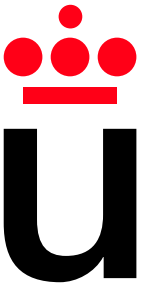

Universidad
Rey Juan Carlos

Escuela Técnica Superior
de Ingeniería Informática

Grado en Ingeniería Informática

Curso 2022-2023

Trabajo Fin de Grado

OPENVPN4ALL: SUITE PARA LA CREACIÓN Y GESTIÓN DE UN SERVIDOR VPN

Autor: Álvaro Berdote Jiménez
Tutor: Vicente Belinchón González
Cotutor: Michel Maes Bermejo

Agradecimientos

Gracias a mis padres por apoyarme siempre en todas las decisiones que he tomado y guiarme con sus consejos.

Agradecer a mi familia por ayudarme de varias formas posibles en mi etapa como estudiante.

También gracias a Iván Vicente León Fernández, fue mi profesor en el grado medio y seguramente si no fuera por él no hubiera tomado la decisión de entrar en la universidad. No me olvido de Susana, mi profesora en el grado superior que despertó mis ganas de iniciarme en el mundo del desarrollo.

He tenido docentes magníficos en este grado y me gustaría dedicaros el siguiente párrafo a todos vosotros. Gracias a los docentes de la universidad que con su pasión y entusiasmo por enseñar, transmiten sus conocimientos y dan alas a los estudiantes para seguir aprendiendo y mejorar.

Por último y no menos importante, agradecimientos a Vicente Belinchón González y Michel Maes Bermejo por ayudarme a lo largo del proyecto y aportar ideas para mejorarlo, no podría haber tenido mejores tutores para este proyecto.

Resumen

OPENVPN4ALL es una aplicación web que permite crear un servidor VPN de forma sencilla y sin tener conocimientos avanzados de networking ni de informática.

La idea inicial es hacer uso de *openvpn* [1] y gestionar la configuración de ficheros y certificados de forma automática mediante scripts, haciendo uso de una interfaz gráfica facilitando así enormemente la tarea de crear un servidor VPN a un usuario inexperto.

Esta herramienta puede ser muy útil para que usuarios no profesionales así como pequeñas y medianas empresas que no se pueden permitir un router con VPN incorporada o software de pago puedan disponer de un servidor totalmente funcional y fácil de gestionar.

Para ello nuestro objetivo es construir una interfaz que permita crear un usuario, poder generar un fichero de conexión para dicho usuario y poder eliminarlo. También se quiere permitir editar y poder cambiar fácilmente la configuración del servidor VPN así como disponer de un panel con información actualizada del servidor.

La solución propuesta incluye, tras un análisis de las tecnologías disponibles, las siguientes funcionalidades:

- Información en tiempo real del estado de la VPN y conexiones de usuarios.
- Lectura de logs del servidor en tiempo real.
- Gestión de usuarios con posibilidad de enviar el fichero de conexión por email.
- Configuración del servidor VPN con auto relleno en algunos campos.
- Configuración del cliente de correo interno del servidor VPN para enviar las conexiones de cliente.
- Desconexión y/o eliminación en tiempo real de usuarios.
- Descarga de logs del servidor.

El documento se compone de siete capítulos:

- **Capítulo 1 introducción**: pequeña introducción sobre el proyecto, su ámbito y los problemas encontrados.
- **Capítulo 2 objetivos**: objetivos planteados del proyecto, explicación de los mismos y resolución.
- **Capítulo 3 tecnologías, herramientas y metodologías**: tecnologías, herramientas y metodologías usadas en el proyecto con una breve introducción a las mismas.
- **Capítulo 4 descripción informática**: arquitectura y funcionamiento de los distintos componentes del proyecto.
- **Capítulo 5 pruebas**: demostración y explicación de las pruebas creadas en el proyecto y su organización.
- **Capítulo 6 distribución**: documentación de la dockerización del proyecto y problemas encontrados.
- **Capítulo 7 conclusiones y trabajos futuros**: conclusiones y posibles mejoras del proyecto a futuro.

Palabras clave:

- SpringBoot
- Sveltekit
- Bash scripting
- Java 11+
- TypeScript
- OpenVpn
- EasyRSA

Índice de contenidos

Índice de figuras

1 Introducción

Este capítulo argumenta la motivación a la hora de crear este proyecto y las problemáticas asociadas.

1.1. Motivación

Hoy en día está en boca de todos el teletrabajo y cada vez más gente hace uso de una *VPN*, usarla como cliente puede ser sencilla, pero montar un Servidor *VPN* es más complejo.

Por muchos motivos puede ser muy útil disponer de un servidor *VPN*, ya sea para conectarnos a una red de forma segura en remoto o, por ejemplo utilizar el servidor para proteger nuestra navegación en redes poco fiables, entre otros muchos usos potenciales.

En mi etapa como administrador de sistemas he montado varios servidores *VPN*, muchas veces usando *OpenVPN*. Suele ser un proceso tedioso y nada fácil que hace muy improbable que personas ajenas a la informática o con conocimientos limitados puedan montarse su propio servidor *VPN*.

Para facilitar esta tarea siempre tuve en mente la creación de scripts para automatizar ciertas tareas, pero de nuevo esto no es nada sencillo de manejar para un usuario sin conocimientos.

En mi investigación pude observar que ya existen soluciones para montar un servidor *VPN*, pero la mayoría de ellas tiene un parte de pago, ya sea por número de usuarios o por otro tipo de configuraciones. Lo bueno que ofrece configurar por ti mismo a bajo nivel *OpenVPN* es que es altamente configurable y sin más restricciones que la del hardware de tu servidor y la infraestructura de red.

La solución que se me ocurrió es crear una interfaz gráfica que permita crear un servidor vpn de forma sencilla y sin complicaciones, pensada para un usuario doméstico sin mucha experiencia o para *Pymes* que no pueden permitirse un router con altas prestaciones y *VPN* incorparada.

Por un lado necesitaría la automatización de tareas mediante scripts y por otro una interfaz que permita al usuario interactuar de forma muy sencilla con estos scripts para poder crear el servidor. También sería necesario la gestión de usuarios y por lo tanto se necesita algún tipo de persistencia, ahí entraría en juego una base de datos.

Para orquestar estas llamadas a base de datos y realizar cierta lógica con los datos y las llamadas a los scripts sería necesario de disponer de un Backend, y un Frontend para la interfaz gráfica.

Otro objetivo primordial del proyecto sería que fuera de código abierto y que todo el mundo pueda acceder a él y usarlo fácilmente, para ello habría que de alguna forma empaquetar Frontend, Backend, Scripts y Base de datos, la forma estándar en la industria suele ser la creación de imágenes *Docker*.

El proyecto será open-source para que la mayor cantidad de usuarios pueda disfrutar del mismo y también contribuir y mejorarlo.

1.2. Problemática

Existen varias problemáticas en este proyecto, la primera de ellas sería que no hay una API como tal para integrar *OpenVPN* con un Backend (Al menos en el momento que realicé la investigación) por lo que tendría que informarme muy de como funciona OpenVPN, de que ficheros hace uso y que directorios necesita para funcionar correctamente. De manera que, nuestra integración se basará únicamente en ficheros y directorios generados por el Backend, lo cual no es la mejor forma de hacerlo, ya que si la integridad del sistema de ficheros se ve afectada nuestra la aplicación podría quedar en un estado corrupto.

La segunda problemática sería que los scripts están bastante ligados al sistema operativo, ya que la integración con *OpenVPN* será muy diferente dependiendo del sistema de ficheros (comandos distintos, sistemas de ficheros distintos, etc) por si esto fuera poco también tenemos dependencia entre distribuciones del propio sistema operativo, como puede ser *Debian* o *ArchLinux* que funcionan totalmente

distintos en algunos aspectos, por lo que hay que intentar escoger el sistema operativo más estable para empaquetar los Scripts y *OpenVPN* y que permita su posterior *"Dockerización"*.

Otra problemática bastante importante y que no podemos automatizar es la necesidad de abrir ciertos puertos en el router para que funcione el servidor VPN.

Por último *dockerizar* esta aplicación es una tarea bastante compleja, ya que no es únicamente la típica aplicación con un Backend, Frontend y Base de datos, hay que tener en cuenta sistema operativo, scripts y *OpenVPN* así como también el uso de interfaces de red, ya que como veremos más adelante hay que gestionar *iptables* y una interfaz de red virtual que se crea para establecer la red de la VPN.

2
Objetivos

En esta se sección se abordadrán los objetivos generales del trabajo así como algunos más específicos que han ido surgiendo a la hora de desarrollar la aplicación.

2.1. Objetivos planteados

2.1.1. Instalación y configuración automática

Uno de los primeros objetivos era que los usuarios pudieran abstraerse del bajo de nivel y puedan configurar el servidor VPN escribiendo únicamente el puerto deseado y la dirección ip del gateway.

La parte de la Instalación de *OpenVpn* y de otras dependencias asociadas como *EasyRSA* deberían ser desconocidas totalmente por el usuario e instalarse y configurarse automáticamente.

2.1.2. Gestión de usuarios

La gestión de usuarios del servidor VPN es compleja, ya que para crear un usuario se debe crear un nuevo certificado firmado por la *CA* del servidor. Y para borrarlo hay que encontrar el certificado creado y revocarlo, esto normalmente requiere conocimientos de *OpenSSL* o similares.

Otra tarea compleja es construir el fichero de configuración del cliente que incluye parámetros de conexión y también ficheros para poder establecer la conexión de forma segura, como por ejemplo el certificado del cliente, la clave *ta* para *tls* y otros.

Para facilitar esta tarea se quiere crear un módulo para la gestión de usuarios donde se pueda crear un usuario nuevo y se puedan realizar borrados sobre los que ya existen. Esto es realmente útil, ya que antes para crear un único cliente había que introducir bastantes comandos y configurar uno a uno el fichero de conexión del cliente.

2.1.3. Envío de clientes a los usuarios

Una vez hemos creado el fichero de conexión del cliente hemos de poder descargarlo y enviarlo al usuario. Al principio se planteó un botón de descarga y luego el administrador del servidor distribuye el fichero manualmente, esto se vio bastante limitante, por lo que se decidió montar un sistema de mailing mediante el cual se envía automáticamente por mail dicho fichero al usuario.

Para poder llevar a cabo este objetivo se necesitó construir un cliente de mailing para el servidor, donde el administrador puede configurar la cuenta de correo que enviará dichos mails a los clientes.

2.1.4. Gestión de la configuración

La configuración del servidor y del cliente interno de correo se tiene que poder cambiar. Siendo conscientes de que el usuario medio de esta aplicación no tiene que saber algunos parámetros de configuración, como por ejemplo la dirección ip pública de su red se necesita realizar un auto rellenado del mismo.

Para ello se creó un módulo que permite la configuración del servidor. En un principio se investigó como poder gestionar la apertura del puerto del servidor automáticamente, pero esto fue imposible debido a que cada router tiene una interfaz diferente, además de que no siempre se puede acceder a la interfaz de administración de un rotuer.

2.1.5. Estado del servidor

Este objetivo es muy importante, ya que dota de una funcionalidad muy útil a la aplicación. Desde un principio se pensó en poder monitorizar el estado del servidor, por ejemplo el número de usuarios conectados, el ancho de banda consumido, etc.

También se vio la necesidad de poder desconectar a un cliente o eliminarlo en caliente. Esto fue complejo, ya que a priori no se actualiza esta información hasta que nos se reinicia el servidor, pero realizando una investigación se encontró un mecanismo para poderlo acometer.

2.1.6. Descarga de logs

A la hora de ejecutar cualquier script se hace logging en un fichero para saber si la ejecución ha sido exitosa o si de lo contrario ha ocurrido algún error. El usuario ha de poder descargarse estos logs para tener información del error o del flujo de ejecución normal.

Eso puede tener ciertas limitaciones, ya que hay que descargarse el log y consultarlo, así que se planteó la posibilidad de consultar los logs del servidor en tiempo real, para ello se creó un módulo que permite consultar los logs, descargarlos, copiarlos e incluso realizar una búsqueda de líneas que contengan cierta palabra.

3

Tecnologías, Herramientas y Metodologías

En esta sección se listarán las tecnologias, herramientas y metodologías usadas en la aplicación y el porque se han usado.

3.1. Tecnologías

3.1.1. Java

He utilizado Java [2] como lenguaje de programación en el Backend, dado que tengo bastante experiencia programando con Java y además tiene una gran cantidad de librerías e información disponible. El proyecto está definido con la versión 19 de Java, haciendo uso de lambdas y programación funcional.

3.1.2. Spring

Spring [3] es un framework modular que permite crear aplicaciones web de manera sencilla. He escogido este framework por ser uno de los más utilizados en la industria y porque tiene mucha información disponible.

3.1.3. Spring Boot

Spring Boot [4] es una herramienta que permite crear una aplicación *Spring* de forma rápida y sencilla. Podríamos decir que es un scaffolding, por lo que es ampliamente recomendable usarlo para crear una aplicación *Spring*.

3.1.4. Spring Security

Spring Security [5] es un framework que permite gestionar entre muchas cosas la autenticación, autorización de una aplicación *Spring*. Ha sido utilizado para la parte de autenticación mediante token JWT y protección de acceso a zonas restringidas de la aplicación.

3.1.5. JWT

JWT [6] es un estándar para la creación de tokens. Ha sido usado para crear el token que permite la autenticación en la aplicación.

3.1.6. Jupiter (Junit5)

Jupiter [7] es el framework de testing que ha sido usado para los test unitarios del Backend. Jupiter permite crear test parametrizables de manera sencilla y nos brinda algunos métodos muy interesantes para usar en los test unitarios.

3.1.7. Lombok

Lombok [8] es una librería que gestiona código repetitivo. A la hora de crear *getters* y *setters* o constructores en la clase *Java* es muy común repetir código, para mejorar estos casos *Lombok* permite hacer uso de anotaciones para implementar dichos métodos repetitivos.

3.1.8. Swagger

Swagger [9] es una herramienta para documentar APIs REST de forma sencilla y mediante anotaciones, también permite probar la API REST y es especialmente útil en este tipo de proyectos.

