Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo de un front-end en ReactJS para un gestor de APIs

Autor:
Marc Yeste Antoli

Supervisor:
Rafael Goterris
Tutor académico:
Rafael Berlanga Llavori

Fecha de lectura: 13 de Julio de 2023
Curso académico 2022/2023

Resumen

En este documento se redactará el proceso de desarollo llevado a cabo durante tres meses en la empresa CloudAPPi SL[8], para obtener como resultado una API que ayuda a las empresas o particulares en la creación de nuevas APIs, y en la gestión y desarrollo de las mismas. El proyecto anteriormente mencionado recibe el nombre de APIQuality [13] y está desarrollado por un equipo de desarrolladores con metodología SCRUM. El producto final permitirá al usuario importar APIs, escanear la definición de las APIs en busca de vulnerabilidades y crear o importar guías de estilo para comprobar que las APIs las respetan.

En la memoria se documenta, la planificación del proyecto, el seguimiento del mismo, la forma de trabajar con la metodología SCRUM, y las distintas funcionalidades desarrolladas en cada sprint como desarrollador Front-End del proyecto.

Palabras clave

Desarrollo de API, Scrum, ReactJS, Redux, API First

Keywords

API development, Scrum, ReactJS, Redux, API First

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Las API [16](Interfaces de Programación de Aplicaciones, por sus siglas en inglés) se han vuelto fundamentales en el mundo empresarial debido a varios factores. Las empresas están abandonando las grandes aplicaciones monolíticas en favor de componentes más pequeños y microservicios independientes. Las API permiten la comunicación entre estos componentes, facilitando la integración y ofreciendo servicios en tiempo real.

Una API[15] es un conjunto de reglas y protocolos que permite a diferentes aplicaciones y sistemas interactuar entre sí de manera estandarizada. Proporciona una interfaz de comunicación que define cómo las aplicaciones pueden solicitar y compartir datos, funcionalidades o servicios específicos.

Una API web consta de varias partes. En primer lugar, los endpoints son los puntos de acceso definidos dentro de la API a través de los cuales se pueden realizar solicitudes y recibir respuestas. Estos endpoints actúan como puertas de entrada para acceder a diferentes funciones o recursos de la API. Cada endpoint tiene una URL específica que indica su ubicación y puede estar asociado con diferentes métodos HTTP (Hypertext Transfer Protocol), como GET, POST, PUT o DELETE, para realizar diferentes operaciones.

Además de los endpoints, una API puede incluir parámetros que permiten a los usuarios enviar datos adicionales con sus solicitudes. Estos parámetros pueden ser de diferentes tipos, como consultas en la URL (parámetros de consulta), datos enviados en el cuerpo de la solicitud (parámetros de formulario) o incluso información de autenticación para validar el acceso a recursos protegidos.

Dentro del ámbito de las API, OpenAPI es una especificación ampliamente utilizada que proporciona un formato común y estandarizado para describir y documentar las API. También conocida como Swagger, OpenAPI permite definir la estructura, el comportamiento y los detalles técnicos de una API de manera clara y precisa.

OpenAPI utiliza un formato basado en JSON o YAML para describir la API, lo que facilita su comprensión tanto para humanos como para máquinas. Esta especificación describe los endpoints disponibles en la API, los métodos de solicitud permitidos (GET, POST, PUT, DELETE, etc.), los parámetros requeridos y opcionales, los formatos de datos admitidos y las respuestas esperadas.

Al utilizar OpenAPI, los desarrolladores y usuarios de la API pueden obtener una descripción detallada de cómo interactuar con ella. Esto incluye información sobre los endpoints disponibles, la estructura de las solicitudes y respuestas, los posibles códigos de estado HTTP y los esquemas de datos utilizados. Además, OpenAPI permite agregar anotaciones adicionales, como descripciones, ejemplos y validaciones, que ayudan a comprender y utilizar correctamente la API.

Dentro de los sistemas de información de las empresas, las API desempeñan un papel crucial al permitir la interconexión y comunicación fluida entre diferentes aplicaciones y servicios. Al proporcionar una interfaz estandarizada, las API permiten que las aplicaciones se comuniquen entre sí de manera eficiente, independientemente de las tecnologías subyacentes utilizadas en cada una. Esto facilita la integración de sistemas heterogéneos y fomenta la reutilización de componentes y datos, lo que a su vez reduce los costos de desarrollo y mantenimiento.

Además, las API también permiten a las empresas abrir sus datos y funcionalidades para su uso por parte de terceros, como socios comerciales o desarrolladores externos. Esto fomenta la colaboración y la creación de un ecosistema diverso de aplicaciones y servicios que pueden ampliar las capacidades y el alcance de las empresas. Las API también brindan a las empresas la capacidad de monetizar sus servicios al ofrecer acceso controlado y pago por uso a través de APIs públicas.

En este contexto, CloudAPPi ha concebido la idea de crear una herramienta llamada API Quality. API Quality es un producto basado en la nube (SaaS) que tiene como objetivo permitir a las empresas diseñar, validar y desplegar sus API de una manera más eficiente y segura, todo siguiendo los enstándares de OpenAPI, y a través del concepto de API First que más tarde se explicará.

1.2. Objetivo y alcance del proyecto

Como se ha expuesto en la sección anterior con API Quality se quiere conseguir una herramienta que mejore el proceso de creacion y gestion de APIs para empresas y desarrolladores, lo que les permitira ahorrar tiempo y recursos mientras garantizan la calidad y seguridad de su API.El equipo que respalda esta herramienta utiliza la metodología ágil SCRUM para su trabajo, la cual se explicará en mayor detalle más adelante. Como resultado, cada miembro del equipo tiene objetivos individuales específicos.

Dentro del equipo, mis objetivos como desarrollador Front-End son los siguientes:

- Desarrollar componentes en React[10] que permita a los usuarios interactuar con la aplicación de manera sencilla y sin complicaciones.

- Estudiar los Hooks de React y crear custom Hooks para mejorar la eficiencia y la reutilización del código en la aplicación.

- Estudiar e implementar la metodología Redux para manejar el estado global de la aplicación y mejorar su escalabilidad.

- Garantizar una integración adecuada entre los componentes Front-End y Back-End de la aplicación mediante una comunicación constante y efectiva con el equipo de Back-End.

- Testear y validar los componentes de la aplicación para garantizar su calidad y seguridad.

- Asistir a la planificación de las tareas de desarrollo. y a las reuniones diarias de seguimiento del proyecto.

- Definicion de recursos de APIs con OpenAPI

- Documentar los aspectos técnicos y funcionales de las tareas desarrolladas.

1.3. Estructura de la memoria

Una vez hecha ya la introducción con el contexto y los objetivos del proyecto, dividiremos el resto de memoria en tres partes principales. El siguiente capítulo describe la empresa donde se ha hecho la estancia de prácticas, el alcance del proyecto, las herramientas utilizadas y la metodología elegida para desarrollar el proyecto.

A continuación, se describirá la planificación temporal y seguimiento del proyecto, así como la estimación de costes del mismo, en la que se tiene en cuenta tanto el gasto de personal como el de las herramientas utilizadas.

Finalmente, el último capítulo lo dedicaremos a la implementación de las distintas tareas del proyecto, donde se expondrá la definición de la tarea, las dificultades encontradas y la solución final.

Capítulo 2

Descripción del proyecto

2.1. Descripción de la empresa

La empresa donde se realizan las prácticas es CloudAPPi SL, cuya sede principal se encuentra en Madrid. También cuenta con una sede en Castellón en el edificio Espaitec 2 situado dentro del Campus del Riu Sec, donde se han realizado las prácticas.

CloudAppi es una empresa líder en el desarrollo de aplicaciones y en la transformación digital a través de las API. Su objetivo es ayudar a las empresas y startups a alcanzar una ventaja competitiva mediante esta tecnología. En este sentido, CloudAppi cuenta con un equipo altamente especializado en el desarrollo de soluciones informáticas personalizadas para cada cliente, con el objetivo de ayudarles a mejorar sus procesos y optimizar sus resultados.

Este proyecto nace de la necesidad de la propia empresa de gestionar la creación de APIs y hacer el proceso más fácil e intuitivo. APIQuality servirá tanto a los proyectos internos que CloudAppi está desarrollando, como a otras empresas que estén interesadas en contratar los servicios de APIQuality.

2.2. Diseño y arquitectura del proyecto

El proyecto se ha construido de acuerdo a la arquitectura cliente-servidor[20]. Esta tecnología se basa en el principio de separación de preocupaciones. El cliente se centra en la presentación y la interacción con el usuario, mientras que el servidor se encarga del procesamiento de datos y la lógica empresarial. Esta separación permite una mayor modularidad y escalabilidad de la aplicación, ya que cada componente puede evolucionar de manera independiente sin afectar al otro.

La comunicación entre el cliente y el servidor [21] se hace mediante el protocolo HTTP (Hypertext Transfer Protocol). Este protocolo consiste en que el cliente envía una solicitud HTTP al servidor para obtener información o realizar una acción específica. Esta solicitud puede

ser una solicitud GET para obtener datos, una solicitud POST para enviar datos al servidor, una solicitud PUT o PATCH para actualizar datos existentes, o una solicitud DELETE para eliminar datos. El servidor recibe la solicitud del cliente y procesa la acción solicitada. Esto puede incluir la consulta de una base de datos, ejecutar lógica empresarial, autenticar al usuario, etc. Después de procesar la solicitud, el servidor genera una respuesta HTTP que contiene el resultado de la acción solicitada y la envía al cliente.

