UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería del Software

Sistema de Soporte y Docencia para el Despliegue Multiusuario de Node-RED

Realizado por
Pablo Carvajal Benítez

Tutorizado por
Daniel Garrido Márquez

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, SEPTIEMBRE DE 2024

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DEL SOFTWARE

Sistema de Soporte y Docencia para el Despliegue Multiusuario de Node-RED

Support and Teaching System for Multi-user Deployment of Node-RED

Realizado por

Pablo Carvajal Benítez

Tutorizado por

Daniel Garrido Márquez

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2024

Fecha defensa: Septiembre de 2024

Resumen

El presente Trabajo de Fin de grado tiene como eje central el desarrollo de un sistema multiusuario para el despliegue en la nube de Node-Red, una herramienta de desarrollo con gran relevancia en el ámbito del Internet de las Cosas. El mencionado sistema funcionará como soporte de Node-RED con el objetivo de eliminar las limitaciones que presenta actualmente, facilitando su acceso desde cualquier ubicación gracias al despliegue en la nube y permitiendo su uso simultáneo por múltiples usuarios.

El sistema en cuestión está compuesto por una aplicación web con un sistema de autenticación de usuarios con varios roles, donde cada usuario dispondrá de su propia instancia independiente de Node-RED, integrada en el sistema mediante el uso de contenedores para poder trabajar con la herramienta con normalidad. Además, el sistema conformado por la aplicación web desarrollada, las instancias de Node-RED y la base de datos, estará totalmente desplegado en la nube, lo que permitirá que los usuarios puedan almacenar y compartir sus flujos de trabajo con otros usuarios gracias al sistema de clases, sin necesidad de instalar nada y permitiendo continuar con el trabajo en cualquier otro dispositivo.

Palabras clave: Aplicación web, nube, multiusuario, Internet de las cosas, docencia

Abstract

The central focus of this Bachelor's Thesis is the development of a multi-user system for the cloud deployment of Node-RED, a development tool of great relevance in the field of the Internet of Things. The mentioned system will serve as support for Node-RED with the goal of addressing its current limitations, facilitating access from any location through cloud deployment, and enabling simultaneous use by multiple users.

The system in question consists of a web application with a user authentication system that supports multiple roles, where each user will have their own independent instance of Node-RED. These instances are integrated into the system using containers, allowing users to work with the tool normally. Additionally, the system, which includes the developed web application, the Node-RED instances, and the database, will be fully deployed in the cloud. This cloud deployment will enable users to store and share their workflows with others through a class system, without the need for any installations and allowing them to continue working on any other device.

Keywords: Web application, cloud, multiuser, IoT, teaching

Índice

1. Introducción

1.1 Motivación

Node-RED es una excelente herramienta pero que presenta algunas dificultades, ya que puede requerir una instalación local compleja y su uso para varios usuarios en el mismo dispositivo resulta complicado, ya que al no haber aislamiento entre las sesiones de los usuarios pueden ocurrir conflictos de configuraciones o flujos de trabajo, estas limitaciones, entre otras que se comentarán a continuación, complican el avance de muchos usuarios a la hora de aprender sobre su uso. Por ello, con el sistema desarrollado en el presente TFG, se busca eliminar todos estos inconvenientes, así como potenciar sus virtudes, entre las que encontramos su beneficioso uso en la docencia.

Node-RED destaca porque los usuarios pueden desarrollar prototipos rápidos de aplicaciones escribiendo poco código, por eso resulta contraproducente que una herramienta que pueden utilizar muchos usuarios sin amplios conocimientos técnicos demande una instalación compleja.

Otro aspecto problemático es la gestión del almacenamiento de los flujos de trabajo. En lugar de permitir su almacenamiento en una base de datos en la nube, Node-RED exige descargar los flujos en formato JSON y guardarlos localmente si se quieren conservar. Este procedimiento es poco adecuado, ya que aumenta la probabilidad de que el archivo se pierda.

Estas son solo algunos de las dificultades que presenta el uso de Node-RED, que en conjunto hacen que la experiencia de usuario no sea la mejor e incluso que consigan que usuarios se frustren antes estos inconvenientes y les impida aprovechar todas las ventajas que tiene Node-RED como es aprender sobre el desarrollo de aplicaciones para el Internet de las Cosas.

1.2 Objetivos

Este Trabajo de Fin de Grado presenta dos objetivos generales. El primero es eliminar las limitaciones en el uso de Node-RED y el segundo es potenciar su posible uso docente.

El objetivo de eliminar las limitaciones en el uso de Node-RED está subdividido en objetivos más específicos, que se comentarán a continuación:

- Despliegue en la nube de Node-RED: Este objetivo es muy importante, ya que poder usar Node-RED en la nube eliminaría varias limitaciones de la herramienta. Entre ellos se encuentra la obligación de una instalación local. Al poder acceder a Node-RED desde la nube se permitiría que los usuarios puedan utilizar la herramienta sin preocuparse con tener un dispositivo lo suficientemente potente o el tener que realizar configuraciones complejas, por lo que solo existiría el requisito de tener conexión a internet
- Despliegue en la nube de la base de datos: Otra limitación que se elimina es la de la persistencia de datos, ya que actualmente para conservar los flujos de trabajo se deben descargar en formato JSON y almacenar localmente cada flujo de trabajo como un archivo JSON, lo cual no es adecuado y se pueden perder los datos fácilmente. El poder almacenar los flujos de trabajo en la nube permite conservar los flujos de trabajo en un lugar seguro, además de tener la posibilidad de usar Node-RED desde otros dispositivos y continuar con el trabajo accediendo a los flujos de trabajo almacenados en la nube
- Sistema de autenticación de usuarios: Otro objetivo crucial es el de contar con un sistema de usuarios, donde se garantice que cada usuario sea el único capaz de acceder a sus flujos de trabajo y se permita utilizar su propia instancia independiente de Node-RED, evitando así los conflictos que surjan cuando varios usuarios utilicen la herramienta en un mismo dispositivo

El segundo objetivo general es el de potenciar su uso docente, para esto se ha implementado un sistema de clases donde habrá varios roles de usuario, el de alumno y profesor. Estos roles tendrán funcionalidades diferenciadas, ya que los usuarios con el rol de profesor podrán crear

clases e invitar a alumnos mediante correo electrónico. Además de compartir flujos con los alumnos a modo de plantilla y ver los flujos de trabajo compartidos por los alumnos para evaluarlos. Los profesores también podrán gestionar estas clases, pudiendo ver y eliminar a los participantes, entre otras funcionalidades. Por otro lado, los alumnos podrán unirse a estas clases, utilizar los flujos de trabajo de los profesores y entregar los flujos de trabajo realizados para su evaluación. Estas funcionalidades hacen que el aprendizaje de Node-RED en el ámbito docente sea mucho más colaborativo y fluido.

1.3 Estructura de la memoria

Esta será la estructura seguida en la memoria:

1. **Introducción:** se exponen los motivos y los objetivos clave del proyecto
2. **Tecnologías utilizadas:** Se enumeran las tecnologías que se han utilizado en el proyecto
3. **Metodología:** Se va a describir que tipo de metodología de trabajo se ha utilizado
4. **Captura y análisis de requisitos:** Se enumeran los requisitos del sistema.
5. **Diseño y arquitectura:** Se describe como se ha estructurado el sistema. Se muestran los diagramas de componentes y de navegación
6. **Desarrollo e Implementación:** Se van a explicar los hitos de la fase de desarrollo del sistema
7. **Despliegue:** Se comenta paso a paso como se ha desplegado el sistema y las dificultades presentadas
8. **Conclusiones y Líneas Futuras:** Se hace una conclusión del proyecto y se comentan las posibles mejoras en el futuro

Al final del texto se incluirá la lista de referencias bibliográficas, así como los apéndices con el manual de usuario, de despliegue y el de instalación.

2. Tecnologías utilizadas

2.1 Node-RED

Node-RED es una herramienta de desarrollo visual. Utiliza nodos conectados para crear flujos de trabajo sin necesidad de escribir una gran cantidad de código, por lo que es una herramienta ideal para quienes no tienen experiencia en programación. Está diseñado principalmente para aplicaciones IoT.

En Node-RED, puedes construir flujos de trabajo conectando visualmente los nodos predefinidos comentados anteriormente, que son bloques funcionales diseñados para tareas específicas. Estos nodos, que incluyen entradas, procesamiento y salidas, se conectan para formar lo que se conoce como un flujo de trabajo

En este Trabajo de Fin de Grado, todo ha girado en torno a potenciar esta herramienta para que el máximo número de usuarios puedan aprovecharse de sus grandes ventajas.

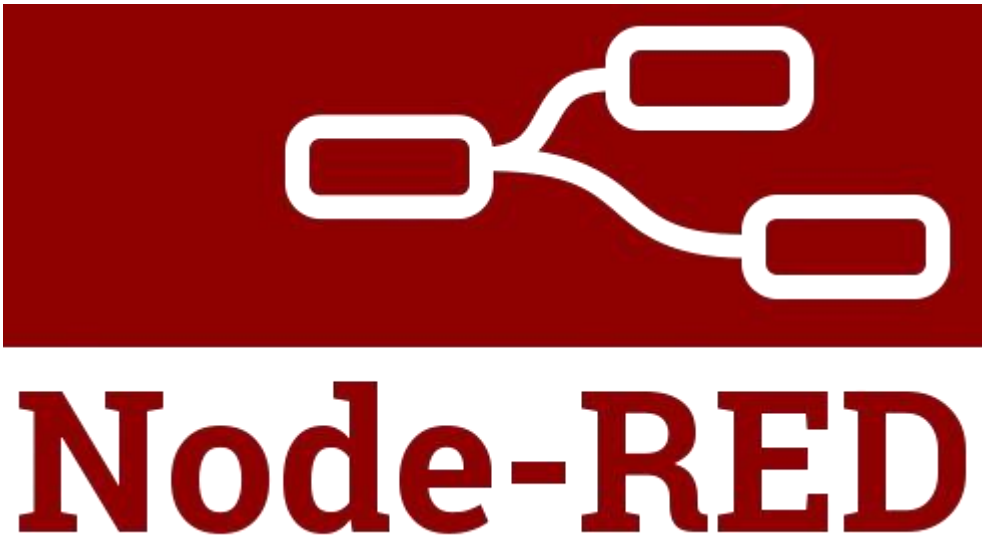


Figura 2.1: Logo de Node-RED. Fuente: Medium, 2022

2.2 JavaScript

JavaScript es un lenguaje muy importante para el desarrollo de sitios web, junto con HTML y CSS, se utiliza sobre todo en el lado del cliente para crear páginas interactivas. JavaScript permite modificar el contenido de la página en tiempo real, gestionar interacciones del usuario y mejorar la usabilidad sin necesidad de recargar la página completa.

Con el tiempo, JavaScript ha evolucionado y ahora también se usa en el Backend con Node.js. Ha avanzado tanto que destaca por su versatilidad, haciendo que, según W3Techs, más del 90% de los sitios web emplean JavaScript, lo que demuestra su potencial y su impacto en el desarrollo web.

JavaScript se ha utilizado en todos los componentes del sistema, respaldando lo anterior. Se ha utilizado en el Backend con Node.js, en el Frontend con React e incluso las instancias de Node-RED utilizan JavaScript. Utilizar el mismo lenguaje de programación para todos los componentes simplifica el desarrollo.

Figura 2.2: Logo de JavaScript. Fuente: Logos World, 2024

2.3 Node.js

Node.js es un entorno de ejecución para poder ejecutar código escrito en lenguaje JavaScript en el servidor. Facilita el desarrollo de aplicaciones Backend, y esto permite que se pueda desarrollar una aplicación web completa utilizando el mismo lenguaje de programación para ambos componentes.

Node.js ha sido utilizado para desarrollar el Backend del sistema, debido a su uso de JavaScript, haciendo que no se deba aprender o utilizar un lenguaje de programación diferente solo para el Backend.

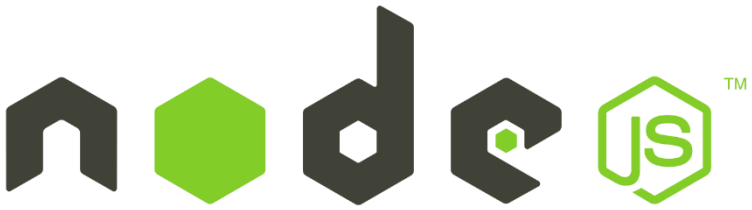


Figura 2.3: Logo de Node.js. Fuente: Xurxo Dev, 2015

2.4 Express.js

Express.js es un framework de Node.js que proporciona herramientas que facilitan la implementación de aplicación webs y APIs.

Ha sido utilizado en este sistema para realizar la implementación de la API Rest, ayudando a simplificar la gestión de rutas y middlewares, y facilitar el manejo de peticiones y respuestas HTTP.

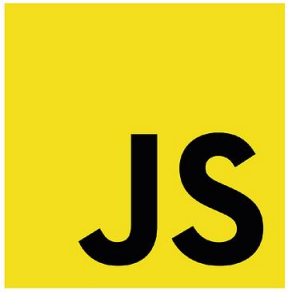


Figura 2.4: Logo de Express.js. Fuente: Medium: 2020

2.5 React

ReactJS es una librería JavaScript que se utiliza para construir interfaces de usuario de manera sencilla, todo esto utilizando componentes, que es una vista que se puede reutilizar en cualquier lugar de la aplicación. Por ejemplo, un componente podría ser la cabecera, la cual se define una vez y se puede reutilizar en cualquier otra parte del código. Aunque el lenguaje de programación que se usa es JavaScript, React utiliza la extensión JSX, esto permite escribir código HTML y CSS dentro de JavaScript.

Ha sido utilizado en este sistema para desarrollar el componente del Frontend del sistema, ya que es una librería de JavaScript, por lo que se utilizará el mismo lenguaje de programación que para el Backend y tiene la ventaja de que permite construir y reutilizar interfaces de usuario interactivas.

Figura 2.5 Logo de React. Fuente: Logos World, 2023

2.6 HTML

HTML es esencial para la creación y organización de contenido en internet. Utiliza tags y elementos para establecer la estructura de los sitios web. HTML forma la base de todos los sitios web, ya que proporciona el esqueleto sobre el cual se construyen las páginas.

Ha sido utilizado junto a JavaScript y CSS en los componentes de React para el Frontend.

Figura 2.6: Logo de HTML. Fuente: Pixabay, 2017

2.7 CSS

CSS sirve para establecer el aspecto visual de los sitios web. Posibilita la aplicación de estilos a los componentes de HTML, tales como tonalidades, tipografías y espacios, con el fin de mejorar la apariencia de las páginas.

Ha sido utilizado junto a JavaScript y HTML en el Frontend para mejorar el aspecto visual del sistema.

Figura 2.7: Logo de CSS. Fuente: 1000 logos, 2024

2.8 Bootstrap

Bootstrap es una herramienta de diseño que permite desarrollar páginas web y aplicaciones online de forma ágil y efectiva. Proporciona una colección de componentes y herramientas predefinidos, como botones, formularios, tablas y navegación, que facilitan el diseño sin necesidad de escribir una gran cantidad de código CSS desde cero

Ha sido utilizado en este sistema como framework de CSS en el Frontend porque proporciona una gran cantidad de componentes y estilos predefinidos.

Figura 2.8: Logo de Bootstrap. Fuente: PNGWing

2.9 MongoDB

MongoDB es una base de datos de tipo no relacional y guarda la información en formato Binary JSON. MongoDB no utiliza tablas y filas, sino que organiza los datos en colecciones de documentos.

MongoDB ha sido utilizado como la base de datos de este sistema.

Figura 2.9: Logo de MongoDB. Fuente: Wikimedia Commons, 2019

2.10 MongoDB Atlas

MongoDB Atlas es un servicio totalmente manejado de base de datos en la nube para MongoDB. Proporciona una manera simple y efectiva de desplegar, administrar y escalar bases de datos MongoDB en la nube sin tener que ocuparse de la infraestructura de detrás. Además, maneja automáticamente tareas como la copia de seguridad y la actualización del software.

Se ha utilizado MongoDB Atlas como la solución en la nube para gestionar la base de datos MongoDB.

Figura 2.10: Logo de MongoDB Atlas. Fuente: FlowyGo, 2020

2.11 MongoDB Compass

MongoDB Compass se trata de la interfaz visual para conectarse con bases de datos MongoDB. Proporciona a los usuarios la capacidad de analizar y manipular datos de forma visual e intuitiva, evitando la necesidad de usar la línea de comandos.

Se ha utilizado MongoDB Compass como herramienta para visualizar y administrar los datos almacenados en MongoDB de manera sencilla.

Figura 2.11: Logo de MongoDB Compass. Fuente: Medium, 2021

2.12 Visual Studio Code

Visual Studio Code es un entorno de desarrollo de programación ligero. Su popularidad se debe a la rapidez que ofrece y a su compatibilidad con numerosos lenguajes de programación y tecnologías. ofrece funciones como resaltado de código, autocompletado, depuración y control de versiones a través de Git.

Se ha utilizado como la herramienta para desarrollar todo el código del sistema.

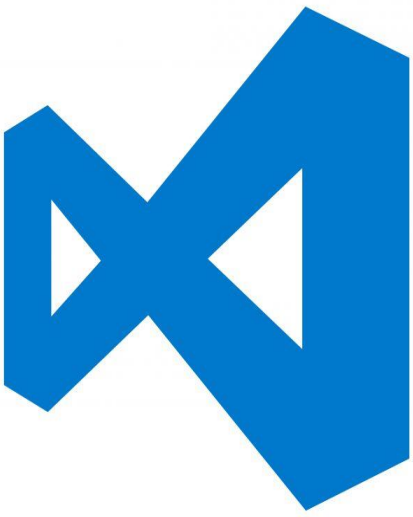


Figura 2.12: Logo de Visual Studio Code. Fuente: Seek Vectors

2.13 Git

Git es una herramienta distribuida de control de versiones que posibilita a los programadores seguir y controlar las modificaciones en el código fuente a lo largo del tiempo.

Aunque este proyecto ha sido realizado individualmente, se ha utilizado Git como control de versiones para documentar cualquier cambio en el código

Figura 2.13: Logo de Git. Fuente: Wikimedia Commons, 2012

2.14 GitHub

GitHub es una plataforma online de desarrollo que emplea Git para versionar y colaborar en proyectos de software. Permite almacenar los repositorios de código, trabajar con otros desarrolladores mediante pull requests y gestionar versiones del código fuente de manera eficiente.

Se ha empleado GitHub como la plataforma de gestión de versiones para alojar los proyectos.

Para la creación de este sistema se han empleado 3 repositorios, uno para el Backend, otro para el Frontend y otro para la versión modificada de Node-RED.

- Backend: https://github.com/pablocrv12/multiuser-node-red
- Frontend: https://github.com/pablocrv12/vm-node-red
- Node-RED modificado: https://github.com/pablocrv12/node-red-modified

Figura 2.14: Logo de GitHub. Fuente: Logos World, 2024

2.15 GitHub Desktop

Es una aplicación de escritorio que facilita el uso de Git y GitHub sin necesidad de interactuar directamente con la línea de comandos. Ofrece una interfaz gráfica intuitiva para gestionar repositorios, realizar commits, manejar ramas, y sincronizar cambios con GitHub.

En este proyecto se ha utilizado ya que hace mucho más sencillo el proceso de ejecutar los comandos, Por ejemplo, para realizar un comando 'Push', no se tiene que realizar desde la terminal, simplemente hay que pulsar un botón desde la aplicación

Figura 2.15: Logo de GitHub Desktop. Fuente: Wikimedia Commons, 2015

2.16 Postman

Postman es una herramienta de pruebas de API. La interfaz sencilla y fácil de usar ayuda a documentar, diseñar y comprobar las APIs. Puedes crear peticiones de endpoints de API, enviar varios tipos de datos y evaluar respuestas.

En este proyecto ha sido realmente útil en la fase de desarrollo, permitiendo probar el funcionamiento de la API cuándo el Frontend aún no estaba implementado del todo.

Figura 2.16: Logo de Postman. Fuente: Wikimedia Commons, 2021

2.17 Swagger

Swagger es tanto una herramienta como especificación para diseñar, construir y documentar APIs RESTful. Ofrece un formato estandarizado para describir la estructura de una API, incluyendo sus endpoints, métodos, parámetros y respuestas. Esta descripción permite generar documentación y validaciones automáticas

En este proyecto se ha utilizado una vez finalizado el desarrollo de la API para documentar todos los endpoints y poder probarlos con facilidad.

Figura 2.17: Logo de Swagger. Fuente: LinkedIn, 2020

2.18 MagicDraw

MagicDraw es una herramienta de diseño y modelado de software que se emplea mayormente en la creación de diagramas y modelos visuales. Además, es conocido por utilizar estándares como UML, BPMN y SysML.

Se ha utilizado para elaborar los diagramas de clase y componentes.

Figura 2.18: Logo de MagicDraw. Fuente: G2

2.19 Draw.io

Draw.io es una herramienta gratuita que se utiliza en línea, es decir sin necesidad de instalarla y sirve para elaborar diagramas y gráficos. La interfaz es fácil de usar y permite a los usuarios crear todo tipo de diagramas.

En este TFG se ha utilizado para diseñar los diagramas de navegación del Frontend.

Figura 2.19: Logo de Draw.io. Fuente: Pinterest

2.20 Docker

Docker es una herramienta que simplifica el desarrollo, despliegue y ejecución de programas utilizando contenedores.

Docker ha sido utilizado tanta para contenerizar las instancias de Node-RED para integrarlas al sistema como para contenerizar todos los componentes en el despliegue en la nube.

Figura 2.20: Logo de Docker. Fuente: Wikimedia Commons, 2017

2.21 Docker Desktop

Docker Desktop es una aplicación que proporciona una interfaz gráfica para gestionar contenedores y aplicaciones Docker en entornos locales de desarrollo. Está disponible para Windows y macOS y facilita la instalación y uso de Docker al ofrecer una experiencia de usuario simplificada para trabajar con contenedores.

En el proyecto ha sido utilizado para construir los contenedores Docker en local en el entorno de desarrollo del proyecto.

Figura 2.21: Logo de Docker Desktop. Fuente: AWS Community, 2024

2.22 Google Cloud

Google Cloud es un servicio de la nube de Google que ofrece diversas soluciones para computación, almacenamiento, análisis de datos y más. Facilita a las compañías y programadores crear, implementar y controlar aplicaciones y servicios utilizando la infraestructura de Google.

En este proyecto se va a utilizar Google Cloud como plataforma elegida para desplegar en la nube debido a su amplia cantidad de servicios y su intuitiva interfaz de usuario.

Figura 2.22: Logo de Google Cloud. Fuente: Wikimedia Commons, 2021

2.23 Google Cloud Run

Google Cloud Run se trata de una plataforma serverless que posibilita la ejecución de aplicaciones en la nube a través de contenedores sin la obligación de administrar la infraestructura subyacente. Con Google Cloud Run se puede ejecutar aplicaciones basadas en contenedores de manera fácil y escalable, pagando solo por el tiempo de ejecución real del código.

Para este sistema se utilizará Google Cloud Run como servicio para el despliegue, ya que cada componente irá en un contenedor Docker.

Figura 2.23: Logo de Google Cloud Run. Fuente: LinkedIn, 2023

3. Metodología

En el ámbito del desarrollo de software, es conocido por todos, la importancia que tiene usar una metodología. Sin embargo, a veces puede resultar difícil elegir el tipo de metodología a utilizar para su proyecto.

Se ha empleado una metodología ágil con desarrollo iterativo para llevar a cabo este Trabajo de Fin de Grado. En un desarrollo iterativo e incremental el proyecto se planifica en diversos bloques temporales llamados iteraciones. Las iteraciones son como proyectos más pequeños, donde se repite un proceso de trabajo similar en cada uno para obtener un resultado completo del producto final. En cada iteración, el producto se desarrolla en base a los avances obtenidos en las iteraciones previas.

Los pasos definidos para este proyecto son los siguientes:

1. **Formación en las tecnologías**: Durante esta fase inicial, se han analizado todas las tecnologías que se emplearán en la ejecución del proyecto. Se han utilizado las documentaciones oficiales y otros recursos web
2. **Captura y análisis de requisitos:** Durante esta etapa se han realizado la captura de requisitos y la especificación de los casos de uso
3. **Diseño y arquitectura**: Aquí se han tomado las decisiones del diseño del sistema y se han definido los diagramas de clases, de componentes y de navegación

Ahora empiezan a ejecutarse las 3 iteraciones. Que serían los pasos 4, 5 y 6. Cada iteración contiene los pasos de desarrollo, prueba y validación.

4. **Iteración 1**: Fase inicial del desarrollo, se crea el servidor, la API, se crea la base de datos y se hace la primera modificación sobre el código fuente de Node-RED y se integra al Backend usando Docker

5. **Iteración 2**: Se implementa toda la lógica del sistema de autenticación de usuarios y se desarrollan las primeras vistas en el Frontend
6. **Iteración 3**: Se implementan todas las funcionalidades de las clases y se añaden los roles de usuario y también se desarrollan las vistas de las clases
7. **Despliegue**: Se realizan las configuraciones para el despliegue en la nube del sistema
8. **Documentación**: Se desarrolla toda la documentación relacionada con el proyecto

4. Captura y Análisis de Requisitos

En este apartado, se detallarán los requisitos recogidos durante la fase de recolección y obtención de los mismos.