3.1.9. FasterXML

FasterXML [10] es la herramienta que usa Spring para parsear los objetos JSON utilizados en la aplicación.

3.1.10. WebSocket

WebSocket [11] es una tecnología de comunicación que permite establecer un canal TCP entre cliente y servidor. Es usado para la comunicación en tiempo real entre el Backend y el navegador del cliente. El uso de websocket para enviar información de estado de la VPN y los logs en tiempo real ha sido una de las mejoras más importantes introducidas en la aplicación.

3.1.11. JSON

JSON [12] es un formato de representación de datos que se usa para la comunicación entre distintos sistemas, es un estándar en la industria y permite la comunicación entre Frontend y Backend.

3.1.12. Mockito

Mockito [13] es un framework que permite crear dobles en los test unitarios de manera sencilla. A la hora de crear test unitarios es imprescindible crear dobles de las dependencias externas para facilitar probar distintos casos y agilizar el test. Ha sido usado en los test unitarios del Backend ampliamente.

3.1.13. RestAssured

RestAssured [14] es un framework que sirve para testear API REST. Lo he usado en los test de integración del Backend para testear todos los endpoints expuestos que usa el Frontend. Hace un uso basado en la arquitectura builder que queda bastante limpio en código y es sencillo de usar.

3.1.14. Hibernate

Hibernate [15] es un ORM de mapeo objeto-relacional que he usado para el mapeo de entidades en la capa de repositorios del Backend. Lo he usado por la sencillez de configuración en Spring.

3.1.15. Bash

Bash [16] es un lenguaje de scripting que he usado para implementar todos los scripts del proyecto. He decidido usar este lenguaje porque tengo bastante experiencia usándolo.

3.1.16. Iptables

Iptables [17] es un programa de Linux para gestionar tablas del firewall. El uso de iptables es imprescindible para el correcto funcionamiento del servidor VPN, se ha de configurar una serie de enrutamientos entre las redes del servidor y también reglas de conexión para mejorar la seguridad del servidor.

3.1.17. OpenVPN

OpenVPN [1] es un software de código abierto que permite configurar un servidor VPN. He usado OpenVPN ya que tengo cierta experiencia y además es altamente configurable.

3.1.18. Docker

Docker [18] es una plataforma que permite crear contenedores software. Es un estándar en la industria para desplegar aplicaciones de forma sencilla.

3.1.19. Docker Compose

Docker compose [19] permite gestionar el arranque de varios contenedores a la vez. En vez de generar un contenedor con todas las piezas de la aplicación decidí crear un docker compose y tratar cada pieza como un contendor distinto. Esto evita generar varias versiones de un contendor innecesariamente, además de muchas otras ventajas como poder usar distintas versiones de otros contenedores.

3.1.20. Sveltekit

Sveltekit [20] es un framework para crear aplicaciones web, lo he usado para crear el Frontend. Es un framework relativamente nuevo y no muy extendido en la industria, pero que tiene bastantes características muy interesantes como puede ser SSR, SPA, MPA, SSG. Está basado en *Vite* y su objetivo principal

es construir aplicaciones web con un alto desempeño. Lo he usado porque ya he trabajo con Sveltekit en proyectos anteriores y me parece un framework bastante interesante y con un gran potencial.

3.1.21. TypeScript

TypeScript [21] es un lenguaje de programación basado en JavaScript que permite introducir tipos y compilador que permite ganar en escalabilidad de código para grandes proyectos. En mi opinión es uno de los mejores lenguajes de programación actualmente y es imprescindible a la hora de crear una aplicación web.

3.1.22. JavaScript

JavaScript [22] es un lenguaje interpretado que permite construir aplicaciones web. Es inevitable usar javascript en algunas partes de la aplicación, pero en la mayoría de casos se ha usado typescript.

3.1.23. HTML

HTML [23] es un lenguaje de marcada que permite crear páginas web. Para la maquetación de las distintas páginas web es imprescindible usar html.

3.1.24. Tailwind

Tailwind [24] es un framework para facilitar el uso de *CSS*. En mi caso estoy más especializado en Backend, por lo que no soy un experto *CSS* y este tipo de frameworks facilitan bastante el crear componentes web modernos y reactivos.

3.1.25. NodeJS

NodeJS [25] es un entorno de ejecución para JavaScript. Es usado para poder compilar y ejecutar el Frontend.

3.1.26. EasyRSA

EasyRSA [26] es una herramienta para gestionar autoridades de certificación y certificados digitales. Para autenticar a los usuarios del servidor se usa un

sistema de certificados digitales donde el servidor VPN es una CA que emite, firma y revoca los certificados de los usuarios del servidor, para facilitar esta tarea se usa EasyRSA.

3.1.27. Ubuntu

Ubuntu [27] es una distribución de Linux basada en Debian. Ubuntu es altamente conocida por su estabilidad y facilidad de uso, características principales a la hora de escoger el sistema operativo de la aplicación.

3.1.28. MySQL

MySQL [28] es un sistema de gestión de bases de datos relacionales. He usado MySQL porque es una base de datos gratuita, fácil de usar, rápida y con múltiples capas de seguridad.

3.2. Herramientas

3.2.1. SourceTree

SourceTree [29] es una herramienta para usar GIT con una interfaz gráfica. He usado SourceTree por la facilidad de uso, tiene una interfaz bastante limpia y amigable.

3.2.2. IntelIJ

IntelIJ [30] es un entorno de desarrollo para Java, Kotlin y otros lenguajes. Se ha usado por la gran cantidad de características diferenciales que hace ser mucho más eficiente que otros IDEs como NetBeans o Eclipse.

3.2.3. Visual Studio Code

Visual Studio Code [31] es un editor de código para prácticamente casi todos los lenguajes de programación. He usado este editor para el Frontend por la gran cantidad de plugins que permiten trabajar cómodamente y de manera eficiente con TypeScript.

3.2.4. Git

Git [32] es un software de control de versiones. Es imprescindible usar un sistema de control de versiones a la hora de crear y mantener un proyecto software. Git es el software de control de versiones más usado en la actualidad.

3.2.5. GitHub

GitHub [33] es una plataforma colaborativa que permite alojar proyectos software. La he usado por ser una de las plataformas más usadas, tener un espacio inicial gratuito y tener integrado el sistema de CI Github Actions.

3.2.6. Postman

Postman [34] es un software usado principalmente para testear API REST, pese a que Swagger te permite ya probar los endpoints del Backend he usado también Postman para algunos casos, sobre todo para los websocket que no se pueden probar con Swagger.

3.2.7. DBeaver

DBeaver [35] es un cliente SQL para varios tipos de bases de datos. He usado DBeaver como cliente para la base de datos, ya que sirve para todo tipos de bases de datos gracias a que usa JDBC y estoy acostumbrado a su interfaz.

3.3. Metodologías

3.3.1. TDD

Al principio del proyecto uno de los requisitos no funcionales era usar TDD como metodología de desarrollo. Esta metodología establece una serie de pasos:

1. Establecer un requisito.
2. Realizar los test de los requisitos.
3. Implementar la funcionalidad.
4. Evaluar los test.
5. Si algún test no pasa no se continúa con el siguiente requisito.
6. Volver al paso 1.

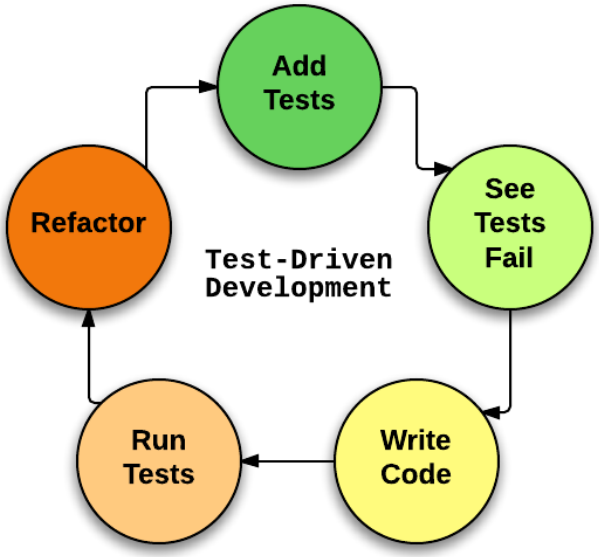

Figura 3.1: Ciclo de TDD

Esta metodología de desarrollo 3.1 es bastante útil y aporta una gran red de seguridad en el caso de que haya que realizar una factorización del código. Debido a ciertas circunstancias no se pudo usar esta metodología al principio del proyecto, pero en etapas posteriores se ha usado y se ha conseguido una amplia red de seguridad con los test implementados.

Ha sido bastante útil, ya que al hacer algunos fix se ha podido detectar que afectaba a otras partes de la aplicación y por lo tanto corregir problemas que se podrían dar en un futuro.

3.3.2. Git flow

Es una metodología para gestionar la creación de ramas en un proyecto GIT.

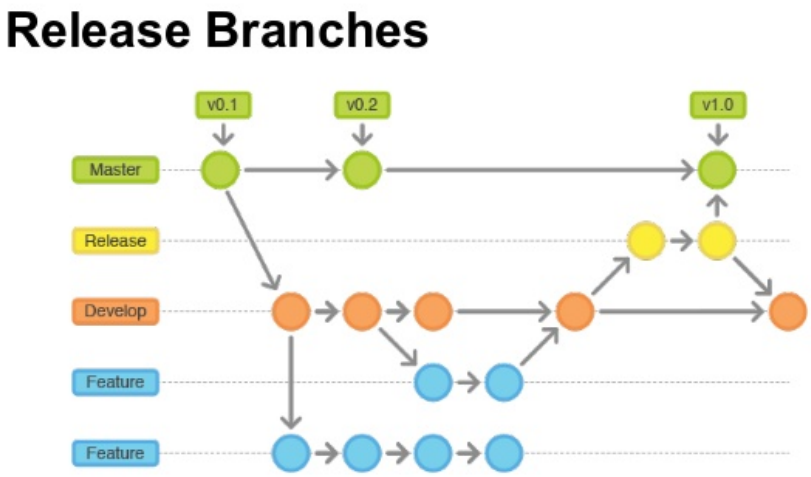

Figura 3.2: Esquema de ramas en Gitflow

Sigue una serie de normas, la más usada en el proyecto es la de la creación de ramas *feature* por cada requisito a implementar, la rama *master* debe contener la versión en producción del software y *develop* la versión en desarrollo de donde parten y se mergean las ramas *feature*.

Este tipo de metodología 3.2 se suele usar sobre todo cuando hay varios equipos trabajando sobre un proyecto, en mi caso al ser un único desarrollador puede que no tenga mucho sentido, pero quería también practicar esta metodología, ya que es bastante usada en la industria y muchos la consideran un estándar.

3.3.3. Arquitectura hexagonal

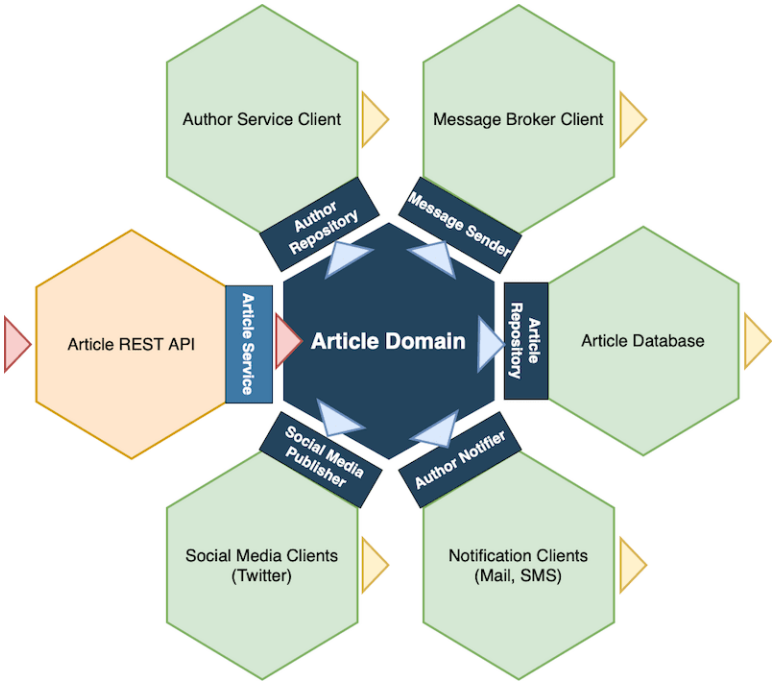

Figura 3.3: Arquitectura hexagonal

Este tipo de arquitectura de software 3.3 proporciona un marco de referencia para guiar la construcción del software definiendo una serie de normas. Si se usa correctamente se puede lograr un sistema con las siguientes características:

- Testeable.
- Mantenible.
- Independiente de Frontend y Base de datos.
- Tolerante al cambio.
- Escalable.

En aplicaciones Spring es muy común usar este tipo de arquitecturas donde tenemos distintas capas respecto a un core o un dominio y es así como se ha construido el Backend. Más adelante se detallará esta parte y veremos la implementación de esta arquitectura.

3.3.4. SOLID

SOLID (*Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion*) son una serie de principios que permiten la escalabilidad y mantenibilidad del código.

4

Descripción Informática

En este capítulo se procederá a explicar el funcionamiento de la aplicación y su arquitectura, así como aspectos técnicos de la misma.

4.1. Requisitos

En esta sección abordaremos los requisitos planteados en la aplicación, tanto los iniciales como los que se han ido planteando a lo largo del desarrollo. También se nombrarán otro tipo de requisitos a la hora de desarrollar el proyecto que no están relacionados en sí con el propio funcionamiento de la aplicación.

