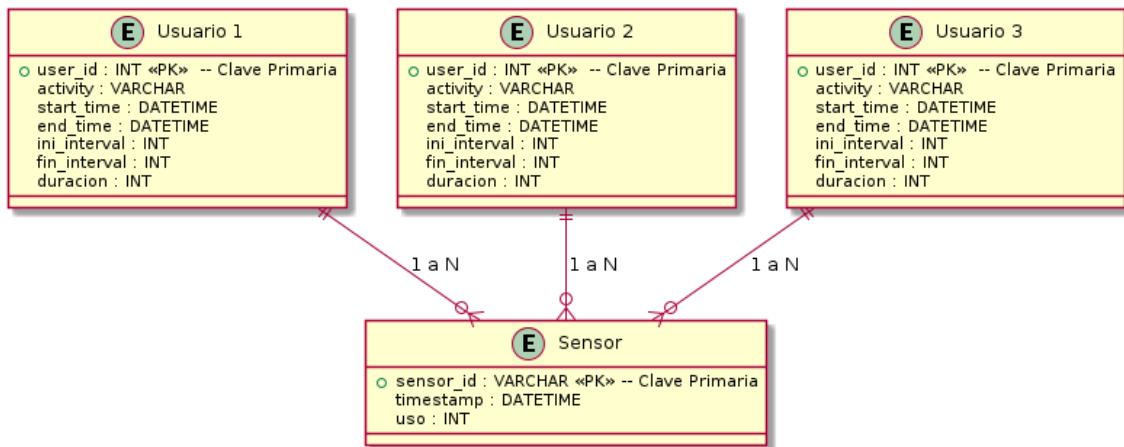


Figura 25 - Diagrama E/R de la BD

Donde cada entidad de Usuario contiene:

- activity: la actividad realizada por el usuario (ej. sleep, toileting, cooking).
- start_time: el inicio de la actividad.
- end_time: el final de la actividad.
- ini_interval: intervalo inicial de la actividad.
- fin_interval: intervalo final de la actividad.
- duracion: duración de la actividad en minutos.

Y cada Sensor:

- sensor_id: identificador del sensor (ej. grifo).
- timestamp: fecha y hora de la medición.
- uso: valor que indica si el sensor se ha utilizado (valor 0 ó 1)

Sumado a lo anterior, cada usuario (Usuario 1, Usuario 2, Usuario 3) tiene una relación de 1 a N con la entidad Sensor, lo que significa que un usuario puede estar relacionado con varios registros de sensores, pero un registro de sensor puede pertenecer a un único usuario a la vez.

5.1.- Configuración inicial

El proyecto inicia con:

- Importación de las bibliotecas necesarias y la configuración de la aplicación Flask.
- Se crea una instancia de la aplicación Flask.
- Se habilita CORS para toda la aplicación. Esto significa que todas las rutas o endpoints que definas en Flask podrán recibir solicitudes desde cualquier origen.
- Habilito el modo de depuración de Flask, lo cual ha sido muy útil durante el desarrollo porque permite reiniciar automáticamente la aplicación cuando detecta cambios en el código y proporciona mensajes de error detallados.
- Se configura el servidor Flask para que acepte conexiones desde cualquier dirección IP

```
from flask import Flask, jsonify, request
from flask_cors import CORS
from Elasticsearch import Elasticsearch

app = Flask(__name__)
CORS(app) # Habilita CORS para todas las rutas

if __name__ == '__main__':
    # Permitir conexiones desde cualquier IP de la red local
    app.run(debug=True, host='0.0.0.0')
```

Figura 26 - Esquema básico Flask (Backend)

5.2.- Conexión con la BD

La conexión a ElasticSearch se configura directamente en el archivo app.py (correspondiente con el backend del proyecto) mediante la creación de una instancia de ElasticSearch, especificando el host y el puerto. Esta conexión se utiliza posteriormente en las diversas funciones para interactuar con la base de datos.

```
es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
```

Figura 27 - Línea de Código conexión con ElasticSearch

5.3.- Endpoints

El backend proporciona múltiples endpoints que permiten interactuar con los datos almacenados en ElasticSearch. Estos endpoints están diseñados para realizar diversas operaciones como obtener datos de sensores, manejar usuarios, y gestionar alertas.

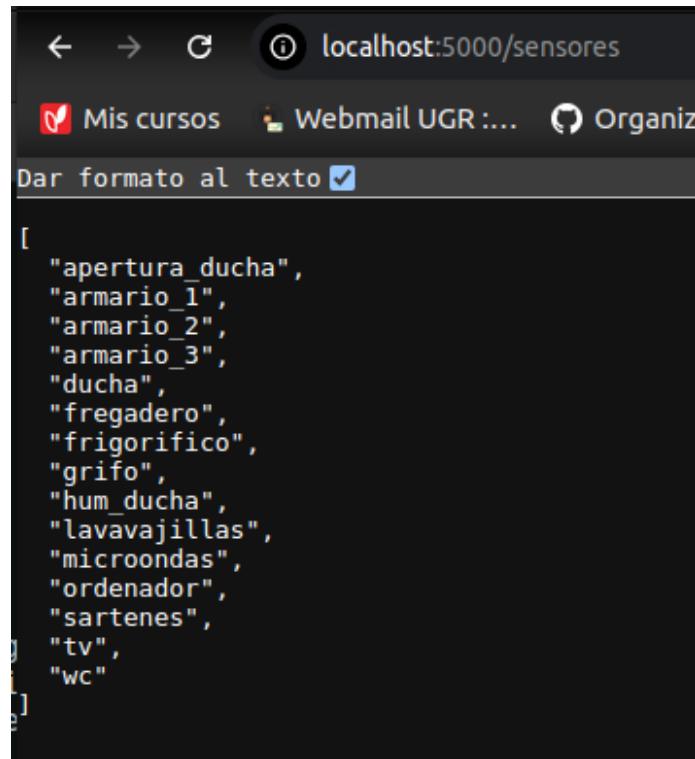
Todos los endpoints tienen la misma estructura:

```
@app.route('/usuarios', methods=['GET'])
def get_users():

    LÓGICA DE LA CONSULTA

    return jsonify(users)
```

Figura 28 - Ejemplo de estructura de un endpoint



A screenshot of a web browser window titled "localhost:5000/sensores". The page content is a JSON array of sensor names:

```
[ "apertura_ducha", "armario_1", "armario_2", "armario_3", "ducha", "fregadero", "frigorifico", "grifo", "hum_ducha", "lavavajillas", "microondas", "ordenador", "sartenes", "tv", "wc" ]
```

Figura 29 - Captura de ejemplo del contenido endpoint "sensores"

A continuación, se describen de manera general las funcionalidades ofrecidas gracias a los endpoints:

- Gestión de Usuarios: El sistema incluye endpoints para la gestión de usuarios, que permiten obtener información sobre los usuarios registrados en la base de datos. Se pueden consultar datos básicos de los usuarios, así como actividades específicas asociadas a cada uno de ellos.
- Consulta de Sensores: Los endpoints relacionados con los sensores permiten obtener información sobre los diferentes tipos de sensores que están almacenados en la base de datos. Esto es crucial para la visualización de datos en la PWA, ya que permite a los usuarios seleccionar y visualizar información específica de cada tipo de sensor.
- Gestión de Alertas: Hay endpoints dedicados a la recepción y almacenamiento de alertas generadas en el sistema. Estas alertas pueden ser consultadas posteriormente, tanto de manera individual (última alerta recibida) como de manera acumulativa (todas las alertas almacenadas).

- Simulación de Alertas: Existe un endpoint específico que permite simular la generación de alertas. Esta funcionalidad es útil durante las pruebas del sistema, ya que permite verificar cómo se manejarían las alertas en un entorno real sin necesidad de datos de producción.

5.4.- Requirements.txt

Este es el archivo que contiene los requerimientos y/o dependencias que necesita el sistema poseer para el correcto funcionamiento del desarrollo web.

El contenido de dicho documento .txt es el siguiente:

```

certbot==1.21.0
certbot-apache==1.21.0
certbot-nginx==1.21.0
certifi==2020.6.20
elastic-transport==8.13.1
elasticsearch==8.14.0
Flask==2.0.3
Flask-Cors==4.0.1
Flask-SocketIO==5.3.6
python-apt==2.4.0+ubuntu3
python-augeas==0.5.0
python-dateutil==2.8.1
python-engineio==4.9.1
python-socketio==5.11.3
Werkzeug==2.0.3

```

Figura 30 - Contenido Requirements.txt

5.5.- Puerto y lanzamiento

Para lanzar el backend se pueden usar dos comandos principalmente:

flask run

python app.py

Además, se ha de comentar que el puerto usado por el servidor es el puerto :5000 (como se ha podido ver en la [Figura 29](#)).

6. Frontend

El frontend del proyecto está organizado en una serie de componentes, cada uno encargado de manejar una parte específica de la interfaz de usuario. Estos componentes están interconectados para formar una aplicación coherente y funcional.

Esta parte del proyecto, a diferencia del backend, consta un conjunto de archivos de diferentes tipos. Contiene desde los típicos archivos html, hasta typescript, pasando por un importante manifest, entre otros (que se explicarán en los siguientes apartados).

A continuación, se muestra un esquema de la organización de carpetas y archivos del frontend de la PWA.

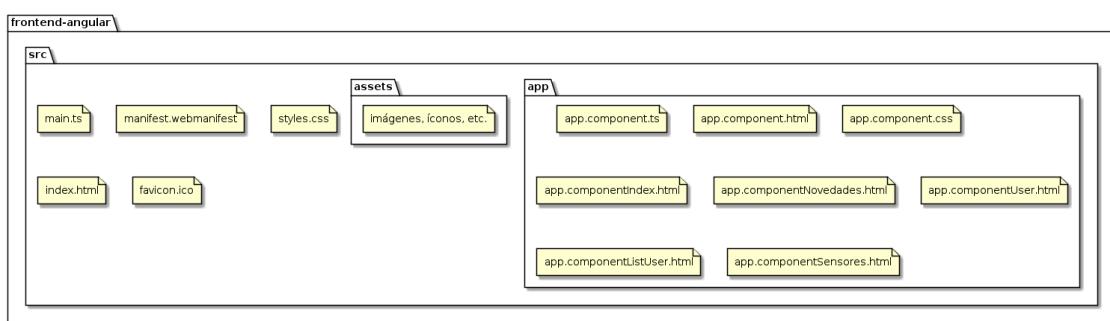


Figura 31 - Organización de carpetas de frontend

6.1.- Manifest.webmanifest

Esta es una parte esencial del proyecto para la implementación de características de PWA, las cuales permiten a la aplicación comportarse como una aplicación nativa en dispositivos móviles, e incluso con capacidades offline.

