TRABAJOS FIN DE GRADO

Universidad de Jaén

Escuela Politécnica Superior de Linares

Desarrollo de motor de informes IoT

Autor: Alejandro Núñez Rivas

Grado: Ingeniería en Telemática

Tutor: Prof. D. José Ángel Fernández Prieto

Departamento del tutor: Ingeniería de Telecomunicación

Tutor: Francisco Jesús Moreno Infantes

Empresa: Alibérico

Fecha: 10/02/2025

Universidad de Jaén

Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

Trabajo Fin de Grado

DESARROLLO DE MOTOR DE INFORMES IOT

Alumno: Alejandro Núñez Rivas
Tutor: Prof. D. José Ángel Fernández Prieto
Tutor: Francisco Jesús Moreno Infantes
Depto.: Ingeniería de Telecomunicación

Firma del autor

NUÑEZ RIVAS ALEJANDRO - 26511122B
Firmado digitalmente por NUÑEZ RIVAS ALEJANDRO - 26511122B
Fecha: 2025.02.10 23:19:43 +01'00'

Firma de los tutores

MORENO INFANTES FRANCISCO JESÚS - 26259282K
Firmado digitalmente por MORENO INFANTES FRANCISCO JESÚS - 26259282K
Fecha: 2025.02.11 12:56:51 +01'00'

FERNANDEZ PRIETO JOSE ANGEL - 26222606F
Firmado digitalmente por FERNANDEZ PRIETO JOSE ANGEL - 26222606F
Fecha: 2025.02.11 14:24:27 +01'00'

Índice

1. Índice de figuras y tablas

- **Figuras**

- **Tablas**

2. Resumen

En el presente Trabajo de Fin de Grado (TFG) se desarrolla una solución tecnológica destinada a resolver el problema identificado en la empresa Alibérico relacionado con la generación manual de gráficas y sus informes correspondientes. La metodología que se propone da lugar a una automatización de este proceso incrementando la eficiencia y reduciendo la posibilidad de que ocurran errores humanos. Además, se incorpora una capa de autenticación robusta que garantiza la seguridad del sistema junto con una gestión de los niveles de acceso que se adaptan a las necesidades específicas de la empresa.

Este sistema cuenta con una arquitectura basada en 3 componentes principales, un cliente web desarrollado con el *Framework Angular* al que se podrá acceder para generar estas gráficas, un servidor que contiene una *API* desarrollada en *Node.js* para la visualización y extracción de las variables de la base de datos, y la base de datos *MySQL* que servirá como almacenamiento de los datos locales que conlleva el ámbito local.

Además de estos componentes locales, el proyecto cuenta con una conexión al servidor remoto de Alibérico para obtener los datos de la red de sensores de la empresa en tiempo real, junto con 2 conexiones a repositorios privados en la nube de *Azure DevOps* para llevar un control de versiones del *Front-End* y del *Back-End*.

El servidor web dispone de 2 áreas principales en base al permiso del que disponga el usuario que inicie sesión.

Los usuarios con permiso *Administración* y *Total* disponen tanto de funciones para la gestión de usuarios y sensores en cuanto a modificación, creación y eliminación, como de visualización de los mismos.

Y en cuanto a los usuarios con permiso de *Configuración de gráficas*, disponen de la función principal por la que se generó la necesidad de este TFG, *Motor de informes de IoT*, en el que el usuario podrá generar gráficas recogiendo los datos del servidor de Alibérico en tiempo real en base a, distintos tipos de gráficas, sucursales, líneas de producción, y sensores que se cargaran de manera dinámica de la base de datos local. Además, este usuario dispondrá de las herramientas para guardar las gráficas para su posterior visualización y para la generación de informes en formato PDF (Portable Document Format, Formato de documento portátil) o en forma de URL (Uniform Resource Locator, localizador uniforme de recursos) única en la que se podrá compartir o inyectar desde otra aplicación.

3. Abstract

In this Final Degree Project, a technological solution is developed to address the issue identified in the company Alibérico related to the manual generation of graphs and their corresponding reports. The proposed methodology leads to the automation of this process, increasing efficiency and reducing the possibility of human errors. Additionally, a robust authentication layer is incorporated to ensure system security, along with an access level management system that adapts to the company's specific needs.

This system is based on an architecture consisting of three main components: a web client developed using the Angular Framework, which can be accessed to generate these graphs, a server that contains an API developed in Node.js for the visualization and extraction of variables from the database, and the MySQL database, which will serve as storage for the local data within the defined scope.

In addition to these local components, the project includes a connection to Alibérico's remote server to obtain real-time data from the company's sensor network, along with two connections to private repositories in Azure DevOps' cloud to manage version control for both the Front-End and the Back-End.

The web server provides two main functional areas based on the permissions assigned to the logged-in user.

Users with Administration and Total permissions have access to functionalities for user and sensor management, including modification, creation, and deletion, as well as data visualization.

Users with Graph Configuration permissions have access to the core feature that justified the need for this Final Degree Project, the IoT Report Engine, in which the user can generate graphs by retrieving data from Alibérico’s server in real time based on various parameters such as graph types, branches, production lines, and dynamically loaded sensors from the local database. Additionally, these users have tools to save graphs for future reference and to generate reports in PDF format (Portable Document Format) or as a unique URL (Uniform Resource Locator), which can be shared or embedded in other applications.

4. Introducción

En el presente Trabajo de Fin de Grado se aborda una de las problemáticas más recurrentes en su operativa diaria: la dependencia de procesos manuales para la recopilación, procesamiento y representación visual de datos. La automatización de estos procedimientos no solo incrementa la eficiencia operativa, sino que también reduce significativamente el margen de error humano, mejorando la precisión y fiabilidad de la información presentada.

Para garantizar un sistema robusto, escalable y seguro, se ha diseñado una arquitectura modular basada en tres componentes principales:

- Cliente web desarrollado con el *Framework Angular*, que proporciona una interfaz intuitiva y accesible desde cualquier navegador para la generación, personalización y gestión de gráficas dinámicas.

- *API* implementada en *Node.js*, encargada de la lógica del negocio, la extracción y procesamiento de las variables almacenadas en la base de datos, así como la comunicación con el cliente web.

- Base de datos *MySQL*, utilizada para el almacenamiento estructurado de datos locales, permitiendo consultas eficientes y asegurando la persistencia de la información en el ámbito interno de la empresa.

Además de estos componentes locales, el sistema está diseñado para integrarse con la *API* remota de Alibérico, desde donde se obtienen los datos en tiempo real de la red de sensores de la empresa. Esto permite que los informes generados reflejen con precisión el estado actual de los procesos y faciliten la toma de decisiones basadas en datos actualizados. Asimismo, el proyecto cuenta con dos conexiones a repositorios de código en la nube mediante *Azure DevOps*, uno destinado al desarrollo y mantenimiento del cliente web en Angular, y otro para la *API* en *Node.js*, lo que garantiza un control eficiente de versiones, facilita la colaboración en el desarrollo y permite una evolución estructurada del sistema a lo largo del tiempo.

Para garantizar la seguridad y el control del sistema, se incorpora una capa de autenticación avanzada, que gestiona los niveles de acceso en función de los permisos asignados a cada usuario. Esto asegura que solo los perfiles autorizados puedan acceder a funciones específicas, los posibles valores de permiso que pueden ser adquiridos:

- *Administración*, caracterizado por poder gestionar los distintos elementos de la base de datos como son los usuarios o las empresas términos de creación, modificación, eliminación o consulta.

- *Configuración de gráficas*, designado para la creación de gráficas e informes en base a conjuntos de estas tanto en formato PDF como URL única para su posterior consulta.

- *Total*, diseñado para poseer todas las funcionalidades que posee la aplicación web.

Asimismo, el proyecto cuenta con un sistema de gestión de versiones basado en dos repositorios privados en la nube mediante *Azure DevOps*, lo que permite un control exhaustivo del desarrollo del *Front-End* y el *Back-End*, asegurando la integridad del código y facilitando futuras actualizaciones o mejoras.

Primero, se realizará un estado del arte con un estudio acerca de las posibilidades que existen acerca de una librería para generar gráficas, el sistema de control de versiones y un *Framework* para levantar el servidor local de *Node.js*. Así como las ventajas y desventajas de cada una de estas tecnologías junto con la solución adoptada finalmente para cada aspecto mencionado.

Seguidamente, se presentará una descripción del sistema en la que se mostrará la arquitectura final del sistema con todas las interconexiones y tecnologías que forman parte de la solución junto con el desarrollo teórico de las tecnologías mencionadas.

A continuación, se comentarán los problemas e inconvenientes que se han presentado a lo largo del desarrollo del TFG y el impacto que han podido producir en la realización del mismo.

Una vez comentado los problemas en el desarrollo, se comentarán las posibles líneas de futuro que presenta el actual trabajo como posibilidades de continuación para ampliar las funcionalidades que ya contiene.

Adicionalmente en un anexo, se ha descrito un manual de usuario para que la aplicación una vez implementada en el sistema de la empresa de Alibérico pueda ser consultada por el futuro usuario que utilice la aplicación en caso de que tenga alguna duda sobre el funcionamiento de la misma.

Finalmente, se ha desarrollado un último anexo sobre una guía de implementación para que el administrador que se disponga a implementar la solución en el entorno de Alibérico pueda consultarlo en caso de que tenga alguna duda sobre cómo desplegar los distintos elementos que componen la solución.

5. Objetivos

El objetivo principal de este TFG es diseñar una aplicación web para representar las señales de proceso de las líneas de fabricación de una forma gráfica. En la misma, un usuario debe poder seleccionar entre un conjunto de señales, un rango de fechas y el tipo de gráfico que quiere utilizar para mostrarla. Además, el sistema debe permitir guardar configuraciones de informes que estarán disponibles a distintos usuarios. El TFG se ha llevado cabo en colaboración con la empresa Alibérico.

Para alcanzar este objetivo principal se han determinado unos objetivos secundarios más específicos que se detallan a continuación:

- Habilitar una zona de gestión administrativa para gestionar las distintas entidades y relaciones que componen el sistema como dar de alta/baja, modificar usuarios, empresas, etc.
- Dotar la aplicación de una capa de seguridad en base al permiso que disponga el usuario que inicie sesión.
- Obtener los datos en tiempo real mediante una conexión a la plataforma de Alibérico
- Establecer seguridad en el despliegue de la aplicación web con HTTPS
- Poder acceder a la aplicación mediante un usuario registrado con directorio activo
- Poder guardar los informes en PDF o incrustar el mismo en otras webs.

6. Antecedentes

Este TFG surge de la necesidad de la empresa Alibérico de implementar un sistema para que el personal de todas sus sedes pueda generar gráficas en base a los datos que generan la red de sensores de *IoT(Internet of Things)*[1] que disponen las instalaciones de forma dinámica e instantánea.

Hasta el momento el equipo de TI (Tecnologías de la Información) tenían que programar una página nueva bajo demanda de los operarios con las gráficas que se necesitaran en el momento, además de luego tener que sacar un recorte de pantalla para enviar los resultados dentro de un informe que también tenían que realizar manualmente.

Esta dinámica supone un gasto de recursos humanos extra que podría verse amedrentado en el caso de que existiera una aplicación con la que el equipo de programadores de la empresa pudiera generar una serie de gráficas de manera dinámica e instantánea en las que mostraran los datos que se requirieran en el momento además de generar un informe automáticamente en base a esas gráficas.

[1] Interconexión colectiva de dispositivos electrónicos junto con la tecnología que produce esta conexión.

7. Estado del arte

7.1. Librería para generar gráficas

En primer lugar, se ha realizado un estudio para analizar las diferentes soluciones para resolver el problema de buscar una librería[2] compatible con *Angular* 18 y *TypeScript* para que recoja los datos de una *API (application programming interface)* [3] externa, y los transforme en distintos tipos de gráficas como son lineales, de barras o circulares. Además de la posibilidad de elegir el rango temporal de la gráfica en cuestión y poder añadir series extras para tener la posibilidad de contrastar información en una misma gráfica.

Chart.js es una librería de gráficas simple y ligera diseñada para aplicaciones web que requieran gran personalización de estilos visuales, cuenta con una ligereza excepcional debido a su pequeño tamaño comparado a otras librerías similares contando con aproximadamente 60KB (Kilo Bytes). También cuenta con una fácil integración con proyectos de *Angular* debido a su sólido soporte con *TypeScript*, además de que es un software de código abierto[4]. Aunque, por el contrario, su tamaño provoca serias limitaciones en cuanto a rendimiento se refiere, debido a que no soporta tipos de gráficos avanzados como mapas o gráficos en 3D junto a que no ofrece interactividad básica con la gráfica.

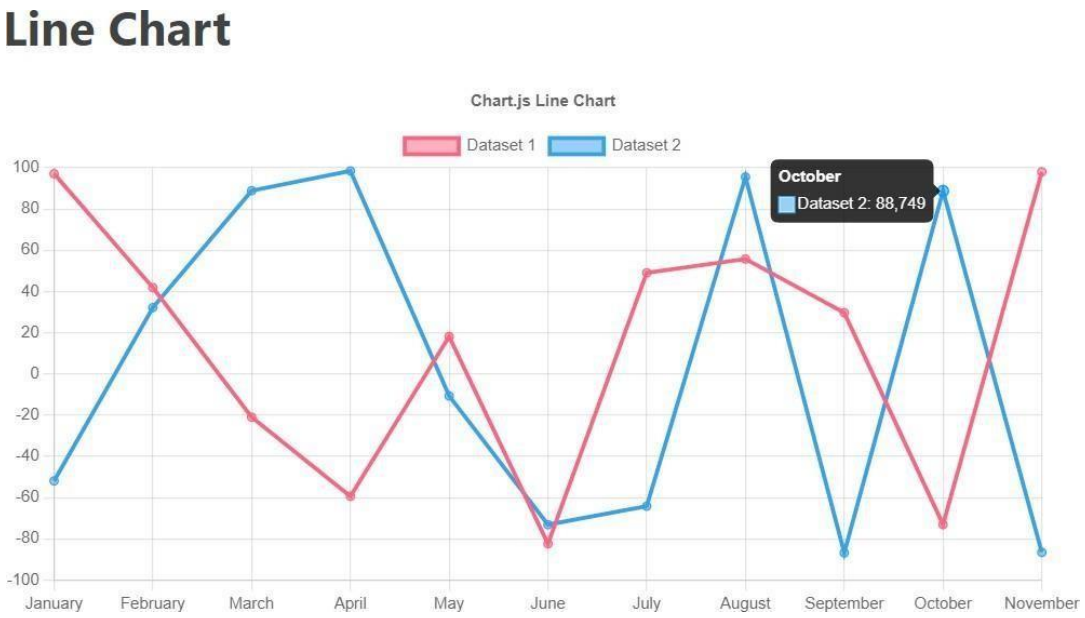

Figura 1: Gráfica de ejemplo de Chart.js

Fuente: https://www.chartjs.org/docs/latest/samples/line/line.html

[2] Fragmentos de código desarrollados con fines de resolver inconvenientes en programación o incrementar la eficiencia.
[3] Conjunto de herramientas que facilitan la comunicación de datos entre diferentes aplicaciones y sistemas.
[4] Libre uso para que cualquier desarrollador pueda utilizar el código fuente de una aplicación.

Highcharts es conocida por su flexibilidad, facilidad de uso y soporte empresarial. Soporta tipos de gráficos simples como complejos como mapas y 3D, cuenta con una excelente documentación especialmente útil para desarrolladores que trabajen con *Angular* y *TypeScript*, una IA (Inteligencia Artificial) que mediante un chat que proporciona todo tipo de información y ayuda respecto a la librería e incluye un soporte técnico y mantenimiento continuo ideal para aplicaciones críticas. Además de que el equipo de TI de Alibérico ha trabajado con esta librería con anterioridad, lo que supone que en el futuro puedan resolver incidencias de manera más eficiente. Por contra parte, este soporte viene dado junto con una licencia comercial de pago, además de que esta librería tiene dependencia de su ecosistema teniendo que integrar librerías adicionales (como *HighStock* o *Highmaps*) lo que puede llevar a retrasos y dificultades en el desarrollo.

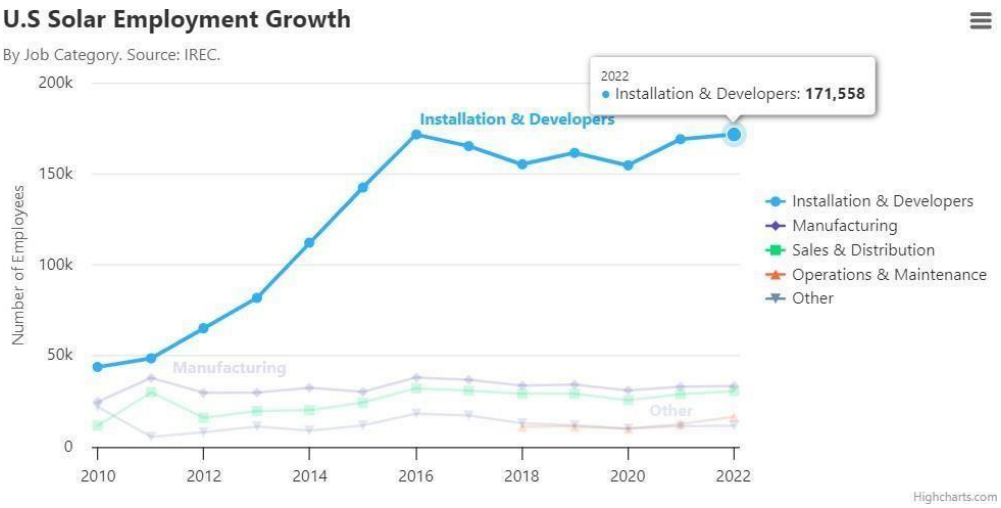

Figura 2: Gráfica de ejemplo de Highcharts

Fuente: https://www.highcharts.com/demo/highcharts/line-chart

ECharts está desarrollada por *Apache* y cuenta con un gran soporte para gráficos avanzados y una rica interactividad con las gráficas que se desplieguen, soporta tipos de gráficos complejos como los mapas y los gráficos en 3D, contiene excelentes opciones de interacción con las gráficas como *zoom* o *tooltip*[5] dinámico. A su vez cuenta con una gran optimización para manejar grandes cantidades de datos sin tener que sacrificar la fluidez del sistema. Todas estas características positivas y robustez se ven reflejados en su gran tamaño, pesando aproximadamente 500KB lo cual puede llegar a afectar a al rendimiento a algunas aplicaciones ligeras y tantas características conllevan mayor complejidad a la hora de aprender a utilizar esta librería por lo que la curva de aprendizaje es mayor debido a su extensión.

[5] Elemento gráfico que aparece temporalmente cuando el usuario interactúa con un componente.

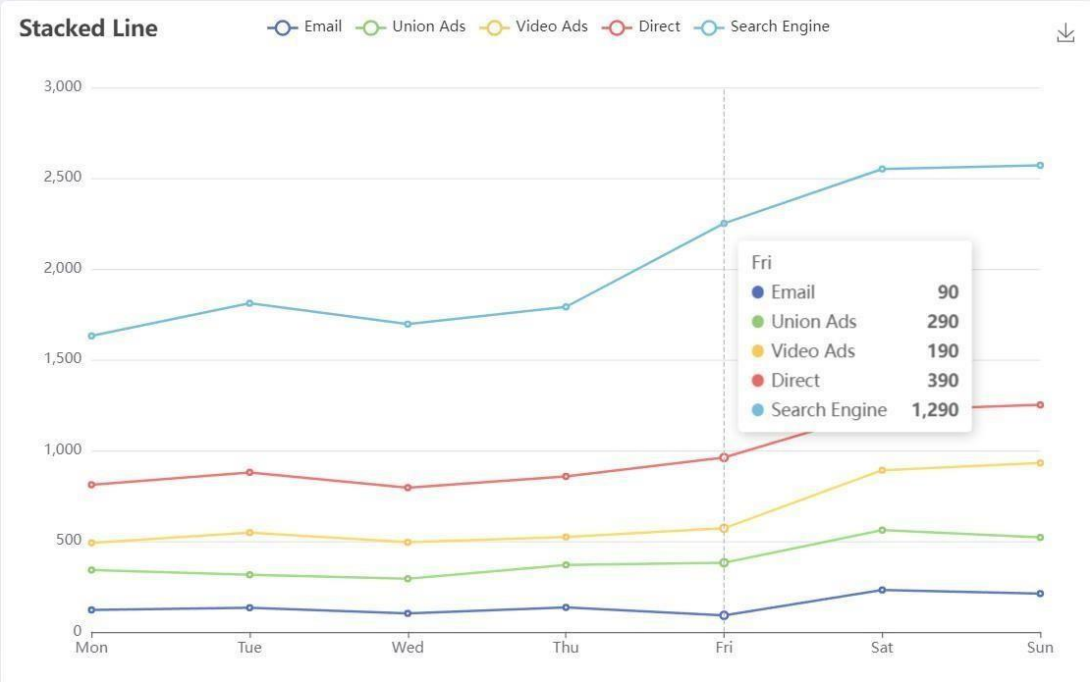

Figura 3: Gráfica de ejemplo de ECharts

Fuente: https://echarts.apache.org/examples/en/editor.html?c=line-stack

Una vez analizadas las diferentes características de las librerías elegidas se ha llegado a la conclusión de elegir la librería de *Highcharts* por siguientes factores: cuenta con un peso razonable sin llegar a ser demasiado liviano como para estar limitado en cuanto a funciones como *Chart.js* y sin ser demasiado pesado como *ECharts* para que afecte al rendimiento de la aplicación. Además de contar con una alta interactividad para visualizar mejor las gráficas y tener el soporte empresarial que beneficia en gran medida el mantenimiento de la aplicación contando con el ámbito en que se desarrolla de la empresa Alibérico. Y el coste no supone un problema ya que la empresa dispone de una licencia para el uso de esta librería.

7.2. Control de versiones

Tras haber realizado un análisis exhaustivo acerca de las distintas librerías disponibles para realizar las gráficas de la aplicación, el siguiente apartado se centrará en evaluar distintas soluciones de control de versiones, un componente clave para la gestión y organización del desarrollo colaborativo.

Git es un sistema de control de versiones distribuido diseñado para ser rápido, flexible y adecuado para proyectos de pequeña o gran escala. Al ser distribuido cada desarrollador dispone de una copia del historial de cambios del repositorio lo que le permite trabajar sin conexión y redundancia. Las operaciones locales como *commits*[6] o *merges*[7] son

[6] Acción de Git que permite guardar de manera permanente los cambios realizados en los archivos de un repositorio.
[7] Acción de Git que permite anexionar la rama actual de trabajo con la rama principal de un proyecto.

extremadamente rápidas al no depender de un servidor central, además cuenta con una gran comunidad que proporciona amplio soporte, documentación y herramientas, contando también con una integración nativa entre la nube de *Azure DevOps* y *Visual Studio Code*. Por el contrario, *Git* puede llegar a ser algo complejo para desarrolladores novatos debido a su terminología y comando más complejos, junto con que la gestión de ramas remotas puede ser confuso especialmente en proyectos grandes.

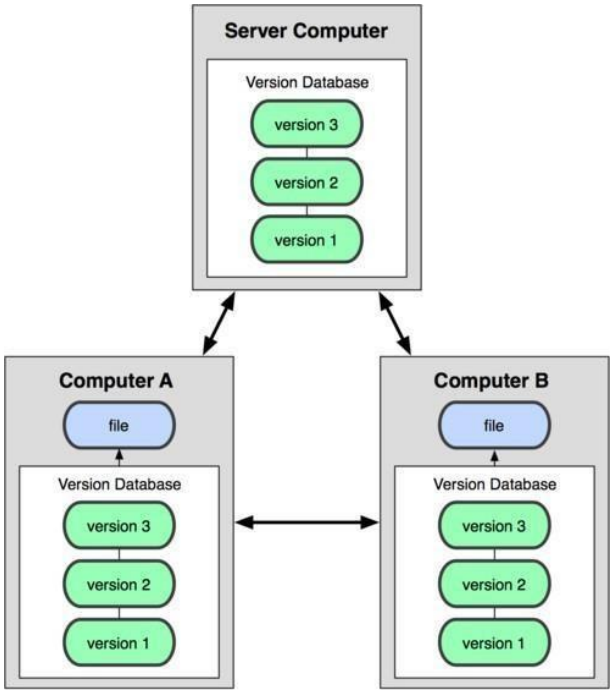

Figura 4: Control de versiones distribuido

Fuente: https://aulasoftwarelibre.github.io/taller-de-git/cvs/

Subversion (SVN) ofrece un enfoque tradicional basado en un servidor central además de ser un sistema centralizado. Su diseño centralizado facilita la accesibilidad a nuevos usuarios por la simplicidad de sus comandos, es ideal para proyectos que requieran de una supervisión central de los cambios que se realicen además de ser adecuado para proyectos pequeños o sin múltiples ramas activas al mismo tiempo. En cambio, su dependencia de un servidor central provoca una falta de redundancia que provoca que los usuarios no puedan realizar algunas operaciones, además las operaciones requieren de una comunicación constante con este servidor lo que ralentiza el flujo de trabajo.