4.1.1. Requisitos funcionales

1. El usuario debe ser capaz de crear y configurar la VPN en una única pantalla.

2. El sistema debe permitir asignar roles a los usuarios.

3. El sistema debe permitir únicamente los administradores entrar en la interfaz de administración.

4. El sistema debe permitir únicamente los usuarios hacer uso del servidor VPN.

5. El usuario debe ser capaz de descargarse los ficheros de configuración.
6. El usuario debe ser capaz de enviar los ficheros por email.
7. El usuario debe ser capaz de crear el primer usuario para que el sistema no establezca uno predeterminado de antemano.
8. El sistema debe mantener un registro de los usuarios conectados.
9. El usuario debe ser capaz de consultar el estado del servidor en todo momento.
10. El usuario debe ser capaz de apagar o encender el servidor fácilmente.
11. El usuario debe ser capaz de consultar los logs en tiempo real.
12. El usuario debe ser capaz de descargar los logs fácilmente.
13. El sistema debe permitir desconectar usuarios y eliminarlos sin afectar el servicio a los demás usuarios.
14. El sistema debe proteger páginas restringidas si no se tienen suficientes permisos.

4.1.2. Requisitos no funcionales

1. El sistema debe ser fácil de usar.
2. El sistema debe ser seguro.
3. El sistema será desarrollado usando las mejores prácticas.
4. El sistema debe ser eficiente.

4.2. Arquitectura y funcionamiento

OPENVPN4ALL está compuesto por tres partes fundamentales que permiten la gestión del servidor VPN e interacción con *OpenVpn*.

A continuación se mostrará un diagrama básico para representar las distintas partes de la aplicación y como interactúan entre ellas.

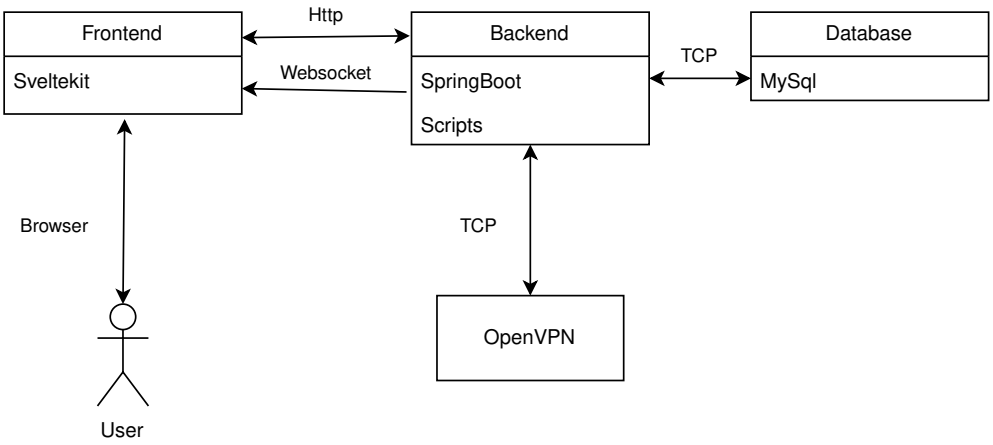

Figura 4.1: Arquitectura de la aplicación

Como podemos observar en la imágen 4.1, el Backend es la única parte que se comunica con la base de datos, es una comunicación bi-direccional, ya que el backend recibe y envía información a la base de datos. El Frontend se comunica con el Backend mediante peticiones *HTTP*, pero también recibe información en tiempo real mediante *Websocket*. OpenVPN se comunica con el Backend bi-direccionalmente mediante TCP. Por último el usuario puede interaccionar con el sistema mediante la interfaz gráfica del Frontend.

4.2.1. Frontend

La estructura de carpetas propuesta por el framework *Sveltekit* es la siguiente:

- src: carpeta donde se almacenan el código principal del proyecto.
  - lib: carpeta para almacenar código real accionado con utilidades o componentes.
  - routes: carpeta donde se guardan las páginas del sitio web, cada carpeta corresponderá con una página. Pueden existir dos ficheros dentro de cada carpeta de una página: *+page.svelte* es el código del cliente web y *+page.ts* es el código que se ejecuta en el servidor (*SSR*). dentro de la raíz de esta carpeta existen los ficheros *+page.svelte* y *+layout.svelte* los cuales son las partes comunes a todas las páginas del Frontend.

Existen más carpetas, pero estas serían las más relevantes del proyecto.

Para construir el Frontend se han usado tres tecnologías principalmente: *SvelteKit*, *TailwindCss* y *TypeScript*.

Figura 4.2: Stack tecnológico del Frontend

Estas tecnologías 4.2 nos permiten tener una interfaz gráfica con componentes reactivos que cambian su estado basándonos en los cambios que puede causar el usuario o el propio Backend. También se ha implementado *SSR* [36] en varias pantallas, haciendo que la experiencia de usuario mejore notablemente al cargar la página.

A continuación se procederá a explicar el funcionamiento de cada una de las páginas del Frontend.

4.2.1.1. Partes comunes

Debido al funcionamiento de la aplicación existen páginas a las que solo se pueden acceder con permisos, en nuestro caso estos permisos se gestionan mediante un token generado a partir del login de un usuario administrador. Las páginas que no están protegidas son la página principal y la página de login.

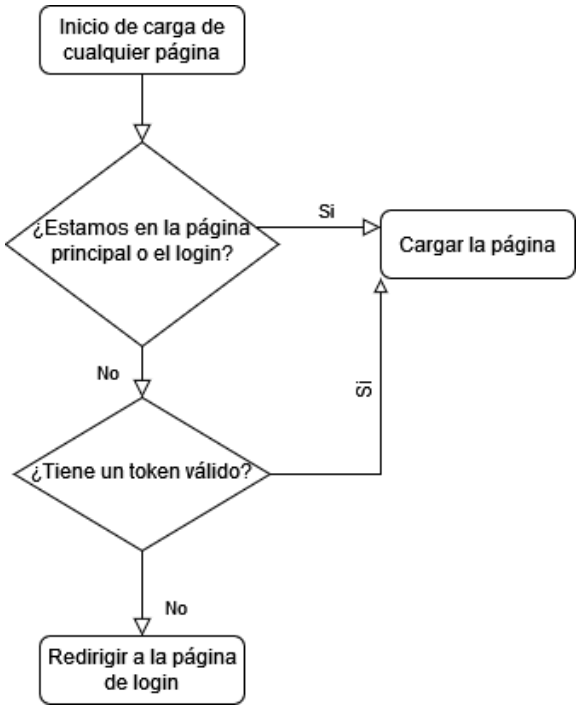

Figura 4.3: Flujo de control en Frontend

Para llevar a cabo dicha protección se utiliza una parte común a todas las páginas que hace la comprobación que se muestra en la figura 4.3.

Existen tres ventanas modales que se usan en varias pantallas de la aplicación, una para mostrar los mensajes informativos 4.4, otra para mostrar mensajes de error 4.5 y por último una para mostrar una confirmación cuando se realizan operaciones peligrosas o que no son reversibles 4.6.

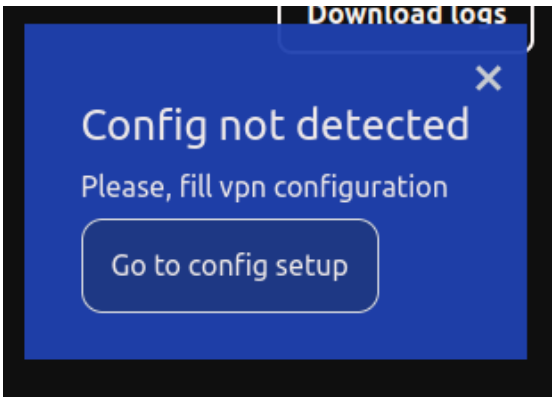

Figura 4.4: Ventana modal informativa

Figura 4.5: Ventana modal error

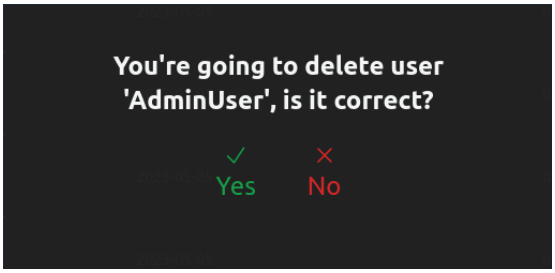

Figura 4.6: Ventana modal confirmación

Todos los formularios de la aplicación tienen validación reactiva, es decir cuando el usuario introduce algo en un campo se valida y se le da feedback al usuario si fuera incorrecto 4.7, si algunos de los campos no es correcto se deshabilita el botón de enviar el formulario.

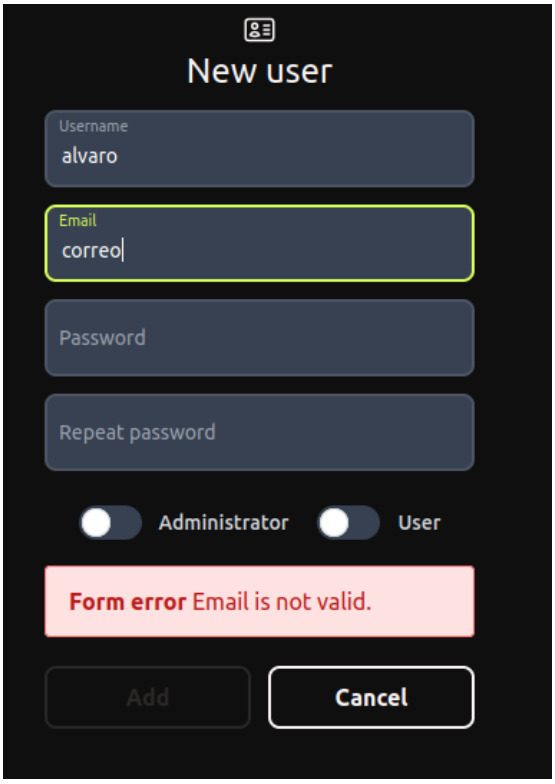

Figura 4.7: validación de los formularios

También a la hora de dar feedback de la operación que se está realizando se le muestra al usuario un spinner o un mensaje informativo de que la operación en cuestión ha finalizado, esto impacta positivamente en la usabilidad de la aplicación.

Existe también un componente común a la mayoría de las pantallas 4.8, se trata de un header que indica en que página estamos actualmente.

Figura 4.8: Header de la aplicación

4.2.1.2. Página de inicio

Esta página 4.9 muestra información sobre la aplicación y si es la primera vez que usamos la aplicación y no tenemos ningún usuario registrado permite crear el primer usuario.

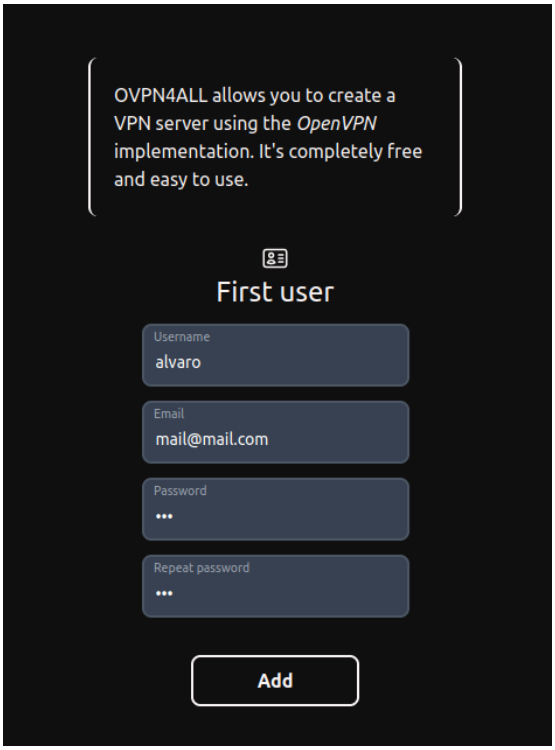

Figura 4.9: Pantalla de inicio

El control de errores de todos los formularios de la aplicación son reactivos, es decir cada vez que el formulario cambia se hace una comprobación y se le informa al usuario si hay algún campo incorrecto. También siempre se hace doble control en Backend, por seguridad.

Todas las llamadas al Backend tienen control de errores, es decir si se devuelve un código distinto de 2XX se recoge en el body de la respuesta el error y se muestra al usuario. Si no es un error controlado del Backend se muestra un error genérico de comunicación.

Resaltar que esta es una de las pocas páginas de la aplicación que no está protegida, es decir, no hace falta tener un token válido para acceder a la misma.

Para explicar el funcionamiento de cada pantalla se mostrará un diagrama de comunicación. Si algún proceso del Frontend se realiza mediante *SSR*, se indicará en el diagrama 4.10.