El archivo manifest.webmanifest es un archivo en formato JSON que contiene una serie de propiedades clave que describen la aplicación, por lo tanto, es clave en esta configuración, proporcionando los metadatos necesarios para que los navegadores móviles puedan reconocer e instalar la aplicación en la pantalla de inicio del dispositivo. Este archivo incluye información sobre el nombre de la aplicación, los iconos de la aplicación en diferentes resoluciones, la orientación de la pantalla, y otros detalles que mejoran la experiencia del usuario.

```
{
  "name": "Marco TFG",
  "short_name": "Marco TFG",
  "start_url": "/",
  "id": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#1976d2",
  ...
  ...
  "icons": [
    {
      "src": "assets/icon-gran.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

Figura 32 - Ejemplo de parte del contenido del manifest

6.2.- Index.html y styles.css

El archivo index.html es el punto de entrada principal de la PWA. Este archivo contiene la estructura básica del HTML que carga la aplicación Angular. Incluye la referencia al archivo manifest.webmanifest, que es esencial para configurar la PWA y hacer que el sitio web sea instalable en dispositivos móviles. Además, define el título del proyecto, el favicon y otros metadatos importantes para la correcta visualización y comportamiento de la aplicación en diversos dispositivos

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Marco TFG</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link rel="manifest" href="manifest.webmanifest">
  <meta name="theme-color" content="#4b6e52">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Figura 33 - Index.html

Como se aprecia en la imagen se cuelga del body la aplicación progresiva y se incluye el manifest.

Por otro lado, el archivo styles.css se ha utilizado para estilos muy generales, como puede ser el tipo de letra y el color de fondo de la web/PWA. Como, por ejemplo:

```
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&display=swap');

body {
  font-family: 'Poppins', sans-serif;
  font-size: 18px;
  margin: 0;
  padding: 0;
  background-color: #e3f2e1;
  color: #3e4a3f;
}
```

Figura 34 - styles.css

6.3.- Main.ts

Se trata de un archivo de tipo typescript cuya función es arrancar la aplicación. Contiene el código que inicializa el entorno de ejecución de Angular y carga el módulo raíz de la aplicación, que a su vez controla la renderización de la interfaz de usuario en el navegador.

Este archivo se encuentra claramente diferenciado en partes:

- Importaciones: código donde se recogen los imports necesarios para el correcto funcionamiento de la aplicación progresiva. Bien de archivos creados para el propio proyecto o de dependencias de bibliotecas propias de Angular. Por ejemplo:

```
import { AppComponent, UserComponent, HomeComponent, ListUserComponent, SensoresComponent, NovedadesComponent } from './app/app.component';
import { HttpClientModule } from '@angular/common/http';
```

Figura 35 - Ejemplo de imports en main.ts

- Definición de rutas: Creamos un array routes que define las rutas de la app. Cada objeto dentro del array representa una ruta con una path y un component asociado.

Cuando un usuario navega a una path específica, el componente correspondiente se renderiza. Como, por ejemplo:

```
const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'usuarios', component: ListUserComponent },
  { path: 'usuarios/:id', component: UserComponent },
];
```

Figura 36 - Ejemplo de rutas en main.ts

- Definición de NgModule: configura el módulo raíz de la aplicación. Este decorador toma un objeto de configuración que incluye: declarations, imports y bootstrap: Este array especifica el componente raíz que Angular debe cargar al iniciar la aplicación.

```
@NgModule({
  declarations: [AppComponent, UserComponent, HomeComponent,
    ListUserComponent, SensoresComponent, NovedadesComponent],
  imports: [BrowserModule, HttpClientModule, CommonModule,
    RouterModule.forRoot(routes)],
  bootstrap: [AppComponent]
})
```

Figura 37 - Definición de NgModule

- Registro del Service Worker: Se registra con navigator.serviceWorker.register('..../dist/frontend-angular/ngsw-worker.js'), apuntando al archivo que gestiona la funcionalidad offline.
Sirve para que el navegador ejecuta en segundo plano, separado de la página web. Se utiliza principalmente para habilitar características como notificaciones push y sincronización en segundo plano, y es clave para permitir que las aplicaciones funcionen sin conexión, es decir offline.

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('../dist/frontend-angular/ngsw-worker.js').then(registration => {
      console.log('ServiceWorker registration successful with scope: ', registration.scope);
    }, err => {
      console.log('ServiceWorker registration failed: ', err);
    });
  });
}
```

Figura 38 - Registro del ServiceWorker

- Bootstrap del Módulo Principal: Este método inicia la aplicación cargando el módulo raíz, en este caso concreto AppModule, lo que desencadena el proceso de compilación y renderización de la aplicación en el navegador.

```
i
platformBrowserDynamic().bootstrapModule(AppModule);
```

Figura 39 - Bootstrap del main.ts

6.4.- Componentes propios de la PWA

6.4.1.- app.component.ts

Este archivo es el principal en la aplicación progresiva, ya que define la lógica y la funcionalidad de todos los componentes. Dichos componentes son los responsables de las distintas vistas y funcionalidades dentro de la aplicación.

A continuación, muestro un diagrama de clases que trata con el contenido del archivo app.component.ts, es decir con las clases que se declaran e implementan en este archivo, así como las relaciones que existen entre ellas.

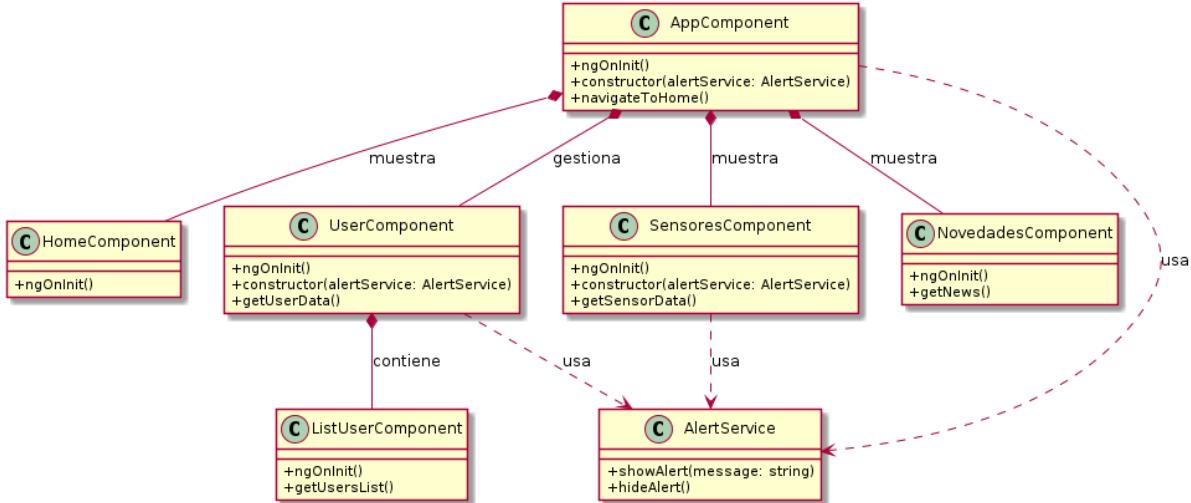


Figura 40 - Diagrama de clases de los componentes del frontend

El archivo contiene las importaciones de las bibliotecas usadas para el desarrollo.

- Component y OnInit: Son clases importadas desde @angular/core. Component se utiliza para definir un componente Angular, mientras que OnInit es una interfaz que asegura la implementación del método ngOnInit, el cual se ejecuta al inicializar el componente.
- HttpClient: Importado desde @angular/common/http, este servicio se utiliza para realizar peticiones HTTP, como la obtención de datos desde un backend.
- DomSanitizer y SafeResourceUrl: Importados desde @angular/platform-browser, estos se usan para sanitizar URLs antes de usarlas en contextos seguros (en el caso que nos concierne, al usarlo en un iframe), evitando problemas de seguridad.
- ActivatedRoute: Importado desde @angular/router, este servicio permite acceder a la información de la ruta activa, como parámetros en la URL, lo cual es útil para obtener identificadores u otros datos.
- Observable, interval y switchMap: Importados desde rxjs, estos son fundamentales para manejar operaciones asíncronas y realizar tareas repetitivas en intervalos regulares, como el polling de datos. En este caso se ha usado para las alertas de Grafana.

- Injectable: se debe de importar desde @angular/core, y se trata de un decorador que marca una clase como un servicio inyectable, permitiendo así que otros componentes y/o servicios del frontend lo usen a través de la inyección de dependencias.
- Router y NavigationEnd: Importados desde @angular/router, estos permiten navegar entre diferentes rutas y reaccionar a los eventos de navegación.

Además el archivo se ha usado para contener todas las clases referidas a los componentes así como sus implementaciones. También contiene clases auxiliares como AlertService que es la clase que se encarga del tratado correcto de las alertas de Grafana.

Todos los componentes usados en la PWA tienen la misma estructura:

- Una parte donde se declara el componente en sí y se vincula con su selector, html y css correspondientes. Como, por ejemplo:

```
@Component({
  selector: 'app-listUser',
  templateUrl: './app.componentListUser.html',
  styleUrls: ['./app.component.css']
})
```

Figura 41 - Ejemplo de declaración de un componente

- Luego se implementa la propia clase, con su constructor y todos los métodos necesarios para la funcionalidad a llevar a cabo. Si tiene que acceder a algún endpoints, almacenar valores para comunicarse con el html y demás tareas, toda esa lógica se recoge en cada componente.

Dependiendo del tipo de clase que se requiera la cabecera podrá ser de varios tipos:

```
export class AlertService {}

export class NovedadesComponent implements OnInit {}
```

Figura 42 - Ejemplos de cabecera de clase

dependiendo de si se requiere controlar la estructura visual de la aplicación. Es decir, si se requiere de modificaciones en tiempo real de la interfaz de usuario, es necesario implementar OnInit en la clase y definir el método ngOnInit, que es un ciclo de vida de Angular que se ejecuta una vez que Angular ha inicializado todas las propiedades ligadas a datos del componente.

Se permite tanto la declaración de variables como de funciones, así como su implementación. Por ejemplo:

```
sensores: string[] = [];
iframeUrl: SafeResourceUrl | undefined;
```

Figura 43 - Ejemplo declaración de variables de una clase

```
elegirSensor(event: any) {
  const sensorSeleccionado = event.target.value;
  this.cambiarSensor(sensorSeleccionado);
}
```

Figura 44 - Ejemplo de función dentro de una clase

Este es un ejemplo de constructor. Debe estar presente en cada clase y por supuesto en cada componente.

```
constructor(private alertService: AlertService, private router: Router) {}
```

Figura 45 - Ejemplo de constructor de clase

La implementación del ngOnInit, en general ha de ser así:

```
ngOnInit(): void {
  LÓGICA DE CADA CLASE
```

Figura 46 - Ejemplo de implementación de ngOnInit

Asimismo, si el componente debe tener acceso y manejar la información de un endpoints, es dentro de la clase donde se ha de manejar estas situaciones.