4.1 Requisitos

4.1.1 Requisitos funcionales

- Gestión de usuarios.
    - **RF-01.** Un usuario puede registrarse en el sistema con su nombre completo, correo electrónico, rol y contraseña
    - **RF-02.** Un usuario puede iniciar sesión con su correo electrónico y contraseña
    - **RF-03.** Un usuario puede cerrar sesión
    - **RF-04**. Un usuario puede ver sus datos de perfil
    - **RF-05**. Un usuario puede cambiar su correo electrónico
    - **RF-06.** Un usuario puede cambiar su nombre completo
    - **RF-07**. Un usuario puede cambiar su contraseña
    - **RF-08.** Un usuario puede restablecer su contraseña
    - **RF-09**. Un usuario puede eliminar su cuenta de usuario del sistema

- Gestión de Node-RED y flujos de trabajo.
    - **RF-10.** El usuario puede acceder a su instancia independiente de Node-RED
    - **RF-11.** El usuario puede guardar sus flujos de trabajo
    - **RF-12.** El usuario puede sobrescribir sus flujos de trabajo
    - **RF-13.** El usuario puede cargar sus flujos de trabajo

- **RF-14.** El usuario puede modificar el nombre de sus flujos de trabajo
- **RF-15.** El usuario puede eliminar sus flujos de trabajo
- **RF-16.** El usuario con el rol de profesor puede cargar los flujos de trabajo que los alumnos hayan subido a sus clases
- **RF-17.** El usuario con el rol de alumno puede cargar los flujos de trabajo que los profesores hayan subido a sus clases

- Gestión de clases
  - **RF-18.** El usuario con el rol de profesor puede crear una nueva clase
  - **RF-19.** El usuario con el rol de profesor puede ver sus clases
  - **RF-20.** El usuario con el rol de profesor puede acceder a sus clases
  - **RF-21.** El usuario con el rol de profesor puede eliminar sus clases
  - **RF-22**. El usuario con el rol de profesor puede modificar el nombre de sus clases
  - **RF-23.** El usuario con el rol de profesor puede invitar a alumnos a sus clases mediante correo electrónico
  - **RF-24.** El usuario con el rol de profesor puede ver los alumnos que hay en sus clases
  - **RF-25.** El usuario con el rol de profesor puede expulsar a los alumnos de sus clases
  - **RF-26.** El usuario con el rol de profesor puede publicar flujos de trabajo en sus clases
  - **RF-27**. El usuario con el rol de profesor puede cancelar la publicación de flujos de trabajo en sus clases
  - **RF-28**. El usuario con el rol de profesor puede ver los flujos de trabajo que han sido subidos a sus clases
  - **RF-29.** El usuario con el rol de profesor puede eliminar los flujos de trabajo que han sido subidos a sus clases
  - **RF-30**. El usuario con el rol de alumno puede unirse a una clase mediante un código de invitación
  - **RF-31.** El usuario con el rol de alumno puede publicar flujos de trabajo en las clases a las que se ha unido

- **RF-32**. El usuario con el rol de alumno puede cancelar la publicación de flujos de trabajo en las clases a las que se ha unido
- **RF-33**. El usuario con el rol de alumno puede ver los flujos de trabajo que los profesores han publicado en las clases a las que se ha unido
- **RF-34.** El usuario con el rol de alumno puede abandonar una clase

4.1.2 Requisitos no funcionales

- Seguridad
  - **RNF-01.** Solo el usuario tendrá la capacidad de actualizar sus propios datos personales.
  - **RNF-02.** El usuario es el único autorizado para modificar sus flujos de trabajo.
  - **RNF-03.** El profesor que ha creado una clase será el único capaz de modificarla
  - **RNF-04.** El profesor que ha creado una clase será el único capaz de acceder a los flujos de trabajos subidos por los alumnos
  - **RNF-05.** El profesor que ha creado una clase será el único capaz de eliminar flujos de trabajos subidos por los alumnos
  - **RNF-06.** El profesor que ha creado una clase será el único capaz de eliminar flujos de trabajos subidos por los alumnos
  - **RNF-07.** El profesor que ha creado una clase será el único capaz de expulsar alumnos de las clases
  - **RNF-08.** El alumno solo podrá ver los flujos de trabajo subidos por el profesor a una clase
  - **RNF-09.** El alumno solo podrá ver los flujos de trabajo subidos por el profesor a una clase
  - **RNF-10.** Será necesario estar registrado para poder acceder a Node-RED
  - **RNF-11.** Cada dirección de correo electrónico puede ser utilizada únicamente para una cuenta de usuario
  - **RNF-12.** Las comunicaciones deberán estar cifradas mediante el protocolo HTTPS
  - **RNF-13.** Cada usuario tendrá una instancia independiente de Node-RED

- Usabilidad
    - **RNF-14.** La interfaz de usuario ofrecerá una experiencia intuitiva
    - **RNF-15**. La interfaz de usuario deberá ser consistente con los colores

- Accesibilidad.
    - **RNF-16**. El sistema estará accesible para los usuarios desde cualquier sitio con acceso a internet
    - **RNF-17**. Los usuarios podrán acceder al sistema sin instalar nada en su dispositivo

4.2 Casos de uso

4.2.1: Registro

Identificador	RF-01
Descripción	Un usuario puede registrarse en el sistema con su nombre completo, correo electrónico, rol y contraseña
Precondición	El usuario no está registrado en el sistema
Postcondición	El usuario tiene la cuenta creada y almacenada en la base de datos
Secuencia principal	
1. El usuario pulsa el botón “Registrarse” 2. El sistema muestra la pantalla de registro 3. El usuario introduce su nombre 4. El usuario introduce sus apellidos 5. El usuario introduce su correo electrónico 6. El usuario introduce su contraseña 7. El usuario elije su rol de usuario 8. El usuario pulsa en el botón “Unirse” 9. El sistema comprueba que los datos son correctos 10. El sistema almacena la cuenta en la base de datos 11. El sistema muestra un mensaje indicando que la cuenta se ha creado con éxito	

12. El sistema muestra la página de inicio de sesión
Secuencias alternativas
9.b0 El sistema comprueba que el correo electrónico ya está en la base de datos 9.b1 El sistema muestra un mensaje informando que el correo electrónico no es correcto
9.c0 El sistema comprueba que quedan campos sin completar 9.c1 El sistema muestra un mensaje indicando que quedan campos sin completar
9.e0 El sistema comprueba que la contraseña introducida tiene menos de 8 caracteres 9.e1 El sistema muestra un mensaje indicando que la contraseña debe tener 8 caracteres o más

Tabla 4.2.1: CU RF-01

4.2.2: Inicio de sesión

Identificador	RF-02
Descripción	Un usuario puede iniciar sesión con su correo electrónico y contraseña
Precondición	El usuario accede a la aplicación y ha pulsado el botón de iniciar sesión
Postcondición	El usuario accede a su cuenta
Secuencia principal	
1. El usuario pulsa el botón “Iniciar sesión” 2. El sistema muestra la pantalla de inicio de sesión 3. El usuario introduce su correo electrónico 4. El usuario introduce su contraseña 5. El usuario pulsa el botón “Iniciar sesión” 6. El sistema comprueba que hay un usuario en la base de datos con esos datos 7. El sistema muestra la pantalla de inicio de la aplicación con los permisos de usuario registrado	
Secuencias alternativas	
6.b0 El sistema comprueba que no existe una cuenta con ese correo electrónico	

6.b1 El sistema muestra un mensaje indicando que no existe ningún usuario con esos datos
6.c0 El sistema comprueba que la contraseña es incorrecta 6.c1 El sistema muestra un mensaje indicando que la contraseña es incorrecta

Tabla 4.2.2: CU RF-02

4.2.3: Cerrar sesión

Identificador	RF-03
Descripción	Un usuario puede cerrar sesión
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario accede a la pantalla principal de la aplicación sin estar autenticado
Secuencia principal	
1. El usuario pulsa el botón “Cerrar sesión” 2. El sistema cierra sesión y lo redirige a la pantalla principal de la aplicación	

Tabla 4.2.3: CU RF-03

4.2.4: Ver perfil

Identificador	RF-04
Descripción	Un usuario puede ver sus datos de perfil
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario accede a la pantalla donde se pueden ver los datos del usuario
Secuencia principal	
1. El usuario pulsa el botón “Ver perfil” 2. El sistema muestra la pantalla donde se muestran los datos del usuario	

Tabla 4.2.4: CU RF-04

4.2.5: Cambiar correo electrónico

Identificador	RF-05

Descripción	Un usuario puede cambiar su correo electrónico
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha cambiado su correo electrónico
Secuencia principal	
1. El usuario pulsa el botón “Ver perfil” 2. El sistema muestra la pantalla donde se pueden ver los datos del usuario 3. El usuario sustituye el texto del correo electrónico 4. El usuario pulsa el botón "Guardar cambios" 5. El sistema muestra la pantalla del perfil donde se pueden ver los datos del usuario	
Secuencias alternativas	
8.b0 El usuario introduce un correo electrónico que ya está registrado 8.b1 El sistema muestra un mensaje indicando que el correo electrónico ya está registrado	

Tabla 4.2.5: CU RF-05

4.2.6: Cambiar nombre

Identificador	RF-06
Descripción	Un usuario puede cambiar su nombre completo
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha cambiado su nombre completo
Secuencia principal	
1. El usuario pulsa el botón “Ver perfil” 2. El sistema muestra la pantalla donde se muestran los datos del usuario 3. El usuario sustituye el texto del nombre completo 4. El usuario pulsa el botón "Guardar cambios" 5. El sistema muestra la pantalla del perfil donde se pueden ver los datos del usuario	

Tabla 4.2.6: CU RF-06

4.2.7: Cambiar contraseña

Identificador	RF-07
Descripción	Un usuario puede cambiar su contraseña
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha cambiado su contraseña
Secuencia principal	
6. El usuario pulsa el botón “Ver perfil” 7. El sistema muestra la pantalla donde se muestra un formulario para introducir cambiar la contraseña 8. El usuario introduce una nueva contraseña 9. El usuario pulsa el botón "Guardar cambios" 10. El sistema muestra un mensaje indicando que la contraseña se ha cambiado correctamente	
Secuencias alternativas	
8.b0 El usuario introduce una contraseña con menos de 8 caracteres 8.b1 El sistema muestra un mensaje indicando que no se ha podido cambiar la contraseña	

Tabla 4.2.7: CU RF-07

4.2.8: Reestablecer contraseña

Identificador	RF-08
Descripción	Un usuario puede reestablecer su contraseña
Precondición	El usuario se ha registrado
Postcondición	La contraseña del usuario se ha modificado en la base de datos
Secuencia principal	
1. El usuario pulsa el botón “No recuerdo contraseña” 2. El sistema muestra la pantalla donde se muestra un formulario para introducir el correo electrónico de recuperación 3. El usuario introduce el correo electrónico 4. El usuario pulsa el botón "Enviar"	

5. El sistema muestra un mensaje indicando que se ha enviado el correo electrónico de recuperación 6. El usuario accede a su correo electrónico y pulsa en el enlace de recuperación 7. El sistema muestra la pantalla con un formulario para introducir la nueva contraseña 8. El usuario introduce una contraseña 9. El sistema muestra un mensaje confirmando que se ha cambiado la contraseña 10. El sistema redirige al usuario a la página de inicio de sesión
Secuencias alternativas
3.b0 El usuario introduce un correo electrónico que no está registrado 3.b1 El sistema indica que el correo electrónico no existe
6.b0 El usuario no hace click en el enlace dos horas después de recibirlo 6.b1 El sistema muestra la pantalla con un formulario para introducir la nueva contraseña 6.b2 El usuario introduce una contraseña 6.b3 El sistema muestra un mensaje indicando que el tiempo para restablecer la contraseña ha expirado 6.b4 El sistema redirige al usuario a la página de inicio de sesión

Tabla 4.2.8: CU RF-08

4.2.9: Eliminar cuenta

Identificador	RF-09
Descripción	Un usuario puede eliminar su cuenta de usuario del sistema
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha eliminado su cuenta del sistema
Secuencia principal	
1. El usuario pulsa el botón “Ver perfil” 2. El sistema muestra la pantalla donde se pueden ver los datos del usuario 3. El usuario pulsa el botón "Eliminar cuenta" 4. El sistema muestra una ventana emergente para confirmar la acción	

5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra la pantalla inicial de la aplicación
Secuencias alternativas
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema cierra la ventana emergente

Tabla 4.2.09: CU RF-09

4.2.10: Acceder a Node-RED

Identificador	RF-10
Descripción	Un usuario puede acceder a su instancia independiente de Node-RED
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha accedido a Node-RED
Secuencia principal	
1. El usuario pulsa el botón “Acceder a Node-RED” 2. El sistema muestra una animación de carga mientras se configura la instancia de Node-RED 3. El sistema redirige al usuario a su instancia de Node-RED	

Tabla 4.2.10 CU RF-10

4.2.11: Guardar flujo

Identificador	RF-11
Descripción	Un usuario puede guardar sus flujos de trabajo
Precondición	El usuario ha accedido a Node-RED
Postcondición	El usuario ha guardado un flujo de trabajo en la base de datos
Secuencia principal	
1. El usuario crea un flujo de trabajo en Node-RED 2. El usuario pulsa el boton "Guardar flujos" 3. El sistema muestra la lista de flujos de trabajo del usuario en Node-RED 4. El usuario pulsa en el flujo de trabajo que quiere guardar	

5. El sistema muestra un mensaje indicando que se ha guardado el flujo de trabajo

Tabla 4.2.11: CU RF-11

4.2.12: Sobrescribir flujo

Identificador	RF-11
Descripción	Un usuario puede sobrescribir sus flujos de trabajo
Precondición	El usuario ha accedido a Node-RED
Postcondición	El usuario ha sobrescrito un flujo de trabajo en la base de datos
Secuencia principal	
1. El usuario crea un flujo de trabajo en Node-RED 2. El usuario pulsa el boton "Guardar flujos" 3. El sistema muestra la lista de flujos de trabajo del usuario en Node-RED 4. El usuario pulsa en el flujo de trabajo que quiere guardar 5. El sistema muestra un mensaje de confirmación indicando que ese flujo ya está guardado en la base de datos y que el flujo se va a sobrescribir si continúa 6. El usuario pulsa el botón "Aceptar" 7. El sistema muestra un mensaje indicando que se ha actualizado el flujo de trabajo	
Secuencias alternativas	
6.b0 El usuario pulsa el botón "Cancelar" 6.b1 El sistema muestra otra vez la lista de flujos de trabajo	

Tabla 4.2.12: CU RF-12

4.2.13: Cargar flujo

Identificador	RF-13
Descripción	Un usuario puede cargar sus flujos de trabajo
Precondición	El usuario ha accedido a Node-RED
Postcondición	El usuario ha cargado un flujo de trabajo en Node-RED
Secuencia principal	

1. El usuario pulsa el botón "Cargar flujo" 2. El sistema muestra la lista de flujos de trabajo que el usuario tiene guardado en su base de datos 3. El usuario pulsa sobre el flujo de trabajo que quiere cargar 4. El sistema muestra el flujo de trabajo en la interfaz de Node-RED

Tabla 4.2.13: CU RF-13

4.2.14: Cambiar nombre de un flujo de trabajo

Identificador	RF-14
Descripción	Un usuario puede cambiar el nombre de sus flujos de trabajo
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha cambiado el nombre de un flujo de trabajo en la base de datos
Secuencia principal	
1. El usuario pulsa el botón "Editar flujos" 2. El sistema muestra la lista de flujos de trabajo que el usuario tiene guardado en su base de datos 3. El usuario pulsa sobre el botón "Cambiar nombre" sobre el flujo de trabajo que quiera modificar 4. El sistema muestra un formulario para ingresar el nuevo nombre del flujo de trabajo 5. El usuario introduce el nuevo nombre 6. El usuario pulsa el botón "Guardar cambios" 7. El sistema muestra un mensaje indicando que el nombre del flujo se ha modificado correctamente 8. El sistema redirige al usuario a la pantalla de editar flujos	

Tabla 4.2.14: CU RF-14

4.2.15 Eliminar un flujo de trabajo

Identificador	RF-15

Descripción	Un usuario puede eliminar sus flujos de trabajo
Precondición	El usuario ha iniciado sesión
Postcondición	El usuario ha eliminado un flujo de trabajo en la base de datos
Secuencia principal	
1. El usuario pulsa el botón "Editar flujos" 2. El sistema muestra la lista de flujos de trabajo que el usuario tiene guardado en su base de datos 3. El usuario pulsa sobre el botón "Eliminar" sobre el flujo de trabajo que quiera eliminar 4. El sistema muestra un mensaje de confirmación 5. El usuario pulsa sobre el botón "Eliminar" 6. El sistema muestra un mensaje indicando que el flujo se ha eliminado correctamente 7. El sistema muestra la pantalla de editar flujos	
Secuencias alternativas	
5.b0 El usuario pulsa sobre el botón "Cancelar" 5.b1 El sistema muestra la pantalla de editar flujos	

Tabla 4.2.15: CU RF-15

4.2.16: Cargar flujos de un alumno

Identificador	RF-16
Descripción	Un usuario con el rol de profesor puede cargar los flujos de trabajo que los alumnos hayan subido a sus clases
Precondición	El usuario ha accedido a Node-RED y tiene el rol de profesor
Postcondición	El usuario ha cargado un flujo de trabajo en Node-RED
Secuencia principal	
1. El usuario pulsa el botón “mis clases” 2. El sistema muestra una lista con todas las clases creadas por el usuario	

3. El usuario pulsa sobre la clase a la que quiere acceder 4. El sistema muestra una lista con los flujos de trabajo que hay en la clase 5. El usuario pulsa sobre el flujo de trabajo que quiere cargar 6. El sistema muestra el flujo de trabajo en la interfaz de Node-RED

Tabla 4.2.16: CU RF-13

4.2.17: Cargar flujos de un profesor

Identificador	RF-17
Descripción	Un usuario con el rol de alumno puede cargar los flujos de trabajo que los profesores hayan subido a sus clases.
Precondición	El usuario ha accedido a Node-RED y tiene el rol de alumno
Postcondición	El usuario ha cargado un flujo de trabajo en Node-RED
Secuencia principal	
1. El usuario pulsa el botón “mis clases” 2. El sistema muestra una lista con todas las clases a las que se ha unido el usuario 3. El usuario pulsa sobre la clase a la que quiere acceder 4. El sistema muestra una lista con los flujos de trabajo que el profesor ha subido en la clase 5. El usuario pulsa sobre el flujo de trabajo que quiere cargar 6. El sistema muestra el flujo de trabajo en la interfaz de Node-RED	

Tabla 4.2.17: CU RF-17

4.2.18: Crear clase

Identificador	RF-18
Descripción	Un usuario con el rol de profesor puede crear una nueva clase
Precondición	El usuario ha iniciado sesión y tiene el rol de profesor
Postcondición	El usuario ha creado una nueva clase
Secuencia principal	

1. El usuario pulsa el botón “Nueva clase” 2. El sistema muestra un formulario para introducir el nombre de la clase 3. El usuario introduce el nombre de la clase 4. El usuario pulsa el botón "Crear" 5. El sistema comprueba que el nombre de la clase es correcto 6. El sistema muestra un mensaje de clase creada con éxito 7. El sistema muestra la página anterior
Secuencias alternativas
5.b0 El sistema muestra un mensaje indicando que ya existe una clase con ese nombre 5.b1 El sistema muestra de nuevo el formulario para crear una clase

Tabla 4.2.18: CU RF-18

4.2.19: Ver clases

Identificador	RF-19
Descripción	Un usuario con el rol de profesor puede ver sus clases
Precondición	El usuario tiene alguna clase creada
Postcondición	El usuario puede ver la lista con todas sus clases creadas
Secuencia principal	
1. El usuario pulsa el botón “Mis clases” 2. El sistema muestra una lista con todas las clases creadas por el usuario	

Tabla 4.2.19: CU RF-19

4.2.20: Acceder a una clase

Identificador	RF-20
Descripción	Un usuario con el rol de profesor puede acceder a sus clases
Precondición	El usuario tiene alguna clase creada
Postcondición	El usuario ha accedido a una clase
Secuencia principal	
3. El usuario pulsa el botón “Mis clases”	

4. El sistema muestra una lista con las clases creadas por el usuario 5. El usuario pulsa el botón "Acceder" sobre la clase a la que quiera acceder 6. El sistema muestra una pantalla con la vista de la clase

Tabla 4.2.20: CU RF-20

4.2.21: Eliminar una clase

Identificador	RF-21
Descripción	Un usuario con el rol de profesor puede eliminar sus clases
Precondición	El usuario tiene alguna clase creada
Postcondición	El usuario ha eliminado una clase
Secuencia principal	
1. El usuario pulsa el botón “Mis clases” 2. El sistema muestra una lista con las clases creadas por el usuario 3. El usuario pulsa el botón "Eliminar clase" sobre la clase que quiera eliminar 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema indica que la clase se ha eliminado correctamente 7. El sistema muestra una lista con las clases creadas por el usuario	
Secuencias alternativas	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra una lista con las clases creadas por el usuario	

Tabla 4.2.21: CU RF-21

4.2.22: Modificar clase

Identificador	RF-22
Descripción	Un usuario con el rol de profesor puede modificar el nombre de sus clases
Precondición	El usuario ha accedido a una clase creada
Postcondición	El usuario ha modificado el nombre de una clase
Secuencia principal	

1. El usuario pulsa el botón “Mis clases” 2. El sistema muestra una lista con las clases creadas por el usuario 3. El usuario pulsa el botón "Acceder" sobre la clase a la que quiera acceder 4. El sistema muestra una pantalla con la vista de la clase 5. El usuario pulsa el botón "Modificar" 6. El sistema muestra un formulario para introducir un nuevo nombre 7. El usuario pulsa en el botón "Guardar cambios" 8. El sistema muestra un mensaje indicando que el nombre de la clase se ha modificado correctamente

Tabla 4.2.22: CU RF-22

4.2.23: Invitar alumnos

Identificador	RF-23
Descripción	Un usuario con el rol de profesor puede invitar a alumnos a sus clases mediante correo electrónico
Precondición	El usuario ha accedido a una clase creada
Postcondición	El usuario ha enviado un correo electrónico para invitar a sus alumnos
Secuencia principal	
1. El usuario pulsa el botón “Invitar” 2. El sistema muestra un formulario donde se pueden introducir los correos electrónicos de los usuarios que se quiera invitar 3. El usuario pulsa el botón introduce los correos electrónicos 4. El usuario pulsa el botón invitar 5. El sistema indica que las invitaciones se han enviado correctamente 6. El sistema redirige al usuario a la pantalla de clases	

Tabla 4.2.23: CU RF-23

4.2.24 Ver participantes

Identificador	RF-24

Descripción	Un usuario con el rol de profesor puede ver los alumnos que hay en sus clases.
Precondición	El usuario ha accedido a una clase creada
Postcondición	El usuario ve la lista de los participantes de la clase
Secuencia principal	
1. El usuario pulsa el botón “ver participantes” 2. El sistema muestra una lista con los usuarios que se han unido a la clase	

Tabla 4.2.24: CU RF-24

4.2.25: Expulsar a un alumno

Identificador	RF-25
Descripción	Un usuario con el rol de profesor puede expulsar a los alumnos de sus clases
Precondición	El usuario ha accedido a una clase creada que tiene alumnos
Postcondición	El usuario ha eliminado a un alumno de su clase
Secuencia principal	
1. El usuario pulsa el botón “ver participantes” 2. El sistema muestra una lista con los usuarios que se han unido a la clase 3. El usuario pulsa el botón "Expulsar" 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Expulsar" 6. El sistema muestra una lista con los usuarios que se han unido a la clase, menos con el usuario expulsado	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra una lista con los usuarios que se han unido a la clase, menos con el usuario expulsado	

Tabla 4.2.25: CU RF-25

4.2.26: Publicar flujos

Identificador	RF-26

Descripción	Un usuario con el rol de profesor puede publicar flujos de trabajo en sus clases
Precondición	El usuario ha accedido a una clase
Postcondición	El usuario ha publicado un flujo de trabajo en su clase
Secuencia principal	
1. El usuario pulsa el botón “Publicar flujo” 2. El sistema muestra una lista con los flujos de trabajo del usuario en la base de datos 3. El usuario pulsa el botón "Entregar" sobre el flujo de trabajo que quiera publicar 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra un mensaje indicando que el flujo de ha publicado correctamente 7. El sistema muestra la pantalla con la lista de flujos de trabajo	
Secuencias alternativas	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra la pantalla con la lista de flujos de trabajo	

Tabla 4.2.26: CU RF-26

4.2.27: Cancelar publicación de flujos

Identificador	RF-27
Descripción	Un usuario con el rol de profesor puede publicar flujos de trabajo en sus clases
Precondición	El usuario ha publicado un flujo de trabajo en una clase
Postcondición	El usuario ha cancelado la publicación del flujo de trabajo en su clase
Secuencia principal	
1. El usuario pulsa el botón “Publicar flujo” 2. El sistema muestra una lista con los flujos de trabajo del usuario en la base de datos	

3. El usuario pulsa el botón "Cancelar publicación" sobre el flujo de trabajo publicado del que quiera cancelar su publicación 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra un mensaje indicando que la publicación del flujo ha sido cancelada correctamente 7. El sistema muestra la pantalla con la lista de flujos de trabajo
Secuencias alternativas
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra la pantalla con la lista de flujos de trabajo

Tabla 4.2.27: CU RF-27

4.2.28: Ver flujos clase

Identificador	RF-28
Descripción	Un usuario con el rol de profesor puede ver los flujos de trabajo que han sido subidos a sus clases
Precondición	El usuario ha accedido a una clase que tiene flujos de trabajos ya publicados
Postcondición	El usuario puede ver una lista con los flujos de trabajo subidos a sus clases
Secuencia principal	
1. El usuario pulsa el botón “Ver flujos” 2. El sistema muestra una lista con los flujos de trabajo que han sido publicados en la clase	

Tabla 4.2.28: CU RF-28

4.2.29: Eliminar flujo

Identificador	RF-29
Descripción	Un usuario con el rol de profesor puede eliminar los flujos de trabajo que han sido subidos a sus clases