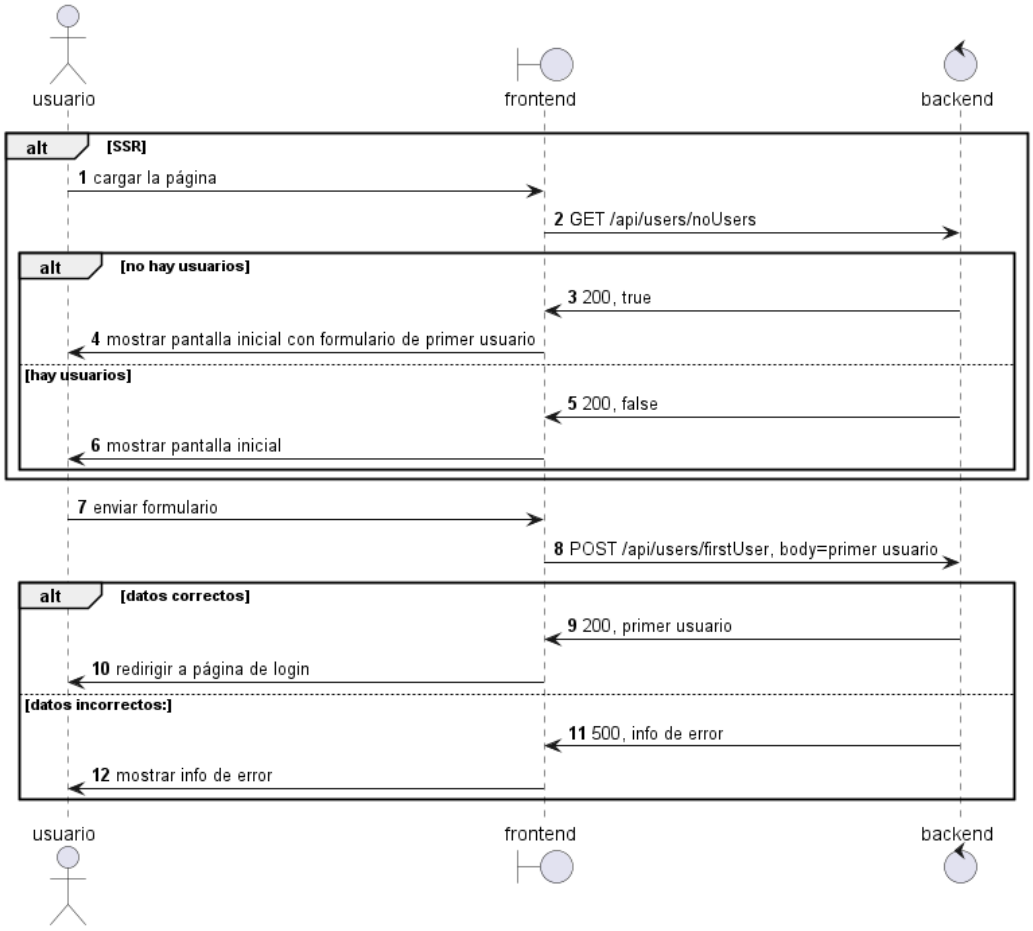

Figura 4.10: Diagrama de comunicación de la pantalla de inicio

4.2.1.3. Página de login

Una vez tenemos un usuario administrador usaremos esta página 4.11 para loguearnos en la aplicación y así poder empezar a crear el servidor VPN. Esta página no está protegida y se puede acceder a la misma sin la presencia de un token válido.

Es importante comentar que una vez nos hemos logueado correctamente, se nos crea un token JWT que será válido durante un periodo de tiempo para no tener que estarnos autenticando contra el Backend en cada petición que hagamos. El token se almacena en memoria del navegador, es decir como una *Cookie*.

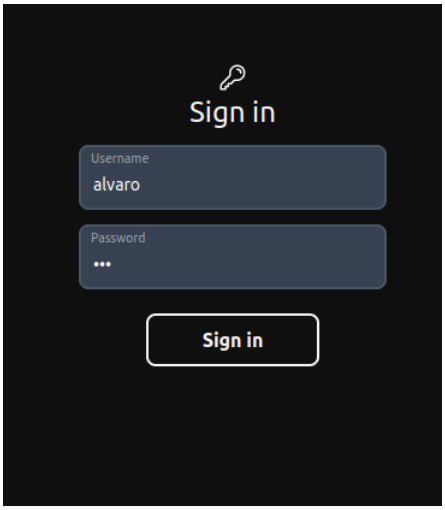

Figura 4.11: Pantalla de login

Como se ha comentado en la pantalla anterior se realiza una validación de los datos reactiva, y si cumple las normas se le permite al usuario enviar los datos a Backend donde se hace una segunda validación.

En el diagrama 4.12 podemos observar como es el proceso de login.

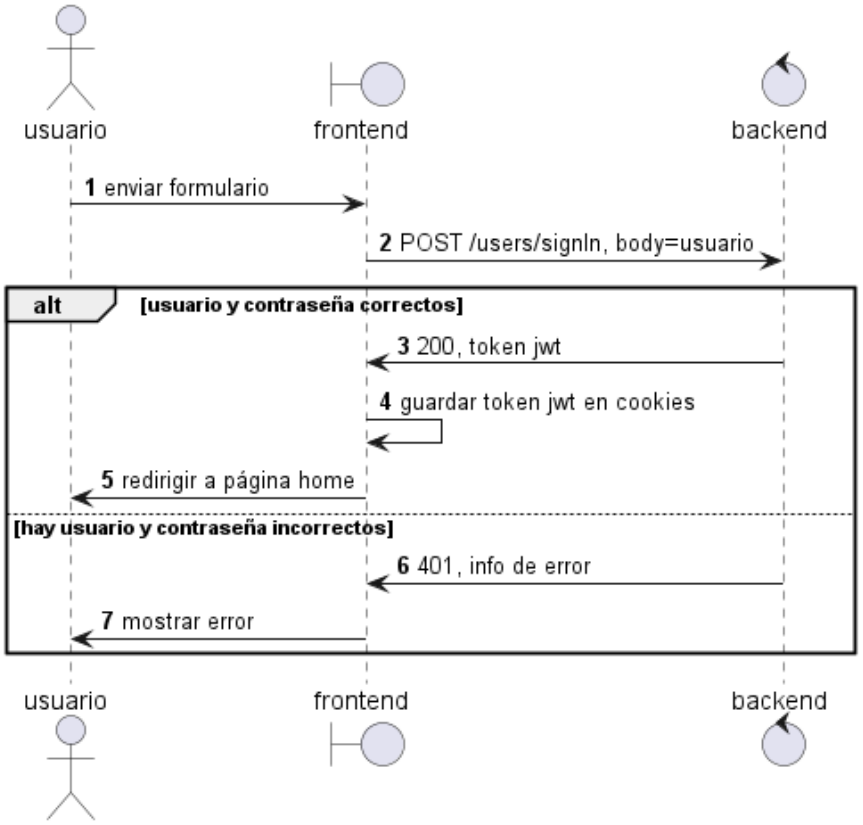

Figura 4.12: Diagrama de comunicación de la pantalla de login

4.2.1.4. Página home

Esta página 4.13 es la que ofrece más funcionalidades de toda la aplicación, permite visualizar la configuración, estado de la VPN y usuarios conectados en tiempo real usando Websockets [11].

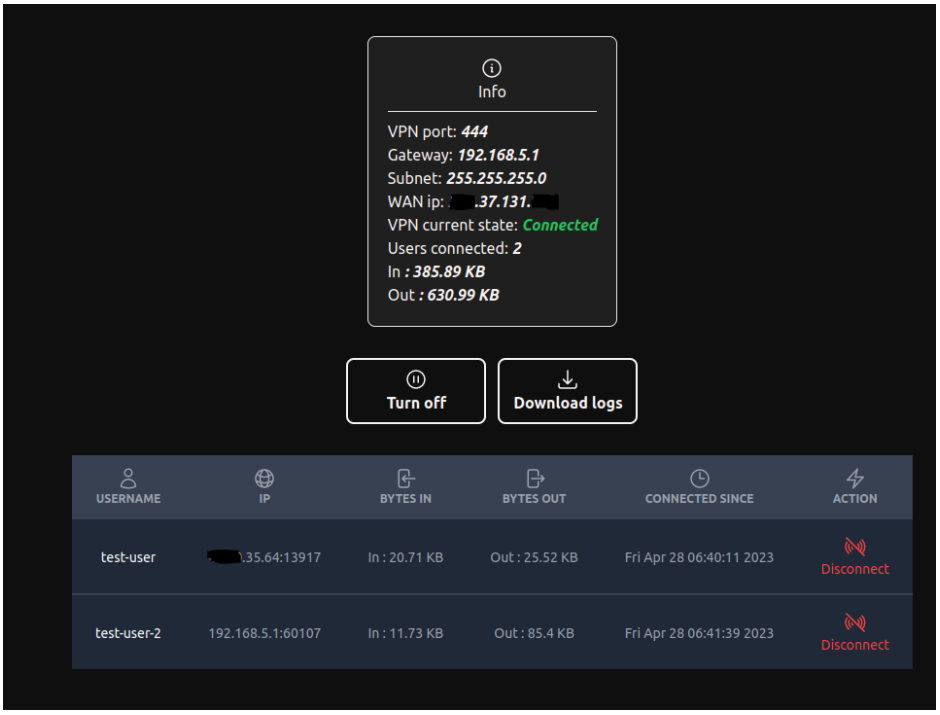

Figura 4.13: Pantalla home

Como podemos observar tenemos un pequeño contenedor donde se nos muestra información del servidor, como el puerto, dirección ip del gateway, máscara de subred y la dirección pública del servidor VPN. También tenemos datos de los usuarios conectados y el consumo de red, estos datos son actualizados en tiempo real, ya que se reciben periódicamente a través de Websockets.

También tenemos el botón para apagar o encender el servidor y el botón que permite descargarse todos los logs del servidor en un .zip. La tabla de usuarios nos muestra cuantos usuarios tenemos conectados actualmente e información más específica sobre ellos, como por ejemplo el nombre de usuario, su dirección ip, el consumo de red, desde cuando llevan conectados y por último un botón para desconectarlos. Esta tabla también muestra sus datos en tiempo real.

A la izquierda podemos observar unos botones que nos indican en que parte de la página estamos y si queremos ver el contenido de un log en específico.

A continuación se mostrará un diagrama de secuencia 4.14 con las distintas operaciones que pude realizar el usuario, así como las llamadas SSR que realiza el Frontend.

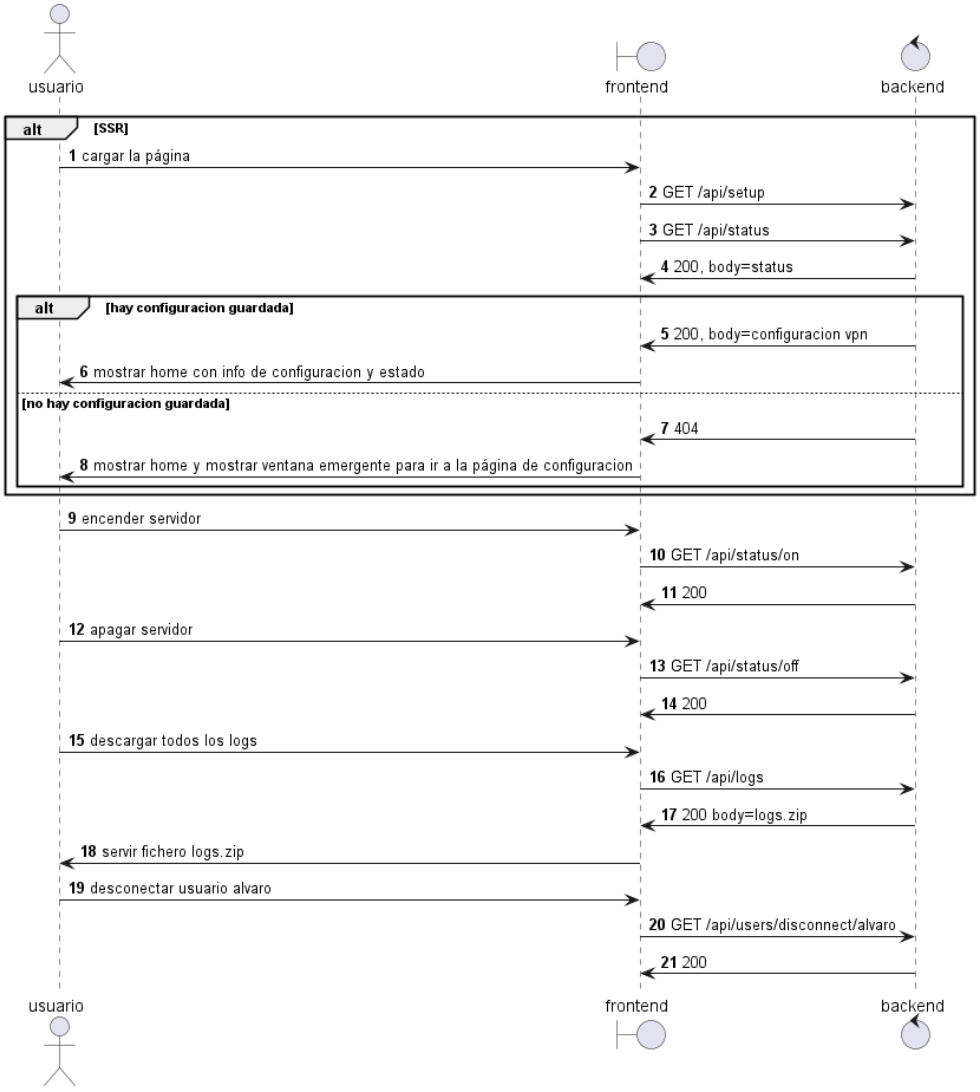

Figura 4.14: Diagrama de comunicación de la pantalla home

La ventaja de usar Websockets es que establecemos una comunicación TCP con el servidor que no se cierra, por lo que la comunicación es más ágil y permite evitar soluciones poco eficientes como *polling* y llamadas masivas a endpoints.

Existen diferentes topics a los cuales desde el Frontend se va consultando cada cierto periodo de tiempo para actualizar la información, en otros topics el Backend envía la información de manera periódica y el Frontend la consume en tiempo real. A continuación se mostrará un diagrama de comunicación 4.15 para ver como son las llamadas a estos topics.