Por ejemplo:

```
ngOnInit(): void {
  this.fetchUsers();
}

fetchUsers(): void {
  this.http.get<string[]>('http://192.168.1.129:5000/usuarios').subscribe(
    (data) => {
      this.mensajes = data;
    },
    (error) => {
      console.error('Error fetching users', error);
      this.message = 'Error al obtener los usuarios';
    }
)
}
```

Figura 47 - Ejemplo de ngOnInit con función externa

Así pues, cada componente está cuidadosamente diseñado para manejar una parte específica de la aplicación, asegurando que la experiencia del usuario sea fluida y que la interacción con los datos sea dinámica y en tiempo real.

Seguidamente, se explica con detalle todos y cada uno de los componentes que forman la aplicación.

6.4.1.1.- NovedadesComponent

Este componente es el responsable de gestionar y mostrar las novedades (alertas de grafana) en la aplicación. Este componente interactúa con el AlertService para obtener las últimas alertas registradas y activar un aviso visual cuando se detecta una nueva alerta.

Implementa un mecanismo de polling mediante el método pollAlerts() del AlertService, que consulta al servidor cada 5 segundos para verificar si hay nuevas alertas. Si se detecta una nueva alerta, se activa una notificación visual para el usuario.

Gestiona la navegación del Router para activar/desactivar la notificación cuando el usuario navega fuera de la página de novedades. Esto asegura que el usuario solo vea el aviso cuando realmente haya novedades nuevas y relevantes.

6.4.1.2.- HomeComponent

El HomeComponent gestiona la página principal o de inicio de la aplicación. Sirve como un menú desde donde el usuario puede acceder a diferentes secciones de la aplicación, como la lista de usuarios y la gestión de sensores.

Se definen diferentes variables que representan las opciones del menú principal (Usuarios y Sensores), facilitando la navegación entre las distintas partes de la aplicación.

También mediante un mecanismo de polling hace que se pueda visualizar la alerta de novedades o no.

Además, escucha los eventos de navegación para ocultar el indicador de novedades cuando el usuario visita la página de novedades, evitando duplicidad o confusión en la interfaz de usuario.

6.4.1.3.- SensoresComponent

Es el encargado de la gestión y visualización de datos relacionados con diferentes tipos de sensores. Este componente permite al usuario seleccionar un sensor específico y visualizar los datos correspondientes en tiempo real.

Al inicializarse, el componente realiza una petición al backend para obtener una lista de todos los tipos de sensores disponibles. Esta lista se utiliza para poblar un menú de selección o para realizar otras acciones basadas en el tipo de sensor.

SensoresComponent permite cambiar la visualización en función del sensor seleccionado. Cada tipo de sensor tiene URLs específicas para cargar gráficos en iframe, y estas URLs son sanitizadas antes de ser usadas para garantizar la seguridad.

Gracias al método cambiarSensor() se puede actualizar dinámicamente el contenido mostrado en función del sensor seleccionado por el usuario. El componente está diseñado para

ser interactivo y ofrecer datos en tiempo real, mejorando la experiencia del usuario en el monitoreo de sensores.

6.4.1.4.- ListUserComponent

El ListUserComponent gestiona la visualización de una lista de usuarios en la aplicación. Interactúa con el backend para obtener y mostrar información sobre los usuarios registrados mediante una solicitud HTTP. Esta lista se almacena y se muestra en la interfaz, permitiendo al usuario visualizar y posiblemente interactuar con los datos de los usuarios.

En caso de que la solicitud para obtener usuarios falle, el componente maneja el error mostrando un mensaje de error en lugar de la lista de usuarios. Esto asegura que la interfaz de usuario sea robusta y pueda manejar fallos de red o del servidor sin causar interrupciones significativas.

6.4.1.5.- UserComponent

Se encarga de gestionar y mostrar la información específica de un usuario, incluidas sus actividades registradas. permitiendo la visualización detallada y dinámica de datos relacionados con cada usuario.

Utiliza el servicio ActivatedRoute para capturar el id del usuario desde la URL, para realizar peticiones al backend y obtener los datos específicos de ese usuario. Dependiendo del id del usuario, el componente carga un dashboard específico en un iframe que muestra datos generales del usuario. Además, permite seleccionar y visualizar diferentes actividades del usuario, como sleep, cooking, entre otras, cargando los gráficos correspondientes.

Similar al manejo de sensores en el SensoresComponent, este componente permite al usuario seleccionar una actividad específica y visualizar los datos correspondientes en un iframe. Las URLs para estos gráficos son generadas dinámicamente y sanitizadas para asegurar la seguridad.

6.4.1.6.- AppComponent

El AppComponent es el componente raíz de la aplicación, que actúa como el contenedor principal para todos los demás componentes. Pese a no tener lógica interna es fundamental ya que es donde Angular empieza a renderizar la aplicación.

6.5.- Archivos HTML

Los archivos html son los que se encargan de la construcción de la interfaz de usuario. Definen la estructura visual de los componentes y determinan cómo se presenta la información al usuario.

Cada archivo HTML asociado a un componente específico contiene elementos que interactúan con el código TypeScript, permitiendo así:

- Definir la Estructura Visual: Cada archivo HTML define la disposición de los elementos en la página, como títulos o botones. Proporciona una organización clara de la información y la usabilidad de la aplicación.
- Interacción con Datos Dinámicos: sirviéndonos de la interpolación ({{ }}), de directivas (*ngIf, *ngFor), y de enlaces de propiedades ([property]), los archivos HTML permiten que la información proveniente de los componentes TypeScript se muestre dinámicamente. Como es el caso de las listas de usuarios, detalles de sensores, y datos específicos de usuarios o actividades.
- Manejo de Eventos: también manejan la interacción del usuario con la aplicación, como clics en botones o cambios en menús desplegables. Estos eventos están vinculados a métodos en los componentes TypeScript, lo que permite actualizar la vista o realizar otras acciones en respuesta a la interacción del usuario.

En lo que sigue se irán detallando cada uno de los archivos html.

6.5.1.- app.component.html

En este archivo HTML se define la estructura principal de la aplicación. Contiene los elementos que son comunes en todas las vistas, como la cabecera, el pie de página, y la disposición general del contenido. Además, aprovechando el mecanismo interno de herencia, hace que lo definido en este html de forma general también se aplique en los demás componentes de la aplicación. Así pues, sirve como estructura base para todas las demás vistas de la aplicación, proporcionando una interfaz coherente y una navegación fácil entre diferentes páginas de la aplicación progresiva.

Las secciones que destacan en este archivo son:

- Cabecera: La cabecera incluye el logotipo de la aplicación, el título, y los enlaces de navegación principales hacia las secciones de "Usuarios" y "Sensores".

```
<div>
  <a class="btn" [routerLink]=["/usuarios"]>Usua-
  rios</a>
  <a class="btn" [routerLink]=["/sensores"]>Sensores</a>
</div>
```

Figura 48 - Contenedor de botones de la cabecera

- Contenedor principal (<main>): Aquí se ubica el router-outlet, un marcador de posición que Angular utiliza para renderizar las diferentes vistas de la aplicación según la ruta seleccionada.

```
<main>
  <router-outlet></router-outlet>
</main>
```

Figura 49 - Contenedor main

- Pie de página: El pie de página incluye enlaces a contacto mediante WhatsApp y correo electrónico, así como información sobre los derechos de autor.

6.5.2.- app.componentIndex.html

Archivo HTML correspondiente al componente HomeComponent y determina la estructura de la página de inicio de la aplicación.

Gracias al mecanismo de herencia la estructura de esta página también contiene la cabecera y el pie de página, sin necesidad de expresarlo en el archivo.

Como estructura propia de esta página tenemos:

- Título y menú principal: Presenta un título "Menú principal" y botones de navegación que dirigen a los usuarios a las secciones de "Usuarios" y "Sensores". También incluye un ícono de alerta que, si hay novedades, dirige al usuario a la sección de novedades, que incluye un ícono para hacerlo más visual e intuitivo.
- Interacción con alertas: Si se detectan nuevas alertas, mediante la variable showNovedades, se muestra un ícono que alerta al usuario sobre estas novedades y permite su acceso directo a la página de novedades. En caso contrario no se muestra, así el usuario sólo podrá elegir entre la opción “Usuarios” o “Sensores”.

```
<a *ngIf="showNovedades" [routerLink]=["/novedades"]></a>
```

Figura 50 - Ejemplo de botón condicional: Novedades

Como se aprecia en la explicación, este archivo se centra en ofrecer un punto de entrada intuitivo para que los usuarios naveguen fácilmente por las principales funcionalidades de la aplicación.

6.5.3.- app.componentNovedades.html

Se determina la estructura de la vista de novedades, que está diseñada para mostrar al usuario las alertas recientes y sus correspondientes dashboards.

Por supuesto, también se visualizará la cabecera y el pie de página en esta vista. El contenido que muestra este HTML es:

- Visualización de alertas: Si hay una alerta activa, se muestra un iframe con el dashboard correspondiente, permitiendo a los usuarios ver la información detallada asociada con la alerta.

- Mensaje condicional: En caso de que no haya alertas recientes con un dashboard asociado, se muestra un mensaje informando al usuario de esta situación.

```
<div *ngIf="!aviso">
  <h1>No hay alertas recientes con un dashboard aso-
ciado.</h1>
</div>
```

Figura 51 - Estructura con condicional : Si no hay alertas a mostrar

- Además, cabe destacar que a esta página sólo se podría acceder, en caso de que no existiese alerta alguna, mediante la ruta específica en la barra de direcciones, puesto que el acceso directo desde el menú principal no mostraría el ícono correspondiente a la alerta y su funcionalidad concreta.

Este archivo es fundamental para mantener al usuario informado sobre eventos importantes que requieren su atención, ofreciendo una interfaz clara y accesible para revisar dichas alertas.

6.5.4.- app.componentListUser.html

ListUserComponent tiene este archivo HTML vinculado. En el cual se define la estructura para mostrar una lista de usuarios, contenidos en la base de datos de ElasticSearch, en la aplicación.

Esta página al igual que en todas las demás tenemos las secciones de cabecera y pie de página. Las secciones representativas de esta página son:

- Listado de usuarios: Utiliza una lista que se genera dinámicamente a partir de los datos obtenidos del backend. Cada usuario en la lista está vinculado a una ruta específica que permite acceder a sus detalles individuales.
- Interacción mediante enlaces: Cada usuario en la lista se representa como un enlace que, al ser clicado, redirige al UserComponent para ver los detalles y actividades de ese usuario en particular.

```
<ul>
  <li *ngFor="let mensaje of mensajes; let i = index">
    <a [routerLink]=["/usuarios", i + 1]>{{ mensaje }}</a>
  </li>
</ul>
```

Figura 52 - Lógica que muestra los botones de "Usuario X" en el menú

Proporciona una interfaz sencilla pero eficaz para navegar y acceder a los datos de usuarios individuales, facilitando la gestión y visualización de la información relevante de cada uno.