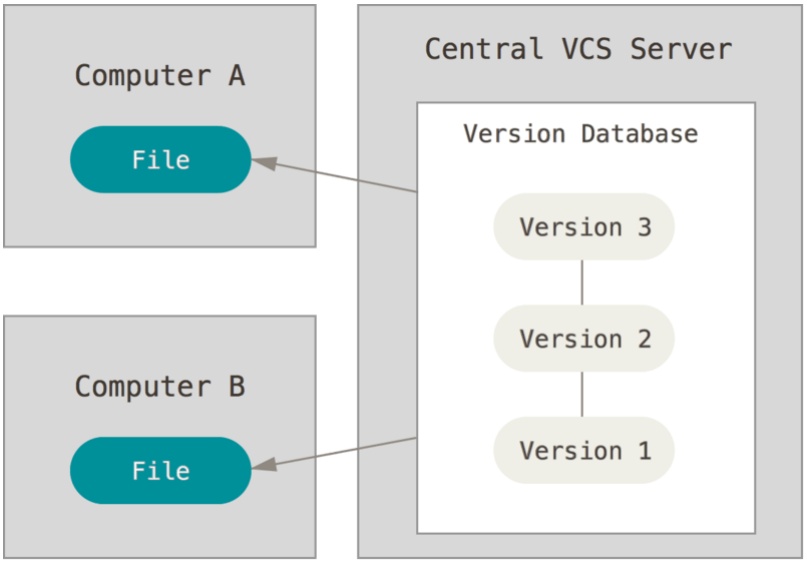

Figura 5: Sistema de control de versiones centralizado.

Fuente: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

Mercurial ofrece un enfoque similar a *Git* debido a su carácter distribuido, pero cuenta con mayor simplicidad. Dispone de una interfaz menos compleja de usuario y, además permite trabajar sin conexión y que cada uno de los desarrolladores tengan una copia del historial de versiones. Sin embargo, al crecer en popularidad *Git*, ha decrecido la comunidad de este control de versiones por lo que ya no posee tanto soporte para herramientas como había hace unos años, junto con que carece de la inmensa variedad de *plugins*[8] y herramientas adicionales que posee *Git*.

Tras haber realizado un análisis respecto a todo lo que ofrece cada una de los sistemas de control de versiones se ha determinado que el mejor sistema que cumple con los requisitos y cumpliría mejor la solución resultaría ser *Git* debido a su inmensa comunidad en continuo crecimiento respecto a las demás propuestas que han estado decreciendo en popularidad debido al auge de *Git*, además de la integración nativa de la nube con el *IDE* que resulta en un ahorro muy considerable de tiempo y esfuerzo en implementar la solución.

7.3. Framework para levantar la API del servidor local con Node.js

En el desarrollo del servidor se ha considerado una amplia gama de *Frameworks* que complementen la funcionalidad del entorno de trabajo de *Node.js* ofreciendo la posibilidad de levantar un servidor de recursos local que haga una función de intermediario entre el cliente y la base de datos, además de una visualización de los datos.

Fastify se encuentra diseñado para ser rápido y fácil de utilizar, fue inspirado en Express y ofrece una experiencia optimizada para el entorno de *Node.js*. Este *Framework*

[8] Componente de software adicional para complementar la funcionalidad de la aplicación principal.

posee un rendimiento muy alto debido a su alta velocidad pudiendo llegar a tratar con 30.000 peticiones por segundo, además de ser muy extensible por la capacidad de añadirle *plugins* y su mantenimiento para las declaraciones de archivos *TypeScript*. Sin embargo, aunque esté basada en *Express* no posee su popularidad por lo que no dispone de la misma cantidad de recursos y adopción y conlleva peor compatibilidad para algunos paquetes *npm*.

Express es el *Framework* más popular para construir servidores web con *Node.js*, conocido por su simplicidad, flexibilidad y amplia comunidad de soporte. Su *API* además de ser muy sencilla cuenta con una gran flexibilidad para gestión de rutas, *middlewares*[9] y peticiones *HTTP*, junto con su ampliada variedad de documentación y cantidad de recursos. Aunque no llega a ofrecer un rendimiento tan eficiente en operaciones intensivas como *Fastify* o su carencia de estructura definida puede llegar a dar lugar a una desorganización en el desarrollo.

Koa, desarrollado por *Express*, es un *Framework* minimalista que utiliza funciones asíncronas para ofrecer una experiencia moderna y sencilla. Su visión minimalista ofrece las funciones esenciales para construir el servidor a medida, al prescindir de muchas características provoca que sea el más liviano de todas las opciones, además de que ofrece la posibilidad de configurar los middlewares con total libertad del desarrollador sin imponer configuraciones predeterminadas.

Después de haber recopilado tanto las ventajas como las desventajas de las posibles soluciones frente al problema de la elección del *Framework* de *Node.js* para levantar el servidor de recursos local se ha decidido optar por el *Express* como el designado para resolver el problema debido a que resulta ser el *Framework* con mayor documentación y comunidad para obtener información, además de que no requerimos de una velocidad alta para el sistema por lo que no sería imprescindible utilizar *Fastify*.

[9] Componente software dispuesto entre diferentes aplicaciones y sistemas para procesar solicitudes y respuestas.

8. Descripción del sistema

8.1. Arquitectura del sistema

El sistema se encuentra dividido por ámbitos en local y remoto. En el ámbito remoto se encuentran el servidor de la red de datos en tiempo real de Alibérico y un servidor de almacenamiento para las copias de seguridad en la nube del cliente web y otro para el servidor de la aplicación. En el ámbito local se encuentra implementado el cliente web que está conectado a la *API Node.JS* y este a su vez a una base de datos local *MySQL*.

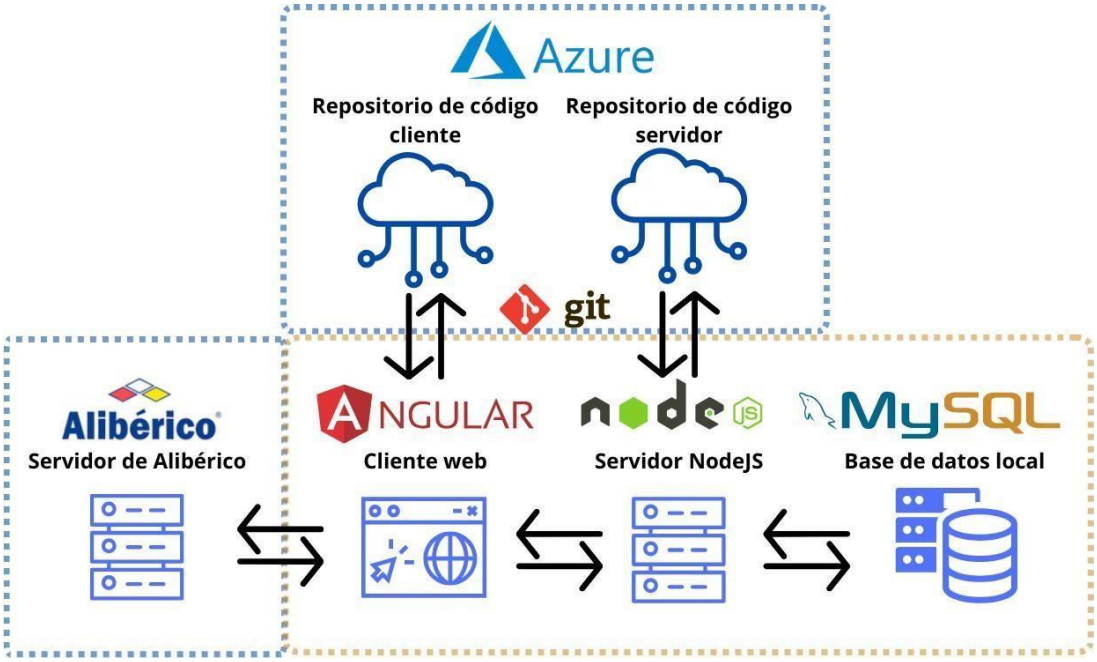

Figura 6: Arquitectura del sistema

En el ámbito local del sistema el cliente web se ha programado utilizando el *Framework Angular*, concretamente la versión 18.2.8, con las tecnologías *HTML*, *CSS (Cascading Style Sheets)* y *TypeScript*. Este cliente web se conecta a la *API* para obtener los datos locales y esta mediante la tecnología *TypeORM (Type Object-Relational Mapping)* traduce peticiones *HTTP* a sentencias *SQL* para realizar consultas a la base de datos. Y por último este servidor de recursos está conectado a una base de datos local *MySQL* que contiene todas las estructuras de datos que se utilizan en la aplicación web.

En el ámbito remoto se dispone del servidor de recursos privado de Alibérico, en el cual se dispone de información de cada sensor como puede ser, el nombre del mismo o la función que desempeña, además de vectores de datos en formato *JSON (JavaScript Object*

Notation)[10] limitados mediante una fecha inicio y una fecha fin en la cual se encuentran los datos de las mediciones del sensor en cuestión. Además, se encuentran dos repositorios de código alojados en la nube de *Azure DevOps* que desempeñan la función de, respectivamente, almacenar las copias de seguridad del código del cliente web y del servidor *Node.JS*.

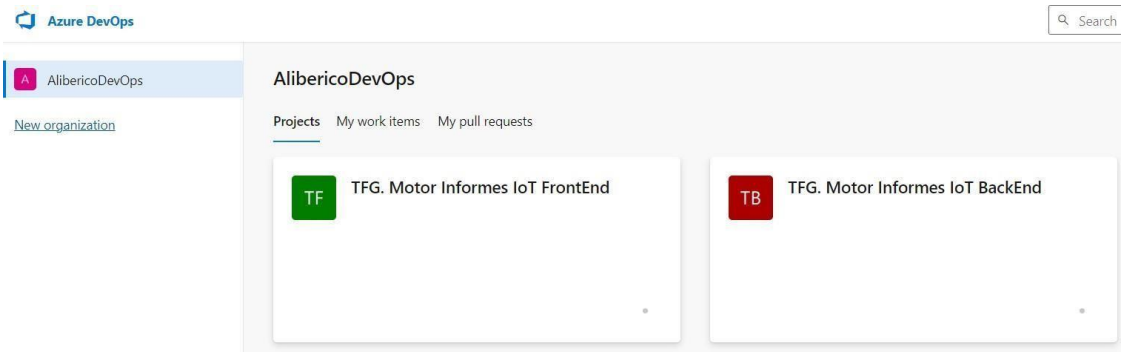

Figura 7: Panel principal de la nube de Azure DevOps

[10] Formato de intercambio de datos que representa estos datos por pares de claves con sus valores.

8.2. Cliente web

8.2.1. Angular

Un *Framework* es un marco de trabajo o forma de desarrollar aplicaciones que brinda una estructura inicial a partir de la cual se cimienta el proyecto para poder estructurarlo y desarrollarlo de manera más efectiva y organizada.

Angular es un ejemplo de *Framework* muy eficiente para proyectos *SPA (Single Page Application)* los cuales se caracterizan por tener una sola única página *HTML*. El gestor de rutas de *Angular* permite poder gestionar rutas sin tener que recargar página, su división en componentes lo convierte en una herramienta muy versátil debido a su capacidad de modificar solo ciertos aspectos de la aplicación o de la movilidad de variables mediante su *Data Binding* que sincroniza la interfaz con los datos de forma dinámica.

La estructura de carpetas que se utiliza a la hora de desarrollar en este *Framework* consiste en una carpeta denominada con el nombre del proyecto, seguido de una carpeta *source*, de manera abreviada *src* y dentro de esta una carpeta *application*, de manera abreviada *app* en la cual se destinarán los compontes que se generen.

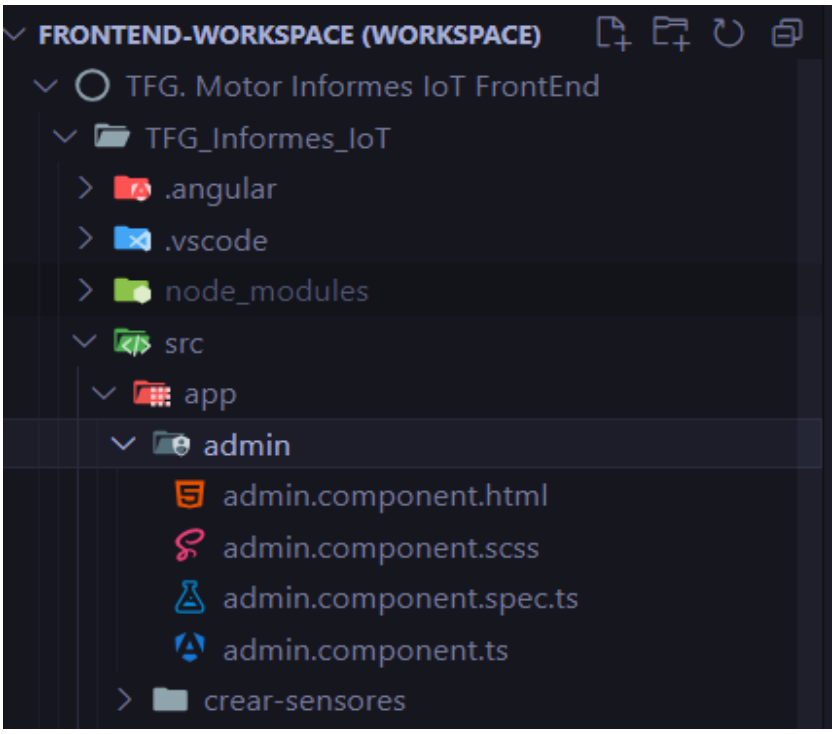

Figura 8: Estructura de ficheros del Frontend en VS Code

El marco de trabajo de *Angular* consiste en una estructura basada en componentes y en un enrutador que produce una navegación entre ellos, los componentes se crean utilizando el comando "*ng generate component <nombre_del_componente>*" en el *CLI*[11] *(Command-Line Interface)*, lo que resultará en la generación de una carpeta en la cual se genera un fichero *HTML* para la estructura del componente, otro *CSS* para los estilos y la visualización de la vista y dos archivos *TypeScript*, el archivo con la extensión *spec.ts* para pruebas unitarias del propio componente, además del fichero con extensión *.ts* en el que se desarrollará el dinamismo de la aplicación web.

Este *Framework* cuenta también con un sistema de monitorización de cambios llamado *Live reload* el cual automáticamente detecta los cambios que realicemos en el código una vez desplegado y vuelve a compilar el código para así ahorrar mucho tiempo al desarrollador de compilar el código cada vez que se produzcan cambios en el mismo, así mismo esta funcionalidad detecta si los cambios del código afectan a la funcionalidad, y en caso de que no afecten a esta como podría ser la adición de una línea comentada, el *CLI* avisa de que no se habrán efectuado cambios.

En el archivo *app.component.html* se inyecta la etiqueta del enrutador y entonces la gestión de las vistas deriva en el mismo. En el fichero *app.routes.ts* se encuentran definidas las rutas de la aplicación junto al componente que corresponde y además de informar si se encuentra protegido por el *AuthGuard*. Este *Guard*[12] se encarga de restringir el acceso a las vistas en base a una variable *permiso* enlazada a los usuarios que designa a que vistas tiene acceso.

```
  { path: 'admin', component: AdminComponent, canActivate: [AuthGuard] },
  { path: 'main/graficasGuardadas', component: GraficasGuardadasComponent, canActivate: [AuthGuard] },
  { path: 'main/motorGraficas', component: MainComponent, canActivate: [AuthGuard] },
  { path: 'main/informes/:idInformeGlobal', component: InformesComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: '**', redirectTo: 'login' } // Redirigir a login por defecto
];
```

Figura 9: Archivo de rutas app.routes.ts

[11] Interfaz de usuario basada en texto que permite la ejecución de comandos al sistema operativo o software.
[12] Funcionalidad de Angular que proporciona una capa de seguridad en el enrutamiento de la aplicación web.

```
  if (user.permiso === 'Configuración de gráficas') {
    if (requestedUrl.startsWith('main')) {
      return true;
    } else {
      this.router.navigate(['/main/motorGraficas']);
      return false;
    }
  }

  this.router.navigate(['/login']);
  return false;
}
```

Figura 10: Archivo AuthGuard

Una vez creados los componentes y el enlace con el enrutador se despliega la aplicación con el comando "*ng serve*" en el *CLI* y se levantará la aplicación web de manera predeterminada en la ruta "*http://localhost:4200/*"

```
PS C:\Users\anune.ALEX.000\Desktop\Universidad\4º de Carrera\TFG\AzureDevOps\TFG. Mot
IoT> ng serve
Initial chunk files | Names         |  Raw size
main.js             | main          | 474.62 kB |
polyfills.js        | polyfills     |  90.20 kB |
styles.css          | styles        | 200 bytes |

                    | Initial total | 565.02 kB

Application bundle generation complete. [6.141 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  ➜  Local:   http://localhost:4200/
  ➜  press h + enter to show help
```

Figura 11: Despliegue de la aplicación sobre HTTP en el CLI

8.2.2. HTML

HTML (HyperText Markup Language) es una tecnología que funciona como el componente más básico de una aplicación web definiendo mediante etiquetas los distintos contenidos que se muestran en la página. Su funcionamiento generalmente suele ir conjuntado de otras tecnologías como *CSS*, encargado de proporcionar los diferentes estilos a las marcas de *HTML*, y *JavaScript* o alguna sus variaciones como podría ser *TypeScript* cuya funcionalidad recae en proporcionar un comportamiento dinámico a la página web desplegada.

8.2.3. CSS

CSS es un lenguaje basado en estilos que sirve para describir la presentación de documentos *HTML* o *XML (Extensible Markup Language)* entre otras derivaciones de *XML*. Describe cómo debe ser renderizado un componente en términos de tamaño, color, alineamiento, forma, etc. Nos permite separar el contenido respecto de su presentación lo que repercute en mayor eficiencia a la hora de desarrollar un proyecto.

Este lenguaje nos permite seguir un diseño responsivo que permita la correcta visualización de todos los elementos de la aplicación web en multitud de dispositivos y resoluciones.

Para garantizar que la aplicación se ajuste a los diferentes tipos de dispositivos que se conecten a la aplicación web, esta ha sido desarrollada teniendo en cuenta las excepciones que suponen diferentes resoluciones en las distintas vistas que contiene la aplicación. Este comportamiento se realiza mediante el decorador *@media* en el fichero *CSS* que permite establecer una condición entre paréntesis y en el caso de que se cumpla esa condición aplica el código que se haya programado dentro de los corchetes.

```css
@media (max-width: 1023px) {

  #logo-alucoat {

  width: 12ch; /* Reducir el tamaño */
  position: static; /* Quitar posicionamiento relativo */
  margin: 1ch auto; /* Centrar la imagen */
  float: none; /* Quitar el float */
  margin-left: 10ch;
  margin-top: 0ch;

  }
}
```

Figura 12: Código para comportamiento responsive

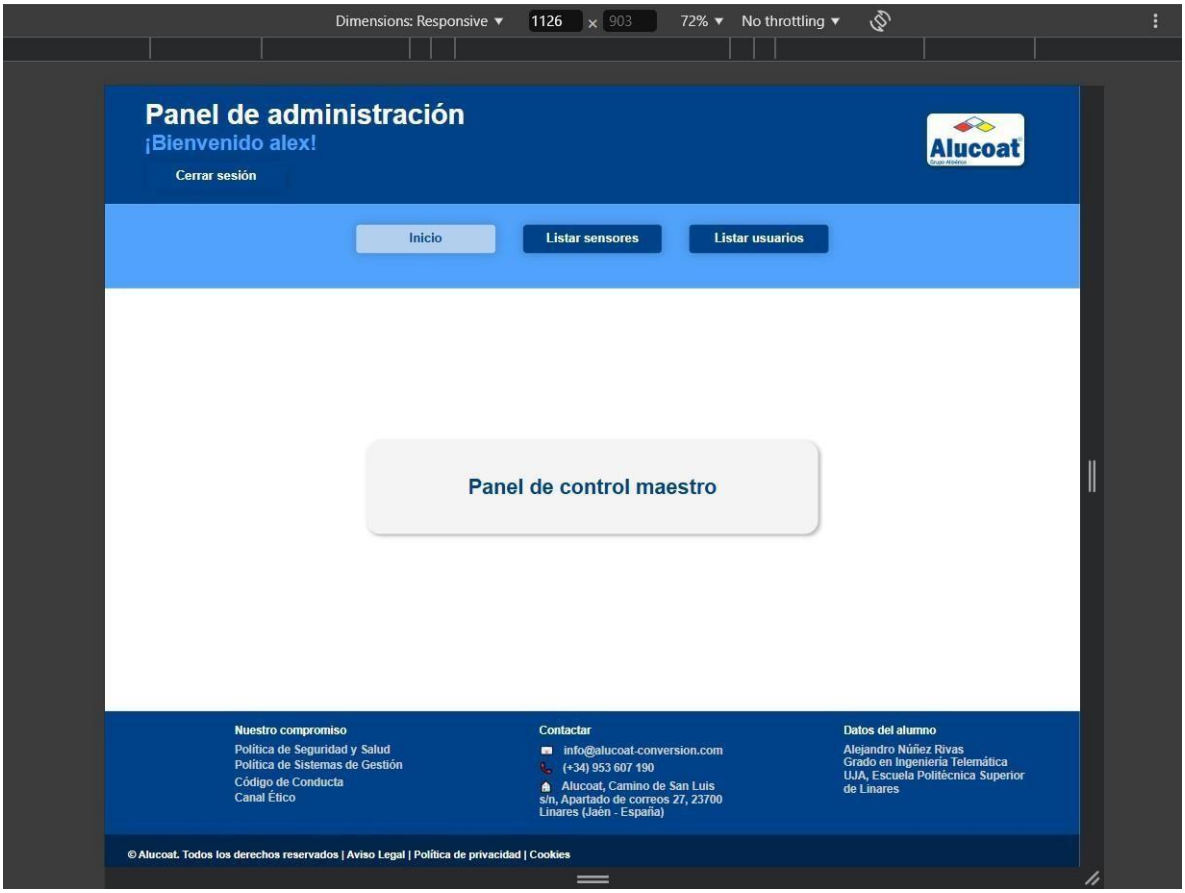

Figura 13: Herramienta de Google para modificar la resolución de la vista

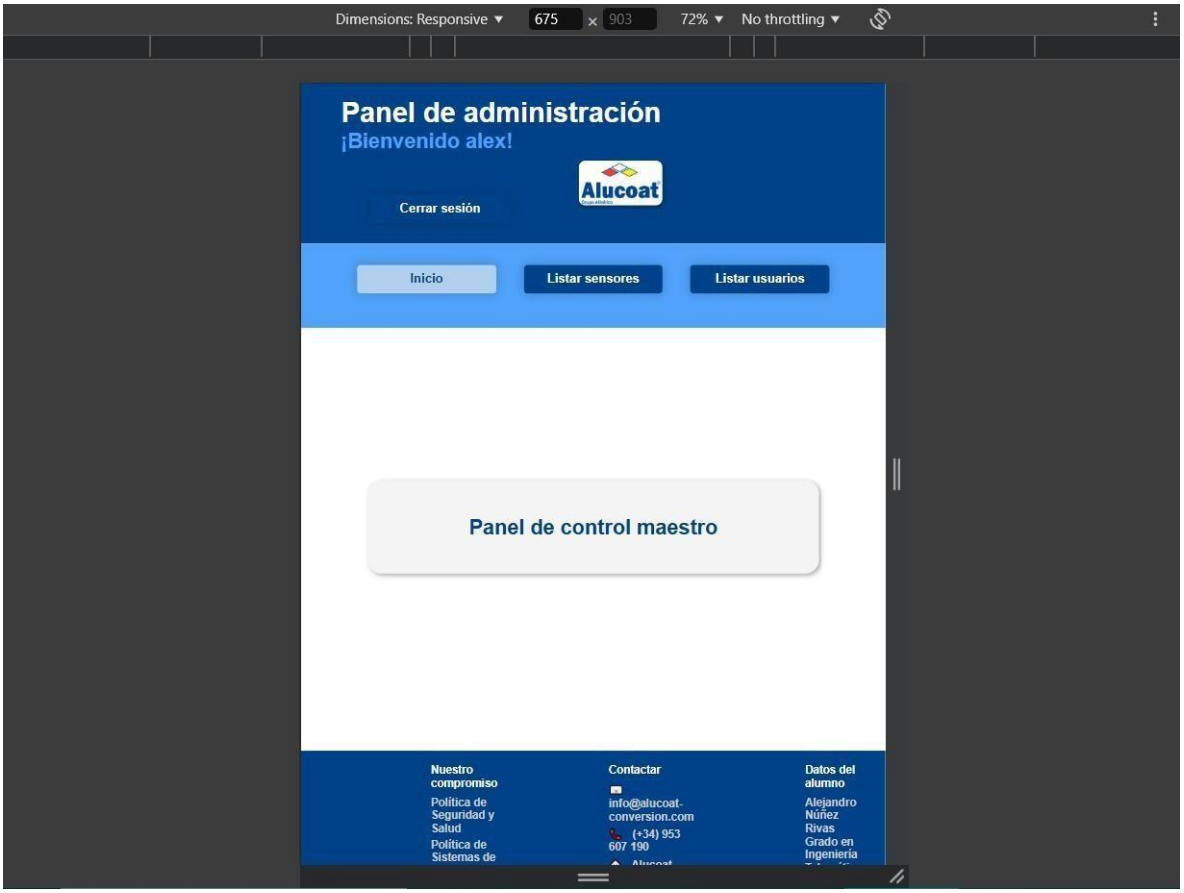

Figura 14: Cambios en el diseño de la vista en base a la resolución

8.2.4. TypeScript

JavaScript es un lenguaje de programación ampliamente extendido debido a sus características de ligereza o compilado, se encuentra comúnmente utilizado para páginas web, aunque también se encuentra implementado para diversidad de entornos fuera de la programación web como puede ser *Node.JS*. Conforme ha ido evolucionando este lenguaje y se fue consolidando en la industria del desarrollo software, empezaron a aparecer derivaciones del mismo para mejorar la experiencia del desarrollador, abordar las limitaciones que supone o para fines de especialización, como pueden ser *TypeScript*, *CoffeScript* o *PureScript*.