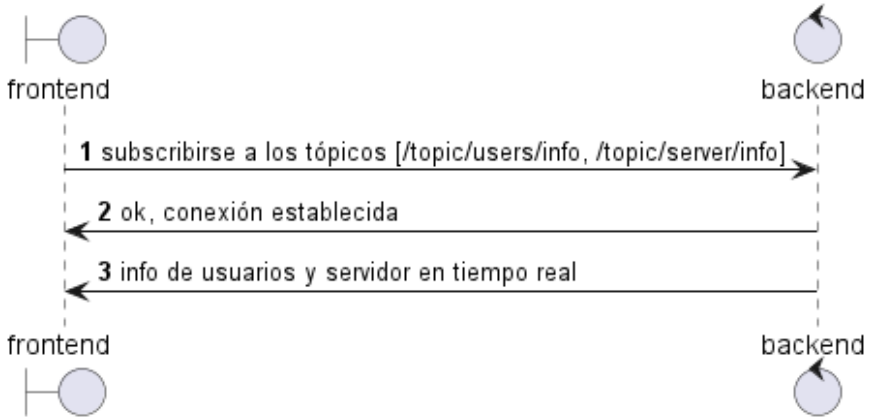

Figura 4.15: Diagrama de comunicación de home usando websockets

4.2.1.5. Página de logs

La página de logs 4.16 permite consultar los distintos logs del servidor en tiempo real. También permite descargar los logs, copiar el contenido en el portapapeles y filtrar las líneas que contengan la palabra que se introduzca en el botón de búsqueda. Esta funcionalidad es bastante útil porque permite

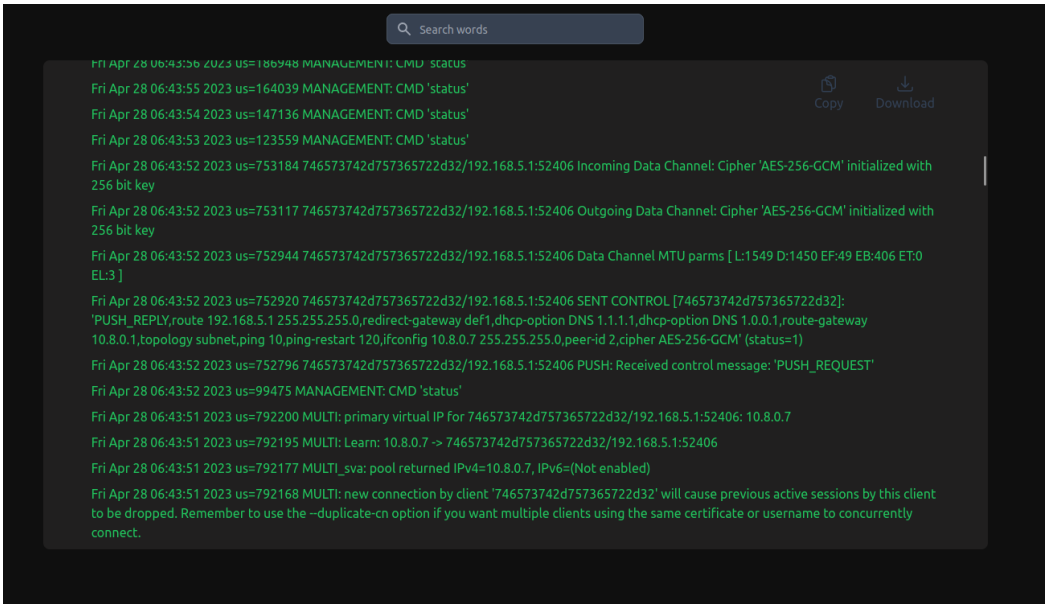

Figura 4.16: Pantalla de logs

Cabe destacar que las líneas de los logs más recientes se escriben al principio, los botones de copiar y descargar son flotantes y se mantienen en el mismo sitio aunque se haga scroll del log. Esta funcionalidad es especialmente útil para poder hacer un diagnóstico rápido del estado del servidor sin tener que conectarse por consola al mismo. Por ejemplo si ocurre algún error analizando el log podríamos saber que ha sucedido.

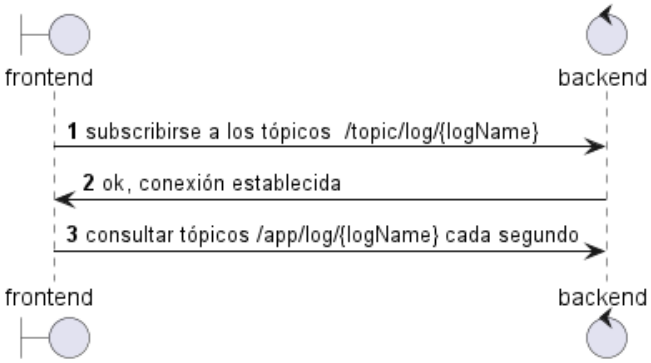

Figura 4.17: Diagrama de comunicación de la pantalla de logs

En el diagrama 4.17 podemos observar como es el proceso de suscripción y de consulta de un topic.

4.2.1.6. Página de usuarios

La página de usuarios 4.18 permite la creación, eliminación, descarga de fichero de conexión y envío de fichero de conexión. Es una tabla paginada donde podemos visualizar: nombre, email, fecha de creación, rol y acciones asociadas.

A la hora de obtener los usuarios es importante usar paginación, ya que si por ejemplo existieran muchos usuarios registrados se podría saturar el Frontend.

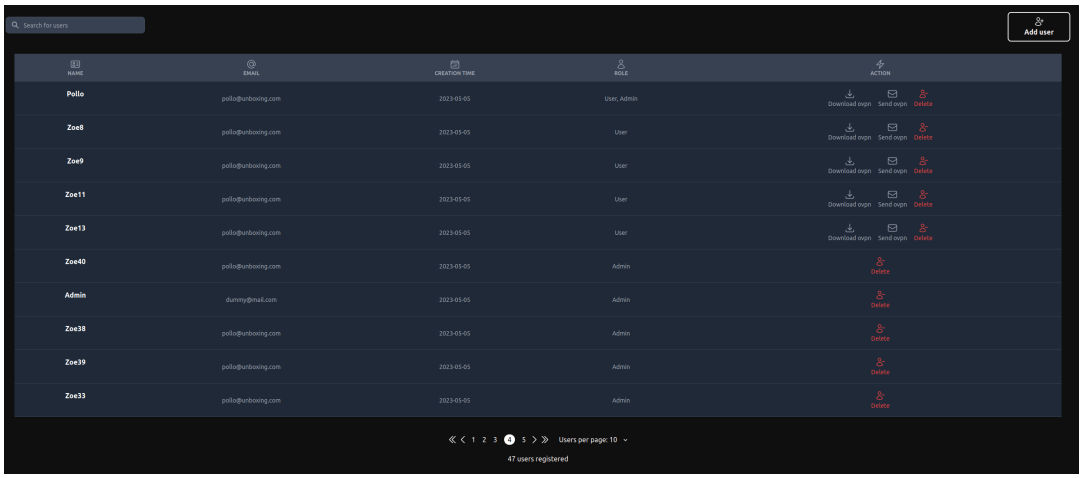

Figura 4.18: Pantalla de usuarios

Como podemos observar en la imagen 4.18 tenemos un cuadro de búsqueda que permite encontrar al usuario por nombre de forma rápida y eficiente. En la parte inferior disponemos de la navegación por página donde podremos desplazarnos fácilmente por cada página de la tabla. El botón para añadir usuarios nos muestra la pantalla de registro de usuarios. Entre las acciones que podemos realizar a los usuarios disponemos de: eliminación, descarga del fichero de conexión y envío del fichero de conexión. Estas dos últimas quedan reservadas para los usuarios que tengan al menos el rol de usuario, ya que si solo tienen el rol de administrador en principio su labor es administrar el servidor VPN y no conectarse. Ambos roles (administrador y usuario) pueden combinarse .

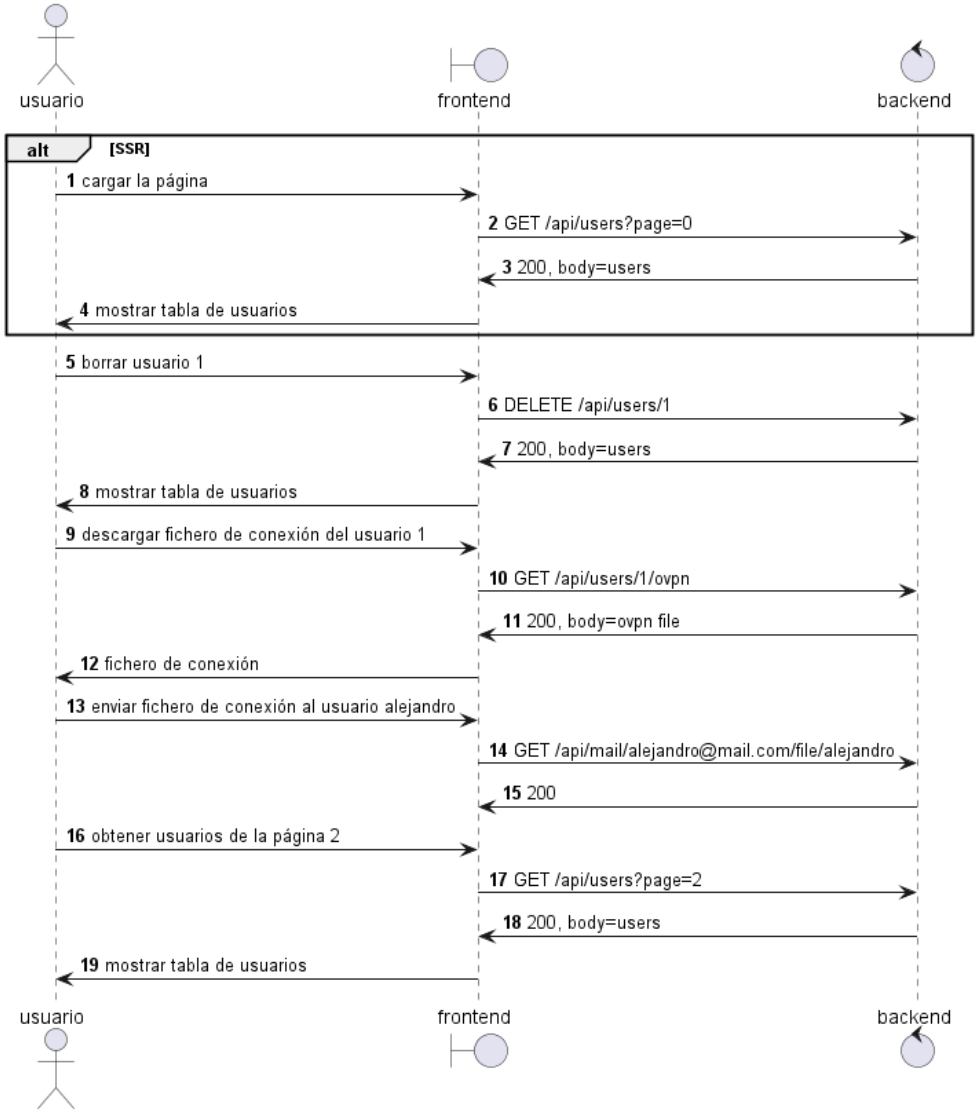

Figura 4.19: Diagrama de comunicación de la pantalla de usuarios

En el diagrama 4.19 se exponen las distintas operaciones que se pueden realizar desde la página de usuarios.

4.2.1.7. Página de registro

La página de registro 4.20 es muy similar a la página de registro del primer usuario, ya que se reutiliza el formulario quitando la elección del rol de usuario.

Para registrar a un usuario es necesario introducir el nombre de usuario, email, contraseña dos veces y su rol. La comprobación de estos campos se realiza de forma reactiva y se le informa al usuario de manera inmediata si alguno de los campos es incorrecto.

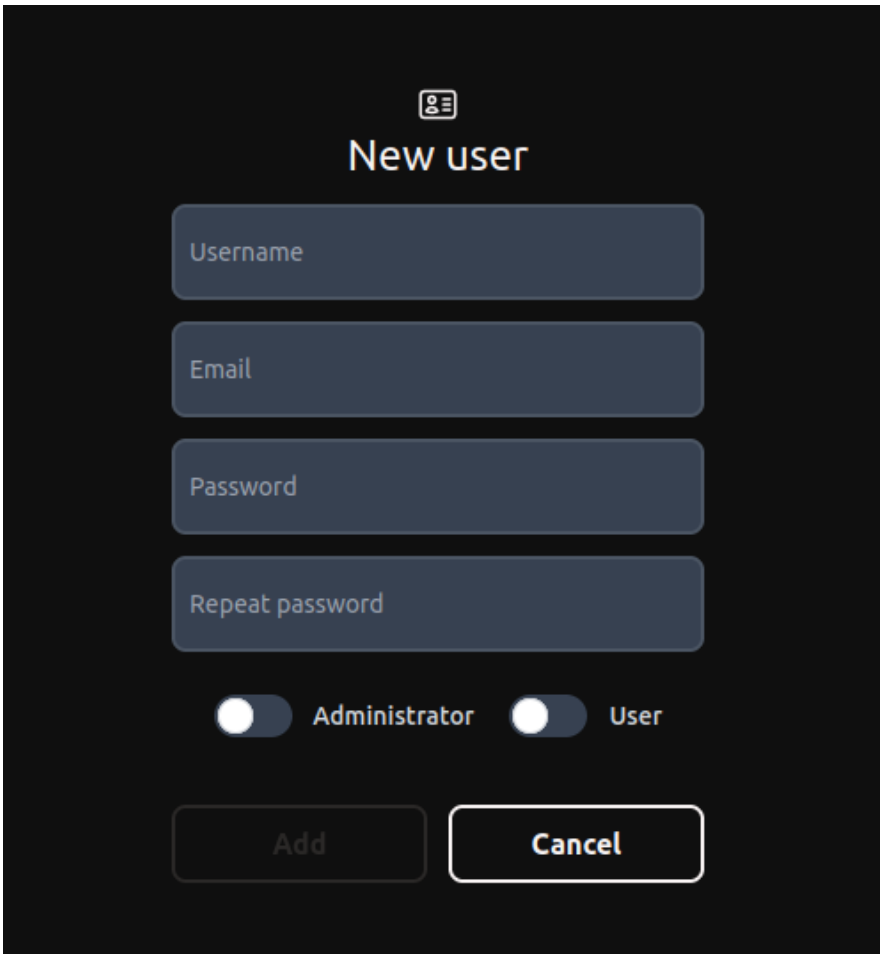

Figura 4.20: Pantalla de registro

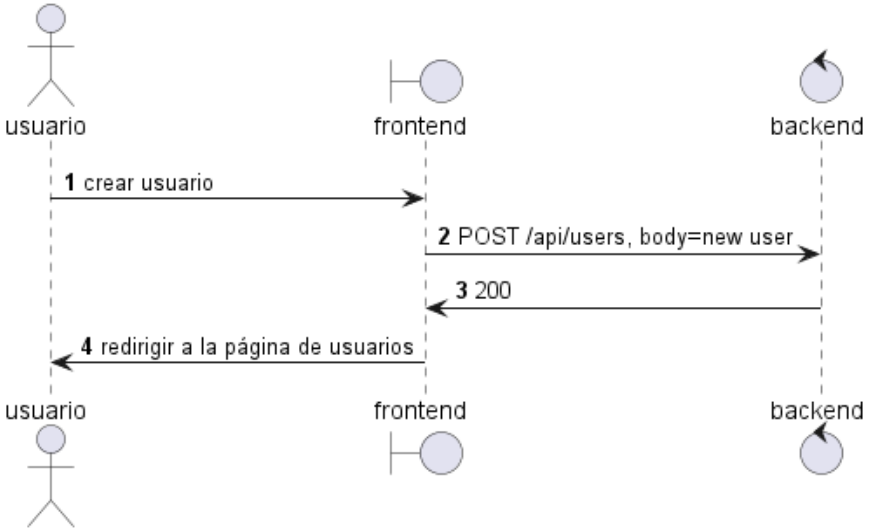

Figura 4.21: Diagrama de comunicación de la pantalla de registro

En el diagrama 4.21 se documenta el proceso de comunicación al registrar de un usuario.