6.5.5.- app.componentUser.html

Este es el archivo HTML del UserComponent cuyo propósito es determinar la estructura para la visualización de los datos y actividades de un usuario específico. Hereda también la cabecera y el pie de página.

Esta página en concreto tiene un html algo más complejo ya que el contenido a mostrar depende de las selecciones que haga el usuario. Destacamos en ella:

- Bloque div condicional: Ya que esta página viene precedida por la que se pide seleccionar entre los usuarios que hay, el contenido a mostrar estará vinculado con un condicional para asegurar que existe usuario del cual mostrar información.

```
<div class="sensores" *ngIf="usuarioId > 0">
```

Figura 53 - Estructura que asegura que haya contenido a mostrar

- Dashboard general: Muestra un iframe con el dashboard general de actividades del usuario, permitiendo una visión global de su comportamiento o datos recogidos.
- Selección de actividad: Ofrece un menú para seleccionar diferentes actividades (como sleep, cooking, etc.). Además, dependiendo del tipo de pantalla donde se abra la aplicación este cambiará la forma en que se muestra.

- Desplegable: en pantallas pequeñas, como pueden ser móviles.

```
<select class="sensor-select" (change)="cambiarActividad($event)">
    <option *ngFor="let actividad of actividades" [value]="actividad">{{ actividad }}</option>
</select>
```

Figura 54 - Menú desplegable de actividades

- Lista de botones: para la selección rápida en dispositivos con pantallas más grandes.

```
<ul class="sensor-list">
    <li class="sen_menu" *ngFor="let actividad of actividades" (click)="setIframeAct(actividad)"
        style="cursor: pointer;">
        {{ actividad }}
    </li>
</ul>
```

Figura 55 - Menú vertical (listado) de actividades

- Visualización específica de actividad: Una vez seleccionada una actividad, se carga un segundo iframe que muestra los datos específicos de esa actividad, permitiendo una visualización detallada.

```
<div *ngIf="iframeAct">
    <h1>Vista de Actividad: {{ actividadSeleccionada }}</h1>
    <iframe [src]="iframeAct"></iframe>
</div>
```

Figura 56 - Iframe de actividad

Gracias a este archivo se permite una interacción rica con los datos del usuario, ofreciendo tanto una visión general como detalles específicos a través de una interfaz flexible y adaptativa.

6.5.6.- app.componentSensores.html

Definimos la interfaz para la visualización de datos de sensores, a través de este archivo HTML ligado al componente SensoresComponent. Como ya se explicó anteriormente en el html que mostraba información sobre los usuarios, en este caso se sigue la misma dinámica, que dependiendo del tipo de pantalla que se ejecute el contenido será mostrado de una forma concreta u otra. Como no podía ser de otra manera esta página también cuenta con un pie de página y una cabecera. Las partes que destacan en la estructura de esta página son:

- Visualización general de sensores: Muestra iframes que contienen gráficos o datos de uso general de sensores, proporcionando una visión general del comportamiento del sistema.
- Selección de sensor: Similar al UserComponent, este archivo incluye un menú desplegable para seleccionar entre diferentes tipos de sensores y una lista de botones para facilitar la selección en pantallas grandes.

```
<div class="sen_med">
  <div *ngIf="sensores.length > 0; else loading">
    <h4>Elige sensor</h4>
    <!-- Menú desplegable para pantallas pequeñas -->
    <select class="sensor-select" (change)="elegirSensor($event)">
      <option *ngFor="let sensor of sensores" [value]="sensor">{{ sensor }}</option>
    </select>
    <!-- Lista de botones para pantallas grandes -->
    <ul class="sensor-list">
      <li class="sen_menu" *ngFor="let sensor of sensores" (click)="cambiarSensor(sensor)" style="cursor: pointer;">
        {{ sensor }}
      </li>
    </ul>
  </div>
```

Figura 57 - Menú desplegable de diferentes tipos de sensor

- Visualización específica de sensor: Una vez seleccionado un sensor, se cargan iframes adicionales que muestran datos específicos de ese sensor, permitiendo una monitorización detallada.

- Mensaje durante carga de datos: como la cantidad de información es mayor al obtener todos los tipos de sensores, se ha optado por incluir un mensaje mientras cargan los datos de la BD. Aunque realmente ahora mismo no se puede apreciar puesto que el volumen de datos es ínfimo, en caso de que en un futuro la aplicación contase con una mayor cantidad de sensores, ya estaría preparada para tal fin.

```
<ng-template #loading>
  <p>Cargando tipos de sensores...</p>
</ng-template>
```

Figura 58 - Mensaje de carga

Como se ha demostrado con las explicaciones correspondientes, este archivo es imprescindible para la gestión de sensores dentro de la aplicación, ofreciendo a los usuarios una interfaz interactiva para visualizar y seleccionar datos de diferentes dispositivos conectados. Además ofrece complementos que pueden ser útiles en un futuro.

6.6.- Archivo CSS

La hoja de estilo de todas las componentes del proyecto se recoge en el archivo app.component.css. Este archivo CSS es el encargado de definir los estilos visuales para el componente principal y sus subcomponentes, mejorando la presentación de la aplicación.

Como cualquier otra hoja de estilo CSS, se puede modificar cualquier etiqueta.

Lo más destacado del archivo en cuestión es que contiene reglas específicas que responden a distintas resoluciones de pantalla mediante el uso de media queries. Este hecho garantiza que la aplicación sea completamente responsive (adaptativa a diferentes tamaños de dispositivos, desde móviles hasta pantallas grandes).

Así logramos mantener una coherencia visual de forma que la experiencia de usuario sea fluida y agradable.

A continuación, se detallarán algunos aspectos concretos de esta hoja de estilo:

- Estilo adaptable: cambia dependiendo del tamaño de la pantalla, lo que permite una experiencia fluida tanto en dispositivos móviles como en monitores. Para este fin se han usado media queries, como en este ejemplo:

```
@media (min-width: 768px) {
    .medio {
        margin-top: 150px;
        margin-left: 30px;
    }
}
```

Figura 59 - Ejemplo de uso media queries

- Modificaciones del tipo de botón dinámicamente: en función del tipo de pantalla que se trata se muestra un tipo de botón u otro. Por ejemplo, en monitores el diseño contenía una lista de opciones y en móvil un menú desplegable, para eso se han hecho uso de los display, como en el ejemplo siguiente:

```
.sensor-select {
    display: none;
}
```

Figura 60 - Display: none

- Cabecera y pie de página: diseñados utilizando grid, lo que permite una distribución equilibrada de los elementos (logotipo, títulos y botones de navegación en la cabecera; información de contacto en el pie de página). En el ejemplo siguiente lo vemos:

```
header {
    display: grid;
    grid-template-columns: 1fr 6fr;
}

footer {
    display: grid;
    grid-template-columns: 1fr auto 1fr;
}
```

Figure 61 - Grid de la cabecera y pie de página

- Botones interactivos: Los botones tienen un diseño uniforme con esquinas redondeadas (usando border-radius), relleno (con padding), y un efecto de transformación que aumenta el tamaño ligeramente al pasar el cursor por encima (gracias al uso de transform: scale(1.05)), lo que mejora la interacción con el usuario. En el ejemplo siguiente se muestran únicamente las partes indicadas, no el estilo completo de los botones:

```
.btn-navigate {
  padding: 18px 35px;
  border-radius: 12px;
  transition: background-color 0.3s, transform 0.3s;
  cursor: pointer;
}

.btn-navigate:hover {
  background-color: #6aa68b;
  transform: scale(1.05);
}
```

Figura 62 - Ejemplo del estilo de los botones

- Organización eficiente: gracias a los media queries se permite modificar el estilo dependiendo de la pantalla, y además para cambiar la estructura se ha hecho uso de grid. De forma que si el tamaño de la pantalla era mayor la organización cambiaba y aumentaban las columnas, sin embargo, si era un tamaño de pantalla menor, el formato de salida será siempre una única columna para permitir el correcto visionado de la aplicación.

```
.sensores {
  display: grid;
  grid-template-columns: 1fr; /* Una sola columna en móvil */
}

.sensores {
  display: grid;
  grid-template-columns: 5fr 1fr 4fr; /*4 columnas en monitor*/
}
```

Figura 63 - Grid de la estructura general

6.7.- Otros archivos

En el frontend del proyecto existen otros archivos, generados por defecto, de los cuales algunos sí han tenido cierta relevancia en el desarrollo y requieren al menos de una mención.

6.7.1.- angular.json

Aquí se definen las configuraciones del proyecto y sus entornos, como las configuraciones de desarrollo y producción, las rutas de los archivos de salida, y la forma en que se construyen los estilos, scripts y demás.

También especifica las configuraciones para el servicio, la compilación, y el service worker cuando está activado.

Además, durante el desarrollo se han tenido que modificar los valores por defecto de los budgets, que determina los límites de tamaño para los archivos, puesto que el archivo dedicado al css ocupaba más tamaño.

6.7.2.- proxy.conf.json

Básicamente, redirige la solicitud de ngsw-worker.js al archivo correspondiente dentro de la carpeta /dist cuando se trabaja con el servidor de desarrollo.

6.7.3.- ngsw-worker.js

Se trata del service worker generado para la aplicación web progresiva, implementa características como el caché, actualizaciones automáticas y soporte offline.

6.7.4.- ngsw-config.json

Configuración del service worker para la aplicación. Gestiona qué archivos deben ser cacheados, cómo deben comportarse las actualizaciones, y cómo manejar las solicitudes de la API.

6.8.- Comandos usados y puerto

El comando para compilar la aplicación para producción es:

ng build --configuration production

Crea y/o modifica la carpeta /dist que se utiliza para contener los archivos finales que serán distribuidos.

En esta carpeta se incluyen los archivos optimizados, se transforman los archivos en versiones empaquetadas de ellos mismos y además se almacenan archivos estáticos como imágenes u otros recursos. Gracias a esta carpeta y a un servidor web estático se podrá desplegar la aplicación en producción.

Durante el desarrollo y para comprobar el funcionamiento de forma dinámica, se ha usado el comando:

ng serve

Con lo cual para acceder a la aplicación web, se debería de acceder al puerto correspondiente al servidor estático, que normalmente es el :8080. En caso de querer acceder de forma dinámica, se hace mediante el puerto :4200.

7. Grafana

Como se ha comentado anteriormente, Grafana es una plataforma de análisis y monitoreo de código abierto que permite visualizar métricas, consultas y gráficos a partir de múltiples fuentes de datos. Ha sido una herramienta fundamental para alcanzar el objetivo y lograr el propósito final de la PWA.

7.1.- Instalación, primeros pasos, conexión y dashboard

Requiere de una instalación previa, que en el caso de este proyecto se ha hecho mediante comandos en la terminal de ubuntu.

```
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"  
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -  
sudo apt update  
sudo apt install grafana  
sudo systemctl start grafana-server  
sudo systemctl enable grafana-server  
sudo systemctl status grafana-server
```

Una vez lo tenemos instalado, podemos acceder a Grafana mediante el puerto :3000.

