



*Facultad  
de  
Ciencias*

**Desarrollo de un piloto web para PYMES  
configurable por las propias PYMES**  
**Development of a web pilot for SMEs that can  
be configured by the SMEs themselves**

Trabajo de Fin de Grado  
para acceder al

**GRADO EN INGENIERÍA INFORMATICA**

Autor: Rubén García Macho

Director: Pablo Prieto Torralbo

Noviembre - 2024

## Agradecimientos

*Este proyecto significa el fin de una gran etapa, por lo que lo justo es agradecer a quienes me han llevado aquí.*

*A mi familia y amigos por acompañarme y apoyarme todos estos años, su paciencia, ánimos y su presencia siempre que ha sido necesitada.*

*A mis compañeros en la universidad, en especial a Javi y a Paula, que sin ellos el paso por esta etapa hubiese sido muy distinta y seguro que más sufrida.*

*A mis compañeros en Incentro, por adoptarme como uno más de ellos en los 3 meses que estuve, y enseñarme todo lo que he aprendido, desde conocimientos técnicos hasta que ambiente buscar en un lugar de trabajo en el futuro.*

*A todos los docentes por tener la paciencia de volver a explicar los conocimientos necesarios las veces que hicieran falta*

*Y como no, a Pablo, el tutor de este TFG, gracias por todo en este tiempo siendo mi tutor y por la ayuda prestada a pesar de no ser su área de conocimiento principal.*

*'Si he logrado ver más lejos ha sido porque he subido a hombros de gigantes'*  
*-Isaac Newton*

# Índice

<b>1. Introducción</b>	<b>1</b>
1.1. Antecedentes y objetivos . . . . .	1
1.2. Alcance . . . . .	1
1.3. Metodología y organización del proyecto . . . . .	2
<b>2. Tecnologías y herramientas utilizadas</b>	<b>3</b>
2.1. Tecnologías . . . . .	3
2.1.1. Tecnologías para la programación <i>Backend</i> . . . . .	3
2.1.2. Tecnologías para las pruebas unitarias del código . . . . .	3
2.1.3. Tecnologías para la programación <i>Frontend</i> y web . . . . .	3
2.1.4. Tecnologías para la comunicación <i>Backend-Frontend</i> . . . . .	4
2.1.5. Tecnologías para el almacenamiento de datos . . . . .	4
2.1.6. Tecnologías de trabajo colaborativo . . . . .	5
2.1.7. Tecnologías para los despliegues . . . . .	5
2.2. Herramientas . . . . .	6
2.2.1. Herramientas para el desarrollo del proyecto . . . . .	6
2.2.2. Herramientas para las pruebas unitarias del código . . . . .	6
2.2.3. Herramientas para la programación <i>Frontend</i> . . . . .	7
2.2.4. Herramientas para la comunicación con el gestor de contenidos . . . . .	7
2.2.5. Herramientas de documentación . . . . .	7
2.2.6. Herramientas de calidad . . . . .	7
<b>3. Especificación de requisitos</b>	<b>8</b>
3.1. Explicación de la aplicación . . . . .	8
3.1.1. Vista principal . . . . .	8
3.1.2. Vistas de servicios . . . . .	10
3.1.3. Vista de recursos . . . . .	11
3.1.4. Vistas de proyectos . . . . .	12
3.1.5. Vistas de empleo . . . . .	13
3.1.6. Vista de contacto . . . . .	14
3.2. Captura de requisitos . . . . .	14
3.3. Requisitos no funcionales . . . . .	15
3.4. Requisitos funcionales . . . . .	16
<b>4. Diseño del sistema</b>	<b>18</b>
4.1. Diseño arquitectónico . . . . .	18
4.2. Diseño detallado . . . . .	20
<b>5. Implementación</b>	<b>25</b>
5.1. Conexión con <i>Contentful</i> . . . . .	25
5.2. Utilización del gestor de contenidos . . . . .	26
5.3. Capa del controlador . . . . .	27
5.4. Métodos del <i>middleware</i> . . . . .	29

5.5. Componentes <i>Frontend</i> . . . . .	30
5.6. Capa de servicio del <i>Frontend</i> . . . . .	31
<b>6. Pruebas y despliegue</b>	<b>32</b>
6.1. Pruebas unitarias . . . . .	32
6.2. Pruebas de integración . . . . .	33
6.3. Pruebas de aceptación . . . . .	34
6.4. Pruebas de calidad . . . . .	35
6.5. Despliegue . . . . .	37
<b>7. Conclusiones</b>	<b>38</b>

## Resumen

El trabajo se ha desarrollado en una primera fase dentro de la empresa Incentro, siendo terminado fuera de ella. El objetivo del trabajo es desarrollar un proyecto piloto de página web que permita a una PYME publicitarse y ofrecer una serie de servicios, como registrar consultas. Esta página web, además, debe ser flexible para permitir su configuración desde la propia web por los propios trabajadores de la empresa.

El proyecto abordará el diseño de esta web desde cero, trabajando en tres capas principales: Frontend, Backend y Gestor de contenidos. El Frontend gestionará las vistas y secciones solicitadas por la empresa, mientras que el Backend y el gestor de contenidos se actualizarán para soportar las funcionalidades y almacenar tanto los datos ingresados por los usuarios como la configuración del Frontend.

Se utilizará la metodología Agile para el desarrollo, con reuniones periódicas con los gerentes de la empresa (Incentro) para verificar que la funcionalidad entregada cumple con sus expectativas iniciales.

En cuanto a las tecnologías utilizadas, se implementará una estructura de tres niveles, con el Frontend desarrollado en React y TypeScript, y el Backend también en TypeScript. Además, se hará uso de APIs para la interacción con el sistema de gestión de contenidos (CMS) y se empleará Git para el control de versiones, junto con herramientas como SonarQube y Jest para garantizar la calidad del código. Adicionalmente, el proyecto se desplegará en la nube, y tanto el Frontend como el Backend estarán gestionados a través de contenedores Docker, garantizando así un entorno aislado y reproducible para su despliegue y operación.

**Palabras clave:** React, Typescript, web, CMS, Contentful

## Abstract

The project was developed in a preliminary phase in the incentro Company, finishing while not being there. The goal of this project is to develop a pilot website for an SME to advertise and provide various services, such as registering enquires. Additionally, the website must be flexible to allow configuration directly from the website itself by the company's employees.

The project will address the design of this website from scratch, working across three main layers: Frontend, Backend, and Content Management System (CMS). The Frontend will manage the views and sections requested by the company, while the Backend and CMS will be updated to support functionalities and store both user-entered data and Frontend configurations.

The development will follow the Agile methodology, with regular meetings with company managers(Incetro) to ensure that the delivered functionality meets their initial expectations.

Regarding the technologies used, a three-tier structure will be implemented. Frontend developed in React and TypeScript, and Backend also in TypeScript. Additionally, APIs will be used to interact with the CMS, and Git will be employed for version control, along with tools such as SonarQube and Jest to ensure code quality. Furthermore, both the Frontend and Backend will be deployed in the Cloud and managed via Docker containers, ensuring an isolated and reproducible environment for deployment and operation.

**Keywords:** React, Typescript, web, CMS, Contentful

## **1. Introducción**

El proyecto del TFG se desarrolla en el Área especializada en *Content Management Systems* de la empresa Incentro. Se trata de una empresa de consultoría con sede en Holanda que ofrece servicios en los que proporciona ayuda en el desarrollo de aplicaciones software junto con mantenimiento de herramientas y aplicaciones en múltiples entornos empresariales.

En este proyecto participan dos trabajadores creando un equipo de trabajo: uno toma los roles de Jefe de proyecto, desarrollador *Frontend*, desarrollador *Backend* y desarrollador de pruebas y el otro toma el rol de desarrollador del sistema de gestión de contenidos. El autor de este trabajo es el primer trabajador, el otro papel lo ha desarrollado otro estudiante de prácticas.

En esta memoria del proyecto cubriremos los distintos aspectos en los que se ha desarrollado la aplicación, desde las tecnologías y herramientas utilizadas hasta las pruebas y despliegue, pasando por los requisitos, el diseño arquitectónico y cómo se ha implementado el proyecto entero.

En este apartado aportaremos las explicaciones antecedentes del proyecto: Alcance, objetivos y que metodologías se llevarán a cabo en el proyecto

### **1.1. Antecedentes y objetivos**

La aplicación la encarga una empresa emulada que trabaja en el sector de la ciberseguridad.

El objetivo es crear una página que sirva de forma publicitaria a la hora de ofertar los distintos servicios que ofrece, las áreas en las que trabaja, que productos ofrece junto con sus tarifas, su contacto para poder encontrar nuevos clientes y, de forma adicional, tendrá las funciones de portal de empleo por el que podrá captar nuevos posibles empleados.

### **1.2. Alcance**

La aplicación afectará principalmente a los departamentos de *marketing*, Recursos humanos y a la parte alta de la empresa, facilitando su trabajo a la hora de promocionarse haciendo que se pueda aumentar la cantidad de nuevos posibles clientes dentro de la industria, generar nuevas oportunidades de empleo y hacer que la empresa crezca. Con esta aplicación facilitaremos que se encuentre su empresa desde los motores de búsqueda más frecuentes y que se la pueda hallar con mayor facilidad. Facilitará los trabajos de promoción permitiendo abrirse al mercado al mostrar los servicios que se realizan en la empresa.

La aplicación debe estar disponible públicamente y tendrá localizaciones en las lenguas más extendidas en los principales mercados del cliente: angloparlante e hispano.

### 1.3. Metodología y organización del proyecto

Durante el desarrollo utilizamos una metodología ágil basada en **Scrum**[13] que utilizará *Sprints* de dos semanas. Éste tipo de metodología se la define como **Agile**[2] donde las entregas se conforman con tareas, facilitando un proceso de entregas continuo, cumpliendo todas las fases de los ciclos de desarrollo software y trabajando con el cliente para satisfacer sus necesidades y requisitos.

Esta metodología se ha elegido durante el desarrollo por la flexibilidad, adaptabilidad y colaboración continua entre el cliente y el equipo de desarrollo, pudiendo comprobar que los requisitos del cliente y proyecto desarrollado estén en todo momento en sincronía, haciendo que se desperdicie el menor tiempo posible y haya un menor ratio en cuanto al numero de ineficiencias encontradas.

Durante el proyecto, el jefe de proyecto tomará la responsabilidad de *Scrum Master*[13], asegurando que se sigan los principios de la metodología ágil, programando las reuniones necesarias, definiendo los *Sprints* y coordinando a los miembros del equipo.

Al principio del proyecto creamos un *backlog*[13] en el que definiremos las tareas necesarias para alcanzar *Minimally Viable Project*(MVP)[3]. Cada tarea se divide en categorías y se les asigna una demanda y unas dependencias entre tareas. Durante el desarrollo del proyecto se pueden modificar las tareas, añadiendo o borrando nuevas para adecuarse a las necesidades del cliente.

Al principio de cada *Sprint* definiremos las diferentes tareas correspondientes al *backlog* en una reunión extraordinaria denominada *Sprint Planning Meeting* en la que se estima el tiempo que llevarían las tareas, de esta forma también se pueden modificar las prioridades de las tareas en función del tiempo que puedan llevar.