4.2.1.8. Página de configuración del servidor

A la hora de configurar el servidor VPN se pueden incluir multitud de parámetros de configuración, por ejemplo el tipo de cifrado, tcp o udp, etc. Para mantener la simplicidad a la hora de gestionar la configuración se ha limitado dichos parámetros a: puerto del servidor, dirección ip del gateway, máscara de subred y la dirección pública del servidor.

La página de configuración del servidor 4.22 es un formulario con dichos parámetros. Existe un botón para auto rellenar la dirección ip pública.

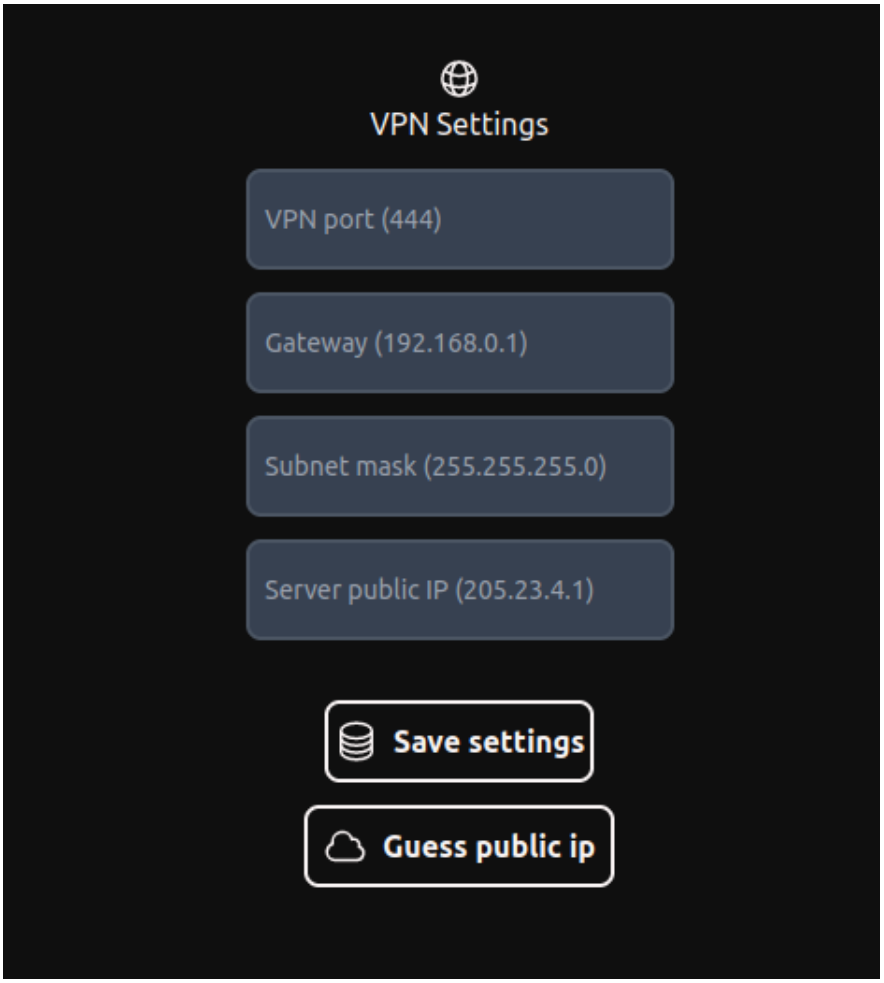

Figura 4.22: Pantalla de configuración del servidor

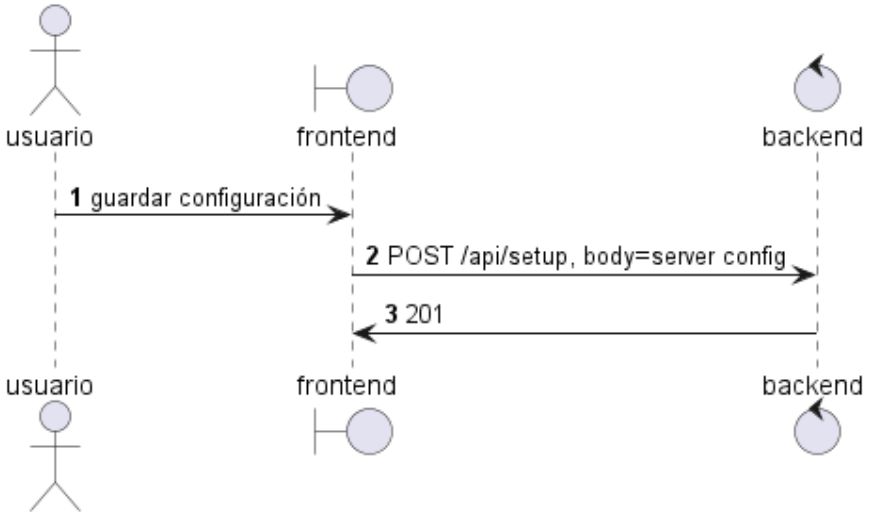

Figura 4.23: Diagrama de comunicación de la pantalla de configuración del servidor

En el diagrama 4.23 se puede observar el proceso de comunicación al guardar la configuración.

4.2.1.9. Página de configuración del cliente de correo

Para poder enviar por correo el fichero de configuración es necesario configurar el correo que utilizará el cliente. Será necesario introducir el servidor de correo *SMTP*, puerto del servidor de correo, dirección de correo, contraseña y si es un servidor *STARTTLS*. Una vez enviado se harán las validaciones necesarias en el Backend para comprobar su validez.

El formulario 4.24 permite enviar dicha configuración de correo.

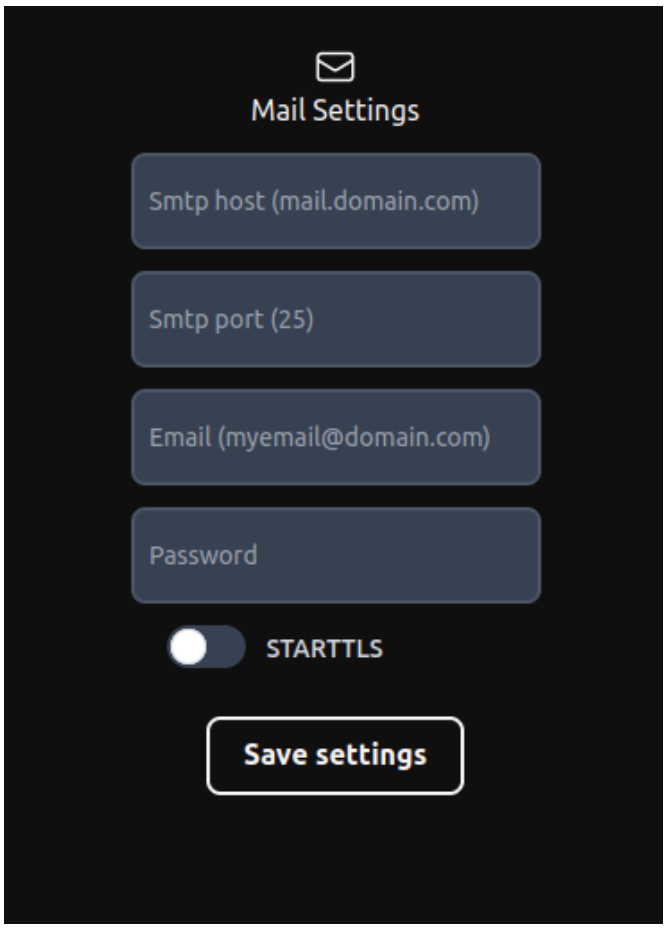

Figura 4.24: Pantalla de configuración del correo

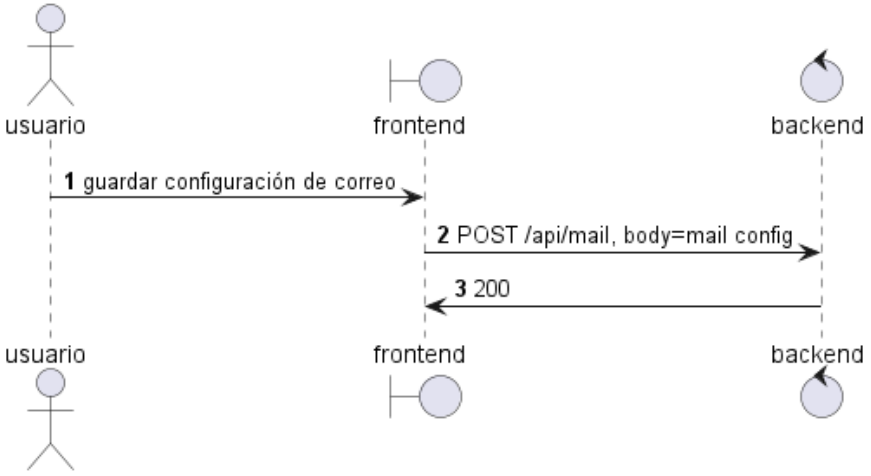

Figura 4.25: Diagrama de comunicación de la pantalla de configuración del correo

En el diagrama 4.25 se documenta el proceso de comunicación para enviar la configuración de correo.

4.2.2. Backend

A la hora de construir el Backend se ha seguido la arquitectura hexagonal 4.26, que es un estándar en la industria. Esta arquitectura divide las distintas partes de la aplicación por capas y cada capa tiene una responsabilidad bien definida.

- Controller: capa donde se exponen los endpoint, se definen características del endpoint, entrada y salida de los datos.
- Service: capa donde se ejecuta la lógica de negocio y tiene el mayor peso.
- Repository: capa donde se modelan las entidades que representan una tabla en la base de datos y sus respectivas consultas.

La arquitectura 4.26 puede tener más capas y ser más compleja, pero en el caso de la aplicación estas serían las capas principales. Para hacerse una idea general de como suele ser la arquitectura de una aplicación Spring, mostraré el siguiente diagrama:

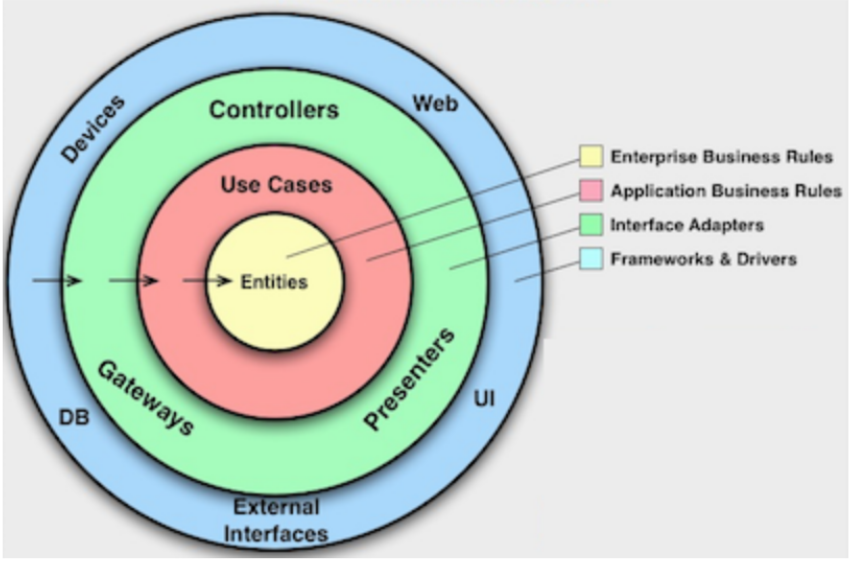

Figura 4.26: Arquitectura hexagonal en aplicaciones Spring

4.2.2.1. Token de autenticación

Todas las peticiones que recibe el Backend son validadas previamente, salvo aquellas urls que no están protegidas, como por ejemplo: */api/users/signIn*, */api/users/firstUser*, */api/users/noUsers*. Para poder llevar a cabo dicho proceso se ha hecho uso del módulo *Spring Security* el cual nos permite configurar filtros de forma sencilla para gestionar temas relacionados con la seguridad de la aplicación.

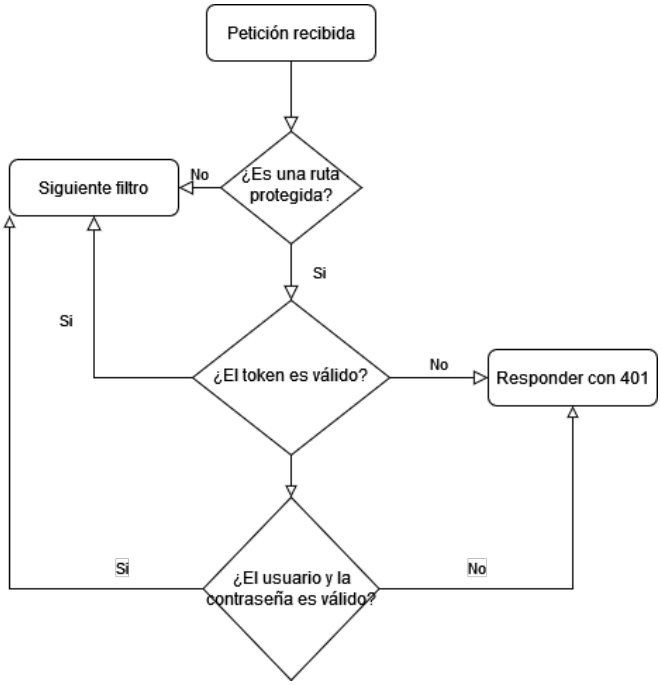

Figura 4.27: Proceso de validación del token

ç

En el proceso 4.27 se comprueba que el token que proporciona el usuario sea válido, usuario y contraseña correcta y por último que dicho usuario tenga el rol de administrador.