TypeScript funciona construido a un nivel mayor sobre *JavaScript*, lo que supone que tenga la misma estructura a la hora de escribir código, pero tenga ciertos añadidos que hacen que resulte más cómodo trabajar con este lenguaje y si fuese necesario compilar previamente a *JavaScript* antes de desplegar el entorno como es en el caso de *Node.JS*. Este lenguaje cuenta con interfaces que sirven para describir la forma de objetos de manera muy intuitiva y rápida, además de contar con tipado por inferencia el cual reconoce por el valor de una variable el tipo del que se trata y lo mantiene.

8.2.5. Cookies

Las *Cookies* son cadenas de texto de pequeño tamaño que se utilizan en el entorno del desarrollo web para que el servidor pueda proceder a verificar la autenticación del usuario sin tener que volver a introducir los parámetros de inicio de sesión, entre otros posibles casos de uso.

En cuanto a la duración de la *cookie* se pueden clasificar de sesión y *cookies* persistentes. Las persistentes se almacenan de manera prolongada y las de sesión solo se trata la información durante el transcurso de tiempo en el que el usuario esté visitando la página web.

En términos del titular de la misma se clasifican en propias o de terceros, siendo estas últimas tratadas por otra entidad ajena.

Además pueden diferir en cuanto a funcionalidad en técnicas, de personalización, de análisis y publicitarias.

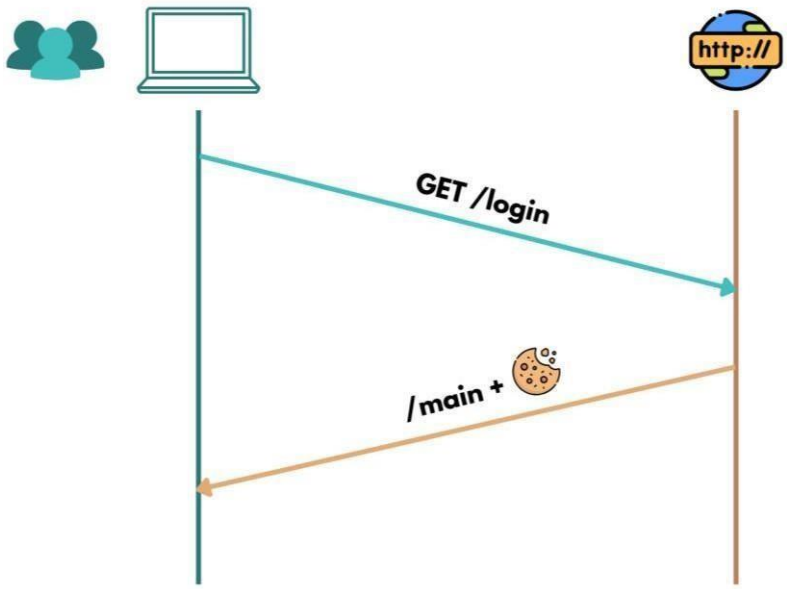

Figura 15: Diagrama de ejemplo de intercambio de cookies

El cliente web del entorno establece una cookie de sesión, propia y técnica con fines de llevar un control de acceso de los usuarios que naveguen por la aplicación web. Se denomina *CurrentUser* y contiene los valores de *idUsuario*, *nombreUsuario*, *Permiso* además de un *array*[13] con las empresas que esté asociado y las gráficas que posea guardadas.

[13] Serie consecutiva ordenada de elementos en programación que pueden ser del mismo tipo o distintos.

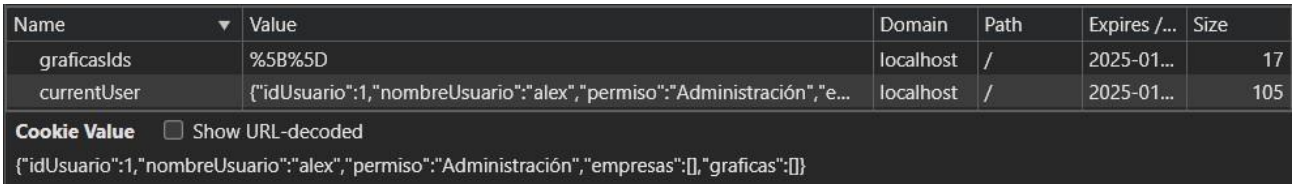

Figura 16: Cookie de la aplicación web

El ciclo de vida de la *cookie* parte de su creación cuando el usuario inicia sesión, con el siguiente código que se detalla a continuación:

```
setCurrentUser(user: Usuario): void {
  const userString = JSON.stringify(user);
  const maxAgeInSeconds = 7 * 24 * 60 * 60; // 7 días
  document.cookie = `currentUser=${userString}; path=/; max-age=${maxAgeInSeconds};`;
}
```

Figura 17: Función que crea la cookie

1. El servidor web obtiene el valor del usuario de la base de datos

2. Pasa a formato *JSON* la información del usuario y lo almacena en la variable *UserString*

3. Se almacena en una variable la cantidad en segundos de tiempo que se va a almacenar la *cookie*

4. Se forma la estructura de la *cookie* con las variables anteriores yse pasa al documento.

Esta cookie se consulta cada vez que se necesite comprobar el usuario que esté navegando la aplicación mediante el *Auth.service* que obtiene la cookie, se convierte en una variable y esta a su vez a formato *JSON*.

```
getCurrentUser(): any {
  const currentUser = this.cookieService.get('currentUser');
  return currentUser ? JSON.parse(currentUser) : null;
}
```

Figura 18: Código que obtiene el usuario

Por último cuando el usuario cierre sesión en la aplicación web se eliminará la cookie hasta que el usuario vuelva a iniciar sesión. Para ello se establece de expiración un valor de una fecha pasada lo que asegurará su eliminación.

```
logout(): void {

  document.cookie = 'currentUser=;expires=Thu, 01 Jan 1970 00:00:00 GMT;path=/;';
  this.router.navigate(['/login']);

}
```

Figura 19: Código eliminación de cookie

8.3. API de recursos local

8.3.1. Node.JS

Node.JS es un entorno de ejecución de *JavaScript open source* que permite tanto crear servidores, como aplicaciones web, como herramientas de línea de comandos entre otros diversos casos de uso. En este caso *Node.JS* permite levantar un servidor que sirve de intermediario entre el *Front-End*[14] y la base de datos para poder visualizar y extraer los datos que se encuentran en la base de datos de manera segura y eficiente.

En el caso de desarrollar el código utilizando el lenguaje *TypeScript*, primero tendrá que desarrollar el código en el mismo y antes de ejecutar el comando que despliega el servidor habrá que ejecutar el comando *TypeScript Compiler* "*tsc*" para que se compile previamente todo el código a *JavaScript* y lo dirija a la carpeta *Distribution "dist"* y así *Node.JS* no tenga problemas a la hora de desplegar el entorno.

Una vez se haya compilado el código al lenguaje *JavaScript*, se tendrá que abrir el *CLI* y escribir el comando "*node dist/index.js*" el cual mediante el comando *node* levantará el servidor en la ruta que haya sido configurada por defecto.

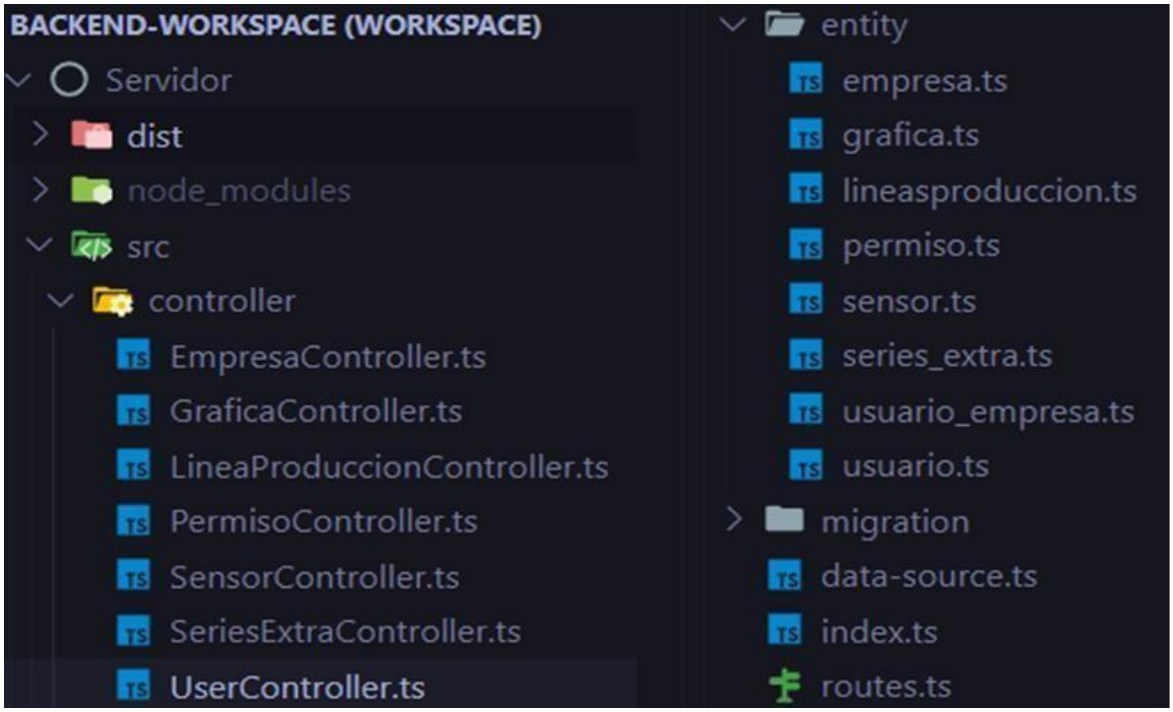

Figura 20: Estructura de carpetas del backend

[14] Parte de una aplicación que interactúa directamente con el usuario final y referencia a la interfaz gráfica.

En primer lugar, en el archivo *data-source.ts* se define la conexión con la base de datos, parametrizando el tipo de servidor que en este caso es *MySQL*, la dirección del servidor que al ser una base de datos en local definiremos *Localhost*, el usuario y contraseña, el nombre de la base de datos, si queremos que se sincronicen los cambios conforme modifiquemos la base de datos, el registro de eventos y consultas *SQL (Structured Query Language)*, las entidades que se hayan definido, las migraciones[15] si hubiere y para especificar las clases de subscriptores de eventos.

```
import { Usuario_Empresa } from './entity/usuario_empresa.js';
import { Series_extra } from './entity/series_extra.js'

export const AppDataSource = new DataSource({
    type: "mysql",
    host: "localhost",
    port: 3306,
    username: "root",
    password: "root",
    database: "Motor_Informes_IoT",
    synchronize: false,
    logging: false,
    entities: [Usuario,Empresa,Sensor,LineaProduccion,Grafica,Perm
    migrations: [],
    subscribers: [],
})
```

Figura 21: Archivo Data-source.js del servidor Node.js

Seguidamente se procederá a definir las entidades tal y como se quiera que se caractericen en la base de datos para que tenga coherencia el sistema. Después de las importaciones necesarias por las relaciones que se definan en este archivo se definirá las variables de la entidad además de las relaciones que posea con otras entidades.

[15] Mecanismo de Node.JS para trasladar toda la estructura de tablas y relaciones a una base de datos.

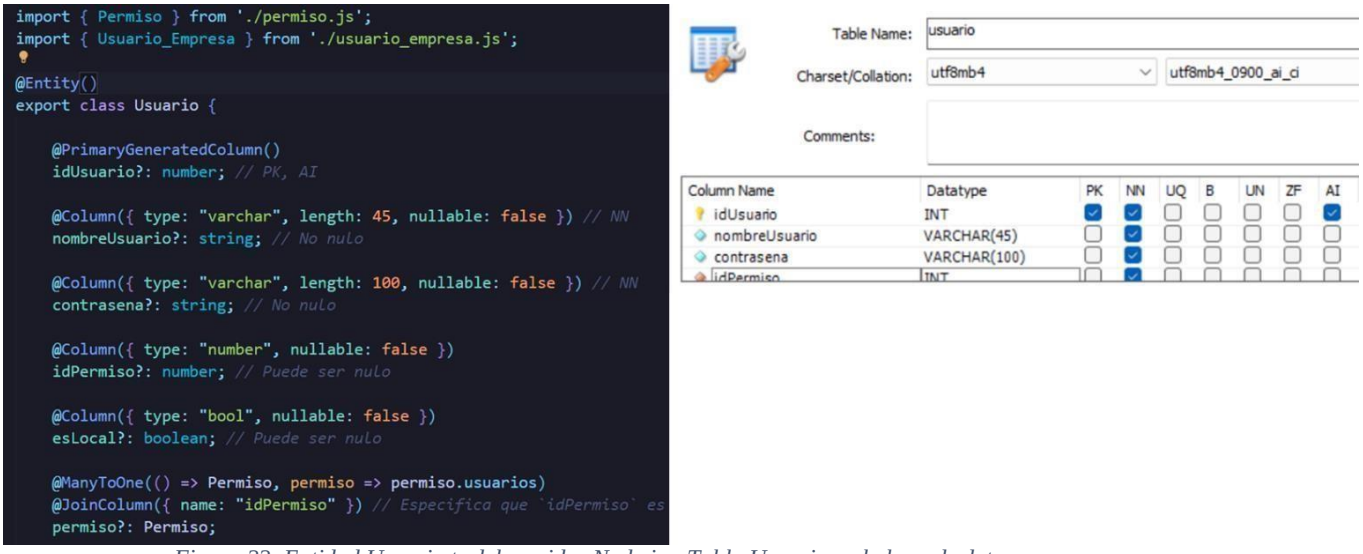

Figura 22: Entidad Usuario.ts del servidor Node.js y Tabla Usuario en la base de datos

En los controladores de cada entidad se desarrollan las funciones que posean cada una de las entidades definiendo el nombre de la función y lo que ocurrirá una vez se llame a esta.

```
import { Usuario_Empresa } from "../entity/usuario_empresa.js";
import { Permiso } from "../entity/permiso.js";


export class UserController {

    private userRepository = AppDataSource.getRepository(Usuario);

    //----------------------- Función para obtener todos los usuarios ----
    async all(request: Request, response: Response, next: NextFunction) {
        try {
            const users = await this.userRepository.find({
                relations: ['usuarioEmpresas', 'usuarioEmpresas.empresa',
            });

            const formattedUsers = users.map(user => ({
                idUsuario: user.idUsuario,
                nombreUsuario: user.nombreUsuario,
                permiso: user.permiso,
                empresas: user.usuarioEmpresas ? user.usuarioEmpresas
```

Figura 23: Archivo UserController.ts del servidor Node.js

El archivo *index.ts* se utiliza para levantar el servidor *express* y se definirá las funciones que relacionan los métodos *HTTP (Hypertext Transfer Protocol)* y rutas de recursos asociadas.

```
AppDataSource.initialize().then(async () => {

    //----------------------- Crear aplicación Express --------
    const app = express();

    app.use(cors({
        origin: 'http://localhost:4200'
    }));

    app.use(express.json());

    // ----------------------- Endpoint para iniciar sesión ---
    app.post('/login', async (req: Request, res: Response) => {
        try {
            await UserController.login(req, res);
        } catch (error) {
            res.status(500).send("Error en la ruta de login");
        }
    });
```

Figura 24: Archivo index.ts del servidor index.js

Y por último en el archivo *routes.ts* se definen las rutas que relacionan el método *HTTP* junto con la ruta del recurso.

```
import { SeriesExtraController } from "./controller/Se

// Definición de rutas de la API
export const Routes = [
    // ----------------------- Rutas de Usuario ------
    {
        method: "get",
        route: "/usuarios",
        controller: UserController,
        action: "all"    // Obtener todos los usuarios
    },
    {
        method: "get",
        route: "/usuarios/:idUsuario",
        controller: UserController,
        action: "one"    // Obtener un usuario por ID
    },
```

Figura 25: Archivo routes.ts

8.3.2. TypeORM

Un *ORM (Object Relational Mapping)* es un método de programar que brinda la facilidad de realizar peticiones de consulta y modificación a una base de datos sin tener que utilizar la sintaxis *SQL*, utilizando para ello programación orientada a objetos. Esta mejora agiliza enormemente el desarrollo de código debido a su rapidez de implementación y a la sencillez que proporciona.

TypeORM es por tanto la integración que une las tecnologías de *ORM* con el lenguaje *TypeScript*.

A la hora de implementar esta tecnología en el proyecto se han integrado una serie de funciones en el archivo *Auth.service* que hacen la función de *TypeORM*. En estas funciones se introduce los parámetros que se quiere mandar al *Back-End*[16], y en formato de peticiones *HTTP* se define el método correspondiente que se quiere realizar.

```
//--------------------------------- Función para guardar seriesExtra ----------
guardarSerieExtra(serieExtraData: any): Observable<any> {
  return this.http.post('http://localhost:3000/seriesextra', serieExtraData);
}


//--------------------------------- Función para eliminar gráficas ------------
eliminarGrafica(idGrafica: number): Observable<any> {
  return this.http.delete(`http://localhost:3000/graficas/${idGrafica}`);
}
```

Figura 26: Código TypeORM

Como se puede observar en la imagen anterior se define una función a la que se le introduce ya sean los datos para dar de alta un objeto o el identificador único de un objeto para eliminarlo, y entonces se devolverá una petición en formato de método HTTP que realizará la consulta que se haya programado.

[16] Parte de una aplicación que gestiona la lógica del procesamiento de solicitudes del cliente y el almacenamiento de los datos.

8.4. API de recursos de Alibérico

La API de Alibérico funciona como la fuente de datos que alimenta el motor de informes para la visualización de estos datos en tiempo real de manera gráfica.

Para ello después de establecer el túnel VPN para poder acceder a la red privada de la empresa de manera segura, se accederá a la ruta correspondiente de la API de monitorización de recursos que extrae datos de la red de sensores que dispone. A continuación se detallan los distintos aspectos a tener en cuenta antes de realizar las consultas.

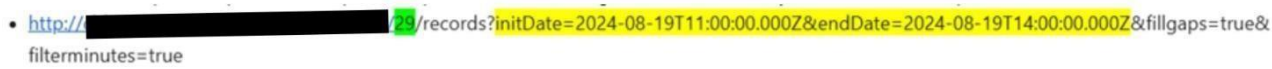

Figura *27: URL del servidor de datos de Alibérico*

1. En primer lugar la ruta */<idSensor>* muestra información general acerca de un sensor en concreto que consultamos por su identificador único, en esta vista se encuentra información como el nombre del sensor, información adicional o el tipo de magnitud que mide entre otras variables.

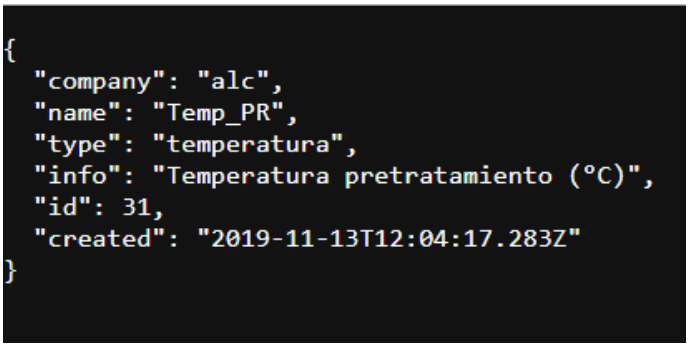

Figura 28: JSON de un sensor

2. Seguidamente encontramos una ruta algo más compleja que delimita el rango de datos que se van a extraer del servidor. Mediante el parámetro *initDate* se establece el principio de la muestra de datos en formato AAAA-MM-DDTHH:MM:SS.000Z y con el parámetro *endDate* el final del rango siguiendo el mismo formato de consulta. Esta vista nos proporcionará en base al tipo de sensor elegido una forma distinta de parametrizar los datos, un ejemplo sería el sensor de conductividad de pretratamiento que muestra un JSON con la fecha por horas junto con un array de 60 valores muestreados una vez por minuto.

```
  {
    "date": "2024-12-20T10:00:00.000Z",
    "data": [87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4,
88, 88, 88, 88, 88, 88, 88, 88, 88, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4, 87.4]
  },
  {
    "date": "2024-12-20T11:00:00.000Z",
    "data": [87.4, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8, 86.8,
85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 85, 88.2, 88.2, 88.2, 88.2, 88.2, 88.2, 88.2, 88.2, 88.2, 88.2,
  },
  {
    "date": "2024-12-20T12:00:00.000Z",
    "data": [85.8, 85.8, 85.8, 85.8, 85.8, 85.8, 87, 87, 87, 87, 87, 87.6, 87.6, 87.6, 87.6, 87.6, 88.2, 88.
88.2, 87.6, 87.6, 87.6, 87.6, 87.6, 87.6, 87.6, 87.6, 87.6, 87.6, 86.4, 86.4, 86.4, 86.4, 86.4, 86.4, 86.4,
  },
```


Figura 29: Valores del sensor Conductividad PR

Para poder realizar las peticiones a la API de Alibérico se ha desarrollado el siguiente código que detalla a continuación el procedimiento para la solicitud de los recursos en base a un identificador del sensor del que se van a extraer los datos, y la fecha de inicio y fin para delimitar la visualización de los datos:

```
cargarDatosAPI(elemento: any, index: number) {
  const idSensorPrincipal = elemento.sensorSeleccionado?.idSensor; 1


  if (!idSensorPrincipal) {
      console.error('Error: idSensorPrincipal no está definido.');
      return;
  }

  const initDatePrincipal = new Date(elemento.fechaInicioGrafica).toISOString();
  const endDatePrincipal = new Date(elemento.fechaFinGrafica).toISOString(); 2

  const urlPrincipal = `http://********/${idSensorPrincipal}/records?initDate=${initDatePrincipal}&endDate=${endDatePrincipal}&fillg
  console.log('URL generada:', urlPrincipal); 3

  // Petición HTTP para el sensor principal
  this.http.get(urlPrincipal).subscribe((response: any) => { 4
```

Figura 30: Código solicitud de datos al servidor de Alibérico

1. En primer lugar se extrae el valor del selector de sensor que el usuario ha seleccionado en el motor de informes y se almacena en una variable *idSensorPrincipal*.

2. En segundo lugar se extraen los valores de los selectores de fecha inicio y fecha fin además de convertir a formato String[17] para poder guardarlos en una variable.

3. En tercer lugar se construye la petición de la misma manera que se ha comentado anteriormente y se introducen los valores de las variables que se han almacenado en esta función.

4. Por último se realiza la petición de HTTP con método GET para solicitar los recursos al servidor.

[17] Tipo de variable en programación que define una cadena de caracteres.

8.5. Base de datos

Una base de datos se refiere a aquel sistema informático que almacena datos estructurados en un formato determinado para poder modificar o consultar esos datos a largo plazo y poder acceder tanto de forma local como de forma remota. Generalmente estos sistemas se encuentran controlados por gestores de bases de datos lo que convierte al conjunto de datos con su gestor en un sistema de bases de datos.

SQL es el lenguaje de programación utilizado por la inmensa mayoría de bases de datos relaciones que permite consultar, modificar y definir los datos.

MySQL es un sistema de bases de datos relacional bajo licencia dual de público general y licencia comercial de la empresa *Oracle* y que como su propio nombre indica utiliza el lenguaje *SQL*. Este sistema se basa en el [modelo cliente – servidor][18] por lo que toma la función de almacenamiento para que el servidor recopile o modifique la información a modo de consultas *SQL*. Además, cuenta con una aplicación con interfaz visual que resulta muy útil para gestionar las entidades y las relaciones entre estas llamadas *MySQL Workbench*.