Durante el *sprint* realizaremos reuniones de sincronización entre los miembros del equipo para conocer el estado de las tareas realizadas, en proceso y futuras y hacer reconfiguraciones de estrategia en cuanto a la planificación *sprint*. Dichas reuniones son las conocidas como *Daily* o *Weekly meetings*, que tienen la intencion de ser breves para no interrumpir el flujo de trabajo y ser meramente informativas en cuanto al desarrollo del proyecto y de las próximas tareas.

Utilizaremos una pizarra **Kanban** para realizar una asignación de tareas a los miembros del equipo y conocer su estado en el que se encuentra cada tarea. Estos estados son: *backlog*, análisis, desarrollo, pruebas y terminada. Además, esta pizarra se puede emplear en las reuniones de sincronización para facilitar el seguimiento del *Sprint*. En este proyecto se ha empleado la herramienta *Notion*, en la que se puede definir una pizarra Kanban y permite una personalización completa.

## 2. Tecnologías y herramientas utilizadas

Para el desarrollo se han requerido distintas tecnologías y herramientas que se explicarán a lo largo de las siguientes secciones junto con el papel que desarrollan en el proyecto.

Las decisiones de utilización de estas tecnologías y herramientas se tomaron al inicio del proyecto, siguiendo la formación y tecnologías proporcionadas por Incentro.

### 2.1. Tecnologías

#### 2.1.1. Tecnologías para la programación *Backend*

**TypeScript** se emplea como lenguaje de programación para el *backend*. Se ha realizado una programación lo más eficiente posible reduciendo el numero de llamadas a nuestro gestor de contenidos debido a la cantidad de datos que se tratan por llamada, el tiempo de respuesta del gestor de contenidos y los propios límites de la API de este.

Las llamadas del *Frontend* se estructuran como llamadas *Post* o *Get* junto con los parámetros de recepción y su resultado.

#### 2.1.2. Tecnologías para las pruebas unitarias del código

Para obtener la cobertura del código y hacer tests unitarios para probar el buen funcionamiento de los métodos se emplea **Jest**[12, 19], framework de Javascript y Typescript con el que se pueden automatizar la ejecución de los tests, obtener unas métricas de cobertura y dando *feedback* sobre donde ser mas exhaustivo en el testing, haciendo que se puedan automatizar las pruebas con facilidad.

#### 2.1.3. Tecnologías para la programación *Frontend* y web

- **TypeScript:** Es el lenguaje de programación que se utiliza en el *frontend*. Este lenguaje engloba *Javascript* de forma total y añade tipos estáticos y junto con el tipado añade objetos, lo utilizamos debido a los beneficios que nos proporciona por sus distintos *frameworks* y lo extendido que está este lenguaje, por lo que podremos tener más soporte sobre el funcionamiento del lenguaje y sus particularidades.
- **HTML:**(Lenguaje de Etiquetas de Hipertexto) se utiliza para la realización de las páginas web en base a una estructura lógica, para que permita compatibilidad entre distintos dispositivos y navegadores.
- **CSS:** Hojas de estilo para realizar la presentación de la página web HTML. Estos estilos incumben la totalidad de la página web, desde tipografías hasta la visibilidad de ciertos elementos y su disposición de forma visual. Utilizamos CSS para obtener una cohesión entre todas las vistas de la página web.

#### 2.1.4. Tecnologías para la comunicación *Backend-Frontend*

- **Axios**[4]: Es un cliente HTTP basado en solicitudes para comunicar de forma asíncrona *frontend* y *backend*. Para hacer la llamada se indica el tipo de ésta, junto con una URL que se encuentra en una variable de entorno. En caso de que la llamada no sea exitosa, se retorna un dato indicando que es un error, en caso de que la llamada sea exitosa se retorna en un *data* indicando los datos de la respuesta y algunos metadatos de ésta.
- **Express**[10]: Express es un *framework* que proporciona un conjunto robusto de características (como una capacidad de implementar un *middleware*) que usaremos durante el desarrollo de la aplicación. Proporciona una base muy sólida para definir las llamadas en *backend* y poder crear el código de forma más fácil de mantener.
- **JSON**: Es una notación de objetos de *Javascript*. Se utiliza mayoritariamente para las comunicaciones *frontend-backend*, tiene como característica principal lo ligera que es para almacenar y transportar datos. Además, nuestro gestor de contenidos proporciona todos sus datos en formato JSON.
- **Multer**: Multer es un *middleware* que trabaja con Express que nos sirve para poder enviar ficheros con mayor facilidad entre *frontend* y *backend*.

#### 2.1.5. Tecnologías para el almacenamiento de datos

Para almacenar los datos hemos usado un sistema gestor de contenidos(CMS) llamado **Contentful**[20]. Contentful nos proporciona un sitio donde poder crear y modificar contenido de nuestra aplicación, haciendo que la modificación de los datos sea inmediata y escalable. Al tener el almacenamiento de nuestros datos fuera del sistema facilita las labores de gestión, de modo que el puesto tradicional de un gestor de bases de datos puede ser desempeñado por un trabajador menos cualificado, ya que solo tendría que dedicarse a la creación y edición de contenidos.

Contentful nos proporciona una interacción simplificada en la que podemos definir y modificar contenidos mediante un sistema propuesto por ellos, que se ocupa de las partes técnicas de gestionar los contenidos, por ejemplo, si se quiere cambiar la imagen con la que se le da la bienvenida al usuario en la vista de inicio, se debe ir al contenido de la vista e ir al segmento de ésta donde se podrá ver la imagen y se podrá cambiar, sin tener que preocuparse por ningún aspecto técnico.

Hemos decidido usar esta tecnología debido a su naturaleza emergente en los últimos años y el auge que está teniendo en el mercado extranjero[9], a parte de por las ventajas que han sido comentadas anteriormente frente a una base de datos convencional.

### 2.1.6. Tecnologías de trabajo colaborativo

- **Git**<sup>1</sup>: Para proporcionar un trabajo colaborativo entre todos los miembros del equipo se tiene un repositorio en el que se proporciona el código con un control de versiones, capacidad de uso de ramas para el desarrollo de código reduciendo conflictos y mejorando la capacidad de trabajo distribuido.
- **Notion**: Hemos podido tener la capacidad de desarrollar todas las prácticas proporcionando una capacidad de dividir las tareas de forma consistente con los principios *Agile* gracias a Notion.

### 2.1.7. Tecnologías para los despliegues

- **Docker**: Docker es una tecnología para desplegar en contenedores los servicios en *backend* y *frontend*. Gracias a Docker podremos tratar las aplicaciones como una unidad, haciendo que se puedan resolver dependencias de ejecución en cualquier sistema, sin importar que software tenga instalado. Esto crea un entorno seguro y controlado con el que podemos eliminar problemas que pueden surgir mediante actualizaciones de software, por ejemplo.
- **Google Cloud**[1]: Mediante Google Cloud hemos conseguido los servicios para desplegar la aplicación, obteniendo una plataforma de procesamiento como *Google Cloud Run*. Hemos usado esta tecnología debido a la escalabilidad que proporciona y las integraciones que facilitan los despliegues.

---

<sup>1</sup><https://github.com/RubaGarcia/EsSEC>

## 2.2. Herramientas

### 2.2.1. Herramientas para el desarrollo del proyecto

**Visual Studio Code** es el entorno en el que se realiza la programación y pruebas en el desarrollo software. Se ha empleado por el soporte que tiene en cuanto a extensiones para ciertos tipos de trabajo, funcionalidades nativas del IDE junto con las añadidas de forma gratuita por la comunidad.

Para la gestión del proyecto utilizamos **Node.js**[15]. Al utilizar esta herramienta podremos gestionar la construcción del proyecto de forma mas óptima, la configuración del proyecto se gestiona mediante el fichero package.json[22], desde el package.json configuraremos todos los ajustes del proyecto, desde las dependencias a librerías que tiene hasta que comandos se utilizan para compilar éste.

```
1  {
2    "name": "EsSEC",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "build": "tsc -b && vite build",
9      "lint": "eslint . --ext ts,tsx --report-unused-disable-
10        directives --max-warnings 0",
11      "lint:fix": "eslint --fix \"src/**/*.{js,jsx,ts,tsx}\",
12      "format": "prettier --write \"src/**/*.{js,jsx,ts,tsx,css,md}\"
13        --config ./prettierrc.json"
14    },
15    "dependencies": {
16      "@contentful/content-source-maps": "^0.11.1",
17      "@headlessui/react": "^2.1.4",
18      "@heroicons/react": "^2.1.5",
19      // [...]
20    },
21    "devDependencies": {
22      "@types/react": "^18.3.3",
23      "@types/react-dom": "^18.3.0",
24      "@typescript-eslint/eslint-plugin": "^7.15.0",
25      "vite": "^5.3.4"
26    }
}
```

Listing 1: package.json resumido

### 2.2.2. Herramientas para las pruebas unitarias del código

Para poder proporcionar una cobertura completa de las pruebas unitarias de código, utilizamos el *framework* **Jest** para poder hacer mímicas las salidas de ciertas funciones de nuestro código para poder obtener la mayor casuística posible a nuestro trabajo, haciendo que podamos simular métodos y clases para obtener un resultado.

### 2.2.3. Herramientas para la programación *Frontend*

Para programar el *frontend* utilizaremos el *framework* **React**[26], que usaremos con *TypeScript* para estructurar la lógica y que mas tarde en la página web se representará con HTML y CSS. La principal función por la que se utiliza React es por la facilidad a la hora de usar funciones ya programadas de otras librerías y el uso de *componentes*, haciendo que haya el menor código repetido posible.[17]

### 2.2.4. Herramientas para la comunicación con el gestor de contenidos

La comunicación con nuestro gestor de contenidos se realizará mediante las propias API's de contentful. Se han utilizado estas porque nos proporcionan una mayor facilidad a la hora de confeccionar las llamadas. Todas las comunicaciones a las API's se hacen utilizando elementos JSON. Las API's necesarias han sido:

- **Content Delivery API**[7]: Se ha utilizado para obtener los contenidos de la página web, por ejemplo la estructura del *Header*, los elementos del *portfolio*...
- **Content Management API**[8]: Se ha utilizado para poder añadir nuevas entradas al gestor de contenidos, un ejemplo son las entradas que se crean de solicitantes en un trabajo cuando se rellena el formulario de aplicación.

### 2.2.5. Herramientas de documentación

Para obtener una buena documentación hemos utilizado **Swagger**[24]. Swagger es un sistema que proporciona la opción de tener una documentación completa de API's visible de forma clara en nuestro propio servicio, haciendo que el acceso a la documentación sea mucho mas fácil, desde la url de la API REST[21] de nuestro *backend*.

### 2.2.6. Herramientas de calidad

En los requisitos de desarrollo de nuestro proyecto estaba establecido un seguimiento de métricas de calidad en desarrollo, cosa que hizo que utilizásemos **SonarQube**[6]. Esta plataforma evalúa el código fuente y da unas métricas de código duplicado, código inalcanzable, cobertura de código, comentarios, complejidad ciclomática, complejidad cognitiva y posibles errores que produzcan un mal funcionamiento del código. Debido a unos problemas con el código tuvimos que obtener las métricas de cobertura mediante Jest.

*SonarQube* evalúa los posibles problemas, *bugs*, vulnerabilidades y *code smells* del código y genera una métrica de deuda técnica, que nos proporciona un tiempo estimado que se tendría que dedicar a solucionar todos los problemas encontrados por la herramienta.

### 3. Especificación de requisitos

En esta parte de la memoria realizaremos una explicación breve de la aplicación que pide el cliente, junto con la metodología seguida para realizar la captura de requisitos para obtener tanto los requisitos funcionales como no funcionales que van a marcar las guías a seguir en el proyecto.