Para el desarrollo del cliente en el proyecto se decidió utilizar el patrón de diseño Redux implementado con React. Por una parte, React es una biblioteca de JavaScript que se utiliza para construir interfaces de usuario interactivas y reutilizables. Su objetivo principal es simplificar el desarrollo de aplicaciones web al permitir la creación de componentes reutilizables y manejar eficientemente las actualizaciones de la interfaz en función de los cambios en los datos. Los principios de diseño de React son los siguientes:

- **Componentes**: React se basa en el concepto de componentes, que son bloques de construcción reutilizables y autocontenidos de una interfaz de usuario. Los componentes pueden contener lógica y estado propios, y se pueden combinar para formar interfaces más complejas.
- **Unidireccionalidad**: React sigue un flujo de datos unidireccional, lo que significa que los datos fluyen desde un componente principal hacia sus componentes secundarios. Esto ayuda a mantener un estado predecible y facilita el seguimiento de cómo cambian los datos en la aplicación.
- **Virtual DOM**: React utiliza una representación en memoria de la interfaz de usuario llamada "Virtual DOM". Esta representación permite que React realice comparaciones eficientes entre el estado actual de la interfaz y los cambios que deben aplicarse, minimizando las manipulaciones directas del DOM y mejorando el rendimiento.
- **Reconciliación**: React realiza una comparación eficiente entre el Virtual DOM y el DOM real para determinar los cambios mínimos necesarios y actualizar solo las partes afectadas. Esto se conoce como reconciliación, y permite una actualización eficiente de la interfaz de usuario sin necesidad de recargar la página completa.

Por otra parte, Redux es un patrón de diseño para gestionar el estado de la aplicación, especialmente en aplicaciones grandes y complejas. Redux cuenta con varias partes principales:

- **Store**: El estado global de la aplicación se almacena en un objeto llamado "store". El store es inmutable y se mantiene en un solo lugar. Solo puede modificarse a través de acciones.
- **Actions**: Son objetos que describen un cambio en el estado de la aplicación. Una acción tiene un tipo y opcionalmente puede tener datos adicionales asociados.
- **Dispatcher**: Es una función que toma una acción y la envía al store. Es el punto central donde todas las acciones pasan antes de ser procesadas por el store.
- **Reducer**: Es una función pura que recibe una acción y el estado actual, y devuelve un nuevo estado. El reductor especifica cómo debe actualizarse el estado en función del tipo

de acción recibida. Los reducers son responsables de realizar cambios en el estado del store.

- **Middleware**: El middleware proporciona una extensibilidad adicional al proceso de Redux. Puede interceptar acciones antes de que lleguen al reductor, realizar operaciones asíncronas, realizar registro de errores, etc.

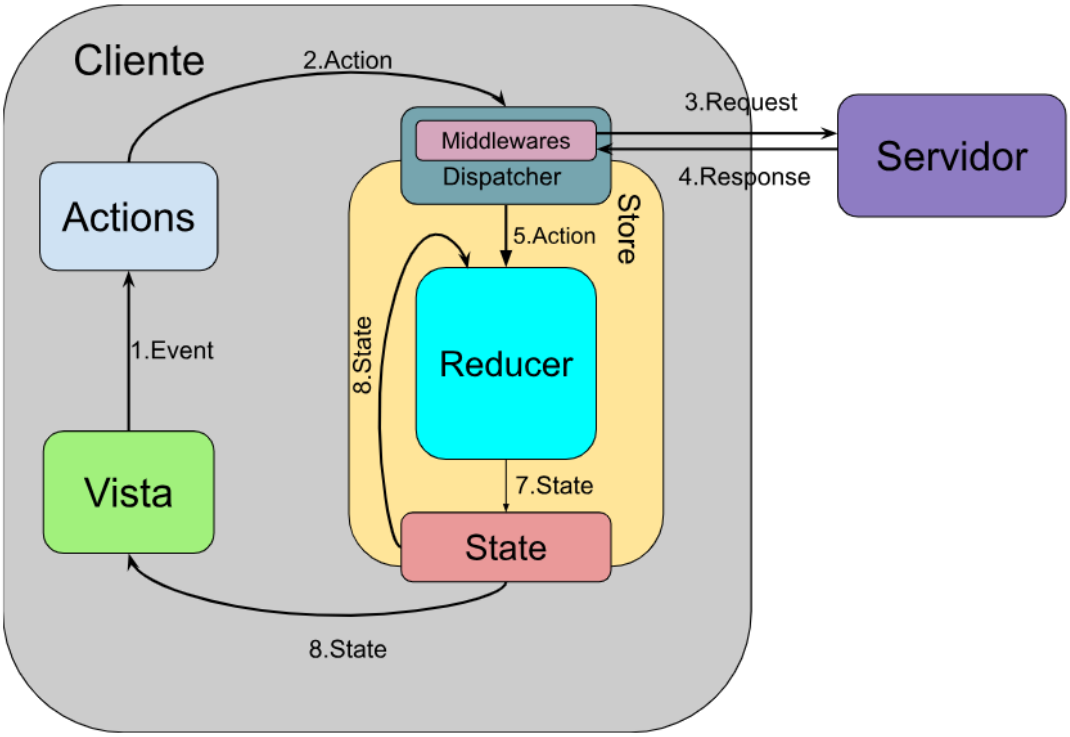

Figura 2.1: Funcionamiento de Redux. [18]

En la Figura 2.1 se puede ver flujo de datos del sistema usando la metodología Redux en el cliente para manejar el estado de la aplicación, y como se comunica con el servidor mediante el protocolo HTTP. El funcionamiento del sistema se inicia con la interacción del usuario con la vista creada con los componentes de React. Estos componentes lanzan un evento que a su vez llama a una acción. La acción es pasada al dispatcher, que mediante el middleware se comunica con el servidor utilizando el protocolo HTTP. El servidor se encarga de procesar la petición y devuelve una respuesta. Esta respuesta es recogida por el middleware donde se realiza en registro de errores. Si no hay errores, se pasa la respuesta al reducer que se encarga de actualizar el state. Por último, se manda el nuevo estado para que la vista se actualice y el usuario vea los cambios.

2.3. Definición del proyecto

El proyecto APIQuality, en el que he estado trabajando durante la estancia de prácticas en la empresa CloudAPPi SL, pretende ser una herramienta para que los desarrolladores puedan centrarse en la creación de una API de calidad. Mientras la herramienta se encarga de supervisar y ayudar en todos los pasos de su creación, todo esto a través del concepto de API First.

API First es un enfoque que se basa en la premisa de que una API bien diseñada es la base para una aplicación sólida y escalable. Al diseñar la API primero, los desarrolladores pueden asegurarse de que los servicios necesarios estén disponibles y de que sean fáciles de usar para otros equipos que desarrollen aplicaciones. Esto permite que los desarrolladores se centren en la funcionalidad de la aplicación en sí, sabiendo que la API proporcionará los servicios necesarios para su implementación.

Para cumplir con todos estos requisitos API Quality cuenta con una serie de herramientas:

- **Definición**: Como herramientas de definición cuenta, por una parte, con la posibilidad de manipular los archivos de definición, escanear los errores de definición y escanear las vulnerabilidades. Por otra parte, se pueden definir estándares corporativos de definición, estándares de API security y perfiles a diferentes estilos de APIs, ver Figura 2.2.

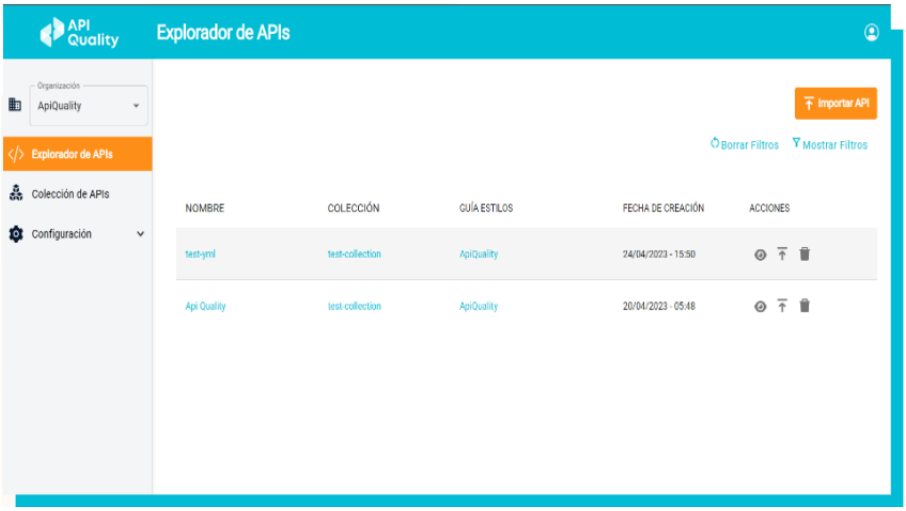

Figura 2.2: Vista de definición de APIs

- **Bugs de definición**: Permite localizar bugs de definición y definir perfiles de reglas de estilo. Una vez encontrados los bugs, la herramienta ofrece las posibles soluciones y muestra las estadísticas de los bugs, ver Figura 2.3.

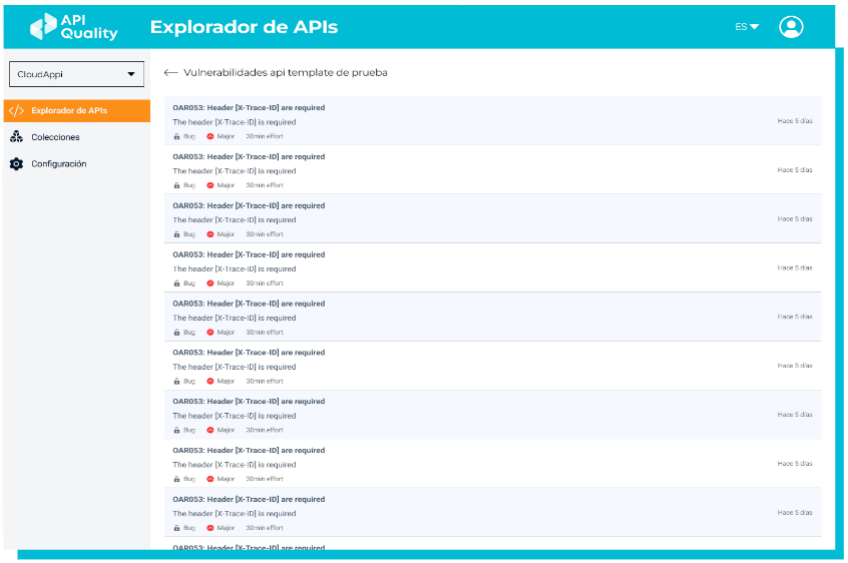

Figura 2.3: Vista de bugs de definición

- **Vulnerabilidades de seguridad**: Al igual que con los bugs de definición, también permite localizar vulnerabilidades de seguridad (OWASP) y ofrece las posibles soluciones y muestra las estadísticas, ver Figura 2.4.