Precondición	El usuario ha accedido a una clase que tiene flujos de trabajos ya publicados
Postcondición	Se ha eliminado el flujo de trabajo de la clase
Secuencia principal	
1. El usuario pulsa el botón “Ver flujos” 2. El sistema muestra una lista con los flujos de trabajo que han sido publicados en la clase 3. El usuario pulsa el botón "Eliminar" 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Eliminar" 6. El sistema muestra un mensaje indicando que se ha eliminado correctamente 7. El sistema muestra la lista de los flujos de trabajos publicados	
Secuencias alternativas	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra la lista de flujos de trabajo publicados	

Tabla 4.2.29: CU RF-29

4.2.30: Unirse a una clase

Identificador	RF-30
Descripción	Un usuario con el rol de alumno puede unirse a una clase mediante un código de invitación.
Precondición	El usuario tiene un código de invitación
Postcondición	El usuario se ha unido a una clase
Secuencia principal	
1. El usuario pulsa el botón “Unirse” 2. El sistema muestra una ventana emergente para que introduzcas el código de invitación 3. El usuario introduce el código de invitación 4. El usuario pulsa el botón "Aceptar" 5. El sistema comprueba que el enlace de invitación es correcto	

6. El sistema muestra un mensaje indicando que el usuario se ha unido a la clase exitosamente
Secuencias alternativas
5.b0 El sistema comprueba que el enlace de invitación es incorrecto 5.b1 El sistema muestra un mensaje indicando que ha habido un error al unirse a la clase

Tabla 4.2.30: CU RF-30

4.2.31 Entregar flujo

Identificador	RF-31
Descripción	Un usuario con el rol de alumno puede publicar flujos de trabajo en las clases a las que se ha unido
Precondición	El usuario ha accedido a una clase
Postcondición	El flujo de trabajo ha sido publicado en la clase
Secuencia principal	
1. El usuario pulsa el botón “Subir flujo” 2. El sistema muestra una lista con los flujos de trabajo que el usuario tiene almacenados en la base de datos 3. El usuario pulsa el botón "Entregar" 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra un mensaje indicando que se ha entregado correctamente 7. El sistema muestra una lista con los flujos de trabajo que el usuario tiene almacenados en la base de datos	
Secuencias alternativas	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra una lista con los flujos de trabajo que el usuario tiene almacenados en la base de datos	

Tabla 4.2.31: CU RF-31

4.2.32: Cancelar publicación de flujos

Identificador	RF-32
Descripción	Un usuario con el rol de alumno puede publicar flujos de trabajo en las clases a las que se ha unido
Precondición	El usuario ha publicado un flujo de trabajo en una clase
Postcondición	El usuario ha cancelado la publicación del flujo de trabajo en su clase
Secuencia principal	
1. El usuario pulsa el botón “Publicar flujo” 2. El sistema muestra una lista con los flujos de trabajo del usuario en la base de datos 3. El usuario pulsa el botón "Cancelar publicación" sobre el flujo de trabajo publicado del que quiera cancelar su publicación 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra un mensaje indicando que la publicación del flujo ha sido cancelada correctamente 7. El sistema muestra la pantalla con la lista de flujos de trabajo	
Secuencias alternativas	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra la pantalla con la lista de flujos de trabajo	

Tabla 4.2.32: CU RF-32

4.2.33: Ver flujos profesor

Identificador	RF-33
Descripción	Un usuario con el rol de alumno puede ver los flujos de trabajo que los profesores han publicado en las clases a las que se ha unido
Precondición	El usuario ha accedido a una clase
Postcondición	El usuario puede ver la lista de los flujos de trabajo publicados por el profesor en la clase

Secuencia principal
1. El usuario pulsa el botón “Flujos” 2. El sistema muestra una lista con los flujos de trabajo que el profesor ha publicado en la clase

Tabla 4.2.33: CU RF-33

4.2.34: Abandonar clase

Identificador	RF-34
Descripción	Un usuario con el rol de alumno puede abandonar una clase
Precondición	El usuario se ha unido a varias clases
Postcondición	El usuario ha abandonado la clase
Secuencia principal	
1. El usuario pulsa el botón “Mis clases” 2. El sistema muestra una lista con las clases a las que se ha unido el usuario 3. El usuario pulsa el botón "Abandonar" 4. El sistema muestra una ventana emergente para confirmar la acción 5. El usuario pulsa el botón "Confirmar" 6. El sistema muestra la lista de las clases, pero sin la clase eliminada	
5.b0 El usuario pulsa el botón "Cancelar" 5.b1 El sistema muestra la lista de las clases	

Tabla 4.2.34: CU RF-34

5. Diseño y Arquitectura

5.1 Arquitectura del sistema

5.1.1 Descripción general

La arquitectura del sistema es un punto muy importante en cualquier proyecto de software, ya que define la estructura y el diseño general del sistema, asegurando que los componentes interactúen correctamente. El sistema consta de 5 componentes diferenciados, los 5 componentes son:

- **Backend del sistema**: Se encarga de la lógica del negocio y gestión de las solicitudes del cliente, conectando los demás componentes con la base de datos
- **Frontend del sistema**: Se ocupa de la interfaz de usuario y mostrar la información al usuario haciendo peticiones al Backend
- **Base de datos**: Se encarga de almacenar todos los datos utilizados en el sistema
- **Backend de Node-RED**: Gestiona la lógica de los flujos de trabajo que se utilizan en Node-RED
- **Frontend de Node-RED**: Proporciona la interfaz gráfica para que los usuarios puedan trabajar con los flujos de trabajo y hace peticiones al Backend del sistema para guardar y cargar flujos de trabajo

Estos 5 componentes funcionan correctamente tanto de manera independiente como integrados entre sí.

Por lo tanto, es importante tener claro antes de empezar a desarrollar para implementar los requisitos funcionales que necesita cada componente para poder comunicarse con los demás y seguir manteniendo su funcionalidad. Por ejemplo, para implementar la funcionalidad de que un alumno pueda cargar los flujos de trabajo que ha subido el profesor a una clase necesita tener el JWT del alumno para poder hacer la petición al Backend. Todas estas cuestiones de diseño son importantes que se tengan en cuenta porque si se comienza a desarrollar puede que luego volver a atrás para solucionar el fallo sea muy complicado.

Otra parte importante del diseño es planificar como se van a usar las instancias independientes de Node-RED, ya que se debe garantizar que para cada usuario hay una instancia aislada. Para ello, se ha contenerizado la aplicación de Node-RED con las modificaciones realizadas utilizando Docker y cuando un usuario quiera acceder a su instancia independiente, el Backend lanzará el Docker identificado con el ID del usuario y añadiéndole como variable de entorno el JWT del usuario. Es decir, se lanzará un contenedor diferente por cada usuario que quiera usar Node-RED. Una vez que el usuario deje de usar la herramienta, se detendrá dicho contenedor.

Por último, no solo deben poder comunicarse entre ellos, sino que hay que tener en cuenta de que todo el sistema debe desplegarse y funcionar en la nube

5.1.2 Diagrama de arquitectura

Aquí se muestra el diagrama de arquitectura. Se puede ver como interactúan los componentes entre sí.

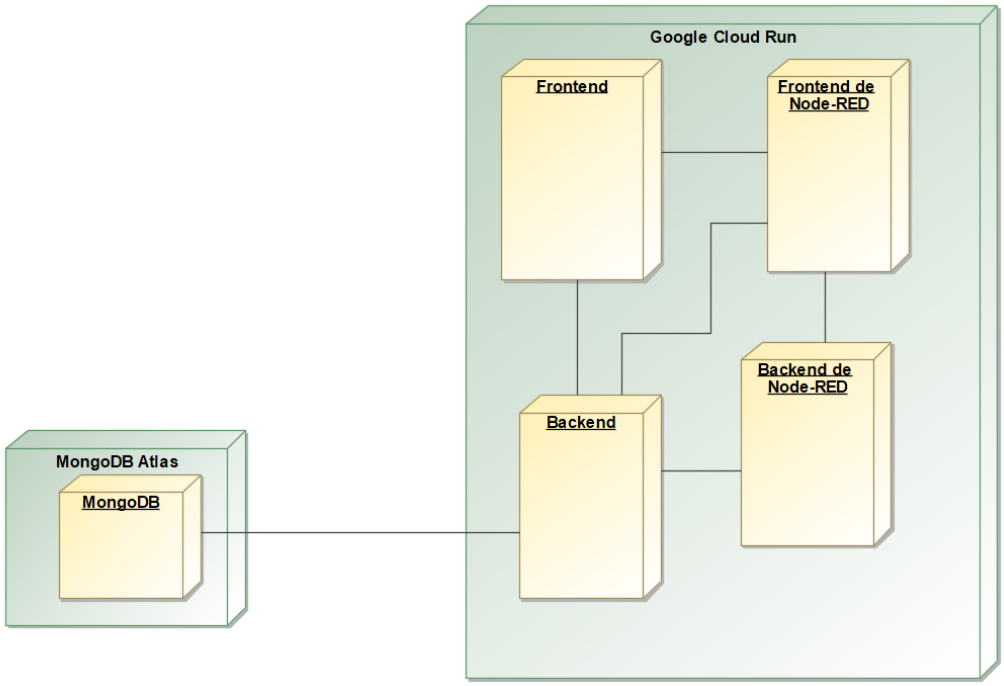


Figura 5.1.2.1: Diagrama de Arquitectura

En la figura se observa un esquema que representa las relaciones existentes entre los elementos del sistema.

Se va a explicar una a una las interacciones que ocurren en el sistema:

- **Interacción Backend - MongoDB**: El Backend será el componente encargado de gestionar todas las peticiones de lectura y escritura con la base de datos
- **Interacción Backend - Frontend**: El Frontend utilizará la API del Backend para hacer todas las peticiones relacionadas con los usuarios y las clases
- **Interacción Backend - Frontend de Node-RED**: El Frontend utilizará la API del Backend para hacer todas las peticiones relacionadas con los flujos de trabajo
- **Interacción Backend - Backend de Node-RED**: Cuando el Backend lance el contenedor de Node-RED, se le pasará como variable de entorno el JWT del usuario, que el Backend de Node-RED utilizará
- **Interacción Backend de Node-RED** - Frontend de Node-RED: El Frontend utilizará la API del Backend de Node-RED para realizar todas las peticiones con el funcionamiento de Node-RED y de obtener el JWT obtenido como variable de entorno

- **Interacción Frontend - Frontend de Node-RED**: Esta interacción ocurre cuando el usuario quiera navegar desde una página a otra

5.2 Modelo de la base de datos

Para la base de datos se va a utilizar MongoDB Atlas, que es una base de datos no relacional.

Al ser una BD no relacional, MongoDB no tiene clases, sino colecciones, ni tampoco tiene relaciones, sino referencias.

A pesar de ello, se ha realizado un diagrama de clases donde se sustituyen las colecciones por clases y las referencias por relaciones, con el objetivo de poder ver cómo funciona la base de datos de forma visual.

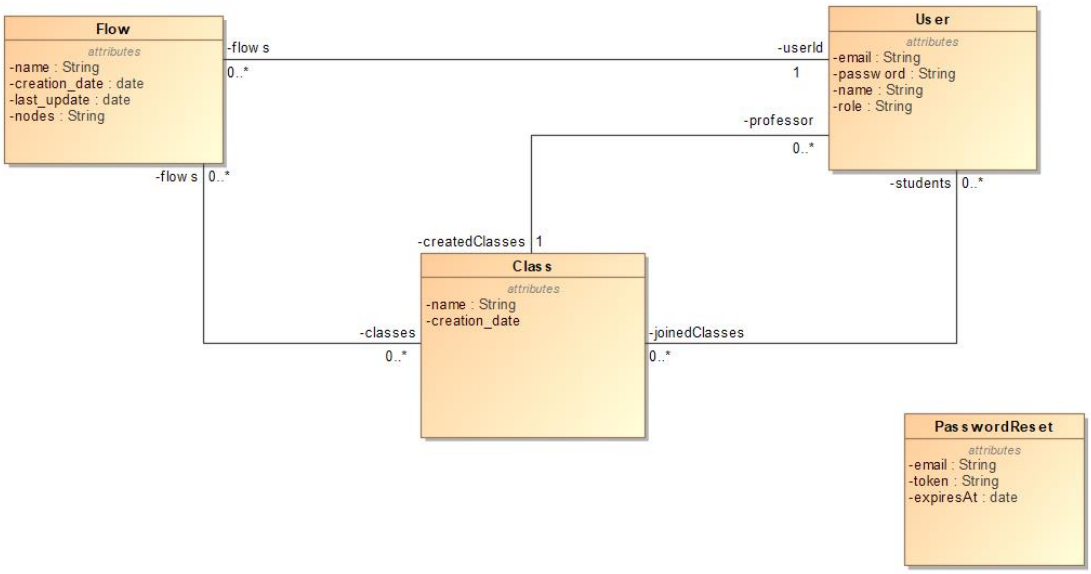


Figura 5.2.1: Diagrama de Clases

5.3 Backend de la aplicación

En este componente se va a desarrollar toda la lógica de negocio de la aplicación gracias a una API Rest, toda la lógica de creación de contenedores para acceder a Node-RED y toda la lógica de usuario. Además, gracias a la API Rest, servirá como enlace entre varios componentes del sistema.

5.3.1 Estructura del backend

El Backend está organizado siguiendo las buenas prácticas sacadas del artículo https://www.freecodecamp.org/news/rest-api-design-best-practices-build-a-rest-api/

Es importante que el Backend tenga una estructura que permita la independencia de las responsabilidades en los métodos definidos.

Para ello, se ha estructurado el Backend de la siguiente forma:

- **Rutas:** Son los endpoint que usará el cliente para hacer las peticiones.
- **Modelo**: Aquí se definen las colecciones de la base de datos.
- **Datos:** Aquí se utilizarán los métodos que interactúen directamente con la base de datos.
- **Servicio:** El servicio se encargará de la lógica de negocio.
- **Controlador:** Manejan las peticiones del cliente y sirve de Intermediario entre las rutas y el servicio

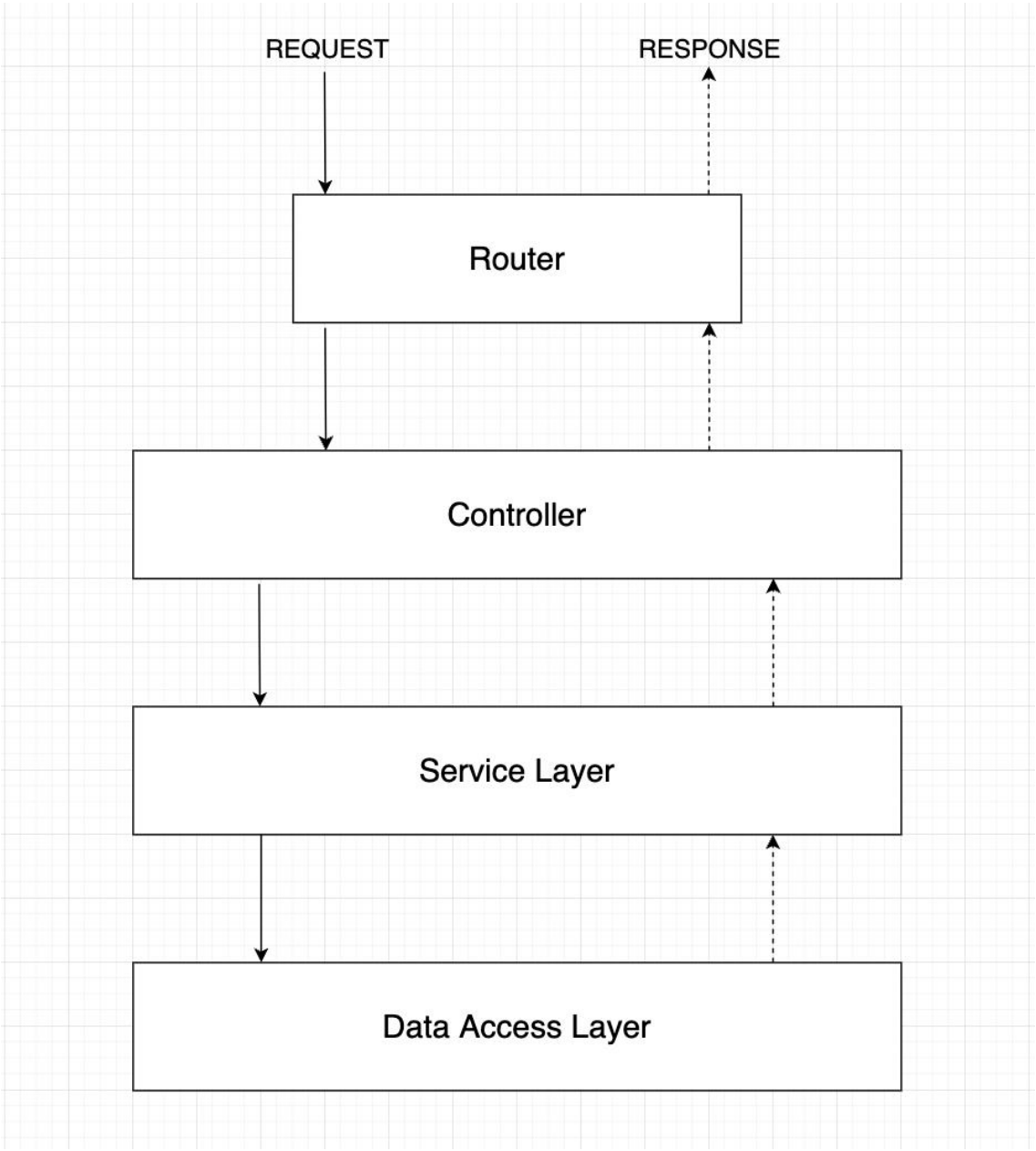


Figura 5.3.1.1: Arquitectura del Backend. Fuente: FreeCodeCamp, 2022

5.3.2 API del backend

La API Rest está dividida en cuatro partes:

- **Flujos de trabajo**: Aquí se encuentran todos los endpoints necesarios para los requisitos funcionales relacionados con los flujos de trabajo
- **Usuarios**: Aquí se encuentran todos los endpoints necesarios para los requisitos funcionales relacionados con usuarios

- **Clases:** Aquí se encuentran todos los endpoints necesarios para los requisitos funcionales relacionados con las clases
- **Node-RED**: Aquí se encuentran las rutas para lanzar o detener una instancia de Node-RED
- **Reset Password:** Aquí están las rutas que hacen el proceso de recuperación de contraseña
- **Email:** Las rutas relacionadas con enviar un correo electrónico

Para documentar la API Rest del sistema, se ha utilizado Swagger:

Figura 5.3.2.1: API de Usuarios

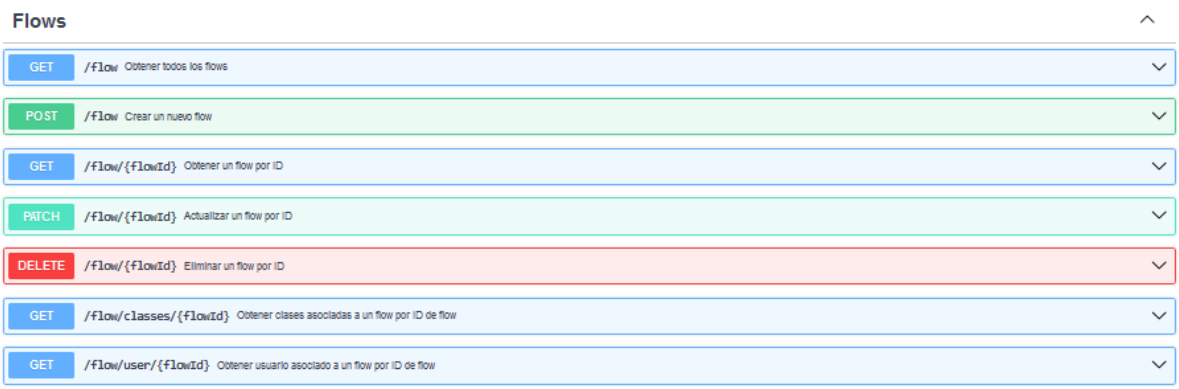


Figura 5.3.2.2: API de Flows

Figura 5.3.2.3: API de Clases

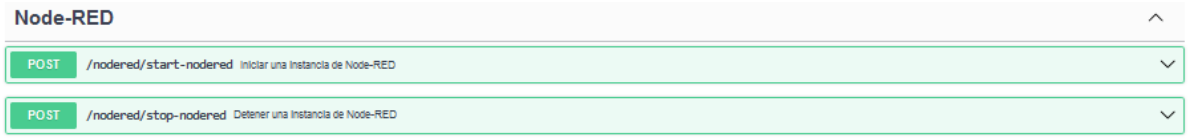


Figura 5.3.2.4: API de Node-RED

Figura 5.3.2.5: API de Reset Password

Figura 5.3.2.6: API de Email

5.3 Frontend del sistema

El sistema cuenta con dos roles de usuario diferentes, uno es el de profesor y otro el de alumno, aunque ambos comparten funcionalidades también tienen otras exclusivas para ese rol y por lo tanto también tienen vistas diferentes

A continuación, se van a mostrar los diagramas de navegación para cada rol de usuario. Las figuras rectangulares de color blanco representan las vistas comunes para ambos roles, mientras que las de color azul representan las individuales para ese rol.

Diagrama de navegación. Rol de profesor:

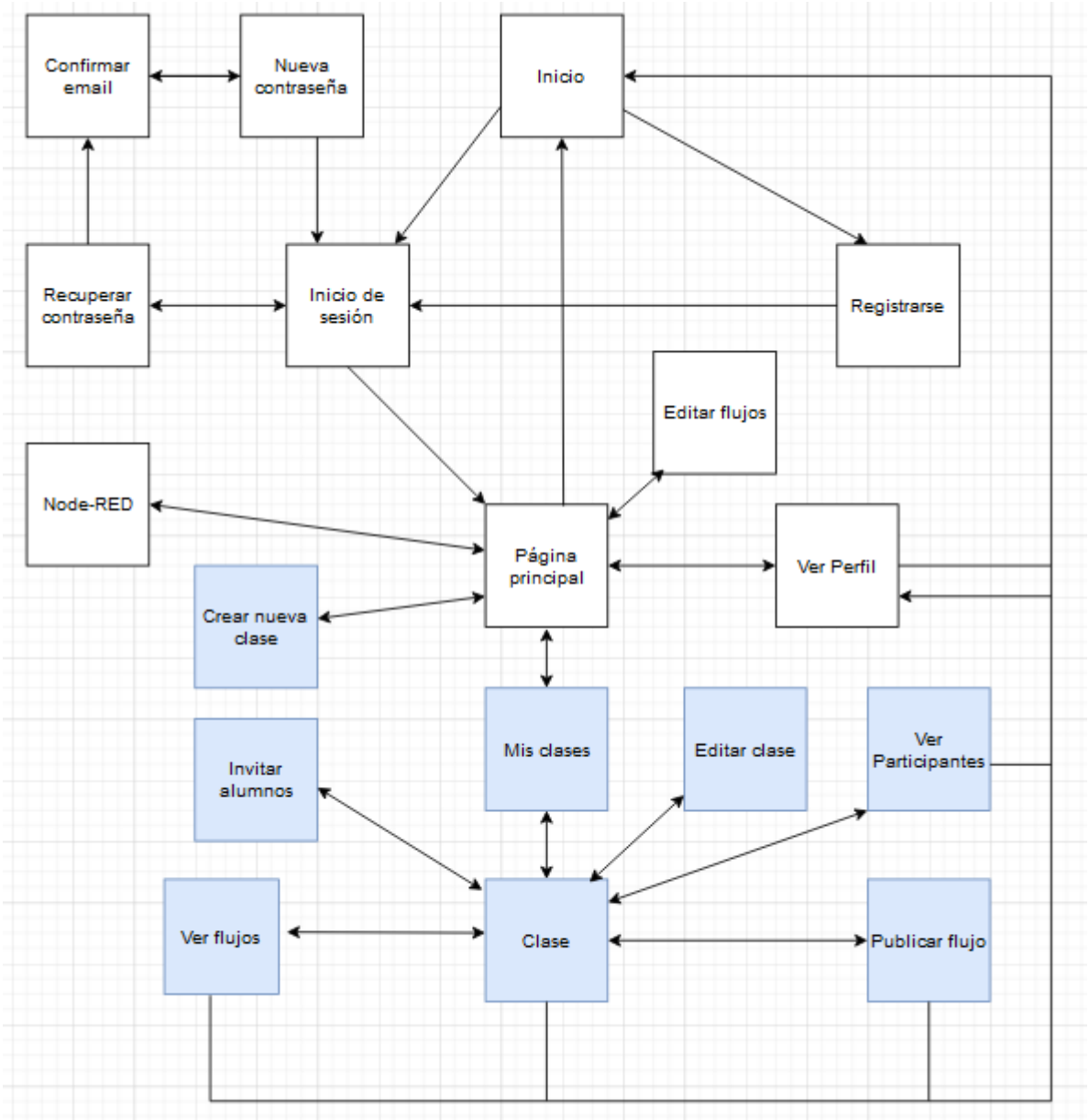


Figura 5.3.1: Diagrama de Navegación para el rol de profesor

Como se puede observar, los usuarios con el rol de profesor son los únicos capaces de crear clases, invitar alumnos o publicar flujos a modo de plantilla.