#### 3.1. Explicación de la aplicación

La aplicación tiene 13 vistas principales, divididas en las páginas principal, servicios, recursos, proyectos, empleo y contacto. Cada una de ellas tiene sus propios requisitos y sub-páginas. Todas las páginas cuentan con en mismo *Header* y *Footer* en el que se mantienen las mismas funcionalidades de navegación en la aplicación y configuración de prioridades(Localización). Todas las vistas tienen versión disponible para cada tipo de dispositivo(Teléfono Móvil, *Tablet* y ordenador).

##### 3.1.1. Vista principal

La vista principal, reconocida por el usuario como página principal o *Landing Page* se divide en 3 secciones principales: *Hero*, Portfolio y la opinión de los anteriores clientes. Debido a que la función de esta página es que sea la primera que puede ver un posible cliente, su principal funcionalidad es la orientación de este en cuanto a saber si nuestra empresa cumple sus estándares.

- **Hero:** En nuestro Héroe o *Hero* obtenemos la primera información que puede leer un cliente a la hora de llegar a nuestra página, en éste tenemos de forma esquemática unos conceptos que pueden ser de utilidad para que el propio usuario identifique las principales funciones de la empresa.

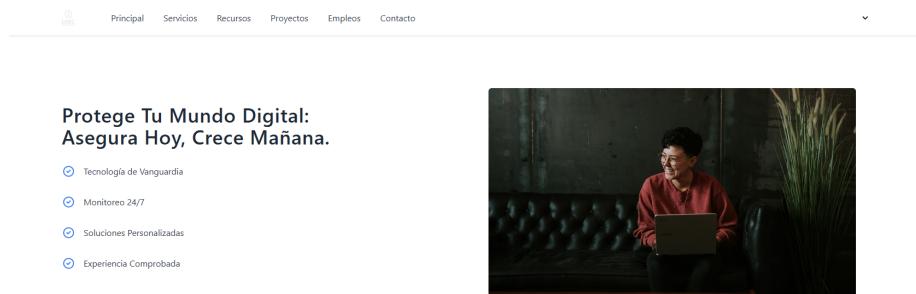


Figura 1: *Hero* y *Header* de la pagina principal

- **Portfolio:** Es un componente que nos proporciona una breve muestra de qué trabajo se ha realizado en anteriores ocasiones y en qué ámbitos trabaja la empresa.

#### Portfolio



Figura 2: Portfolio de la página de llegada

- **Opinión de clientes previos:** Tenemos un carrusel que le sirve al cliente de una posible orientación extra para conocer qué servicios se realizan.

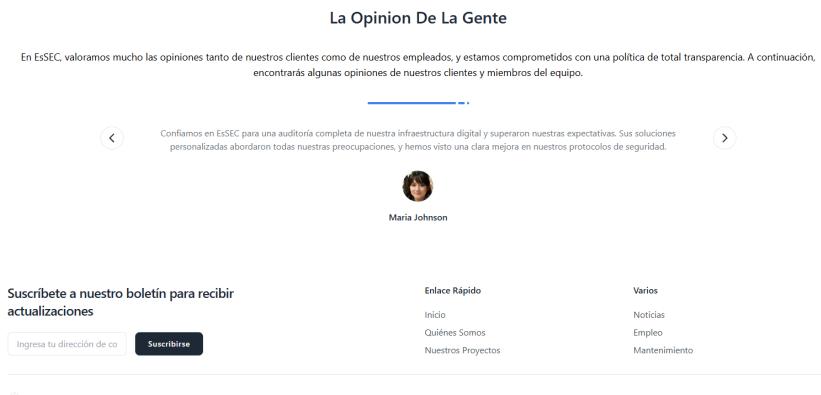


Figura 3: Carrusel con críticas de clientes previos

### 3.1.2. Vistas de servicios

En las vistas que comprenden a los servicios tenemos 6 vistas en total incluyendo la vista raíz que sirve para navegar hasta los servicios previos, que son Auditorías, Productos, Mantenimiento, Planes de mejora y el Kit digital.

En la vista que nos dirige a los servicios ofertados por la empresa tenemos un *layout* simple en el que se muestran los distintos servicios junto con un breve resumen de éstos con la principal función de orientar al usuario para obtener información del servicio que más deseé.

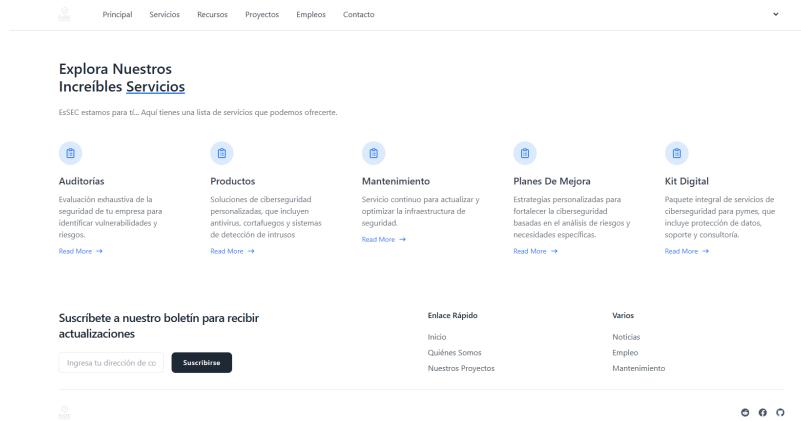


Figura 4: Página directorio de los servicios ofrecidos

En las páginas de cada uno de los servicios hay una estructura parecida: Un *Hero* para indicar de forma resumida el servicio ofrecido y unas críticas que dan una visión mas completa de como se realiza el servicio. En cada una de las vistas se hace de forma distinta pero por facilitar la lectura de esta memoria se ha preferido no dejar las imágenes. De forma alternativa, en alguna de las vistas, como puede ser la vista de los Productos, para dar mas información, se han añadido nuevos elementos.



Figura 5: Selección de productos ofertados

### 3.1.3. Vista de recursos

En los recursos tendremos una página directorio que nos indica que tipo de elementos podremos encontrar en nuestra página, los elementos se conforman de una imagen, un título y un texto que muestra un pequeño resumen de este, esta página ayudará mas a optimizar el SEO(Search Engine Optimization) haciendo que al tener más elementos multiseccionales y mas texto nuestra optimización del motor de búsqueda sea mejor.

The image shows a screenshot of a website's "Recursos" (Resources) page. At the top, there is a navigation bar with links for "Principal", "Servicios", "Recursos", "Proyectos", "Empleos", and "Contacto". Below the navigation, there is a heading "Recursos". The page displays five resource cards, each with an image, a title, a brief description, and a date.

- Detección de Amenazas en Tiempo Real**: Includes an image of a shield icon, a brief description about real-time threat detection, and the date "22/8/2024".
- Soluciones de Seguridad Personalizadas**: Includes an image of a network diagram, a brief description about personalized security solutions, and the date "21/8/2024".
- Global Tech Corp Asegura Su Red Contra Amenazas Zero-Day**: Includes an image of a modern building, a brief description about Global Tech Corp's zero-day protection, and the date "22/8/2024".
- Cortafuegos de Nueva Generación**: Includes an image of a digital binary code with locks, a brief description about advanced firewalls, and the date "21/8/2024".
- Sopporte**: Includes an image of a control room with multiple monitors, a brief description about support, and the date "21/8/2024".
- Cumplimiento y Gestión de Riesgos**: Includes an image of a padlock and a document, a brief description about compliance and risk management, and the date "21/8/2024".

Figura 6: Página de recursos

### 3.1.4. Vistas de proyectos

En la vista de proyectos buscamos tener una imagen mas completa de un portfolio, en el que se muestren los distintos áreas de trabajo de la empresa, tendremos un filtro en el que se muestran todas las distintas áreas de trabajo, en los elementos del portfolio tendremos una imagen con cada elemento, un título y un resumen, al hacer click en el título podremos acceder a la página en detalle del proyecto.

The screenshot shows a web-based project management system. At the top, there is a navigation bar with links for 'Principal', 'Servicios', 'Recursos', 'Proyectos', 'Empleos', and 'Contacto'. Below the navigation bar, the title 'Proyectos Recientes' is displayed. Underneath this, there are three tabs: 'All', 'Mejora de Seguridad', 'Seguridad de la Información', and 'Mejora de Protección de Datos'. The 'Mejora de Seguridad' tab is selected. The main content area displays two project cards. The first card, titled 'MEJORA DE SEGURIDAD' and 'Reestructuración De Ciberseguridad Para FinTech Inc', includes a brief description: 'Llevamos A Cabo Una Reestructuración Integral De Ciberseguridad Para FinTech Inc, Un Jugador Importante En El Sector De Tecnología Financiera. Nuestro Proyecto Incluyó La Mejora De Las Capacidades De Detección De Amenazas, La Realización De Evaluaciones De Vulnerabilidades Y La Implementación De Medidas Avanzadas De Protección De Datos. El Resultado Fue Una Postura De Seguridad Significativamente Mejorada Y Cumplimiento Con L...'. To the right of the text is a stylized graphic of a smartphone surrounded by glowing circles and lines representing data and security measures. The second card, titled 'SEGURIDAD DE LA INFORMACIÓN' and 'Global Tech Corp Asegura Su Red Contra Amenazas Zero-Day', includes a brief description: 'Global Tech Corp, Un Proveedor Líder De Soluciones Tecnológicas, Enfrentaba Crecientes Desafíos Por Amenazas Zero-Day Que Superaban Sus Medidas De Seguridad Tradicionales. Después De Asociarse Con Nosotros, Implementaron Nuestro Sistema De Detección De Amenazas En Tiempo Real Impulsado Por IA. En Solo Tres Meses, Bloquearon Con Éxito ...'. To the right of the text is a photograph of a modern building with glass windows and greenery in front.

Figura 7: Página General de Proyectos

En la página en detalle del proyecto, se obtienen mas datos sobre éste haciendo disponible información sobre que tecnologías se usaron, en que industria se hizo, que satisfacción tuvo el cliente, entre otras. También tendremos una descripción detallada de nuestro proyecto y como se solventó el problema propuesto por el cliente.

The screenshot shows a detailed view of a project. At the top, there is a navigation bar with links for 'Principal', 'Servicios', 'Recursos', 'Proyectos', 'Empleos', and 'Contacto'. Below the navigation bar, the title 'Reestructuración de Ciberseguridad para FinTech Inc' is displayed. To the left of the title, there are filters: 'Mejora de Seguridad', 'Año 2024', 'Industria FinTech', 'Tecnología Financiera', 'Servicio Reestructuración de Ciberseguridad', and 'Satisfacción Alta'. To the right of the title is a stylized graphic of a smartphone surrounded by glowing circles and lines representing data and security measures. Below the title, there is a detailed description of the project: 'Llevamos a cabo una reestructuración integral de ciberseguridad para FinTech Inc, un jugador importante en el sector de tecnología financiera. Nuestro proyecto incluyó la mejora de las capacidades de detección de amenazas, la realización de evaluaciones de vulnerabilidades y la implementación de medidas avanzadas de protección de datos. El resultado fue una postura de seguridad significativamente mejorada y cumplimiento con las regulaciones del sector.'