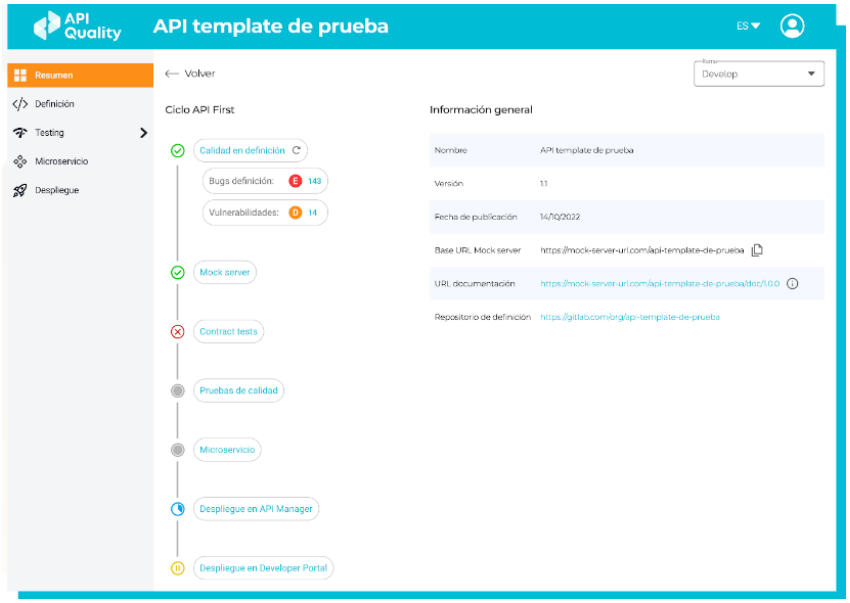

Figura 2.4: Vista de vulnerabilidades de seguridad

- **Testing**: Como herramientas de testing cuenta creación de mocks, contract testing, Pentesting y validación de parámetros.

- **Despliegue**: Para la ayuda con el despliegue de las APIs cuenta con la generación de arquetipo, monitorización y login, despliegue en los principales API Managements del mercado y vinculación con Developer portal.

Por otra parte, en APIQuality existe el concepto de organización, todas las API pertenecen a una organización. Cada usuario puede tener más de una organización donde importar las API, así como crearlas si es necesario. También se puede invitar a otro usuario a tu organización y darle permisos para que pueda editar las API o las guías de estilo.

Todas estas funcionalidades has sido desarrolladas por un equipo con metodología de trabajo SCRUM, en la que se entrará en detalle en la próxima sección. Dentro de este equipo entré en el equipo de desarrollo de Front-End. La estancia de prácticas empezó en el tramo final de desarrollo del proyecto, por lo que muchas funcionalidades ya estaban implementadas. En el capítulo 4 se entrará en detalle en las implementaciones realizadas durante las prácticas.

2.4. Metodología

Para desarrollar el proyecto me uní a un equipo que estaba trabajando con la metodología ágil **SCRUM** [19].

Se decidió usar esta metodología, ya que permite a los equipos desarrollar productos de manera rápida y adaptativa, fomentando la colaboración y la comunicación efectiva entre todos los miembros del equipo.

En Scrum, el trabajo se organiza en ciclos llamados "sprints". Cada sprint tiene una duración fija, en nuestro caso de 2 semanas o 10 dias laborales, durante la cual se realiza una cantidad de trabajo específica. Al comienzo de cada sprint, el equipo se reune en una reunión de planificación del sprint, llamada sprint planning, para definir los objetivos y las tareas que se abordarán en ese periodo y estimar el tiempo que se dedicará a cada tarea por parte de los miembros del equipo.

El equipo de desarrollo en esta metodología está compuesto por tres roles principales:

- **Product Owner**: Es el encargado de definir los requisitos del producto, así como optimizar y maximizar el valor del producto. El PO establece la visión, las prioridades y administra el backlog [14] del producto(una lista de todas las funcionalidades y tareas pendientes). Por otra parte, el Product Owner también tiene que marcar el objetivo de cada sprint de manera clara y acordada con el equipo de desarrollo.

- **Scrum Master**: Se encarga de que se sigan los principios de SCRUM y de mantener la comunicación entre todo el equipo. En nuestro equipo estaa siempre en contacto con el

Product Owner y nos transmite los objetivos de cada sprint en las reuniones que el propio Scrum Master dirige. Por otra parte, también es el encargado de convocar todos los días una reunión.

- **Equipo de Desarrollo**: Está formado por profesionales encargados de diseñar, desarrollar y probar el software. Nuestro equipo de desarrollo se divide en dos, el equipo de Front-End y el de Back-End, cada equipo con 2 desarrolladores y un arquitecto de software que se encarga de ayudar a los desarrolladores en las dificultades que se encuentran, supervisar las tareas y dar el visto bueno para que el código pasase a producción.

El flujo de trabajo de cada sprint empieza con el anteriormente mencionado sprint planning, una reunión donde el product owner y el scrum master exponen los objetivos y tareas de cada sprint. Una vez expuestas las tareas, el equipo de desarrollo estima las tareas. Para la estimación de las tareas, cada miembro del equipo estima el tiempo que le costaría realizar la tarea y luego se hace una media con los tiempos estimados por cada miembro, de donde sale la estimación final. Una vez expuestas y estimadas todas las tareas, el scrum master repartía las tareas del sprint a los miembros del equipo de desarrollo.

Durante un sprint, el equipo se reune diariamente en la llamada "Daily", una reunión breve de 30 minutos donde cada miembro expone los avances realizados en la jornada anterior, si se ha encontrado con algún obstáculo y los planes para esa jornada.

Al final de cada spint, se lleva a cabo una revisión del sprint o "Sprint Retrospective", donde los miembros del equipo de desarrollo muestran las funcionalidades implementadas. También es el momento en el que el equipo reflexiona sobre su rendimiento y busca formas para mejorar el proceso de trabajo. Por último, los arquitectos prueban las nuevas funcionalidades y si dan el visto bueno sube los cambios del entorno develop, donde el equipo de desarrollo hace sus pruebas, a producción.

Para terminar con la metodología usada para este proyecto, hay un equipo llamado Quality Assurance (QA), que se encarga de realizar pruebas exhaustivas y rigurosas en el entorno de producción para identificar errores, defectos o fallos en el software. Si encuentran alguna incidencia se lo comunican al scrum master y este asigna las incidencias al equipo de desarrollo.

2.5. Herramientas utilizadas en el proyecto

En este apartado se nombrarán las herramientas empleadas para la realización del proyecto APIQuality:

- **Visual Studio Code**: Editor de código fuente que ofrece una interfaz de usuario altamente personalizable y extensible, con soporte para una amplia variedad de lenguajes de programación, en nuestro caso lo utilizamos con JavaScript y React.js.[12]

- **GitLab**: Plataforma de desarrollo colaborativo basada en web que proporciona herramientas para la gestión de repositorios de código fuente, seguimiento de problemas, integración continua y despliegue continuo (CI/CD), y colaboración en proyectos de software.[7]

- **Redmine**: Proporciona herramientas y funcionalidades para la planificación, seguimiento y control de proyectos, así como la colaboración entre los miembros del equipo. Esta herramienta ayudaba al scrum master a organizar tareas, asignar responsabilidades, gestionar el tiempo y mantener un registro de problemas y su resolución.[11]

- **Figma**: Herramienta de diseño y prototipado colaborativa basada en la nube. Ofrece un entorno de diseño colaborativo que permite a los miembros de un equipo trabajar en proyectos de manera simultánea y en tiempo real. Esto significa que múltiples diseñadores pueden editar y comentar sobre el mismo archivo al mismo tiempo, lo que fomenta la colaboración y agiliza el proceso de diseño.[6]

- **Microcks**: Herramienta de código abierto utilizada para la generación automática de mocks, pruebas y documentación de API. A Microcks le pasamos la especificación OpenAPI para generar automáticamente documentación detallada de nuestra API. Esta documentación incluye descripciones de los endpoints, parámetros esperados, códigos de respuesta, ejemplos y otra información relevante para facilitar el uso y la comprensión de la API al equipo de desarrollo.[9]

- **Redux DevTools**: Es una extensión para el navegador que ofrece herramienta de desarrollo para el estado de gestión de aplicaciones JavaScript utilizando la biblioteca Redux. Proporciona una interfaz gráfica que permite visualizar y depurar los cambios en el estado de la aplicación a lo largo del tiempo.[5]

Debido a la metodología de trabajo SCRUM, en la que es muy importante la comunicación, y que el equipo de APIQuality tiene miembros en Madrid y Castellón, se decidió usar Google Meet, para las reuniones mencionadas en la sección anterior, y Google Chat para que la comunicación entre los miembros del equipo sea rápida y eficaz.

Capítulo 3

Planificación del proyecto

3.1. Planificación temporal del proyecto

A continuación, mediante un diagrama de Gantt (Figura 3.1), se presentará la planificación temporal del proyecto a lo largo de las 300 horas de prácticas realizadas.

Debido a que el proyecto en el que iba a trabajar usaba React como lenguaje de programación y no lo había usado nunca, las prácticas empezaron con un periodo de formación de 35 horas, donde hice un curso de Udemi llamado *React, de cero a experto*. Una vez terminado el periodo de formación me uní al equipo de desarrollo de APIQuality y la metodología de trabajo ágil SCRUM, que dividía el tiempo de desarrollo en sprints.

El resumen de los sprints realizados en el proyecto se puede ver en la Tabla 3.1 y en el Capítulo 4 se explicará las tareas realizadas en cada sprint de forma detallada.

Tabla 3.1: Sprints realizados en el proyeto

Sprint 1	Estudio del código y vincular Google Analitycs.
Sprint 2	Crear un nuevo modal con un selector con búsqueda.
Sprint 3	Selector de idiomas.
Sprint 4	Crear un nuevo endpoint.
Sprint 5	Añadir posibilidad de iniciarse sesión externamente.
Sprint 2	Estudio y modificación de estilos css.