Diagrama de navegación. Rol de alumno:

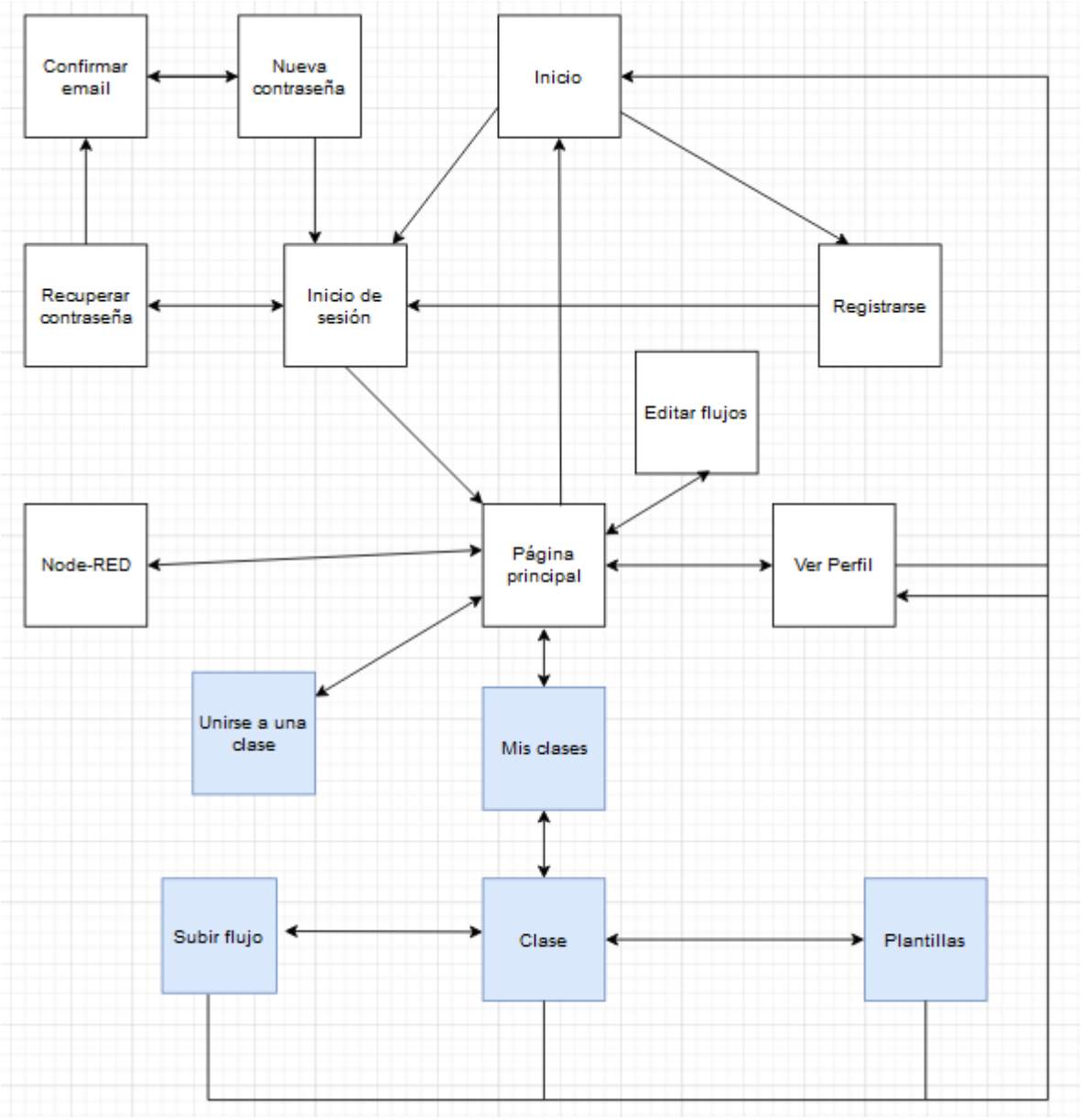


Figura 5.3.2: Diagrama de Navegación para el rol de alumno

En este caso, los alumnos son capaces de ver las plantillas subidas por los profesores a las clases o entregar flujos de trabajo para ser corregidos por los profesores.

5.4 Node-RED

Desde el Frontend del sistema se puede navegar a la aplicación de Node-RED de igual forma que a cualquier otra sección.

Como se ha comentado antes, se ha realizado una extensión del Frontend de Node-RED para poder usar funcionalidades incluidas en el Backend del sistema.

Para ello, se ha mantenido toda la estructura de la interfaz base de Node-RED, permitiendo usar la aplicación con normalidad, pero se añade en la parte superior una cabecera que permite acceder a las vistas incluidas.

Se va a mostrar un diagrama de navegación donde se puede ver cómo se navega desde el Frontend del sistema al Frontend de Node-red, incluyendo cómo se accede a las nuevas vistas añadidas:

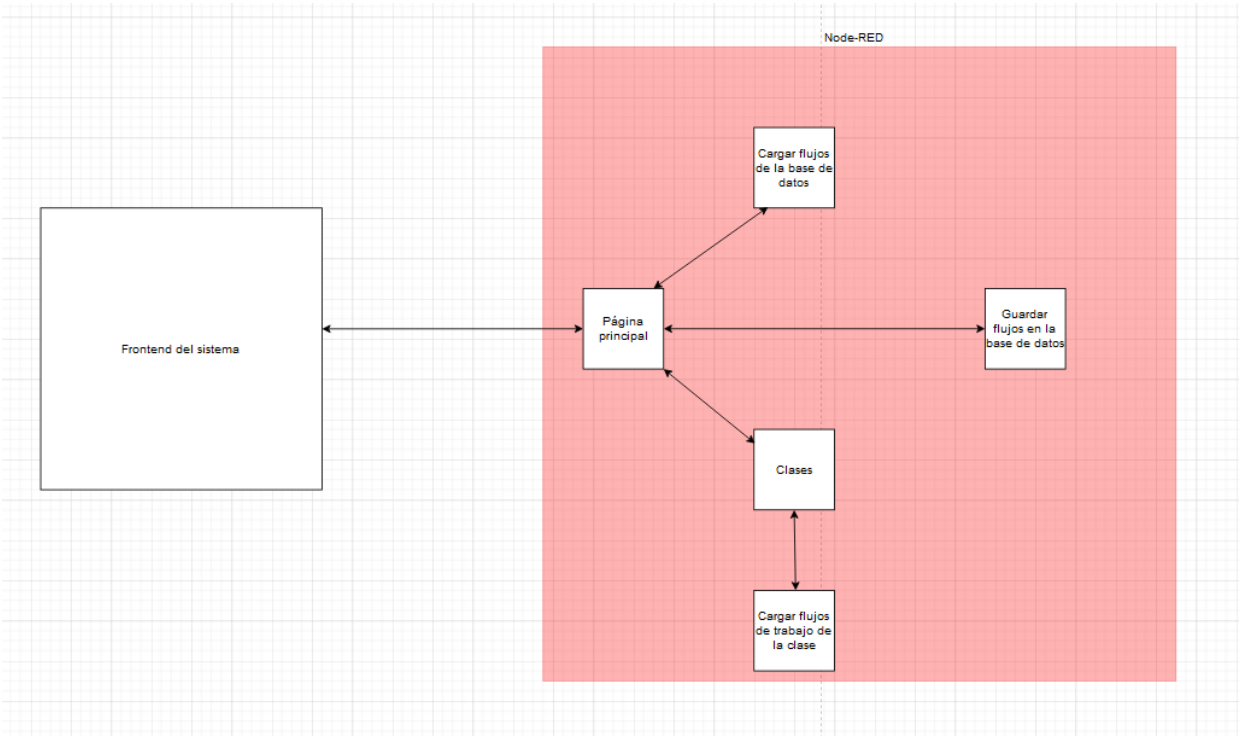


Figura 5.4.1: Diagrama de Navegación para Node-RED modificado

6. Desarrollo e Implementación

En esta sección se van a comentar las partes más cruciales de la etapa de desarrollo.

6.1 Definición de la base de datos en Node.js

Para configurar la base de datos en nuestro Backend, se ha utilizado la librería Mongoose, que facilita bastante la configuración.

Entonces, usando Mongoose, para que el Backend pueda comunicarse con la base de datos se tienen que seguir 3 pasos:
- Conectarse mediante la cadena de conexión
- Crear un esquema por cada colección
- Exportar cada colección como un modelo

Para realizar la conexión basta con este método

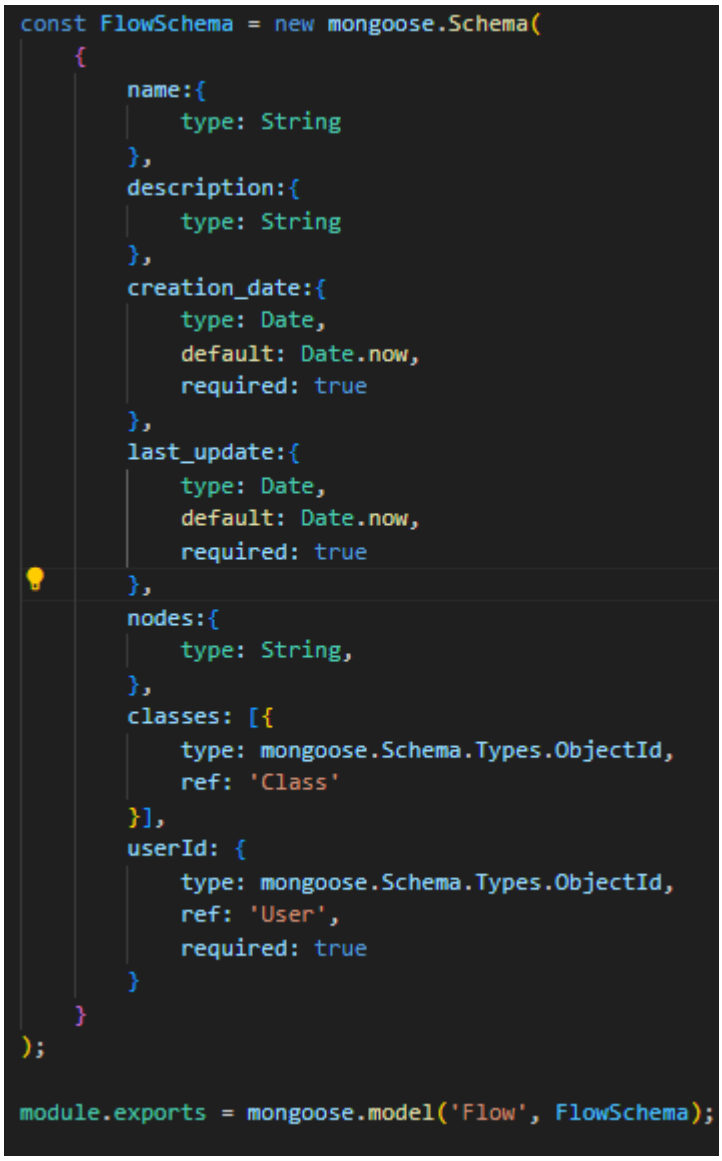
```
module.exports = () => {
    const connect = async () => {
        try {
            await mongoose.connect(process.env.DB_URI);
            console.log("Conexión correcta");
        } catch (err) {
            console.error("DB: Error!!", err);
        }
    };

    connect();
}
```

Figura 6.1.1: Conectarse a la base de datos de MongoDB

La cadena de conexión está en la variable de entorno DB_URI por seguridad.

Ahora se van a crear las colecciones de la base de datos, para ello hay que definir un nuevo esquema de Mongoose con los mismos campos de la colección.

Se muestra un ejemplo con la colección Flow:



```
const FlowSchema = new mongoose.Schema(
    {
        name:{
            type: String
        },
        description:{
            type: String
        },
        creation_date:{
            type: Date,
            default: Date.now,
            required: true
        },
        last_update:{
            type: Date,
            default: Date.now,
            required: true
        },
        nodes:{
            type: String,
        },
        classes: [{
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Class'
        }],
        userId: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User',
            required: true
        }
    }
);

module.exports = mongoose.model('Flow', FlowSchema);
```

Figura 6.1.2: Esquema de Flow

Por último, se exporta el modelo de la colección, con este modelo ya se pueden hacer operaciones sobre ella.

Con esto ya tendríamos todo lo necesario para utilizar la base de datos, tenemos la conexión con la base de datos, las colecciones creadas y ya podemos hacer operaciones como insertar, actualizar o borrar, entre otras.

6.2 Sistema de autenticación de usuarios

Garantizar la seguridad del sistema y de los usuarios ha sido algo que siempre ha estado presente mientras se ha planificado y desarrollado el proyecto. Como es un sistema que utiliza muchas peticiones entre componentes es importante garantizar que nadie puede realizar una suplantación de identidad o interceptar una petición.

Se ha utilizado la librería Passport que permite definir varias estrategias de autenticación. No obstante, se ha optado por utilizar JSON Web Token (JWT) debido a que es la mejor opción para este sistema.

Cuando un usuario se conecta, se le entrega un token que caduca en dos horas, que es el tiempo estipulado en este sistema. En cada solicitud del usuario al Backend, el JWT se incluye en la cabecera. Así, el Backend verifica si el token es válido y se encuentra activo al recibir la respuesta.

Este token al ser asignado al usuario también es firmado por una clave secreta del sistema. La firma por la clave secreta hace que cuando llega una petición del usuario al Backend, este pueda comprobar que la clave secreta es la misma con la que ha firmado el token, todo esto ayuda a detectar si alguien ha modificado el token.

La firma secreta ha sido generada por el siguiente método. Una vez generada se guarda en las variables de entorno

```
const crypto = require('crypto');
console.log(crypto.randomBytes(64).toString('hex'));
```

Figura 6.2.1: Generación de la firma secreta

Entonces, para implementar esta estrategia, primero se debe crear la estrategia utilizando la librería Passport.js:

```
const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = process.env.JWT_SECRET;

passport.use(new JwtStrategy(opts, async function (jwt_payload, done) {
    try {
        const user = await UserModel.findOne({ _id: jwt_payload.id });
        if (user) {
            return done(null, user);
        } else {
            return done(null, false);
        }
    } catch (err) {
        return done(err, false);
    }
}));
```

Figura 6.2.2: Creación de una estrategia de autenticación

Aquí se puede ver que primero se define las opciones, que es donde se asigna la clave secreta.

Después se crea la estrategia usando el método de Passport, donde se elige la estrategia de JWT. En el método se extrae del payload del token el Id del usuario y se comprueba si hay un usuario en la base de datos con ese mismo Id

Cuando un usuario inicia sesión, se le asigna el token firmado por la clave secreta:

```
const payload = { email: user.email, id: user._id };
const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: "120m" });
```

Figura 6.2.3: Asignación del JWT tras iniciar sesión

De momento, ya se le asigna el token al usuario correctamente, pero el Backend debe prepararse para proteger las rutas y comprobar el token del usuario cuando alguien hace una petición a esa ruta. Aquí se puede ver que todas las rutas están protegidas.

```
router
  .get('/', passport.authenticate('jwt', { session: false }), flowController.getAllFlows)
  .get("/:flowId", passport.authenticate('jwt', { session: false }), flowController.getOneFlow)
  .get("/classes/:flowId", passport.authenticate('jwt', { session: false }), flowController.getClassesByFlow)
  .get("/user/:flowId", passport.authenticate('jwt', { session: false }), flowController.getUserByFlow)
  .post("/", passport.authenticate('jwt', { session: false }), flowController.createNewFlow)
  .patch("/:flowId", passport.authenticate('jwt', { session: false }), flowController.updateFlow)
  .delete("/:flowId", passport.authenticate('jwt', { session: false }), flowController.deleteFlow);

module.exports = router;
```

Figura 6.2.4: Rutas de Flows

Entonces para hacer peticiones al Backend se necesita obligatoriamente pasar por la cabecera un JWT, por lo tanto, en el Frontend hay que asegurarse de pasar este JWT en cada petición al Backend.

```
useEffect(() => {
  const token = localStorage.getItem('token');
  if (token) {
    const decodedToken = parseJwt(token);
    if (decodedToken && decodedToken.id) {
      setUserId(decodedToken.id);
      axios.get(`https://backend-service-830425129942.europe-west1.run.app/api/v1/user/${decodedToken.id}`, {
        headers: {
          Authorization: `Bearer ${token}`
        }
      })
      .then(res => {
        setCurrentName(res.data.data.name);
        setCurrentEmail(res.data.data.email);
      })
      .catch(error => {
        console.error('Error al obtener los datos del usuario:', error);
      });
    }
  }
}, []);
```

Figura 6.2.5: Petición al backend para obtener un usuario

Evidentemente, se debe hacer lo mismo desde el Frontend de Node-RED, ya que tendrá que hacer peticiones al Backend del sistema también. Para ello se debe asegurar que el Frontend de Node-RED tenga acceso al JWT del usuario, en el siguiente apartado se explicará cómo se llevado a cabo esa cuestión.

6.3 Modificación e integración de Node-RED

La idea es que cada usuario tenga su propia instancia de Node-RED, eso garantiza que los trabajos de varios usuarios no se puedan cruzar. Para ello, en la fase de diseño se decidió que

cuando un usuario quiera acceder a su instancia de Node-RED, el Backend lance un contenedor con la instancia identificada con la Id del usuario (Figura 6.3.1).

Otra cosa a tener en cuenta es que Node-RED debe hacer peticiones al Backend del sistema y como se ha comentado en el apartado anterior, el Backend requiere el JWT para garantizar que la petición la ha hecho el usuario identificado. Para ello se ha decidido que en el momento de lanzar el contenedor de Node-RED, se la asigne como variable de entorno el JWT.

```
app.post('/start-nodered', passport.authenticate('jwt', { session: false }), (req, res) => {
    const userId = req.user._id;
    const token = req.headers.authorization.split(' ')[1];
    const userPort = assignPortToUser(userId);

    const command = `docker run -d -e JWT_TOKEN="${token}" --name nodered-${userId} -p ${userPort}:1880 node-red-modified:latest`;
```

Figura 6.3.1: Código para lanzar el docker de Node-RED

A pesar de su simplicidad, es importante tener en cuenta que solo el Backend de Node-RED podrá acceder a la variable de entorno, por lo tanto, el Frontend debe comunicarse con su Backend para recibir el JWT. Se ha incluido una nueva ruta en la API de Node-RED para extraer el token de las variables de entorno.

```
    adminApp.get('/jwtoken', function(req, res) {
        try {
            const jwtToken = process.env.JWT_TOKEN;
            if (!jwtToken) {
                throw new Error("JWT_TOKEN is not defined in environment variables");
            }

            res.json({ jwtToken: jwtToken });
        } catch (error) {
            console.error("Error fetching JWT token:", error.message);
            res.status(500).json({ error: error.message });
        }
    });
}
```

Figura 6.3.2: Nueva ruta en el Backend de Node-RED para el JWT recibido por variable de entorno

Este código mostrado pertenece a la etapa de desarrollo del proyecto, esto cambiará en la etapa de producción, ya que en lugar de lanzar el contenedor Docker se desplegará un

contenedor utilizando la API de Google Cloud. No obstante, conserva la misma configuración, dado que también se le asigna una variable de entorno al implementar Node-RED y este accede a dicha variable de entorno de la misma manera, por lo tanto, el desempeño es idéntico.

En relación con la modificación del código fuente de Node-RED, un requisito funcional es permitir a los usuarios cargar y guardar flujos de trabajo en Node-RED. Esto implica modificar el código para que pueda interactuar con el Backend y la base de datos del sistema.

Comprender el código de Node-RED y la estructura de archivos ha sido la parte más difícil del proyecto, ya que no había descripciones ni guías que explicaran el código.

Sin embargo, una vez realizada la tarea de comprender el código y sabiendo en que archivos se puede modificar, el desarrollo de esta modificación fue siguiendo los mismos pasos que en el Frontend del sistema, se añadieron los botones para cargar y guardar los flujos en Node-RED.

6.4 Sistema de clases

Como Node-RED tiene un gran potencial docente, se ha buscado implementar un sistema de clases donde los usuarios con el rol de profesor puedan crear clases, invitar alumnos, publicar flujos de trabajo o ver los flujos de trabajo de los alumnos. Por otro lado, los usuarios con el rol de alumno podrán unirse a clases mediante un código de invitación, entregar sus flujos en una clase o utilizar los flujos de trabajo publicados por el profesor de la clase.

Lo primero que se debía hacer para implementar esto son los roles de usuario, ya que hasta el momento no estaba implementado. Para implementar esto, simplemente se le añade un campo role al esquema de User:

```
const UserSchema = new mongoose.Schema({
    email: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    name: {
        type: String
    },
    role: {
        type: String,
        enum: ['admin', 'student', 'professor', 'user'],
        default: 'user'
    },
    createdClasses: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Class'
    }],
    joinedClasses: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Class'
    }],
    flows: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Flow'
    }]
});

module.exports = mongoose.model('User', UserSchema);
```

Figura 6.4.1: Esquema de usuario

El rol se elige en la página de registro, en el componente Register.jsx. En el formulario se añade el rol para que se elija mediante un checkbox.

Figura 6.4.2 Una parte del formulario de registro

Es cierto que implementar las funcionalidades de las clases y las vistas no requiere de hacer algo demasiado complejo, sin embargo, tiene una cantidad enorme de vistas y endpoints que ha consumido bastante tiempo de trabajo, sobre todo porque hay que desarrollar prácticamente el doble de vistas y endpoints al tener dos roles.

6.5 Sistema de envío de correos electrónicos

Aunque la invitación a una clase por correo electrónico es una funcionalidad que pertenece al sistema de clases, es preferible comentarse de forma separada ya que se han tenido que configurar varias cosas precisamente para desarrollar esta funcionalidad.

Para crear la invitación por email, se empleará Nodemailer, una herramienta que hace más sencillo este proceso.

La primera tarea consiste en desarrollar el transporter, el cual se encargará de enviar los correos electrónicos desde el código.

```
const transporter = nodemailer.createTransport({
    service: 'gmail',
    host: "smtp.gmail.com",
    port: 587,
    secure: false,
    auth: {
        user: process.env.EMAIL_USER,
        pass: process.env.EMAIL_PASS
    }
});
```

Figura 6.5.1 Definición del transporter

Luego hay que definir la función para construir el correo que se va a enviar, asignándole el transporter que será quien envíe dicho correo:

```
// Función para enviar correo de invitación
const sendInviteEmail = (recipientEmail, className, inviteLink) => {
    return new Promise((resolve, reject) => {
        const mailOptions = {
            from: process.env.EMAIL_USER,
            to: recipientEmail,
            subject: `Multiuser-NodeRED - Invitación a la clase ${className}`,
            text: `Has sido invitado a unirte a la clase ${className}. Utiliza el siguiente código para unirte a la clase: ${inviteLink}

            Regístrate o inicia sesión aquí: https://frontend-service-830425129942.europe-west1.run.app`
        };

        transporter.sendMail(mailOptions, (error, info) => {
            if (error) {
                console.error('Error al enviar el email:', error);
                return reject(error);
            }
            console.log('Email enviado:', info.response);
            resolve(info);
        });
    });
};
```

Figura 6.5.2 Función para enviar un email

A esta función se le llama desde el siguiente endpoint:

```
  .post('/send-invite', async (req, res) => {
    const { recipientEmails, className, classId } = req.body;
    const inviteLink = `${classId}`;
    try {
        await Promise.all(recipientEmails.map(email => {
            return sendMail.sendInviteEmail(email, className, inviteLink);
        }));
        res.status(200).send({ message: 'Invitations sent successfully' });
    } catch (error) {
        res.status(500).send({ message: 'Error sending invitations', error: error.message });
    }
});
```

Figura 6.5.3 Ruta del método de la invitación

Y a este endpoint se le llama desde el Frontend, que es donde se le pasan los parámetros que se necesitan

```
const handleSubmit = async (e) => {
    e.preventDefault();
    const recipientEmails = emails.split(',').map(email => email.trim());
    const token = localStorage.getItem('token');
    if (token) {
        try {
            await axios.post('https://backend-service-830425129942.europe-west1.run.app/api/v1/class/send-invite', { recipientEmails, className, classId }, {
                headers: {
                    Authorization: `Bearer ${token}`
                }
            });
            alert('Invitaciones enviadas exitosamente');
            navigate(-1);  // Redirige a la página anterior
        } catch (error) {
            console.error('Error enviando las invitaciones:', error);
        }
    } else {
        console.error('No token found');
    }
};
```

Figura 6.5.4 Solicitud desde el Frontend para enviar un correo

Esta es la otra función que utiliza correo electrónico, que es la de recuperar una contraseña, como se puede ver sigue una estructura muy similar e incluso reutiliza el transporter.

```
// Función para enviar correo de recuperación de contraseña
const sendResetPasswordEmail = (recipientEmail, resetLink) => {
    return new Promise((resolve, reject) => {
        const mailOptions = {
            from: "process.env.EMAIL_USER",
            to: recipientEmail,
            subject: 'Restablecimiento de contraseña',
            text: `Has solicitado restablecer tu contraseña. Usa el siguiente enlace para establecer una nueva contraseña: ${resetLink}.
                   El enlace caducará en una hora.`
        };

        transporter.sendMail(mailOptions, (error, info) => {
            if (error) {
                console.error('Error al enviar el email:', error);
                return reject(error);
            }
            console.log('Email enviado:', info.response);
            resolve(info);
        });
    });
};
```

Figura 6.5.5 Función para enviar un correo

7. Despliegue

Tras finalizar el desarrollo del sistema, se debe desplegar en un entorno de producción.

Para el despliegue se ha elegido el servicio de Google Cloud Run, que permite desplegar aplicaciones dockerizadas y se encarga de toda la configuración que hay detrás. Además, como se va a desplegar cada componente por separado, se va a dockerizar cada componente.

La etapa de despliegue sería sencilla si se pudiera desplegar cada componente manualmente por separado. Esto es cierto que podemos hacerlo con el Backend y el Frontend, pero en el caso de las instancias de Node-RED se tienen que desplegar de forma automática cuando un nuevo usuario se registre y quiera acceder a su instancia. Cabe destacar que no es necesario desplegar la base de datos, ya que al utilizar MongoDB Atlas ya se encuentra desplegada en la nube.

El despliegue del Backend y Frontend no se van a explicar en este apartado, ya que es un proceso que solo se debe realizar una vez y la mayor parte de este proceso radica en la configuración de la plataforma de Google Cloud. Por lo tanto, esa parte se detallará paso por paso en el Apéndice B - Manual de Despliegue.