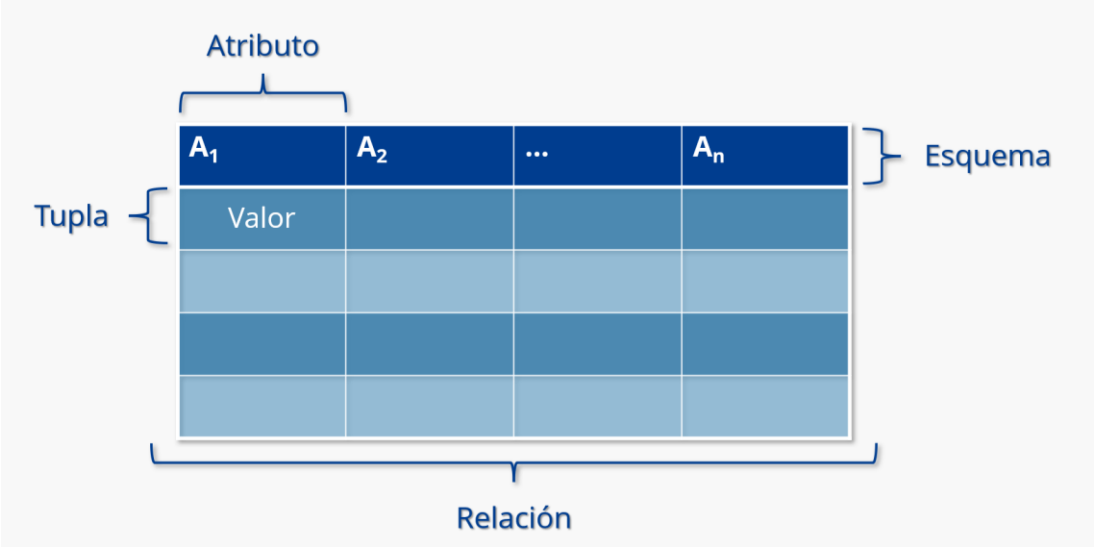

Figura 31: Estructura base de datos relacional

Fuente: https://www.ionos.mx/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos-relacionales/

Una parte crucial en el desarrollo del entorno de la aplicación es el diseño de la base de datos local. Esta base de datos contiene datos como usuarios, permisos y gráficas, además de una estructura jerárquica de los sistemas que se guardan de la empresa Alibérico, como son las distintas sedes del mismo, las líneas de producción o los sensores que contiene.

[18] Arquitectura para el diseño de aplicaciones distribuidas en el cual un dispositivo funciona como cliente realizando las peticiones generales a otro dispositivo servidor que las procesará y devolverá respuestas.

Para tener una representación visual de la arquitectura de la base de datos se ha diseñado un diagrama ER (Entidad - Relación) que relaciona tanto las tablas con su respectiva variable que las une, como del grado de relación que presenta la unión de estas tablas.

A continuación se presenta el diagrama Entidad-Relación de la base de datos además de un análisis detallado de cada una de las tablas que presentan con sus respectivas variables.

8.5.1. Diseño de la base de datos

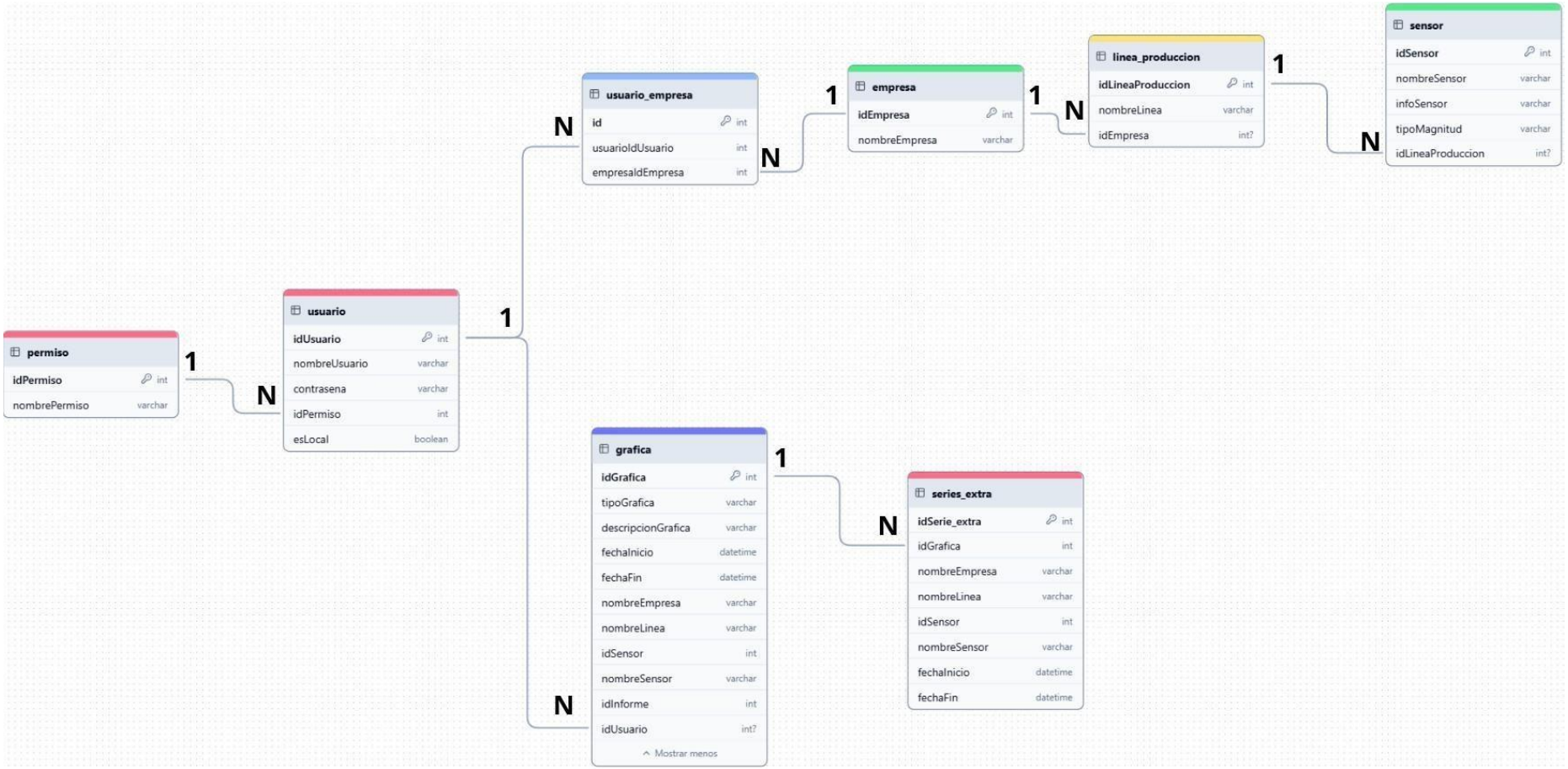

Figura *32: Diagrama Entidad-Relación de la base de datos local*

8.5.2. Usuarios

Tabla que identifica los *Usuarios* que constituyen los posibles empleados de la empresa Alibérico. Esta tabla se relaciona con la tabla permiso mediante la variable *idPermiso*, con la tabla *usuario_empresa* con la variable *idUsuario* y con la tabla grafica también con la variable *idUsuario*.

Atributo	Definición	Clave foránea
idUsuario	Identificador único del usuario	-
nombreUsuario	Nombre de usuario	-
contraseña	Contraseña del usuario	-
esLocal	Booleano que indica si el usuario es local o parte del directorio activo	-
idPermiso	Identificador del permiso que tiene el usuario	**Permiso** (idPermiso)

Tabla 1: Entidad Usuarios

8.5.3. Permiso

Tabla que contiene los posibles valores de permisos que son asociados a los usuarios. Estos posibles valores son *Configuración de gráficas*, *Administración* o *Total*. Esta tabla se encuentra relacionada con la tabla *Usuario* con la variable *idPermiso*.

Atributo	Definición	Clave foránea
idPermiso	Identificador único del permiso	-
nombrePermiso	Nombre del permiso	-

Tabla 2: Entidad Permiso

8.5.4. Grafica

Tabla que contiene los parámetros de las gráficas que se generen y guarden en la aplicación web. Esta tabla se encuentra relacionada con la tabla de *Usuario* mediante la variable *idUsuario* y con la tabla *series_extra* mediante la variable *idGrafica.*

Atributo	Definición	Clave foránea
idGrafica	Identificador único de la gráfica	-
idUsuario	Identificador único del usuario que ha generado la gráfica	**Usuario** (idUsuario)

tipoGrafica	Determina el tipo de gráfica con 3 valores posibles “lineal”, “barras” o “circular”	-
descripcionGrafica	Detalles opcionales de la gráfica o del contexto en el que se ha generado	-
nombreEmpresa	Nombre de la empresa que contiene el sensor que ha generado los datos	-
nombreLinea	Nombre de la línea de producción que contiene al sensor	-
idSensor	Identificador único del sensor que ha generado los datos de la gráfica	-
nombreSensor	Nombre del sensor que ha generado los datos	-
fechaInicio	Fecha de inicio para delimitar los datos recopilados	-
fechaFin	Fecha de fin para delimitar los datos recopilados	-

Tabla 3: Entidad Grafica

8.5.5. Series_extra

Tabla que contiene los parámetros de las series extras si es que existiesen respecto de una gráfica. Esta tabla se encuentra relacionada con la tabla *grafica* mediante la variable *idGrafica*.

Atributo	Definición	Clave foránea
idSerieExtra	Identificador único de la serie extra respecto de una gráfica	-
idGrafica	Identificador único de la gráfica	**Gráfica** (idGráfica)
nombreEmpresa	Nombre de la empresa que contiene el sensor que ha generado los datos	-
nombreLinea	Nombre de la línea de producción que contiene al sensor	-
idSensor	Identificador único del sensor que ha generado los datos de la gráfica	-

nombreSensor	Nombre del sensor que ha generado los datos	-
fechaInicio	Fecha de inicio para delimitar los datos recopilados	-
fechaFin	Fecha de fin para delimitar los datos recopilados	-

Tabla 4: Entidad Series_extra

8.5.6. Usuario_empresa

Esta tabla sirve de intermediario entre las tablas *Usuario* y *Empresa* para que se pueda realizar la relación N a N que contienen estas tablas. Esta tabla se relaciona con la tabla *Usuario* mediante la variable *usuarioIdUsuario* y con la tabla *Empresa* mediante la variable *EmpresaIdEmpresa*.

Atributo	Definición	Clave foránea
id	Identificador único de la relación entre usuario y empresa	-
usuarioIdUsuario	Identificador único del usuario que tiene relación con la empresa	**Usuario** (idUsuario)
EmpresaIdEmpresa	Identificador único de la empresa que tiene relación con el usuario	**Empresa** (idEmpresa)

Tabla 5: Entidad Usuario_empresa

8.5.7. Empresa

Tabla que contiene las diferentes empresas del Grupo Alibérico. Se encuentra relacionada con la tabla *usuario_empresa* mediante la variable *idEmpresa* y con la tabla *linea_produccion* mediante también la misma variable.

Atributo	Definición	Clave foránea
idEmpresa	Identificador único de la empresa	-
nombreEmpresa	Nombre representativo de la sucursal de la empresa	-

Tabla 6: Entidad Empresa

8.5.8. Linea_produccion

Tabla que contiene las distintas líneas de producción que se encuentran en una sucursal determinada. Se encuentra relacionada con la tabla *Empresa* mediante la variable *idEmpresa* y con la tabla *Sensor* mediante la variable *idLineaProduccion*.

Atributo	Definición	Clave foránea
idLineaProduccion	Identificador único de la línea de producción	-
nombreLinea	Nombre de la línea de producción	-
idEmpresa	Identificador único de la empresa que contiene a las línea de producción	**Empresa** (idEmpresa)

Tabla 7: Entidad Linea_produccion

8.5.9. Sensor

Tabla que contiene los sensores que se encuentran en una línea de producción en concreto. Esta tabla se encuentra relacionada con la tabla *Linea_produccion* mediante la variable *idLineaProduccion*.

Atributo	Definición	Clave foránea
idSensor	Identificador único del sensor	-
nombreSensor	Nombre del sensor	-
infoSensor	Información adicional	-
tipoMagnitud	Tipo de magnitud que mide el sensor	-
idLineaProduccion	Identificador único de la línea de producción	**Linea_Produccion** (idLineaProduccion)

Tabla 8: Entidad Sensor

8.6. Control de versiones

Git es un sistema de control de versiones distribuido utilizado para la gestión de proyectos de desarrollo software de forma eficaz. El marco de trabajo de este sistema consiste en la revisión de cambios respecto a la última versión subida al servidor y, a continuación, la sincronización de esos cambios con el entorno local para generar una nueva versión del código. Este sistema dispone de un método para crear ramas de un mismo punto de inicio para así poder desarrollar el software aislando los cambios de manera temporal hasta que se anexione esa rama principal mediante una petición de *PR (Pull Request)*, o, por el contrario, se descarte.

Figura 33: Diagrama de ramas de Git.

Fuente: https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

Con fines de llevar el control de versiones se ha desarrollado el entorno utilizando este sistema. Desde *Azure DevOps* tenemos una conexión directa con el *IDE (Integrated Development Environment)* en este caso con *Visual Studio Code*, para utilizar *Git*.

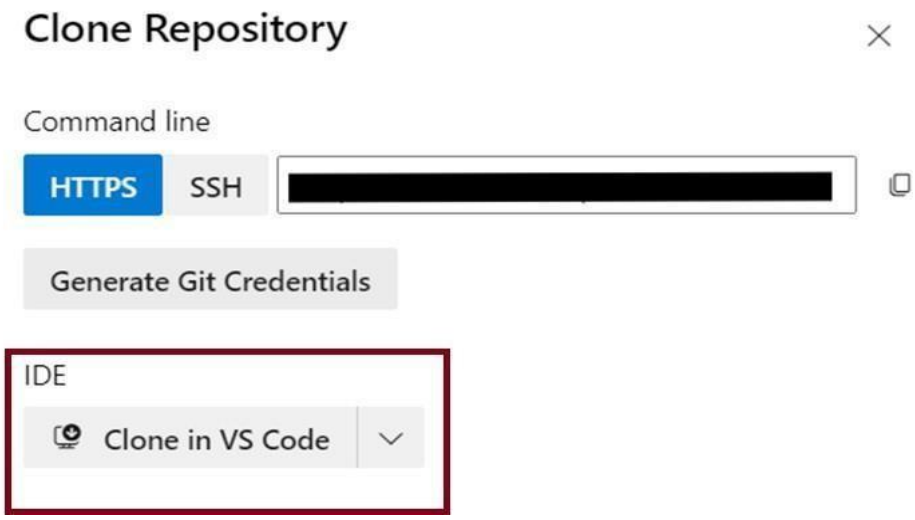

Figura 34: Integración nativa de Azure DevOps con VS Code

Esta conexión nos proporcionará un interfaz muy intuitivo en el que podremos tanto tener un histórico de los cambios que hemos ido realizando en el tiempo, como los cambios que aún están pendientes por ser subidos a la plataforma y, además, poder realizar comandos *Git* gestionar estos cambios.

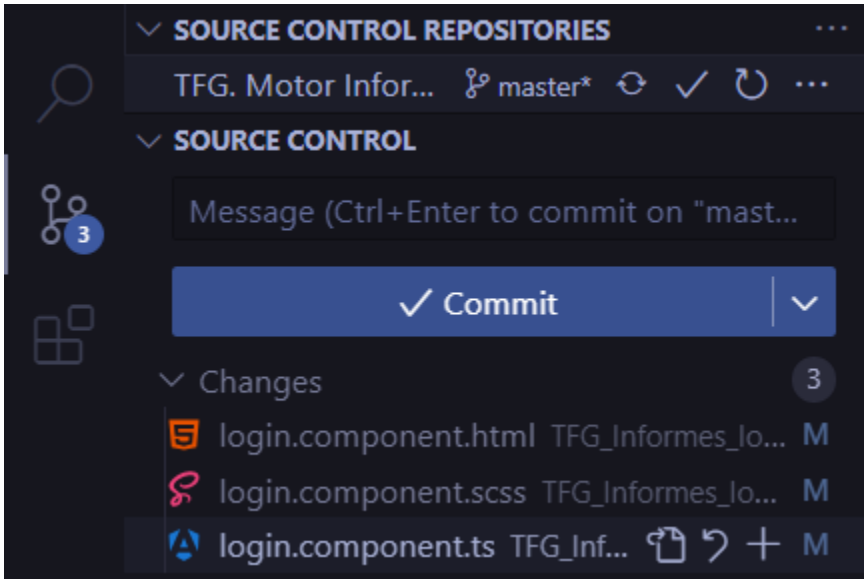

Figura 35: Vista superior de la zona de control de versiones de Visual Studio Code

En este panel superior se podrán observar datos como la rama en la que se encuentre el desarrollador, los archivos en los que se han producido cambios desde la última versión que se encuentra subida a la plataforma, además de disponer de las herramientas para realizar comandos *Git*, o gestionar los cambios que se hayan producido en los archivos como descartándolos en caso de que se quiera volver a su versión anterior.

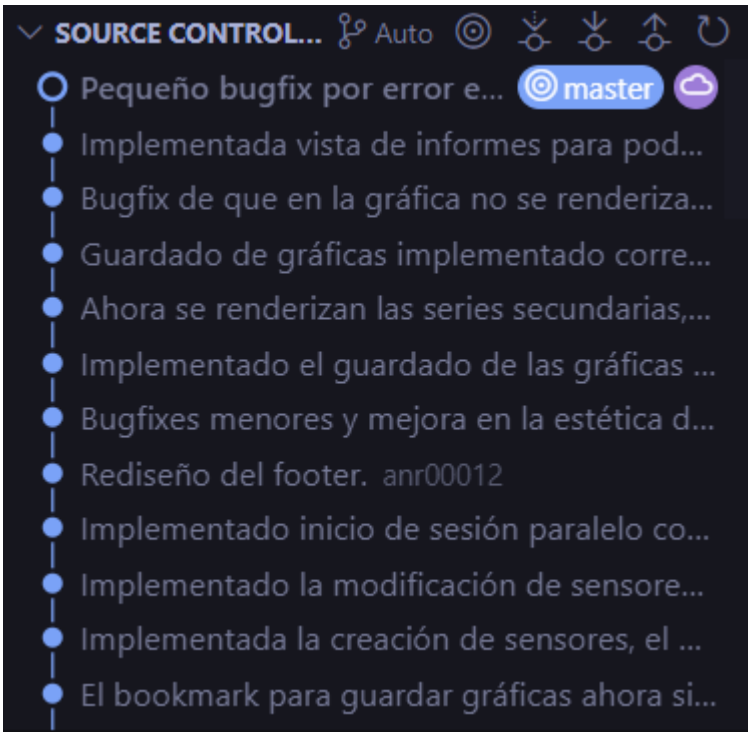

Figura 36: Vista inferior del panel de control de versiones de Visual Studio Code

En el panel inferior se encuentra una gráfica de control de origen en la que se puede visualizar un historial de las distintas actualizaciones que ha tenido el código en su totalidad a lo largo de todo el desarrollo. Además, de unos botones que permiten el cambio de rama, o comandos de *Git* varios para descargar el código del repositorio.

8.7. Seguridad

8.7.1. Copias de seguridad

La nube se refiere a un modelo tecnológico con fines de almacenamiento, procesamiento y software accesible mediante *Internet* y propietario. En este caso se ha utilizado la plataforma de *Azure DevOps* propietaria de *Microsoft* en el entorno de la organización de *AlibericoDevOps*.

Estos modelos de servicio en la nube permiten a las organizaciones acceder a los recursos que alojen en esta plataforma de manera remota desde cualquier lugar ya que es accesible mediante *Internet* y sin la necesidad del mantenimiento o gestión de la infraestructura física. En nuestro caso se está recurriendo a un modelo de una infraestructura bajo servicio o *IaaS (Infrastructure as a Service)* el cual nos proporciona unos recursos básicos de computación como es el almacenamiento del código de la aplicación web y servidor de recursos.

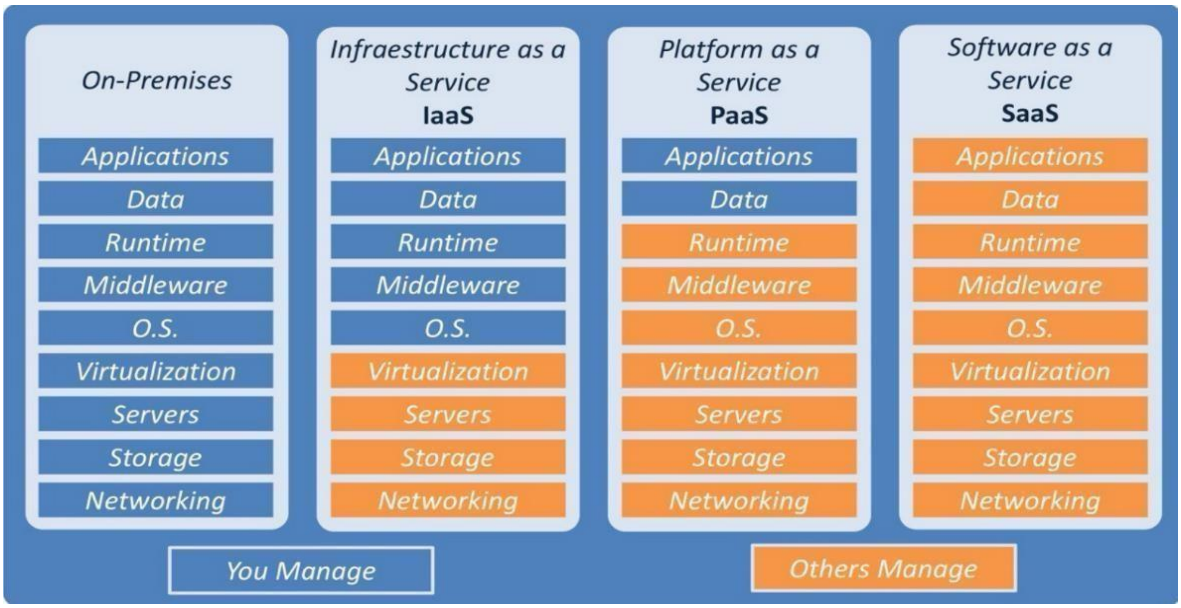

Figura 37: Niveles de gestión

Una vez hayamos iniciado sesión con *Microsoft* la plataforma analizará si poseemos los permisos otorgados por la organización en cuestión para visualizar el contenido que dispone, en este caso en el entorno de *AlibericoDevOps* nos encontramos dos repositorios independientes, "*TFG. Motor de Informes IoT FrontEnd*" y "*TFG. Motor de Informes IoT BackEnd*" en donde se aloja el código del cliente web de la aplicación y el servidor de recursos respectivamente. Si accedemos a alguno de ellos individualmente podremos observar los miembros pertenecientes al servicio, estadísticas del proyecto y el control de las versiones de manera cronológica y detallada para su análisis en caso de que fuese necesario.

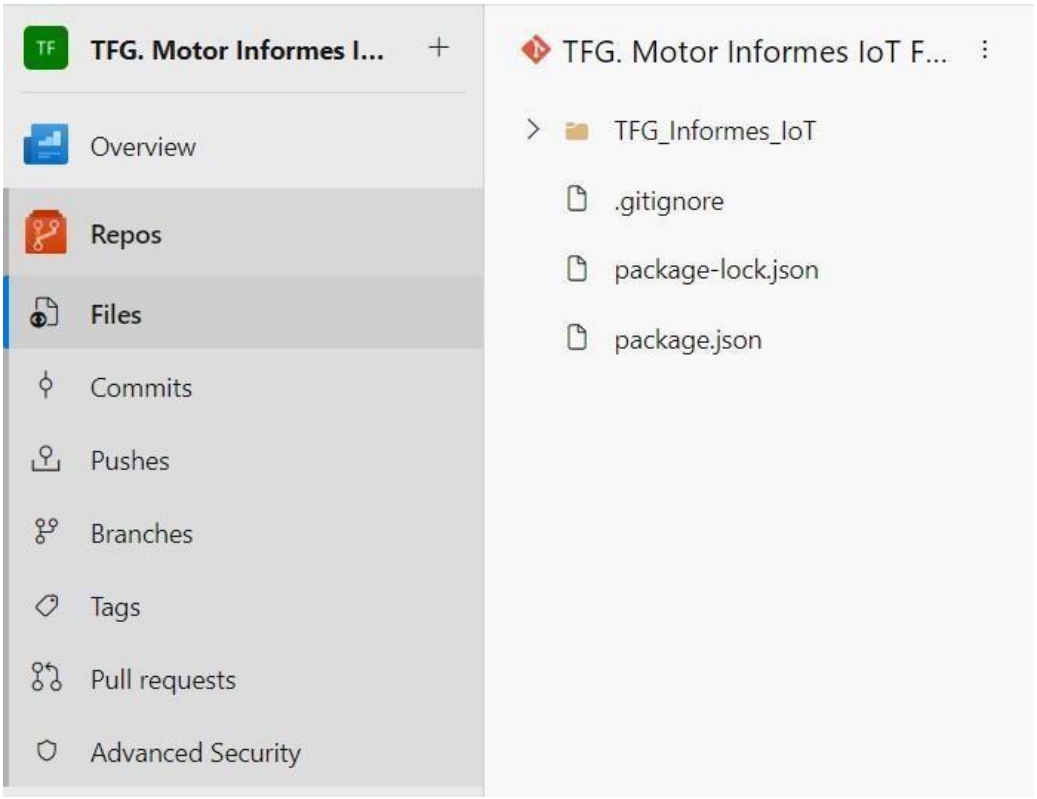

Figura 38: Panel de archivos de la nube Azure DevOps

8.7.2. HTTPS

El protocolo *HTTP* es un protocolo a nivel de aplicación según la torre de protocolos del modelo *OSI (Open Systems Interconnection)* el cual nos permite mandar peticiones de consulta de datos o recursos a una página web. Existen multitud de métodos para ello, pero los más conocidos son los métodos *GET*, para obtener recursos de un dominio, *POST* que nos permite publicar algún tipo de información, *PUT* para actualizar un recurso ya existente, o *DELETE* para eliminar algún elemento.