De primeras se nos pide un usuario y contraseña, que por defecto son: admin. Posteriormente, lo siguiente que se tuvo que hacer en Grafana fue crear una conexión con la base de datos mediante el puerto correspondiente, en este caso :9200 (puerto por defecto de ElasticSearch).

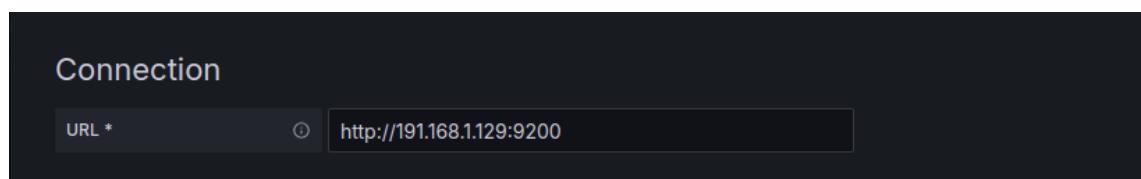


Figura 64 - Conexión BD en Grafana

También hubo que incluir detalles específicos de la base de datos del proyecto, como el índice o timestamp.

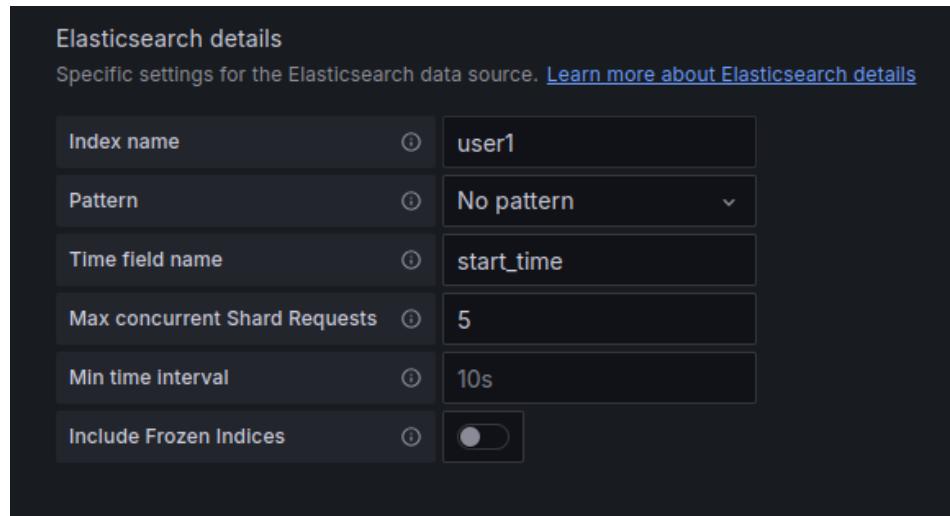


Figura 65 - Información relevante en conexión con Grafana

Esto que parece y es sencillo a simple vista es porque Grafana tiene por defecto la conexión con bases de datos de ElasticSearch, como de otras como Prometheus. Sin embargo, para MongoDB no estaba el plugin disponible.

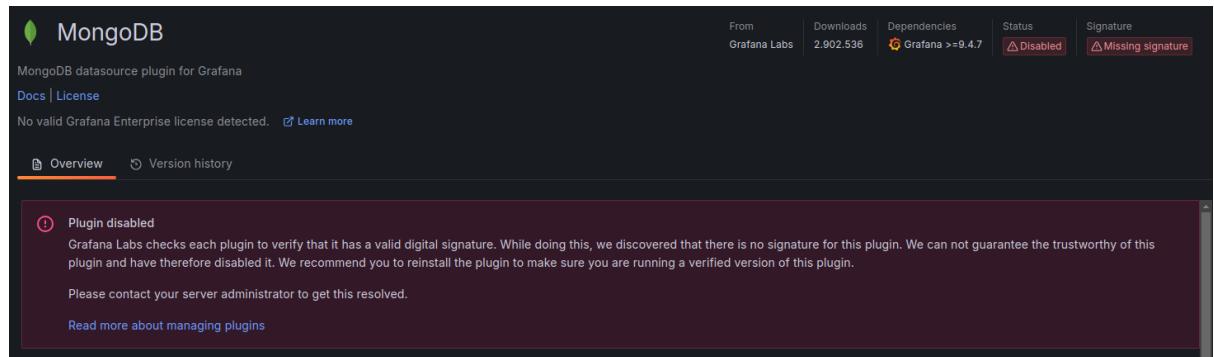


Figura 66 - Error con MongoDB

Esto hizo que se tuviese que cambiar la base de datos de MongoDB. Tras surgir este imprevisto y después de meditar sobre cual podría ser una solución que no modificase en demasiado los planes iniciales, se decidió utilizar ElasticSearch. Ahora ya sí, se contaba con conexión directa a Grafana y además ha resultado ser una óptima solución para el manejo de ficheros.

Para trabajar con Grafana se procuraron unos archivos con mediciones de los sensores de 3 usuarios diferentes y de distintas actividades, a lo largo de un día concreto.

Para su uso, los archivos cuya extensión era .tsv se tuvieron que modificar y cambiar el formato a JSON, de forma que ElasticSearch los leyese sin problema.

```

Sleep      (0, 219)    2024-07-05 02:00:00    2024-07-05 05:39:00

{
  "index": {
    "_index": "user1"
  }
}
{
  "activity": "sleep",
  "start_time": "2024-07-05T02:00:00",
  "end_time": "2024-07-05T05:39:00",
  "ini_interval": 0,
  "fin_interval": 219,
  "duracion": 219
}

```

Figura 67 - Comparativa entre archivo final JSON e inicial .tsv

Además de las diferencias lógicas del formato, se aprecia en la imagen anterior como las líneas del fichero final contienen datos que no se tenían en el de partida. Esto se debe a que se trataron los datos y se crearon campos nuevos referentes al tiempo para hacer más sencillo el trabajo a la hora a mostrar visualizaciones, De la misma manera que se separaron en distintos campos algunos de los valores o se modificó el formato de fecha para que fuese compatible con la BD en ElasticSearch y se incluyeron nombres a los campos que inicialmente no tenían.

Una vez teníamos la conexión con la base de datos hecha, estaba en disposición de hacer los dashboard. Donde mediante una consulta a la BD y seleccionando el tipo de gráfico, podemos conseguir visualizaciones de los datos que se leen de la BD.

Estas visualizaciones son las que se muestran en la aplicación web progresiva.

7.2.- Alertas

Este tema merece una mención aparte puesto que se trata de una funcionalidad importante en el desarrollo, y es que gracias a esta característica de Grafana se permite recibir notificaciones cuando ciertos valores o métricas exceden los umbrales predefinidos.

Las alertas pueden ser creadas/modificadas como cada usuario quiera. Para ello dispone de las Alerting Rule, que son las encargadas de saber:

- La métrica se debe supervisar, mediante una consulta determinada.

- La frecuencia a la que debe evaluarse la métrica.
- Condiciones que deben cumplirse para que se dispare la alerta. Por ejemplo, si un valor está por encima o por debajo de un umbral definido.

En este proyecto, las alertas se han definido para que cada minuto se refresque la información y se evalúe la métrica, y con la condición de que si se modifica o añade algún valor a alguna de las actividades de un usuario, salte la alarma y lo notifique.

Las notificaciones se producen cuando se cumple la condición y puede hacerse por medio de diferentes canales como correo electrónico, slack, PagerDuty... En el caso que estamos tratando no cabe duda de que la opción a elegir sería Webhook. Así cuando la alerta programada detecte algún cambio se notificará al backend gracias a esta función. Para este fin, se creó un contact point donde se especificaba la URL, que previamente había sido creado un endpoint en el backend en dicha dirección.

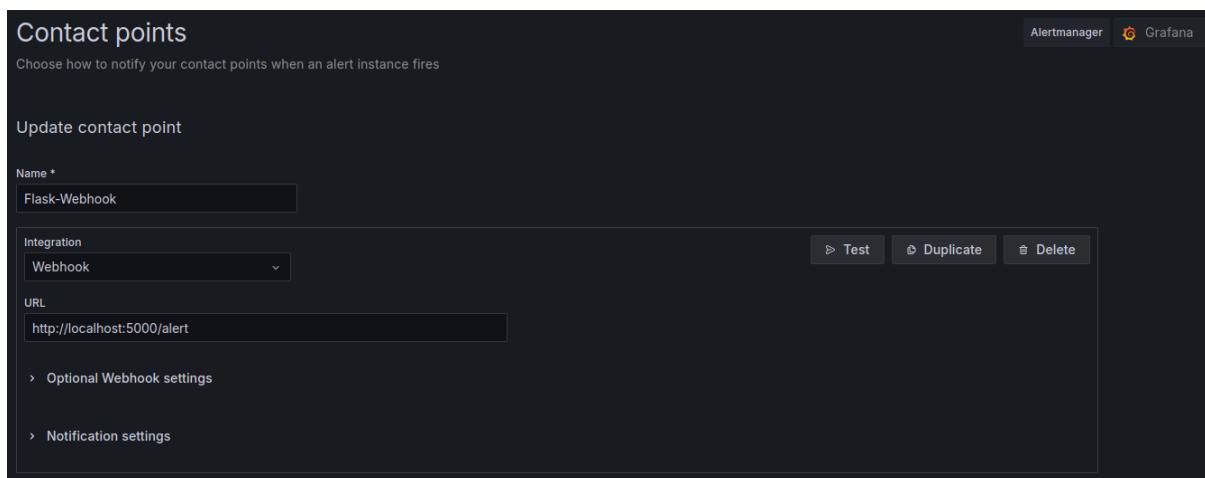


Figura 68 - Conexión Webhook para alertas

8. ElasticSearch

Como se ha comentado con anterioridad, la primera opción para gestionar la base de datos no era ElasticSearch, pero el propósito final no se vio afectado. Ya que se ha utilizado ElasticSearch como base de datos para almacenar y gestionar los datos de los sensores y usuarios. Estos datos son luego accesibles y presentados en la PWA mediante visualizaciones en Grafana, lo que permite monitorizar y analizar la información en tiempo real.

Los datos recopilados por los sensores se indexan en ElasticSearch y son consultados a través del backend en Flask, proporcionando una solución eficiente y rápida para el manejo de grandes volúmenes de información.