En esta sección, se va a poner el foco en explicar cómo se ha realizado el despliegue de las instancias de Node-RED. Como se ha comentado, esta parte es de los puntos más críticos del desarrollo, ya que no se trata simplemente de desplegar un componente en la nube, sino que se deben desplegar las instancias de Node-RED de manera independiente para cada usuario y en el momento en el que lo solicite el usuario. A su vez, estas instancias también deberán ser detenidas una vez que el usuario deje de usarlas para liberar espacio.

En un primer momento, se pensó que al pulsar el botón de Node-RED, enviara una petición al backend y que este ejecutara el mismo código que se ha utilizado para los componentes anteriores, y aunque esto funcionaba cuando el Backend estaba ejecutándose en local, no

funcionaba una vez que el Backend estaba desplegado en Google Cloud Run, y esto es debido a que el entorno de Google Cloud no reconoce el comando `gcloud`, por lo que se necesitaba seguir otra estrategia.

Por lo tanto, se cambió de estrategia, que fue hacer uso de la API de Google Cloud desde el código.

Se ha definido un endpoint en el Backend, llamado start-nodered en el que se configura las credenciales para poder usar la API de Google Cloud desde el código y se comprueba si ya existe el mismo servicio desplegado. Si existe, significa que el usuario ya ha usado Node-Red y ya tiene el servicio desplegado, por lo tanto, únicamente hay que hacer un redespliegue cambiando el JWT antiguo por el actual y se le redirige al enlace del servicio, esto es mucho más rápido que tener que eliminar y volver a desplegar el servicio de nuevo. En el caso de que no esté desplegado el servicio, se despliega y se le asigna también el JWT.

En las siguientes imágenes se va a mostrar todo el proceso que se hace desde que el usuario llama al endpoint hasta que se despliega la instancia de Node-RED.

En primer lugar, se define la ruta del endpoint

```
router
.post('/start-nodered', passport.authenticate('jwt', { session: false }), noderedController.startNodered)
```

Figura 7.1: Ruta del endpoint

La siguiente imagen muestra el controlador que maneja la petición del usuario y da la respuesta.

```
const startNodeRed = async (req, res) => {
    const userId = req.user._id;
    const token = req.headers.authorization.split(' ')[1];

    try {
        const response = await nodeRedService.startNodeRed(userId, token);
        res.status(200).send(response);
    } catch (error) {
        console.error(`Error launching Node-RED container: ${error}`);
        res.status(500).send({
            success: false,
            message: 'Error launching Node-RED container',
            error: error.message,
        });
    }
};
```

Figura 7.2: Controlador para desplegar Node-RED

Ahora se muestra el servicio que realiza todo el proceso de despliegue

```
const startNodeRed = async (userId, token) => {
    const serviceName = `node-red-service-${userId}`;
    const imageName = 'gcr.io/multiuser-node-red/node-red-image';
    const projectId = 'multiuser-node-red';
    const region = 'europe-west1';

    try {
        const auth = new google.auth.GoogleAuth({
            keyFile: path.join(__dirname, process.env.GOOGLE_APPLICATION_CREDENTIALS),
            scopes: ['https://www.googleapis.com/auth/cloud-platform'],
        });

        const authClient = await auth.getClient();

        let serviceUrl;
        try {
            const describeRequest = {
                name: `projects/${projectId}/locations/${region}/services/${serviceName}`,
                auth: authClient,
            };
```

Figura 7.3: Se configura las credenciales y los parámetros del despliegue

A continuación, se comprueba si la instancia de ese usuario ya está desplegada, en ese caso, no es necesario hacer un despliegue, sino que basta con hacer un redespliegue actualizando el JWT como variable de entorno. Este redespliegue es bastante rápido.

Cabe destacar que ahora mismo cuando un usuario sale de su instancia de Node-RED se elimina instantáneamente, por lo que el redespliegue no se va a utilizar en esta versión del proyecto. Sin embargo, en la sección de líneas futuras se propondrá implementar una function que detenga automáticamente una instancia después de un tiempo de inactividad, con lo cual se podría usar el redespliegue ya implementado.

```
let serviceUrl;
try {
    const describeRequest = {
        name: `projects/${projectId}/locations/${region}/services/${serviceName}`,
        auth: authClient,
    };

    const describeResponse = await run.projects.locations.services.get(describeRequest);
    serviceUrl = describeResponse.data.status.url;

    if (serviceUrl) {
        console.log('Service already exists:', serviceUrl);

        describeResponse.data.spec.template.spec.containers[0].env = [{
            name: 'JWT_TOKEN',
            value: token,
        }];

        const updateRequest = {
            name: `projects/${projectId}/locations/${region}/services/${serviceName}`,
            requestBody: describeResponse.data,
            auth: authClient,
        };

        const updateResponse = await run.projects.locations.services.replaceService(updateRequest);
        serviceUrl = updateResponse.data.status.url;

        return {
            success: true,
            message: 'Node-RED container updated successfully',
            url: serviceUrl,
        };
    }
} catch (error) {
    if (error.code === 404) {
        console.log('Service does not exist, will create a new one.');
    } else {
        throw error;
    }
}
```

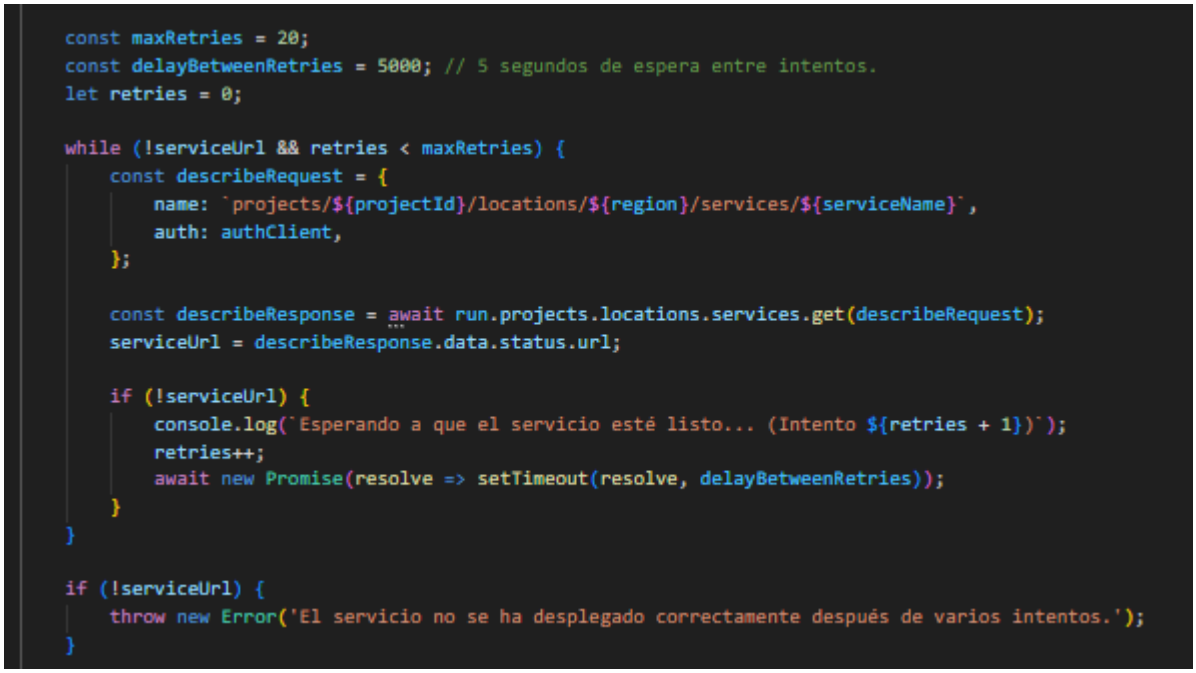
Figura 7.4: Se comprueba si está desplegado

Ahora se muestra el caso en que la instancia no está desplegada para ese usuario.

```
const createRequest = {
    parent: `projects/${projectId}/locations/${region}`,
    requestBody: {
        apiVersion: 'serving.knative.dev/v1',
        kind: 'Service',
        metadata: {
            name: serviceName,
            namespace: projectId,
        },
        spec: {
            template: {
                spec: {
                    containers: [
                        {
                            image: imageName,
                            env: [
                                {
                                    name: 'JWT_TOKEN',
                                    value: token,
                                },
                            ],
                        },
                    ],
                },
            },
        },
    },
    auth: authClient,
};

const createResponse = await run.projects.locations.services.create(createRequest);
```

Figura 7.5: Se construye la petición a Google Cloud con las variables definidas al principio del método

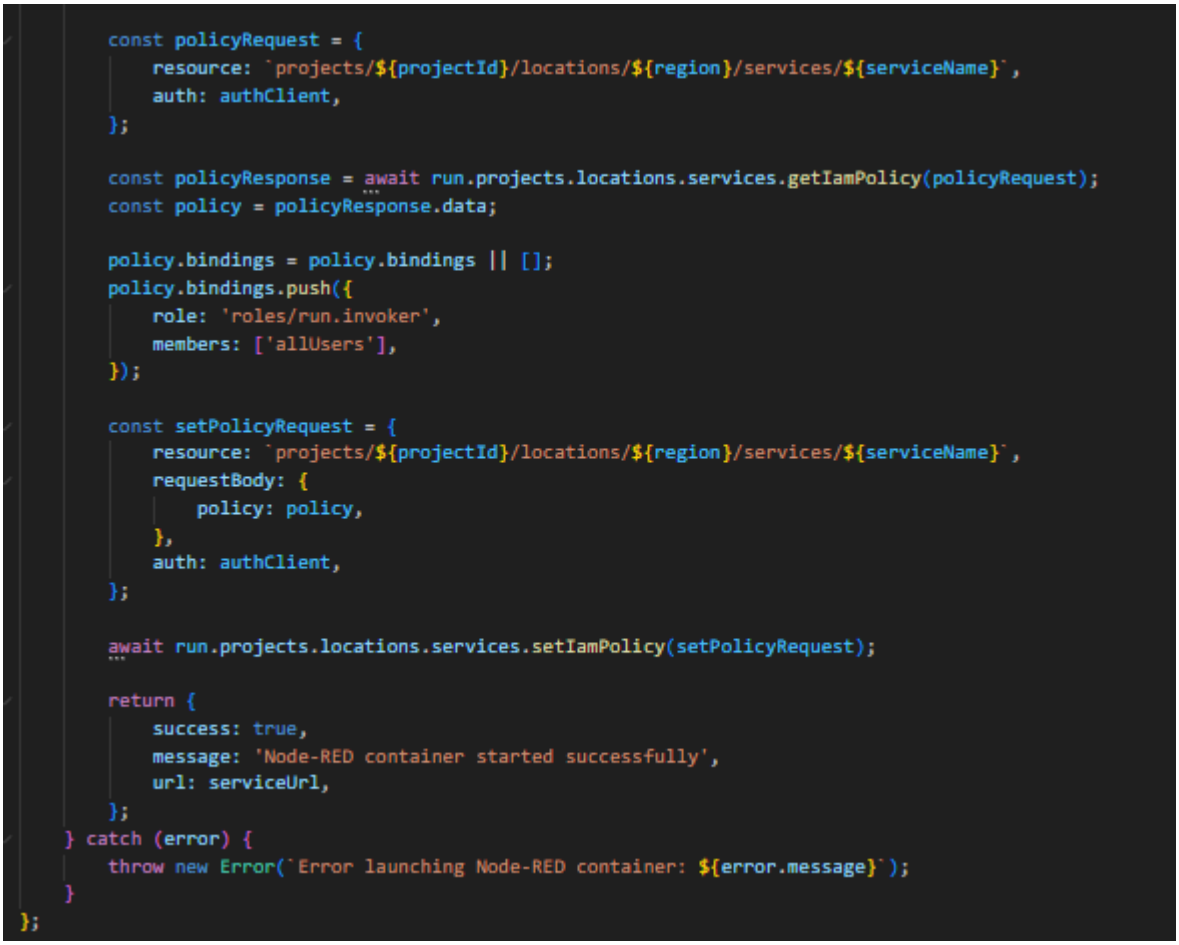
Ahora se muestra cómo se hace la petición de despliegue y se espera hasta que el servicio esté desplegado para mandar la respuesta al Frontend.



```
const maxRetries = 20;
const delayBetweenRetries = 5000; // 5 segundos de espera entre intentos.
let retries = 0;

while (!serviceUrl && retries < maxRetries) {
    const describeRequest = {
        name: `projects/${projectId}/locations/${region}/services/${serviceName}`,
        auth: authClient,
    };

    const describeResponse = await run.projects.locations.services.get(describeRequest);
    serviceUrl = describeResponse.data.status.url;

    if (!serviceUrl) {
        console.log(`Esperando a que el servicio esté listo... (Intento ${retries + 1})`);
        retries++;
        await new Promise(resolve => setTimeout(resolve, delayBetweenRetries));
    }
}

if (!serviceUrl) {
    throw new Error('El servicio no se ha desplegado correctamente después de varios intentos.');
}
```

Figura 7.6: Espera hasta que el servicio esté desplegado en Google Cloud Run

Por último, en la siguiente imagen se configura la política de IAM para que cualquier usuario pueda acceder al servicio desplegado. La política de IAM en Google Cloud es la que define qué usuarios tienen acceso a los recursos de Google Cloud y qué tipo de acceso tienen.



```
        const policyRequest = {
            resource: `projects/${projectId}/locations/${region}/services/${serviceName}`,
            auth: authClient,
        };

        const policyResponse = await run.projects.locations.services.getIamPolicy(policyRequest);
        const policy = policyResponse.data;

        policy.bindings = policy.bindings || [];
        policy.bindings.push({
            role: 'roles/run.invoker',
            members: ['allUsers'],
        });

        const setPolicyRequest = {
            resource: `projects/${projectId}/locations/${region}/services/${serviceName}`,
            requestBody: {
                policy: policy,
            },
            auth: authClient,
        };

        await run.projects.locations.services.setIamPolicy(setPolicyRequest);

        return {
            success: true,
            message: 'Node-RED container started successfully',
            url: serviceUrl,
        };
    } catch (error) {
        throw new Error(`Error launching Node-RED container: ${error.message}`);
    }
};
```

Figura 7.7: Configuración política de IAM

Esta estrategia es más compleja y hay que tener en cuenta más detalles para desplegar un servicio, como configurar la política de IAM. Sin embargo, ofrece una solución idónea ya que cumple con todos los requisitos que se necesitaban, como asignar un identificador único al nombre de cada servicio relacionado con el userID del sistema o modificar el JTW para poder hacer peticiones al Backend.

Ya se ha desplegado la instancia de Node-RED. Sin embargo, cuando un usuario deje de trabajar se tendrá que detener para liberar recursos. Esto se ha implementado con otro

endpoint que sigue la misma estructura de separación de responsabilidades que se ha usado hasta ahora.

```
.post('/stop-nodered', passport.authenticate('jwt', { session: false }), nodeRedController.stopNodeRed);
```

Figura 7.8: Ruta para detener Node-RED

Ahora se mostrará el controlador que gestiona las peticiones y devuelve la respuesta.

```
const stopNodeRed = async (req, res) => {
    const userId = req.user._id;

    try {
        const response = await nodeRedService.stopNodeRed(userId);
        res.status(200).send(response);
    } catch (error) {
        console.error(`Error stopping Node-RED service: ${error}`);
        res.status(500).send({
            success: false,
            message: 'Error stopping Node-RED service',
            error: error.message,
        });
    }
};
```

Figura 7.9: Controlador para detener Node-RED

Y ahora el servicio, que hace uso de nuevo de la API de Google para detener el servicio ya desplegado identificado con el ID del usuario.

```
const stopNodeRed = async (userId) => {
    const serviceName = `node-red-service-${userId}`;
    const projectId = 'multiuser-node-red';
    const region = 'europe-west1';

    try {
        const auth = new google.auth.GoogleAuth({
            keyFile: path.join(__dirname, process.env.GOOGLE_APPLICATION_CREDENTIALS),
            scopes: ['https://www.googleapis.com/auth/cloud-platform'],
        });

        const authClient = await auth.getClient();

        const deleteRequest = {
            name: `projects/${projectId}/locations/${region}/services/${serviceName}`,
            auth: authClient,
        };

        await run.projects.locations.services.delete(deleteRequest);
        console.log('Service deleted:', serviceName);

        return {
            success: true,
            message: 'Node-RED service stopped and removed successfully',
        };
    } catch (error) {
        throw new Error(`Error stopping Node-RED service: ${error.message}`);
    }
};
```

Figura 7.10: Servicio para detener Node-RED

Por tanto, cuando el usuario quiera usar Node-RED se despliega el servicio y cuando quiera salir se detiene, por lo que se liberan todos los recursos hasta que el usuario quiera volver a trabajar.

8 Conclusiones y Líneas Futuras

8.1 Conclusiones

En este Trabajo Fin de Grado se ha realizado un sistema multiusuario desplegado en la nube que da soporte a Node-RED. Llevarlo a cabo ha sido todo un desafío, ya que, aunque he sido capaz de poner en práctica la mayoría de los conocimientos obtenidos durante la carrera, ha sido obligatorio investigar y aprender nuevas tecnologías.

Cuando surgió la idea del proyecto había muchas dudas en el aire, se requerían cumplir objetivos que no se estaba del todo seguro la forma en la que podría hacerse o incluso si podría hacerse. Después de investigar y antes de realizar el anteproyecto parece ser que había ideas que acercaban a la solución. Sin embargo, hasta que no se realiza no se puede estar del todo seguro. Así que llevar a cabo este proyecto ha requerido de paciencia y de no desanimarse cuando surgía algún imprevisto.

Uno de los momentos más complicados fue en la modificación del código fuente de Node-RED, ya que previamente se necesitaba investigar y aprender cómo estaba estructurado el código, para estar seguro de qué archivos se podían modificar. Esto no fue nada fácil, ya que él código fuente contiene cientos de archivos y carpetas estructurados de una forma no demasiado clara y tampoco existe una documentación que pudiera servir de ayuda. Por lo tanto, esta tarea requirió de días de investigación.

Por otra parte, realizar este proyecto ha sido una gran experiencia. El llevar a cabo un proyecto desde cero, cuando solo era una idea, realizar todas las fases del desarrollo hasta finalizarlo ha sido una experiencia enriquecedora, tanto profesional como personalmente.

Profesionalmente, ha sido muy útil para reforzar todos los conocimientos vistos en el grado. Se han tenido que aplicar conocimientos de diseño, análisis o desarrollo, entre otros. Además, ha servido para aprender muchas cosas no vistas durante el grado, como es el uso de la plataforma Google Cloud o de Docker. A pesar de los momentos complicados que se han comentado antes, son cuestiones que ocurren frecuentemente en el ámbito laboral, por lo que este proyecto ha sido de ayuda también para prepararse y saber afrontar esas situaciones.

8.2 Líneas futuras

En el futuro se podrían implementar las siguientes funcionalidades:

- Eliminar automáticamente las instancias de Node-RED en Google Cloud Run: Esto podría ser útil si hay usuarios inactivos durante un largo periodo de tiempo, se podría eliminar automáticamente y dejar de consumir recursos. Ahora mismo, las instancias se detienen una vez el usuario salga de Node-RED. Sin embargo, podría ser interesante que se detengan automáticamente tras un tiempo sin usarse, ya que si se mantiene la instancia desplegada, cuando el usuario quiera entrar de nuevo sería mucho más rápido que lo que ocurre ahora, que cada vez que el usuario quiere acceder a Node-RED se debe desplegar de nuevo la instancia. Aunque requiere de una mayor investigación para realizar esto, una idea sería utilizar Google Cloud Functions, creando una función que se ejecute en base a un cron job programado con Cloud Scheduler

- Mejora de la interfaz de usuario: Aunque se ha realizado una interfaz intuitiva, aún es muy mejorable. Se podría dedicar más tiempo al estudio y desarrollo de técnicas de diseño para ofrecer una experiencia de usuario más pulida y eficiente

- Realizar una mayor cantidad de pruebas: Se han llevado a cabo pruebas para verificar que el sistema funciona de manera adecuada. Sin embargo, lo ideal sería que probaran múltiples usuarios a la vez

- Tablón de comentarios: Añadir una sección donde los profesores puedan hacer comentarios en las clases para que los alumnos lo vean, aquí se podría poner indicaciones o enunciados a ejercicios

- Calificar ejercicios: A los flujos de trabajo se le podría añadir un campo opcional que sea la calificación, el cual podrá modificar un profesor cuando el alumno lo suba a su clase

- Colaboración grupal: Se implementaría lo necesario para que varios usuarios puedan trabajar en la misma instancia de Node-RED a la vez. Aunque este es un paso más ambicioso también puede ser bastante útil

- Ampliación de la base de datos: La base de datos actual tiene capacidad suficiente para los datos que se prevén en un corto plazo, pero lo más conveniente es ir preparando una ampliación de esta por si hay un aumento considerable en el número de usuarios

Bibliografía

[1] REST API Design Best Practices Handbook – How to Build a REST API with JavaScript, Node.js, and Express.js - https://www.freecodecamp.org/news/rest-api-design-best-practices-build-a-rest-api/

[2] Qué es JavaScript, para qué sirve y cómo funciona https://blog.hubspot.es/website/que-es-javascript#que-es -

[3] Qué es NodeJS y para qué sirve - https://openwebinars.net/blog/que-es-nodejs/

[4] Qué es MongoDB - https://openwebinars.net/blog/que-es-mongodb/

[5] MongoDB Atlas Tutorial - https://www.mongodb.com/resources/products/platform/mongodb-atlas-tutorial

[6] ¿Qué es JSON Web Token? - https://www.aluracursos.com/blog/que-es-json-web-token

[7] Acerca de GitHub de escritorio - https://docs.github.com/es/desktop/overview/about-github-desktop

[8] ¿Qué es Visual Studio Code (VS Code)? - https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas#tree-1

[9] Learn What is MongoDB Atlas Database - https://www.simplilearn.com/tutorials/mongodb-tutorial/what-is-mangodb-atlas-database

[10] ¿Cómo utilizar Postman para las pruebas de API? - https://qalified.com/es/blog/postman-para-api-testing/

[11] [2024] Tutorial sobre Draw.io – La mejor herramienta para diseñar diagramas online - https://beatrizcalvo.com/tutorial-draw-io-herramienta-diagramas/

[12] Docker: ¿qué es y cómo se usa? - https://datascientest.com/es/docker-todo-que-saber

[13] Descripción general de Google Cloud - https://cloud.google.com/docs/overview?hl=es-419

[14] ¿Qué es Cloud Run? - https://cloud.google.com/run/docs/overview/what-is-cloud-run?hl=es-419

[15] What is Swagger? - https://swagger.io/tools/open-source/getting-started/

[16] Node-RED, la herramienta de programación visual para el Internet of Things. Pickdata - https://www.pickdata.net/es/noticias/node-red-programacion-visual-iot

[17] Introduction to Node-RED. Medium - https://medium.com/@caglar.cavdar94/introduction-to-node-red-dfbc64293287

[18] Logo de JavaScript. Logos World - https://logos-world.net/javascript-logo/

[19] Sánchez Fernández, J. (2024). Cómo crear un API REST con Node.js. Xurxo Dev - https://xurxodev.com/como-crear-un-api-rest-con-node-js/

[20] Fernando, R. (2024). About Express.js. Medium - https://medium.com/@rahulrulz680/about-express-js-69551246f533

[21] Logo de React. Logos World - https://logos-world.net/react-logo/

[22] Logo de HTML5. Pixabay - https://pixabay.com/illustrations/logo-html-html5-icon-2582748/

[23] Logo de CSS. 1000 Logos - https://1000logos.net/css-logo/

[24] Logo de Bootstrap. PNGWing - https://www.pngwing.com/es/free-png-ahobq

[25] Logo de MongoDB. Wikimedia Commons - https://es.m.wikipedia.org/wiki/Archivo:MongoDB_Logo.svg

[26] Fiori, A. (2020). MongoDB Atlas – Creating a Cloud Environment for Practice. FlowyGo - https://flowygo.com/en/blog/mongodb-atlas-creating-a-cloud-environment-for-practice/

[27] Foo, L. Y. (2021). MongoDB Compass: A No-Code Tool for Data Analysts. Medium - https://yflooi.medium.com/mongodb-compass-a-no-code-tool-for-data-analysts-e3cef605a670

[28] Logo de Visual Studio Code. Seek Vectors - https://seekvectors.com/post/visual-studio-code-logo

[29] Logo de Git. Wikimedia Commons - https://es.m.wikipedia.org/wiki/Archivo:Git-logo.svg

[30] Logo de GitHub. Logos World - https://logos-world.net/github-logo/

[31] Logo de GitHub Desktop. Wikimedia Commons - https://commons.wikimedia.org/wiki/File:Github-desktop-logo-symbol.svg

[32] Logo de Postman. Wikimedia Commons - https://commons.wikimedia.org/wiki/File:Postman_%28software%29.png

[33] Cerbino, L. N. (2020). Documentando una API con Swagger y .NET Core. LinkedIn - https://www.linkedin.com/pulse/documentando-una-api-con-swagger-y-net-core-lucas-

nahuel-cerbino-/

[34] Logo de MagicDraw. G2 - https://www.g2.com/products/magicdraw/reviews

[35] Castro, B. Logo de Draw.io. Pinterest -

https://es.pinterest.com/pin/458382068324321600/

[36] Logo de Docker. Wikimedia Commons -

https://ca.m.wikipedia.org/wiki/Fitxer:Docker_logo.png

[37] Explore Docker Desktop. AWS Community -

https://community.aws/content/2lBh6Gi9X6yMs2186bHLpziZ0t3/explore-docker-desktop

[39] Logo de Google Cloud. Wikimedia Commons -

https://es.wikipedia.org/wiki/Archivo:Google_Cloud_logo.svg

[40] Zakaria. (2023). Demystifying Google Cloud Run Pricing: Untangling CPU & Memory. LinkedIn - https://www.linkedin.com/pulse/demystifying-google-cloud-run-pricing-untangling-cpu-memory-zakaria/

[41] Node.js v22.8.0 documentation - https://nodejs.org/docs/latest/api/

[42] Aprende React - https://es.react.dev/learn

[43] Get started with Bootrstrap - https://getbootstrap.com/docs/4.1/getting-started/introduction/

[44] Docker manuals - https://docs.docker.com/manuals/

[45] Express Documentation - https://expressjs.com/

[46] Documentación de Google Cloud -

https://cloud.google.com/docs?_gl=1*9bnrt6*_up*MQ..&gclid=CjwKCAjwxY-3BhAuEiwAu7Y6s3oDrFzGE5p9p02xjSvfez2tLNoVNZaPKMIzr1A5WgOVrZBEgR_SQhoCZeUQAvD_BwE&gclsrc=aw.ds&hl=es-419

[47] Documentación de Cloud Run - https://cloud.google.com/run/docs?hl=es-419

Apéndice A - Manual de Instalación

En el primer apéndice de este documento, se hará una explicación de como poder instalar y ejecutar el sistema.