Para subsanar el fallo de seguridad que supone transmitir información en texto plano sin encriptar, como ocurre con el protocolo *HTTP*, se desarrolló el protocolo *SSL (Secure Sockets Layer)*. Este protocolo proporciona una capa de encriptación necesaria para garantizar comunicaciones privadas y seguras mediante un proceso de autenticación. Dicho proceso incluye un intercambio de claves públicas y privadas, respaldado y validado por una autoridad de certificación confiable. Este protocolo ha ido evolucionando hasta convertirse en lo que hoy en día se denomina como *TLS (Transport Secure Layer)* dejando obsoleto a *SSL*.

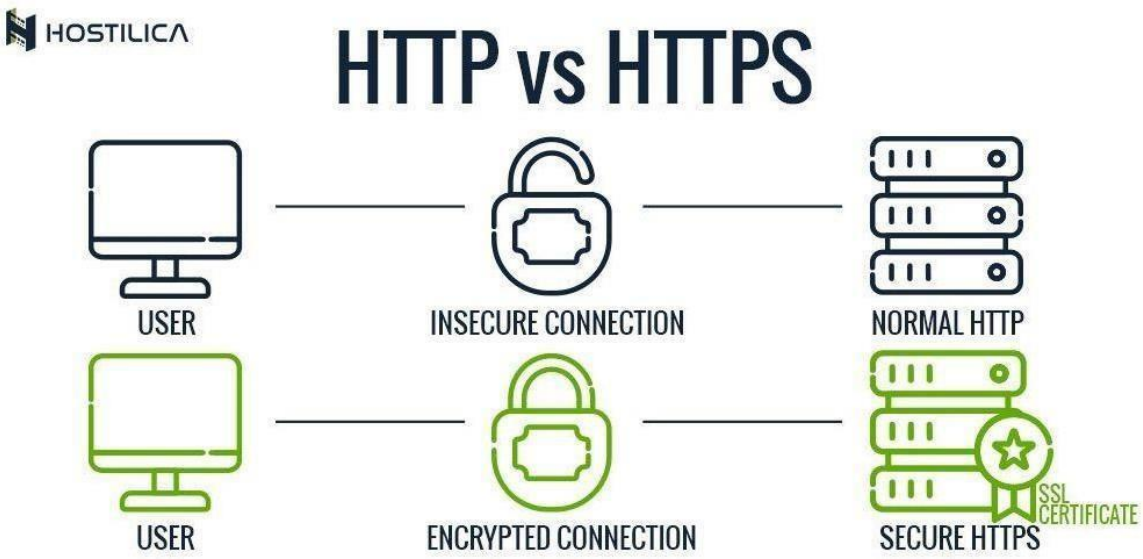

Figura *39: Diferencias entre HTTP y HTTPS*

Fuente: https://hostilica.com/blog/http-vs-https/

Por tanto, al uso conjunto del protocolo *HTTP* para consulta y modificación de recursos y *TLS* para proporcionar un sistema de autenticación y cifrado para establecer comunicaciones seguras se le denomina como *HTTPS*.

Por defecto el entorno de desarrollo despliega la aplicación web sobre *HTTP*, aunque se puede modificar el comando para levantar el servicio para que utilice un certificado expedida por una [autoridad de certificación][19] local que haya expedido esos certificados auto firmados por el desarrollador mediante la herramienta *mkcert*.

Una vez creado el certificado auto firmado y la llave se desplegará el servicio web mediante el comando "*<ng serve --ssl --ssl-cert "cert.crt" --ssl-key "cert.key">*" y con ello se levantará la aplicación web utilizando *HTTPS* sin el aviso de sitio web inseguro ya que se dispone de la entidad de certificación instalada en el entorno de desarrollo local.

[19] Entidad fiable encargada de la emisión de certificados digitales para seguridad web.

```
PS C:\Users\anune.ALEX.000\Desktop\Universidad\4º de Carrera\TFG\AzureDevOps\TFG. Mo
d\TFG_Informes_IoT> ng serve --ssl --ssl-cert "cert.crt" --ssl-key "cert.key"
Initial chunk files | Names         |  Raw size
main.js             | main          | 474.62 kB |
polyfills.js        | polyfills     |  90.20 kB |
styles.css          | styles        | 200 bytes |

                    | Initial total | 565.02 kB

Application bundle generation complete. [5.451 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  → Local:   https://localhost:4200/
  → press h + enter to show help
```

Figura 40: Despliegue del servidor web sobre HTTPS en el CLI

8.7.3. Acceso mediante Directorio Activo

Un *AD (Active Directory)* es el conjunto de un servidor que realiza una función de base de datos y el servicio que desempeña para interconectar usuarios con los recursos de la red que soliciten. Supone una gran medida de ciberseguridad para la autenticación y acceso a recursos de los usuarios de una red privada.

Figura 41: Diagrama interconexión Directorio Activo

Fuente: https://es.linkedin.com/pulse/active-directory-ad-perfectnumbers

El acceso a la aplicación web del entorno mediante directorio activo fue uno de los objetivos que se marcaron al inicio del desarrollo del proyecto y si bien se ha realizado la implementación, no es posible su uso debido a que no se ha conseguido acceso a un usuario perteneciente al servidor *LDAP*[20] *(Lightweight Directory Access Protocol, protocolo ligero de acceso a directorios)* de la empresa, ni el permiso necesario para la conexión con el mismo.

Este acceso se ha diseñado de manera que se realice simultáneamente con el inicio de sesión estándar de un usuario que se encuentre registrado en la base de datos local, con la

[20] Servidor que gestiona el acceso de usuarios a los recursos de un directorio.

diferencia que si el usuario dispone de un nombre que contenga el dominio[21] *@aliberico.com* entonces se le realizará una autenticación con la base de datos remota del directorio activo.

A continuación se detalla el código y los pasos necesarios para la autenticación de un usuario con el directorio activo:

1. En primer lugar la función analizará si el nombre que ha proporcionado el usuario contiene el dominio de Alibérico, y en caso de tenerlo se derivarán sus parámetros en la función del *Auth.service* correspondiente.

```
login(): void {
  if (this.username.includes('@aliberico.com')) {
    // Autenticación vía Active Directory
    this.authService.loginWithAD(this.username, this.password).subscribe({
```

Figura 42: Código de inicio de sesión con directorio activo

2. En el archivo *Auth.service* se dispondrán los parámetros introducidos por el usuario en el cuerpo de la petición *HTTP* y entonces se realizará una petición *POST* para solicitar este inicio de sesión.

```
//---------------------------------- Función iniciar sesion Active Directory ----------------------------------//
loginWithAD(username: string, password: string): Observable<any> {
  const body = { username, password };
  return this.http.post('http://localhost:3000/loginAD', body);
}
```

Figura 43: Código de solicitud de autenticación con directorio activo

3. Por último la petición llega al Back-End del sistema que se encargará de reenviar la petición al servidor LDAP de Alibérico y manejar el control de errores.

[21] Grupo de usuarios que comparten un mismo espacio de nombres común y son gestionados de manera centralizada.

```
// ----------------------- Función para iniciar sesión con Active Directory -----------------------//
static async loginAD(req: Request, res: Response): Promise<void> {
    const { username, password } = req.body;

    if (!username || !password) {
        res.status(400).json({ message: "Faltan credenciales" });
        return;
    }

    const client = ldap.createClient({
        url: 'ldap://example-aliberico.com', // Dirección del servidor LDAP
    });

    const DN = `cn=${username},dc=example,dc=com`; // Ajusta el Distinguished Name

    client.bind(DN, password, (err) => {
        if (err) {
            console.error('Error al autenticar con LDAP:', err);
            res.status(401).json({ message: 'Usuario o contraseña incorrectos en Active Directory' });
        } else {
            console.log('Autenticación exitosa en LDAP');
            res.status(200).json({
                username,
            });
        }
    });
}
```

Figura 44: Código para iniciar sesión en el Back-End

8.7.4. Gestión de perfiles de usuario

Limitar la visualización de contenido sensible es una práctica imprescindible a la hora de desarrollar una aplicación web, por lo que para delimitar el contenido que puede visualizar cada persona dentro de la empresa de Alibérico se ha diseñado un esquema de roles que diferencian claramente los permisos con los que cuenta cada usuario limitando las funciones a las que tiene acceso en base a este permiso.

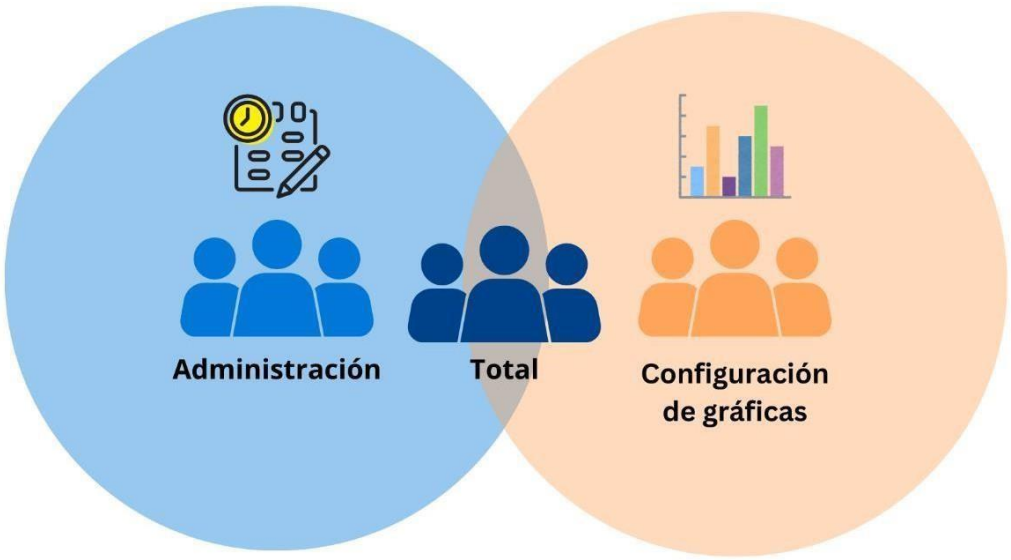

Figura 45: Diagrama de tipos de permisos en el servidor web

Cada uno de los usuarios dispone de una variable llamada *Permiso* en las variables de su clase *Usuario* que podrá exclusivamente tomar los valores de *Configuración de gráficas, Administración* o *Total* que definirá que funciones estarán limitadas en la aplicación web. Estos roles se caracterizan de la siguiente manera:

El usuario con permiso *Configurador de gráficas* se caracterizará por solo tener acceso a las vistas del *Motor de gráficas* y *Gráficas guardadas* desde el panel de navegación y por la exclusiva función de generar gráficas, informes y el almacenamiento de los mismos.

El permiso de *Administración* proporcionará acceso a las vistas de *Inicio, Listar sensores y Listar usuarios* desde el panel de navegación lo que se proporcionarán las funciones de gestión de usuarios y sensores.

Y, por último, el permiso *Total* incluye todas las funciones y privilegios que posee los permisos de *Configurador de Gráficas y Administración.*

Angular posee un sistema de protección de rutas llamado *Guards*, en este caso el *Auth.Guard* se encuentra aplicado a todas y cada una de las rutas de la aplicación (menos el panel de inicio de sesión) verificando que el usuario que se encuentre navegando en este disponga del permiso necesario para hacerlo. Para aplicar este *Guard* hay que implementar el código "*canActivate: [AuthGuard]*" al final de cada ruta en el archivo *app.routes.ts.*

```
export const routes: Routes = [
  { path: 'admin/listarSensores', component: ListarSensoresComponent, canActivate: [AuthGuard]  },
  { path: 'admin/listarSensores/Sensor/:idSensor', component: GestionSensoresComponent, canActivate: [AuthGuard]
  { path: 'admin/gestionSensores', component: GestionSensoresComponent, canActivate: [AuthGuard]  },
  { path: 'admin/gestionPermisos', component: GestionPermisosComponent, canActivate: [AuthGuard]  },
  { path: 'admin/crearUsuarios', component: CrearUsuariosComponent, canActivate: [AuthGuard]  },
  { path: 'admin/crearSensores', component: CrearSensoresComponent, canActivate: [AuthGuard]  },
  { path: 'admin/modificarSensores/:idSensor', component: ModificarSensoresComponent, canActivate: [AuthGuard] }
  { path: 'admin/modificarUsuarios/:idUsuario', component: ModificarUsuariosComponent, canActivate: [AuthGuard]
  { path: 'admin/eliminarUsuarios', component: EliminarUsuariosComponent, canActivate: [AuthGuard] },
  { path: 'admin/listarUsuarios', component: ListarUsuariosComponent, canActivate: [AuthGuard] },
  { path: 'admin/listarUsuarios/Usuario/:idUsuario', component: GestionUsuarioComponent, canActivate: [AuthGuard
  { path: 'admin', component: AdminComponent, canActivate: [AuthGuard] },
  { path: 'main/graficasGuardadas', component: Graficas (alias) class InformesComponent                    [AuthGuard] },
  { path: 'main/motorGraficas', component: MainComponen import InformesComponent
  { path: 'main/informes/:idInformeGlobal', component: InformesComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: '**', redirectTo: 'login' } // Redirigir a Login por defecto
];
```

Figura 46: Archivo app.routes.ts

```
const user = this.authService.getCurrentUser();

if (!user) {
  this.router.navigate(['/login']);
  return false;
}
```

Figura 47: Archivo Auth.guard.ts sobre el caso de permiso sin usuario

El *Auth.guard* primero intenta obtener el usuario que se encontrase en la sesión actual mediante el *authService*, y entonces procede a comprobar si existe ese usuario. En el caso de que encuentre un valor nulo procedería a redirigir a la ruta por defecto de inicio de sesión */login*.

```
if (user.permiso === 'Configuración de gráficas') {
  if (requestedUrl.startsWith('main')) {
    return true;
  } else {
    this.router.navigate(['/main/motorGraficas']);
    return false;
  }
}
```

Figura 48: Archivo Auth.guard.ts sobre el caso de permiso Configuración de gráficas

En el caso de que el *Auth.Guard* encuentre un usuario, y este usuario tenga el campo de la variable *Permiso* con el valor *Configuración de gráficas* entonces tiene permiso para acceder a las rutas que empiecen por */main*, en caso de solicitar una ruta que no empiece por esta cadena de caracteres será redirigido a la ruta por defecto de este permiso que es */main/motorGraficas*.

```
if (user.permiso === 'Administración') {
  if (requestedUrl.startsWith('admin')) {
    return true;
  } else {
    this.router.navigate(['/admin']);
    return false;
  }
}
```

Figura 49: Archivo Auth.guard.ts sobre el caso de permiso Administración

Si el usuario dispone del permiso llamado *Administración* y solicita una vista que empiece por la cadena *admin* se le permitiría el recurso, en caso contrario sería redirigido a su ruta por defecto que es */admin*.

```
if (user.permiso === 'total') {
  return true;
}
```

Figura 50: Archivo Auth.guard.ts sobre el caso de permiso total

Y, por último, si el *Auth.Guard* encuentre un usuario, y este usuario tenga el campo de la variable *Permiso* con el valor *total* entonces tiene permiso para acceder a cualquier ruta que desee.

8.7.5. VPN

Debido a que el servidor de recursos de la empresa Alibérico no se encuentra accesible mediante *Internet*, sino que se encuentra desplegado en la red local se requiere de un túnel *VPN (Virtual Private Network)* para acceder a los datos de los sensores. Una *VPN* es una conexión de red privada que proporciona un canal de comunicación seguro enmascarando las direcciones *IP*[22] *(Internet Protocol)* y cifrando los datos de la comunicación.

[22] Direcciones que identifican de manera única a los dispositivos de una red.

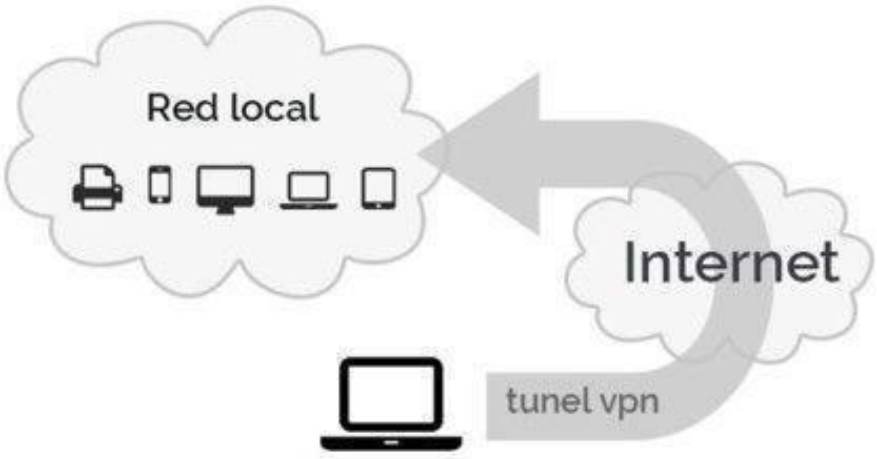

Figura 51: Diagrama sobre el funcionamiento de VPN

Fuente: https://www.xataka.com/basics/que-es-una-conexion-vpn-para-que-sirve-y-que-ventajas-tiene

Para establecer la conexión *VPN* con la empresa Alibérico se tendría que añadir un nuevo túnel al sistema operativo proporcionando los parámetros del proveedor de *VPN*, la dirección del servidor, y si precisa de algún tipo de información de sesión como un nombre de usuario y contraseña, o un certificado.

Figura 52: Creación de una conexión VPN en Windows 11

Una vez autenticado el servidor de *VPN* se solicitará una clave *2FA*[23] *(2 Factor Authentication)* que se podrá rellenar con una aplicación móvil externa y finalmente se obtendrá acceso a la red privada de Alibérico mediante un túnel seguro *VPN* pudiendo así realizar peticiones al servidor de datos.

[23] Autenticación con 2 pasos que se verifican en diferentes medios para verificar la identidad de un usuario.

8.7.6. Hash de contraseñas

Una función *Hash* se trata de una función matemática unidireccional, es decir, a partir de un valor determinado se aplica de forma muy sencilla proporcionando su valor *hash* de manera instantánea, pero en el sentido contrario resulta imposible encontrar de manera matemática el valor concreto respecto a su valor *hash*. Esta propiedad ha llevado a este tipo de funciones a convertirse en sistemas de seguridad muy útiles como puede ser la firma digital o la verificación de la autenticidad de archivos.

Figura 53: Diagrama de funcionamiento sobre la función Hash criptográfica

Fuente: https://www.criptonoticias.com/criptopedia/que-es-funcion-hash-como-influye-bitcoin/

En el entorno se ha aplicado el sistema de *Hash* mediante la librería *Bcrypt* en el servidor local *Node.JS*. Esta librería recurre al algoritmo *Blowfish* para realizar su función *hash*. Este algoritmo utiliza un [cifrado simétrico][24] que opera sobre bloques de 64 bits con claves de longitud variable que pueden oscilar entre 32 y 448 bits.

Para implementar este sistema de seguridad se ha desarrollado el siguiente código para encriptar las contraseñas que se soliciten del servidor web para iniciar sesión o para crear un nuevo usuario:

[24] Método de encriptación en el cual se utiliza la misma clave para cifrar como para descifrar un recurso.

```
// Encriptar la contraseña
const saltRounds = 10;
const hashedPassword = await bcrypt.hash(contrasena, saltRounds);
```

Figura 54: Código para encriptar contraseña

Para encriptar la contraseña que se ha introducido cuando se crea un usuario, primero se debe establecer el valor de rondas de *salt*[25] para indicar cuantas veces se iterará la función hash internamente y entonces se realizará la función.

```
// Comparar la contraseña proporcionada con la almacenada
const passwordMatch = await bcrypt.compare(password, usuario.contrasena);
if (!passwordMatch) {
    return res.status(401).json({ message: 'Contraseña incorrecta' });
}
```

Figura 55: Código para verificar contraseña al iniciar sesión

Cuando un usuario inicie sesión en la aplicación web el valor que introduce de contraseña se mandará en texto plano hacia el servidor que realizará la función hash y entonces comparará con el valor hash que se encuentre en la base de datos, en caso de discernir enviará un error al cliente indiciando que la contraseña no es correcta.

[25] Cadena de caracteres aleatoria que se añade al principio o fin de una contraseña para prevenir algunos ataques de ciberseguridad.

9. Problemas en el desarrollo

Durante el desarrollo del proyecto, se identificaron diversos problemas que han representado un desafío significativo a nivel técnico y han podido tener impacto en las características finales de la solución desarrollada.

9.1. Escasez de información actualizada de Angular

Al elegir la versión 18 del *Framework Angular* cuando solo llevaba unos pocos meses de su salida también repercutió el desarrollo de la aplicación web debido a que la documentación oficial se encontraba actualizada pero no abarcaba todos los aspectos que se requerían en el desarrollo y al consultar documentación de versiones anteriores surgió el inconveniente que la forma de desarrollo era diferente a la actual por lo que tampoco resultaba de utilidad.

9.2. Problema con extensiones de archivos al compilar TypeScript a JavaScript

El último problema que ha conllevado un retraso a la hora del desarrollo del servidor *Node.JS* fue un error en las importaciones de entidades de todos los archivos. Este error derivó en una semana completa de investigación de soluciones para el mismo que implicó un notorio retraso y esfuerzo de recursos humanos.

Figura 56: Importaciones de archivos

Cuando se importan otras entidades en un controlador se debe importar el nombre con el que han sido definidos además de la ruta en la que se encuentran. El problema está en que no debería hacer falta incluir la extensión de los archivos porque además de que si la incluimos nos genera un error el *IDE* y no nos deja compilar a *JavaScript* pero si resulta de que no se le incluyen las extensiones el CLI genera un error sobre que no encuentra las importaciones.

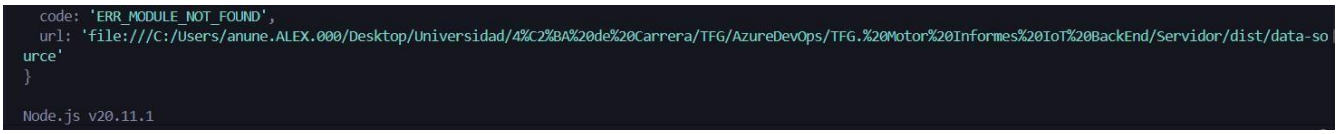

Figura 57: Error Módulo no encontrado

Para subsanar este fallo se ha tenido que incluir la extensión *.js* característica de *JavaScript* en las importaciones del resto de los archivos aunque sean *TypeScript*.

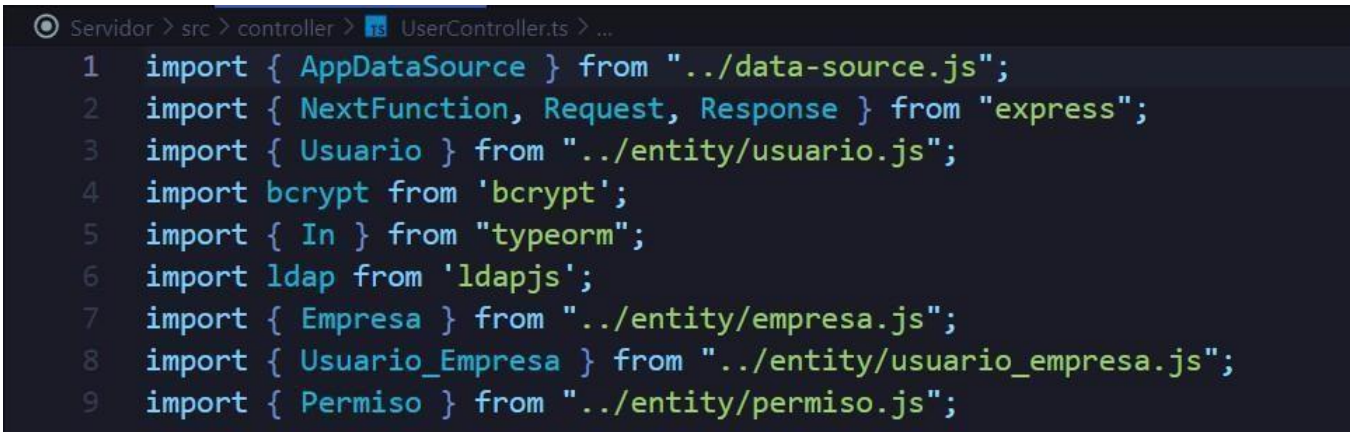

Figura 58: Importaciones con extensión .js

10. Conclusiones

El propósito de este TFG era desarrollar un software que automatizara un proceso que se estaba realizando de manera manual por el personal de la empresa Alibérico, desarrollando el mismo siguiendo unos estándares de seguridad y gestión para poder ser implementado en un entorno de producción.