Figura 8: Página de proyecto

### 3.1.5. Vistas de empleo

En la vista de empleo se busca que sea una parte de la empresa en la que se oferten trabajos y se pueda postularse a estos, en la página principal se podrán ver todos los trabajos ofertados junto con una pequeña descripción. También la página contará con un *Hero* en el que hay una descripción de la propia vista para poder orientar al usuario de forma correcta en nuestra aplicación.

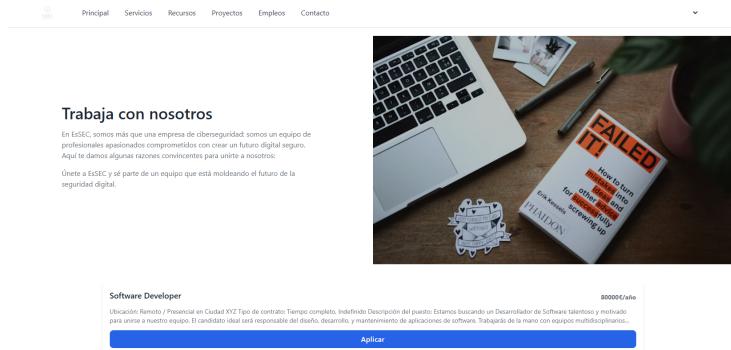


Figura 9: Página de oferta de empleos

En la página de cada trabajo en concreto, tendremos un Héroe junto con un título de la propuesta de empleo, un subtítulo en el que consta el sueldo de esta oferta y un cuerpo de oferta con la descripción del empleo, debajo tendremos un botón en el que, al interactuar con él, se abrirá un modal en el que se pedirán los datos para apuntarse a la oferta, cuando se envíen los datos se almacenarán en nuestro gestor de contenidos mostrando un mensaje en la esquina superior derecha de la pantalla.

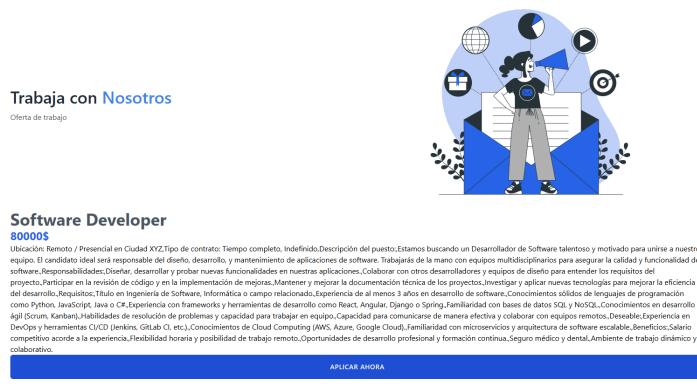


Figura 10: Página de un empleo concreto

### 3.1.6. Vista de contacto

En esta vista, podremos ver todos los empleados de la empresa y en la parte superior de ésta habrá unos botones que sirven de menú para filtrar qué empleados están trabajando en qué equipos. En las tarjetas de cada empleado podremos ver el nombre del empleado, su cargo y en caso de que haya un *link* a redes sociales(*GitHub*, *LinkedIn* y *Reddit*) se mostrará el logo de la red social con un vínculo a la página del empleado.

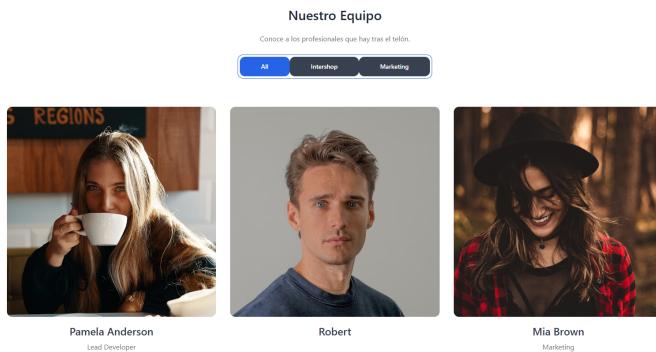


Figura 11: Página de contacto

### 3.2. Captura de requisitos

Los requisitos se han obtenido mediante reuniones con el cliente emulado por Incentro. En estas reuniones se ha obtenido la funcionalidad de la aplicación y los requisitos, descritos mas adelante, tanto los requisitos funcionales como los no funcionales.

Durante las reuniones con el cliente, se determina también el aspecto, los valores de estilo y las funcionalidades mínimas de cada vista, determinando completamente como funcionará la aplicación.

Las reuniones en las que se definen estos parámetros mediante la comunicación con el cliente se realizan de forma presencial y en Google Meet debido a la naturaleza de trabajo híbrido de la oficina.

### 3.3. Requisitos no funcionales

En los requisitos no funcionales describimos que características debe de haber en el sistema, incluyendo partes técnicas que tendremos que implementar en la aplicación. Los campos que suelen comprender los requisitos no funcionales suelen ser el rendimiento y tiempo de respuesta, la seguridad en nuestra aplicación y la calidad del código. [16]

Los requisitos se indicarán en la siguiente tabla, obteniendo unas especificaciones que busquen la eficiencia del proyecto en relación a cuantos accesos puede haber en la aplicación, de seguridad en cuanto al cifrado de las comunicaciones y la seguridad de los datos en nuestro sistema, y en el código comprendiendo la calidad del código y las plataformas desde las que se puede acceder a la aplicación.

Describiremos los requisitos por su código, en negrita la descripción corta y la descripción del requisito.

NFUN001 **Volumen de datos** - La aplicación deberá admitir un volumen de datos (incluyendo datos relacionados con las vistas) de más de 2000 elementos, permitiendo una escalabilidad a futuro.

NFUN002 **Seguridad** - La aplicación deberá tener una conexión cifrada y los datos de ésta deberán ser guardados de forma segura

NFUN003 **Visualización-Navegadores** - La aplicación deberá de poderse visualizar desde todos los navegadores principales (Microsoft Edge, Mozilla Firefox, Google Chrome y Safari).

NFUN004 **Visualización-Responsividad** - La aplicación tendrá que tener responsividad, cubriendo todos los ratios de pantalla: dando prioridad a los correspondientes a las pantallas de escritorio y de dispositivos móviles.

NFUN005 **Visualización-Temas** - La aplicación deberá de comprender temas oscuro y claro.

NFUN006 **Soporte de localización** - La aplicación deberá de dar soporte a los lenguajes inglés y español, traduciendo de forma completa la aplicación.

NFUN007 **Calidad del código** - El código deberá de tener una cobertura mínima del 70 % proporcionada por la herramienta Jest.

En sonarQube deberá de permitir un aprobado por los ajustes de serie del programa, que son:

- 0 Bugs
- 0 Vulnerabilidades
- Deuda Técnica < 3 %
- Código duplicado < 3 %

### 3.4. Requisitos funcionales

En los requisitos funcionales recogemos los servicios que deben de ser implementados en la aplicación, por lo que indican cómo debería de interactuar la aplicación y como se espera que responda. [16]

En esta lista donde recopilaremos los requisitos funcionales, tendremos el identificador de requisito (FUNXXX), a continuación tendremos en negrita la descripción corta del requisito, y a continuación, separado por un guion la descripción completa del requisito

**FUN001 Gestión de vistas** - La aplicación tendrá que tener 6 vistas y las asociadas a ellas:

- Home
- Services
- Services-Read More
- Resources
- Projects
- Project Detail
- Jobs
- Job Detail
- Contact

En cada una de las vistas se tendrá que representar el producto de forma clara.

**FUN002 Header** - aparecerá el logo de la empresa y un menú responsivo que nos llevará a las diferentes vistas y la opción de cambiar de idioma.

**FUN003 Footer** - En el pie de página debe de haber un formulario para suscribirse para tener nuevas noticias. Un menú con dos divisiones: Quick Link y Miscellaneous.

Deabajo estará el logotipo de la empresa y botones de redes sociales.

**FUN004 Home** - La pantalla principal debe de tener un resumen de la empresa, un portfolio y un carrusel con la opinión de otros usuarios.

**FUN005 Services** - En la vista de servicios deben encontrarse todas los áreas de trabajo de la empresa con una breve descripción y se podrá ampliar la información con el botón Read More.

**FUN006 Services-ReadMore** - Al ampliar la información la descripción aumenta y se deben mostrar opiniones de clientes sobre dicho servicio o productos ofrecidos en caso de tenerlos.

**FUN007 Resources** - Debe tener un apartado que indique las noticias con título, descripción y, si es relevante, la fecha en la que se publica.

FUN008 **Project** - Deberá haber un apartado con los proyectos recientes con un filtro que diferencie el ámbito, en la página debe haber una breve descripción, debe ampliar la información.

FUN009 **Project-Detail** - La información que se deberá ampliar seguirá el siguiente esquema:

- título del proyecto
- ámbito
- año de finalización
- industria
- servicio
- satisfacción
- descripción ampliada

FUN010 **Jobs** - Se busca que los usuarios tengan un portal de empleo con una descripción de la empresa con los empleos, que tendrán el sueldo bruto anual y una descripción que viene dado por el recruiter. La página debe tener opiniones de los trabajadores.

FUN011 **Jobs-Detail** - Cuando se amplia la información aparece el texto completo y un botón para postularse en el que se introduce nombre, apellidos, email y un drag and drop para introducir archivos como el currículum y poder enviar.

FUN012 **Contact** - En la pantalla de contacto se mostrarán los distintos miembros de la empresa con posibilidad de filtrarlos por el equipo del que forman parte. Se indicarán los miembros del equipo con nombre completo y el puesto que ocupan.

FUN013 **Notificaciones** - Las notificaciones en la página se mostrarán en la esquina superior derecha de la pantalla o en la parte superior en el centro de la pantalla, haciendo que se vean de forma clara.

## 4. Diseño del sistema

Tras obtener los requisitos funcionales y no funcionales de la aplicación, se realiza el diseño de nuestro sistema. En este definiremos dos partes: el diseño arquitectónico y el diseño en detalle.[11, 25]

- **Diseño Arquitectónico:** En este definiremos la estructura y organización de nuestro proyecto.
- **Diseño Detallado:** Diseño que contempla los diagramas en mayor detalle de la implementación.

### 4.1. Diseño arquitectónico

Temporalmente, el diseño arquitectónico es el primero en ser realizado. En él describimos la estructura a alto nivel, sus componentes principales y su relación.

Nuestra aplicación tiene 3 partes muy diferenciadas: el *Frontend*, el *Backend* y nuestro gestor de contenidos.

- **Frontend:** Parte web con un diseño basado en HTML con CSS y codificado en TypeScript, realizado íntegramente bajo el framework de React.
- **Backend:** Diseño que se encarga de comunicar el gestor de contenidos con el *frontend*, también encargándose de la lógica de negocio. Realizado en TypeScript.

En orden de realizar la comunicación *frontend-backend* utilizaremos una API REST que utiliza la comunicación HTTP con las primitivas Get, Post, Put y Delete, entre otras, aunque nosotros hayamos usado en la gran mayoría las primitivas Get y Post debido a la estructura de nuestra página. Para el formato en el flujo de datos usamos el formato JSON porque es un formato estándar y facilita el tratamiento de los datos.

Los servidores utilizados para el *frontend* y *backend* estarán alojados en servidores de Google Cloud bajo el servicio de Google Cloud Run. El servidor utilizado para ejecutar el gestor de contenidos lo aloja la propia empresa del gestor de contenidos, en este caso *contentful*.