Nombre de tarea	Dura	Comienzo	Fin
Proyecto	**300 hrs**	**mié 01/03/23**	**vie 26/05/23**
Formación	**40 hrs**	**mié 01/03/23**	**mar 14/03/23**
Onboarding	5 hrs	mié 01/03/23	jue 02/03/23
React: de cero a experto	35 hrs	jue 02/03/23	mar 14/03/23
Desarrollo Proyecto	**260 hrs**	**mar 14/03/23**	**vie 26/05/23**
Sprint 1	50 hrs	mar 14/03/23	lun 27/03/23
Sprint 2	50 hrs	mar 28/03/23	mié 12/04/23
Sprint 3	50 hrs	jue 13/04/23	mié 26/04/23
Sprint 4	50 hrs	jue 27/04/23	mié 10/05/23
Sprint 5	50 hrs	jue 11/05/23	mié 24/05/23
Sprint 6	10 hrs	jue 25/05/23	vie 26/05/23

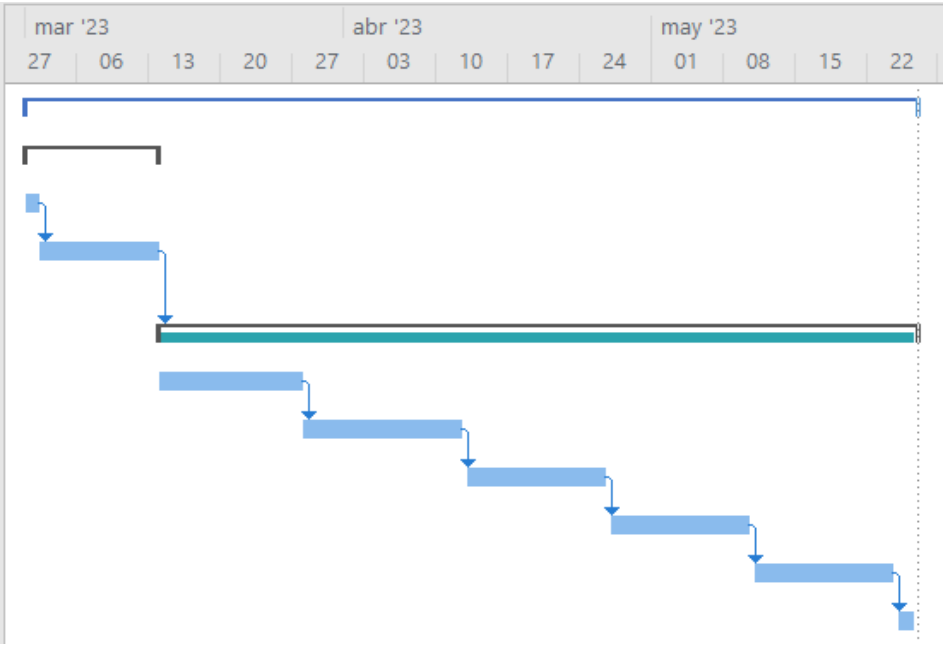

Figura 3.1: Diagrama de Gantt de la planificación inicial del proyecto

3.2. Seguimiento del proyecto

3.2.1. Control del proyecto

La jornada de trabajo para desarrollar el proyecto ha sido de 5 horas diarias de lunes a viernes. La jornada empezaba a las 9:00 y terminaba a las 14:00, aunque el horario era flexible, podía entrar más tarde y salir más tarde, mientras se cumplieran las 5 horas diarias. Algunos días teníamos las reuniones de equipo por la tarde por la disposición de algunos miembros del equipo, por lo que terminaba un poco antes de las 14:00 y me conectaba a las reuniones por la tarde. Las prácticas empezaron el 1 de Marzo y terminaron el 26 Mayo, aunque como se ha explicado en la sección anterior, empecé con un periodo de formación y me uní al equipo de desarrollo de APIQuality el 14 de Marzo.

Para el control del correcto desarrollo del proyecto, el equipo seguía la metodología de trabajo ágil SCRUM anteriormente descrita. El equipo se reunía todos los días para explicar el trabajo desarrollado en la jornada anterior, estas reuniones también servían para comentar y resolver entre todos las dificultades encontradas en el desarrollo de las tareas. Aparte de estar reuniones diarias, estaban las reuniones para planificar los objetivos principales de cada sprint, donde se estimaba el tiempo empleado para cada tarea. Finalmente, al final de cada sprint se hacía otra reunión donde se repasaba si se cumplían los objetivos expuestos en el sprint planning y se debatía si había alguna forma de mejorar la forma de trabajar del equipo.

Para el seguimiento del proyecto han sido muy útiles todas las herramientas citadas en la Sección 2.4, pero en especial el Redmine ha sido la que más nos ha ayudado con el control del desarrollo del proyecto. Con esta herramienta, cada miembro del equipo de desarrollo podía ver las tareas que se le habían asignado en el sprint con su descripción, nivel de prioridad y tiempo estimado para el desarrollo de la tarea. Las tareas podían estar en diferentes estados:

- **Pendiente**: El desarrollador aún no se había puesto con la tarea.
- **En curso**: El desarrollador estaba trabajando en la tarea. Dentro de este estado, el desarrollador podía poner un porcentaje, que indicaba el avance de la tarea, y el tiempo dedicado a la tarea.
- **Bloqueada**: El desarrollo había encontrado un problema que no le permitía seguir con el desarrollo de la tarea. Normalmente, era porque faltaba implementar algo en el Back o en el Front para continuar.
- **Code Review**: La tarea ya se había finalizado y el desarrollador había abierto un Merge Request para que el arquitecto validara los cambios.
- **QA Validation**: El Merge Request ya había sido validado y la tarea pasaba a la revisión por parte del equipo de Quality Assurance.

Todas estas características ayudan a los desarrolladores a planificar su sprint, pero sobre todo, ayudan al scrum master a controlar que el sprint se esté desarrollando con normalidad o si hay algún problema.

3.2.2. Desviación de la planificación

En cuanto a la desviación respecto a la planificación inicial del proyecto. Si bien todos los sprints empezaron y terminaron cuando se esperaba, al ser un equipo relativamente grande y con muchas tareas, siempre se quedaba alguna tarea menor a medio hacer. En estos casos la tarea se pasaba al siguiente sprint y se priorizaba para que terminase en uno o dos días como mucho.

Por otra parte, estaban los problemas o errores que encontraba el equipo de Quality Assurance. Este equipo creaba una tarea por cada error que se encontraba, y eran asignadas a los miembros del equipo. Estas tareas no estaban en la planificación inicial del proyecto, por lo que siempre retrasaban un poco el resto de tareas del sprint.

Finalmente, durante el tercer sprint tuve una lesión en la rodilla y me operaron el 18 de abril. Debido a la operación, estuve de baja hasta el lunes 24 de abril. Esto no afectó a la planificación general del sprint, que se pudo realizar en el tiempo estimado, pero sí afectó a la repartición de las tareas, ya que los otros miembros del equipo se repartieron mis tareas y yo hice una tarea menor, que consistía en crear un selector de idiomas en la configuración del perfil.

3.3. Estimación de recursos y costes

En esta sección trataremos de estimar el coste del proyecto y los recursos utilizados para desarrollarlo. Durante el análisis se tendrán en cuenta tanto los costes de recursos humanos como el hardware y software que han sido utilizados para implementar el proyecto.

Para esta sección hay que tener en cuenta que mi estancia de prácticas empieza cuando el proyecto ya llevaba tiempo en desarrollo. Al no saber el tiempo real de desarrollo del proyecto, ya que al terminar las prácticas el proyecto aún no estaba terminado, en esta estimación se tendrá en cuenta solo los costes durante el transcurso de las prácticas.

En cuanto a los costes de los recursos humanos. Contamos con que el proyecto ha sido desarrollado por un equipo con metodología SCRUM, que como se explica en la Sección 2.3, está formado por un Product Owner, que al ser el propietario de la empresa no supone un coste, el Scrum Master, que era un programador veterano, los dos arquitectos, el de Back y el de Front, dos programadores junior y dos becarios. Desconociendo los salarios reales de los empleados de la empresa, se ha optado por hacer la estimación con el salario medio en España de cada puesto [17].A continuación se muestra una estimación de los gastos de recursos humanos a lo largo de las 300 horas de prácticas.

En cuanto a los gastos en software para la implementación del proyecto han sido nulos, ya que todos los programas citados en la sección 2.3 son gratuitos o como es el caso de Udemy, no han supuesto un gasto extra en la empresa, puesto que ya contaban con una cuenta creada y usada por más usuarios de la empresa. Por otra parte, con los gastos de hardware realizaremos una estimación en cuanto al coste proporcional al uso de estos durante el desarrollo del proyecto, ya que pueden ser reutilizados por otros empleados. El hardware aportado por la empresa ha sido de un ordenador portátil, contando con un periodo de amortización de 4 años y con que

Tabla 3.2: Desglose de gastos en recursos humanos

Rol	**Precio (€/hora)**	**Horas**	**Coste total**
Product Owner	0	300	0
Scrum Master	15,00	300	4500,00
Arquitecto 1	25,00	300	7500,00
Arquitecto 2	25,00	300	7500,00
Programador Junior 1	10,00	300	3000,00
Programador Junior 2	10,00	300	3000,00
Becarios	0	300	0
Coste total de todos los roles			22500

las prácticas han durado 2 meses:

$$\text{Precio mes} = [500\ €] \cdot \left[\tfrac{1\ \text{año}}{12\ \text{meses}}\right] \div [4\ \text{años}] = 10{,}41\ €/\text{mes}$$

$$\text{Coste Total del Hardware} = [\text{Precio mes}] \cdot [2\ \text{meses}] = 20{,}83\ €$$

Finalmente, una vez calculado los costes por separado de empleados, el software y el hardware, se puede calcular el coste total del proyecto durante el transcurso de la estancia de practicas. Este coste es igual a **22.520,83€**.

Capítulo 4

Implementación del proyecto

En este capítulo abordaremos las principales tareas que he desarrollado durante cada sprint. Para ello, primero se describirá con detalle la tarea a realizar, seguidamente, se expondrá la investigación hecha antes de realizar la tarea y las dificultades encontradas durante el desarrollo. Finalmente, se explicará la implementación final de la tarea.