Entorno de Producción

Como se ha comentado, para usar el sistema en el entorno de producción no es necesario una instalación local.

Al estar desplegado en la nube, solo será necesario tener conexión a internet, acceder al siguiente enlace y ya se podrá utilizar el sistema con normalidad.
https://multi-node-red-830425129942.europe-west1.run.app/

A pesar de ello, y aunque no se pueda utilizar localmente, el código fuente de esta versión de producción están en los siguientes repositorios, en sus respectivas ramas main:

- Backend: https://github.com/pablocrv12/multiuser-node-red
- Frontend: https://github.com/pablocrv12/vm-node-red
- Node-RED: https://github.com/pablocrv12/node-red-modified

Entorno de desarrollo

En el caso de que se quiera instalar y utilizar el sistema de forma local, se necesitan hacer unas configuraciones previas

Para ejecutar el sistema en local se necesita tener instalado Node.js, npm y Docker Desktop. Además, se necesita ejecutar tanto el Backend como el Frontend y construir un Docker de la versión modificada de Node-RED.

Pasos para ejecutar el Backend:

1. Clonar el repositorio: https://github.com/pablocrv12/multiuser-node-red
2. Cambiar a la rama "desarrollo"
3. Configurar las variables de entorno

   Se debe abrir el proyecto, crear un archivo .env en el directorio ráiz y darle valor a las siguientes variables de entorno:

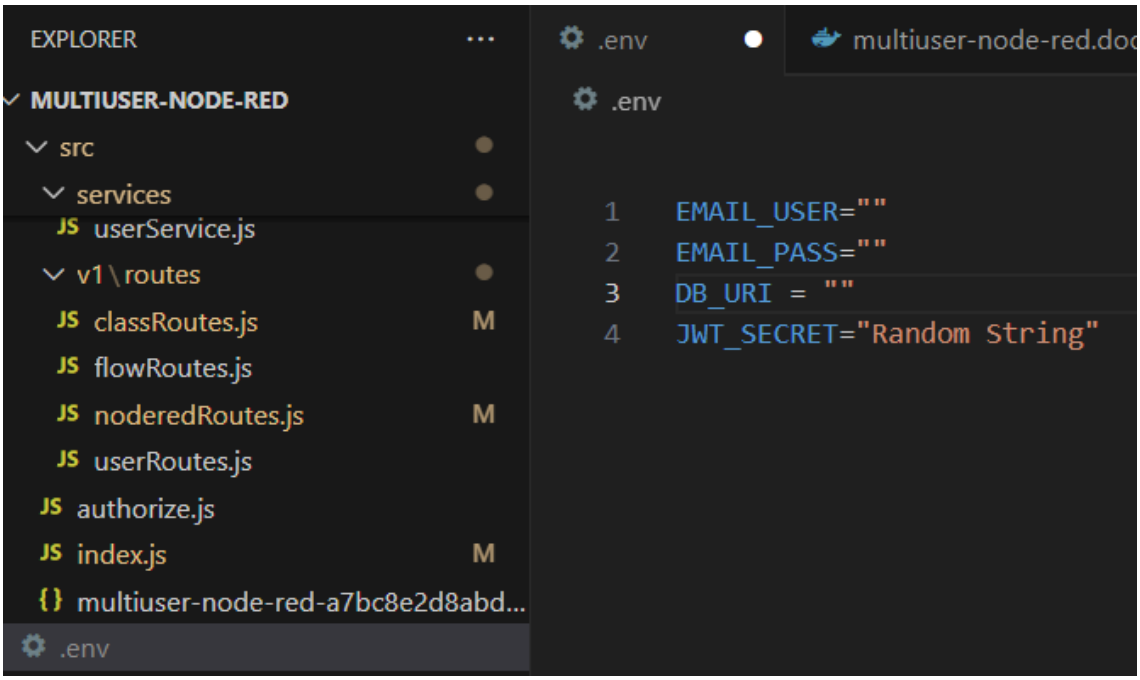


Figura A.1: Variables de entorno a cambiar

   En EMAIL_USER se debe poner un correo electrónico y en EMAIL_PASS la contraseña, este email es para ser el encargado de enviar los correos electrónicos de invitación y recuperación de la constraseña.

   En DB_URI se debe poner una cadena de conexión a una base de datos MongoDB.

4. Abrir un terminal y ubicarse en el directorio raíz del proyecto

5. Ejecutar npm install en el terminal para instalar las dependencias
6. Ejecutar npm run dev para ejecutar la aplicación

Pasos para ejecutar el Frontend:

En el caso del frontend, se siguen pasos similares a los del backend, aunque con una diferencia importante: no es necesario configurar variables de entorno.

1. Clonar el repositorio: https://github.com/pablocrv12/vm-node-red
2. Cambiar a la rama "desarrollo"
3. Abrir un terminal y ubicarse en el directorio raíz del proyecto
4. Ejecutar npm install en el terminal para instalar las dependencias
5. Ejecutar npm run dev para ejecutar la aplicación

Pasos para construir el Docker de Node-RED:

1. Clonar el repositorio: https://github.com/pablocrv12/node-red-modified
2. Cambiar a la rama "desarrollo"
3. Abrir Docker Desktop
4. En una terminal, ubicarse en el directorio raiz del proyecto
5. Ejecutar el comando 'npm install --legacy-peer-deps' en el terminal para instalar las dependencias
6. Ejecutar el comando 'docker build --no-cache -t node-red-modified:latest .'

Cuando la imagen de Docker termine de construirse, ya se podrá utilizar el sistema.

Todo lo necesario está instalado. Ahora, para poder usar la aplicación se necesita:

- Tener en ejecución el backend (Usando npm run dev)
- Tener en ejecución el Frontend (Usando npm run dev)
- Tener en ejecución Docker desktop (Solo se necesita abrir la aplicación)

Para utilizar el sistema, se debe acceder al enlace con el puerto donde se esté ejecutando el Frontend. Por ejemplo http://localhost:5173

Por último, es importante que cuando se vaya a dejar de trabajar con Node-RED en local se pulse el botón "Salir de Node-RED", ya que en otro caso el contenedor seguiría corriendo y estaría consumiendo recursos en el PC

Apéndice B - Manual de Despliegue

En este apéndice se va a comentar el proceso que se ha llevado a cabo para desplegar los distintos componentes del sistema. Este apéndice podría servir de guía para desplegar el sistema en la plataforma de Google Cloud si se ha utilizado el código fuente del sistema.

Se va a mostrar paso a paso como se ha realizado el despliegue del sistema. Cabe destacar que se ha utilizado una base de datos MongoDB Atlas, que ya se encuentra desplegada en la nube, por lo que este componente será el único que no necesite desplegarse en Google Cloud.

B.1 Configuración de Google Cloud

Para poder desplegar un servicio en Google Cloud Run, se necesitan hacer algunas configuraciones en la plataforma de Google.

Lo primero que hay que hacer es crear la cuenta de usuario y elegir la facturación. Después se crea un proyecto:

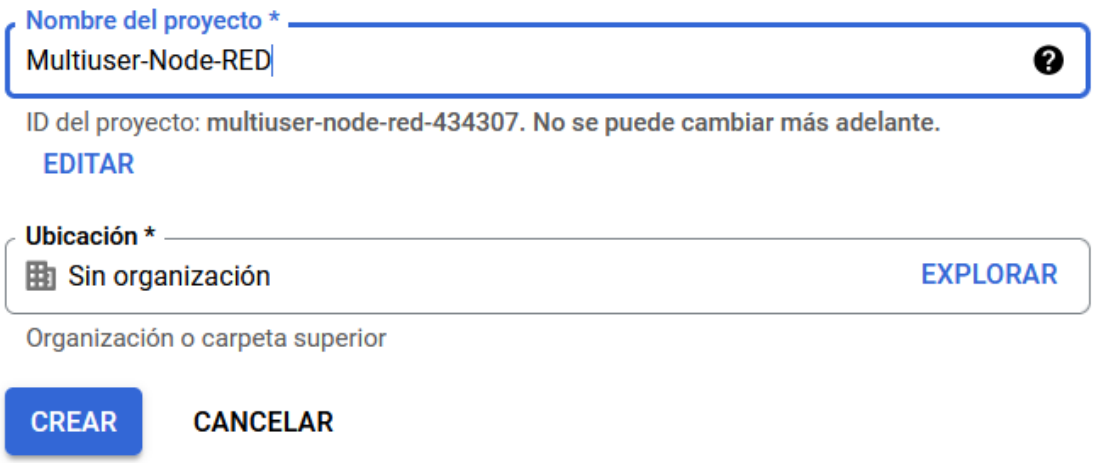


Figura B.1.1 Creación de un proyecto

Se habilitan las API y servicios que se van a utilizar:

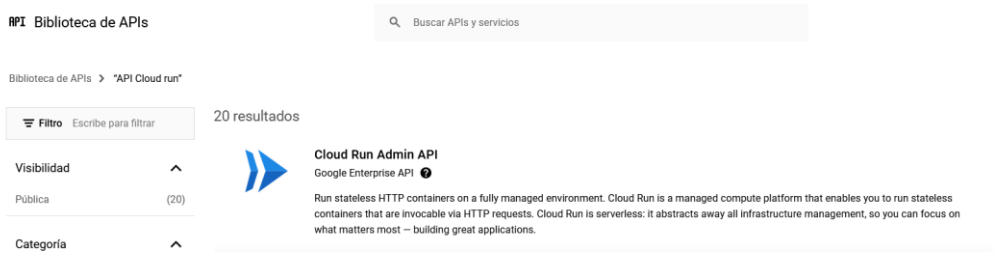


Figura B.1.2 Biblioteca de APIs

Para poder desplegar aplicaciones existen dos opciones. La primera consiste en desplegar desde la terminal de la maquina local y la segunda desde el código. Será necesario configurar las dos porque se utilizarán más adelante

Para poder desplegar aplicaciones desde la terminal se necesita Instalar Google Cloud SDK, con eso ya podremos utilizar los comandos para interactuar con Google Cloud. Después de instalarlo, se necesita ejecutar el comando gcloud init y tendremos que iniciar sesión e indicar el proyecto en el que se va a trabajar, el cuál creamos anteriormente.

Para poder desplegar servicios desde el código, hay que crear una cuenta de servicio y otorgarle permisos de administrador. Esto generará unas credenciales que se tendrán que usar en el código para poder usar la API de Google Cloud Run. Se verá mas detalladamente a continuación.

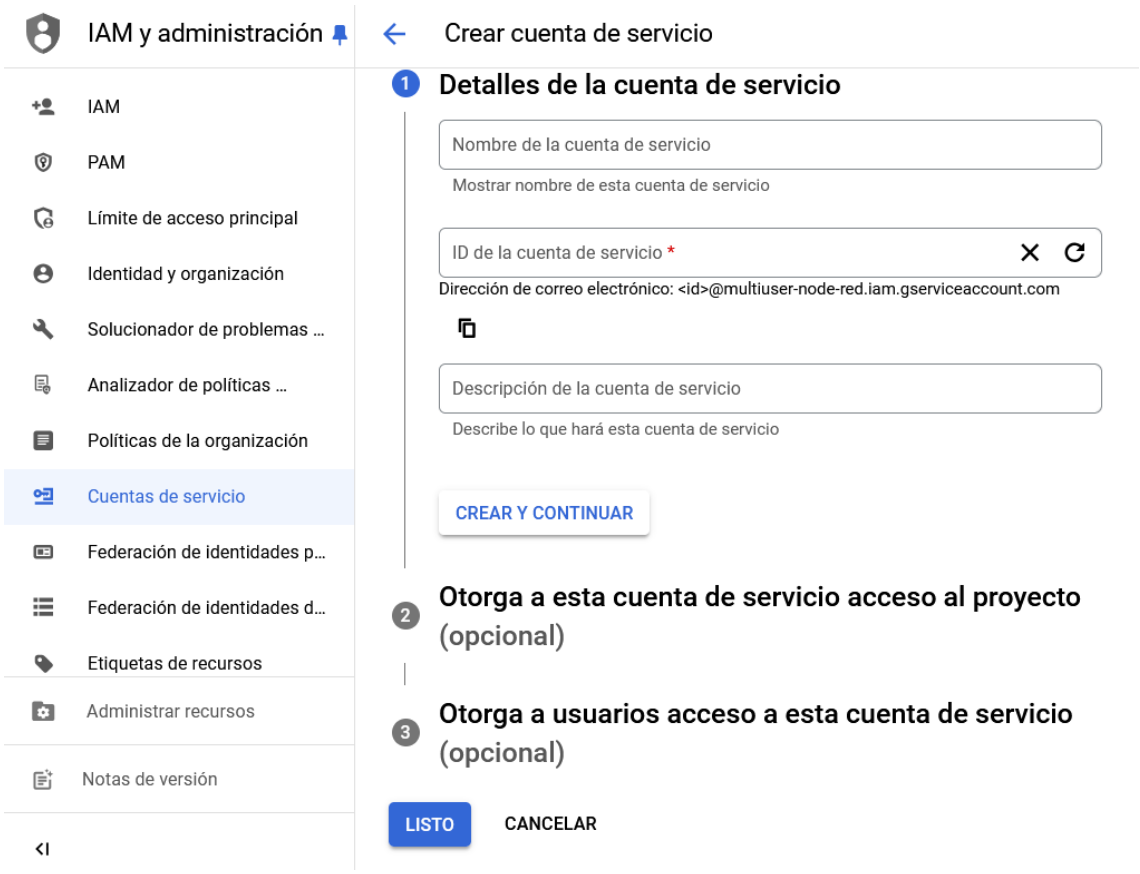
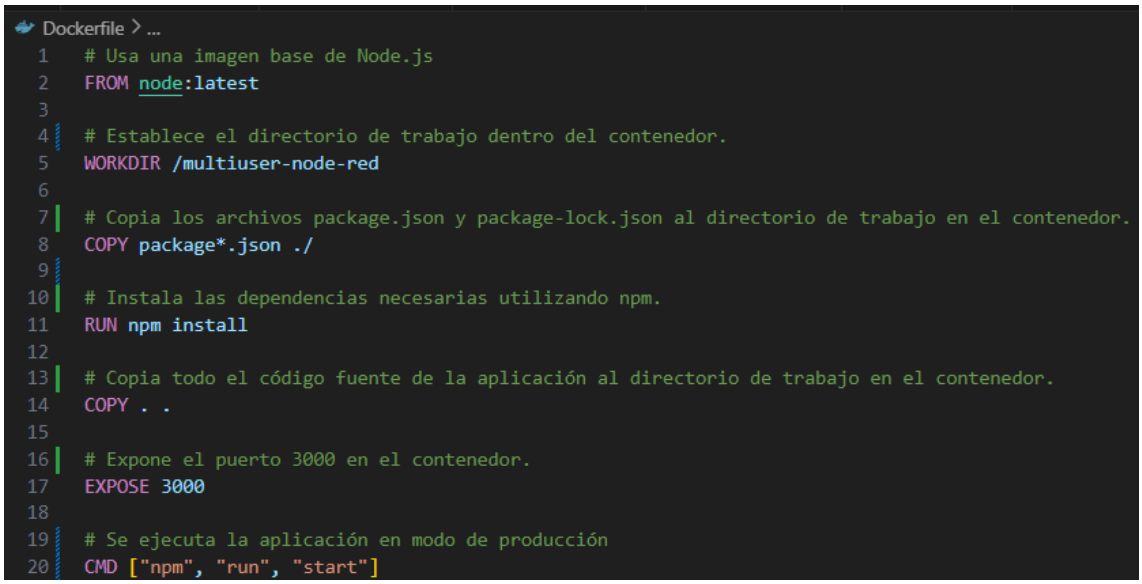


Figura B.1.3 Creación cuenta de servicio

B.2 Despliegue del Backend

Para desplegar el Backend, hay que realizar varios pasos.

En primer lugar, se necesita dockerizar el backend. Para ello se debe crear una Imagen de docker:

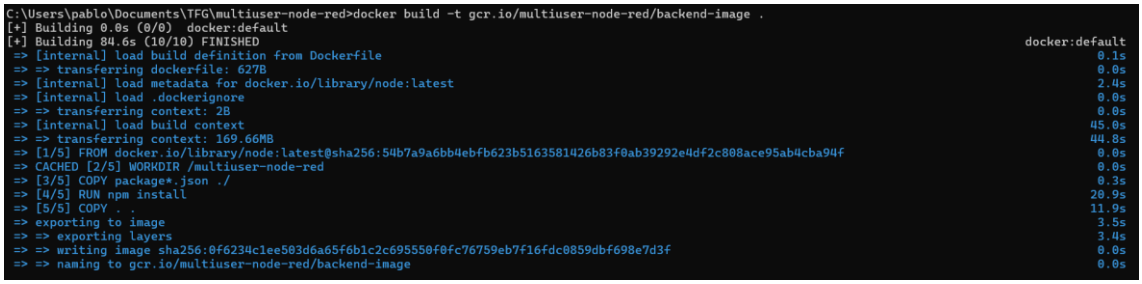


```dockerfile
# Usa una imagen base de Node.js
FROM node:latest

# Establece el directorio de trabajo dentro del contenedor.
WORKDIR /multiuser-node-red

# Copia los archivos package.json y package-lock.json al directorio de trabajo en el contenedor.
COPY package*.json ./

# Instala las dependencias necesarias utilizando npm.
RUN npm install

# Copia todo el código fuente de la aplicación al directorio de trabajo en el contenedor.
COPY . .

# Expone el puerto 3000 en el contenedor.
EXPOSE 3000

# Se ejecuta la aplicación en modo de producción
CMD ["npm", "run", "start"]
```

Figura B.2.1 Dockerfile del Backend

Una vez que se ha definido la Imagen, hay que construir el contenedor Docker utilizando los comandos desde la terminal. Hay que tener en cuenta que para poder construir una Imagen se necesita tener en ejecución la aplicación de Docker Desktop, esto es porque Docker Desktop proporciona el Docker Engine necesario para construir Imágenes.

Ahora, para poder construir la imagen hay que ubicarse en la carpeta donde se encuentra el Dockerfile y se ejecuta el siguiente comando:



```
C:\Users\pablo\Documents\TFG\multiuser-node-red>docker build -t gcr.io/multiuser-node-red/backend-image .
[+] Building 0.0s (0/0)  docker:default
[+] Building 84.6s (10/10) FINISHED                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                          0.1s
 => => transferring dockerfile: 627B                                                                          0.0s
 => [internal] load metadata for docker.io/library/node:latest                                                2.4s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 2B                                                                               0.0s
 => [internal] load build context                                                                            45.0s
 => => transferring context: 169.66MB                                                                        44.8s
 => [1/5] FROM docker.io/library/node:latest@sha256:54b7a9a6bb4ebfb623b5163581426b83f0ab39292e4df2c808ace95ab4cba94f  0.0s
 => CACHED [2/5] WORKDIR /multiuser-node-red                                                                  0.0s
 => [3/5] COPY package*.json ./                                                                               0.3s
 => [4/5] RUN npm install                                                                                    20.9s
 => [5/5] COPY . .                                                                                           11.9s
 => exporting to image                                                                                        3.5s
 => => exporting layers                                                                                       3.4s
 => => writing image sha256:0f6234c1ee503d6a65f6b1c2c695550f0fc76759eb7f16fdc0859dbf698e7d3f                  0.0s
 => => naming to gcr.io/multiuser-node-red/backend-image                                                      0.0s
```

Figura B.2.2: Construir el Docker del Backend

Como se puede observar, hay que Indicar el nombre del proyecto y el nombre que se desea que tenga el contenedor.

Una vez que se ha construido el Docker en nuestro local, hay que ejecutar el comando push para desplegar el contenedor a Google Cloud.

```
C:\Users\pablo\Documents\TFG\multiuser-node-red>docker push gcr.io/multiuser-node-red/backend-image
Using default tag: latest
The push refers to repository [gcr.io/multiuser-node-red/backend-image]
41e349cb1eaa: Pushed
07936f84a00a: Pushed
25a595efa9e7: Pushed
c76bc7f60ab9: Layer already exists
fdf5ce66ec59: Layer already exists
441cf1c442f5: Layer already exists
42073a4c3c76: Layer already exists
7aaeaeabc9cf: Layer already exists
28e03088bc15: Layer already exists
0d80db6a0977: Layer already exists
916d866d5b0d: Layer already exists
8f4ceb8cc1a2: Layer already exists
latest: digest: sha256:af7f1fd92401dbccc9e030dc5870e67c0a62ca26382ccbc3a135cfa3aba7dfe3 size: 2844
```

Figura B.2.3: Hacer push del contenedor

Para comprobar que el contenedor está desplegado en Google Cloud hay que irse al apartado de Artifact registry en Google Cloud:

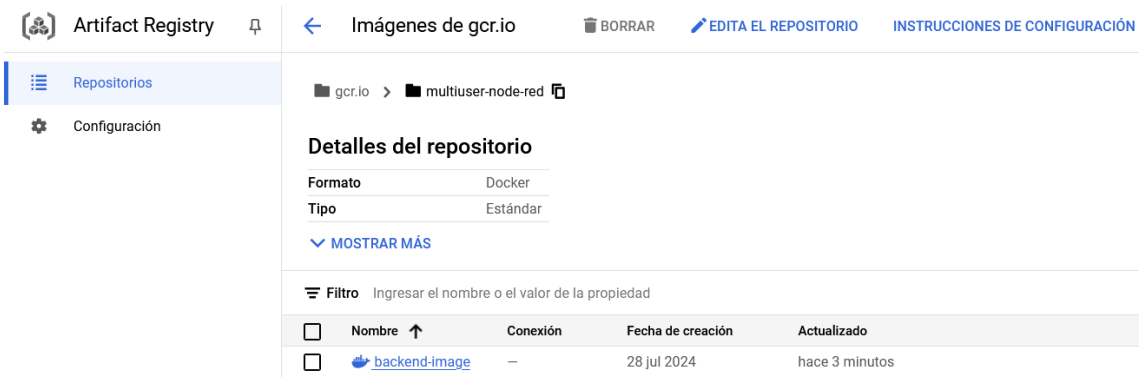


Figura B.2.4: Artifact Registry

O también se puede comprobar desde la terminal:

```
C:\Users\pablo\Documents\TFG\multiuser-node-red>gcloud container images list-tags gcr.io/multiuser-node-red/backend-image
DIGEST        TAGS    TIMESTAMP
af7f1fd92401  latest  2024-09-01T10:45:38
```

Figura B.2.5: Comprobar imagen desde la terminal

Cuando se tenga el contenedor desplegado en Google Cloud, el último paso es desplegar un servicio para ejecutar el contenedor. Para ello se ha utilizado Google Cloud Run, ya que permite ejecutar contenedores sin necesidad de que se gestione la infraestructura subyacente.

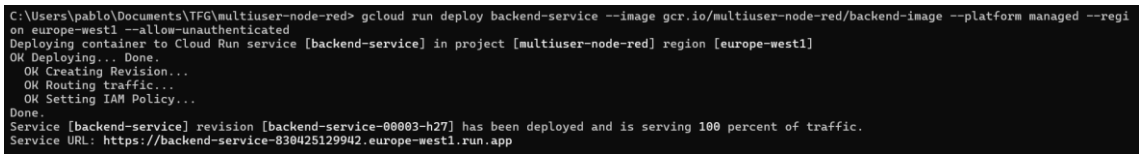


Figura B.2.6: Despliegue del contenedor del Backend

Ya se puede comprobar que el servicio está desplegado desde Google Cloud Run:

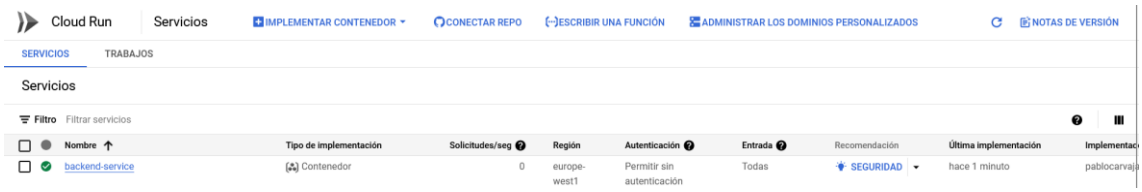


Figura B.2.7: Google Cloud Run

Ya está el servicio completamente desplegado y listo para usar.