Durante el desarrollo, utilizaremos diversos servidores alojados como contenedores en *localhost* para poder modificar el código sin tocar entornos de desarrollo que puede ser que produzcan errores con el resto de éste.

En la aplicación se pueden diferenciar 3 capas de forma clara:

- **Frontend(Capa de presentación):** Son las vistas que tiene el usuario, hay un proceso de mostrado y recopilación de información. Debe ser entendible y fácil de usar. Se le denomina *frontend* y es programado en TypeScript bajo el *framework* de React.
- **Backend(Capa de negocio):** En esta capa se aloja toda la lógica de negocio, es formada por el conjunto de programas que procesan la información que pasa por la capa de presentación y que se encontrará en la capa de datos, también procesa la información que se mostrará en la propia capa de negocio. Es denominado *backend* y lo realizamos en TypeScript.
- **Gestor de contenidos(Capa de datos):** Almacena los datos necesarios por la capa de presentación y recibe las peticiones de la capa de negocio para realizar lecturas, almacenar, editar o eliminar datos en función de los requerimientos del usuario. Lo llamamos gestor de contenidos y lo hemos realizado con *contentful*. Gracias a que almacenamos los datos de la capa de presentación, a parte de los datos típicos que almacenaría una base de datos, podemos tener un *frontend* mucho mas ligero de lo que suelen ser las páginas de la misma categoría, haciendo que las cargas sean mucho mas rápidas, el servidor tenga menos estrés y el usuario tenga una mejor experiencia a la hora de interactuar con nuestra página. En los requisitos no funcionales(**RNF001**) se especifica que la aplicación deberá de poder albergar más de 2000 datos, cosa que se consigue de forma exitosa debido a que según Contentful[18] podremos albergar hasta 5 veces más por entorno.

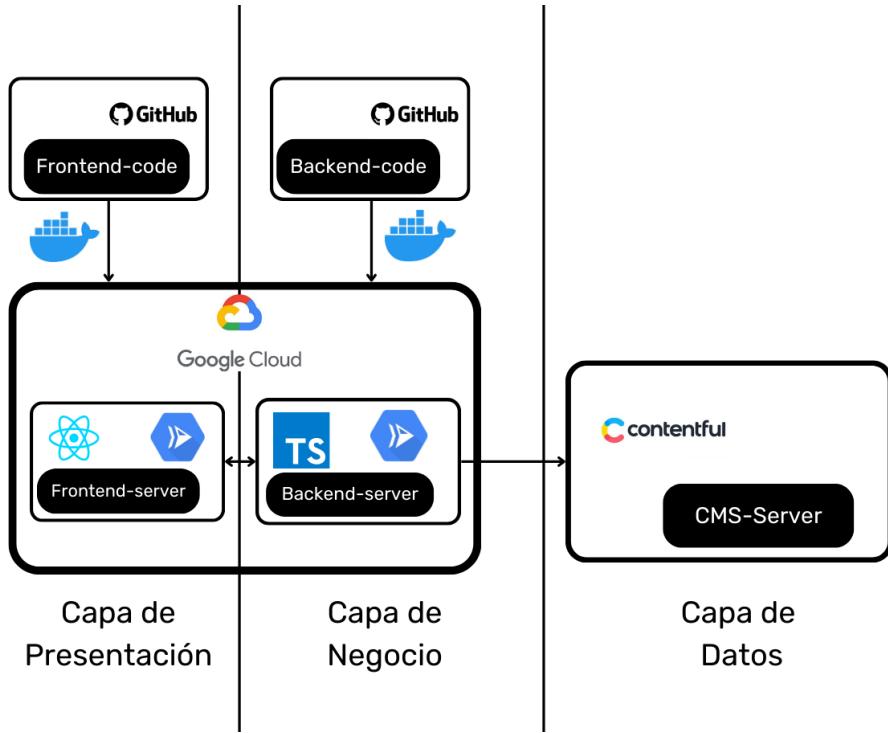


Figura 12: Diseño arquitectónico

#### 4.2. Diseño detallado

El diseño detallado se utiliza para describir los distintos componentes de la aplicación y su comportamiento cuando se conoce el diseño arquitectónico, de manera que se pueda proceder a la construcción de la aplicación.

En este diseño se realizan unos diagramas que utilizaremos para el desarrollo de la aplicación y la descripción de los recursos que disponemos para usar la aplicación una vez desarrollada.

El primer diagrama que realizamos es el diagrama de despliegue, que representa la arquitectura de ejecución de la aplicación. Muestra el *hardware* y *software* necesario.

En el diagrama de despliegue encontramos elementos de color morado, que representan el *hardware* y servidores, en gris encontraremos las instancias de google cloud, que se encuentran en mayor detalle en la siguiente figura, y en color azul estará el *software*, conformado por los códigos de Frontend y Backend, que se despliegan mediante contenedores en Docker en las instancias de Google Cloud Run. En los textos en las líneas tenemos los protocolos de comunicación entre equipos *hardware*.

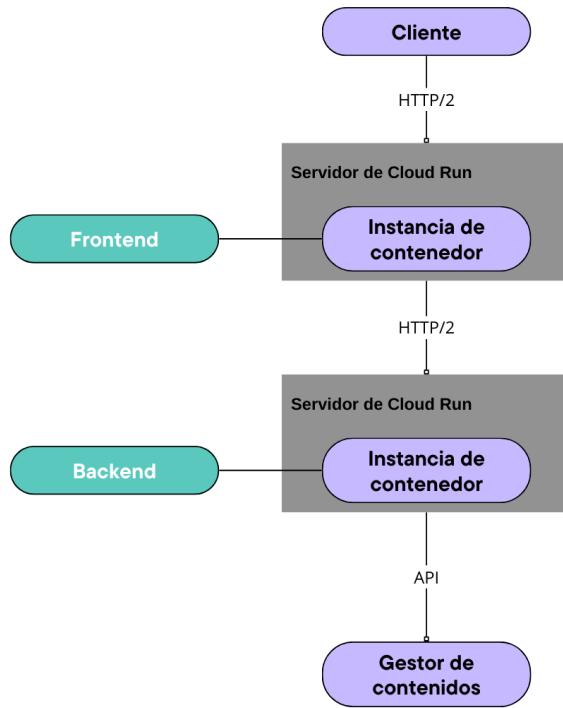


Figura 13: Diagrama de despliegue

En cuanto al *hardware*, tenemos 2 tipos de servidores distintos: los servidores de Google Cloud en los que se ejecutará el código de *frontend* y *backend* y un servidor de AWS (Amazon Web Services) en el que se aloja nuestro gestor de contenidos de mano de contentful.

Debido a la arquitectura de los servicios en los servidores de Google Cloud Run, la comunicación entre los servidores de *frontend* y *backend* se realiza mediante comunicaciones HTTP/2. En el *frontend* se definen *endpoints* bajo los que mostrar ciertas vistas con las que interactuará el usuario final, y en el *backend* se definirán los *endpoints* de forma que coincidan con la arquitectura de una API REST, de forma que la comunicación se facilite. En los servidores de Google Cloud se solventa los problemas de balanceo de carga mediante un sistema de ajuste de escala de servidores basado en solicitudes, replicando nuestro servicio las veces que haga falta en función de cómo de saturado esté nuestro servidor y balanceará la carga entre los distintos servicios que se clonen.

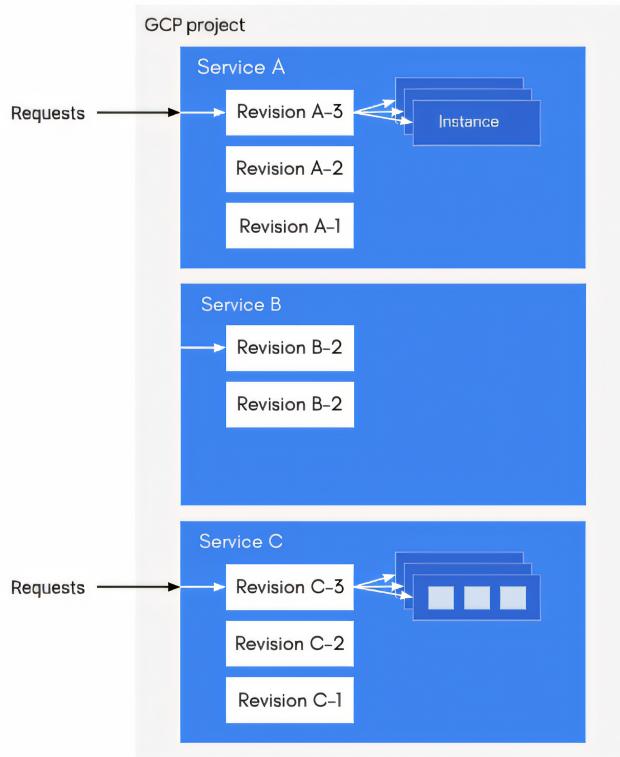


Figura 14: Arquitectura de un servidor de Google Cloud [14]

En cuanto a la seguridad en nuestros servicios, siguiendo con uno de los requisitos no funcionales(**RNF002**), en los servidores desplegados utilizando Google Cloud, tenemos una capa de seguridad obtenida mediante una TLS gestionada por el dueño de los servidores[1], en nuestro servicio *backend* tenemos implementada, aunque en el entorno de desarrollo desactivada, una autenticación mediante CORS que solo permite comunicarse con nuestro *backend* al servidor con la ip de nuestro servicio *frontend*. Los servidores y comunicación de contentful están también cifrados mediante comunicación HTTPS y los datos están encriptados bajo encriptación AES-256 [5].

Para mostrar el funcionamiento de nuestro modelo de datos hemos obtenido un modelado de nuestro gestor de contenidos en el que mostramos como interactúan todos los elementos de nuestro sistema gestor, el modelado funcionará como funciona un diagrama de clases.

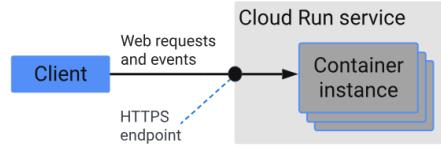


Figura 15: Sistema de comunicaciones de servicios en Google Cloud[1]

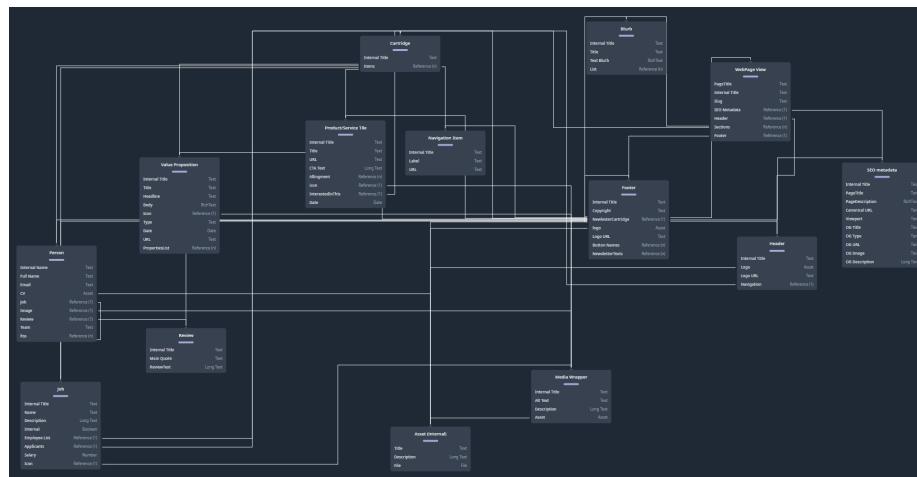


Figura 16: Modelo de datos del gestor de contenidos

En nuestra estructura de datos deberemos de diferenciar dos ámbitos de uso de la página: Los datos que se usen en *frontend* y los datos que almacenaría una base de datos tradicional.