Como se ha podido ver en [Figura 27](#) la conexión con el backend en Flask es muy simple, al igual que sus consultas, de las que abajo se proporciona un ejemplo.

```
@app.route('/sensores', methods=['GET'])
def obtener_tipos_sensores():
    try:
        query = {
            "size": 0,
            "aggs": {
                "tipos_sensores": {
                    "terms": {
                        "field": "sensor_id.keyword",
                        "size": 1000
                    }
                }
            }
        }
        resultado = es.search(index="sensores", body=query)
        tipos_sensores = [bucket["key"] for bucket in resultado["aggregations"]["tipos_sensores"]["buckets"]]
    except Exception as e:
        print(f"Error en el servidor: {e}")
        return jsonify({"error": "Error interno del servidor"}), 500
    return jsonify(tipos_sensores)
```

Figura 69 - Ejemplo de consulta: Tipos de sensores

En este proyecto se han usado 4 índices: sensores, user1, user2, user3. Los cuales se pueden ver con el siguiente comando:

```
curl -X GET "localhost:9200/_cat/indices?v"
```

Otro comando importante utilizado durante el desarrollo ha sido:

```
curl -X POST "localhost:9200/_bulk?pretty" -H "Content-Type: application/json" --data-binary @/home/marco/Escritorio/MarcoTFG/NOMBRE_ARCHIVO.json
```

Cuyo fin es subir archivos y crear índices o actualizar contenidos.

Esto se ha podido llevar a cabo gracias a que los datos en ElasticSearch se almacenan en forma de documentos JSON, ya que es un formato flexible y común para datos estructurados y no estructurados.

Para terminar, se va a mostrar parte del contenido de los ficheros JSON almacenados en la base de datos.

```
{"index": {"_index": "sensores"}}
{"sensor_id": "grifo", "timestamp": "2024-07-05T02:03:00", "uso": 0}
{"index": {"_index": "sensores"}}
{"sensor_id": "grifo", "timestamp": "2024-07-05T02:04:00", "uso": 0}
{"index": {"_index": "sensores"}}
{"sensor_id": "grifo", "timestamp": "2024-07-05T02:05:00", "uso": 0}
{"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:06:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:07:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:08:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:09:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:10:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:11:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:12:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:13:00", "uso": 0}
 {"index": {"_index": "sensores"}}
 {"sensor_id": "grifo", "timestamp": "2024-07-05T02:14:00", "uso": 0}
["index": {"_index": "sensores"}]
```

Figura 70 - Extraído del archivo de sensores

```
{"index": {"_index": "user1"}}
{"activity": "sleep", "start_time": "2024-07-05T02:00:00", "end_time": "2024-07-05T05:39:00", "ini_interval": 0, "fin_interval": 219, "duracion": 219}
{"index": {"_index": "user1"}}
{"activity": "toileting", "start_time": "2024-07-05T05:40:00", "end_time": "2024-07-05T05:49:00", "ini_interval": 220, "fin_interval": 229, "duracion": 9}
{"index": {"_index": "user1"}}
{"activity": "sleep", "start_time": "2024-07-05T05:50:00", "end_time": "2024-07-05T07:12:00", "ini_interval": 230, "fin_interval": 312, "duracion": 82}
{"index": {"_index": "user1"}}
{"activity": "cooking", "start_time": "2024-07-05T07:13:00", "end_time": "2024-07-05T07:13:00", "ini_interval": 313, "fin_interval": 313, "duracion": 0}
{"index": {"_index": "user1"}}
{"activity": "sleep", "start_time": "2024-07-05T07:14:00", "end_time": "2024-07-05T07:18:00", "ini_interval": 314, "fin_interval": 318, "duracion": 4}
{"index": {"_index": "user1"}}
{"activity": "resting", "start_time": "2024-07-05T07:19:00", "end_time": "2024-07-05T07:39:00", "ini_interval": 319, "fin_interval": 339, "duracion": 20}
{"index": {"_index": "user1"}}
{"activity": "sleep", "start_time": "2024-07-05T07:40:00", "end_time": "2024-07-05T07:44:00", "ini_interval": 340, "fin_interval": 344, "duracion": 4}
 {"index": {"_index": "user1"}}
 {"activity": "cooking", "start_time": "2024-07-05T07:45:00", "end_time": "2024-07-05T07:46:00", "ini_interval": 345, "fin_interval": 346, "duracion": 1}
 {"index": {"_index": "user1"}}
 {"activity": "sleep", "start_time": "2024-07-05T07:47:00", "end_time": "2024-07-05T07:49:00", "ini_interval": 347, "fin_interval": 349, "duracion": 2}
 {"index": {"_index": "user1"}}
 {"activity": "exit", "start_time": "2024-07-05T07:50:00", "end_time": "2024-07-05T11:39:00", "ini_interval": 350, "fin_interval": 579, "duracion": 229}
 {"index": {"_index": "user1"}]
```

Figura 71 - Extraído del archivo del Usuario 1

9. Problemas surgidos

Durante el desarrollo del proyecto han ido apareciendo imprevistos y problemas, que se han tenido que ir solventando sobre la marcha de la mejor manera posible.

9.1.- Problema HTTPS

La idea inicial es que la PWA se permitiese instalar en otros dispositivos, pero esto no ha podido ser posible, ya que para que esta característica de la PWA pudiese llevarse a cabo debíamos trabajar en un entorno seguro con HTTPS y no era así.

Se intentaron varias soluciones como hacer seguro el frontend usando ngrok o localhost.run, pero se producía conflicto al mezclar HTTP del backend con HTTPS del frontend. Además, que esta solución no parecía demasiado buena para ser definitiva ya que generaba cada acceso un entorno seguro distinto.

También se probó con un dominio de IONOS y con sus certificados correspondientes, pero surgieron otros problemas de conexión y finalmente tampoco llegó a buen puerto.

Sin embargo, pese a no contar con certificados SSL, el backend podría soportar estos certificados, simplemente modificando una línea:

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
    app.run(ssl_context=('cert.pem', 'key.pem'))
```

Figura 72 - Ejemplo de inclusión de certificados en Flask

cert.pem y key.pem debería sustituirse por la ruta a dichos certificados.

9.2.- Problema con implementación JWT

Además, el problema anterior, derivó en otros como la implementación de JSON Web Tokens para autenticar y autorizar usuarios, que le hubiese dado un valor añadido extra al proyecto y en el backend no suponía nada de gran complejidad.

A continuación, se muestran las líneas de código que habría que añadir para tal fin.

```
from flask_jwt_extended import JWTManager, create_access_token,  
jwt_required, get_jwt_identity  
  
app.config['JWT_SECRET_KEY'] = 'CONTRASEÑA'  
jwt = JWTManager(app)  
  
@app.route('/login', methods=['POST'])  
def login():  
    username = request.json.get('username', None)  
    password = request.json.get('password', None)  
  
    if username != 'USER EN CUESTION' or password != 'CONTRASEÑA':  
        return jsonify({'msg': "Usuario o contraseña incorrectos"}),  
401  
  
    access_token = create_access_token(identity=username)  
    return jsonify(access_token=access_token)  
  