4.1. Sprint 1: Estudio del código

Antes de empezar con el primer sprint hice el curso de Udemy React de cero a experto, ya que si bien sí había utilizado JavaScript anteriormente, nunca había trabajado con la biblioteca React. Aun así, el primer sprint fue para familiarizarme con el código de APIQuality.

Los primeros días del sprint los usé para ver como estaba distribuido el código dentro del proyecto e intentar comprender la funcionalidad de cada carpeta. Una vez hecha la primera toma de contacto con el código de la API, me asignaron dos tareas, una de validación de formularios y otra de integración con Google Analytics.

4.1.1. Validación de formularios

La tarea de validación de formularios fue la primera tarea que hice en el proyecto. Me ayudó con la familiarización del código del proyecto, ya que me obligó a buscar por todos las vistas que tenían formularios.

Esta tarea consistía en validar los campos de los formularios que ya estaban creados en nuestro proyecto, como el de iniciar sesión o el de importar una API. Dentro de la arquitectura del proyecto, para la realización de la tarea solo se modificaba la vista, en este caso el componente de React Formik[2]. Por lo que primero tuve que buscar en la documentación para comprender como funcionaba. Formik simplifica la tarea de trabajar con formularios al proporcionar una abstracción de alto nivel sobre los componentes de formularios en React. Al utilizar Formik,

puedes definir fácilmente formularios, manejar eventos y validaciones, rastrear valores de campo y gestionar el estado del formulario de manera eficiente.

Para la tarea que se me asigno de validar los campos de todos los formularios voy a poner de ejemplo el formulario de iniciar sesión, ya que es de los más sencillos al solo tener que validar el email y la contraseña.

Para validar los campos, primero hay que crear una constante que sea igual a Yup.object().shape(), esta biblioteca se usa para definir un esquema de validación para un objeto en JavaScript. En este caso queremos validar que el correo electrónico tenga forma de correo electrónico, es decir, que contenga un "@ɀ un "."después, y que la contraseña tenga un tamaño mínimo de 2 y un máximo de 50. También hay que validar que los campos estén rellenados, ya que los dos son obligatorios para hacer el login, para ello se añade la propiedad required.

```
const validate = Yup.object().shape({
  username: Yup.string()
    .email('errors.email')
    .matches(/^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$/, 'errors.email')
    .required('errors.required'),
  password: Yup.string().min(2, 'errors.min').max(50, 'errors.max').required('errors.required')
})
```

Figura 4.1: Código de la constante para la validación

La segunda parte de la tarea consiste en pasarle al componente formik el esquema de validación. Para ello, formik cuenta con validationSchema, un parámetro donde se le pasa el esquema de validación, línea 3 de la Figura 4.2. Una vez, el componente ya tiene el esquema de validación, a cada campo del formulario hay que pasarle el parámetro error, que hace que salga el error en caso de que no cumpla el esquema de validación, y el helperText, para que salga un mensaje de ayuda para rellenar el campo correctamente, líneas 18 y 19 de la Figura 4.2.

```
<Formik
  initialValues={initialValues}
  validationSchema={validate}
  onSubmit={({ username, password }) => authLogin(username, password)}>
  {(formik) => (
    <Form className="public-form" component={FormControl}>
      <Field
        variant="outlined"
        component={TextField}
        tabIndex={0}
        type="text"
        fullWidth
        name="username"
        id="username"
        value={innerEmail.email}
        label={t('public.labels.user')}
        onChange={formik.handleChange}
        error={Boolean(formik.touched.username && formik.errors.username)}
        helperText={formik.touched.username && t(`${formik.errors.username || ''}`)}
        onBlur={formik.handleBlur}
        disabled
      />
```

Figura 4.2: Código del componente formik

4.1.2. Google analytics

La segunda tarea de este sprint, consiste en vincular Google Analiytics para hacer seguimiento de la actividad de la web de APIQuality. Además, se quería realizar el seguimiento en los dos entornos, en development y en production.

Esta tarea tenía dos partes, la de crear una cuenta en Google Analiytics y configurarla para que distinguiera los dos entornos, y mi parte, que consistía en hacer el script para vincular la web con la cuenta y asegurarme de que el ID de medición se reemplazara por el correspondiente a cada entorno.

Para realizar la vinculación con Google Analiytics, se decidió hacer dos scripts situados en el header del proyecto. Uno para asignar el valor a la variable *ID-MEDICION* según el entorno, y otro para configurar los parámetros de la medición de Google Analytics.

```
<script>
  // Configurar la medición de Google Analytics
  window.dataLayer = window.dataLayer || []
  function gtag() {
    dataLayer.push(arguments)
  }
  gtag('js', new Date())
  gtag('config', ID_MEDICION)

</script>
```

Figura 4.3: Script de configuración

```
<script>
  // Asignar valor a la variable ID_MEDICION según el entorno
  var ID_MEDICION = '%NODE_ENV%' == 'production' ? 'G-31C85PK3S6' : 'G-KLHVHLK6JS'
  var script = document.createElement('script')
  script.type = 'text/javascript'
  script.src = `https://www.googletagmanager.com/gtag/js?id=${ID_MEDICION}`
  document.getElementsByTagName('head')[0].prepend(script)
</script>
```

Figura 4.4: Script para distinguir entornos

En la primera línea de la Figura 4.4 es donde se comprueba en que entorno se está ejecutando el proyecto y se asigna a la variable *ID-MEDICION* el ID correspondiente. Esto se hace con un condicional ternario, donde *NODE-ENV* es la variable global en la que se encuentra el nombre del entorno de ejecución. En el resto de líneas se establece la vinculación con Google Analiytics.

4.2. Sprint 2: Crear Selector con búsqueda

Para el segundo sprint aumentó un poco la dificultad y en vez de modificar los componentes que ya existían en el proyecto, se me encargó la tarea de crear un nuevo componente de React que actuase como un selector con la posibilidad de filtrar la lista de opciones por el nombre. Es decir, aparte de actuar como un selector normal, tenía que tener una barra de búsqueda que filtrase las opciones del desplegable. Por otro lado, este modal también tenía que tener la posibilidad de tener un botón al final del desplegable que permitiese crear nuevas opciones en la lista.

Este componente tenía que ser genérico y que se pudiese sustituir en todos los selectores que había en ese momento en la aplicación sin dar problemas. Para la explicación del código desarrollado, primero se expondrá el componente creado y luego se mostrará como se utiliza el modal con el ejemplo del selector de organizaciones.

```
<FormControl disabled={isDisabled} fullWidth>
  <InputLabel htmlFor={fieldName}>{label}</InputLabel>
  <Select
    labelId={fieldName}
    id={fieldName}
    name={fieldName}
    value={searchFilter !== '' ? '' : defaultValue}
    label={label}
    onBlur={formik ? formik.handleBlur : null}
    onChange={(e) => {
      setValue(e.target.value || '')
    }}>
    <ListSubheader>
      <TextField
        size="small"
        className="mt-2 mb-3"
        fullWidth
        autoFocus
        required
        placeholder={t('generic.search')}
        onKeyDown={(e) => e.stopPropagation()}
        value={searchFilter}
        onChange={onSearch}
        InputProps={{
          startAdornment: (
            <InputAdornment position="start">
              <SearchIcon />
            </InputAdornment>
          ),
          endAdornment: (
            <InputAdornment position="end">
              <IconButton onClick={clearFilter}>
                <CloseIcon />
              </IconButton>
            </InputAdornment>
          )
        }}
      />
    </ListSubheader>
```

Figura 4.5: Código selector con búsqueda

En la Figura 4.5 se puede ver el código utilizado para crear el selector con búsqueda, para crearlo se utilizan elementos de la biblioteca *Material-UI*. A continuación se van a explicar los elementos de este código:

- Con la etiqueta FromControl se encapsula todo dentro de un formulario
- La elemento select crea un selector típico de *Material-UI*, se les pasa el nombre, la etiqueta, el valor por defecto, en caso de que exista, y con la propiedad *OnChange* se la pasa la función que establece el valor seleccionado.
- En el elemento ListSubheades es donde se introduce la funcionalidad de la barra de búsqueda para que el usuario pueda introducir el nombre deseado. Con la propiedad *OnChange* se ejecuta la función *OnSearch* cada vez que el usuario escribe en la barra de búsqueda. Esta función se encarga de setear en *searchFilter*, que es un *useState*, el *string* introducido.

```
{items &&
  items.length > 0 &&
  items
    .filter(
      (item) =>
        (searchFilter !== '' &&
          itemName(item).toLowerCase().includes(searchFilter.toLowerCase())) ||
        searchFilter === ''
    )
    .map((item, index) => {
      return (
        <MenuItem key={`menuItem_${index}`} value={itemKey(item)}>
          {itemName(item)}
        </MenuItem>
      )
    })}
```

Figura 4.6: Código función de filtrado

En la Figura 4.6 se puede ver el código utilizado para filtrar la lista de ítems según lo que el usuario ha puesto en la barra de búsqueda. Como se ha explicado antes en *searchFilter* se encuentra el *string* que el usuario ha introducido.

```
const CreateButton = () => (
  <ListSubheader>
    <Button
      onClick={() => {
        onCreate(searchFilter)
        setNewValue(searchFilter)
      }}
      sx={{ textTransform: 'none' }}>
      <MenuItem className="text-info">{`+ ${t('generic.create')} ${label.replace(
        '*',
        ''
      )} ${searchFilter}`}</MenuItem>
    </Button>
  </ListSubheader>
)
```

Figura 4.7: Código del botón

La tarea también pedía que al final del listado apareciese un botón para crear un nuevo ítem en el listado. Como podemos ver en la Figura 4.7 este botón lanza una función que muestra un modal donde crear el nuevo ítem.