En el ámbito de *frontend* tenemos almacenados los datos relacionados con las vistas, donde tendremos una tabla llamada *WebPage View*, que almacenará los datos de nuestra vista, que se relacionará con nuestro *Header*, *Footer* y tendrá una serie de secciones, que tienen una lista de elementos con *Value propositions*, que son el equivalente a un artículo, con su Titulo, Cabecera, cuerpo..., *Product Tiles*, que son referencias internas a otros sitios en la página web, como los *navigation Item*, que son el equivalente a un *Product tile* pero de forma más resumida. También podemos ver tablas como *SEO Metadata* que nos proporcionará una mayor cantidad de datos para promocionar la mejora de la optimización de motores de búsqueda

Los datos mas relacionados con los que almacenaría una base de datos tradicional son los relacionados con las Tablas de persona, debido a las peticiones en las que se guardan correos de contacto y datos personales cuando un posible empleado se postula a un empleo.

La creación del gestor de contenidos y la relación entre sus clases la definimos cuando obtenemos los requisitos del cliente, al conocer cómo funcionará nuestra aplicación podremos definir los elementos y su relación entre ellos, aunque en función de cómo avance el proyecto podremos realizar cambios en la estructura del proyecto sin mucho problema debido a la facilidad que nos proponen los gestores de contenidos como el usado.

## 5. Implementación

En esta parte de la memoria nos centraremos en los aspectos técnicos de la implementación de la aplicación, teniendo en cuenta la arquitectura en 3 capas, incluyendo la conexión con el Gestor de Contenidos y alguna clase auxiliar o patrón de diseño necesario.

Comentaremos aspectos necesarios para el funcionamiento de la aplicación, como la conexión del gestor de contenidos y su uso, controladores y *middleware*, los componentes definidos en el *frontend* y su capa de servicio.

Debido a las particularidades de este desarrollo en cuanto al gestor de contenidos, tendremos un *backend* mucho más ligero, como se ha comentado anteriormente, por lo que se podrá prescindir de las capas de servicio y definición de objetos, dado que vienen definidos bajo el gestor de contenidos y los objetos no necesitan una lógica exhaustiva.

### 5.1. Conexión con *Contentful*

Para conectar nuestro *backend* con nuestro gestor de contenidos utilizaremos las propias API's de *Contentful* debido a que nos proporcionan una mayor facilidad a la hora de enviar y recibir datos del gestor de contenidos.

*Contentful* nos proporciona una colección de APIs con distintas funcionalidades para que nuestro *backend* use los recursos solo cuando son necesitados. Las APIs usadas son las siguientes:

- **Content Delivery API:** Esta API se centra en traer contenido desde el gestor de contenido hasta nuestro *backend* para luego llevarlo al *frontend* en función de los requisitos de la aplicación. En caso de que los elementos sean especialmente pesados(por ejemplo imágenes y vídeos) *contentful* proporciona una url en la que se almacena el recurso para que no haya transferencias especialmente pesadas.
- **Content Management API:** En esta API nos centramos en enviar datos a nuestro gestor de contenidos para que se almacenen. Esta funcionalidad la usamos principalmente en las llamadas Post del *Backend*

Para utilizar las APIs de *contentful* deberemos de crear un cliente por cada API usada, de forma que podremos dividir el trabajo haciendo que funcione de forma asíncrona.

```
1 export const cda_token= process.env.CDA_TOKEN;
2 export const cma_token= process.env.CMA_TOKEN;
3 export const contentful_space= process.env.SPACE;
4 export const contentful_environment= process.env.ENVIRONMENT;
5
6 // Cliente para leer contenido (Content Delivery API)
7 export const deliveryClient = createDeliveryClient({
8   space: contentful_space,
9   environment: contentful_environment,
10  accessToken: cda_token,
11});
```

```

12 // Cliente para gestionar contenido (Content Management API)
13 export const managementClient: ClientAPI = createClient({
14   accessToken: cma_token,
15 });

```

Listing 2: Código en el que se realiza la conexión mediante clientes

Las constantes mostradas en el ejemplo anterior se obtienen mediante los *tokens* de seguridad que proporciona contentful, junto con variables que marcan los IDs del gestor de contenidos y de su entorno de desarrollo.

## 5.2. Utilización del gestor de contenidos

En el gestor de contenidos diferenciamos dos operaciones que se realizan en nuestra aplicación: Obtener y crear contenidos.

- **Obtener contenidos:** Para obtener elementos utilizaremos la propia función `getEntries` del cliente `deliveryClient` de la API de contentful, con los parámetros de configuración necesarios. En nuestro caso filtraremos en función del tipo de contenido (*WebPage View, Persona...*), añadiremos la profundidad a la que llegará el JSON que describirá el objeto con sus enlaces a otros contenidos y un *locale* debido a que contentful nos proporciona la capacidad de añadir localización y podemos elegir la localización que obtenemos a través de los parámetros de configuración.
- **Crear contenidos:** Para crear nuevos contenidos tendremos una serie de funciones que nos proporciona la API debido a la flexibilidad de estado de contenidos en contentful, dado que un contenido puede estar publicado, cambiado o como borrador. Nosotros dejaremos publicado directamente el contenido. Para crear un elemento usando la API tendremos que crear un Asset de forma asíncrona y cuando esté finalizado lo dejaremos publicado, dado que las transacciones no son atómicas. Teniendo en cuenta esto, utilizaremos `createEntry` siguiendo el esquema del tipo de contenido que se quiera subir(por ejemplo en nuestro caso persona) y para acabar la creación del contenido lo publicaremos con `publish`.

Cuando subimos un elemento en bruto, como puede ser un `pdf`, tendremos que crear primero la subida del fichero con `createUpload` almacenando el contenido del elemento, crearemos el fichero de forma acorde a nuestra configuración con `createAsset`, procesaremos el fichero para todas las localizaciones posibles con `processForAllLocales` y publicaremos el asset como anteriormente con `publish`

Dada la estructura del proyecto, hemos realizado unos métodos para automatizar las llamadas a nuestro gestor de contenidos, haciendo un método genérico para traer los elementos de nuestro gestor de contenidos al *frontend*. Para almacenar elementos en el gestor de contenidos deberemos de crear una serie de métodos específicos debido a la estructura concreta de cada tipo de contenido.

```

1  export async function getEntries(contentType: string,
2      internalTitle?: string, locale: string = "es" ) {
3
4      console.log(colors.bgRed(locale))
5
6      if (locale !== "en-US" && locale !== "es") {
7          locale = "en-US";
8      }
9      console.log("El local en la api es: " + locale)
10     try {
11
12         const entries = await client.getEntries({
13             content_type: contentType,
14             include: 10,
15             locale: locale
16         });
17
18         return internalTitle
19             ? entries.items.find(
20                 (entry) => entry.fields.internalTitle === internalTitle,
21                 )
22             : entries.items;
23     } catch (error) {
24         console.error("Error fetching entries:", error);
25         throw error;
26     }
27 }
```

Listing 3: Código para obtener elementos del gestor de contenido

### 5.3. Capa del controlador

La capa del controlador actuará como una interfaz de comunicación *backend-frontend*. Para el desarrollo hemos utilizado una interfaz llamada REST, que se utiliza para intercambiar datos a través de HTTP.

REST nos permite independizar cliente de servidor, haciendo que el *backend* y *frontend* se independicen en distintos servidores, de esta forma, podemos tener una mayor escalabilidad de nuestro servicio, aprovechando al máximo los servidores que nos proporciona Google Cloud.

Al haber utilizado Express, disponemos de un *framework* para estas llamadas que nos dará bastantes facilidades, como por ejemplo la capacidad de segmentar en capas las llamadas definidas por el controlador en un *middleware* y luego la lógica necesaria para el método.

Las llamadas que soporta una API REST son Get, Post, Put y Delete, por venir definidas por el protocolo HTTP, pero nuestra aplicación por las necesidades presentadas no ha requerido de más llamadas que Get y Post. Para definir una llamada en nuestro sistema deberemos de crear un *Router* de Express y éste tendrá como métodos heredados los propios de una API REST, posteriormente los métodos recibirán una serie de parámetros, siendo el primero la ruta a la que corresponde nuestra llamada, y luego una serie de métodos que formarán parte de nuestro *middleware* y el método que nos retornará una respuesta para la llamada.

```

1 const router = Router();
2
3 router.get("/", JobsController.getGeneral);
4
5 router.get(
6   "/:JobId",
7   param("JobId").notEmpty().withMessage("JobId is required"),
8   handleInputErrors,
9   (req, res, next) =>{
10     console.log("req.params", req.params);
11     console.log("req.query", req.query);
12     next();
13   },
14   JobsController.getJob,
15 );
16
17 router.post(
18   "/:JobId",
19   // Multer para manejar la subida de un
20   // unico archivo bajo el campo "file"
21   upload.single("files"),
22   // Controlador que maneja la
23   // creacion del asset y la persona
24   JobsController.obtainEmail
25 );
26
27 export default router;

```

Listing 4: controlador de las rutas de jobs

Para obtener la lista de métodos de controladores y su documentación podemos entrar a la ip del servidor del *backend* en la ruta /docs y tendremos la documentación completa de todas las llamadas, Esta documentación incluye: el tipo de llamada, su ruta, los parámetros necesarios de ésta, la descripción del método y de la respuesta, las posibles respuestas con sus códigos HTTP, y un campo donde se puede hacer la propia llamada para probarla con su respuesta. El desarrollo acabó con 17 llamadas definidas con sus correspondientes métodos.

Figura 17: Pantalla de swagger

#### 5.4. Métodos del *middleware*

Gracias a Express disponemos de un *middleware* que nos permite automatizar aspectos de las llamadas de la API REST para encapsular los elementos que gestionan el control de errores y los que llevan la lógica de la aplicación.

Nuestros principales métodos de *middleware* aseguran que la gestión de errores es correcta, haciendo que todos los datos introducidos o extraídos del gestor de contenido cumplen con las características necesarias.

Los métodos del *middleware* son métodos que toman la petición del método original, la respuesta(que todavía es vacía) y la función para llamar al siguiente método de la llamada del controlador.

```

1 export const handleInputErrors = (req: Request, res: Response, next
2   : NextFunction) => {
3   let errors = validationResult(req)
4   if (!errors.isEmpty()){
5     return res.status(400).json({errors: errors.array()})
6   }
7   next()
}

```

Listing 5: método de *middleware*

En este método vemos que la respuesta de la petición se convierte en 400 en caso de que alguno de los parámetros de la petición sea vacío.

## 5.5. Componentes *Frontend*

Estos componentes son una parte esencial React, dado que es una de las primeras características por las que se usa, porque nos proporciona la capacidad de reutilizar el máximo código posible y tener una mayor mantenibilidad en nuestro código.

Los componentes vienen conformados como funciones, tienen un código con su lógica principal (como por ejemplo en nuestro caso llamadas al *backend* para obtener los datos sobre la vista) y se retorna el código HTML en el que se puede injectar código de TypeScript junto con otros componentes de React.