El token de autenticación sigue el formato estándar *JWT*.

4.2.2.2. Programación funcional (Lambdas)

A lo largo del desarrollo en el Backend se ha usado activamente la programación funcional, que *Java* permite mediante el uso de *Lambdas*

```
1  private boolean isLastAdmin(UserEntity userEntity) {
2    return  this.isAdmin(userEntity)
3            && userRepository.findAll()
4            .stream()
5            .filter(this::isAdmin)
6            .toList()
7            .size() == 1;
8  }
```

Listing 4.1: Código con lambdas

La programación funcional nos permite crear código fácil de testear, fácil de mantener, funciones puras sin efectos colaterales y entre muchos otros beneficios,

podemos reutilizar funciones previamente creadas más fácilmente que en otros paradigmas distintos.

4.2.2.3. Gestión de usuarios

Existe un controlador llamado *UserController* que expone las distintas operaciones que se pueden realizar sobre un usuario. Las operaciones serían las siguientes:

- saveUser (POST /api/users): permite guardar el usuario deseado pasado por body. En la capa de servicio se comprueba si el usuario no ha sido registrado previamente y si el rol proporcionado existe, si todo es correcto se guarda el usuario en base de datos y se ejecuta el script de creación de usuario.
- getUsers (GET /api/users): recibe como parámetros el número de página y tamaño de página y devuelve los usuarios de la aplicación paginados. Es importante no devolver todos los usuarios de una sola vez para no saturar al Frontend.
- getUser (GET /api/users/id): recibe un ID de usuario, busca en la base de datos dicho usuario y si existe responde con los datos del usuario.
- signIn (POST /api/users/signIn): permite autenticar al usuario recibido por body, para ello busca en base de datos al usuario y comprueba que las contraseñas coincidan, si es correcto se genera el token JWT a partir del nombre de usuario y se devuelve el token.
- deleteUser (DELETE /api/users/id): recibe un ID de usuario y si dicho usuario existe en base de datos se borra, si el usuario está conectado se mata la conexión y por último se llama al script de borrado de usuario.
- deleteUserWithName (DELETE /api/users/name): misma operación que la anterior, pero se busca al usuario en base de datos por el nombre.
- downloadOVPNFile (GET /api/users/id/ovpn): dado un usuario si este existe se recupera el fichero de conexión y se retorna al Frontend.
- disconnectUser (GET /api/users/disconnect/userName): dado un usuario si existe y está conectado se mata la conexión con el servidor VPN.
- registerFirstUser (POST /api/users/firstUser): se registra al primer usuario si no existe ninguno registrado previamente.
- noUsers (GET /api/users/noUsers): devuelve 200 si no hay ningún usuario registrado.

4.2.2.4. Gestión de la configuración del servidor

Para poder gestionar la configuración del servidor se ha creado un controlador llamado *ConfigController* que expone las siguientes operaciones:

- getSetup (GET /api/setup): devuelve la configuración actual del servidor si está presente.
- setupServer (POST /api/setup): se valida que la configuración recibida sea válida, si es correcta se guarda en base de datos y se llama al script de generación de configuración para el servidor VPN.

4.2.2.5. Envío de emails

Para enviar por mail el fichero de conexión al usuario se ha creado un controlador llamado *MailController* que permite guardar una configuración de correo y enviar correos. Los endpoints son los siguientes:

- setEmail (POST /api/mail): recibe una configuración de correo, comprueba que los datos sean válidos, envía un correo de prueba y si funciona correctamente guarda en base de datos la configuración del correo.
- sendEmail (POST /api/setup/userMail/file/fileName): localiza el fichero de conexión asociado al nombre de usuario y si existe dicho fichero comprueba si hay una configuración de correo disponible, si es así envía dicho fichero.

Para poder enviar los mails se ha creado un cliente de correo muy sencillo, que solo hace uso del servidor *SMTP* proporcionado.

4.2.2.6. Descarga de todos los logs

Además de la pantalla de logs de Frontend donde puedes descargar un log concreto existe un endpoint que permite descargar todos los logs en un .zip.

- downloadLogs (GET /api/logs): se ejecuta un comando para buscar y comprimir en un .zip todos los logs del servidor y posteriormente se envía.

4.2.2.7. Gestión del estado del servidor

Para obtener información del estado del servidor y poder encender o apagar el mismo se ha creado este controlador llamado *StatusController*.

- getStatus (GET /api/status): obtiene el estado actual del servidor, para ello se ejecuta el comando *pgrep* buscando el proceso de *openvpn* y devuelve si se ha encontrado o no.
- startOpenvpn (GET /api/status/on): enciende el servidor VPN, para ello revisa si previamente ya estaba encendido el servidor y si no lo estaba ejecuta el comando *openvpn* seguido del fichero de configuración del servidor.
- shutdownOpenvpn (GET /api/status/off): apaga el servidor VPN, comprueba que si está encendido, si es así apaga el servidor con el comando *pkill*.
- getBandwidth (GET /api/status/bandwidth): obtiene datos del consumo de red del servidor VPN, para ello se ejecuta el comando *grep* sobre la interfaz de red virtual *tun0* filtrando las líneas que contienen */proc/net/dev* con esa salida se realiza un parseo y se obtiene el consumo de entrada y de salida de dicha interfaz.

4.2.2.8. Websocket

Para poder enviar al Frontend información en tiempo real se ha creado un controlador que contiene todos los tópicos creados con *Websocket*, utilizar esta tecnología es ideal en este tipo de casuísticas donde tenemos que tener información actualizada en un corto periodo de tiempo.

Se tienen a disposición los siguientes tópicos:

- /topic/log/nombreLog: envía en tiempo real el contenido del log especificado.
- /topic/users/info: envía información de los usuarios conectados, esta información se obtiene haciendo uso de la interfaz de administración de *OpenVpn*.
- /topic/server/info: envía información de consumo de red del Servidor.

4.2.2.9. Interfaz de administración de OpenVpn

Para poder administrar el Servidor OPENVPN y obtener información del mismo existe una herramienta llamada Interfaz de administración [37]. Mediante una conexión *TCP* al puerto que se haya configurado en el fichero del servidor podemos enviar comandos al servidor y este nos reportará la información que se solicite.

Para poder hacer uso de dicha interfaz se ha creado un cliente llamado *ManagementInterfaceClient* el cual permite realizar las siguientes operaciones:

- killUser: permite desconectar al usuario especificado.
- status: recibe información del estado del servidor, esta información es parseada mediante una serie de regex y se extraen los usuarios conectados con información detallada de los mismos.

Sólo se puede establecer una conexión con la interfaz de administración, por lo que los diferentes hilos del Backend deben compartir la misma instancia, para ello se ha utilizado el patrón singleton [38]. Para proteger el programa de condiciones de carrera se ha hecho uso de Singleton beans, que son un tipo especial de *beans* implementadas directamente en *Spring*.

4.2.3. Base de datos

Como se ha comentado en secciones anteriores se ha utilizado *MySQL* como base de datos para almacenar las distintas entidades que maneja la aplicación. La base de datos almacena únicamente cuatro entidades, las cuales vienen representadas en el diagrama 4.28.

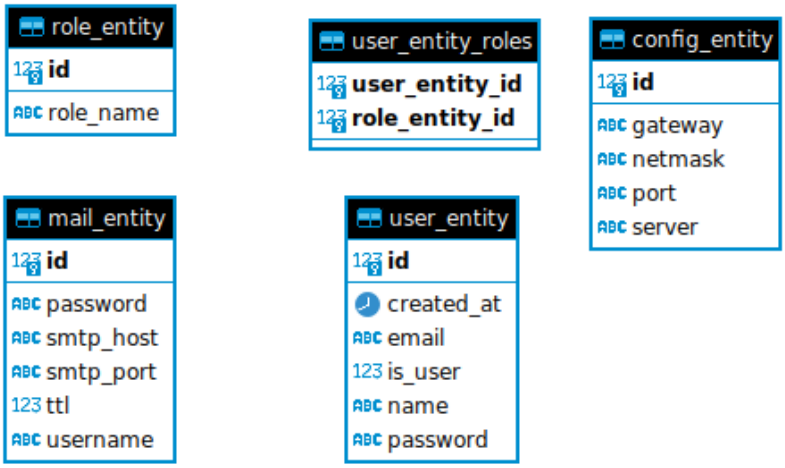

Figura 4.28: Tablas de la base de datos

- *role entity*: representa los distintos roles que puede tener un usuario, actualmente son: administrador y usuario.
- *user entity*: representa un usuario de aplicación, la contraseña está codificada usando un *SALT* aleatorio, que permite verificar que una contraseña es correcta, pero no permite descifrar por completo la contraseña. este tipo de codificador se llama *BCrypt* [39].

- *user entity roles*: permite representar el rol al que pertenece un usuario.
- *mail entity*: representa la configuración de correo que utiliza el cliente de correo para enviar los ficheros de conexión del cliente.
- *config entity*: representa la configuración de red del servidor VPN.

4.2.4. Scripts

Los scripts son una parte fundamental de la aplicación, ya que son el puente que permite gestionar *OpenVPN*. Están programados en *BASH* [16] y permiten automatizar ciertas tareas.

Tienen una estructura genérica: en todos los scripts hay una función que permite hacer logging de la salida de todos los comandos ejecutados y del propio proceso del script, se definen los argumentos del script para comprobar que se esté ejecutando correctamente.

En las siguientes secciones se explicarán cuál es el funcionamiento de cada uno de los scripts.

4.2.4.1. Servidor

- *CreateIptables.sh*: permite crear las reglas *iptables* [17] necesarias para que el servidor VPN funcione correctamente, esto implica el redirigir los paquetes udp de la interfaz de red virtual creada por *OpenVPN* a la interfaz de red real del equipo.
- *CreateServerCert.sh*: descarga la última versión disponible de *EasyRSA* [26], la instala en el quipo y la configura. Posteriormente se genera la autoridad de certificación con la cual el servidor podrá crear y firmar certificados de los usuarios y se genera y firma el propio certificado del servidor y la clave privada.
- *CreateServerConfig.sh*: recibe el puerto, gateway y máscara de subred como parámetros y con una plantilla de configuración permite crear el fichero de configuración del servidor que usa OpenVPN. También hace uso de ficheros creados con el script anteriormente nombrados como son la clave privada del servidor, el certificado, la autoridad de certificación y otros ficheros relacionados con *TLS* [40].
- *DownloadLogs.sh*: crea un fichero *.zip* con todos los logs del servidor.

4.2.4.2. Usuario

- *CreateUserCert.sh*: permite crear el certificado del usuario y firmarlo mediante *EasyRSA* [26], recibe como parámetros el nombre del usuario en hexadecimal y la contraseña del usuario para generar la clave privada.
- *CreateUserVPNFile.sh*: permite configurar el fichero de conexión del cliente, para ello recibe el nombre del usuario, ip del servidor y puerto, rellena la plantilla de configuración del cliente con dichos parámetros e incluye la autoridad de certificación, certificado del usuario y clave privada del usuario.

4.2.5. Plantillas de configuración

Existen diferentes plantillas de configuración que se rellenan con campos específicos para cada caso.

4.2.5.1. Plantilla de configuración del servidor

Permite configurar los distintos parámetros que puede usar *OpenVPN* para el servidor VPN, en este caso la parte que se puede rellenar sería la dirección ip del gateway y la máscara de subred. Por defecto el servidor VPN está configurado con el protocolo *UDP*, conexión en modo túnel, cifrado *AES-256-GCM* curvas ecdh *secp521r1* y otros parámetros adicionales.

```
port <fill_port>
proto udp
dev tun
dh none
remote-cert-tls client
cipher AES-256-GCM
tls-ciphersuites TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
tls-cipher TLS-ECDHE-ECDSA-WITH-AES-256-GCM-SHA384:TLS-ECDHE-ECDSA-WITH-CHACHA20-POLY1305-SHA256
ecdh-curve secp521r1      You, 7 months ago • Proyect setup
tls-version-min 1.2
reneg-sec 0
auth SHA512
topology subnet
server 10.8.0.0 255.255.255.0
ifconfig-pool-persist /var/log/ipp.txt
push "route <fill_gateway> <fill_subnet>"
push "redirect-gateway def1"
push "dhcp-option DNS 1.1.1.1"
push "dhcp-option DNS 1.0.0.1"
client-to-client
keepalive 10 120
max-clients 100
user nobody
group nogroup
persist-key
persist-tun
status /var/log/openvpn-status.log
verb 4
log /var/log/openvpn.log
explicit-exit-notify 1
key-direction 0
management localhost 4447
```

Figura 4.29: Plantilla del servidor

En la imagen 4.29 podemos observar la plantilla de configuración del servidor y los parámetros predeterminados.

4.2.5.2. Plantilla de configuración del cliente

Con la plantilla de configuración del cliente 4.30 se generan los ficheros de conexión del usuario. Esta plantilla se rellena con la dirección ip del servidor y el puerto. Viene pre-configurada con las características definidas en el fichero de configuración del servidor.

```
client
dev tun
proto udp
remote <fill_server> <fill_port>
resolv-retry infinite       You, 7 months ago • Proyect setup
nobind
persist-key
persist-tun
remote-cert-tls server
cipher AES-256-GCM
auth SHA512
tls-ciphersuites TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
verb 3
key-direction 1
```

Figura 4.30: Plantilla de cliente

5 Pruebas

Las pruebas son fundamentales en cualquier proyecto software, es una de las piezas claves que nos permiten asegurar que nuestro software cumple con los requisitos establecidos y nos ayuda a realizar cambios y refactorizar nuestro código teniendo la seguridad de que este sigue cumpliendo con los requisitos.