Finalmente, en la Figura 4.8 se puede ver el uso del componente SelectWithSearch en el selector de organizaciones. A este componente se le pasan distintos parámetros, pero los más importantes son, items, es la lista de ítems que aparecerán en el desplegable, onChangeSelect, este parámetro contiene la función que permite seleccionar un ítem del listado, y onCreate, a este parámetro se la pasa la función que queremos que lance el botón de crear de la Figura 4.7.

```
<FormControl fullWidth size="small" margin="dense">
  <SelectWithSearch
    label={t('sidebar.org')}
    key={new Date()}
    fieldName="select"
    items={user.organizations}
    defaultValue={organization_id || ''}
    onChangeSelect={(value) => handleOnChange(value)}
    onCreate={(value) => newOrganization(value)}
    loading={false}
    customEmpty={organization_id || ''}
    formik={formik}
    itemKey={(item) => item.id}
    itemName={(item) => item.name}
  />
</FormControl>
```

Figura 4.8: Código para usar el componente

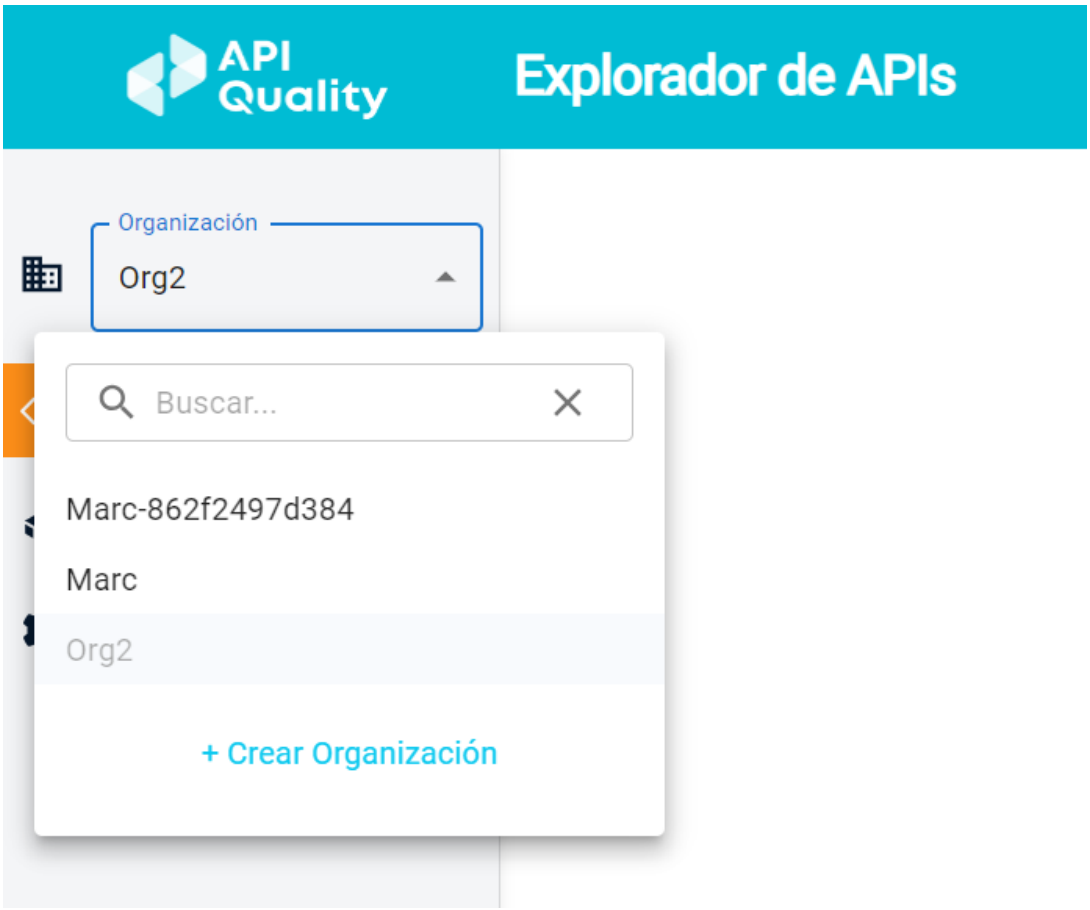

Figura 4.9: Resultado final de la implementación

4.3. Sprint 3: Selección de idiomas

Durante el tercer sprint sufrí una lesión de rodilla por la que me tuvieron que operar, estuve la primera semana del sprint de baja por lo que no pude hacer las tareas que se me habían asignado en el sprint planning. Aun así, la segunda semana del sprint ya pude volver al trabajo y me asignaron una tarea para que el usuario pueda seleccionar el idioma de la web.

Esta tarea se dividía en dos partes. Primero había que añadir un selector con los idiomas disponibles, por el momento solo estaban disponibles el español y el inglés. Este selector al principio tenía que estar ubicado en la cabecera, para que fuera visible en todas las vistas, pero finalmente se decidió ubicarlo en la vista de configuración del perfil de usuario.

La segunda parte de la tarea consistía en que a partir de ahora el back guardaría el idioma del usuario, para así desde el front, cada vez que el usuario iniciara sesión, consultar el idioma que el usuario había seleccionado al configurar su perfil y cargar las traducciones correctas. Así mismo, desde el front había que mandar al back el idioma que el usuario seleccionase en el nuevo selector para guardarlo en la base de datos.

La primera parte de la tarea fue fácil de implementar, ya que durante el primer y el segundo sprint estuve trabajando con los formularios y selectores de formik. Como en la vista de configuración de perfil de usuario ya existía un formulario para poder modificar los datos del usuario, simplemente añadí un componente Select al que se le pasaba un array con los idiomas disponibles para seleccionar.

```
<FormControl
  error={Boolean(formik.touched.language && formik.errors.language)}
  fullWidth>
  <InputLabel id="language">{t('modals.modal_profile.language')}</InputLabel>
  <Select
    labelId="language"
    id="language"
    name="language"
    value={formik.values.language}
    label={t('modals.modal_profile.language')}
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}>
    {languages.map(({ name, code }) => (
      <MenuItem key={code} value={code}>
        {name}
      </MenuItem>
    ))}
  </Select>
</FormControl>
```

Figura 4.10: Código del selecctor de idiomas

La segunda parte de la tarea fue más complicada, fue el primer contacto que tuve con el store del proyecto, si bien no tuve que crear ni modificar ningún endpoint ni reducer. Todos los archivos y funciones necesarias para manejar el store del usuario ya existían, al igual que el formulario. Por lo que solo tuve que añadir el lenguaje seleccionado en el selector, a los datos que se pasaban en el action que lanzaba el formulario. Era el back el que tenía que hacer las modificaciones necesarias para guardar los nuevos datos a la base de datos.

Una vez ya guardado el idioma, faltaba que la API cargase las traducciones correspondientes al idioma seleccionado, para ello el proyecto usaba i18next [3], una biblioteca JavaScript de

internacionalización (i18n) que proporciona una solución completa para gestionar las traducciones. Con esta biblioteca podía cambiar el idioma de la web, pero necesitaba que el idioma se cambiase cada vez que un usuario iniciara sesión o cuando el usuario cambiase el idioma en el selector. Para ello, utilicé por primera vez los Hooks de React, en este caso el useEffect. A este hook se le puede proporcionar un arreglo de dependencias y ejecutará el código que le proporcionemos cada vez que alguna de las dependencias cambie entre renderizaciones. En este caso nuestra dependencia era los datos del usuario, guardados en el state de la aplicación como user, esta variable cambiaba cada vez que iniciaba sesión un usuario o cambiaba sus datos en el formulario de configuración de perfil de usuarios.

```
useEffect(() => {
  if (user) i18next.changeLanguage(user.language)
}, [user])
```

Figura 4.11: Código para cambiar el idioma

En la Figura 4.11 se puede ver como se usa el useEffect, en el array de dependencias le pasamos los datos del usuario, si estos cambian o se cargan por primera vez, se ejecutará la línea 2 que cambia el lenguaje de la web utilizando la biblioteca i18next

4.4. Sprint 4: Crear un nuevo endpoint

En el cuarto sprint se decidió aportar una nueva funcionalidad a la aplicación con la posibilidad de añadir reglas a las guías de estilo. Estas reglas sirven para que el usuario cree su propia guía de estilos, así cuando el usuario escanee la definición de alguna de sus APIs podrá ver las reglas que cumple y las que no.

Para implementar esta nueva funcionalidad nos repartimos las distintas tareas y a mí me asignaron la creación de un endpoint para el manejo de las reglas de estilo. Me pedían que el endpoint fuese capaz de hacer una petición get para obtener todas las reglas de usuario, otra para obtener la información de una regla en concreto y por último una petición put para actualizar la información de una regla. Esta tarea también incluía guardar en el store las reglas, por lo que antes tuve que familiarizarme con la estrategia Redux [4].

Esta estrategia ya la había estudiado en el curso que hice de React al iniciar la estancia de prácticas. Pero esta estrategia se puede implementar de diferentes maneras, por lo que mis compañeros me explicaron como tenían la estrategia implementada en el proyecto y me enseñaron ejemplos de los endpoints ya implementados, que más tarde use de ejemplo para crear el nuevo.

La idea principal detrás de la estrategia Redux en React es tener un flujo de datos unidireccional y predecible. Los componentes solo pueden acceder al estado a través del store y no pueden modificarlo directamente. En cambio, los cambios se realizan a través de acciones que son procesadas por los reducers. Esto facilita el seguimiento y la depuración del estado de la aplicación, especialmente en aplicaciones más grandes o complejas.

En nuestro proyecto se implementa la estrategia mediante tres ficheros. A continuación se explica la implementación de la estrategia con el ejemplo del endpoint que cree para manejar las reglas de estilo:

- **rule.service.js**: Este archivo contiene funciones que interactúan con un servicio externo (a través de Axios [1]) para realizar operaciones relacionadas con las reglas de estilo. Cada función es una llamada a la API y devuelve los datos obtenidos del servicio. En la arquitectura del proyecto expuesta en la Figura 2.1 corresponde al middleware.
- **thunks.js**: En este archivo se definen las acciones asíncronas utilizando la función createAsyncThunk proporcionada por Redux Toolkit. Cada acción asincrónica se crea con una función que envuelve la llamada a una función del servicio correspondiente (definida en rule.service.js). Estas acciones manejan las solicitudes asíncronas y envían los resultados a los reducers. Este archivo corresponde con los dispatcher de la Figura 2.1
- **RulesSlice**: Es un slice creado utilizando createSlice de Redux Toolkit. Un slice define cómo se actualiza una porción específica del estado en respuesta a acciones específicas. El slice RulesSlice se encarga de manejar el estado relacionado con las reglas. En el campo extraReducers del slice, se definen los casos en los que el estado debe actualizarse en respuesta a acciones específicas. En la arquitectura del proyecto se corresponde con los reducers.