En nuestro desarrollo cada vista está por componentes que se llaman entre ellos. Principalmente hay un *layout* compuesto por su *Header* y *Footer* (que son componentes) y las vistas en su interior.

```
1 interface FileDropProps {
2   onFilesSelected: (files: File[]) => void; // Callback para pasar
3   // los archivos al componente padre
4 }
5 const FileDrop: React.FC<FileDropProps> = ({ onFilesSelected }) =>
6   {
7     const [isDragging, setIsDragging] = useState<boolean>(false);
8
9     const handleDrop = (e: DragEvent<HTMLDivElement>) => {
10       e.preventDefault();
11       setIsDragging(false);
12
13       const droppedFiles = Array.from(e.dataTransfer.files);
14       onFilesSelected(droppedFiles); // Pasamos los archivos al padre
15     };
16
17     // mas logica del componente...
18
19     return (
20       <div
21         onDrop={handleDrop}
22         onDragOver={handleDragOver}
23         onDragLeave={handleDragLeave}
24         className={'w-full h-32 border-4 ${
25           isDragging ? 'border-blue-400' : 'border-gray-300',
26           border-dashed flex flex-col justify-center items-center
27           transition-colors duration-300'
28         }
29         >
30           <p className="text-gray-500">Arrastra y suelta los archivos
31           aqui</p>
32         </div>
33     );
34   };
35
36   export default FileDrop;
```

Listing 6: Ejemplo de componente para depositar ficheros

En total se han hecho 40 componentes de *frontend* para crear toda la parte visual de nuestra capa de presentación.

## 5.6. Capa de servicio del *Frontend*

En la capa de servicio nos ocupamos de la comunicación con el *backend* desde el *frontend*, de forma que definiremos las llamadas a la API REST que hemos comentado anteriormente.

Tomaremos la url del servidor *backend* a través de una variable de entorno para poder configurar con mayor facilidad los servidores en un futuro.

```
1  export async function getContact() {
2      const locale = localStorage.getItem('locale') || 'en-US';
3      // Default locale
4      const params = { locale };
5      // Parametro para la consulta GET
6      try {
7          const url = 'contact';
8          const {data} = await api.get(url, {params});
9          return data;
10     } catch (error) {
11         if (isAxiosError(error) && error.response) {
12             throw new Error(error.response.data.message);
13         }
14     }
15 }
```

Como se ve, es una función que realiza la petición a la url con los parámetros requeridos a través de axios, que llamará al *backend* usando la instancia get.

## 6. Pruebas y despliegue

En esta sección describimos que pruebas son las necesarias para comprobar que la aplicación funciona de forma correcta y siguiendo unas guías proporcionadas por la empresa para cubrir los requisitos de calidad.

Realizaremos las pruebas comprendiendo los tests unitarios, de integración, de aceptación y de calidad y comentaremos que metodología y herramientas hemos utilizado para desarrollar los tests.

### 6.1. Pruebas unitarias

El objetivo principal de las pruebas unitarias es asegurar el correcto funcionamiento de nuestro sistema desde el punto de vista de cada método individual.

Hemos realizado las pruebas unitarias con el *framework* Jest, que de por si nos proporciona herramientas para automatizar los tests unitarios. Jest también nos proporciona una buena herramienta para probar llamadas independientes, haciendo que podamos simular la salida de un método con un valor concreto nuestro.

Jest utilizará una estructura de ficheros en la que se definirán los tests a ejecutar. Las pruebas se encontrarán en las carpetas llamadas `__tests__` y tendrán la extensión `.tests.ts`. Jest se adaptará a la estructura de ficheros de nuestro proyecto excepto por esta particularidad que ayudará a localizar los tests en función al ámbito en el que se encuentran.

Los ámbitos de los tests los definiremos con `describe`, y luego cada prueba unitaria vendrá enunciada por `it`, a continuación tendremos un ejemplo del funcionamiento de las directivas anteriormente mencionadas.

```
1 const testGetEndpoint = (endpoint: string) => {
2   it(`deberia retornar una respuesta JSON cuando getEntries
      devuelve datos para ${endpoint}`, async () => {
3     (getEntries as jest.Mock).mockResolvedValueOnce({ data: "mocked
      data" });
4
5     const response = await request(app).get(endpoint);
6
7     expect(response.status).toBe(200);
8     expect(response.body).toBeDefined();
9   });
10
11  it(`deberia retornar un error 500 cuando getEntries lanza un
      error para ${endpoint}`, async () => {
12    (getEntries as jest.Mock).mockRejectedValueOnce(new Error("Test
      error"));
13
14    const response = await request(app).get(endpoint);
15
16    expect(response.status).toBe(500);
17    expect(response.body).toHaveProperty("error");
18  });
19};
20 //...
```

```

22 describe("Controllers", () => {
23   describe("HomeController", () => {
24     describe("GET /general", () => {
25       testGetEndpoint("/api/home/");
26     });
27   });
28
29   //Resto de tests...
30 }
31 }
```

Listing 7: Ejecución de test de endpoint del controlador de Home

En este ejemplo podemos observar como hemos creado funciones para automatizar los tests de *endpoints* de los controladores. En este caso, como hacemos *mock* del método `getEntries` podemos ver si la respuesta que nos da la llamada a `home` es correcta o no mediante el método `expect`.

Con estas pruebas se solventan errores relacionados con el desarrollo del código debido a problemas en llamadas al gestor de contenidos y respuestas a peticiones de la API REST, entre otros.

## 6.2. Pruebas de integración

En las pruebas de integración de un desarrollo software buscamos que los métodos examinados en las pruebas unitarias funcionen de forma correcta entre ellas. En esta fase tiene una especial relevancia el tener en cuenta que los parámetros se pasen de forma correcta entre secciones de la aplicación y que los flujos de acción de la propia aplicación son correctos.

El testing de integración comprenderá la correcta integración de la comunicación *backend*-gestor de contenidos y la integración *backend-frontend*.

Para hacer las pruebas en la comunicación entre el *backend* y el gestor de contenidos, debido a lo nueva que es la herramienta no hay librerías ni *frameworks* que nos sirvan para automatizar las pruebas de comunicación, por lo que tendremos que realizarlas a mano. Por suerte los casos de uso en el que la integración puede fallar son pocos y el gestor de contenidos nos proporciona unos entornos de desarrollo en los que podremos desarrollar las pruebas sin molestar al entorno de producción.

Las pruebas de integración *backend*-Gestor de contenidos, se realizarán haciendo llamadas a la API de contentful buscando todos los elementos vista(que son los que más datos enlazados tienen) y comprobamos que la configuración de los JSON coincide con la esperada. Por suerte, esta parte de las pruebas de integración se puede hacer con Jest en los métodos de obtención de datos de contentful (`getEntries`). Para probar que se almacenaban datos de forma correcta, se utilizaron unos datos concretos y se hicieron inserciones en el gestor de contenidos, comprobando su inserción.

Para la comunicación *frontend-backend* a través de la API REST hemos usando PostMan[23], que nos permite realizar peticiones a URL distintas, pudiendo almacenarlas en colecciones y automatizarlas. Además nos proporcionará la capacidad de poder cambiar los ajustes de las llamadas cuando queramos y cambiar el *body* de las peticiones, pudiendo almacenar variables de entorno para los distintos estados de desarrollo del proyecto.

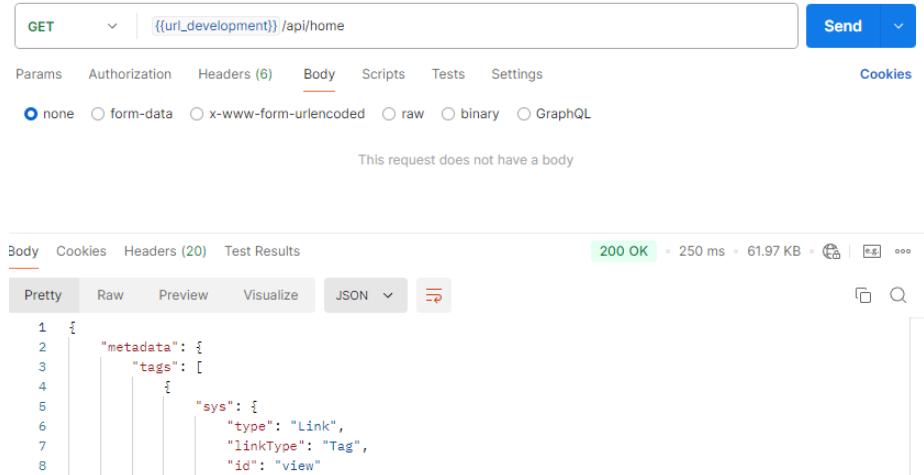


Figura 18: Pantalla de postman para la comunicación frontend-backend

En la imagen, se ve la comunicación con el entorno de producción, que es el que está siendo ejecutado mediante servidores de google cloud. Las pruebas de integración se hacen en local, por lo que debería de hacer la llamada al localhost con el puerto que tenga asignado docker en su dockerfile.

### 6.3. Pruebas de aceptación

En las pruebas de aceptación, como en todas las anteriores, disponemos de un entorno de desarrollo específico. Todas las funcionalidades nuevas deberán pasar por este entorno una vez se hayan hecho las pruebas unitarias y de integración para que se vuelva a comprobar otra vez de forma específica que todo funciona acorde a su funcionalidad deseada. En esta fase se hacen las pruebas en un entorno imitando al real, en lugar de hacer las pruebas en los servidores locales.

Por cada error encontrado, los desarrollos realizan una corrección y vuelven a hacer las pruebas de aceptación. La mayoría de errores que se encuentran en estos casos vienen relacionados con el volumen de datos de la aplicación. Como nuestra aplicación no tiene una gran diferencia entre local y en producción no se han encontrado errores.

## 6.4. Pruebas de calidad

En las pruebas de calidad usaremos SonarQube por las métricas que nos proporciona sobre el código tanto de funcionalidades nuevas como de código ya integrado.

Las métricas que nos da SonarQube nos permiten detectar los *code smells*, *bugs* y vulnerabilidades, que hacen que el código tenga menor calidad. Se tiene en cuenta estándares del lenguaje y código obsoleto, inalcanzable o duplicado.

También se proporcionan métricas de complejidad ciclomática y cognitiva de elementos que pueden incrementarla, tales como bucles anidados, sentencias condicionales, bloques try/catch o un uso exagerado de operadores lógicos, que pueden afectar a la comprensión del código y por defecto a la mantenibilidad de éste, por lo que se fuerza a que las sentencias sean más simples y el código siga unas reglas de complejidad y no se dificulte el desarrollo de nuevo código.

Teniendo en cuenta que tenemos unos márgenes de código duplicado o muerto y una cobertura de 70 % mínimo. Las métricas que usaremos vendrán dadas por 2 fuentes: Jest y SonarQube. Debido a un problema con Sonar, no proporcionaba cobertura de código, por lo que utilizaremos las métricas de cobertura de Jest y las métricas de código duplicado, inalcanzable, *bugs* y *code smells* vendrán de SonarQube. En la métrica de cobertura nos referimos a la cobertura de código en pruebas unitarias por líneas de código.

SonarQube nos retorna una serie de métricas que aproximan cuánto tiempo tardaríamos en arreglar los problemas de mantenibilidad del código. El objetivo es siempre reducir al máximo cuánto tiempo se tardaría en arreglar los problemas del código, dando más prioridad a los errores más importantes y dejando como menos prioritarios los que no afectan al rendimiento o mantenibilidad, como pueden ser los *code smells*.