Tras la revisión y documentación de las nuevas tecnologías que suponían la solución tecnológica se diseñó una arquitectura de un sistema que proporciona un esquema de funcionalidad completo para poder desempeñar las funciones que fueron definidas en los objetivos del proyecto.

El presente TFG tendrá un impacto significativo en la empresa, optimizando la labor del equipo de TI al reducir el tiempo y esfuerzo dedicados a la generación manual de informes. Esta herramienta no solo agilizará la obtención de datos, sino que también proporcionará un valor añadido a nivel empresarial, permitiendo revisar y analizar procesos productivos de manera más eficiente. Además, facilitará la comparación de datos operativos ante posibles reclamaciones, la generación de informes detallados sobre parámetros de proceso para clientes y el estudio del comportamiento de distintas materias primas, como metales, lacas y disolventes. Gracias a esta automatización y capacidad analítica, la empresa podrá mejorar la toma de decisiones, optimizar sus operaciones y garantizar una mayor trazabilidad y control sobre sus procesos.

Si bien se presentaron desafíos e inconvenientes mientras el proyecto era desarrollado que han tenido cierta relevancia en cuanto al tiempo que ha supuesto la configuración del entorno. Estos desafíos han supuesto la imposibilidad de que el acceso al motor de gráficas no llegue a funcionar mediante directorio activo pero la funcionalidad se encuentra implementada para que en un futuro puedan ser subsanadas estas características.

Por último, cabe resaltar el conocimiento adquirido en las tecnologías de desarrollo de software que componen el proyecto como el *Framework* de *Angular*, *Node.JS y MySQL*. Este aprendizaje, plasmado en el presente documento, ha dejado una base sólida para futuros desarrollos en el ámbito de la gestión y análisis de datos.

11. Líneas de futuro

A lo largo del desarrollo del TFG, se ha logrado cumplir con la mayoría de objetivos descritos inicialmente, sentando las bases de un sistema versátil y funcional que se adapta a las necesidades planteadas. No obstante como en todo desarrollo de software han surgido inconvenientes que se establecen como precedentes para unas líneas de desarrollo futuras completando la funcionalidad del sistema.

Uno de estos inconvenientes fue el inicio de sesión con directorio activo el cual se podría terminar de implementar enlazando el *Back-End* del entorno con el servidor *LDAP* de la empresa Alibérico para así poder acceder mediante un usuario con el dominio empresarial de manera simultánea que un inicio de sesión local.

Otro de los imprevistos que surgieron fue la incompatibilidad de que el servidor web fuera levantado sobre *HTTPS* debido al comportamiento del servidor de Alibérico a este tipo de peticiones, el cual sienta una base con la que se trabaje en el futuro para establecer la conexión de manera segura.

Además como continuación del estudio realizado de funcionalidades se podría implementar un sistema que admitiese una parametrización dinámica de las estructuras de datos que le llegarán al motor para darle versatilidad en cuanto al formato de datos que tenga acceso el motor de informes.

12. Estudio económico

Concepto	CANTIDAD UD	Precio Unitario	Subtotal
Licencia educativa de uso de la librería *Highcharts*	1	0€	0€
Entorno de desarrollo *VSCode*	1	0€	0€
Horas de trabajo del graduado en ingeniería telemática	300h	25€	7500€
		Total	**7500€**

Tabla 9: Estudio económico

Impuesto	Porcentaje	Precio total sin impuestos	Subtotal
IVA	21%	7500€	**9075€**

Tabla 10: Coste total con impuestos

El presupuesto necesario para la realización de este TFG sería de **9075€ con impuestos incluidos.**

13. Bibliografía

1. ***Angular Latinoamérica.*** *Guía de arquitectura. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://docs.angular.lat/guide/architecture*
2. ***Mozilla Contributors.*** *HTML: HyperText Markup Language*. MDN Web Docs. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML*
3. ***Mozilla Contributors.*** *CSS: Cascading Style Sheets*. MDN Web Docs. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS*
4. ***Mozilla Contributors.*** *JavaScript*. MDN Web Docs. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript*
5. ***Profile.es.*** *¿Qué es TypeScript vs JavaScript?* Profile.es Blog. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://profile.es/blog/que-es-typescript-vs-javascript/*
6. ***TypeScript.*** *TypeScript in 5 Minutes.* TypeScript Official Documentation. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html*
7. ***Node.js.*** *[s.f.]*. Node.js Official Website. *[Consultado el 11 de diciembre de 2024]. Disponible en: https://nodejs.org/es*
8. ***Platzi.*** *¿Qué es un ORM? Instalando y configurando TypeORM*. Platzi Clases. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://platzi.com/clases/2282-nestjs-typeorm/37294-que-es-un-orm-instalando-y-configurando-typeorm-mo/*
9. ***Microsoft.*** *¿Qué es Git?* Microsoft Learn. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://learn.microsoft.com/es-es/devops/develop/git/what-is-git*
10. ***Cloudflare.*** *¿Qué es la nube?* Cloudflare Learning. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.cloudflare.com/es-es/learning/cloud/what-is-the-cloud/*
11. ***Mozilla Contributors.*** *HTTP: Overview*. MDN Web Docs. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview*
12. ***FiloSottile.*** *mkcert*. GitHub. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://github.com/FiloSottile/mkcert*
13. ***Amazon Web Services.*** *¿Qué es una VPN?* AWS. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://aws.amazon.com/es/what-is/vpn/*
14. ***Microsoft.*** *¿Qué es Git?* Microsoft Learn. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://learn.microsoft.com/es-es/devops/develop/git/what-is-git*

15. ***KeepCoding.*** *¿Qué es una función hash?* KeepCoding Blog. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://keepcoding.io/blog/que-es-una-funcion-hash/*
16. ***Izertis.*** *Encriptación de password en Node.js y MongoDB: bcrypt.* Izertis Blog. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.izertis.com/es/-/blog/encriptacion-de-password-en-nodejs-y-mongodb-bcrypt*
17. ***OpenWebinars.*** *¿Qué es MySQL?* OpenWebinars Blog. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://openwebinars.net/blog/que-es-mysql/*
18. ***Oracle.*** *¿Qué es MySQL?* Oracle. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.oracle.com/es/mysql/what-is-mysql/*
19. ***Oracle.*** *¿Qué es una base de datos?* Oracle. *[s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.oracle.com/es/database/what-is-database/*
20. ***Express.js.*** *[s.f.]*. Express.js Official Website. *[Consultado el 11 de diciembre de 2024]. Disponible en: https://expressjs.com/es/*
21. ***Koa.js.*** *[s.f.]*. Koa.js Official Website. *[Consultado el 11 de diciembre de 2024]. Disponible en: https://koajs.com/*
22. **Ministerio del Interior.** Política de cookies. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.interior.gob.es/opencms/es/politica-de-cookies/
23. **Quest.** ¿Qué es Active Directory? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.quest.com/mx-es/solutions/active-directory/what-is-active-directory.aspx
24. **Amazon Web Services (AWS).** ¿Qué es IoT? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://aws.amazon.com/es/what-is/iot/
25. **UNIR.** Librerías de programación. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.unir.net/revista/ingenieria/librerias-programacion/
26. **IBM.** ¿Qué es una API? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.ibm.com/es-es/topics/api#:~:text=Una%20API%2C%20o%20interfaz%20de,intercambiar%20datos%2C%20caracter%C3%ADsticas%20y%20funcionalidades.
27. **UserGuiding.** Tooltips: ¿Qué son y cómo utilizarlos? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://userguiding.com/es/blog/tooltips
28. **Atlassian.** Glosario de Git: comandos esenciales. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.atlassian.com/es/git/glossary#commands
29. **IONOS.** ¿Qué es un plugin? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.ionos.es/digitalguide/servidores/know-how/que-es-un-plugin/
30. **JSON.** Página oficial de JSON. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.json.org/json-es.html
31. **DominiCode.** Cómo proteger rutas en Angular con los Guards. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://dominicode.com/como-proteger-tus-rutas-en-angular-con-los-guards/
32. **IONOS.** Modelo cliente-servidor: qué es y cómo funciona. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.ionos.es/digitalguide/servidores/know-how/modelo-cliente-servidor/

33. **Autoridades de Certificación.** Gobierno de España. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://firmaelectronica.gob.es/Home/Empresas/Autoridades-Certificacion.html
34. **RedesZone.** ¿Qué es LDAP y cómo funciona? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.redeszone.net/tutoriales/servidores/que-es-ldap-funcionamiento/
35. **HubSpot.** ¿Qué es un dominio? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://blog.hubspot.es/website/que-es-un-dominio
36. **Amazon Web Services (AWS).** Diferencias entre Frontend y Backend. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://aws.amazon.com/es/compare/the-difference-between-frontend-and-backend/
37. **Amazon Web Services (AWS).** ¿Qué es CLI? [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://aws.amazon.com/es/what-is/cli/
38. **Academia de Ciberseguridad.** Blowfish: Conceptos. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://aprende.academia-ciberseguridad.com/books/conceptos/page/blowfish
39. **RedesZone.** Criptografía: algoritmos de clave simétrica y asimétrica. [s.f.]. [Consultado el 11 de diciembre de 2024]. Disponible en: https://www.redeszone.net/tutoriales/seguridad/criptografia-algoritmos-clave-simetrica-asimetrica/

Anexos:
Anexo I. Manual de usuario

En el presente anexo tiene como objetivo proporcionar un manual de usuario detallado que facilite la compresión y el uso de las funcionalidades disponibles en la aplicación web del *motor de informes de IoT*. Según el rol designado de usuario se describen las acciones específicas de control y gestión como son los administradores del sistema, o las acciones de parametrización de gráficas y generación de informes en el caso del rol de los usuarios configuradores de gráficas. Este manual busca garantizar su uso eficiente y organizado de las herramientas disponibles, adaptándose a las responsabilidades de los roles determinados.

I.I. Inicio de Sesión

Para la conexión con la aplicación se requiere de una conexión a *Internet* además de un navegador en el que poder insertar la *URL* para acceder a la misma.

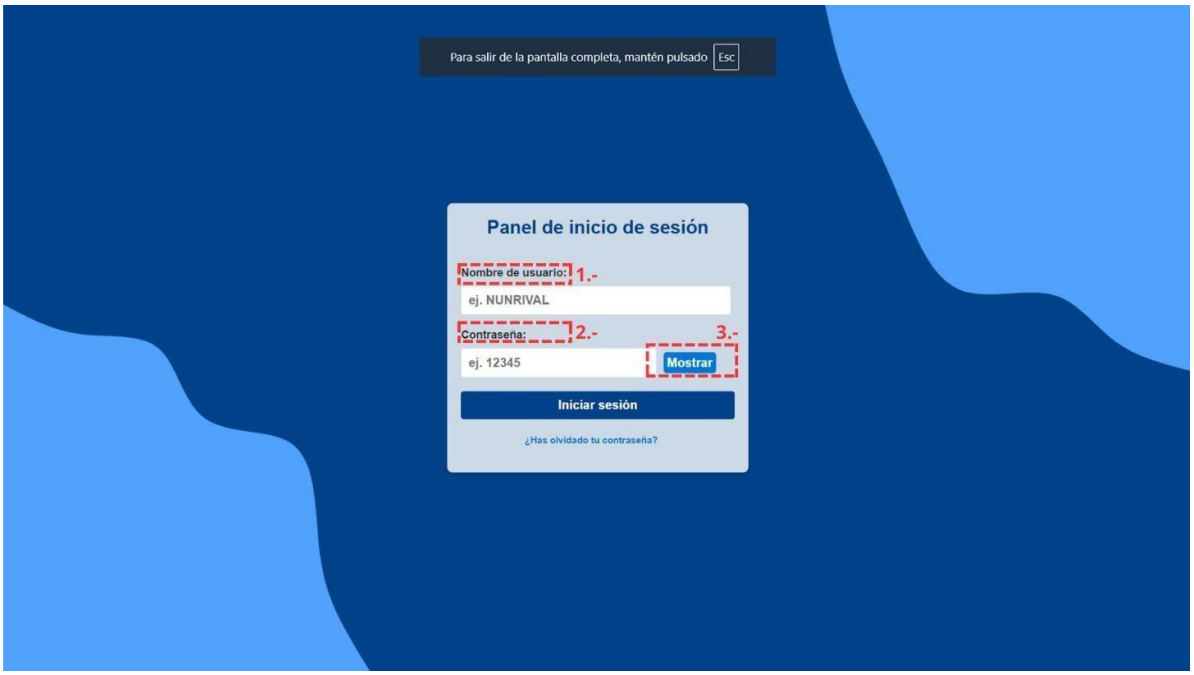

Figura 59: Inicio de sesión de la aplicación web

A la hora de iniciar sesión se debe introducir los parámetros previamente introducidos en la base de datos, en caso de que el usuario se encuentre registrado en el directorio activo de la empresa Alibérico su nombre seguirá la estructura de *nombredeusuario@aliberico.com* y en caso de no estarlo no dispondrá de este dominio. A continuación se detallas aspectos importantes en el panel de inicio de sesión.

1. Nombre de usuario: en este campo el usuario será capaz de introducir el nombre de usuario del que disponga ya sea para autenticarse mediante directorio activo o mediante la base de datos local.

2. Contraseña: en este campo el usuario introducirá su contraseña en texto plano y la aplicación mostrará una cadena de puntos con la longitud de la contraseña.

3. Botón de mostrar contraseña: el usuario podrá tocar este botón para alternar la vista de la contraseña entre puntos y la contraseña en texto plano.

Una vez autenticado exitosamente, el usuario se encontrará con un panel inicial diferente dependiendo del rol que tenga designado, en el caso de *Configuración de gráficas* será redirigido a la ruta */main/motorGraficas* y en el caso de los permisos *total* y *Administración* será redirigido a la ruta */admin*.

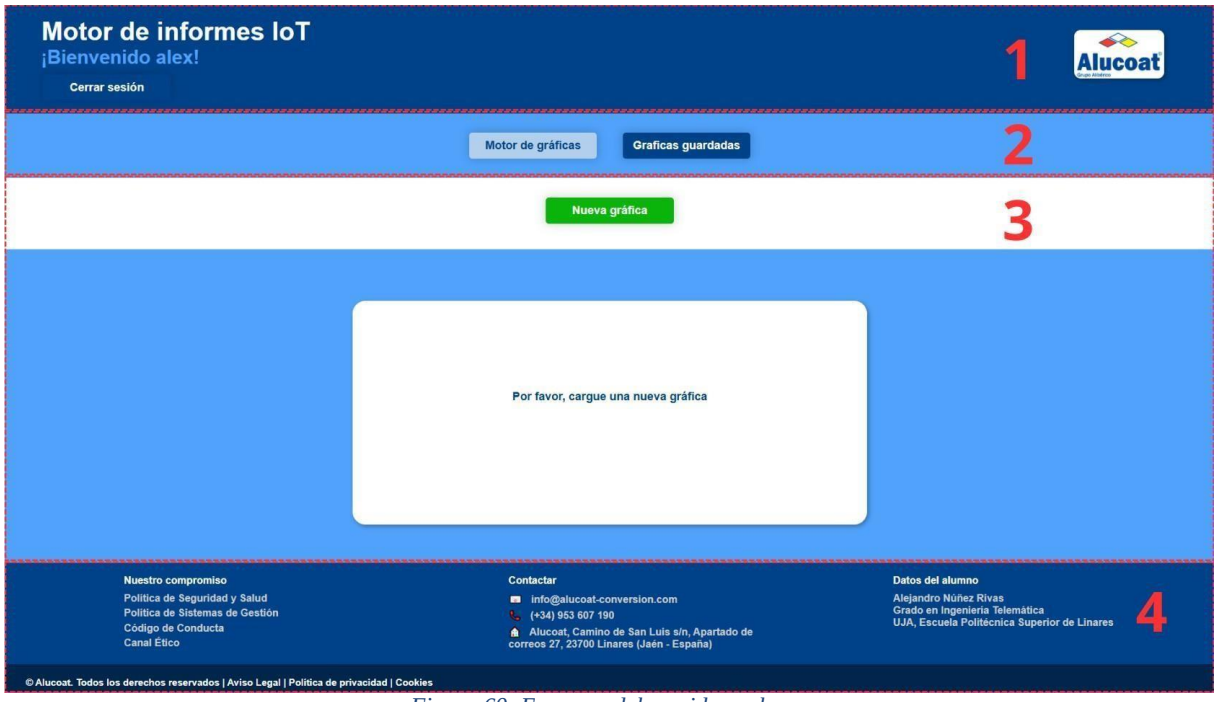

Figura 60: Esquema del servidor web

En la estructura que siguen las vistas de la aplicación web podemos diferenciar 4 zonas. La cabecera, el panel de navegación, el panel principal y el pie de página.

1. Cabecera: es esta zona el usuario puede verificar que ha iniciado sesión con el usuario correspondiente por el nombre que se muestra, además de poder cerrar sesión para volver al panel de inicio de sesión.

2. Panel de navegación: el usuario podrá navegar entre las diferentes vistas de la aplicación. En base a su permiso le aparecerán unas vistas y otras no serán accesibles.

3. Panel principal: en este panel el usuario dispondrá de la información o de las funciones correspondientes a la vista en la que se encuentre.

4. Panel del pie de página: contiene información relacionada con la sucursal de Alucoat, además de información de contacto en caso de requerirla.

I.II. Zona de usuario Configuración de gráficas

I.II.I. Vista Motor de gráficas

Partiendo de la ruta por defecto para usuarios con permiso de configuración de gráficas se encuentra la ruta */main/MotorDeGraficas*. Con este permiso tenemos acceso a la vista del corazón de la aplicación que resulta ser el motor de gráficas y a las gráficas guardadas.

Figura 61: Apartado de configuración de gráficas sin gráficas generadas

En el panel principal de esta vista se encuentra un botón que permite crear una nueva gráfica y debajo un recuadro que nos indica que no hay ninguna gráfica cargada.

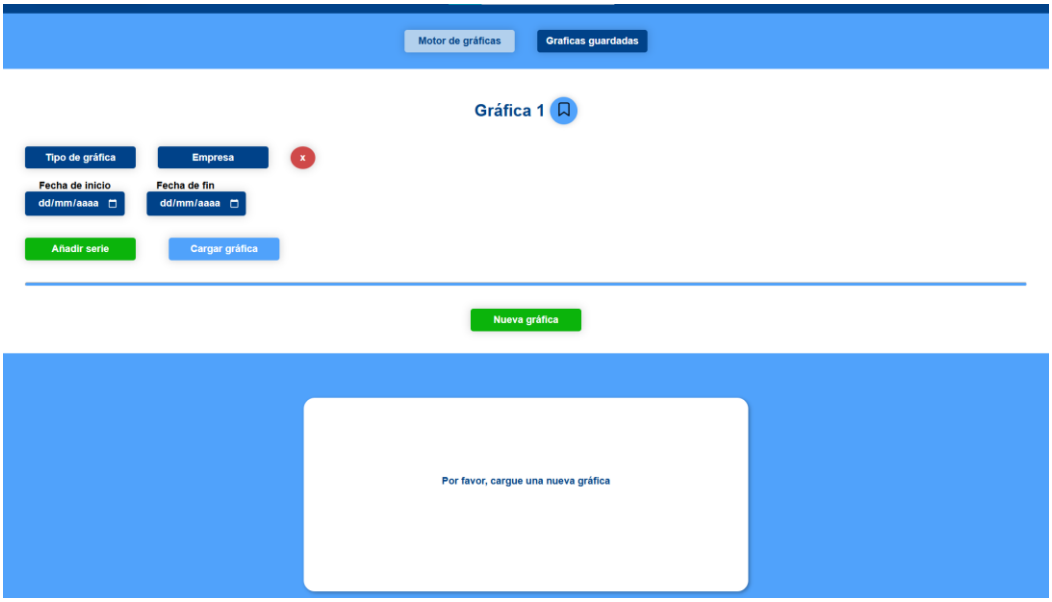

Figura 62: Apartado de configuración de gráficas al pulsar nueva gráfica

Una vez pulsado el botón de crear nueva gráfica se abrirá un desplegable en el que podremos seleccionar los parámetros para crear o guardar la gráfica que se desee, además de añadir series a esa gráfica que se genere o eliminar tanto esas series como las gráficas al completo.

Figura 63: Selectores para generar una gráfica principal

Como se puede observar en la imagen superior conforme se seleccione cada parámetro se añadirá el siguiente al lado hasta completar todos los necesarios antes de generar la gráfica. A continuación se detalla cada uno de los selectores en cuestión:

1. Tipo de gráfica: en este selector se selecciona el tipo de gráfica que se generará con los parámetros. Admite 3 valores, *Lineal, Barras o Circular*.

2. Empresa: indica la sucursal de la que se extraen los datos. Este selector se encuentra limitado por el nombre de las empresas de las que pertenece el usuario.

3. Línea: el usuario indicará en este selector el nombre de la línea de producción correspondiente en base a las líneas pertenecientes a la empresa que se haya seleccionado previamente.

4. Sensor: se indicará el nombre del sensor del que desea extraer los datos y los disponibles se encuentran limitados por los que pertenecen a la línea de producción que se haya seleccionado previamente.

Además de la serie principal es posible añadir series extra adicionales con el objetivo de comparar datos de otras empresas, líneas de producción o sensores en diferente o en el mismo intervalo temporal.

Figura 64: Selectores de la gráfica principal con una serie extra

En cuanto a los botones que disponemos en este panel, en la parte superior, junto al contador de gráfica se encuentra un botón azul con un icono de marcador en el que podremos una vez generada la gráfica guardarla para poder consultar más tarde en la vista de gráficas guardadas. Además, en el lado derecho de los selectores de la serie principal se encuentra un botón rojo con una cruz que sirve para eliminar por completo la gráfica que se haya generado como la zona de selectores asociada a la misma. Del mismo modo una vez se añada una serie extra disponemos de un botón que nos permite eliminar esa serie secundaria en términos de la zona de selectores como de la serie principal.

Una vez se hayan seleccionado todos los parámetros el usuario podrá pulsar el botón de cargar gráfica y se generará la gráfica con o sin series extras en la zona de abajo.

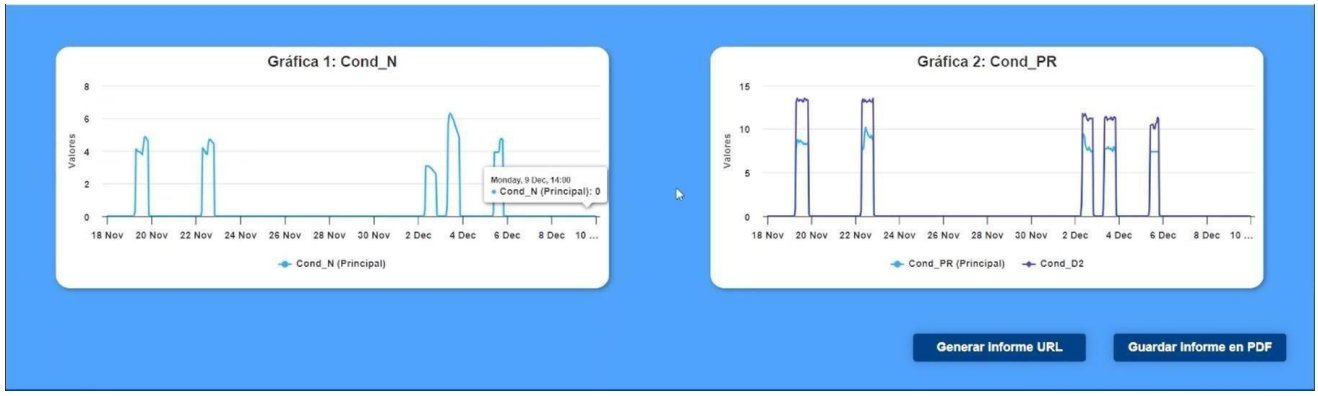

Figura 65: Panel de gráficas generadas

Al haber generado como mínimo una gráfica se desbloquearán dos botones nuevos al final de la sección de las gráficas generadas, el primer botón sirve para generar un informe en formato *PDF* de todas las gráficas generadas, y el siguiente consiste en generar un informe único capaz de ser inyectado en otros entornos en forma de una *URL* única que solo contenga las gráficas que se encuentren en el entorno.

En caso de pulsar el botón de descargar el informe en *PDF* se descargará automáticamente el archivo en el buscador web que se esté utilizando. El informe contiene en el mismo orden en el que se han generado las gráficas además de guardar el estado de manipulación en el que se encuentren, ya que al poder hacer zoom, la gráfica se guarda exactamente como se esté visualizando en el cliente web.