@app.route('/protected', methods=['GET'])  
@jwt_required()  
def protected():  
    current_user = get_jwt_identity()  
    return jsonify(logged_in_as=current_user), 200
```

Figura 73 - Ejemplo de uso de JWT en Flask

Pero actualmente sin HTTPS esta implementación quedará pendiente para futuros trabajos sobre la PWA.

10. Conclusiones

A modo de conclusión, el desarrollo de este proyecto ha permitido la creación de una aplicación web progresiva que combina tecnologías modernas y flexibles para la visualización de datos en tiempo real, procedentes de sensores y usuarios, mediante la integración de Grafana y la base de datos en Elasticsearch. Gracias a este conjunto de tecnologías, se ha logrado el desarrollo de una aplicación sencilla pero adaptable a cualquier dispositivo.

Acto seguido, se detallan los principales logros y aportaciones del proyecto:

- Implementación de una arquitectura sólida:
 - La elección de Angular como framework frontend y Flask en el backend ha facilitado el desarrollo de una PWA modular y escalable. Esta arquitectura ha permitido una correcta separación entre la lógica de presentación, la gestión de usuarios y sensores, y la visualización de datos en tiempo real.
 - La integración con Grafana para la visualización de métricas ha sido uno de los pilares fundamentales, logrando un sistema interactivo y fácilmente configurable para mostrar información de manera comprensible y efectiva al usuario.
- Uso de Elasticsearch como base de datos aporta robustez:
 - La elección de Elasticsearch como sistema de almacenamiento ha permitido un manejo eficiente de los datos de los sensores, con búsquedas rápidas y análisis en tiempo real. Su capacidad para manejar grandes volúmenes de datos ha sido clave para asegurar el buen rendimiento del sistema.
 - Como se ha expresado con anterioridad, Elasticsearch facilita la integración con Grafana, lo que ha permitido generar dashboards dinámicos y personalizables, proporcionando a los usuarios información clara y en tiempo real sobre la actividad de los sensores y usuarios.
- Desarrollo de una interfaz amigable y responsive:

- El diseño del frontend, basado en Angular, se ha centrado en crear una interfaz intuitiva y fácil de usar, con una disposición responsiva que se adapta a diferentes tamaños de pantalla. Esto ha garantizado una experiencia de usuario óptima tanto en dispositivos móviles como en escritorio.
- Contribución al sector:
 - Este proyecto contribuye al desarrollo de soluciones de monitorización de datos en tiempo real, ofreciendo una plataforma escalable y eficiente que podría aplicarse a diversos ámbitos, como la gestión de infraestructuras, el análisis industrial y/o el monitoreo ambiental.
 - La elección de tecnologías de código abierto asegura que el proyecto pueda ser fácilmente extendido, modificado y adaptado a nuevas necesidades y funcionalidades.

11. Trabajos futuros

En este último apartado de la memoria se propondrán posibles futuras mejoras y/o trabajos a realizar para dotar a este proyecto PWA de un valor real en el mercado.

11.1.- Fases futuras a desarrollar

Dado el alcance limitado del proyecto en su fase actual, algunas fases críticas, como el despliegue completo, las pruebas exhaustivas y el mantenimiento continuo, no se han implementado por completo. Sin embargo, el diseño y la implementación del proyecto han sido realizados con la intención de facilitar estas fases en el futuro, asegurando que el sistema pueda ser llevado a producción de manera segura y eficiente, y que pueda ser mantenido y mejorado con el tiempo.

11.1.1.- Fase de Despliegue

Aunque el proyecto actualmente se ejecuta solo en un entorno de desarrollo local (localhost), está preparado para ser desplegado en un entorno de producción. La fase de despliegue futura incluirá los siguientes pasos clave:

- Lanzamiento en un Servidor de Producción: La PWA y el backend desarrollado en Flask serán desplegados en un servidor de producción confiable, como un servidor Nginx o Apache, que manejará las solicitudes de los usuarios de manera eficiente.
- Configuración de HTTPS: Para garantizar la seguridad de los datos transmitidos, se configurará HTTPS utilizando certificados SSL/TLS. Esto protegerá la comunicación entre los usuarios y el servidor, asegurando que los datos sensibles, como la autenticación y la información de los sensores, estén encriptados y protegidos contra accesos no autorizados.
- Optimización de la Infraestructura: El servidor será optimizado para manejar la carga y asegurar una experiencia de usuario fluida. Esto incluye la configuración de un proxy inverso para manejar las solicitudes y la integración con ElasticSearch y Grafana para la visualización de datos en tiempo real.

11.1.2.- Fase de Pruebas

El proyecto ha sido desarrollado con una arquitectura modular que facilita la incorporación de pruebas unitarias, de integración y de aceptación en etapas posteriores. Cada componente del sistema ha sido construido con la capacidad de ser probado individualmente, y la API del backend está documentada de manera que futuras pruebas de integración puedan ser realizadas sin complicaciones.

- Pruebas Unitarias: El código está organizado de manera que los desarrolladores puedan añadir fácilmente pruebas unitarias para verificar la funcionalidad de cada componente en Angular y cada endpoint en Flask.
- Pruebas de Integración: La estructura del sistema permite que futuras pruebas de integración puedan asegurarse de que la comunicación entre el frontend, el backend y la base de datos se realice correctamente
- Pruebas de Rendimiento y Seguridad: Aunque estas no han sido implementadas, el sistema está diseñado para que herramientas de pruebas de rendimiento y seguridad puedan integrarse sin mayores dificultades, permitiendo evaluar cómo se comporta el sistema bajo carga y asegurar la protección de los datos de los usuarios.
- Pruebas de Usuario: Finalmente, se llevarán a cabo pruebas de usuario para asegurar que la PWA sea fácil de usar y cumpla con los requisitos funcionales establecidos, permitiendo realizar ajustes según las necesidades detectadas.

11.1.3.- Fase de Mantenimiento

El proyecto está preparado para ser escalable y mantenible a largo plazo. Se han seguido los principios de desarrollo que aseguran que el código sea fácil de entender, modificar y extender, lo cual es esencial para un mantenimiento eficiente.

- Escalabilidad: La arquitectura elegida para el proyecto (Angular en el frontend, Flask en el backend, y ElasticSearch como base de datos) es conocida por su capacidad para escalar. Esto significa que el sistema puede manejar un mayor número de usuarios, sensores y datos sin requerir cambios significativos en su estructura.

- Mantenibilidad: El código está organizado de manera clara y sigue buenas prácticas de programación, lo que facilita que otros desarrolladores puedan continuar el trabajo, realizar correcciones de errores o implementar nuevas funcionalidades en el futuro.

Se ha garantizado que el proyecto pueda continuar evolucionando y mejorando en el futuro, ofreciendo una base sólida sobre la cual construir, probar y mantener nuevas funcionalidades según sea necesario. El enfoque en la escalabilidad y la mantenibilidad asegura que, aunque el proyecto esté en una etapa inicial, está bien posicionado para crecer y adaptarse a futuras necesidades.

11.2.- Implementaciones adicionales

Para proseguir con el desarrollo de este proyecto, se pueden llevar a cabo diferentes actualizaciones e implementaciones como:

- Autenticación de usuarios: llevar a cabo la implementación gracias a JWT, como se ha comentado en [Figura 73](#).
- Notificaciones push: permitirían enviar mensajes en tiempo real a los usuarios de la aplicación, incluso si no están en la página. Esta implementación en la PWA se podría llevar a cabo con pywebpush y la correcta adaptación del backend.
- Posibilitar un “Modo Noche”: Implementar un modo oscuro mejora la experiencia del usuario. Se puede llevar a cabo en el frontend Angular mediante modificaciones en el fichero CSS.
- Generación de PDF: podría ser útil para exportar datos o visualizaciones. Esto se puede lograr en el backend Flask instalando WeasyPrint que convierte el contenido HTML a PDF de forma simple.
- Incorporación de Machine Learning: que podría usarse para el análisis predictivo de usos de aparatos o detectar fallos

12. Bibliografía

Documentación