En resumen, las acciones asincrónicas se definen en thunks.js, las cuales envían solicitudes

al servicio y manejan las respuestas. El slice RulesSlice maneja cómo se actualiza el estado relacionado con las reglas en respuesta a acciones específicas. Finalmente, las acciones definidas en thunks.js se utilizan en los componentes de React para interactuar con el estado global almacenado en el store. Los componentes pueden despachar estas acciones utilizando el hook useDispatch y acceder al estado actualizado utilizando el hook useSelector.

Esta estructura garantiza que las solicitudes asincrónicas y las actualizaciones del estado se realicen de manera predecible y controlada en la aplicación, siguiendo el flujo unidireccional de Redux.

A continuación se explicará con detalle el código de cada uno de los ficheros creados para la tarea:

4.4.1. Thunks.js

```
export const getRules = createAsyncThunk(
  'rules/getRules',
  async ({ filters, profile_id }, { rejectWithValue }) => {
    try {
      const response = await getRuleService(profile_id, { filters })
      return response
    } catch (err) {
      if (!err.response) {
        throw err
      }
      return rejectWithValue(err.response.data)
    }
  }
)

export const getRuleById = createAsyncThunk(
  'rules/getRuleById',
  async (params, { rejectWithValue }) => {
    try {
      const response = await getRuleByIdService(params)
      return response
    } catch (err) {
      if (!err.response) {
        throw err
      }
      return rejectWithValue(err.response.data)
    }
  }
)

export const updateRule = createAsyncThunk(
  'rules/updateRules',
  async (request, { rejectWithValue }) => {
    try {
      const response = await updateRuleService(request)
      return response
    } catch (err) {
      if (!err.response) {
        throw err
      }
      return rejectWithValue(err.response.data)
    }
  }
)
```

Figura 4.12: Código de thunks.js

- **getRules**: Realiza una llamada a getRuleService para obtener una lista de reglas. Si tiene éxito, devuelve la respuesta. Si ocurre un error, lo maneja utilizando rejectWithValue para devolver el error al estado de Redux.
- **getRuleById**: Realiza una llamada a getRuleByIdService para obtener una regla específica por su ID. Si tiene éxito, devuelve la respuesta. Si ocurre un error, lo maneja de la misma manera que el anterior.
- **updateRule**: Realiza una llamada a updateRuleService para actualizar una regla. Si tiene éxito, devuelve la respuesta. Si ocurre un error lo maneja del mismo modo que los dos anteriores.
- **resetRule**: Crea una acción para reiniciar el estado relacionado con una regla.

4.4.2. rule.service.js

```
import { axiosPrivate } from '../api/axios'

const BASE_URL = 'profiles'

export const getRuleService = async (profile_id, params) => {
  params = params.filters
  const response = await axiosPrivate.get(`${BASE_URL}/${profile_id}/quality-rules`, {
    params
  })
  return response.data.data
}

export const getRuleByIdService = async ({ ruleId, profile_id }) => {
  const response = await axiosPrivate.get(`${BASE_URL}/${profile_id}/quality-rules/${ruleId}`)
  return response.data.data
}

export const getRuleByExternalIdService = async ({ ruleId, profile_id }) => {
  const response = await axiosPrivate.get(
    `${BASE_URL}/${profile_id}/quality-rules/external/${ruleId}`
  )
  return response.data.data
}

export const updateRuleService = async ({
  ruleId,
  activation,
  profile_id,
  severity,
  parameter
}) => {
  const response = await axiosPrivate.put(`${BASE_URL}/${profile_id}/quality-rules/${ruleId}`, {
    activation,
    severity,
    parameter
  })
  return response.data.data
}
```

Figura 4.13: Código de rule.service.js.js

- **getRuleService**: Realiza una solicitud GET a la API para obtener las reglas de calidad relacionadas con un perfil específico. Toma un ID de perfil y parámetros de filtro como entrada.

- **getRuleByIdService**: Realiza una solicitud GET a la API para obtener una regla específica por su ID. Toma un objeto con el ID de regla y el ID de perfil como entrada.
- **updateRuleService**: Realiza una solicitud PUT a la API para actualizar una regla específica. Toma un objeto con el ID de regla, la activación, la gravedad y el parámetro como entrada, que son las propiedades que se pueden modificar de las reglas.

4.4.3. ruleSlice.js

```
import { createSlice } from '@reduxjs/toolkit'
import { resetRule, updateRule, getRules, getRuleById, getRuleByExternalId } from './thunks'

const initialState = {
  loading: true,
  rules: [],
  rule: {},
  error: null,
  profileName: '',
  totalRules: null,
  page: 0,
  itemsPerPage: null
}

const RulesSlice = createSlice({
  name: 'rules',
  initialState,
  extraReducers: (builder) => {
    builder.addCase(getRules.pending, (state) => {
      state.loading = true
    })
    builder.addCase(getRules.fulfilled, (state, action) => {
      state.loading = false
      state.rules = action.payload.entities
      state.profileName = action.payload?.entities.name ? action.payload.entities.name : ''
      state.totalRules = action.payload?.meta.totalItems ? action.payload.meta.totalItems : null
      state.page = action.payload?.meta.currentPage ? action.payload.meta.currentPage - 1 : null
      state.itemsPerPage = action.payload?.meta.itemsPerPage
        ? action.payload.meta.itemsPerPage
        : null
    })
    builder.addCase(getRules.rejected, (state, action) => {
      state.loading = false
      state.rules = []
      state.error = action.payload?.message || action.error?.message || 'error'
    })
```

Figura 4.14: Código de ruleSlice.js

En la Figura 4.14 se puede ver como se utiliza la función createSlice del paquete @reduxjs/-toolkit para crear un slice de Redux llamado RulesSlice. Un slice define cómo se actualiza una porción específica del estado en respuesta a acciones específicas.

Se define el estado inicial del slice en el objeto initialState. Este estado inicial contiene varios campos como loading, rules, rule, error, profileName, totalRules, page, e itemsPerPage. Estos campos representan las diferentes propiedades de las reglas de estilo que se quieren guardar en el state.

A continuación, se utiliza createSlice para definir el slice RulesSlice. Se le proporciona un nombre 'rules' y el objeto initialState. Además, se define la sección extraReducers, que contiene los casos en los que el estado debe actualizarse en respuesta a acciones específicas.

Dentro de extraReducers, se utilizan los métodos addCase para agregar casos correspondientes a las acciones importadas desde el archivo './thunks', que son getRules, getRuleById

y updateRule. Cada caso define cómo se actualizan los campos del estado cuando se despacha una acción específica.

En la Figura 4.14 se pueden observar los casos de la función getRules, aunque estos casos también están en las otras dos funciones. Para el caso getRules.pending, se establece state.loading en true, indicando que la solicitud está en curso. Para el caso getRules.fulfilled, se establece state.loading en false y se actualizan los campos rules, profileName, totalRules, page, y itemsPerPage con los valores proporcionados en action.payload, es decir con los valores quenos ha pasado el back. Por último, en el caso getRules.rejected, seestablece el state.loading a false y establecen state.error con el mensaje de error proporcionado en action.payload o action.error.

Finalmente, se exporta el reducer del slice RulesSlice.reducer para ser utilizado en el store de Redux.

4.5. Sprint 5: Iniciar sesión externamente

Para el quinto sprint se me asignó una tarea que englobaba todo lo aprendido hasta ese momento. La tarea consistía en separar el login en dos pasos, para así dar la posibilidad al usuario de iniciar sesión mediante un método externo.

Hasta ese momento solo se podía iniciar sesión con un formulario en el que se introducía el email y la contraseña, si previamente esa cuenta se había registrado se accedía a la aplicación. Ahora se me pedía crear una primera vista en la que el usuario tenía que introducir el email. Una vez introducido el email había dos posibilidades, si el email no pertenecía a un método de login externo, el usuario era redirigido a la vista de inicio de sesión que ya teníamos, en la que solo tenía que introducir la contraseña. Si el email si pertenecía a un método externo de los que teníamos vinculados, el usuario era redirigido a esa url externa, en la que si iniciaba sesión con éxito, se cargaba la vista principal de nuestra aplicación.

Para esta tarea cree una nueva vista llamada ExternalSignIn y la coloque en el rutado de la API para que fuese la vista por defecto al entrar en la web. En esta vista, cree un formulario en el que se introducía el email, con sus respectivas validaciones para comprobar que se introduce un email válido.

```
<Formik
  initialValues={initialValues}
  validationSchema={validate}
  onSubmit={(email) => authLogin({ email })}>
  {(formik) => (
    <Form className="public-form" component={FormControl}>
      {!searchParams.get('code') && !searchParams.get('state') && (
        <TextField
          tabIndex={0}
          type="email"
          fullWidth
          name="email"
          label={t('public.labels.user')}
          onChange={formik.handleChange}
          error={Boolean(formik.touched.email && formik.errors.email)}
          helperText={formik.touched.email && t(`${formik.errors.email || ''}`)}
          onBlur={formik.handleBlur}
          required
        />
      )}
```

Figura 4.15: Código del formulario de ExternalSignIn.js

Una vez introducido el email, el formulario llama a la función authLogin a la que se le pasa el email introducido, como se puede ver en la línea 3 de la Figura 4.15. Esta función lanza otra función asíncrona externalEmail. Esta función es un endpoint que mediante un método post envía el email al back, este comprueba si el email pertenece a un método de login externo, si lo es devuelve al front la url a la que redirigir al usuario para iniciar sesión.