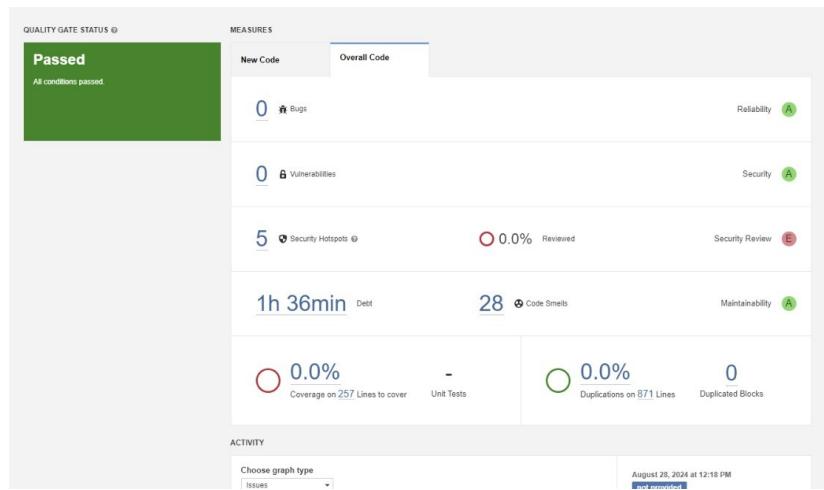


Figura 19: Imagen de métricas de sonar

Archivo	Statements	Branches	Functions	Lines
All files	<b>87.75</b>	<b>68.75</b>	<b>72.41</b>	<b>88.38</b>
src	100	100	100	100
server.ts	100	100	100	100
src/Controllers	78.4	58.33	71.42	79.03
ContactController.ts	100	100	100	100
HomeController.ts	100	100	100	100
JobsController.ts	54.28	60	50	55.88
LayoutController.ts	88.23	50	50	88.23
ProjectsController.ts	71.42	66.66	66.66	71.42
ResourcesController.ts	100	100	100	100
ServicesController.ts	90.62	100	85.71	90.62
src/config	100	100	100	100
cors.ts	100	100	100	100
multer.ts	100	100	100	100
swagger.config.ts	100	100	100	100
src/contentful	78.57	100	60	84.61
contentfulAPI.ts	78.57	100	60	84.61
src/middleware	100	100	100	100
src/routes	100	100	100	100

Cuadro 1: Tabla de coberturas de jest

Con estas pruebas podemos ver qué vulnerabilidades y bugs hay en nuestro código y que errores y problemas de seguridad pueden causar, también podremos ver que aspectos del código podemos mejorar de cara a hacer que la mantenibilidad del código sea mejor.

Cuando acaba el proyecto, tenemos un coverage del 90 % y SonarQube nos da el aprobado, sin tener bugs ni vulnerabilidades y porcentajes de código duplicado y deuda técnica menores que el 3 %, por lo que podemos dar por aprobado requisito no funcional 007 (**NFUN007**)

## 6.5. Despliegue

Para el despliegue utilizaremos Docker junto con Google Cloud. Como se ha explicado anteriormente, Google Cloud proporciona un servidor en el que se aloja un servicio en contenedores de Docker bajo el que nos comunicaremos por HTTP/2.

Para encapsular nuestro servicio en contenedores Docker, necesitaremos realizar los ficheros de configuración de docker, llamados *dockerfile*. En un dockerfile realizamos un scripting bajo el que vamos a definir qué flujo va a tener el contenedor para virtualizar nuestro servicio, así como qué versión de node utilizar, la instalación de paquetes antes de la ejecución, qué permisos tiene la aplicación y bajo qué puerto se comunicará con el exterior.

```
1 FROM node:20 #utilizamos node 20
2
3 WORKDIR /app
4
5 COPY package.json .
6
7 RUN npm install #instalamos dependencias
8
9 COPY . .
10
11 EXPOSE 4000 #exponemos puerto 4000
12
13 CMD ["npm", "run", "dev"] #ejecutamos el script de inicio del
    docker
```

Listing 8: Dockerfile usado en el *frontend*

Debido al uso de Google Cloud Run, podremos automatizar la subida de nuevo código, instalando una aplicación de este servicio que recibe una notificación cada vez que hay un nuevo *commit* hecho y vuelve a construir la aplicación.

Para realizar la elección de ejecutar los servidores como contenedores, realizamos una serie de pruebas de comunicación en local utilizando la herramienta *docker-compose* y al comprobar que las comunicaciones eran exitosas integramos los dos contenedores en sus respectivos entornos de Google cloud.

Google cloud nos proporciona una serie de métricas y una gran cantidad de ajustes de la aplicación que nos permiten tener un control constante sobre esta. Entre las métricas de los servicios tenemos el Recuento de solicitudes, la latitud de las solicitudes, cuantas instancias del contenedor han sido necesitadas, cuanta CPU y memoria requiere el contenedor, bytes enviados/recibidos y el máximo de solicitudes simultáneas. En cuanto a los valores configurables, destacan el registro de revisiones(pudiendo volver a una revisión previa del servicio), las instancias mínimas del servicio para desplegar un contenedor, y la región donde se almacena el servicio, entre otros.

## 7. Conclusiones

En este proyecto hemos realizado una aplicación desde 0, incluyendo su despliegue en la nube, que ha podido adquirir la dimensión suficiente para poderse considerar una aplicación lista para sacar al mercado y venderse como un producto completo. Durante el proyecto se ha completado toda la organización deseada en los objetivos, en los plazos objetivo y siguiendo todos los protocolos definidos en el proyecto.

Al encontrarse este proyecto bajo la supervisión de Incentro, deberíamos de quedarnos a la espera de saber si se continúa el proyecto, teniendo que proporcionar un servicio de mantenimiento, implementando funciones nuevas y arreglando posibles errores, por lo que este proyecto no puede darse por finalizado definitivamente.

Hemos podido aplicar los conocimientos adqueridos durante el grado en distintas áreas de este proyecto, como diseño de proyectos software, formalización de éstos y su gestión, diseño de interfaces mediante la interacción persona-computador, tráfico en redes y su seguridad y gestión de bases de datos, entre otros.

A nivel técnico se han aprendido conocimientos relacionados con TypeScript, React, Docker, qué es un gestor de contenidos y que impacto tienen en el mundo real, cómo se hace el testing en un entorno empresarial.

Habiendo desarrollado este proyecto con nuevas tecnologías hemos podido adquirir conocimientos sobre una serie de tecnologías como gestores de contenidos, entornos *cloud*, contenedores, lenguajes y *frameworks* de desarrollo web, a parte de las habilidades interpersonales que se obtienen al realizar un proyecto en un entorno nuevo como es el entorno empresarial, con un equipo de gente que no nos conocíamos antes de este proyecto. También se han aprendido como funcionan los entornos de producción y comprender como se gestiona un proyecto y que estrategias tomar durante la dirección de un trabajo.

## Referencias

- [1] *¿Qué es Cloud Run?* Google Cloud, 2024. URL: <https://cloud.google.com/run/docs/overview/what-is-cloud-run?hl=es-419> (visitado 11-11-2024).
- [2] Atlassian. *Manifiesto ágil para el desarrollo de software — Atlassian.* Atlassian, 2024. URL: <https://www.atlassian.com/es/agile manifesto> (visitado 11-11-2024).
- [3] Atlassian. *Minimum Viable Product (MVP): What is it Why it Matters.* Atlassian, 2023. URL: <https://www.atlassian.com/agile/product-management/minimum-viable-product> (visitado 11-11-2024).
- [4] Axios. Axios-http.com, 2020. URL: <https://axios-http.com/> (visitado 12-11-2024).
- [5] *Backup, security and hosting — FAQ.* Contentful, 2024. URL: <https://www.contentful.com/faq/backup-security-and-hosting/> (visitado 12-11-2024).
- [6] G. Ann Campbell y Patroklos P Papapetrou. *SonarQube in Action.* Manning Publications, nov. de 2013.
- [7] *Content Delivery API.* Contentful.com, 2015. URL: <https://www.contentful.com/developers/docs/references/content-delivery-api/> (visitado 11-11-2024).
- [8] *Content Management API.* Contentful.com, 2015. URL: <https://www.contentful.com/developers/docs/references/content-management-api/> (visitado 11-11-2024).
- [9] *Enterprise CMS platform.* Contentful.com, sep. de 2024. URL: <https://www.contentful.com/enterprise/> (visitado 12-11-2024).
- [10] *Express - Node.js web application framework.* Expressjs.com, 2024. URL: <https://expressjs.com/> (visitado 12-11-2024).
- [11] M Fowler. “Design - Who needs an architect?” En: *IEEE Software* 20 (sep. de 2003), págs. 11-13. DOI: 10.1109/ms.2003.1231144. URL: <https://ieeexplore.ieee.org/document/1231144> (visitado 12-11-2024).
- [12] Jest. Jestjs.io, 2017. URL: <https://jestjs.io/> (visitado 11-11-2024).
- [13] Dominik Maximini y Springerlink (Online Service. *The Scrum Culture : Introducing Agile Methods in Organizations.* Springer International Publishing, 2015.
- [14] *Modelo de recursos.* Google Cloud, 2024. URL: <https://cloud.google.com/run/docs/resource-model?hl=es-419> (visitado 11-11-2024).
- [15] *Node.js — About Node.js®.* Nodejs.org, 2024. URL: <https://nodejs.org/en/about> (visitado 11-11-2024).

- [16] Northware. *Requerimientos en el desarrollo de software y aplicaciones - Northware*. Northware, mayo de 2022. URL: <https://www.northware.mx/blog/requerimientos-en-el-desarrollo-de-software-y-aplicaciones/> (visitado 11-11-2024).
- [17] *React*. React.dev, 2015. URL: <https://react.dev/> (visitado 11-11-2024).
- [18] *Technical Limits*. Contentful.com, 2023. URL: <https://www.contentful.com/developers/docs/technical-limits/#free-plan> (visitado 12-11-2024).
- [19] *Testing React Apps · Jest*. Jestjs.io, sep. de 2023. URL: <https://jestjs.io/docs/tutorial-react> (visitado 11-11-2024).
- [20] Rouven Wessling. *Headless CMS explained in 1 minute*. Contentful, ago. de 2017. URL: <https://www.contentful.com/headless-cms/> (visitado 11-11-2024).
- [21] *What is a REST API?* Redhat.com, 2020. URL: <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (visitado 12-11-2024).
- [22] *What Is package.json?* Heynode.com, 2024. URL: <https://heynode.com/tutorial/what-packagejson/> (visitado 11-11-2024).
- [23] *What is Postman? Postman API Platform*. Postman API Platform, 2022. URL: <https://www.postman.com/product/what-is-postman/> (visitado 11-11-2024).
- [24] *What is Swagger*. Swagger Docs, 2024. URL: [https://swagger.io/docs/specification/v2\\_0/what-is-swagger/](https://swagger.io/docs/specification/v2_0/what-is-swagger/) (visitado 11-11-2024).
- [25] *WHAT IS YOUR DEFINITION OF SOFTWARE ARCHITECTURE?* URL: [https://insights.sei.cmu.edu/documents/2544/2010\\_010\\_001\\_513810.pdf](https://insights.sei.cmu.edu/documents/2544/2010_010_001_513810.pdf).
- [26] *Your First Component – React*. React.dev, 2024. URL: <https://react.dev/learn/your-first-component> (visitado 11-11-2024).