AlejoDev. (2023, noviembre 20). Navegando por el Mar de TypeScript 🚢 : Conceptos, Ventajas, y Desventajas en el Desarrollo Web 🌐. Medium. <https://medium.com/@alejodev95/introducci%C3%B3n-a-typescript-9740a71aaaee>

Carhuapoma, L. (s/f). *Los Mejores Lenguajes para Backend: Potenciando el Desarrollo de Aplicaciones*. Imagineapps.Co. Recuperado el 5 de septiembre de 2024, de <https://www.imagineapps.co/blog-posts-es/los-mejores-lenguajes-para-backend-potenciando-el-desarrollo-de-aplicaciones>

Chrome DevTools e “Inspeccionar elemento” en otros navegadores. (2023, agosto 28). IONOS Digital Guide; IONOS. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/inspeccionar-elemento/>

de Imagina, E. (2024, septiembre 6). ¿Qué es Elasticsearch?: Funciones y Características. Imaginaformacion.com; Imagina Formación. <https://imaginaformacion.com/tutoriales/que-es-elasticsearch>

de Sousa, B. (2023, enero 24). Google Docs: Qué es, cómo funciona, características y más. Blog IPNET; IPNET Growth Partner. <https://ipnet.cloud/blog/es/google-workspace-es/google-documentos-todo-lo-que-necesitas-saber/>

Flores, F. (2022, julio 22). Qué es Visual Studio Code y qué ventajas ofrece. Openwebinars.net. <https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/>

Formato JSON: qué es y para qué sirve. (s/f). Arsys. Recuperado el 5 de septiembre de 2024, de <https://www.arsys.es/blog/formato-json-que-es-y-para-que-sirve>

García, C. (s/f). ¿Qué es word y para qué sirve? www.cursosfemxa.es. Recuperado el 5 de septiembre de 2024, de <https://www.cursosfemxa.es/blog/word>

Gonçalves, M. J. (2021, octubre 13). *¿Qué es Angular y para qué sirve?* Blog de hiberus; Hiberus. <https://www.hiberus.com/crecemos-contigo/que-es-angular-y-para-que-sirve/>

Jiménez, J. D. P. (2019, enero 20). Qué es HTML5: Definición y funcionamiento. Openwebinars.net. <https://openwebinars.net/blog/que-es-html5/>

MDN Web Docs. (s.f.). *Documentación para desarrolladores web.* <https://developer.mozilla.org/es/>

Muñoz, J. D. (2017, noviembre 17). Qué es Flask y ventajas que ofrece. Openwebinars.net. <https://openwebinars.net/blog/que-es-flask/>

Preguntas Frecuentes. (s/f). Plantuml.com. Recuperado el 6 de septiembre de 2024, de <https://plantuml.com/es-dark/faq>

¿Qué es Grafana? (s/f). Redhat.com. Recuperado el 5 de septiembre de 2024, de <https://www.redhat.com/es/topics/data-services/what-is-grafana>

Sánchez, A. (2023, noviembre 27). ¿Qué es Angular CLI? - Angel Sánchez. Angel Sánchez. <https://aseazul.com/que-es-angular-cli/>

Santos, D. (2023, julio 25). Introducción al CSS: qué es, para qué sirve y otras 10 preguntas frecuentes. Hubspot.es. <https://blog.hubspot.es/website/que-es-css>

Terreros, D. (2022, noviembre 14). Qué es Google Meet, cómo funciona y cómo utilizarlo. Hubspot.es. <https://blog.hubspot.es/marketing/como-usar-google-meet>

Wikipedia contributors. (s/f). WhatsApp. Wikipedia, The Free Encyclopedia. <https://es.wikipedia.org/w/index.php?title=WhatsApp&oldid=162098721>

(S/f). Iebschool.com. Recuperado el 5 de septiembre de 2024, de <https://www.iebschool.com/blog/progressive-web-apps-analitica-usabilidad/>

General

W3Schools. (s.f.). Tutorials. <https://www.w3schools.com/>

Google. (s.f.). Progressive Web Apps (PWA). <https://web.dev/explore/progressive-web-apps?hl=es-419>

PWA Experts. (s.f.). Convierte una aplicación Angular en PWA. <https://PWAExperts.io/tutoriales/convierte-aplicacion-angular-en-PWA>

Asimov Cloud. (2021). Creación de una aplicación web con Flask y Angular. <https://asimov.cloud/blog/programacion-5/creacion-de-una-aplicacion-web-con-flask-y-angular-41>

Ria Pacheco. (2020). Serve Angular app on local network for responsive device previews. <https://dev.to/riapacheco/wsn-serve-angular-app-on-local-network-for-responsive-device-previews-42ie>

Flask

Flask Documentation. (s.f.). Welcome to Flask. <https://flask.palletsprojects.com/en/2.0.x/>

Real Python. (s.f.). Flask Tutorials. <https://realpython.com/tutorials/flask/>

RealCode. (2023). Curso de Flask para principiantes - Tutorial para crear aplicaciones web. YouTube. https://www.youtube.com/watch?v=W-SfC_V7P6o

Traversy Media. (2019). Flask Crash Course. YouTube. https://www.youtube.com/watch?v=Z1RJmh_OqeA

Corey Schafer. (2018). Python Flask Tutorial: Full-Featured Web App. YouTube. <https://www.youtube.com/watch?v=Qr4QMBUPxWo>

Traversy Media. (2018). Flask & MySQL Tutorial. YouTube.

https://www.youtube.com/watch?v=01sAkU_NvOY

Angular

Angular Documentation. (2023). Angular - Application Design Overview. Retrieved from <https://angular.io/guide/architecture>

Angular Documentation. (s.f.). Introduction to Angular. <https://angular.dev/>

Angular University. (s.f.). Courses and Resources for Learning Angular. <https://angular-university.io/>

Academind. (s.f.). Angular Setup & First App. <https://academind.com/tutorials/angular-setup-first-app/>

Angular. (s/f). Angular.lat. <https://docs.angular.lat/guide/service-worker-intro>

Imagina, E. Cómo crear una PWA en Angular; Imagina Formación. <https://imagineformacion.com/tutoriales/como-crear-una-PWA-en-angular>

Progressive web apps in Angular. (s/f). Ionic Framework Docs. <https://ionicframework.com/docs/angular/PWA>

Codehouse Academy. (s/f). Progressive web application (PWA) con angular - dominicode. Youtube. https://www.youtube.com/watch?v=3uSnmMoBv_o

Mosh Hamedani. (2019). Angular Full Course. YouTube. <https://www.youtube.com/watch?v=3qBXWUpoPHo>

Traversy Media. (2019). Angular Crash Course. YouTube. <https://www.youtube.com/watch?v=htPYk6QxacQ>

Code Step By Step. (2021). Learn Angular 12, 13, 14 - Complete Course. YouTube. <https://www.youtube.com/watch?v=3tFYWxbAW2A>

Grafana

Grafana Labs. (s.f.). Grafana Documentation. <https://grafana.com/docs/grafana/latest/>

Grafana Labs. (s.f.). Grafana Tutorials. <https://grafana.com/tutorials/>

La programación, C.-R. (s/f). Aprende grafana en 10 minutos! Youtube.
https://www.youtube.com/watch?v=riFxqD_6XYI

TechGyro. (s/f). Elasticsearch and grafana - connectivity for visualisation. Youtube.
<https://www.youtube.com/watch?v=vzNJ39b9ao0>

Jha, V. (s/f). Grafana dashboard with ELASTICSEARCH datasource. Youtube.
<https://www.youtube.com/watch?v=jB8rEf49B1g>

ElasticSearch

Elastic. (s.f.). Elasticsearch Reference.
<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

Elastic. (s.f.). What is Elasticsearch?. <https://www.elastic.co/what-is/elasticsearch>

Fernandez, O. (2021, enero 7). Introducción a Elasticsearch. Aprender BIG DATA;

AprenderBigData. <https://aprenderbigdata.com/elasticsearch/>

Tech, S. (s/f). Clase 2: Crear, actualizar, eliminar y obtener documentos en ElasticSearch. Youtube. https://www.youtube.com/watch?v=04_ewY-f6FI

Official Elastic Community. (s/f). Primeros pasos con Elasticsearch - Jan 27th Elastic meetup. Youtube. <https://www.youtube.com/watch?v=uTg7w0prP6A>

Elasticsearch Guide. (2023). Getting Started with Elasticsearch. Retrieved from <https://www.elastic.co/guide/index.html>

13. Anexo - Icono PWA y Capturas de la Web y PWA

En este anexo se van a aportar documentos gráficos reales todos ellos capturados de la aplicación final, en formato monitor (pantalla de gran tamaño) y en pantalla de dispositivo móvil, salvo el propio ícono de la aplicación.

Icono de la PWA



Figura 74 - Ícono de la aplicación

Muestra de la aplicación PWA

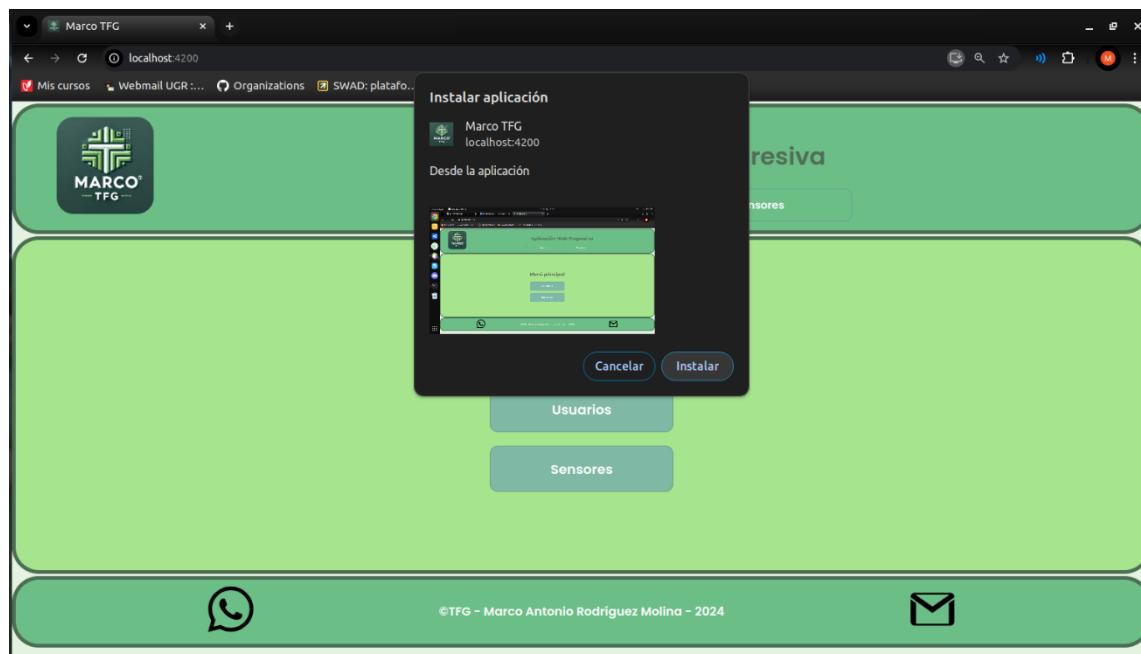


Figura 75 - Captura que muestra la instalación de la PWA

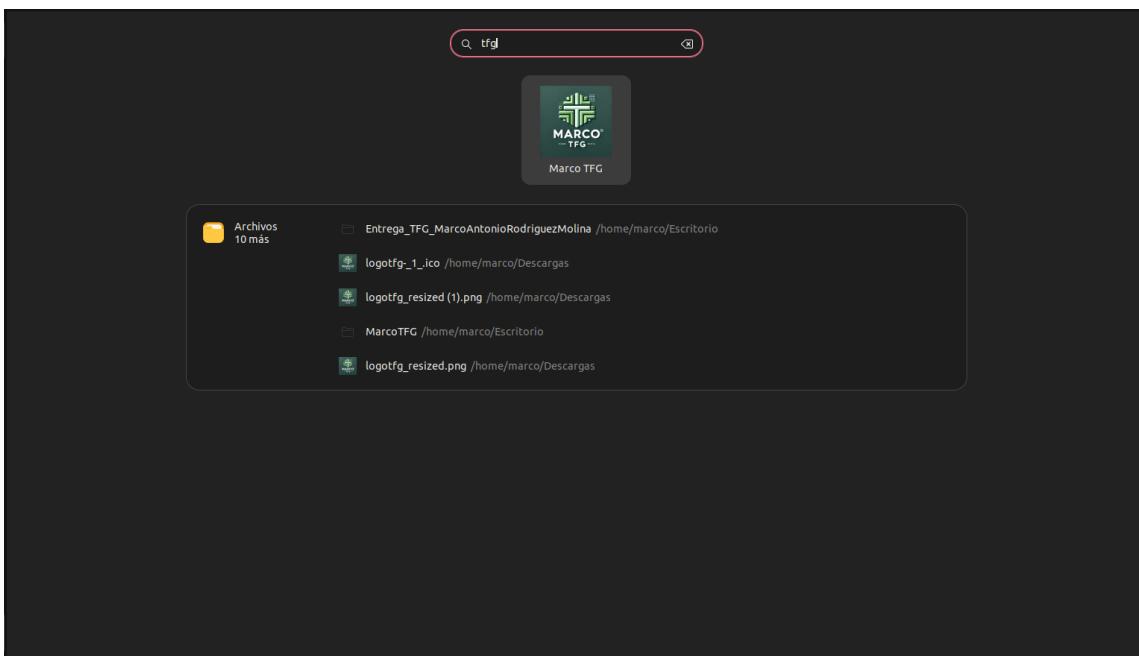


Figura 76 - Captura que muestra la aplicación PWA instalada en nativo



Figura 77 - Captura que muestra la vista desde la aplicación PWA

Dispositivo móvil



Figura 78 - Dispositivo móvil: Capturas de la página de inicio con notificación alerta y sin ella



Figura 79 - Dispositivo móvil: Captura de la página de novedades donde se muestra el dashboard que ha lanzado la alerta

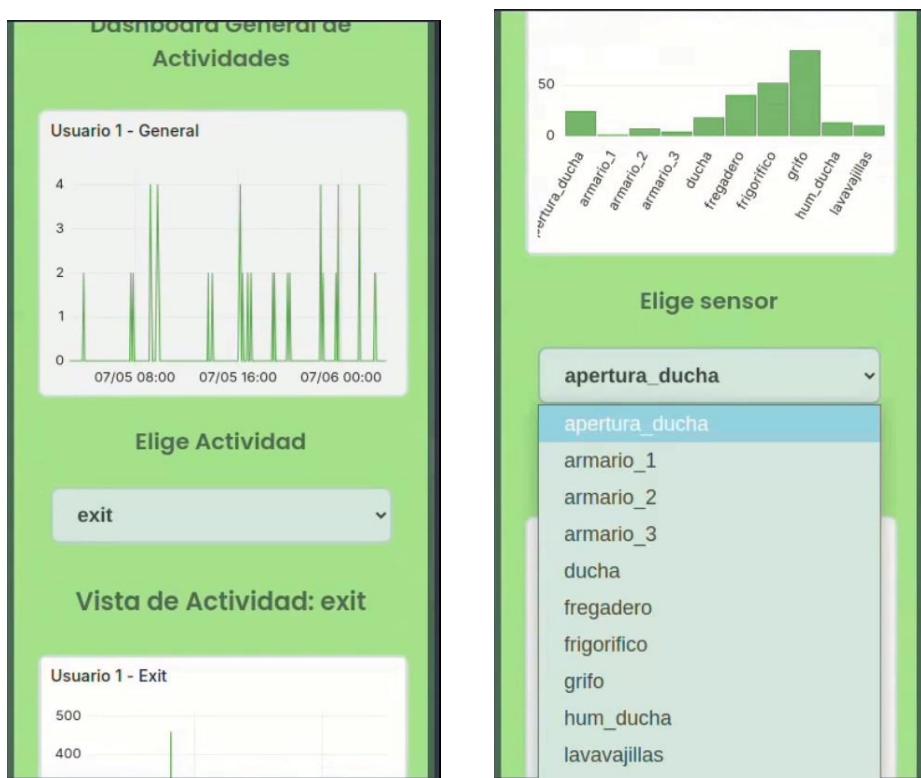


Figura 80 - Dispositivo móvil: Captura de pantalla que muestran las páginas de usuarios y sensores

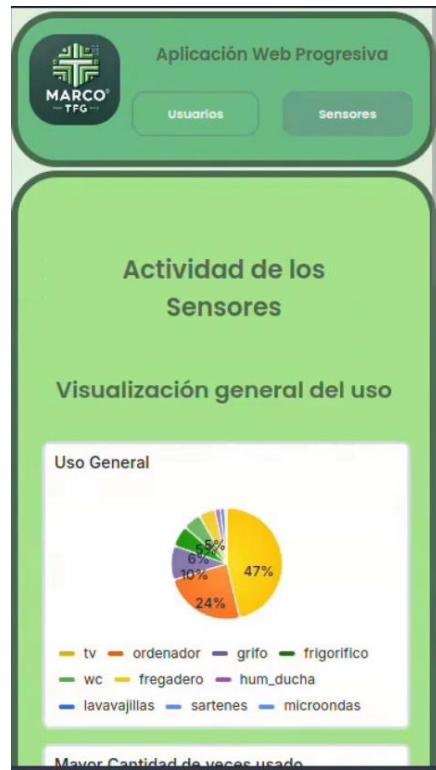


Figura 81 - Dispositivo móvil: Captura que muestra la página de sensores donde se ve la cabecera

Monitor



Figura 82 - Monitor: Captura de pantalla página de inicio sin notificación



Figura 83 - Monitor: Captura de pantalla página de inicio con notificación



Figura 84 - Monitor: Captura de la página de novedades



Figura 85 - Monitor: Captura del menú de selección de usuarios



Figura 86 - Monitor: Captura de la página de actividad de Usuario 1



Figura 87 - Monitor: Captura de la página de los sensores