Volviendo a la función authLogin, de la línea 5 a la 8 de la Figura 4.16, se puede ver como una vez el back devuelve la respuesta, si existe la url se utiliza window.location.replace(r.url) para redirigir la página actual a la URL proporcionada en r.url. Si no existe, se establece el valor de innerEmail utilizando la función setInnerEmail(email), y se establece el valor de showModal en true, lo que muestra el modal de inicio de sesión que existía antes, donde el usuario puede iniciar sesión introduciendo solamente la contraseña, ya que el email ya se ha establecido con la función setInnerEmail.

```
const authLogin = async ({ email }) => {
  dispatch(externalEmail(email))
    .unwrap()
    .then((r) => {
      if (r.url) {
        r.logout && localStorage.setItem('exlog', r.logout)
        window.location.replace(r.url)
      } else {
        localStorage.clear()
        setInnerEmail(email)
        setShowModal(true)
      }
    })
    .catch((e) => {
      // eslint-disable-next-line no-console
      console.error(e)
    })
}
```

Figura 4.16: Código de la función authLogin

Por último, solo queda recoger los códigos de autorización devueltos por el login externo. Para ello cree un segundo endpoint, este endpoint llamado externalLogin manda al back mediante un método post el código y el state devueltos por el login externo. El back comprueba que los parámetros están autorizados y devuelve la respuesta al front, que lanza un error en caso de que no este autorizado o muestra la aplicación en caso contrario.

```
useEffect(() => {
  if (searchParams.get('code') && searchParams.get('state')) {
    dispatch(externalLogin({ code: searchParams.get('code'), state: searchParams.get('state') }))
  }
}, [searchParams, dispatch])
```

Figura 4.17: Código del useEffect

En la Figura 4.17 se puede ver como se utiliza un useEffect que se lanza cuando el login externo finaliza. Dentro del efecto, se verifica si existen los parámetros de búsqueda 'code' y 'state' utilizando el método searchParams.get(). Si existen los dos parámetros se lanza la función que llama al endpoint anteriormente mencionado para finalizar con el login.

4.6. Sprint 6: Estudio y modificación de estilos css

Como se ha expuesto en la planificación del proyecto en la sección 3.1, este sexto y último sprint es más corto, debido a que ya se cumplía las 300 horas de la estancia de prácticas. Por ello, en la planificación de sprint no se me asignaron tareas muy costosas, y se decidió que arreglase algunos errores relacionados con el estilo de la página web, ya que era la única área del front que todavía no había trabajado.

Estas tareas me las asignaba el equipo de QA anteriormente mencionado. Este equipo miraba en el Figma, la aplicación donde estaban los mockups de toda la API, y lo comparaba con los modelos creados por el equipo de desarrollo. La mayoría de las tareas que me asignaron eran sencillas, como cambiar el tamaño de la fuente de algún título o poner en negrita algún literal. Para realizar estas tareas, buscaba el elemento a cambiar. Si tenía una clase den los archivos css asignada, cambiaba las propiedades de esa clase para que fuese igual que el mockup del Figma. Si el elemento no tenía ninguna clase asignada, le asignaba una clase ya creada que cumpliera con los requisitos o creaba una nueva.

Otra de las tareas que me asignaron, relacionada con el estilo de la API, fue la personalización de los elementos de Material-UI. El proyecto usaba muchos elementos de esta biblioteca, por lo fácil que resultaba su implementación. El problema era que estos elementos tenían un estilo por defecto que no era el que queríamos en nuestra aplicación. Para la personalización de estos elementos, primero había que encontrar que clase de css usaban. Estas clases no aparecían en el código de nuestro proyecto, por lo que tuve que usar el inspector de elementos de Google Chrome para encontrarlas.

Una vez localizada la clase del elemento que queremos personalizar, el proceso es el mismo que con las clases creadas en las tareas anteriores. La única diferencia de estas clases es que hay que poner el parámetro '!important' en los atributos que queremos personalizar, de lo contrario, el elemento coge los atributos por defecto de Material-UI.

```
.Text-alert {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 400;
  font-size: 16px;
  color: #747474;
  flex: none;
  order: 1;
  align-self: stretch;
  flex-grow: 0;
}
```

Figura 4.18: Clase css para personalizar el texto de las notificaciones

```
.MuiToolbar-regular {
  min-height: 35px !important;
}

.MuiPaper-root > div > div > div {
  border-radius: 20px;
}

.MuiCheckbox-root {
  color: #00bcd4 !important;
}

.MuiRadio-root {
  color: #00bcd4 !important;
}
.MuiSwitch-thumb {
  background-color: #ffffff !important;
}
.Mui-checked + .MuiSwitch-track {
  background-color: #9ce7ff !important;
  opacity: 1 !important;
}

.MuiSwitch-track {
  opacity: 0.25 !important;
}
```

Figura 4.19: Clases css para personalizar los elementos de Material-UI

Capítulo 5

Conclusiones

Durante el desarrollo de este proyecto como desarrollador Front-End, se establecieron varias metas planificadas que han sido en su mayoría alcanzadas. Se logró la implementación exitosa de componentes en React que permiten a los usuarios interactuar de manera sencilla y sin complicaciones con la aplicación. Además, se estudiaron y aplicaron los Hooks de React, creando custom Hooks para mejorar la eficiencia y reutilización del código en la aplicación.

Asimismo, se estudió e implementó la metodología Redux para manejar el estado global de la aplicación, mejorando así su escalabilidad. Se garantizó una integración adecuada entre los componentes Front-End y Back-End de la aplicación mediante una comunicación constante y efectiva con el equipo de Back-End. También se llevó a cabo el testeo y validación de los componentes de la aplicación para garantizar su calidad y seguridad.

Aunque no se logró cumplir completamente la meta de definir los recursos de las APIs con OpenAPI, se trabajó con esta tecnología. A pesar de no haber realizado personalmente las definiciones en OpenAPI, se logró integrar correctamente la definición de la API de APIQuality con el desarrollo Front-End.

En cuanto a las dificultades encontradas, se destaca la lesión y posterior operación de rodilla que afectó temporalmente mi disponibilidad y modificó el cronograma planificado para el tercer sprint. Sin embargo, gracias a una pronta recuperación, pude reincorporarme rápidamente al trabajo y recuperar las horas perdidas, evitando mayores impactos en el proyecto.

5.1. Ámbito formativo

La estancia de prácticas me ha servido para aprender mucho e iniciarme como programador Front. Durante el grado ya había realizado algún proyecto de página web, pero las prácticas me han ayudado a profundizar en este aspecto. Partía desde cero, ya que nunca había usado ReactJS, pero con el período de formación donde hice el curso aprendí lo básico para ponerme a trabajar. Una vez ya empecé con los sprints, me asignaron tareas que iban aumentando progresivamente de dificultad. Esto hizo que al final trabajase con todo lo que había visto en el curso, desde lo más simple como los formularios a crear nuevos endpoint o un inicio de sesión externo. También quiero destacar la ayuda de los compañeros que siempre estaban dispuestos a explicarme las dudas o poner ejemplos similares a la tarea que tenía que hacer.

5.2. Ámbito profesional

La experiencia profesional en CloudAPPi ha sido muy buena. Al principio tenía un poco de miedo por si no me gustaba la forma de trabajar con la metodología ágil SCRUM. Pero desde el primer día me sentí cómodo con la metodología. Gracias a las reuniones diarias, siempre estaba en contacto con mis compañeros y, como he dicho anteriormente, me ayudaban siempre que lo necesitaba. Tanto los días que íbamos a la oficina, como los que trabajaba desde casa, había un gran ambiente de trabajo. Todos estos factores han hecho que mi primera experiencia profesional en el sector de la informática haya sido muy positiva.

5.3. Ámbito personal

En cuanto al ámbito personal, veo mi estancia en CloudAppi como una experiencia muy positiva. Siempre he tenido claro que el mundo de la informática me gustaba, pero tenía miedo, una vez terminado el grado, a descubrir que no me gustaba el trabajo. Por suerte, tras terminar el grado y la estancia de prácticas, tengo claro que quiero seguir trabajando y mejorando como desarrollador Front e incluso descubrir nuevas áreas dentro de la informática.

Bibliografía

[1] Documentación de axios. https://axios-http.com/es/docs/intro.

[2] Documentación de Formik. https://formik.org/docs/overview.

[3] Documentación de i18next. https://www.i18next.com/.

[4] Documentación de Redux. https://redux.js.org/.

[5] Git con toda la información sobre redux devtools. https://github.com/reduxjs/redux-devtools/tree/main/extension.

[6] Página web de figma. https://www.figma.com/.

[7] Página web de gitlab. https://about.gitlab.com/.

[8] Página web de la empresa de prácticas. https://cloudappi.net/.

[9] Página web de microcks. https://microcks.io/.

[10] Página web de React. https://es.react.dev/.

[11] Página web de redmine. https://www.redmine.org/.

[12] Página web de virtual studio code. https://code.visualstudio.com/.

[13] Página web del proyecto. https://apiquality.io/.

[14] Scrum: ¿qué es el product backlog? https://programacionymas.com/blog/scrum-product-backlog.

[15] Carlos de la Fuente García. Guía práctica para la publicación de datos abiertos usando apis. https://datos.gob.es/sites/default/files/doc/file/guia_publicacion_apis.pdf.

[16] Edward Amoroso, CEO, TAG Cyber. Factores que impulsan el crecimiento de la API en la industria. https://www.csirt-epn.edu.ec/servicios/guias-y-mejores-practicas/85-factores-crecimiento-api.

[17] Indeed. Consultor de salarios medios en España. https://es.indeed.com/career/programador-senior/salaries.

[18] John Hardy Developer. Arquitectura de redux. https://johnhardy.home.blog/2018/03/08/a-pretty-good-visualisation-of-the-redux-architecture/.

[19] Julio Roche, Specialist Director del área de System DevelopmentIntegration. Scrum: roles y responsabilidades. https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html.

[20] Rafael Berlanga. Ei1053 - tecnologias emergentes: Tema 4. arquitecturas orientadas a servicios. https://aulavirtual.uji.es/pluginfile.php/6441701/mod_resource/content/12/Tema%204.pdf.

[21] Rafael Berlanga. Ei1053 - tecnologias emergentes: Tema 5.servicios (api)rest(ful). https://aulavirtual.uji.es/pluginfile.php/6441702/mod_resource/content/12/Tema%205.pdf.