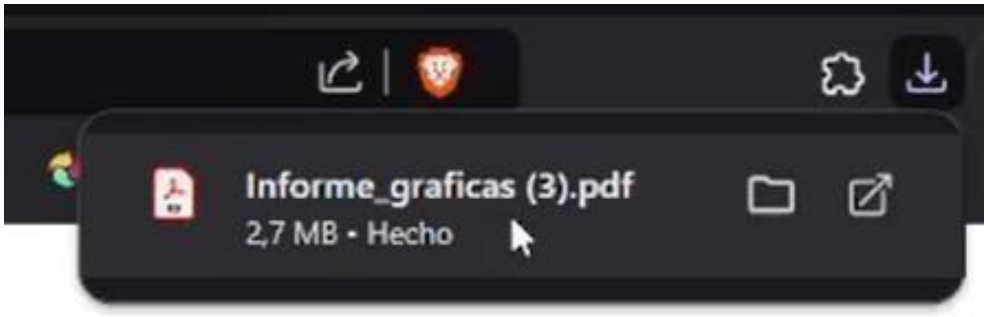

Figura 66: Descarga del informe en PDF

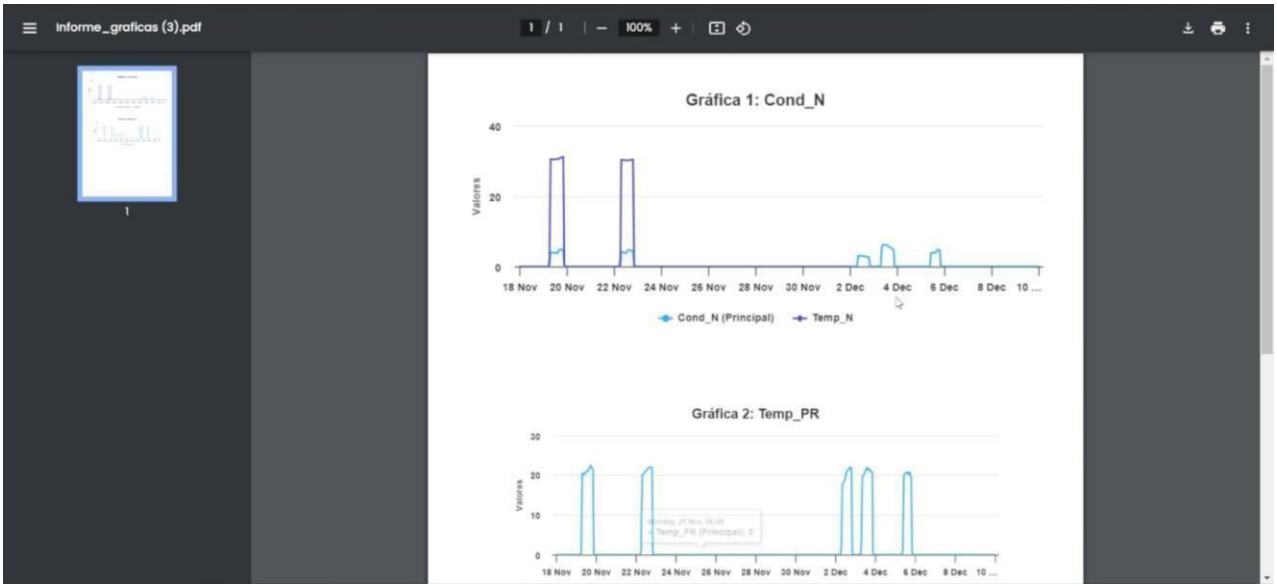

Figura 67: Informe en PDF

Y en el caso de pulsar el botón de informe mediante *URL*, se redirigirá al usuario a una ruta única */informes/<idInforme>* donde sólo se encontrarán las gráficas que tuviese el usuario generados en su cliente web.

Figura 68: Informe en URL

I.II.II. Vista de Gráficas guardadas

Cuando se parametrice una gráfica se tiene la posibilidad de guardar las gráficas en este otra vista para su posterior revisión. Este mecanismo se basa en un botón de marcador que se dispone en el lado derecho de cada título de cada gráfica, en caso de tener un color azul claro se considera como no guardada la gráfica y en caso de pulsar el botón se cambiaría el color a una tonalidad más oscura indicando que se ha guardado la gráfica en la base de datos local.

Figura 69: Comparación de botones guardado y sin guardar

Una vez se guarden gráficas en el motor de informes, el usuario podrá dirigirse a esta otra vista en la que podrá visualizar en orden ascendente respecto al número de la gráfica todas las que haya guardado. En esta vista solo se podrán consultar las gráficas que haya guardado el usuario mismo en específico, en caso de querer compartir gráficas se debería generar un informe en *URL*.

Figura 70: Gráficas guardadas

I.III. Zona de usuario Administración

En el caso de iniciar sesión correctamente un usuario con el permiso de *Administración* el usuario será redirigido a la ruta */admin* en la que se encuentra la vista inicial del panel de control de los administradores, una vista para listar y gestionar los sensores que se encuentran registrados en la base de datos local y una última en la que se puede listar y gestiones los usuarios que de igual manera se encuentran guardados en la base de datos.

I.III.I. Vista Inicio

La vista inicial no ofrece una funcionalidad específica pero sirve para tener una vista en la que

Figura 71 : Panel principal de Administración

I.III.II. Vista Listar sensores

En la vista listar sensores

Figura 72: Panel listar sensores

En la parte superior se dispone de un buscador que filtrará la lista de sensores por el campo del nombre comparando la cadena de caracteres que el usuario introduzca con esta variable y la aplicación descartará el resto de sensores que no empiecen por la cadena que se esté introduciendo exactamente.

A continuación, se disponen 3 filtros de selección que cargan los distintos valores posibles de la base de datos para dar las opciones de manera dinámica. Los detalles de cada selector se detallan a continuación.

- Filtrar Empresa: este filtro lista las empresas que se encuentren en la base de datos y se compara al valor *Empresa* del sensor, en caso de coincidir se mantiene, en caso contrario se vuelve invisible.

- Filtrar Línea de Producción: este selector muestra el nombre de las líneas de producción de la base de datos y se compara al valor *Línea de Producción* de los sensores, en caso de coincidir se mantiene, en caso contrario se vuelve invisible.

- Filtrar Tipo de Magnitud: este selector muestra los tipos de magnitudes que se encuentran en la base de datos y se compara al valor *Tipo de Magnitud* de los sensores, en caso de coincidir se mantiene, en caso contrario se vuelve invisible.

El botón de añadir sensor nos redirige a la vista de creación de nuevos sensores.

En la parte inferior de la vista se carga de manera dinámica el listado de sensores de la base de datos local en donde se puede visualizar las variables que contiene cada uno y los valores que presenta cada sensor en específico. En caso de pulsar la tarjeta de un sensor en específico la aplicación nos dirigirá a la vista de gestión del sensor.

I.III.III. Vista Gestión de sensor

En esta vista se encuentra únicamente la tarjeta del sensor que hayamos seleccionado en la vista de listar sensores además de dos botones que permiten la gestión de este sensor específico.

Figura 73: Panel gestión de sensores

- En caso de pulsar el botón de *Modificar sensor <nombre del sensor>* se nos redirigirá a la vista de modificación del sensor.

- En caso de pulsar el botón de *Eliminar sensor <nombre del sensor>* la aplicación nos mostrará un mensaje en el que se podrá confirmar la eliminación del sensor. Si se pulsa el botón de *Aceptar* se eliminará el sensor y se redirigirá de vuelta a la vista de listar sensores y en el caso de pulsar *Cancelar* simplemente se cancelará la operación y se mantendrá en la vista.

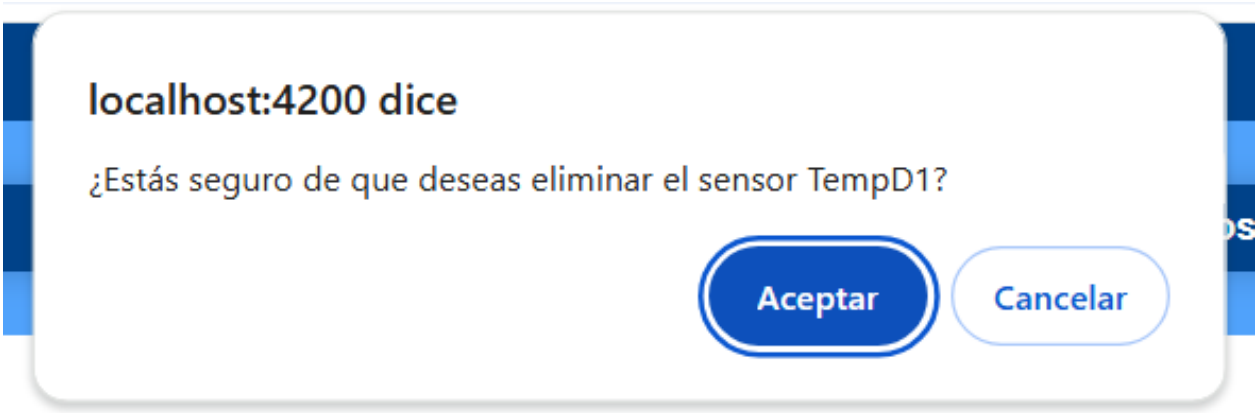

Figura 74: Confirmación de eliminación de sensor

I.III.IV. Vista Modificación de Sensores

Del mismo modo que se puede crear un sensor se puede modificar este mediante un formulario en el que se podrá reescribir los valores de *Nombre de sensor, Información del sensor, Tipo de magnitud y Línea de producción.*

Figura 75: Panel de modificación de sensor

I.III.V. Vista Creación de Sensores

Como se ha comentado se accede a través de la vista de listar sensores mediante el botón de crear un nuevo sensor que encaminará al usuario a esta nueva vista.

Para poder crear un nuevo sensor el usuario administrador podrá introducir una serie de parámetros y selecciones que se detallan a continuación:

Figura 76: Panel de creación de sensor

1. Nombre del sensor: En este campo el usuario podrá introducir una cadena de caracteres que sirvan para denominar al sensor que se va a dar de alta.

2. Información del sensor: Donde se podrá especificar algún tipo de información adicional como especificar más información acerca de la función del sensor.

3. Tipo de magnitud: Selector en el que se especifica el tipo de magnitud que mide el sensor pudiendo elegir entre los valores de *Temperatura, conductividad o golpes por minuto.*

4. Línea de producción: Selector en el que se podrá asociar el sensor que se esté dando alta a una línea de producción existente en la base de datos.

En el formulario se deberá indiciar al menos un carácter para el nombre del sensor y para la información además de estar todos los selectores con valores asignados. Una vez tenido en cuenta estos requisitos, el usuario deberá pulsar el botón de *crear sensor* y la aplicación web mostrará un mensaje comentando si la acción se realizó correctamente o si hubo algún problema en el proceso.

I.III.VI. Vista Listar usuarios

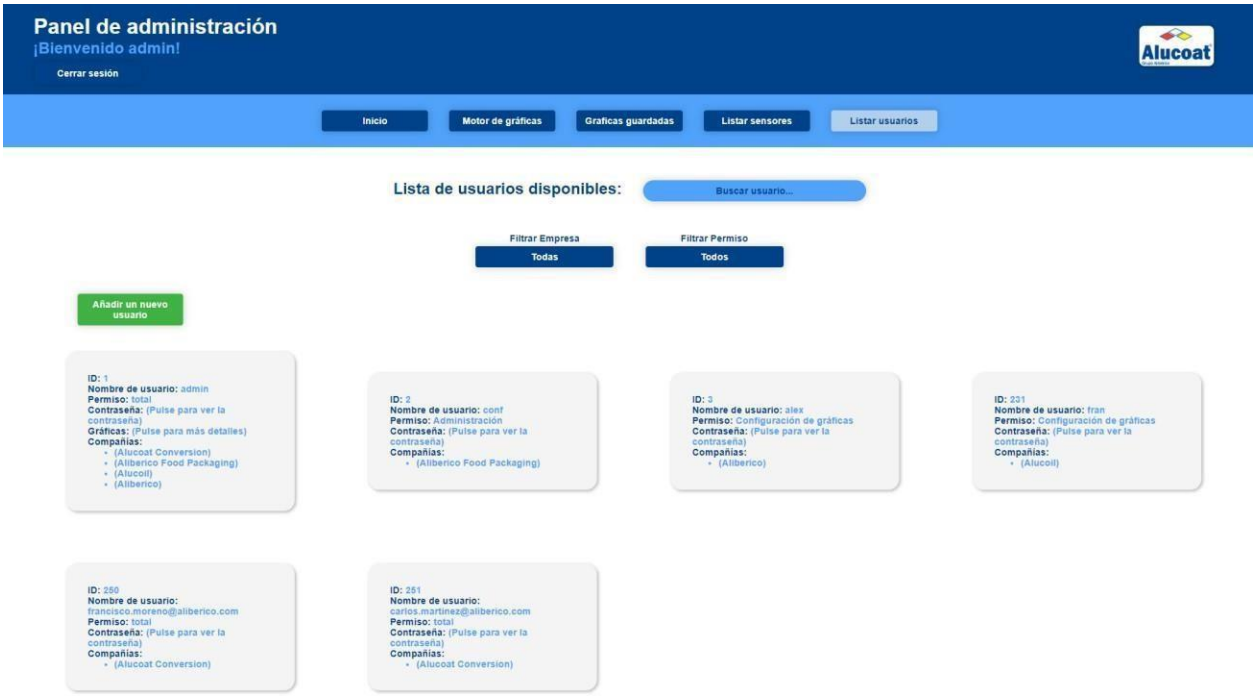

Figura 77: Panel de listar usuarios

En la parte superior se dispone de un buscador que filtrará la lista de usuarios por el campo del nombre comparando la cadena de caracteres que el usuario introduzca con esta variable y la aplicación descartará el resto de usuarios que no empiecen por la cadena que se esté introduciendo exactamente.

A continuación, se disponen 2 filtros de selección que cargan los distintos valores posibles de la base de datos para dar las opciones de manera dinámica. Los detalles de cada selector se detallan a continuación.

- Filtrar Empresa: este filtro lista las empresas que se encuentren en la base de datos y se compara al valor *Empresa* del usuario, en caso de coincidir se mantiene, en caso contrario se vuelve invisible.

- Filtrar Permiso: este selector muestra el nombre de los permisos de la base de datos y se compara al valor *Permiso* de los usuarios, en caso de coincidir se mantiene, en caso contrario se vuelve invisible.

El botón de añadir usuario nos redirige a la vista de creación de nuevos usuarios.

En la parte inferior de la vista se carga de manera dinámica el listado de usuarios de la base de datos local en donde se puede visualizar las variables que contiene cada uno y los valores que presenta cada usuario en específico. En caso de pulsar la tarjeta de un usuario en específico la aplicación nos dirigirá a la vista de gestión del usuario.

I.III.VII. Vista Gestión de usuarios

En esta vista se encuentra únicamente la tarjeta del usuario que hayamos seleccionado en la vista de listar usuarios además de dos botones que permiten la gestión de este usuario específico. Se encuentra información adicional como el *hash* de la contraseña del usuario además de la lista de gráficas que se encuentran guardadas por el usuario.

Figura 78: Panel de gestión usuario

- En caso de pulsar el botón de *Modificar usuario <nombre del usuario>* se nos redirigirá a la vista de modificación del usuario.

- En caso de pulsar el botón de *Eliminar usuario <nombre del usuario>* la aplicación nos mostrará un mensaje en el que se podrá confirmar la eliminación del usuario. Si se pulsa el botón de *Aceptar* se eliminará el usuario y se redirigirá de vuelta a la vista de listar usuario y en el caso de pulsar *Cancelar* simplemente se cancelará la operación y se mantendrá en la vista.

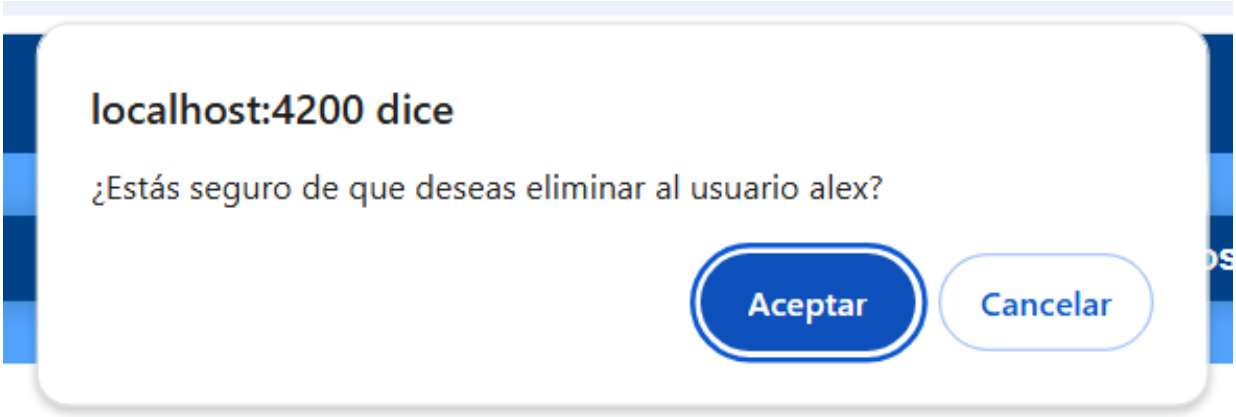

Figura 79: Confirmación de eliminación de usuario

I.III.VIII. Vista Modificación de usuarios

Del mismo modo que se puede crear un usuario se puede modificar este mediante un formulario en el que se podrá reescribir los valores de *Nombre de usuario*, *Contraseña*, *Permiso* y las empresas a las que se encuentre asociado.

Figura 80: Panel de modificación de usuario

I.III.IX. Vista Creación de usuarios

Como se ha comentado se accede a través de la vista de listar usuarios mediante el botón de crear un nuevo usuario que encaminará al usuario a esta nueva vista.

Para poder crear un nuevo usuario el administrador podrá introducir una serie de parámetros y selecciones que se detallan a continuación:

Figura 81: Panel de creación de usuario

1. Nombre del usuario: En este campo el administrador podrá introducir una cadena de caracteres que sirvan para denominar al usuario que se va a dar de alta.

2. Contraseña: Donde se especificará la contraseña en texto plano que contendrá el usuario.

3. Permiso del usuario: Selector en el que se podrá elegir un permiso para el nuevo usuario. Este permiso se carga dinámicamente de la base de datos pudiendo elegir entre *Configuración de gráficas*, *total* o *Administración*.

4. Empresas de las que dispone de permiso: Selector múltiple en el que se podrá elegir la empresa o empresas de las que tendrá permiso para generar gráficas con los datos de la misma. Esta lista se carga dinámicamente de la base de datos pudiendo elegir entre *Alucoat Conversion*, *Aliberico Food Packaging*, *Aliberico* y *Alucoil*.

En el formulario se deberá indiciar al menos un carácter para el nombre del usuario y para la contraseña, además de estar todos los selectores con valores asignados. Una vez tenido en cuenta estos requisitos, el usuario deberá pulsar el botón de *crear usuario* y la aplicación web mostrará un mensaje comentando si la acción se realizó correctamente o si hubo algún problema en el proceso.

Anexo II. Guía de implementación

II.I. Introducción

En este anexo se analiza en detalle cómo se puede implementar los distintos elementos que conforman este TFG para su posterior despliegue en entornos locales. A continuación, se incluyen los requisitos previos y las instrucciones en detalle para desplegar el cliente web con *Angular*, el servidor con *Node.js* y la interconexión de los elementos con la base de datos.

II.II. Requisitos previos

Para el correcto funcionamiento de los sistemas es necesario una serie de requisitos antes de comenzar con el despliegue de los componentes.

- Disponer de una conexión a *Internet* para poder descargar las dependencias necesarias.

- Acceso al repositorio en la nube de *Azure DevOps* de Alibérico para poder enlazar mediante *Git* nuestro *IDE* y así poder descargar el código del *Front-End y Back-End*

- Instalar los programas de *Git, MySQL installer Community y MySQL Workbench y Node.js.*

- Instalar las dependencias de *Npm, Angular, TypeScript, Express, Higcharts, Bcrypt y mkdir.*

El acceso al repositorio de la empresa de Alibérico se gestionará por parte de la empresa y una vez concedido el acceso se detalla a continuación los pasos para realizar la conexión del entorno de desarrollo con la nube de *Azure DevOps*.

Primero se deberá acceder al espacio de trabajo en la nube y una vez dentro se tendrá que elegir el repositorio del que se quiera clonar el código para poder acceder a sus archivos de manera local.

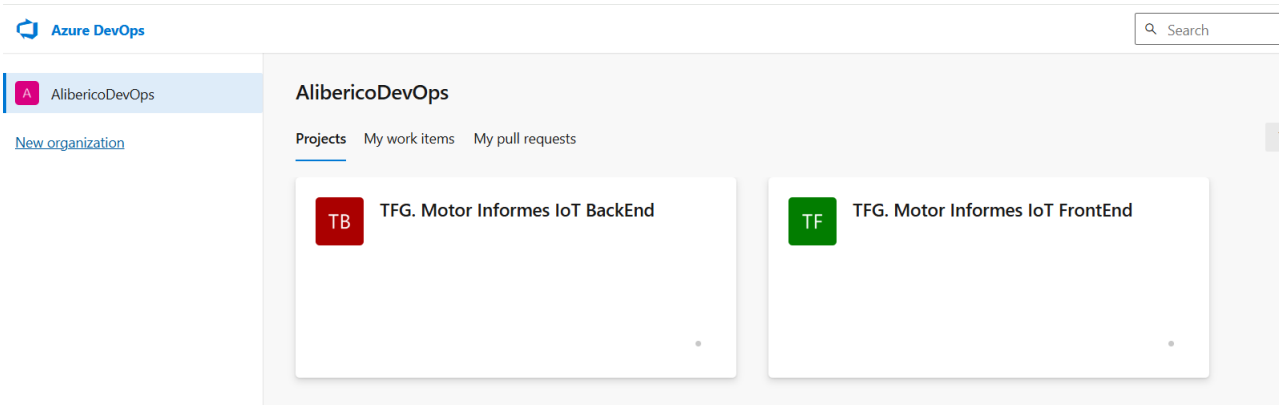

Figura 82: Panel selector de repositorio Azure DevOps

Una vez elegido el repositorio, se encontrará el panel inicial en el que podremos consultar información como resúmenes de estadísticas, los miembros que conforman el espacio de trabajo o una descripción del mismo en caso de que existiese. Para poder descargar el código habrá que dirigirse al apartado de *Repos*.

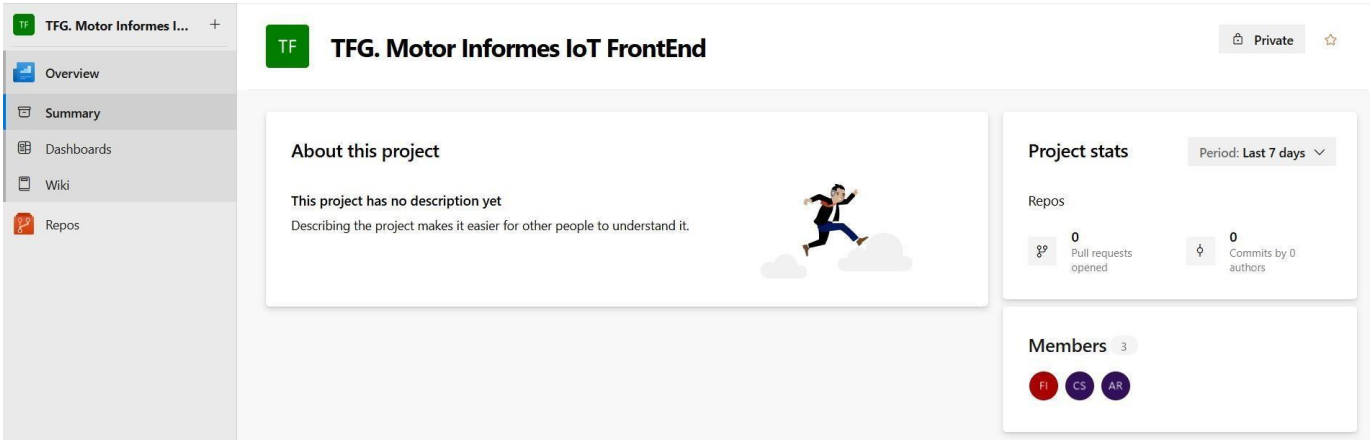

Figura 83: Panel de vista general de repositorio Azure DevOps

En este otro panel se dispondrá de toda la información detallada acerca de los archivos que contiene el repositorio como las peticiones de subida de actualizaciones entre otros aspectos. A continuación se tendrá que dirigir al sub-apartado de *Files* en el que en la zona de la derecha contendrá un botón llamado *Clone* que se deberá pulsar para iniciar el proceso de clonación del repositorio.

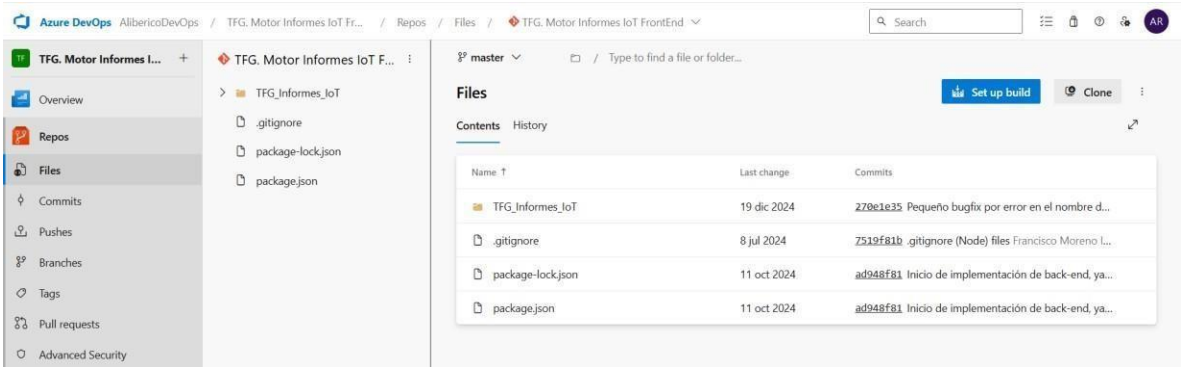

Figura 84: Panel de Files de Azure DevOps

Una vez pulsado el botón se abrirá un desplegable en el que se podrá seleccionar el entorno de desarrollo en el que se esté trabajando y automáticamente se establecerá una conexión nativa entre el repositorio de la nube junto con el *IDE* con la que ya se podrá realizar el control de versiones de ese repositorio en específico, en caso de querer enlazar el otro repositorio deberemos repetir el proceso que se ha detallado para otro directorio de nuestro entorno local.

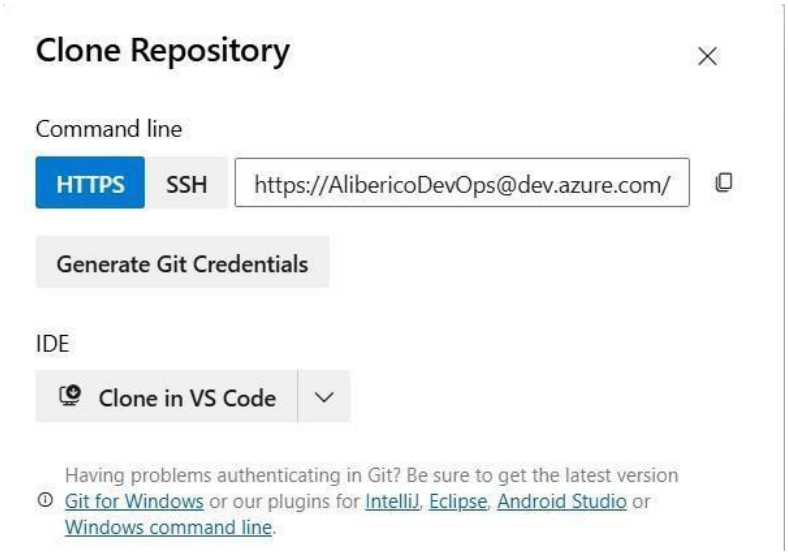

Figura 85: Panel de conexión entre Azure DevOps y VS Code

II.III. Configuración del Front-End

Una vez cumplidos los requisitos previos el primer paso para desplegar el entorno del *Front-End* que abarca el cliente web es abrir un nuevo terminal en el que se habrá de asegurar que se han instalado las dependencias correctamente mediante el comando *<ng version>*.

```
PS C:\Users\lsaez\OneDrive\Desktop\Frontend\TFG. Motor Informes IoT FrontEnd\TFG_Informes_IoT> ng version
>>

     _                      _                 ____ _     ___
    / \   _ __   __ _ _   _| | __ _ _ __     / ___| |   |_ _|
   / △ \ | '_ \ / _` | | | | |/ _` | '__|   | |   | |    | |
  / ___ \| | | | (_| | |_| | | (_| | |      | |___| |___ | |
 /_/   \_\_| |_|\__, |\__,_|_|\__,_|_|       \____|_____|___|
                |___/


Angular CLI: 18.2.8
Node: 22.13.0
Package Manager: npm 9.6.6
OS: win32 x64

Angular: 18.2.8
... animations, cli, common, compiler, compiler-cli, core, forms
... platform-browser, platform-browser-dynamic, router

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1802.8
@angular-devkit/build-angular 18.2.8
@angular-devkit/core         18.2.8
@angular-devkit/schematics   18.2.8
@schematics/angular          18.2.8
rxjs                         7.8.1
typescript                   5.5.4
zone.js                      0.14.10
```

Figura 86: Verificación de versión de Angular en CLI

Seguidamente se escribirá en el *CLI* el comando *<ng serve>* para intentar levantar el servidor web con los parámetros por defecto, en caso de que el *IDE* construya la aplicación web sin problemas ya tendremos el servidor web levantado sobre *HTTP*, en caso contrario se deberá consultar al error que nos muestre la interfaz para que sea subsanado y se pruebe otra vez.

```
PS C:\Users\lsaez\OneDrive\Desktop\Frontend\TFG. Motor Informes IoT FrontEnd\TFG_Informes_IoT> ng serve
Initial chunk files | Names         |  Raw size
main.js             | main          | 474.62 kB |
polyfills.js        | polyfills     |  90.20 kB |
styles.css          | styles        | 200 bytes |

                    | Initial total | 565.02 kB

Application bundle generation complete. [5.184 seconds]

Watch mode enabled. Watching for file changes...
NOTE: Raw file sizes do not reflect development server per-request transformations.
  → Local:   http://localhost:4200/
  → press h + enter to show help
```

Figura 87: Despliegue de servidor web sobre HTTP en CLI

Después de verificar que se ha desplegado correctamente de manera no segura sobre *HTTP* entonces se procederá a introducir el comando que despliega el servidor de manera

segura sobre *HTTPS* introduciendo los parámetros de certificado y llave que ya se encuentran generados en el código.

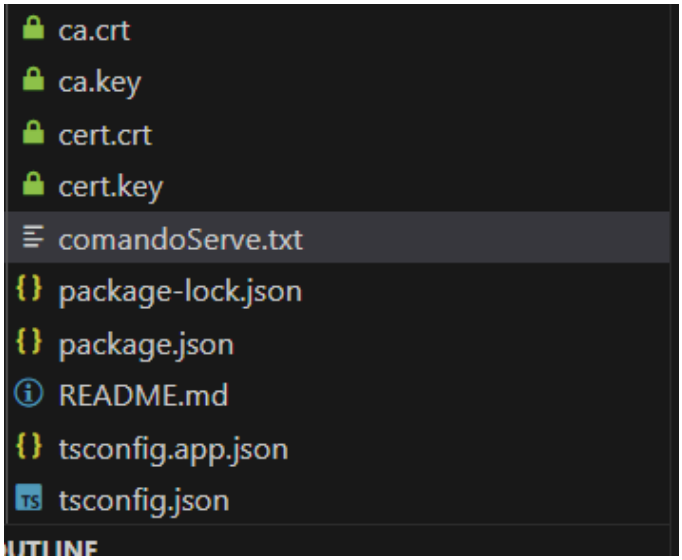

Figura 88: Archivos de autoridad de certificación y certifiado

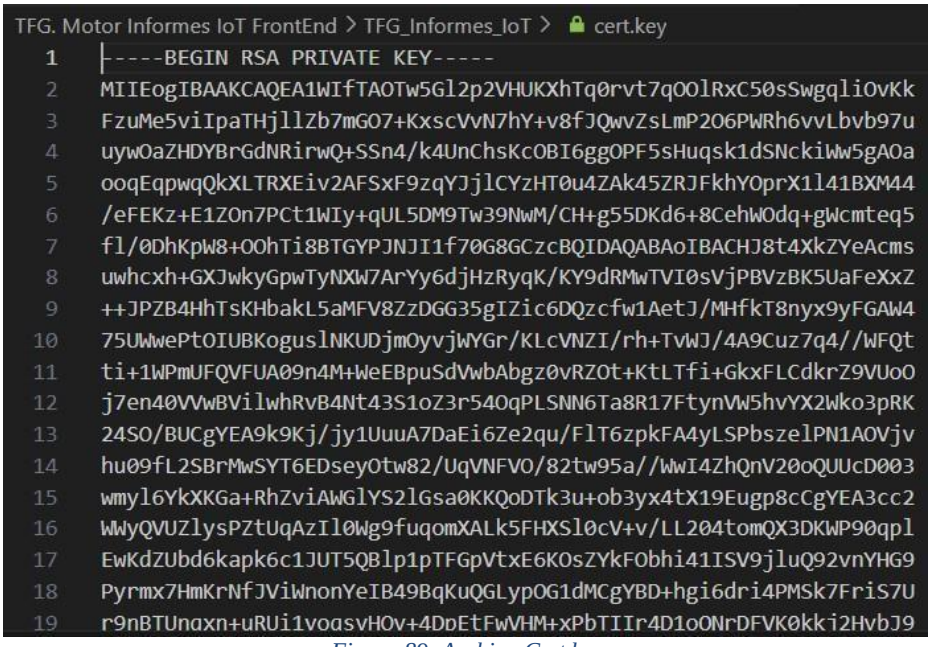

Figura 89: Archivo Cert.key

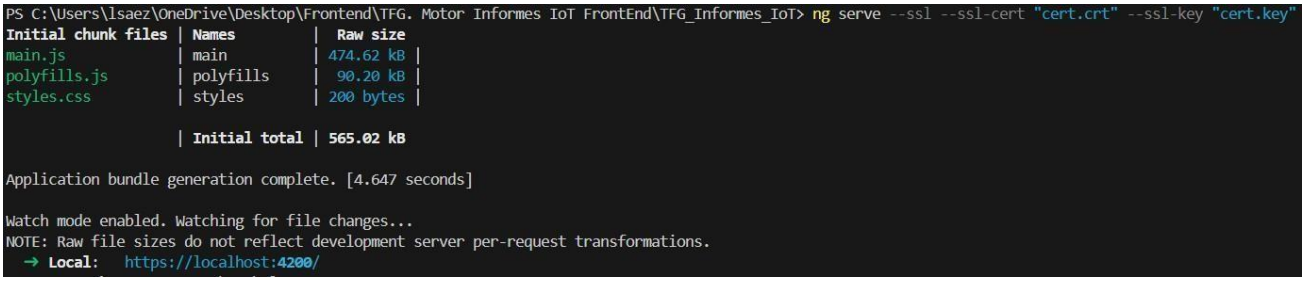

Figura 90: Despliegue del servidor web sobre HTTPS mediante CLI

II.IV. Configuración del Back-End

Para configurar el *Back-End* del entorno que abarca el servidor *Node.js* hay que en primer lugar abrir un nuevo terminal y escribir los comando <*node -v*> y <*npm-v*> para reafirmar que todo se haya instalado correctamente.

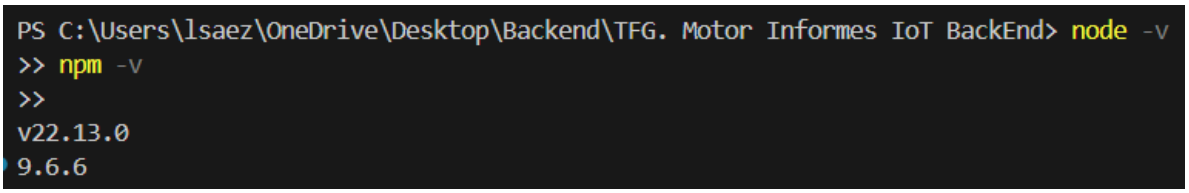

Figura 91: Comprobación de la versión node y npm en el CLI

Después de comprobar que las dependencias se instalaran correctamente se tendrá que modificar el archivo *data-source.ts* para la conexión con la base de datos. Para ello se tendrán que cambiar lo siguientes parámetros para su correcta conexión:

- *Type*: se especificará el tipo de base de datos que se vaya a conectar en este caso *MySQL*.
- *Host*: Dirección de la base de datos, en caso de que se despliegue en local dejarlo en su valor por defecto.
- *Port*: Puerto por defecto al que se conectará a la base de datos, en el caso de *MySQL*, el 3306
- *Username*: Nombre de usuario para autenticarse en la base de datos.
- *Password*: Contraseña para autenticarse en la base de datos.
- *Database*: Nombre de la base de datos.

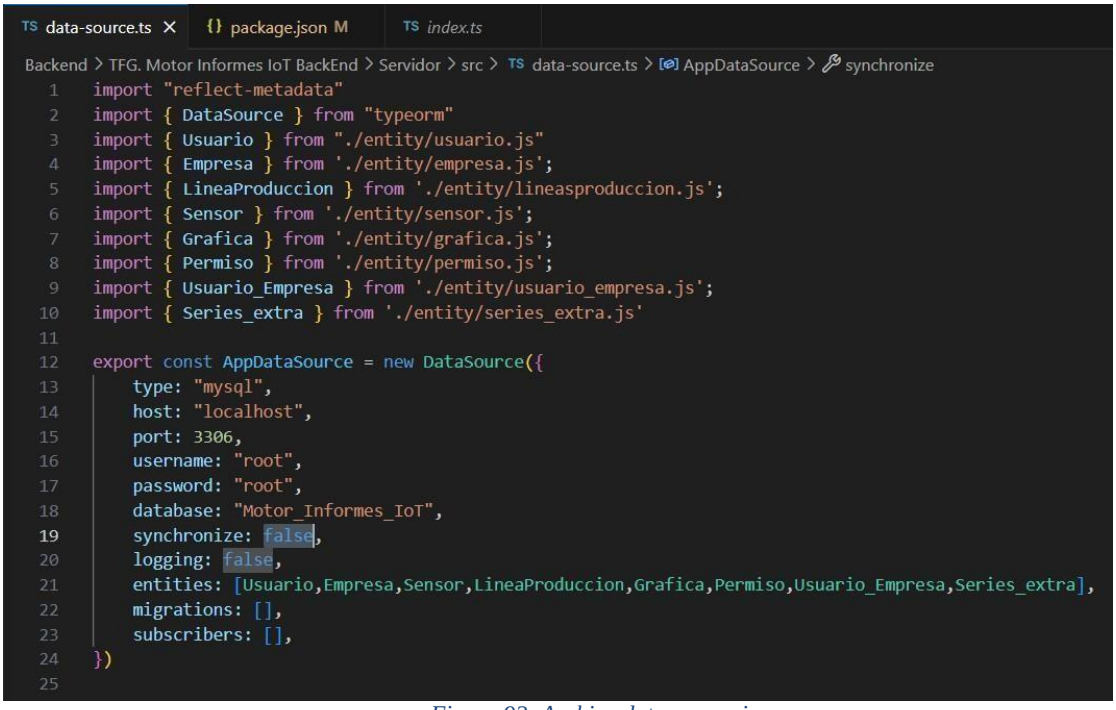

Figura 92: Archivo data-source.js

Por último después de configurar todo lo anterior se deberá escribir el comando *<node **/dist/index.js>* para levantar el servidor *Node*, en caso de que se despliegue correctamente se mostrará el siguiente mensaje indicando que todo se ha ejecutado sin problemas.

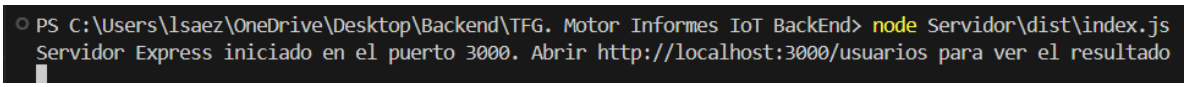

Figura 93: Despliegue del servidor Node.js

II.V. Configuración de la base de datos

En primer lugar se deberá asegurar de que se ha levantado correctamente el servicio de *MySQL* abriendo el *CLI* de *MySQL* e introduciendo el comando de status, en caso de que imprima un error habrá que reinstalar el servicio, y en caso de que imprima las estadísticas del servicio entonces se habrá instalado todo correctamente.

```
mysql> status
--------------
C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe  Ver 8.0.40 for Win64 on x86_64 (MySQL Community Server - GPL)

Connection id:          9
Current database:
Current user:           root@localhost
SSL:                    Cipher in use is TLS_AES_256_GCM_SHA384
Using delimiter:        ;
Server version:         8.0.40 MySQL Community Server - GPL
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    utf8mb4
Db     characterset:    utf8mb4
Client characterset:    cp850
Conn.  characterset:    cp850
TCP port:               3306
Binary data as:         Hexadecimal
Uptime:                 45 min 48 sec

Threads: 3  Questions: 7  Slow queries: 0  Opens: 120  Flush tables: 3  Open tables: 39  Queries per second avg: 0.002
--------------
```

Figura 94: Verificación del estado del servicio MySQL en su CLI

El siguiente paso será crear una conexión mediante el programa *MySQL Workbench*. Para ello se deberá iniciar el programa y seguidamente pulsar el botón con un símbolo de más para configurar una nueva conexión.

Figura 95: Panel de conexiones de MySQL

En la ventana que aparezca se deberán introducir los siguientes apartados para configurar la conexión con el servicio de *MySQL* que se ha levantado anteriormente, el resto de parámetros se deberán dejar por defecto.

- *Conecction name*: Nombre de la conexión con la base de datos que deberá coincidir con el nombre de *Database* que se configure en el *Back-End*.

- *Hostname*: Dirección que tiene el servicio de *MySQL*, al ser en local se deja en la dirección 127.0.0.1 que es sinónimo de localhost.

- *Port*: Puerto del servicio de *MySQL*, se tiene que mantener por defecto en 3306.

- *Username*: Nombre de usuario que deberá coincidir con el *username* que se establezca en el *Back-End.*

- *Password*: Contraseña que deberá coincidir con el *Password* que se establezca en el *Back-End.*

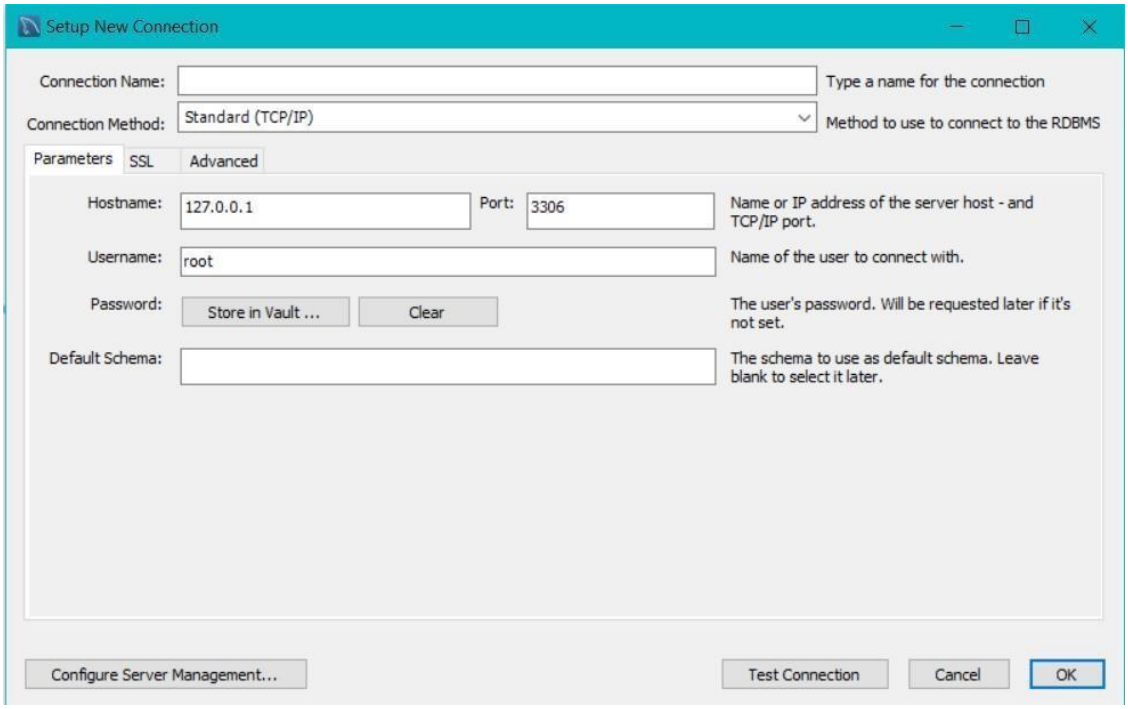

Figura 96: Panel de creación de nueva conexión de MySQL

El último paso será migrar la estructura del *Back-End* a la base de datos de manera manual o de manera automática. De manera manual se deberán de crear la misma estructura de tablas que se encuentra definida en el servidor *Node* con sus variables correspondientes y las relaciones entre las tablas o de manera automática se realizaría un proceso un poco más complejo pero mucho más eficiente de la siguiente manera:

1. En el servidor *Node* se deberá crear una carpeta llamada *Migraciones* en la raíz del proyecto.

2. Seguidamente se abrirá una terminal y se introducirá el siguiente comando <*npx typeorm-ts-node-commonjs migration:generate src/migration/Migracion --dataSource dist/data-source.js*> que creará el archivo de migraciones en el proyecto.

3. Por último se debe ejecutar la migración para que se copie la estructura del servidor a la base de datos mediante el comando <*npx typeorm-ts-node-commonjs migration:run --dataSource dist/data-source.js*>.

Después de ejecutar estos pasos se deberá asegurar en la vista del *MySQL Workbench* que se han generado las tablas de manera correcta, e incluso se puede generar un diagrama *Entidad-Relación* para asegurarse que las relaciones entre las tablas también se han migrado correctamente si se pulsa en *Database* > *Reverse Engineer* > (Se selecciona la conexión creada) > (Se selecciona el esquema de la base de datos) > *Execute* > *Finish*

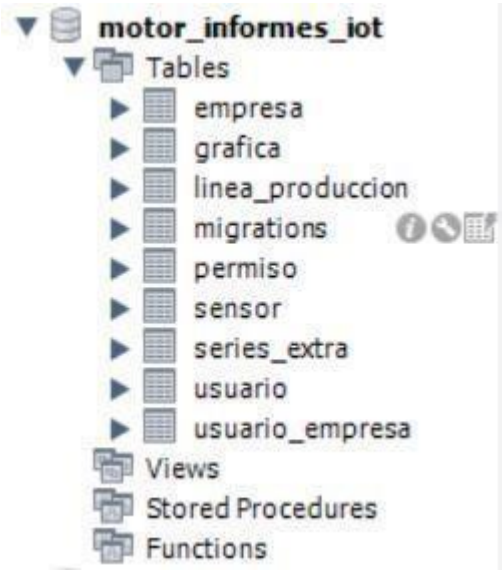

Figura 97: Tablas con las entidades en MySQL Workbench

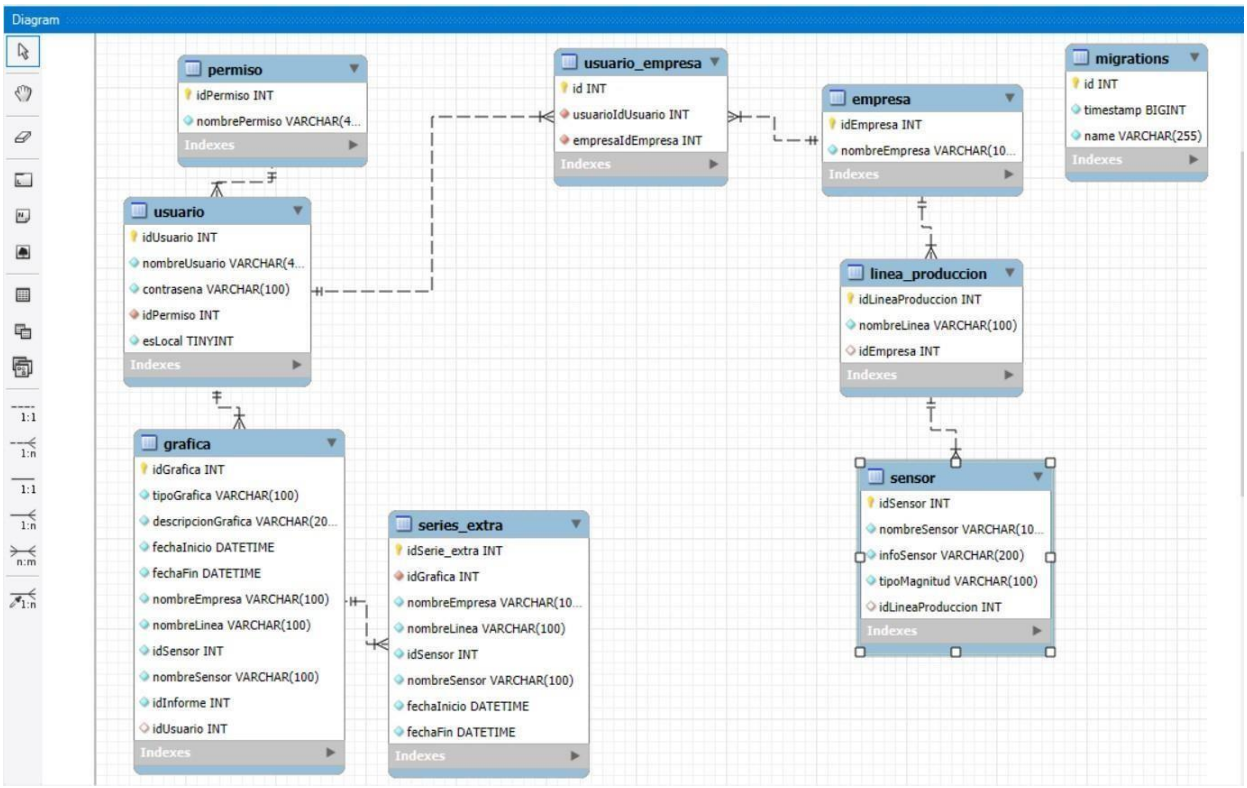

Figura 98: Diagrama Entidad-Relación de la base de datos de MySQL Workbench