Para asegurar el correcto funcionamiento de **OPENVPN4ALL** se han creado una gran batería de test unitarios y de integración, que se explicarán a continuación.

5.1. Pirámide de testing

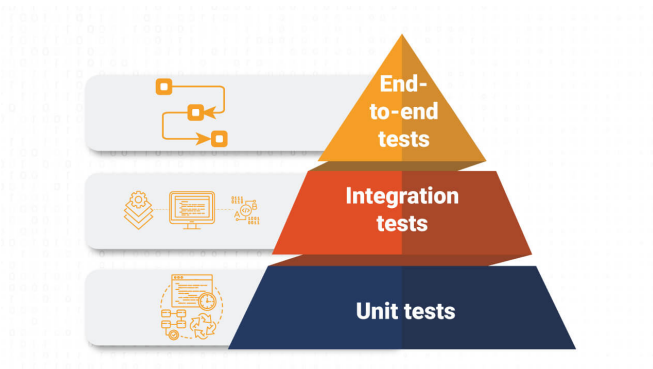

Figura 5.1: Representación gráfica de pirámide de testing

Pirámide de testing 5.1 es un modelo que define como se deben generar los test, este modelo especifica que debe existir una gran base de test unitarios, los cuales deben ser la mayoría. Posteriormente deberemos tener una base importante de test de integración y por último los test del sistema, que serán la minoría.

En este caso particular es algo complejo realizar test de sistema, por lo que han quedado fuera del ámbito del proyecto, pero la base de la pirámide, los cuales son los test unitarios y de integración están más que cubiertos.

5.2. Test unitarios

Los test unitarios son un tipo de test que permiten comprobar el correcto funcionamiento de una unidad aislada de código. En este caso la unidad aislada representa un método de una clase.

Las clases que tienen una mayor carga de lógica y por lo tanto de código son los servicios, pero también hay clases que son críticas y necesitan testearse correctamente, como los validadores que comprueban la entrada de los usuarios. Para testear las clases se ha optado por usar *Mockito* [13] y *Jupiter* [7], de esta manera se pueden crear test unitarios solitarios de forma rápida y sencilla.

En total en el proyecto hay más de 80.000 test 5.2, que permiten comprobar el correcto funcionamiento de la aplicación. Se han testeado los servicios, métodos de validación de la entrada de datos y clases de conversión de datos y parseo.

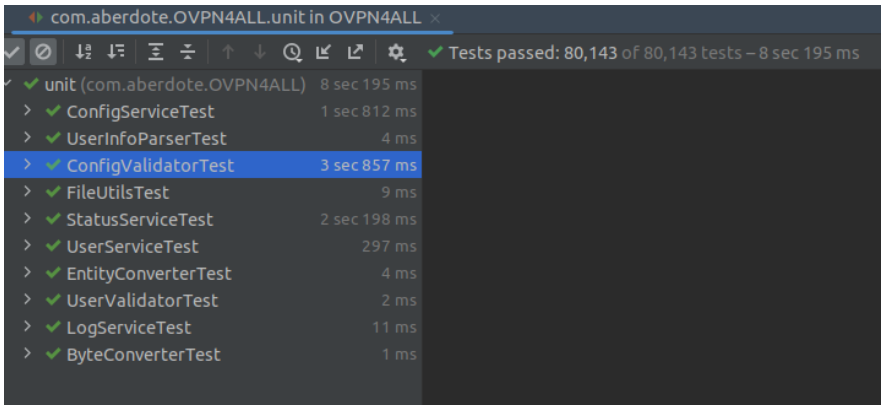

Figura 5.2: Test unitarios

Esta cifra puede parecer algo alta, pero es debido a que se han creado test parametrizables para poder testear correctamente diferentes entradas de los usuarios y verificar que los validadores están funcionando correctamente.

5.3. Test de integración

Los test de integración pueden tener varias interpretaciones, en este caso se han entendido los test de integración como aquellos que permiten verificar el correcta integración de todos los componentes de la aplicación, en este caso se han agrupado los test por controladores, funcionalidad y casuísticas.

En total en el proyecto hay 136 test de integración 5.3, que permiten verificar el correcto funcionamiento de la aplicación.

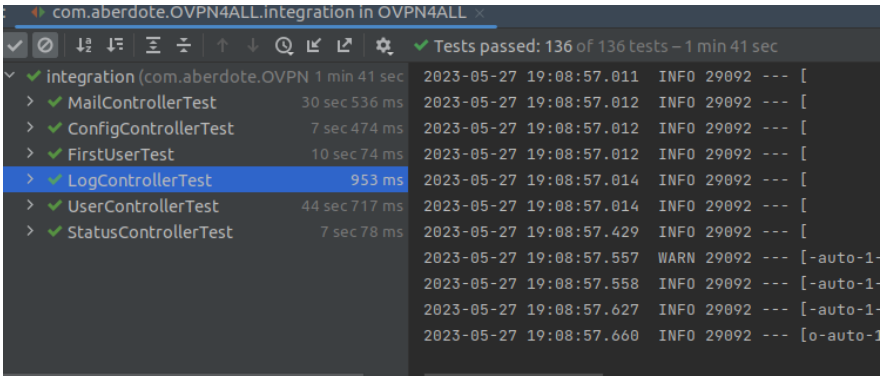

Figura 5.3: Test de integración

Tienen la diferencia con los test unitarios en que estos no usan *mocks* y se ejecutan contra los componentes reales de la aplicación. Esto ha resultado algo complejo, debido a que hay partes de la aplicación que no son nada sencillas de testear y ha llevado bastante tiempo poder plantearlos.

Para elaborar los test de integración se ha utilizado *Jupiter* junto con el framework *RestAssured* [14], el cual permite simular una llamada *REST* y realizar ciertas comprobaciones con la respuesta.

Con la batería de test unitarios y de integración podemos detectar posibles errores y comportamientos no deseados, también es una red de seguridad para futuras modificaciones.

6

Distribución

Una parte fundamental para el éxito de un aplicación es que los usuarios puedan obtener y ejecutar de forma rápida y sencilla.

Esta aplicación no es nada sencilla de distribuir por su arquitectura, además de que los scripts son totalmente dependientes de *Linux*, y en concreto de distribuciones basadas en *Debian*. Para poder distribuirla de forma sencilla se ha pensado en generar una máquina virtual con todo listo para usarse.

Las instrucciones de uso actualizadas se encuentran en el repositorio del proyecto A.1.

Para facilitar la tarea de desplegar los componentes se ha creado un pequeño script, que despliega tanto el Frontend como el Backend y una vez se ha desplegado abre una ventana de firefox con la dirección web del Frontend. También se ha creado un enlace simbólico en la ruta */usr/bin* para que el script se pueda ejecutar desde cualquier ruta.

7

Conclusiones y trabajos futuros

Realizar este proyecto ha sido una experiencia realmente enriquecedora, partí con muy poca experiencia, ya que solo había realizado un proyecto full stack previamente. Tengo algo de experiencia en *Spring* y con *Java* y muy poca experiencia con *SvelteKit*, por lo que el frontend se me hizo algo complicado en algunas partes del proyecto.

He aprendido bastante a largo del proyecto y también he invertido mucho tiempo en él, pero puedo decir que ha merecido la pena y que a día de hoy me veo capaz de realizar proyectos más complejos y basándonos en los errores cometidos en este hacerlo mucho mejor en el futuro.

Cabe destacar que para realizar este proyecto no había mucha información, ya que no se había hecho previamente nada similar o al menos yo no lo he conseguido localizar, por lo que ha habido mucha labor de investigación y por lo tanto el proyecto ha sido bastante complejo en alguna de sus partes. Otra desventaja ha sido mi poca experiencia en el frontend y haber escogido un framework tan poco conocido en la cual no tienes a tu disposición tanta información como otros frameworks.

El resultado final del proyecto es bastante bueno, el sistema funciona bien y cumple con los requisitos establecidos, pero dicho esto y analizando el proyecto me he dado cuenta de que hay muchas cosas por mejorar para que el proyecto alcance un grado de calidad superior y pueda ser usado abiertamente por los usuarios que lo requieran.

El primer error ha sido no haberme centrado primero en construir el backend

directamente en la imagen oficial de *Ubuntu* en *docker*, todo el proyecto se ha desarrollado sobre una máquina virtual y al querer exportarlo a dicha imagen han ocurrido muchos problemas.

Se podrían guardar algunos ficheros directamente en la base de datos, como por ejemplo los ficheros de conexión de los clientes, el propio fichero de configuración del servidor VPN, claves privadas, autoridades de certificación, etc. Ya que se simplificarían bastante algunos procesos de borrado o de creación de usuarios.

El tener información distribuida en dos fuentes (base de datos y sistema de ficheros) hace que el sistema pueda quedar incoherente, algunas operaciones de borrado tienen que borrar en base de datos y en el sistema de ficheros, algo bastante peligroso.

Me gustaría en un futuro solventar estos problemas para poder alcanzar un punto de calidad superior y que los usuarios puedan disponer de una herramienta totalmente gratuita y open-source para crear un servidor VPN, usando la tecnología de *OpenVPN*.

Bibliografía

[1] J. Yonan, 2002, herramienta para la configuración VPN. [Online]. Available: https://openvpn.net/

[2] Oracle, 1995, lenguaje de programación. [Online]. Available: https://www.java.com/es/

[3] SpringSource, 2002, framework para crear aplicaciones web. [Online]. Available: https://spring.io/

[4] ——, 2002, herramienta para crear aplicaciones Spring. [Online]. Available: https://start.spring.io/

[5] ——, 2002, framework para administración de la seguridad en aplicaciones Spring. [Online]. Available: https://docs.spring.io/spring-security/reference/index.html

[6] Estándar para la creación de tokens. [Online]. Available: https://jwt.io/

[7] D. S. Kent Beck, Erich Gamma, biblioteca para creación de test unitarios. [Online]. Available: https://junit.org/junit5/docs/current/user-guide/

[8] Librería para lidiar con código repetitivo. [Online]. Available: https://projectlombok.org/

[9] S. Software, herramienta para documentar servicios web RESTful. [Online]. Available: https://swagger.io/

[10] Herramienta para el parseo XML y Json. [Online]. Available: http://fasterxml.com/

[11] Tecnología para la comunicación entre navegadores y servidores web. [Online]. Available: https://es.wikipedia.org/wiki/WebSocket

[12] Formato de texto para el intercambio de datos. [Online]. Available: https://es.wikipedia.org/wiki/JSON

[13] S. Faber, framework de testing para la creación de objetos dobles. [Online]. Available: https://site.mockito.org/

[14] J. Haleby, framework para testing de servicios REST. [Online]. Available: https://rest-assured.io/

[15] R. Hat, 2001, herramienta de mapeo objeto-relacional. [Online]. Available: https://hibernate.org/

[16] B. Fox, 1989, lenguaje de scripting para Linux. [Online]. Available: https://www.gnu.org/software/bash/

[17] R. Russell, 1998, programa para gestionar las tablas del firewall en Linux. [Online]. Available: https://www.netfilter.org/

[18] S. Hykes, 2013, herramienta para la creaciónd de contenedores de software. [Online]. Available: https://docs.spring.io/spring-security/reference/index.html

[19] ——, herramienta para ejecutar varios contenedores Docker. [Online]. Available: https://docs.docker.com/compose/

[20] R. Harris, 2016, framework para construir aplicaciones web. [Online]. Available: https://kit.svelte.dev/

[21] Microsoft, 2012, lenguaje de programación basado en javascript. [Online]. Available: https://www.typescriptlang.org/l

[22] Netscape, 19955, lenguaje de programación interpretado. [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript

[23] W. W. W. Consortium, 1993, lenguaje de marcado para páginas web. [Online]. Available: https://es.wikipedia.org/wiki/HTML

[24] T. Labs, 2017, framework de utilidad para CSS. [Online]. Available: https://tailwindcss.com/

[25] R. Dahl, 2009, entorno de ejecución multiplataforma. [Online]. Available: https://nodejs.org/es

[26] Oracle, 2000, utilidad para gestionar autoridades de cerificación. [Online]. Available: https://www.mysql.com/

[27] C. Ltd, 2004, distibución de Linux basada en Debian. [Online]. Available: https://ubuntu.com/

[28] Oracle, 2000, sistema de gestión de bases de datos. [Online]. Available: https://www.mysql.com/

[29] Atlassian, interfaz gráfica para git. [Online]. Available: https://www.sourcetreeapp.com/

[30] JetBrains, entorno de desarrollo para Java. [Online]. Available: https://www.jetbrains.com/es-es/idea/

[31] Microsoft, 2015, editor de código multilenguaje. [Online]. Available: https://code.visualstudio.com/

[32] L. Torvalds, 2007, software de control de versiones. [Online]. Available: https://git-scm.com/

[33] T. Preston-Werner, 2008, plataforma de desarrollo colaborativo para alojar proyectos software. [Online]. Available: https://github.com/

[34] P. Inc, 2012, plataforma para la construcción y testeo de APIs. [Online]. Available: https://www.postman.com/

[35] S. Rider, 2010, cliente SQL. [Online]. Available: https://dbeaver.io/

[36] Sistema para cargar contenido eficientemente en páginas web. [Online]. Available: https://es.wikipedia.org/wiki/Script_del_lado_del_servidor

[37] Sistema para controlar el servidor OpenVPN via TCP. [Online]. Available: https://openvpn.net/community-resources/management-interface/

[38] Patron de diseño para crear clases con instancias únicas. [Online]. Available: https://es.wikipedia.org/wiki/Singleton

[39] Codificador de contraseñas seguro. [Online]. Available: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html

[40] Protocolo de red seguro. [Online]. Available: https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte

Apéndices

A

Apéndice 1

A.1. Enlace del proyecto

El proyecto ha sido alojado en la plataforma *Github*, es open-source y todo el mundo que lo desee puede aportar al proyecto y usarlo libremente.

Enlace: OpenVPN4ALL