B.3 Despliegue del Frontend

Para desplegar el Frontend se debe seguir exactamente los mismos pasos que se han hecho para desplegar el Backend, por lo tanto, no se va a explicar de nuevo. La única diferencia rádica en el Dockerfile:

```
# Usa una imagen base de Node.js
FROM node:latest

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /vm-node-red

# Copia los archivos package.json y package-lock.json al directorio de trabajo en el contenedor
COPY package*.json ./

# Instala las dependencias de la aplicación
RUN npm install

# Copia todo el código fuente de la aplicación al directorio de trabajo en el contenedor
COPY . .

# Construye la aplicación de React
RUN npm run build

# Instala el paquete `serve` globalmente para servir archivos estáticos
RUN npm install -g serve

# Expone el puerto 8080 en el contenedor
EXPOSE 8080

# Define el comando por defecto para ejecutar cuando el contenedor se inicie
CMD ["serve", "-s", "dist", "-l", "8080"]
```

Figura B.3.1: Dockerfile del Frontend

Aunque este Dockerfile sigue una estructura muy similar al del Backend, la diferencia está en que también se Instala el paquete Serve, cuya utilidad es servir los archivos estáticos generados durante el proceso de construcción, sin necesidad de configurar un servidor web más complejo.

B.4 Despliegue de las Instancias de Node-RED

Por último, se necesitan desplegar las Instancias de Node-RED. Aquí se encuentra la parte más compleja de la fase de despliegue, ya que no se trata simplemente de desplegar un servicio como se ha hecho con los demás componentes, sino que se debe desplegar un servicio por cada usuario para garantizar que tenga una instancia Independiente de Node-RED. Las instancias se deben desplegar de forma dinámica una vez que el usuario se registre y quiera acceder a Node-RED, por lo que no se puede realizar de forma manual como hasta ahora.

Para llevar a cabo esto, primero se debe construir el contenedor de Node-RED y subirlo a Google Cloud de la misma forma que para los otros componentes.

Aquí está el Dockerfile para Node-RED, que sigue la misma estructura que los demás componentes:

```
# Usa una imagen base de Node.js
FROM node:latest

# Establece el directorio de trabajo dentro del contenedor
WORKDIR /node-red-modified

# Copia los archivos package.json y package-lock.json al directorio de trabajo en el contenedor
COPY package*.json ./

# Instala las dependencias del proyecto especificadas en package.json
RUN npm install --legacy-peer-deps

# Copia todo el código fuente de la aplicación al directorio de trabajo en el contenedor
COPY . .

# Ejecuta el script de construcción del proyecto
RUN npm run build

# Expone el puerto 1880 en el contenedor
EXPOSE 1880

# Define el comando por defecto para ejecutar cuando el contenedor se inicie
CMD ["npm", "start"]
```

Figura B.4.1: dockerfile de Node-RED

Una vez construido el contenedor Docker y subido a Google Cloud, no debemos desplegar el contenedor manualmente como se ha hecho con los demás componentes, sino que será el propio Backend el que en tiempo de ejecución se encargue de desplegar las distintas instancias de Node-RED cuando los usuarios lo soliciten. Este proceso ya se ha explicado en el punto 7. Despliegue.

Apéndice C - Manual de Usuario

En este apéndice se presenta una guía de cómo se utiliza la aplicación.

Guía para el Rol de alumno

Aquí se muestra la primera vista que se ve al abrir la aplicación

Figura C.1: Página principal de la aplicación

Después de darle al botón registrarse se nos abre la vista siguiente:

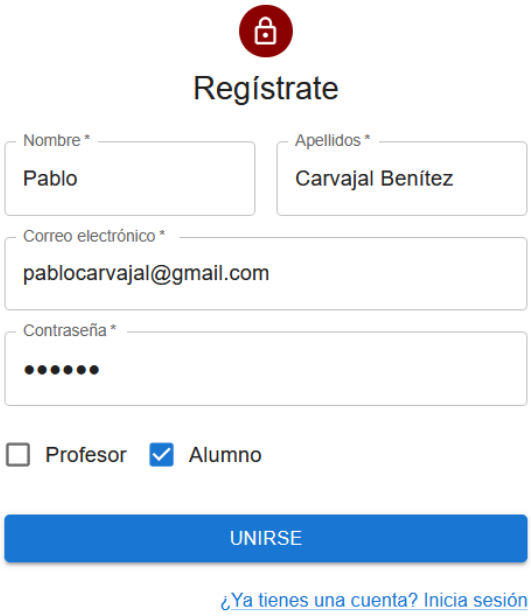


Figura C.2: Página de registro

Se rellenan los datos y se elige el rol de alumno. Cuando se pulse el botón de unirse se redirigirá a la página de inicio de sesión.

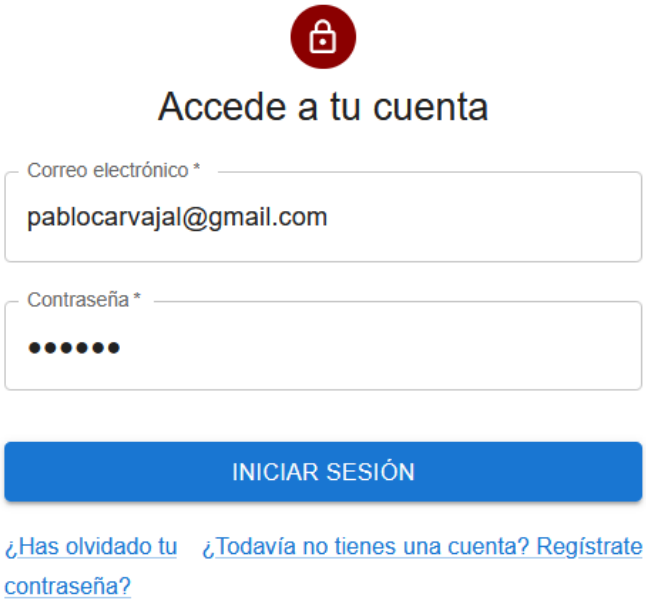


Figura C.3: Página de inicio de sesión

Ahora se puede ver la página principal para los alumnos, se puede ver arriba a la derecha un icono indicando nuestro rol. Además, también se puede ver que está la opción de unirse a una

clase o de acceder a una clase a la que ya se ha unido. También vemos que es posible la opción común tanto para alumnos como profesores de Node-RED de ver el perfil.

Figura C.4: Página principal de un profesor

Si se hace click en ver Perfil se podrán editar los datos

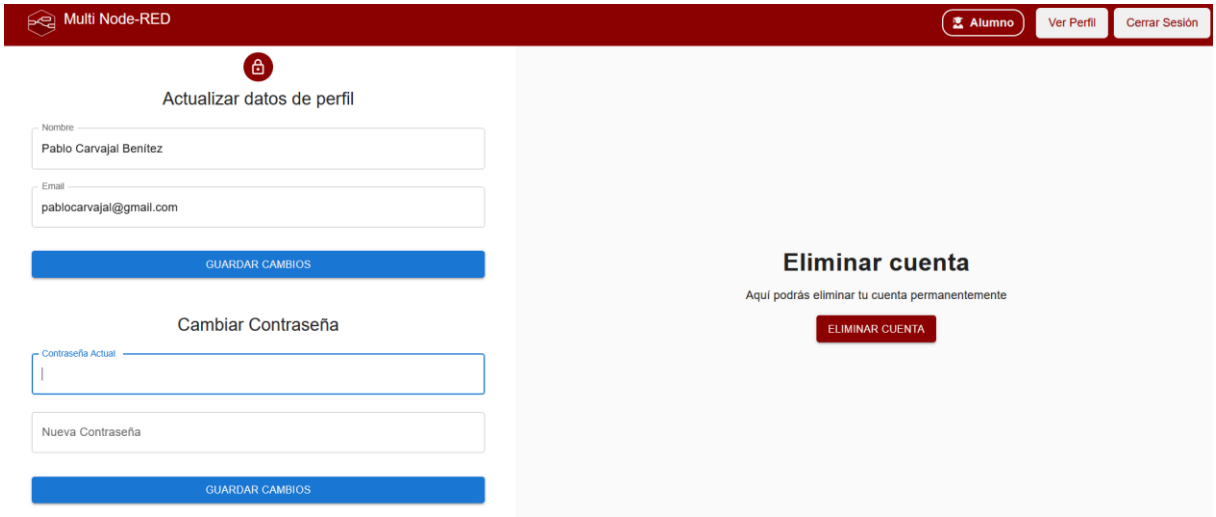


Figura C.5: Editar perfil

Haciendo click en el logo se puede volver a la página principal. Ahora se va a entrar ya a Node-RED para empezar a trabajar.

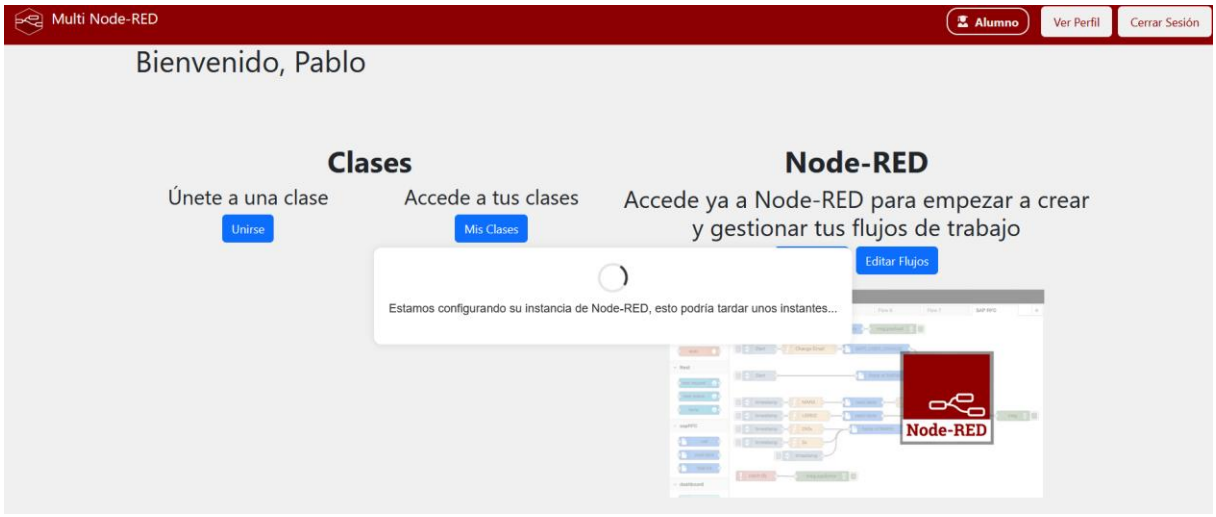


Figura C.6: Acceder a Node-RED

Se puede ver que se está configurando la instancia de Node-RED, la primera vez que se hace puede tardar un poco más, en torno a los 20 segundos, sin embargo, la próxima vez que se acceda ya será instantáneo.

Se redirige a Node-RED y ya se puede empezar a trabajar.

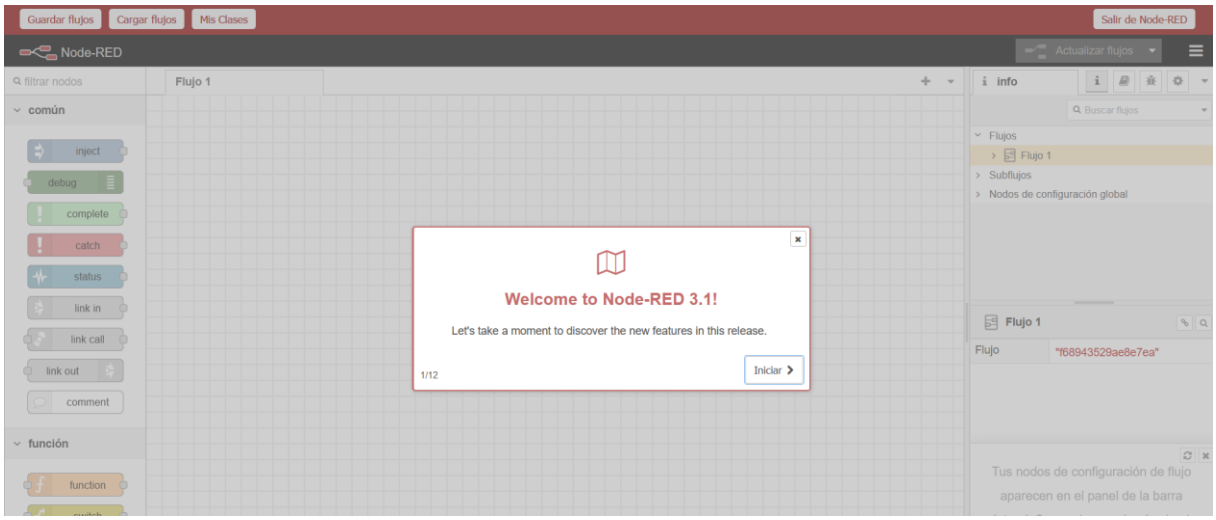


Figura C.7: Página de inicio de Node-RED

Ahora se pueden crear algunos flujos de trabajo:

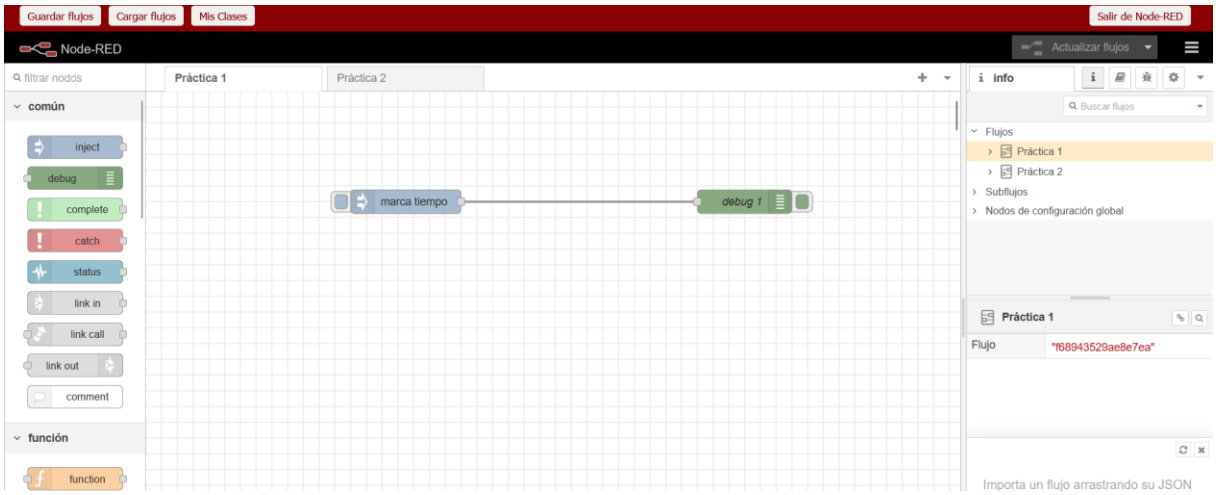


Figura C.8: Crear flujos de trabajo

Se ha creado dos flujos, ahora se pueden guardar en la base de datos del sistema.

Se va a guardar solo el de Práctica 1

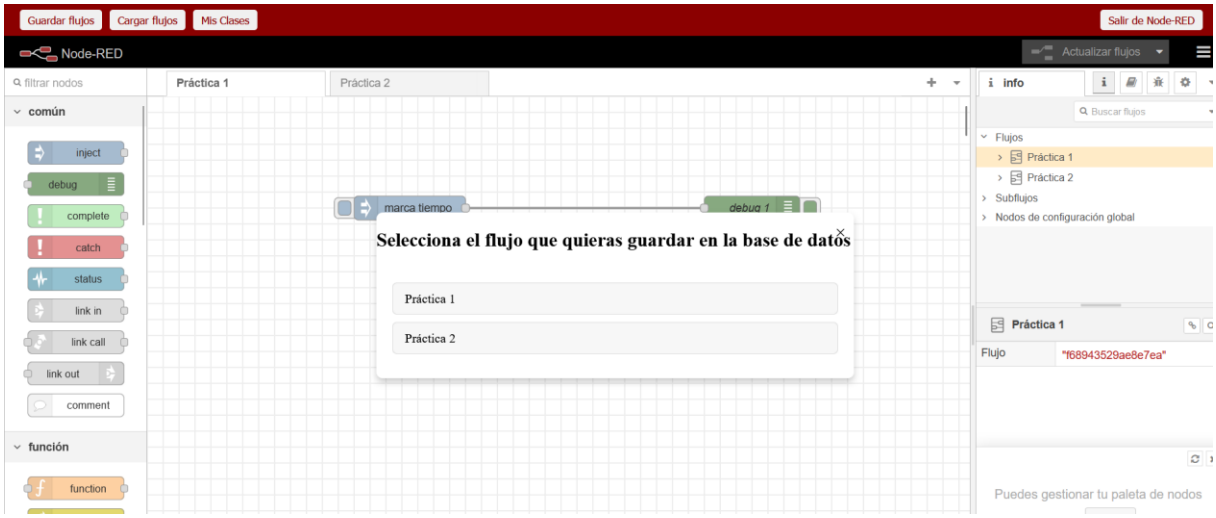


Figura C.9: Guardar flujos

Ahora que se ha guardado el flujo, se puede salir de Node-RED y volver a entrar. Si se pulsa cargar flujos aparece el flujo Práctica 1 que se guardó anteriormente.

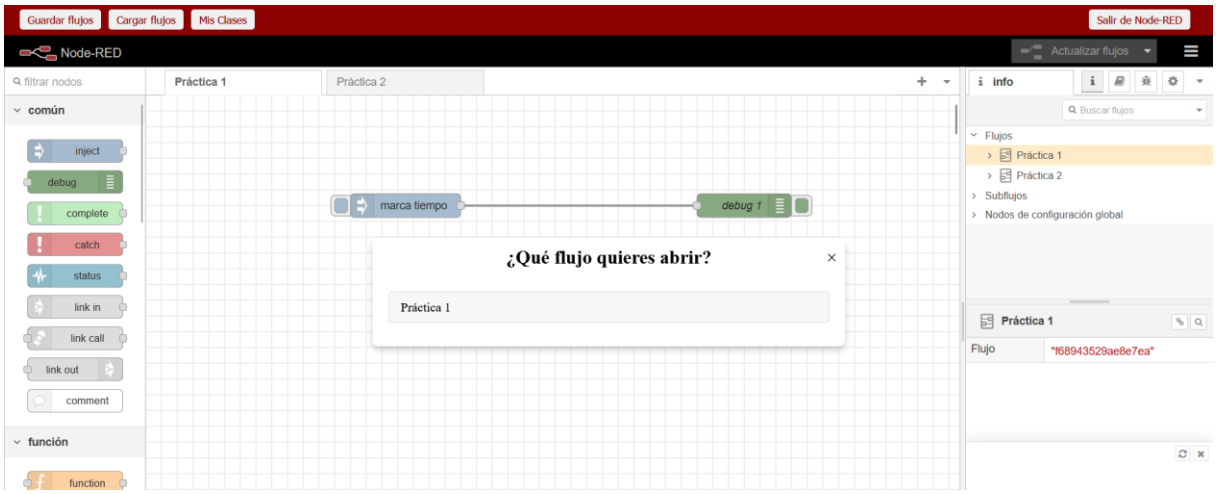


Figura C.10: Cargar flujo

Si se selecciona el flujo se cargará en pantalla.

Figura C.11: Revisar los cambios

Advierte Node-RED que los flujos han sido actualizados, hay que pulsar revisar cambios. Esto se hace por si ya se tiene ese mismo flujo cargado en Node-RED para que no haya conflictos.

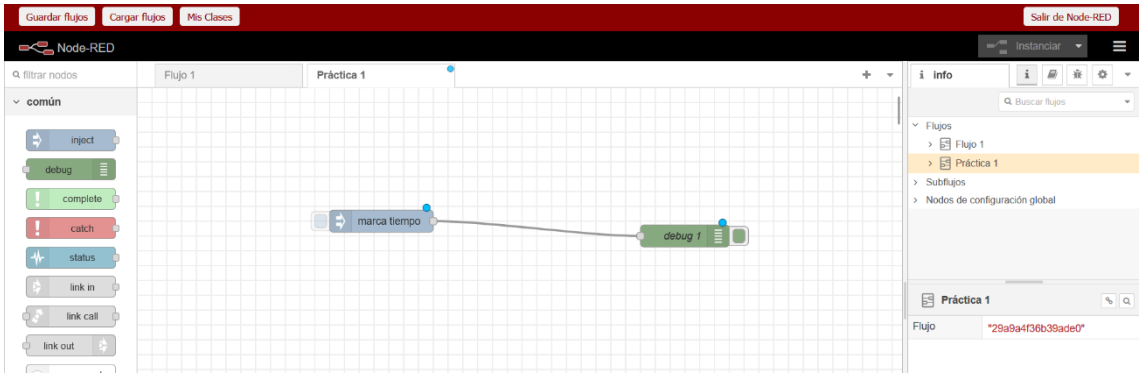


Figura C.12: Flujo cargado con éxito

Ahora, se va a seguir trabajando con el flujo cargado y se van a cambiar algunos nodos.

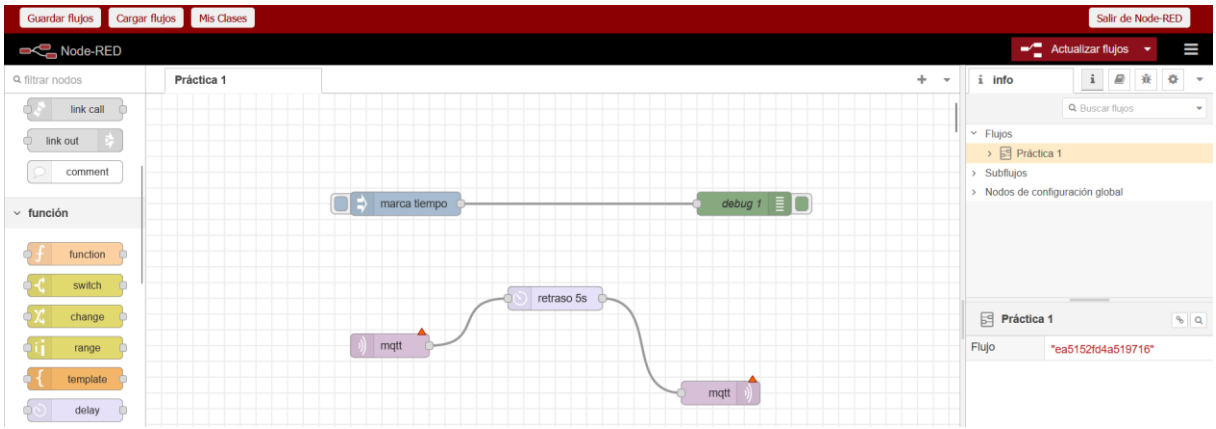


Figura C.13: Flujo cargado con éxito

Ahora que se ha cambiado el flujo cargado, se quiere guardar con el mismo nombre para reflejar los cambios.

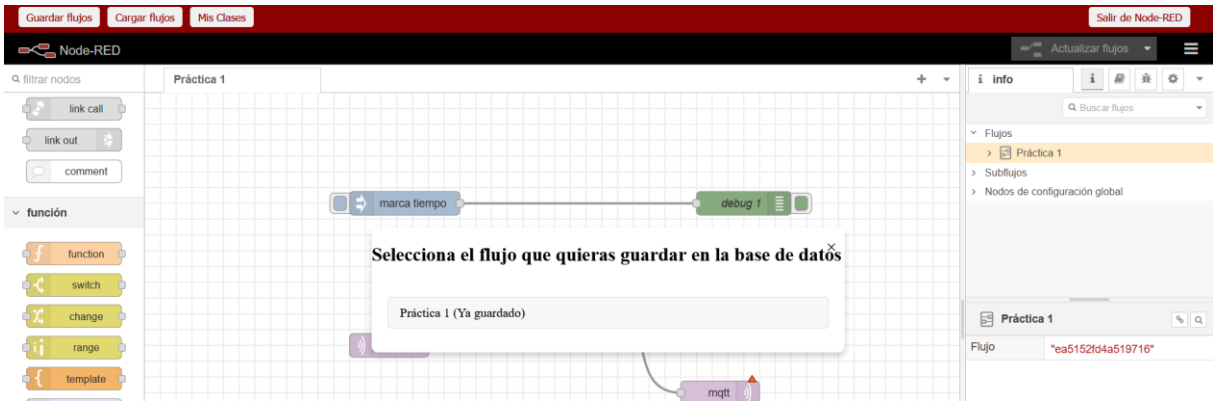


Figura C.14: Flujo cargado con éxito

Se hace click en guardar el flujo, pero un mensaje indica que ese flujo ya existe, por lo tanto, se va a sobrescribir. Como se quieren actualizar los nuevos cambios, se pulsa aceptar y se actualizará el flujo con los nuevos cambios

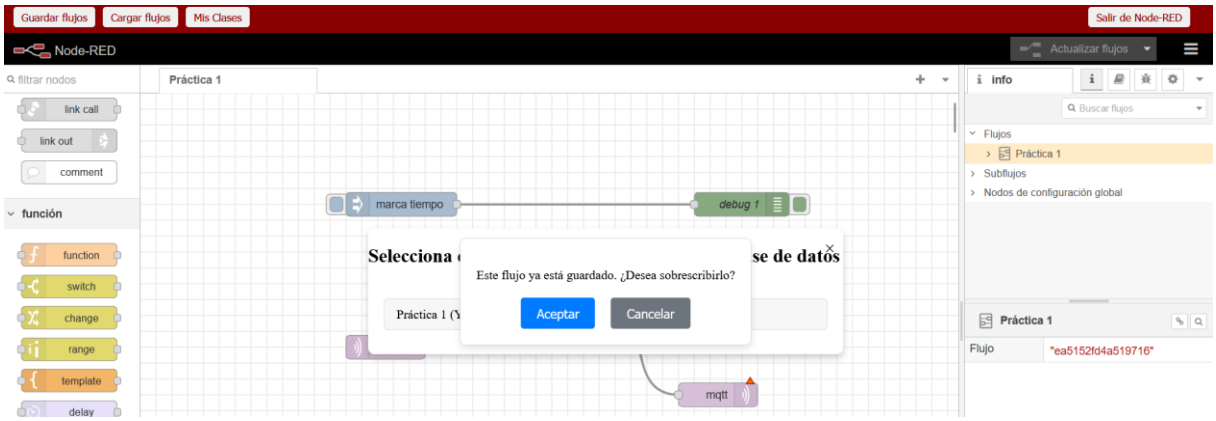


Figura ÇC.15: Flujo cargado con éxito

Ya se ha trabajado con Node-RED, pero aún no se ha unido a ninguna clase. Se ha recibido un correo de un profesor como invitación a una clase.

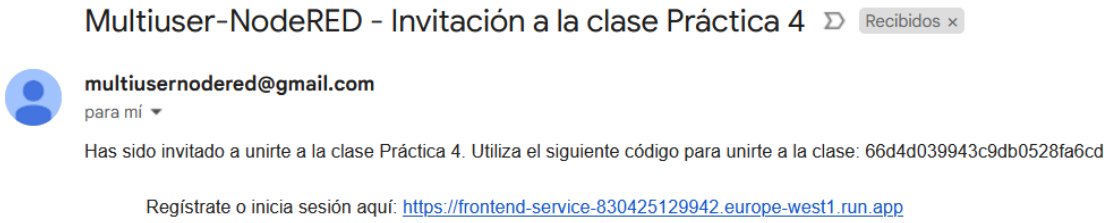


Figura C.16: Correo electrónico de invitación

Se copia el código, se regresa de nuevo la página principal y se usa para unirse a la clase.

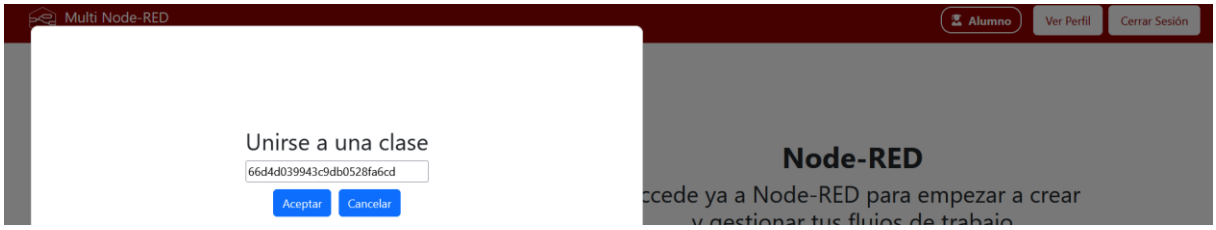


Figura C.17: Unirse a una clase

Ya se ha unido, así que ya se puede ver la clase en la lista de clases y se puede entrar.

Figura C.18: Página de clases

Se puede ver las opciones que dejan en una clase, entregar un flujo y ver los flujos subidos por el profesor.

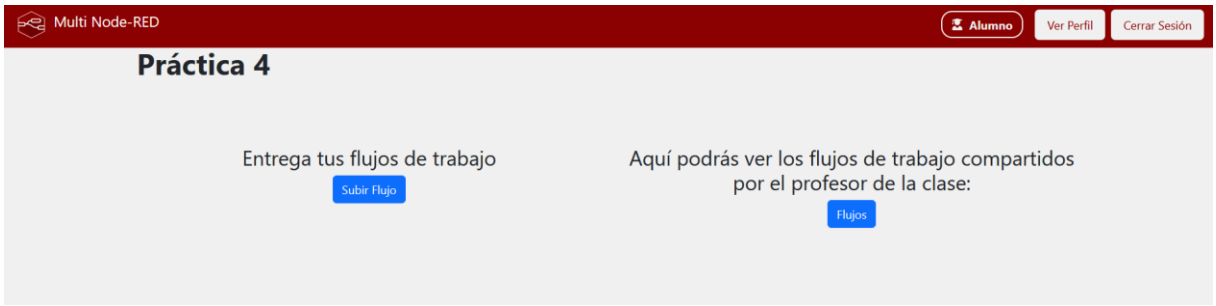


Figura C.19: Detalles de una clase

Se pueden ver los flujos que ha subido el profesor hasta ahora en la clase.

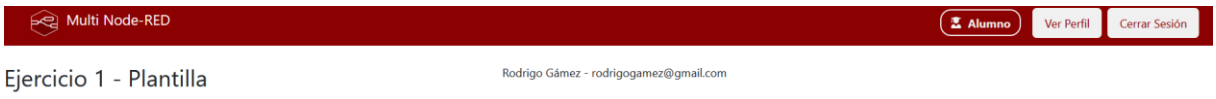


Figura C.20: Flujos del profesor en la clase

Y en el apartado de subir flujo, se puede subir el flujo que se ha creado antes en Node-RED.

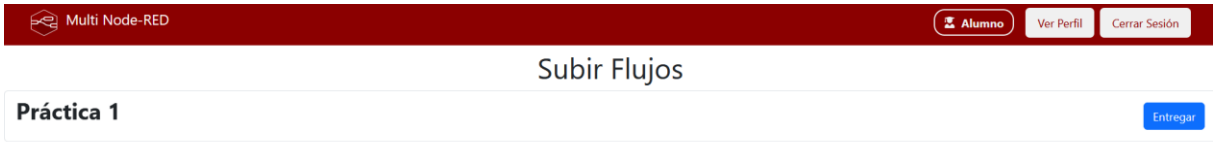


Figura C.21: Subir un flujo a una clase

Ahora que se ha unido a una clase, se puede volver a Node-RED para utilizar los flujos de trabajos compartidos por el profesor de la clase.

Si se hace click en el botón de mis clases, aparece el listado con las clases a las que se ha unido.

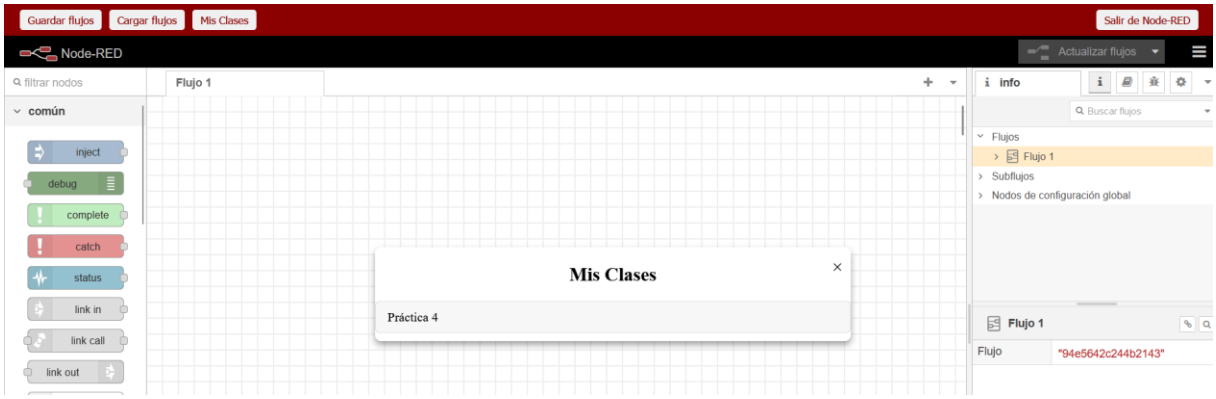


Figura C.22: Página de clases en Node-RED

Y si se hace click en la clase, nos aparece los flujos de trabajo que ha subido el profesor.

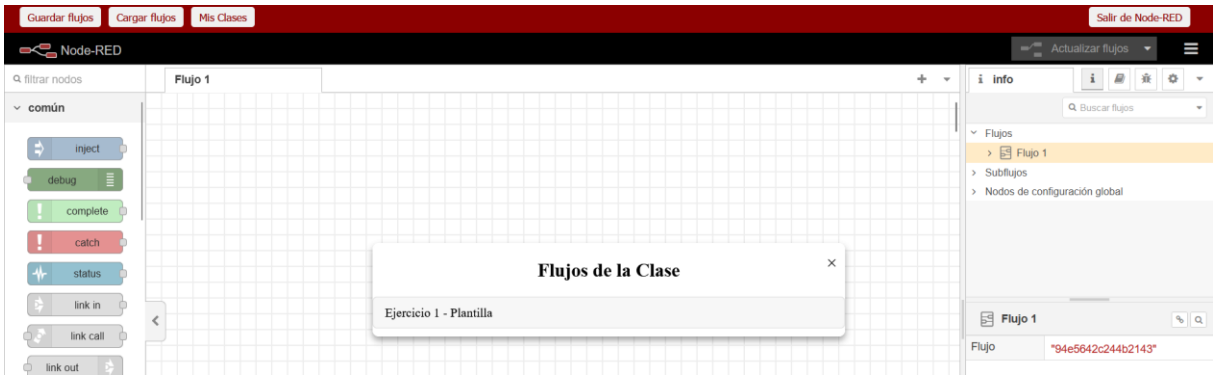


Figura C.23: Flujos de una clase en Node--RED

Se hace click y se carga correctamente en la pantalla de Node-RED para que se pueda trabajar con él.

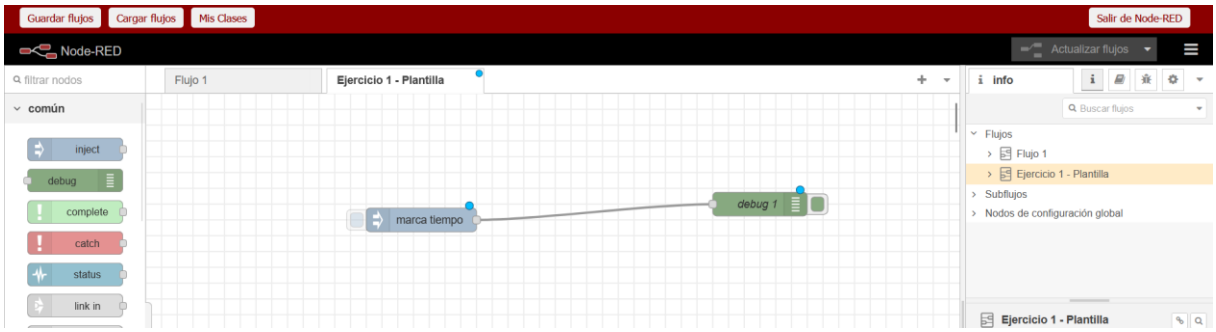


Figura C.24: Cargar un flujo desde una clase

Ya se ha visto todas las funcionalidades para el rol de usuario. Pero antes se va a ver una funcionalidad que es la de recuperar contraseña.

Se cierra sesión y se supone que se ha olvidado la contraseña. se pulsa el enlace de ¿Has olvidado tu contraseña?

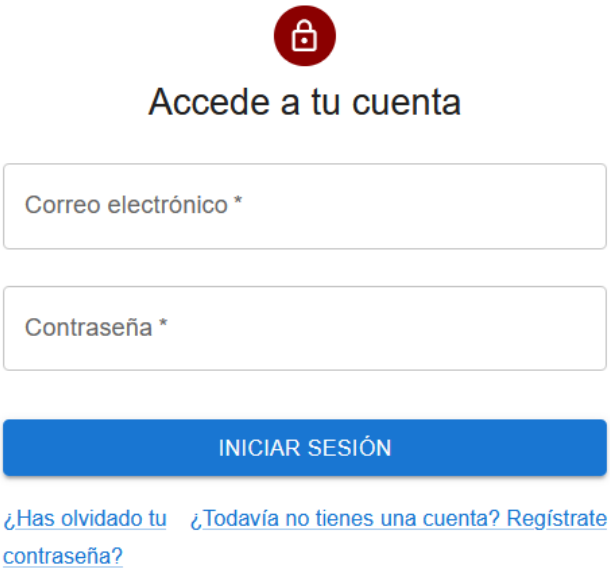


Figura C.25: Inicio de sesión

Ahora se abrirá un formulario donde se tendrá que poner el correo electrónico de recuperación, que debe ser el mismo con el que se ha registrado.

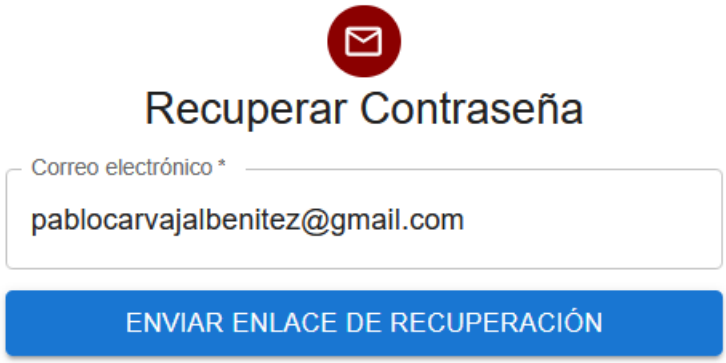


Figura C.26: Solicitar enlace de recuperación

Ahora llegará un correo electrónico

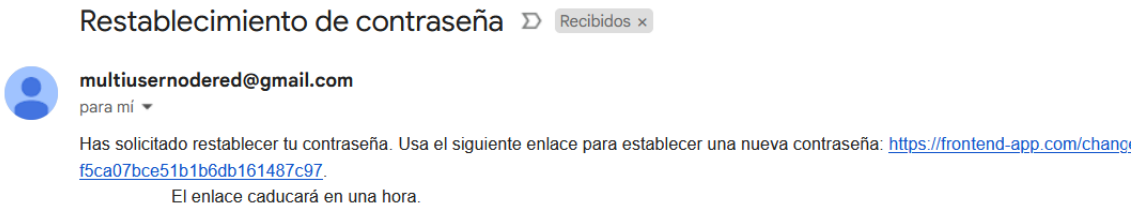


Figura C.27: Correo electrónico de recuperación

Y si se hace click al enlace redirigirá al formulario para poner una nueva contraseña

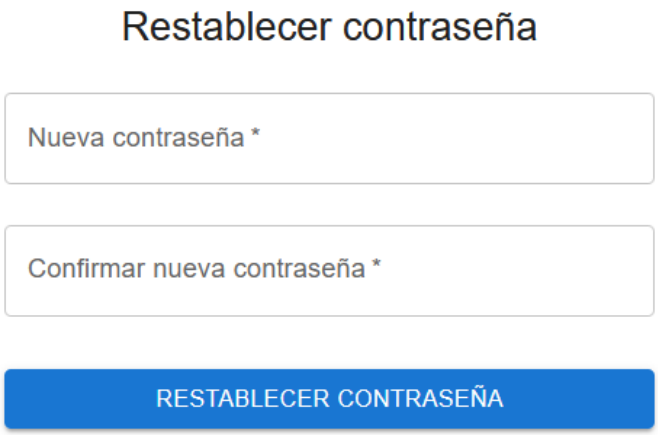


Figura C.28: Formulario de restablecimiento de la contraseña

Se cambia la contraseña y ya se puede usar esa a partir de ahora.

Guía para el Rol de profesor

Se accede a una cuenta con el rol de profesor. Se puede ver que el icono del rol es diferente, al igual que la parte de las clases.

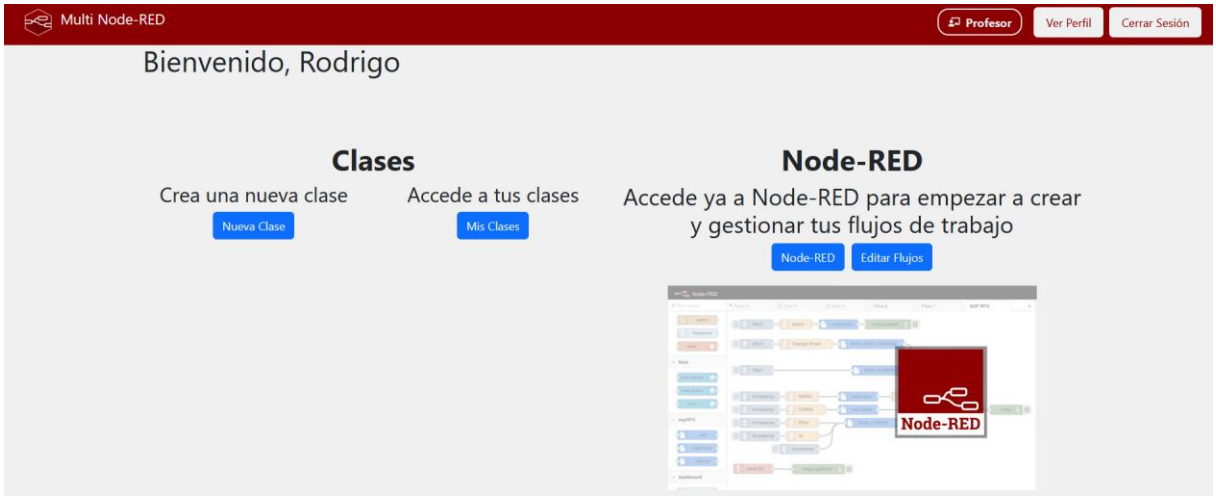


Figura C.29: Página de inicio para un profesor

Antes de mostrar las funcionalidades específicas para el rol de profesor, se va a enseñar la página de Editar flujos, que es común para ambos roles.

Figura C.30: Página de Editar Flujos

Si se pulsa el botón de "Cambiar nombre" se llevará al usuario al formulario para cambiar el nombre

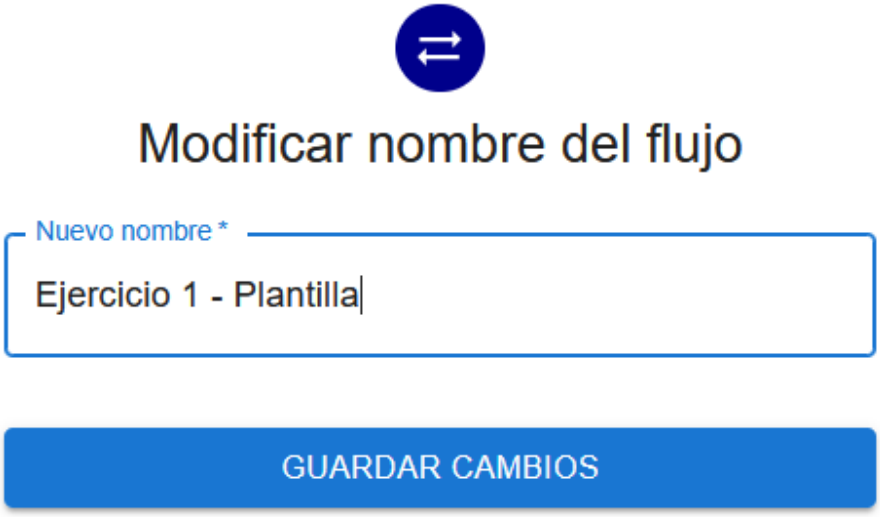


Figura C.31: Cambiar nombre de un flujo

Ahora, volviendo a la página de inicio y pulsando el botón "Mis clases" Se puede ver que ya hay una clase creada

Figura C.32: Clases del profesor

Ahora, se va a volver a la página de inicio y se va a crear una nueva clase

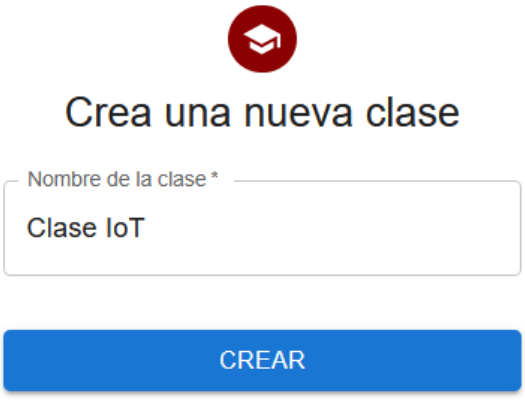


Figura C.33: Creación de una nueva clase

Y se confirma en la misma página de antes que la clase se ha creado.

Figura C.34: Clases del profesor

Se accede a la clase Práctica 4, previamente creada.

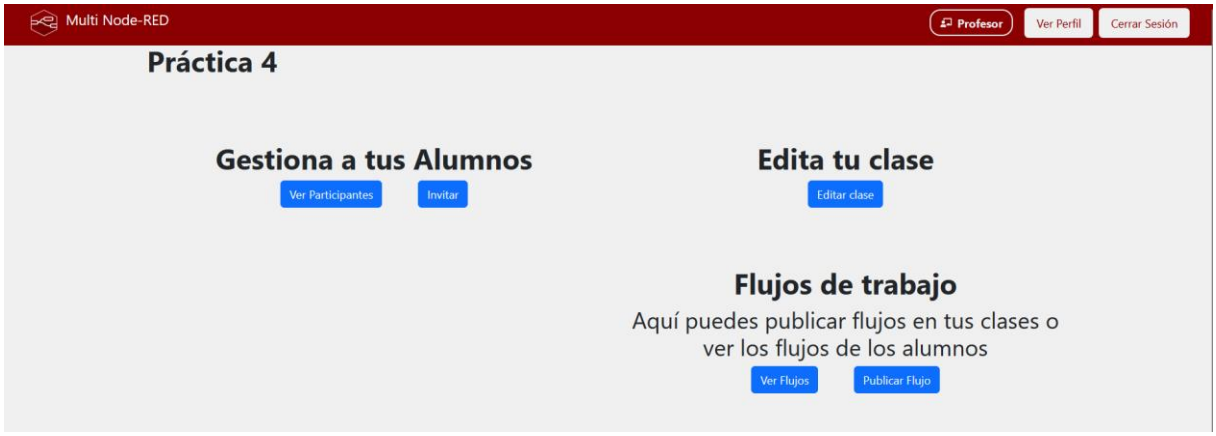


Figura C.35: Detalles de una clase del profesor

Y se ven las opciones que aparecen, que son diferentes en comparación con las del rol de alumno.

hay varias opciones, se puede mirar primero los participantes que hay en la clase.

Figura C.36: Participantes de una clase

Solo hay un alumno de momento, pero se puede invitar a más alumnos si se quiere, solo se necesita poner los correos electrónicos.

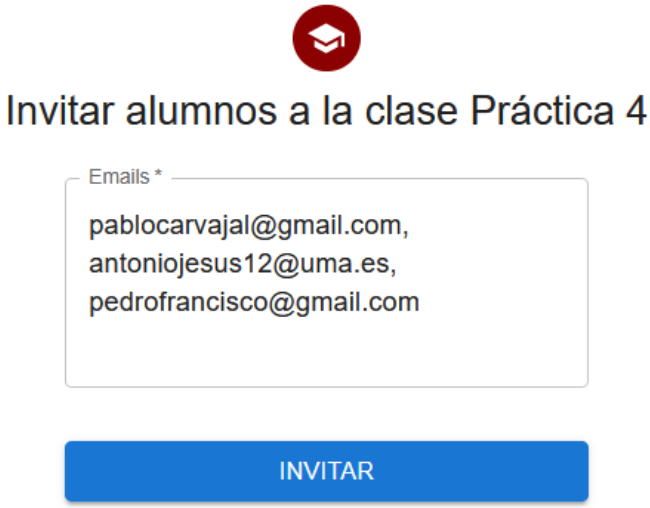


Figura C.37: Invitar alumnos de una clase

También se puede editar una clase y cambiar su nombre.

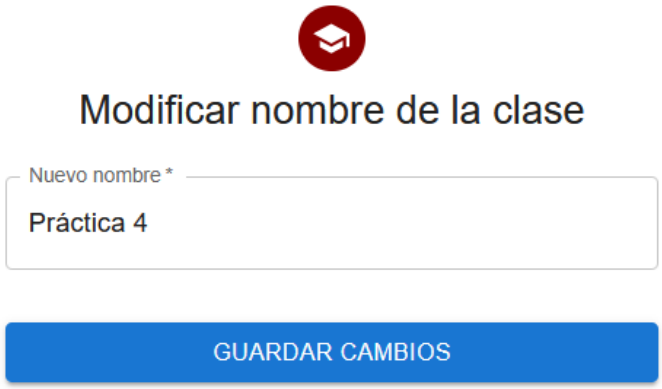


Figura C.38: Editar una clase

En la parte de los flujos, se puede publicar un flujo para que los alumnos puedan usarlos como plantilla.

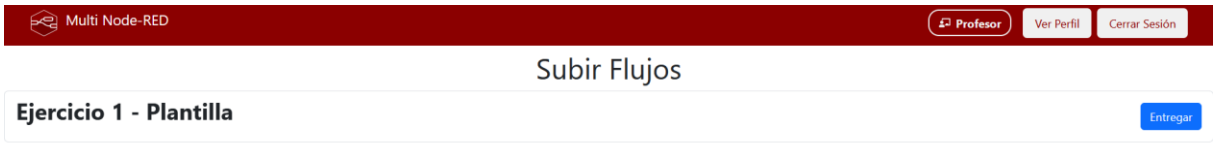


Figura C.39: Subir flujos a una clase

También se pueden ver los flujos que hay subidos ahora mismo en la clase.

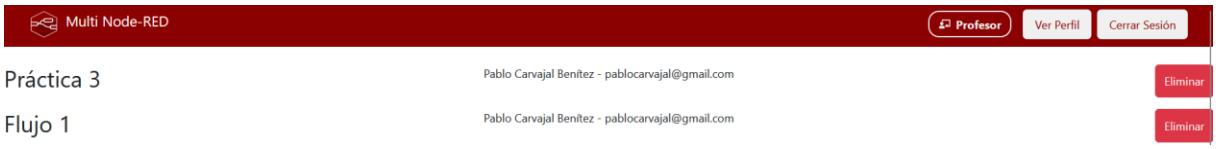


Figura C.40: Flujos de una clase

Vemos que está el de plantilla que ha subido el profesor y el ejercicio Práctica 1 que se ha entregado antes con el alumno.

Las funcionalidades de Node-RED son las mismas para alumnos y profesores, con la diferencia que cuando se quiere cargar los flujos de una clase, los alumnos solo pueden ver los subidos por el profesor mientras que el profesor puede ver los flujos subidos por todos los participantes de la clase.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga