Diseño y desarrollo de un SaaS con arquitectura *serverless* para la gestión del CRUD de negocios online

David Olmos Cubells
Grado de Ingeniería Informática

Consultor: Gregorio Robles Martínez
Junio 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y desarrollo de un SaaS con arquitectura serverless para la gestión del CRUD de negocios online*
Nombre del autor:	*David Olmos Cubells*
Nombre del consultor:	*Gregorio Robles Martínez*
Fecha de entrega (mm/aaaa):	*06/2021*
Área del Trabajo Final:	*Desarrollo Web*
Titulación:	*Grado de Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

El propósito de este proyecto es el desarrollo de un SaaS (Software as a Service) basado en una arquitectura serverless en la nube, concretamente en AWS (Amazon Web Services), para gestionar los datos y las operaciones CRUD (Create, Read, Update and Delete) de un negocio online.

La idea surge de la necesidad de modernizar un conocido proyecto en este ámbito llamado Sonata, pero convirtiéndolo en una aplicación en la nube, deslocalizada y auto escalable.

Esta aplicación consta de tres partes. Un panel de administración, donde los usuarios pueden crear, borrar y modificar entidades de sus tablas de base de datos asociadas. La segunda parte, una pequeña página web donde ver reflejados, en tiempo real, los cambios persistidos en el panel de administración. Y, por último, una API estructurada en funciones Lambda serverless.

La aplicación está desarrollada principalmente con la librería NextJS y escrita en TypeScript. Sin embargo, las funciones de la API, es decir, el Backend, están escritas en JavaScript, concretamente en Node.js. Además, el conjunto de la aplicación está desplegado en AWS, siguiendo una arquitectura serverless.

Abstract (in English, 250 words or less):

The purpose of this project is the development of SaaS (Software as a Service) platform based on cloud serverless framework, specifically on AWS (Amazon Web Services) one, in order to manage the data and CRUD (Create, Read, Update and Delete) operations of an e-commerce.

The idea arises from the need to modernize a well-known project in this field called Sonata, but turning it into a cloud application, delocalized and auto scalable.

This application has three parts. An administration panel, where users can create, delete or modify entities from the tables of their associated databases. The second part, a tiny website where, the persisted changes in the administration panel, are reflected in real time. And finally, an API structured on serverless Lambda functions.

This application is mainly developed using the NextJS library and it is written in TypeScript. However, the API functions, that is, the Backend, are written in JavaScript, specifically Node.js. Also, the application itself is deployed on AWS, following the serverless framework.

Palabras clave (entre 4 y 8):

NextJS, TypeScript, Server-Side Rendering, AWS, Serverless, Sonata

Índice

1. Introducción 1
1.1 Contexto y justificación del Trabajo 1
1.2 Objetivos del Trabajo 2
1.3 Enfoque y método seguido 2
1.4 Planificación del Trabajo 4
1.5 Breve sumario de productos obtenidos 6
1.6 Breve descripción de los otros capítulos de la memoria 6
2. Análisis 8
2.1 Análisis del mercado 8
2.2 Análisis del Público objetivo 11
2.3 Análisis de las tecnologías 11
2.4 Análisis de riesgos y plan de contingencias 14
3. Requisitos 16
3.1 Requisitos funcionales 16
3.2 Requisitos no funcionales 16
3.3 Requisitos tecnológicos 16
3.4 Casos de uso 19
3.5 Alcance del proyecto y limitaciones 23
4. Diseño y Arquitectura 25
4.1 Arquitectura general de la aplicación 25
4.2 Arquitectura del modelo de datos 27
4.3 Diseño de la API REST 28
4.4 Diseño de los módulos 29
4.5 Diseño Gráfico 30
5. Implementación 35
5.1 Implementación Entidades Dinámicas 35
5.2 Implementación de la API REST 36
5.3 Implementación Frontend 38
6. Validación del Proyecto 44
6.1 Pruebas realizadas 44
6.2 Resultado Final 45
7. Conclusiones 48
7.1 Conclusiones obtenidas 48
7.2 Líneas de futuro 49
8. Glosario 50
9. Bibliografía 51

Lista de figuras

Ilustración 1: Tablero de Github Projects 3
Ilustración 2: Planificación temporal inicial 4
Ilustración 3: Planificación temporal final 5
Ilustración 4: WordPress pantalla principal CMS 8
Ilustración 5: PrestaShop pantalla principal CMS 9
Ilustración 6: Sonata Admin pantalla principal CRM 10
Ilustración 7: Análisis de tecnologías (logos) 12
Ilustración 8: Framework más querido por los desarrolladores. Encuesta 2019 13
Ilustración 9: Framework más descargado. Encuesta de 2019 13
Ilustración 10: Diagrama contenido estático 25
Ilustración 11: Diagrama Red de distribución de contenido 26
Ilustración 12: Diagrama Modelo API Serverless 26
Ilustración 13: DynamoDB - Estructura de los documentos de datos 27
Ilustración 14: Cognito - Estructura de los datos de usuario 28
Ilustración 15: Cognito - Estructura de los datos de usuario 2 28
Ilustración 16: Paleta principal de colores 30
Ilustración 17: Paleta secundaria de colores 30
Ilustración 18: Resumen tipografía 31
Ilustración 19: Muestra Iconos FontAwesome 31
Ilustración 20: Esquema estructura SPA 32
Ilustración 21: Vista Login - Baja Fidelidad 32
Ilustración 22: Vista Principal - Baja Fidelidad 33
Ilustración 23: Vista Entidades - Baja Fidelidad 33
Ilustración 24: Vista Login (Móvil) - Baja Fidelidad 33
Ilustración 25: Vista Principal con menú oculto / desplegado (Móvil) – Baja Fidelidad 34
Ilustración 26: Vista Entidades (Móvil) - Baja Fidelidad 34
Ilustración 27: Bucket S3 - Configuración de entidades dinámicas 35
Ilustración 28: Bucket S3 - Ejemplo de una entidad del archivo de configuración 36
Ilustración 29: API Gateway - Estructura de recursos 37
Ilustración 30: API Gateway - Dominios customizados 37
Ilustración 31: Funciones Lambda - Detalle 38
Ilustración 32: Funciones Lambda - Estructuración de Archivos 38
Ilustración 33: Ejemplo Componente Funcional 39
Ilustración 34: Ejemplo Interfaz 40
Ilustración 35: Ejemplo rutas 40
Ilustración 36: Ejemplo estilos 41
Ilustración 37: Estructura de carpetas del proyecto 41
Ilustración 38: Ejemplo Server-Side Props 42
Ilustración 39: Funciones Lambda - Pruebas 44
Ilustración 40: Funciones Lambda - Pruebas II 45
Ilustración 41: Funciones Lambda - Resultados Prueba 45
Ilustración 42: Publica - Vista Principal Dark / Light 45
Ilustración 43: Pública - Vista About 46

Ilustración 44: Publica - Vista 404 46
Ilustración 45: Publica - Vista Bloque Legal 46
Ilustración 46: CRM – Login 46
Ilustración 47: CRM - Vistas principal y perfil 47
Ilustración 48: CRM - Vistas Lista de Entidad / Lista Vacía 47
Ilustración 49: CRM - Vistas Creación Entidad y Entidad 47

1. Introducción

1.1 Contexto y justificación del Trabajo

Desarrollo páginas web para mis familiares y amigos cercanos, algo que compagino con mi trabajo principal y estudios, y una de las partes más costosas y tediosas de dicho desarrollo, siempre ha sido la de popular la información estática de la página web una vez esta está terminada. Esto es, por ejemplo, los elementos de navegación de la cabecera, las redes sociales, las imágenes o los textos legales.

Siempre he pensado que dicha información estaría mejor en la base de datos, donde fuera fácilmente modificable y de este modo, los cambios se podrían ver reflejados al momento. También he pensado siempre, que las personas a quienes les hago sus páginas web conocen mejor su producto como para escribir dichos textos mejor de lo que yo puedo hacerlo, aun siguiendo sus indicaciones.

Sin embargo, para una persona no programadora, esto es imposible, no sabe cómo conectarse a una base de datos y cambiar esa información. Por supuesto hay alternativas, sin ir más lejos, **WordPress** [1] permite hacer esto de una manera muy sencilla, también **PrestaShop** [2] o **Wix** [3].

Todas estas alternativas tienen otras cosas en común, te acotan a su marco de trabajo por definición y requieren de contratar servidores para poner los proyectos en funcionamiento. Estas tecnologías has estado muy bien durante mucho tiempo, pero los tiempos cambian, vivimos en la época de la computación en la nube y de las aplicaciones sin servidor.

Las aplicaciones sin servidor tienen muchas ventajas para los pequeños negocios, pueden tener una página web en producción sin apenas pagos por adelantado, simplemente afrontando los costes de los servicios consumidos, que inicialmente son muy bajos y nada más. Así es como desarrollo mis páginas web y no es algo que quiera abandonar por la falta de una plataforma desde la que realizar operaciones CRUD.

Fue entonces cuando recordé que, en mi puesto de trabajo anterior, usábamos **Sonata** [4]. Sonata es un proyecto de código abierto, fácilmente instalable en los proyectos que hacen uso de la tecnología **Symfony** [5], que precisamente permite realizar estas tareas con muy poca configuración.

A partir de esta premisa es donde nace la idea, crear un SaaS basado en la arquitectura sin servidor en la nube, que sea fácilmente accesible y configurable. De este modo, una vez configurado, cualquier cliente podría acceder y modificar esa información estática y verla reflejada al instante en su página web. Está claro que no se elimina la figura del programador de la ecuación, sigue haciendo falta para preparar dicha configuración, pero queda reducida notablemente.

1.2 Objetivos del Trabajo

Objetivo Principal

Desarrollo de un SaaS (Software as a Service) basado en una arquitectura sin servidor en la nube, concretamente en AWS (Amazon Web Services), para gestionar los datos y las operaciones CRUD (Create, Read, Update and Delete) de un negocio online. Esta aplicación está principalmente pensada para pequeños negocios y comercios que estén comenzando su digitalización.

Objetivos Secundarios

- Investigar las tecnologías propias de AWS.
- Diseñar las interfaces.
- Diseñar la página web donde reflejar las operaciones realizadas en el SaaS.
- Diseñar la estructura de la API.
- Diseñar la estructura de las funciones Lambda.
- Asegurar que la aplicación cumple con los estándares de usabilidad.
- Asegurar que los datos tratados cumplen con los estándares de seguridad y privacidad.
- Permitir la autenticación de usuarios en el sistema y configurar el identificador de su proyecto.
- Implementar un sistema que genere automáticamente las vistas de los diferentes tipos de entidades en función del archivo de configuración del usuario.
- Implementar un sistema que permita listar automáticamente las instancias de la entidad activa seleccionada por el usuario.
- Permitir que el usuario vea reflejados los cambios realizados, de manera instantánea, en su página web.

1.3 Enfoque y método seguido

Tal y como se ha explicado en la justificación del trabajo, no existe ninguna otra plataforma que permita cubrir las necesidades requeridas por completo. Esto se debe a que la mayoría de ellas están acotadas a sus marcos de trabajo y otras de ellas requieren de un marco de trabajo concreto para funcionar.

Por tanto, este proyecto se desarrollará como un nuevo producto que tratará de adaptar las funcionalidades de los productos ya existentes, pero de una manera mucho más agnóstica, deslocalizada de los clientes finales. De este modo, se podrán cubrir las necesidades de un mayor numero de personas a la vez que se cumplen los objetivos descritos en el apartado anterior. Esta manera de proceder parece la más acertada para esta casuística concreta. Aprovechar el hecho de que plataformas de este estilo ya existen, y simplemente conseguir replicar el funcionamiento, pero desacoplando el proceso de los marcos de trabajo que necesite o pueda necesitar el cliente, es decir un SaaS.

En cuanto al método seguido, actualmente existen dos grandes corrientes de metodología a la hora estructurar el trabajo a realizar, en el campo del desarrollo de software. Estas son: la metodología clásica o en cascada y la metodología ágil (***agile***), siendo esta última la elegida.

Como contexto, en la metodología en cascada, las tareas se realizan secuencialmente, una detrás de otra y una fase del desarrollo no puede comenzar si la anterior no ha concluido. En la metodología ágil, por el contrario, el desarrollo es iterativo, donde cada iteración acota un número reducido de tareas y de este modo se va conformando la aplicación en sí.

Ya hemos comentado que la metodología elegida ha sido la ágil, sin embargo, dentro de esta metodología existen muchas maneras de proceder. En este caso se ha elegido el método **Kanban** [6], que se caracteriza por distribuir las tareas, llamadas *cards*, en las diferentes columnas de un espacio llamado *tablero*, donde cada columna simboliza un estado dentro del proceso de desarrollo. Esto es muy útil, permite ver, a simple vista, el estado general de las tareas a realizar y su prioridad, detectar cuellos de botella, etc.

Para implementar este Kanban, se ha hecho uso de **Github Projects** [7], aprovechando que **Github** [8] es el repositorio en el que se encuentra el código del proyecto. Github Projects es una funcionalidad dentro del ecosistema de Github, que permite gestionar, de manera sencilla, proyectos de metodología ágil y concretamente con el método Kanban.

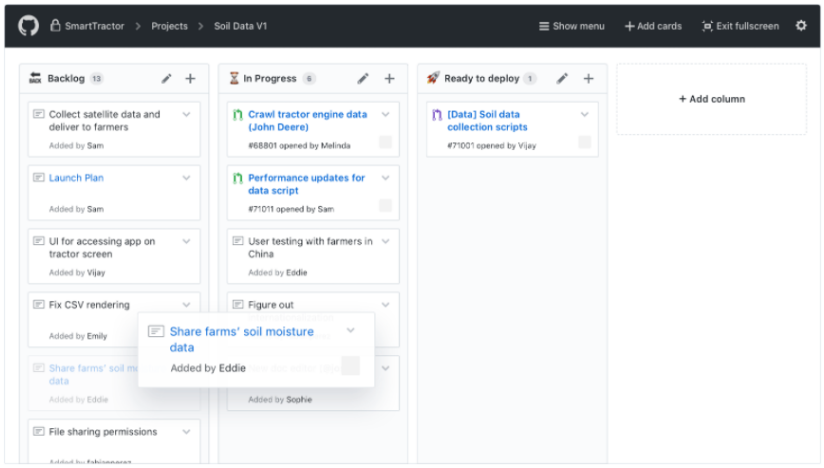

Ilustración 1: Tablero de Github Projects

El proyecto creado se ha compuesto de un total de 5 columnas:

- **Backlog:** Columna general de tareas, es el punto de partida de todas las tareas.

- **To Do:** Columna en la que se encuentran las tareas que deben ser realizadas en la iteración actual del proyecto.

- **In Progress:** Columna en la que se encuentran las tareas que se están realizando en el momento actual, las que están en progreso.

- **Done:** Columna en la que se encuentran las tareas, de la iteración actual, que ya han sido realizadas.

- **Deployed:** Columna en la que se encuentran todas las tareas, de cualquier iteración, que ya han sido terminadas y desplegadas en el entorno de producción.

1.4 Planificación del Trabajo

La planificación del proyecto ha sido dividida, a lo largo del tiempo, en cuatro grandes bloques. Estos bloques representan las cuatro pruebas de evaluación continua y han dividido el proyecto en cuatro fases clave: fase de documentación, fase de diseño, fase de desarrollo y fase de testeo.

Planificación temporal inicial

A continuación, se lista la propuesta de planificación temporal inicial en formato tabla. Para cada tarea se lista: su identificador, su nombre y el plazo de tiempo estimado que ocupará.

ID	Tarea	Inicio	Final	Total de días
1	**PEC 1**	**17/02/2021**	**01/03/2021**	**13**
1.1	Selección del tema	17/02/2021	27/02/2021	11
1.2	Redacción Memoria Inicial	27/02/2021	01/03/2021	3
2	**PEC 2**	**02/03/2021**	**01/04/2021**	**31**
2.1	Redacción Memoria PEC 2	02/03/2021	01/04/2021	31
2.2	Definir los límites del proyecto	02/03/2021	02/03/2021	1
2.3	Investigar las tecnologías propias de terceros: S3, DynamoDB, API Gateway, etc.	03/03/2021	03/03/2021	1
2.4	Diseño de la interfaz de usuario inicial	04/03/2021	11/04/2021	8
2.4.1	*Login*	04/03/2021	05/03/2021	2
2.4.2	Vista principal de administración	06/03/2021	08/03/2021	3
2.4.3	Contenido del apartado de las entidades seleccionadas	09/03/2021	11/04/2021	3
2.5	Diseño del webfolio, página web donde poner en práctica las operaciones realizadas en el SaaS	12/03/2021	15/04/2021	4
2.6	Diseño de la estructura de la API Rest que proveerá API Gateway.	16/03/2021	20/04/2021	5
2.7	Diseño de la jerarquía de las *cloud functions*	21/03/2021	24/04/2021	3
3	**PEC 3**	**05/04/2021**	**21/05/2021**	**47**
3.1	Redacción Memoria PEC 3	05/04/2021	21/05/2021	47
3.2	Preparar un entorno de desarrollo	05/04/2021	07/04/2021	3
3.3	Creación de un primer esqueleto para la plataforma	08/04/2021	17/04/2021	10
3.4	Crear un primer endpoint en API Gateway	18/04/2021	18/04/2021	1
3.5	Comprobar conexión remota con la API Rest desde el entorno de desarrollo	19/04/2021	20/04/2021	2
3.6	Crear endpoint de autentificación para gestión de usuarios	21/04/2021	22/04/2021	2
3.7	Implementar las funcionalidades de *Login* y *Logout* a la plataforma y persistir el token	23/04/2021	25/04/2021	3
3.8	Crear endpoints para las operaciones CRUD dinámicos	26/04/2021	29/04/2021	3
3.9	Implementar la búsqueda y creación de entidades del "cliente"	30/04/2021	02/05/2021	3
3.10	Implementar la gestión del CRUD de la entidad seleccionada	03/05/2021	09/05/2021	7
3.11	Aplicar estilos finales a la plataforma	10/05/2021	11/05/2021	2
3.12	Crear una primera versión de la página principal del *webfolio*	12/05/2021	16/05/2021	5
3.13	Crear primeras versiones de vistas y componentes modificables desde la plataforma	17/05/2021	18/05/2021	2
3.14	Implementar peticiones para recibir la información que puede ser modificada desde la plataforma	19/05/2021	20/05/2021	2
3.15	Aplicar estilos finales al *webfolio*	21/05/2021	21/05/2021	1
4	**PEC 4**	**22/05/2021**	**09/06/2021**	**19**
4.1	Redacción de la memoria final	22/05/2021	09/06/2021	19
4.2	Testeo final de la plataforma	22/05/2021	27/05/2021	6
4.3	Testeo final de la API	28/05/2021	02/06/2021	6
4.4	Testeo final del *webfolio*	02/06/2021	09/06/2021	7

Ilustración 2: Planificación temporal inicial

Planificación temporal final

A continuación, se lista la planificación temporal final en formato tabla. De nuevo, para cada tarea se lista: su identificador, su nombre y el plazo de tiempo estimado que ocupará.

ID	Nombre de la tarea	Fecha de Inicio	Fecha de Fin	Duración
1	**PEC 1**	**17/02/2021**	**01/03/2021**	**13**
1.1	Selección del tema	17/02/2021	27/02/2021	11
1.2	Redacción Memoria Inicial	27/02/2021	01/03/2021	3
2	**PEC 2**	**02/03/2021**	**01/04/2021**	**31**
2.1	Redacción Memoria PEC 2	02/03/2021	01/04/2021	31
2.2	Definir los límites del proyecto	02/03/2021	02/03/2021	1
2.3	Investigar las tecnologías propias de terceros: S3, DynamoDB, API Gateway, etc.	03/03/2021	03/03/2021	1
2.4	Diseño de la interfaz de usuario inicial	04/03/2021	11/04/2021	8
2.4.1	*Login*	04/03/2021	05/03/2021	2
2.4.2	Vista principal de administración	06/03/2021	08/03/2021	3
2.4.3	Contenido del apartado de las entidades seleccionadas	09/03/2021	11/04/2021	3
2.5	Diseño del webfolio, página web donde poner en práctica las operaciones realizadas en el SaaS	12/03/2021	15/04/2021	4
2.6	Diseño de la estructura de la API Rest que proveerá API Gateway.	16/03/2021	20/04/2021	5
2.7	Diseño de la jerarquía de las *cloud functions*	21/03/2021	24/04/2021	3
3	**PEC 3**	**05/04/2021**	**24/05/2021**	**50**
3.1	Redacción Memoria PEC 3	05/04/2021	24/05/2021	50
3.2	Preparar un entorno de desarrollo	05/04/2021	07/04/2021	3
3.3	Creación de un primer esqueleto para la plataforma	08/04/2021	17/04/2021	10
3.4	Crear un primer endpoint en API Gateway	18/04/2021	18/04/2021	1
3.5	Comprobar conexión remota con la API Rest desde el entorno de desarrollo	19/04/2021	20/04/2021	2
3.6	**Creación del *pool* de usuarios en Cognito y configuración de atributos**	21/04/2021	21/04/2021	1
3.7	**Integrar Librería Cognito en la plataforma e implementar sus funcionalidades**	22/04/2021	27/04/2021	6
3.7	**Integrar Librería Nookies en la plataforma e implementar sus funcionalidades**	28/04/2021	29/04/2021	2
3.8	Crear endpoints para las operaciones CRUD dinámicos	30/04/2021	02/05/2021	3
3.9	Implementar la búsqueda y creación de entidades del "cliente"	03/05/2021	05/05/2021	3
3.10	Implementar la gestión del CRUD de la entidad seleccionada	06/05/2021	12/05/2021	7
3.11	Aplicar estilos finales a la plataforma	13/05/2021	14/05/2021	2
3.12	Crear una primera versión de la página principal del *webfolio*	15/05/2021	19/05/2021	5
3.13	Crear primeras versiones de vistas y componentes modificables desde la plataforma	20/05/2021	21/05/2021	2
3.14	Implementar peticiones para recibir la información que puede ser modificada desde la plataforma	22/05/2021	23/05/2021	2
3.15	Aplicar estilos finales al *webfolio*	24/05/2021	24/05/2021	1
4	**PEC 4**	**25/05/2021**	**09/06/2021**	**16**
4.1	Redacción de la memoria final	25/05/2021	09/06/2021	16
4.2	Testeo final de la plataforma	25/05/2021	29/05/2021	5
4.3	Testeo final de la API	30/05/2021	02/06/2021	4
4.4	Testeo final del *webfolio*	03/06/2021	09/06/2021	6

Ilustración 3: Planificación temporal final

Análisis de los cambios en la planificación del proyecto

Al terminar la realización del proyecto, se ha comparado la planificación final temporal, con la indicada inicialmente.

Se puede observar que, la fecha final de entrega no ha sido modificada. Además, en los dos primeros grandes bloques, no hay ningún cambio, las fases de documentación y diseño concluyeron sin inconveniente.

Sin embargo, el tercer bloque y el cuarto han sufrido modificaciones temporales. Concretamente, el tercer bloque ha ocupado tres días más de los esperados, en detrimento de los días reservados al cuarto bloque. Esto ha ocurrido a causa de la integración de la librería de la tecnología **Cognito** [9], servicio de AWS empleado para la gestión de usuarios y que dispone de paquetes de **NPM** [10], para facilitar su integración en proyectos. Dicha integración ha sido muy complicada, el uso de TypeScript ha dificultado la tarea en exceso, pues los tipos de la librería no están correctamente desarrollados. Además, aparentemente, la librería empleada parece tener soporte de sesiones únicamente mediante cookies, las cuales debían ser tratadas desde el lado del servidor para que ningún usuario pueda manipularlas y hecho que propició la integración de otra librería extra, **Nookies** [11], para la gestión de estas.

Por último, y aún con los incidentes ocurridos, el cuarto bloque se ha terminado satisfactoriamente en el plazo esperado, que ha sido inferior al esperado. Como conclusión, podríamos considerar que el calendario se ha cubierto de manera exitosa, teniendo en cuenta los imprevistos que también se han podido gestionar con éxito.

1.5 Breve sumario de productos obtenidos

Como resultado final del trabajo se obtendrá:

- Una página web de administración privada, donde efectuar cambios a las entidades de las tablas descritas en el archivo de configuración del usuario. Desde esta web, el usuario también debe poder hacer cambios en su perfil

- Una página web pública donde ver reflejados dichos cambios.

El producto final consta de:

- El enlace a la página de administración privada: https://admin.olmosdev.com

- El enlace a la página pública: https://olmosdev.com

- El enlace al repositorio en el que se encuentra el código: https://github.com/davolcu/tfg-app

- El enlace al repositorio en el que se encuentra el código de la web pública: https://github.com/davolcu/tfg-web-app

Para acceder a la aplicación, es necesario un usuario y una contraseña, además, sin el correcto identificador de un proyecto tampoco podrían verse datos. Estos son:

- **Usuario:** *davidolmoscubells@gmail.com*

- **Contraseña:** Password1!

- **Identificador de Proyecto:** tfg

1.6 Breve descripción de los otros capítulos de la memoria

Los siguientes apartados de esta memoria son:

2. **Análisis:** Análisis de la situación actual en detalle de los proyectos similares, necesidades que cubren, público objetivo, etc. Este apartado

también incluye un análisis de las diferentes tecnologías necesarias para el desarrollo del proyecto.

3. **Requisitos:** Requisitos del proyecto de manera detallada, tecnologías empleadas y alcance del proyecto. En este apartado también se incluyen los casos de uso del proyecto.

4. **Diseño y Arquitectura:** Diseño del producto. Por un lado, se muestra la estructura de la aplicación. Por otro lado, también se incluye un apartado con el diseño gráfico y las interfaces del proyecto de manera muy superficial.

5. **Implementación:** Fase más técnica del proyecto. Se detalla toda la parte de creación y configuración del entorno de trabajo, base de datos, etc.

6. **Validación del proyecto:** Se exponen las pruebas realizadas. El objetivo de este apartado es dar fe del funcionamiento del proyecto y dejar constancia de sus diferentes características.

7. **Conclusiones:** Describe las conclusiones y las líneas de futuro del proyecto.

2. Análisis

2.1 Análisis del mercado

Haciendo un análisis de las tecnologías que permiten hacer de manera sencilla las tareas para las que se desarrolla el proyecto, se pueden encontrar infinidad de alternativas, por tanto, vamos a tratar las más importantes:

WordPress

WordPress es un CMS, o sistema de gestión de contenidos, que permite crear de manera, más o menos sencilla, cualquier tipo de página web, desde un blog hasta un comercio electrónico sencillo.

La tecnología en sí es gratuita, aunque también dispone de un *Marketplace* en el que comprar plantillas de página web propias de terceros, y también de una serie de planes de contratación premium. De hecho, según las cifras que ofrecen, el 41% de las páginas web están hechas con WordPress.

Esta tecnología permite a sus usuarios principalmente:

- Crear todo tipo de páginas web de manera sencilla.
- Editar rápidamente el HTML y los textos mostrados en la página web. Estos cambios, además, se ven reflejados en el momento.
- Insertar contenido multimedia de manera sencilla.
- Insertar nuevos componentes en su página web.
- Trabajar en los aspectos **SEO** [12] de la página web de una manera muy visual.

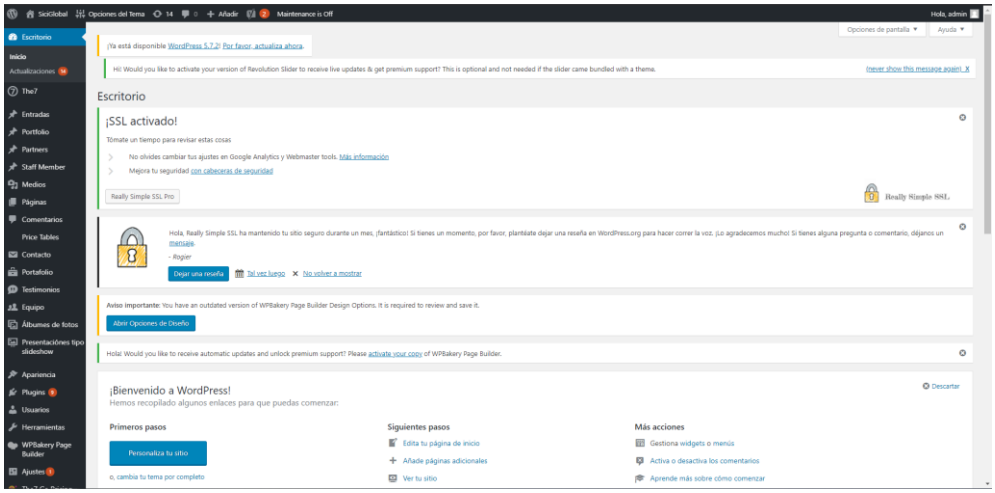

Ilustración 4: WordPress pantalla principal CMS

Presenta una interfaz que, si bien al principio puede resultar abrumadora, al poco tiempo se convierte en muy intuitiva. Y configurar nuevas páginas y componentes es una tarea realmente sencilla.

En realidad, los únicos inconvenientes con esta tecnología son relativos a su vejez. Existe desde hace más de quince años y eso pasa factura, las páginas web son mucho más que un servidor y un dominio, hoy en día.

PrestaShop

PrestaShop también es un CMS, pero libre y de código abierto, que permite crear desde cero comercios electrónicos. Está especialmente pensado para la creación de tiendas online de pequeñas empresas.

Es una tecnología gratuita que cuenta con un mercado de más cinco mil módulos de contenido, algunos gratuitos y otros de pago, que permiten adaptar y extender las funcionalidades de la herramienta según las necesidades del proyecto. Actualmente, más de trescientas mil tiendas online hacen uso de esta tecnología y está disponible en más de setenta y cinco idiomas.

Las principales características que ofrece a sus clientes son:

- Crear y modificar productos con facilidad.
- Navegar rápidamente entre productos.
- Gestionar sus stocks.
- Configurar la tienda, su contenido y su navegación con sencillez.
- Realizar todo tipo de gestiones legales, impuestos, cumplimiento normativo, etc.

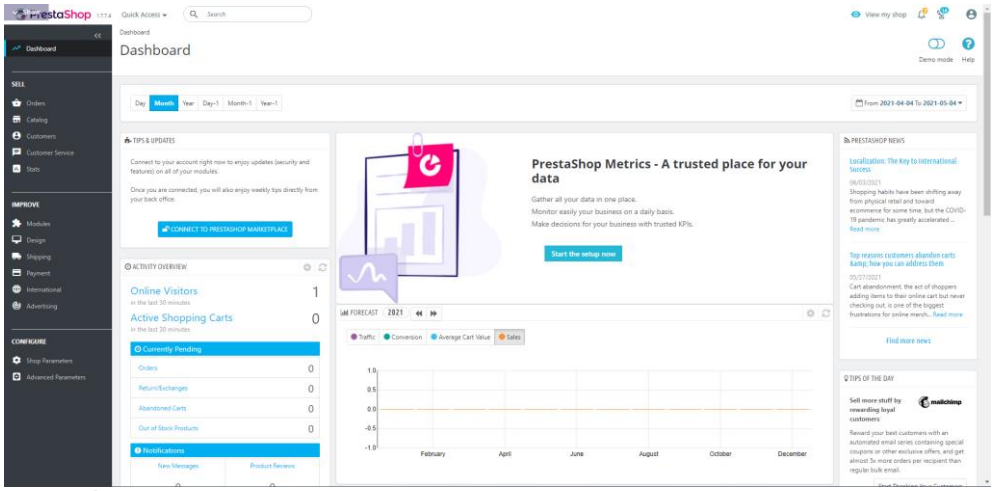

Ilustración 5: PrestaShop pantalla principal CMS

Presenta una interfaz mucho más elegante y organizada que la de la anterior tecnología. Además, se nota que está mucho más enfocada en comercios electrónicos, las gráficas de análisis de datos están siempre al alcance de la mano.

En la misma línea que la anterior tecnología, su única pega es su vejez. Incluir nuevas tecnologías en proyectos que implementen PrestaShop, es una tarea complicada.

Sonata

Sonata es un software construido sobre la tecnología Symfony, que permite crear de manera muy sencilla, y con muy pocos archivos de configuración, un **CRM** [13] interno a pequeña escala. Es un proyecto de código abierto, en el que cada paquete de código cubre una necesidad concreta y está pensado para poder ser ejecutado de manera independiente al resto de paquetes o módulos.

En concreto, el módulo de código que permite crear el CRM a pequeña escala es el **Admin Bundle** [14]. Esta tecnología es un ejemplo a seguir, durante más de diez años se ha mantenido actualizada y ha avanzado con las corrientes tecnológicas modernas, pudiendo incluso ejecutarse desde un contenedor de **Docker** [15]

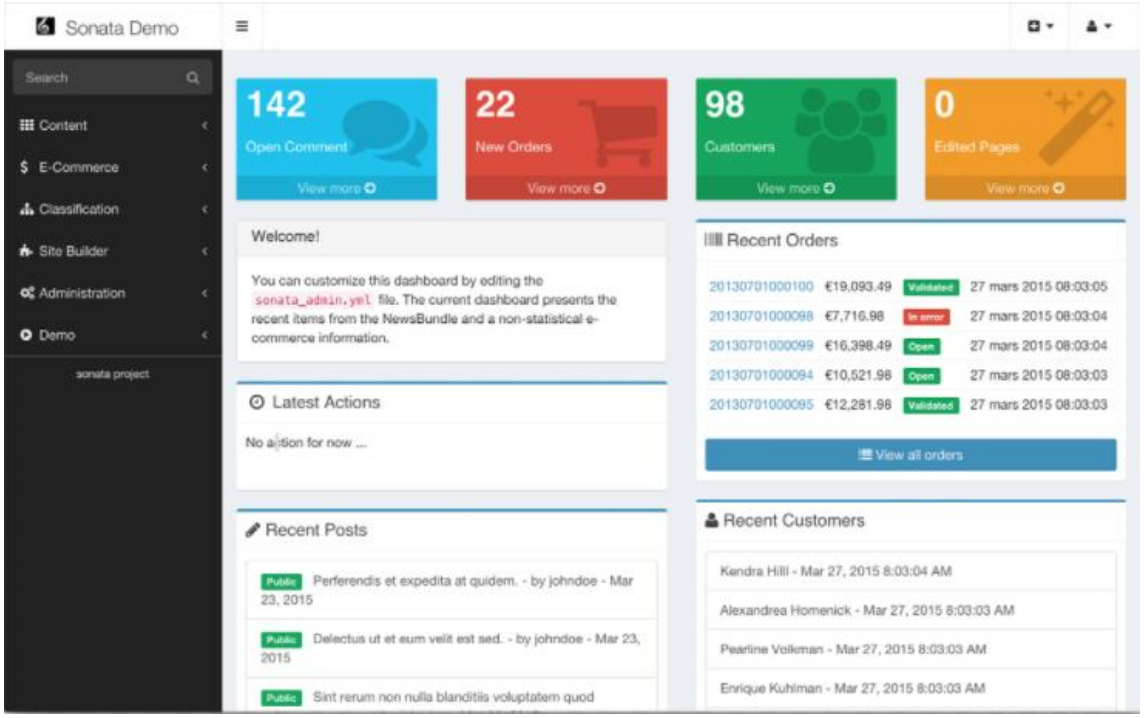

Ilustración 6: Sonata Admin pantalla principal CRM

Sin embargo, como contraposición tenemos que esta tecnología no está pensada para funcionar por si sola, siempre depende de estar instalada en las dependencias de un proyecto que implemente la tecnología Symfony.

Observaciones

Como se ha podido observar, las diferentes tecnologías mostradas, tienen como objetivo facilitar la manipulación de los datos de una página web, así como su contenido y estructura y en algunos casos pueden llegar a ofrecer funcionalidades extra, como la generación de informes.

Sin embargo, como punto común, todas ellas tienen el mismo problema de acoplamiento, falta de libertad y poca escalabilidad. Como se ha visto previamente, el punto más importante de este proyecto es la alta disponibilidad y escalabilidad que ofrecen las tecnologías en la nube, además del desacoplamiento que supone del marco de trabajo en sí.

Para el cliente va a ser igual de sencillo o complicado editar los datos, pero desde luego va a ahorrar mucho tiempo en no tener que formarse para preparar el entorno sobre el cual ejecutar estas tecnologías.

Es por estos motivos, que el desarrollo de un SaaS con arquitectura deslocalizada y auto escalable en la nube cobra mucho sentido, en contraposición a las alternativas existentes en el mercado.

2.2 Análisis del Público objetivo

El público al cual se enfoca este proyecto es hacia cualquier negocio o comercio que esté comenzando su digitalización. De manera sencilla, podría contratar la realización de su página web y ganar acceso a la plataforma que le permite modificar su información de manera sencilla.

Es también público objetivo, todas aquellas empresas que ya estén digitalizadas y cuya infraestructura se encuentre en AWS, dado que únicamente necesitarían generar su archivo de configuración para poder empezar a hacer uso de la plataforma.

2.3 Análisis de las tecnologías

Tecnologías Frontend

En la parte del Frontend de la plataforma, el objetivo es desarrollar interfaces de usuario de manera sencilla y rápida, sin invertir demasiado tiempo en formación en nuevas tecnologías.

Teniendo en cuenta la experiencia del autor del proyecto, las tecnologías a comparar y sobre las que escoger, deben tener como base el lenguaje **JavaScript** [16]. Esto no reduce el cerco especialmente, existen una gran cantidad de librerías y *frameworks* entre los que escoger. Sin embargo, este análisis se centrará en los tres *frameworks* más importantes: **VueJS** [17]**, ReactJS** [18] **y Angular** [19]**.**

Ilustración 7: Análisis de tecnologías (logos)

VueJS

VueJS o Vue.js, es un *framework* de código abierto escrito en JavaScript y enfocado en el desarrollo de interfaces de usuario. Dentro de sus características más importantes tenemos:

- Curva de aprendizaje muy sencilla.
- Arquitectura en un solo archivo, que permite mantener un proyecto muy limpio y organizado.
- Es relativamente reciente, pero está ganando mucha popularidad y comunidad.

ReactJS

ReactJS o React.js, es un *framework* de código abierto desarrollado por Facebook y enfocado en el desarrollo de aplicaciones de página única o **SPA** [20]. Dentro de sus características más importantes tenemos:

- Hace uso del llamado DOM Virtual para ofrecer un gran rendimiento en tiempos de renderizado de componentes.
- La transmisión de datos es jerárquica y unidireccional.
- Tienes la mayor comunidad de desarrolladores entre todos los *framework* de JavaScript.

Angular

Angular, es un *framework* de código abierto desarrollado por Google, escrito en **TypeScript** [21] y enfocado en el desarrollo de aplicaciones de página única. Dentro de sus características más importantes tenemos:

- Ofrece el mayor rendimiento en aplicaciones de página única.
- Cuenta con un **CLI** [22] muy potente que permite empezar a desarrollar aplicaciones muy rápidamente.
- Su arquitectura se enfoca en crear proyectos que sean muy modulares.

A simple vista, podemos ver que ofrecen una serie de características muy similares, por lo que no hay una respuesta clara para elegir la mejor solución. Sin embargo, teniendo en cuenta la experiencia del autor del proyecto, y esta es con ReactJS.

Además, si nos fijamos en los gráficos de las encuestas realizadas por la empresa Stack Overflow en 2019, podemos ver que ReactJS es la tecnología más querida por los desarrolladores y la que más cuota de mercado recoge:

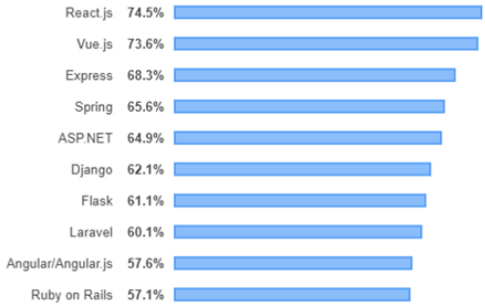

Ilustración 8: Framework más querido por los desarrolladores. Encuesta 2019

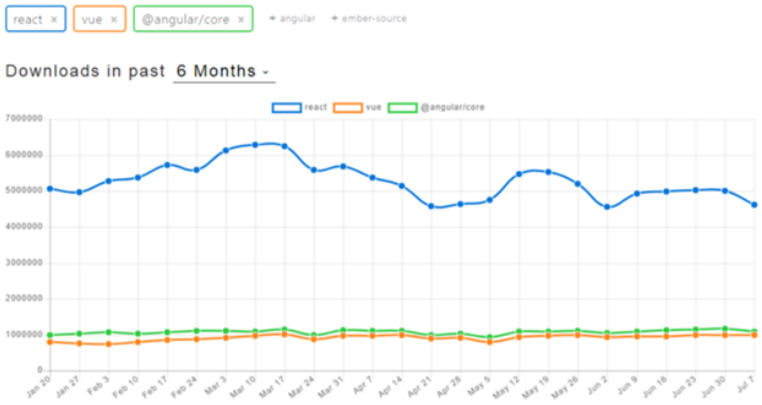

Ilustración 9: Framework más descargado. Encuesta de 2019

Cabe remarcar que ReactJS en sí no va a ser la tecnología empleada para desarrollar el proyecto, sino **NextJS** [23]. NextJS o Next.js es un *framework* construido sobre ReactJS y que está enfocado en ofrecer la mejor experiencia durante el desarrollo de páginas web que sean híbridas estáticas o que implementen *Server-Side Rendering*, términos que expandiremos en los siguientes puntos de la memoria.

NextJS está especializado en el tratamiento de rutas dinámicas, optimización de imágenes, paso a producción de aplicaciones y además ofrece soporte para TypeScript. TypeScript también va a ser la librería en la cual se escriba el proyecto, en lugar de hacer simplemente uso de JavaScript.

Tecnologías Backend

En la parte del Backend de la plataforma, el objetivo es desarrollar funciones Lambda, que puedan ser servidas desde una API REST encargada de gestionar las peticiones http realizadas desde el cliente web.

Como en el caso de las tecnologías Frontend, hay que tener en cuenta la experiencia del autor del proyecto y los lenguajes que las funciones Lambda de AWS ofrecen. Dado que el autor tiene, mayoritariamente, experiencia en JavaScript y **AWS Lambda** [24] ofrece **Node.js** [25], este será el elegido.

Node.js

Node.js es un entorno de programación en tiempo de ejecución de código abierto, basado en JavaScript y enfocado en el desarrollo de arquitecturas de servidor para aplicaciones escalables. Se basa en el motor V8 de Google.

2.4 Análisis de riesgos y plan de contingencias

En el plan de trabajo establecido en la planificación temporal, las principales preocupaciones o factores de riesgos son:

- Subestimación de la complejidad en la programación de la lógica de negocio que se encargue de renderizar dinámicamente los atributos de las entidades en la base de datos o del tratamiento de los propios datos en base al tipo de estos.

- Subestimación de la complejidad para añadir nuevos campos a una tabla de la base de datos mediante una API REST en la nube, pues la función Lambda asociada, debe ser capaz de crear el nuevo campo en una base de datos remota.

- Subestimación de la complejidad de integración de librerías de terceros.

- Imposibilidad para sortear los problemas que pueda causar el **CORS** [26] (Cross-Origin Resource Sharing) a la hora de realizar peticiones a un subdominio para persistir los cambios en los datos.

- Subestimación del alcance real del proyecto, por la necesidad de realizar muchas subtareas, como el proceso de Login, toda la API REST, infraestructura en la nube, etc.

Otro punto crítico es, de hecho, alcanzar la fecha límite de entrega sin haber podido cumplir la planificación inicial en si misma.

Como previsión a todos estos inconvenientes, ya hay una serie de tareas sobreestimadas a lo largo de la planificación temporal, con el objetivo de recortar sus tiempos si fuera necesario. Si esto también fallase, se decidiría recortar funcionalidades que es interesante aportar, pero no son decisivas en el correcto funcionamiento de la aplicación. Ejemplos de esto sería la responsividad de la

plataforma en resoluciones reducidas o la documentación de algunos de los componentes de la aplicación.

3. Requisitos

3.1 Requisitos funcionales

Llegados a este punto, queda clara la situación inicial desde la que parte el proyecto. También las funcionalidades, a grandes rasgos, que debe alcanzar la aplicación, para al menos obtener un funcionamiento sin complicaciones y, por tanto, es posible determinar los requisitos funcionales del proyecto:

- Los usuarios registrados deben poder acceder a su panel de administración y consultar los datos de sus entidades disponibles.
- Los tipos de entidades disponibles deben extraerse automáticamente del archivo de configuración asociado a la cuenta del usuario.
- Los usuarios autorizados deben poder crear, editar y eliminar las entidades de los tipos de entidades disponibles.
- Los cambios realizados deben aplicarse instantánea y automáticamente a los datos mostrados en el frontal.
- Los usuarios deben poder consultar su perfil y modificar sus datos, como si de una entidad se tratase.

3.2 Requisitos no funcionales

Derivados de los requisitos funcionales del sistema, se extraen una serie de requisitos no funcionales, pero que deben cumplirse para garantizar los requisitos anteriores. Estos requisitos no funcionales son:

- El sistema sólo debe permitir acceder al usuario a aquellas entidades a las que tiene acceso, ninguna que pertenezca a otro usuario o grupo de usuarios.
- El sistema debe estar disponible durante las veinticuatro horas del día, todos los días del año.
- El sistema debe funcionar correctamente en cualquier dispositivo y navegador.
- El sistema debe ser fácil de usar.
- Las actualizaciones del sistema no deben poner en riesgo la estabilidad de este y no debe tener vulnerabilidades que pongan en peligro los datos sensibles del usuario.

3.3 Requisitos tecnológicos

Dados los requisitos funcionales y no funcionales del proyecto, es momento de definir las tecnologías y herramientas empleadas para el desarrollo de este.

Tecnologías Frontend

A continuación, se listan las tecnologías relativas al Frontend empleadas:

- **NextJS:** Es un framework de desarrollo Frontend construido sobre ReactJS, que ofrece la mejor experiencia en el desarrollo de páginas web, ya sean híbridas estáticas o implementen Server-Side Rendering. Este framework está especialmente preparado para el paso a producción de las aplicaciones y ofrece soporte para TypeScript, facilidad para la creación de rutas, optimización de imágenes, etc.

- **ReactJS:** Es una librería de código abierto escrita en JavaScript, que se enfoca en el desarrollo de interfaces de usuario. Esta tecnología es mantenida por Facebook y la respalda una gran comunidad.

- **TypeScript:** Es una librería de código abierto, que extiende las funcionalidades de JavaScript mediante el tipado. Permite ahorrar mucho tiempo en el desarrollo de aplicaciones de JavaScript sobre todo en cuanto a la detección de errores se refiere.

- **JavaScript:** Es un lenguaje diseñado, principalmente, para implementar funciones complejas en el ámbito del desarrollo web. Es ligero, interpretado y orientado a objetos.

- **Sass** [27]**:** Es un lenguaje de diseño gráfico que es compilado en CSS. Se emplea para conseguir mantener organizados los estilos de una aplicación conforme va aumentando de tamaño. Permite el uso de variables, reglas de estilo anidadas, funciones, etc.

- **CSS** [28]**:** Es un lenguaje de diseño gráfico, empleado para definir, estilizar y crear la presentación de un documento escrito en un lenguaje de marcado, como puede ser HTML. CSS es la tecnología usada por muchos sitios web para crear páginas visualmente atractivas.

- **Axios** [29]**:** Es una librería escrita en JavaScript, para facilitar la realización de peticiones HTTP. Permite realizar de manera muy sencilla peticiones HTTP desde Node.js, peticiones XMLHttpRequests desde el navegador, transforma automáticamente la información a formato JSON, etc.

Tecnologías Backend

A continuación, se listan las tecnologías relativas al Backend empleadas:

- **Node.js:** Es un entorno de programación en tiempo de ejecución de código abierto, basado en JavaScript y enfocado en el desarrollo de arquitecturas

de servidor para aplicaciones escalables. Se basa en el motor V8 de Google.

- **JSON** [30]**:** Acrónimo de JavaScript Object Notation. Esto significa que su estructura son objetos que siguen la notación de JavaScript. Es un estándar basado en texto cuyo objetivo es el intercambio de datos con una estructura que pueda ser legible por las personas.

- **JWT** [31]**:** Acrónimo de JSON Web Token. Es un estándar abierto basado en JSON para la creación de tokens de acceso que permite propagar identidad y privilegios.

Otras tecnologías

A continuación, se listan las tecnologías propias de terceros (AWS) empleadas:

- **AWS** [32]**:** Acrónimo de Amazon Web Services. Es un conjunto de servicios de computación en la nube que forman una plataforma de computación en la nube, ofrecida por Amazon.

- **S3** [33]**:** Acrónimo de Simple Storage Service. Es un servicio de almacenamiento que ofrece escalabilidad, alta disponibilidad de datos, seguridad y alto rendimiento. Forma parte del ecosistema de AWS.

- **DynamoDB** [34]**:** Es una base de datos del tipo clave-valor y documentos, que ofrece rendimiento en milisegundos de un sólo dígito a cualquier escala. Puede gestionar más de 10 billones de solicitudes al día y admite picos de más de 20 millones de solicitudes por segundo. Forma parte del ecosistema de AWS.

- **Amazon Cloudfront** [35]**:** Es un CDN (Content Delivery Network) rápido, seguro y programable. Permite distribuir datos, aplicaciones y APIs de forma global y completamente segura con bajas latencias y altas velocidades. Forma parte del ecosistema de AWS.

- **API Gateway** [36]**:** Es un servicio que facilita a los desarrolladores, la creación, publicación, mantenimiento, monitoreo y protección a cualquier escala de APIs. Con este servicio se pueden crear tanto API RESTful cómo API WebSockets. Forma parte del ecosistema de AWS.

- **AWS Lambda:** Es un servicio informático para ejecutar código sin servidor. Permite el uso de diferentes lenguajes cómo Python, Node.js, Java, etc. al escribir las funciones y ejecutarlas siguiendo un modelo por contenedores (CLI de Docker) o un modelo sin servidor, es decir el AWS Serverless Application Model. Forma parte del ecosistema de AWS.

- **Amazon Cognito:** Es un servicio informático que permite desarrollar directorios y estructuras de usuarios de forma segura y escalable. Admite

también, el inicio de sesión mediante proveedores de identidad de terceros, como puede ser Google, Facebook o Amazon.

- **AWS Amplify** [37]**:** Es un conjunto de herramientas y servicios que ayuda a los desarrolladores a crear y desplegar aplicaciones completamente escalables y de manera muy sencilla. Se integra con casi todos los servicios de repositorios de código actuales y soporta casi todos los *framework* web conocidos.

Observaciones

El proyecto está principalmente desarrollado en JavaScript y CSS. Concretamente, el Frontend está construido con: JavaScript, bajo la librería de TypeScript, que a su vez está bajo la librería de NextJS y CSS, bajo la librería o preprocesador Sass. No se ha hecho uso de ningún *framework* de maquetación.

En cuanto al Backend, está construido con JavaScript, bajo la librería de Node.js. Esto es así por ser la librería ofrecida por el servicio AWS Lambda, para la programación de funciones Lambda en JavaScript.

La elección de estas tecnologías es únicamente por dos motivos: son tecnologías ampliamente conocidas y con un gran soporte, y por experiencia del autor en ellas. En otras palabras, son las tecnologías con un mayor grado de comodidad para desarrollar el proyecto.

Por último, la base de datos elegida es DynamoDB. La elección de esta tecnología se debe a ser una base de datos documental, fácilmente integrable con funciones Lambda y editable desde estas. Además, tiene un servicio de auto escalado que encaja perfectamente con el objetivo del proyecto.

3.4 Casos de uso

Definidas las funcionalidades del proyecto y sus tecnologías, cabe determinar los actores principales del sistema y las acciones que pueden realizar.

Actores del Sistema

En la aplicación intervendrán tres actores principales:

- **Usuario:** Usuarios comunes de la aplicación. Son aquellos usuarios que únicamente pueden acceder al frontal.

- **Usuario administrador:** Usuarios con acceso al CRM. Son aquellos usuarios que tienen credenciales de acceso al panel administrativo, y que pueden modificar los datos de sus entidades asociadas

- **Sistema:** La propia aplicación. Su función es validar usuarios, que puedan realizar las acciones que desean, prohibir el acceso a páginas específicas, listar dinámicamente las entidades del usuario autenticado, etc.

Casos de uso del usuario

A continuación, se lista los casos de uso principal de un usuario común.

Caso de uso	*Visualizar página web frontal pública*
Funcionalidad	Permitir a un usuario común, acceder a la página web pública
Actor Principal	Usuario
Nivel de Objetivo	Usuario
Intereses	El usuario quiere ver la página web del cliente específico
Precondiciones	El usuario es un usuario cualquiera que desee acceder a la plataforma del cliente La página web es accesible
Garantías en caso de éxito	El sistema resuelve satisfactoriamente la petición y muestra al usuario la página web del cliente, incluyendo los datos necesarios que deban ser leídos de las bases de datos asociadas
Escenario principal de éxito	1. El sistema lee la información necesaria de las bases de datos 2. El sistema resuelve la petición satisfactoriamente

Caso de uso	*Visualizar página web administrativa privada*
Funcionalidad	Denegar a un usuario común, acceder a la página web privada
Actor Principal	Usuario
Nivel de Objetivo	Usuario
Intereses	El usuario quiere ver una página web a la que no tiene acceso
Precondiciones	El usuario es un usuario no autenticado El usuario desea acceder a la plataforma de administración La página web es accesible
Garantías en caso de éxito	El sistema deniega la petición y redirige al usuario no autenticado a la pantalla de *Login*
Escenario principal de éxito	1. El sistema valida que el usuario no está autenticado 2. El sistema redirige al usuario a la pantalla de *Login*

Casos de uso del usuario administrador

A continuación, se lista los casos de uso principal de un usuario administrador. Cabe remarcar, que un usuario administrador, no es necesariamente administrador del sistema, solo administrador de sus entidades y bases de datos asociadas.

Caso de uso	*Login*
Funcionalidad	Permitir a un usuario administrador, entrar en la plataforma
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere acceder a la plataforma
Precondiciones	El usuario está previamente registrado
Garantías en caso de éxito	El sistema redirige al usuario a la pantalla inicial de la aplicación
Escenario principal de éxito	1. El sistema requiere el email y la contraseña del usuario 2. El sistema valida las credenciales 3. El sistema redirige al usuario a la pantalla inicial de la aplicación

Caso de uso	*Modificar el perfil*
Funcionalidad	Permitir a un usuario administrador cambiar su información personal
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere modificar sus datos personales
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión
Garantías en caso de éxito	El sistema muestra al usuario un mensaje indicando que los cambios se han guardado satisfactoriamente y le permite consultarlos
Escenario principal de éxito	1. El sistema muestra los datos actuales 2. El usuario modifica los datos 3. El sistema valida los cambios realizados 4. El sistema muestra los nuevos datos

Caso de uso	*Consultar listado de un tipo de entidad*
Funcionalidad	Permitir a un usuario administrador, consultar el listado de entidades de un tipo de entidad en concreto
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere acceder al listado de un tipo de entidades
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión
Garantías en caso de éxito	El usuario ve el listado de entidades proporcionado por el sistema
Escenario principal de éxito	1. El sistema comprueba que la tabla asociada especificada contiene entidades a mostrar 2. El sistema muestra al usuario el listado de entidades

Caso de uso	*Consultar entidad de un tipo de entidad*
Funcionalidad	Permitir a un usuario administrador, consultar una entidad de un tipo de entidad en concreto
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere acceder a la vista en detalle de una entidad de un tipo de entidades en concreto
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión El listado de dicho tipo de entidades no está vacío La entidad existe
Garantías en caso de éxito	El usuario ve el detalle de la entidad requerida
Escenario principal de éxito	1. El sistema comprueba que la entidad existe 2. El sistema muestra al usuario el detalle de la entidad

Caso de uso	*Crear entidad de un tipo de entidad*
Funcionalidad	Permitir a un usuario administrador, crear una entidad de un tipo de entidad en concreto
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere añadir una nueva entidad al listado de un tipo de entidades

Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión
Garantías en caso de éxito	El usuario crea la entidad satisfactoriamente
Escenario principal de éxito	1. El sistema comprueba que la información requerida es válida 2. El sistema crea la entidad y la muestra en el listado de entidades al usuario

Caso de uso	*Editar entidad de un tipo de entidad*
Funcionalidad	Permitir a un usuario administrador, editar una entidad de un tipo de entidad en concreto
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere añadir una nueva entidad al listado de un tipo de entidades
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión El listado de dicho tipo de entidades no está vacío La entidad existe
Garantías en caso de éxito	El usuario edita la entidad satisfactoriamente
Escenario principal de éxito	1. El sistema comprueba que la información requerida es válida 2. El sistema edita la entidad y la muestra actualizada en el listado de entidades al usuario

Caso de uso	*Eliminar entidad de un tipo de entidad*
Funcionalidad	Permitir a un usuario administrador, eliminar una entidad de un tipo de entidad en concreto
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere eliminar una nueva entidad del listado de un tipo de entidades
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión El listado de dicho tipo de entidades no está vacío La entidad existe
Garantías en caso de éxito	El usuario edita la entidad satisfactoriamente
Escenario principal de éxito	1. El sistema comprueba que la entidad puede ser eliminada 2. El sistema elimina la entidad y muestra el listado de entidades actualizado al usuario

Caso de uso	*Cerrar sesión*
Funcionalidad	Permitir a un usuario administrador cerrar su sesión actual
Actor Principal	Usuario administrador
Nivel de Objetivo	Usuario administrador
Intereses	El usuario quiere cerrar su sesión
Precondiciones	El usuario está previamente registrado El usuario ha iniciado previamente sesión
Garantías en caso de éxito	El usuario cierra sesión y sale de la plataforma
Escenario principal de éxito	1. El sistema comprueba que las credenciales son correctas 2. El sistema invalida la sesión del usuario e invalida sus cookies

	3. El sistema redirige al usuario a la pantalla de inicio de sesión

Cabe remarcar, que para los casos de: crear, ver, editar y eliminar entidades, únicamente se ha dado un caso de uso, sin embargo, aplica de modo genérico a cualquier tipo de entidad del sistema.

Casos de uso del sistema

A continuación, se lista los casos de uso principal del sistema.

Caso de uso	*Importar tipos de entidades*
Funcionalidad	Generar los tipos de entidades a partir del archivo de configuración
Actor Principal	Sistema
Nivel de Objetivo	Sistema
Intereses	El sistema importa los tipos de entidad asociados al proyecto de un usuario, para poder generar las entradas de la barra de navegación
Precondiciones	Existe el archivo de configuración necesario El usuario tiene el identificador de su proyecto configurado correctamente
Garantías en caso de éxito	El sistema muestra correctamente los datos
Escenario principal de éxito	1. El usuario ha importado previamente su archivo de configuración 2. El usuario ha configurado su identificador del proyecto 3. El sistema muestra sus tipos de entidad asociadas

Caso de uso	*Comprueba los cambios de los usuarios*
Funcionalidad	Comprobar la validez en los cambios que los usuarios tratan de persistir
Actor Principal	Sistema
Nivel de Objetivo	Sistema
Intereses	El sistema valida y persiste los datos a modificar
Precondiciones	-
Garantías en caso de éxito	El sistema persiste los cambios en los datos
Escenario principal de éxito	1. El usuario quiere realizar un cambio en cualquier tipo de datos 2. El sistema valida que los cambios puedan ser realizados 3. En caso afirmativo, el sistema persiste los cambios 4. En caso negativo, el sistema deniega los cambios y notifica al usuario del problema

3.5 Alcance del proyecto y limitaciones

El proyecto debe tener un alcance realista y determinado, dada la complejidad que podría llegar a tener. En primera instancia se plantea la creación de diferentes interfaces para la gestión de las entidades disponibles:

- **Gestión de elementos de navegación de la cabecera:** O *Navigation* Items. La gestión de este tipo de entidad permitirá controlar los elementos de navegación que aparezcan o no en la cabecera de la página web.

- **Gestión de redes sociales:** O *Social Media.* La gestión de este tipo de entidad permitirá controlar las redes sociales que aparezcan o no en el cuerpo de la página web.

- **Gestión de términos legales:** O *Legal Blocks.* La gestión de este tipo de entidad permitirá controlar los elementos de términos legales que se muestran en el pie de la página web, así como el texto de dichas páginas.

- **Gestión de perfil de usuario:** La gestión de este tipo de entidad, permitirá editar los datos del usuario activo, así como su email de acceso.

Dicho esto, cabe limitar lo que el proyecto va a ser capaz de manejar inicialmente:

- Se descarta la creación de cuentas de nuevos usuarios. Inicialmente existirá un usuario que hará de usuario administrativo del supuesto cliente.

- Se asume que el cliente, en la mayoría de los casos, debería ser un nuevo cliente a quien, posiblemente, se le tuviese que preparar toda la arquitectura en AWS junto con su archivo de configuración, para poder hacer uso debido de la plataforma.

- Se descarta la creación de nuevas páginas para la página web del cliente, únicamente se podrán añadir nuevas páginas bajo la ruta *"/legal".* Estas serán las páginas configurables desde el tipo de entidad de términos legales.

- La página web sobre la que se vean reflejados los cambios en las entidades estará bajo el mismo dominio que el sistema de gestión. Aunque bien podría ser una página web independiente en otro dominio.

4. Diseño y Arquitectura

4.1 Arquitectura general de la aplicación

Como se ha comentado a lo largo de la memoria, el proyecto implementa una arquitectura de computación en la nube y sigue el modelo de aplicaciones sin servidor o **SAM** [38]. Para tratar de clarificar esto, podemos diferenciar tres estructuras clave:

- **Contenido estático:** Estos son los archivos que NextJS generará cómo resultado de ejecutar su comando *build*. Este comando prepara todo el proyecto para ser servido mediante Server Side Rendering, de esta manera, el usuario no podrá ver cómo JavaScript va mostrándose poco a poco debido a las diferentes peticiones asíncronas, sino que se mostrará todo cuando esté listo para ser mostrado.

Estos archivos estáticos se almacenan en S3, en los llamados ***buckets*** [39]. Estos buckets, pueden ser configurados para servir su contenido como sitios web estáticos. Esto significa, que al acceder a la URL que el *bucket* expone, este mapea automáticamente su contenido de forma estática, mostrando por defecto, en cada carpeta, su archivo *index.html*.

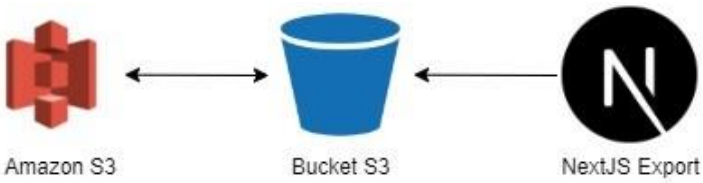

Ilustración 10: Diagrama contenido estático

- **Red de distribución de contenido:** Esto se refiere a la combinación de tres servicios: **Route 53** [40], **Certificate Manager** [41] y Amazon Cloudfront. Estos servicios son los encargados de servir el contenido estático bajo un dominio determinado y de la manera más óptima posible.

El flujo es el siguiente es un poco complejo. En primer lugar, se crea un certificado de https con el servicio de Certificate Manager, para un dominio determinado. A continuación, se crea una distribución Cloudfront que asocia el *bucket* de S3 previamente creado, con el certificado generado y para un dominio determinado. Por último, en la zona hospedada del dominio en Route 53, se asocia el dominio, del cual se es propietario, al identificador de la distribución Cloudfront.

Mediante este proceso, el contenido estático expuesto por el bucket de S3, se vincula a un dominio, permitiendo una gestión de rutas de páginas web sin un servidor para manejar las peticiones. Esto permite obtener una página web completamente funcional estática y sin servidor, que, además, gracias a NextJS,

va a poder tratar peticiones asíncronas sin penalizar al tiempo de renderizado del cliente.

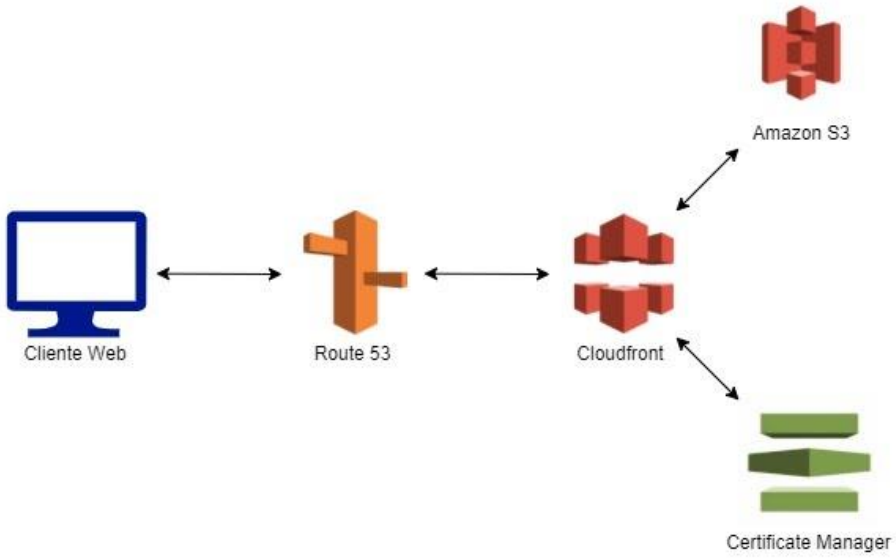

Ilustración 11: Diagrama Red de distribución de contenido

- **Serverless API Model:** O SAM. Esto de nuevo se refiere a la combinación de tres servicios: API Gateway, Amazon Lambda y DynamoDB. Estos servicios son los encargados de atender a las peticiones asíncronas, tanto en la carga de la página web como en las operaciones CRUD.

En este caso, el flujo es más sencillo. Simplemente es necesario crear las tablas que quieran ser modificadas en DynamoDB, a continuación, se crean funciones Lambdas con permisos para leer y escribir en DynamoDB y por último se crea en API Gateway, una API que permita conectar las Lambdas con un subdominio del dominio del cual se es propietario.

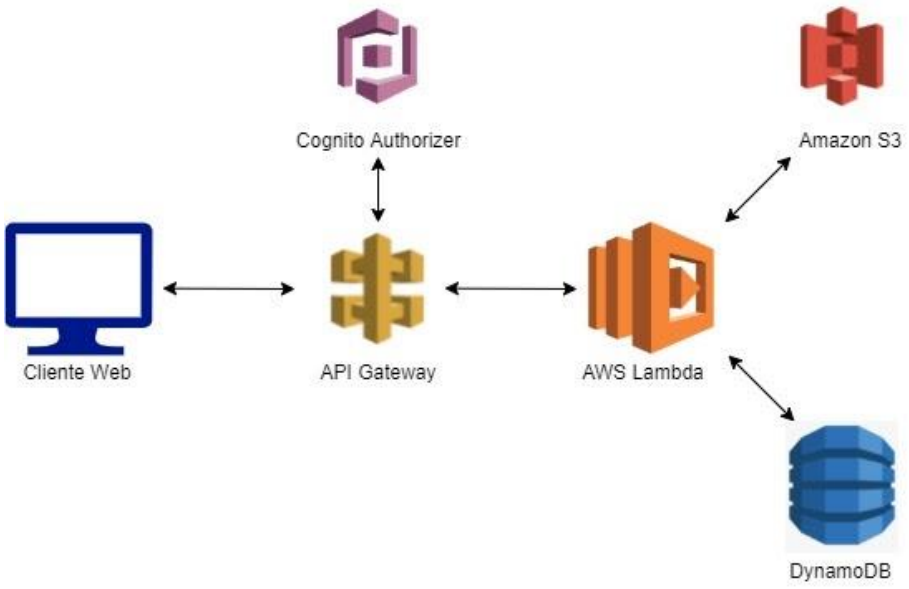

Ilustración 12: Diagrama Modelo API Serverless

Por último, remarcar un aspecto importante. En el caso de este proyecto, se ha hecho uso del servicio AWS Amplify. Este servicio ha sido necesario debido a la naturaleza Server-Side de la aplicación, pues necesitaba de una instancia auto escalable de Node.js para manejar este tipo de renderizado. En el apartado “5. Implementación”, se explica en detalle este proceso.

4.2 Arquitectura del modelo de datos

Una de las partes que compone este proyecto, es sus bases de datos asociadas al usuario. En un caso práctico real, seria el cliente quien aportaría esta información, sin embargo, para la correcta puesta en práctica se ha optado por preparar tres tipos de entidades.

La base de datos elegida es DynamoDB, es decir, una base de datos no relacional y, por tanto, no van a existir relaciones entre las diferentes tablas. La estructura de los tres tipos de entidades, como se plantea en la Ilustración 14, es sencilla. El identificador es el único campo obligatorio y es generado de manera automática, el resto de los campos de los documentos de la tabla son completamente opcionales:

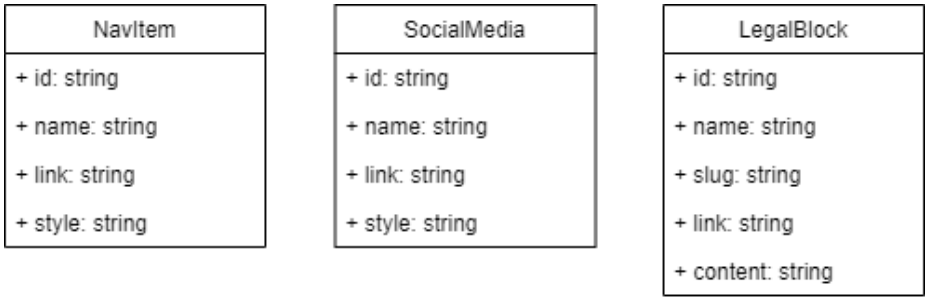

Ilustración 13: DynamoDB - Estructura de los documentos de datos

En tanto en cuanto que el modelo de datos es propio del autor, no ha sido necesario crear instancias, de la base de datos, alternativas para el proceso de desarrollo, pues la privacidad de los datos del cliente era inexistente.

Por otra parte, en cuanto a los usuarios, están configurados desde el *pool* de usuarios de Amazon Cognito. Desde este *pool*, se determina: una serie de atributos opcionales por defecto, otros atributos customizables opcionales u obligatorios, y aquellos atributos que se encargan de autenticar las sesiones del usuario, si esto no es manejado por un proveedor de servicios externo. Por otra parte, los usuarios tienen un campo *username*, interno de Cognito, que hace de identificador único.

Cómo podemos observar en la ilustración 15, sólo algunos de los atributos listados en la ilustración 14 aparecen. Esto se debe a, como ya hemos comentado, que sólo se listan aquellos atributos por defecto de Cognito y no la contraseña, el identificador o el identificador del proyecto.

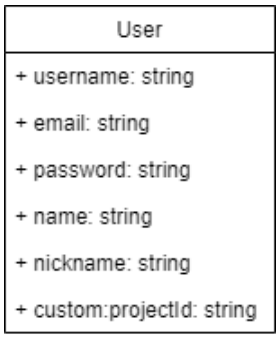

Ilustración 14: Cognito - Estructura de los datos de usuario

Ilustración 15: Cognito - Estructura de los datos de usuario 2

4.3 Diseño de la API REST

Toda la API REST se encontrará bajo el subdominio *api*. Desde este subdominio se definen diferentes *endpoints* para los diferentes casos de uso. Además, todas las peticiones van acompañadas de un ***query string parameter*** [42], que permite definir el id del *bucket* en el que encontrar la configuración esperada. La estructura es la siguiente:

- **/entities:** Este es el *endpoint* fijo que hace referencia a la función Lambda que, dependiendo del identificador de proyecto que reciba como parámetro, devuelve todos los tipos de entidades de un proyecto determinado. Este *endpoint* únicamente habilita el método **GET**.

- **/entities/{id}:** Este es el *endpoint*, de tipo variable, que dependiendo del identificador que proporciona, hace referencia a un tipo de entidad. Puede devolver las instancias de dicha entidad o bien crea una nueva. Sus métodos disponibles son los siguientes:

  - **Método GET:** Lista todas las instancias del tipo de entidad en cuestión.

  - **Método POST:** Crea una nueva instancia del tipo de entidad en cuestión.

- **/entities/{id}/{slug}:** Este es el *endpoint*, de tipo variable, que dependiendo del identificador de segundo nivel que proporciona, hace referencia a una instancia de entidad de un tipo de entidad. Puede devolver la instancia en concreto de dicha entidad o editarla o borrarla. Sus métodos disponibles son los siguientes:

    - **Método GET:** Lista los detalles de la instancia en cuestión.

    - **Método PUT:** Edita la instancia en cuestión.

    - **Método DELETE:** Borra la instancia en cuestión.

Por otro lado, hay una serie de *endpoints* que gracias a la integración con la librería del servicio Amazon Cognito, no es necesario que estén externalizados a una API REST. La librería integrada permite, mediante el uso de los parámetros de configuración del *pool* de usuarios, la creación, autenticación, edición y fin de sesión de usuarios. Las peticiones de las que se encarga Amazon Cognito son:

- **Login de usuarios:** Esta función de la librería, permite autenticar un usuario a partir de los campos para los cuales se ha determinado el acceso a la plataforma, en este caso email y contraseña.

- **Modificación de datos de usuario:** Esta función, cómo su nombre indica, permite editar los datos de un usuario autenticado, definidos como escribibles y que no sean su contraseña.

- **Modificación de contraseña de usuario:** De nuevo, función que permite modificar la contraseña del usuario autenticado, proporcionando su antigua contraseña y la nueva.

- **Cerrado de sesión:** Función que permite invalidar los tokens de sesión del usuario y de esta manera invalidar su sesión o cookie de sesión.

4.4 Diseño de los módulos

Con el objetivo de organizar la aplicación, está se ha divido en una serie de módulos conceptuales, que representan páginas dentro de la propia aplicación.

- **Login:** Página de acceso a la aplicación y se encuentra en */login*. Es el único contenido público de la parte privada de la aplicación. De cualquier modo, no puede irse más allá sin la correcta autenticación.

- **Módulo Inicio:** Página inicial tras iniciar sesión satisfactoriamente y se encuentra en la raíz del dominio. En esta vista se muestra un resumen de los tipos de entidades disponibles y un acceso directo a ellas.

- **Módulo de Gestión de entidades:** Conjunto de páginas desde las cuales se pueden realizar las operaciones CRUD sobre las entidades. Se

encuentra en la dirección */entities/{id}/{slug}*, donde el *id* hace referencia al tipo de entidad y *slug* a la instancia de la entidad.

- **Módulo de Gestión de usuario:** Página de gestión de la información personal del usuario. Se encuentra en */profile*. Si el usuario autenticado no tiene ningún proyecto asociado, estará limitado a esta vista y será siempre redirigido a ella.

4.5 Diseño Gráfico

Definición de estilos y prototipos del proyecto. Entre ellos se incluye: la paleta de colores, la tipografía, los prototipos, etc.

Paleta de colores

La paleta de colores elegida es un conjunto de tonos azules eléctricos, ampliamente utilizada en el sector de la tecnología, junto con complementarios como *new black*, blanco y pequeñas derivaciones del blanco para fondos de las interfaces. La paleta básica quedaría de la siguiente manera:

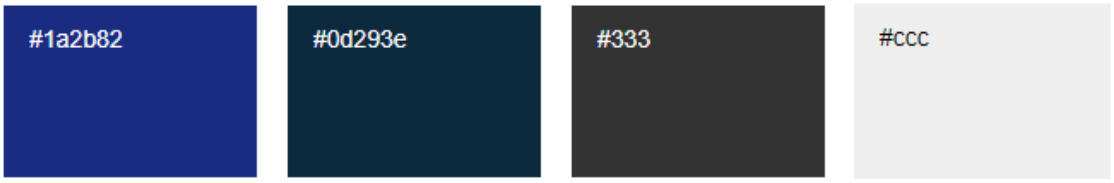

Ilustración 16: Paleta principal de colores

El color de los menús es el *#1a2b82* y *#0d293e* representa el estado activo de los elementos de los menús. Por otra parte, el color del texto plano siempre es #333 (*new black*) y, por último, una combinación de blanco puro y *#ccc* va a ser usado para los fondos base y de la cabecera del menú de administración.

Sin embargo, los colores no se han limitado a estos. Se han añadido una serie de colores derivados y auxiliares, para cubrir acciones y estados secundarios de la plataforma, como bien pueden ser micro interacciones, casos de error, etc. La paleta secundaria quedaría de la siguiente manera:

Ilustración 17: Paleta secundaria de colores

Tipografía

En cuanto a la tipografía, la fuente elegida es *"Raleway",* para todo el proyecto. Y los tamaños de fuente varían dependiendo del peso del texto. Es una fuente de código abierto proporcionada con Google y es conocida por ser la fuente en la que está escrita la página web de VueJS. Ha sido utilizada únicamente en dos pesos, normal y fuerte.

La siguiente ilustración, muestra los diferentes estilos en los que se sirve la fuente en la plataforma:

Título	Normal 20px	#333333
Título Bold	Bold 20px	#333333
Título Contenedor	Normal 16px	#333333
Título Contenedor Bold	Bold 16px	#333333
Texto Común	Normal 14px	#333333
Texto Común Bold	Bold 14px	#333333

Ilustración 18: Resumen tipografía

Iconos y otros elementos

La aplicación también ha hecho uso de iconos para favorecer la experiencia de usuario. Los iconos empleados en todo el proyecto son de dominio público y se corresponden con la capa gratuita de **FontAwesome** [43]. Los iconos pueden encontrarse en: https://fontawesome.com/icons?d=gallery&p=2 y algunos de ellos se muestran en la siguiente ilustración.

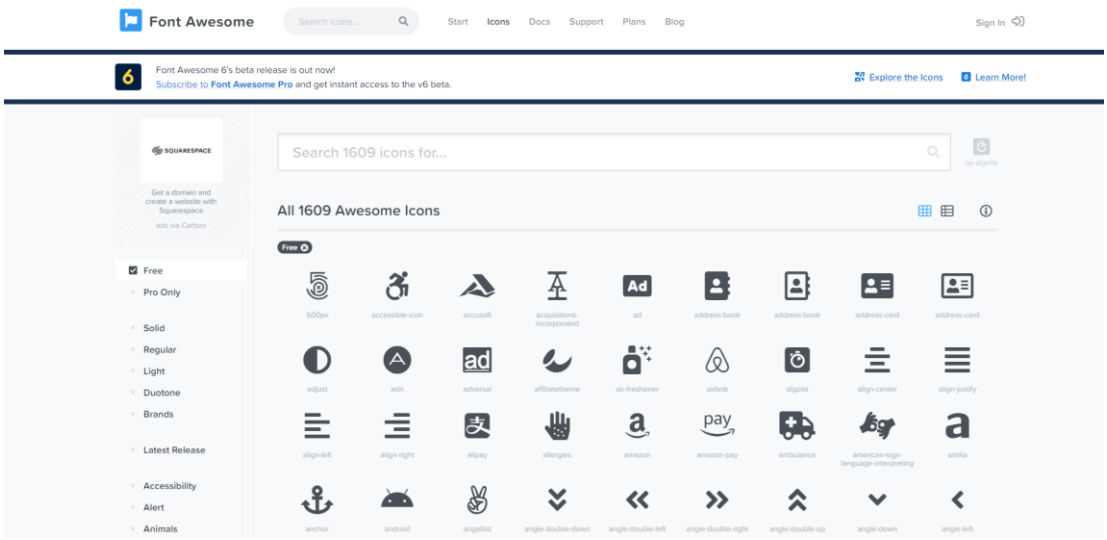

Ilustración 19: Muestra Iconos FontAwesome

Estructura General

Tal y como se ha comentado, el diseño de la aplicación es similar a una aplicación de página única o SPA, gracias al uso de NextJS. Teniendo eso en cuenta, la pantalla se dividirá en:

- **Menú Superior:** Zona donde se muestran los datos del usuario activo y el menú básico de acciones que este puede realizar.

- **Menú Lateral:** Zona donde se encuentra la navegación principal. Aquí se listan los tipos de entidades asociadas al proyecto del usuario activo. En la versión móvil, eso se mostraría pulsando un botón.

- **Contenido Central:** Zona principal donde se muestra el contenido de la vista actual y la información relevante.

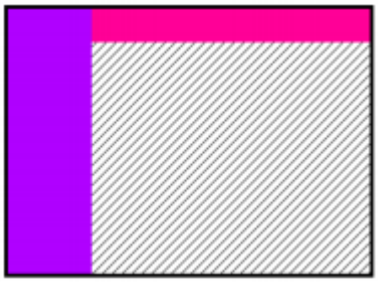

Ilustración 20: Esquema estructura SPA

Prototipado o Interfaz de baja fidelidad

Estos prototipos, en un caso real, se crearían en las primeras reuniones con el cliente, consultando sus ideas y preferencias referentes a la disposición de los elementos en la aplicación. En este caso, el autor del proyecto es quien ha realizado dicha tarea.

A continuación, se muestran los diseños base de las interfaces del panel de administración. Estos diseños son prototipos de baja fidelidad y que no representan el diseño final de la aplicación:

- **Vista de Login:**

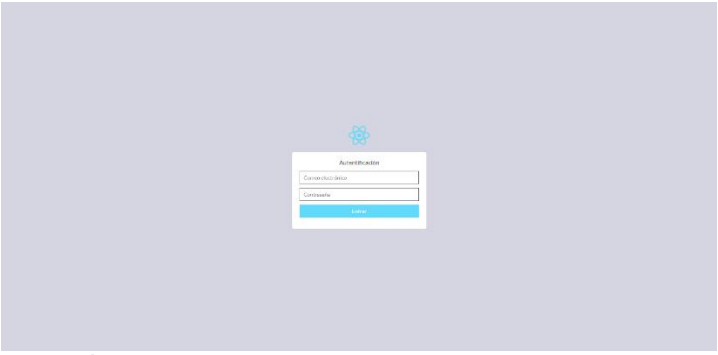

Ilustración 21: Vista Login - Baja Fidelidad

- **Vista Principal:**

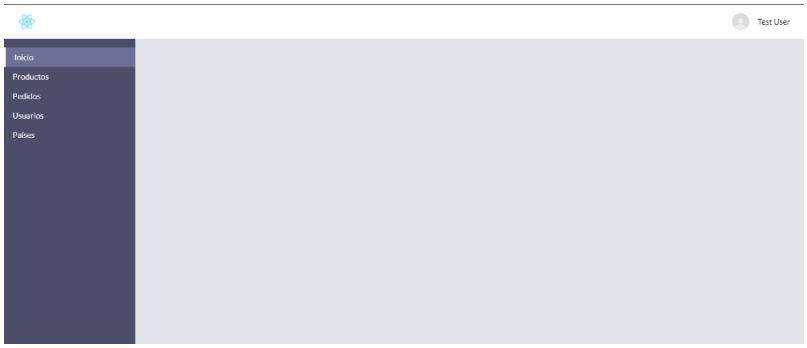

Ilustración 22: Vista Principal - Baja Fidelidad

- **Vista Entidades:**

Ilustración 23: Vista Entidades - Baja Fidelidad

- **Vista de Login (Móvil):**

Ilustración 24: Vista Login (Móvil) - Baja Fidelidad

- **Vista de Principal (Móvil):**

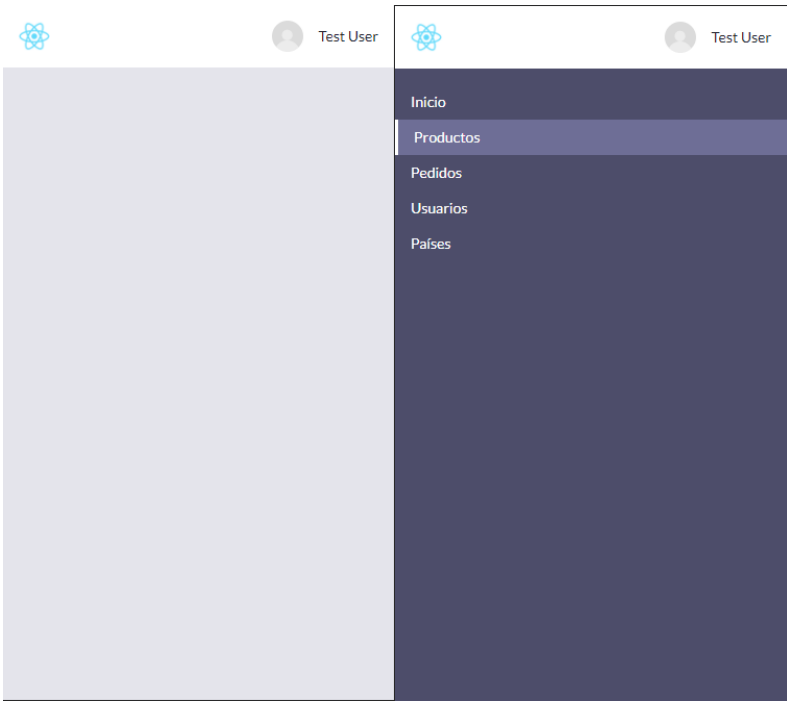

Ilustración 25: Vista Principal con menú oculto / desplegado (Móvil) – Baja Fidelidad

- **Vista de Entidades (Móvil):**

Ilustración 26: Vista Entidades (Móvil) - Baja Fidelidad

5. Implementación

5.1 Implementación Entidades Dinámicas

En primer lugar, las entidades que el usuario verá en su menú principal. Recordemos que estamos asumiendo un caso en el que, el usuario, es el usuario administrador de uno de los clientes del producto y saltando el paso en el que dicho usuario cargaría su archivo de configuración.

Es por ello por lo que debería tener un apartado en su perfil de usuario, desde el que subir a la nube un archivo en formato JSON, el cual se persistiría en la nube mediante una función Lambda, que lo guarda en un bucket de S3. Esta es una de las limitaciones del proyecto, y por lo que dicho usuario únicamente tiene un campo editable, que contiene el identificador del proyecto.

El funcionamiento es el siguiente, se crea un bucket en S3, configurado con el identificador del proyecto:

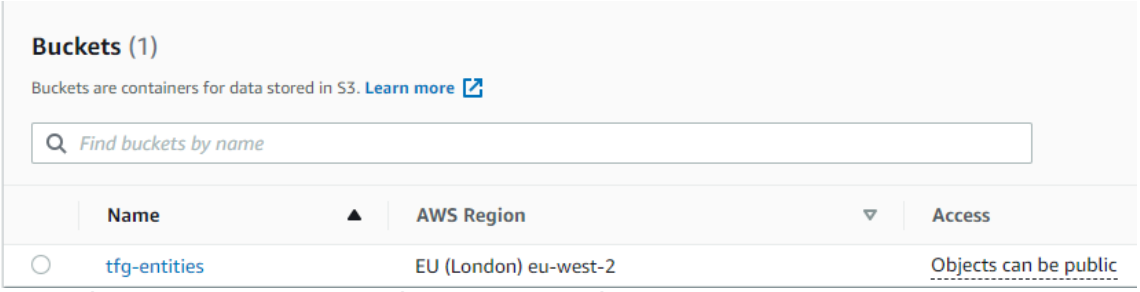

Ilustración 27: Bucket S3 - Configuración de entidades dinámicas

En el interior de dicho bucket se encuentra el archivo de configuración, en el cual se presentan las tablas a las que debe tener acceso el usuario. La información de las tablas se divide en:

- **Key:** Identificador de la tabla en la base de datos.
- **Id:** Identificador de la entidad en el frontal y en la consola de administración.
- **Name:** Nombre de la entidad en el frontal y en la consola de administración.
- **AttributeList:** Lista de los atributos que se presentan al usuario en la vista de administración a la hora de crear o editar una entidad de un tipo determinado. Estos atributos se componen de nombre y tipo. El nombre es el nombre del atributo en la base de datos y el tipo, el tipo de componente que el proyecto debe renderizar para la edición de dicho atributo.

A continuación, se muestra una entidad de ejemplo en dicho formato:

```
"NavItem": {
  "id": "nav-items",
  "name": "Navigation Item",
  "attributeList": [
    {
      "name": "id",
      "type": "text"
    },
    {
      "name": "name",
      "type": "text"
    },
    {
      "name": "link",
      "type": "text"
    },
    {
      "name": "style",
      "type": "text"
    }
  ]
},
```


Ilustración 28: Bucket S3 - Ejemplo de una entidad del archivo de configuración

Como podemos ver en la ilustración, se presentan los atributos id, name, link y style. Estos atributos de tipo texto, se presentan en el panel de administración como *inputs* de tipo texto. Sin embargo, si el tipo fuera ***editor***, en el panel de administración se renderizaría un componente del tipo **WYSIWYG o What You See Is What You Get** [44]**,** es decir, un elemento *textarea* enriquecido en el que se puede escribir HTML directamente, si así se desea.

Cabe decir, que, en un caso real, el archivo de configuración debería incluir el ARN de la cuenta del cliente. El ARN de Amazon Web Services, es un identificador único que incluye datos como el identificador único del usuario, la región en la que se localizan sus recursos, etc. De lo contrario las funciones Lambda no serían capaces de leer o escribir en dichos recursos.

5.2 Implementación de la API REST

Para comunicar la página de administración con la base de datos, se ha desarrollado una API REST. Esta API se aloja en AWS, concretamente en el servicio API Gateway, el cual se comunica con funciones Lambda.

API Gateway

Esto se encargar de exponer los *endpoints* necesarios y comunicarlos con las funciones Lambda. El objetivo de la API REST, es permitir la realización de las operaciones CRUD sobre las entidades mediante peticiones HTTP. API Gateway, también se encarga de asociar la API con un dominio propio del usuario, en este caso, toda la API se sirve bajo el subdominio *api*.

La API en API Gateway, se prepara estableciendo una serie de recursos con métodos habilitados. Estos recursos tienen una URL asociada y bajo peticiones a dichos endpoints se disparan, además, estas URLs pueden tener variables

asociadas. De este modo, un mismo endpoint puede servir para llamadas de diferentes entidades de un mismo CRUD. En este proyecto, la estructura es la siguiente:

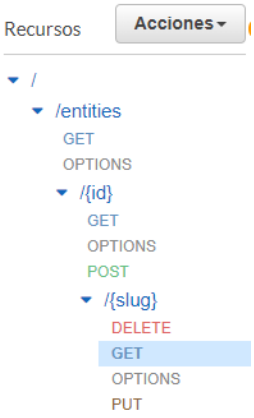

Ilustración 29: API Gateway - Estructura de recursos

Por defecto, todos los recursos tienen habilitado el método OPTIONS. Es en este método donde se configura el CORS (Cross Origin Resource Sharing), que permite a un endpoint bajo un dominio concreto, ser llamado desde otro dominio diferente.

Por otra parte, podemos ver la definición de subdominios propios, que permite parsear la URL generada aleatoriamente de los endpoints de la API, al dominio personalizado.

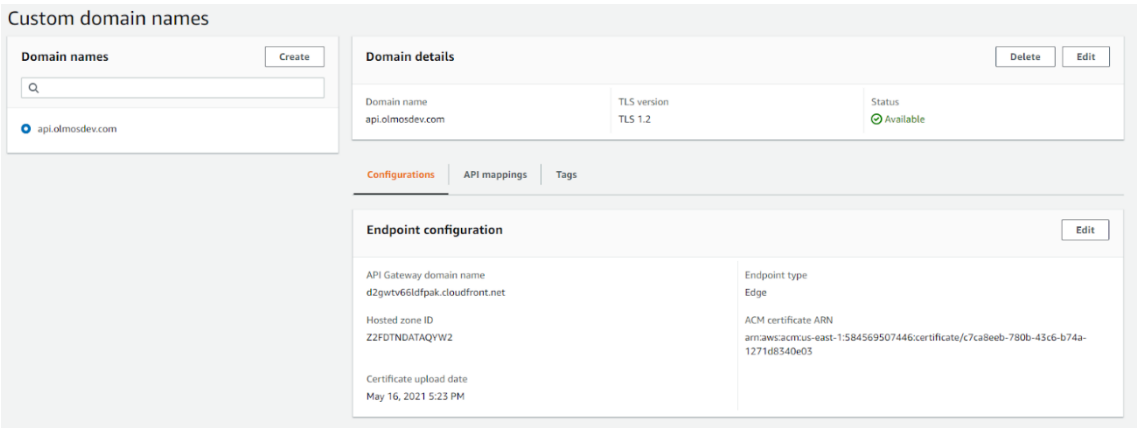

Ilustración 30: API Gateway - Dominios customizados

AWS Lambdas

Las Lambdas están escritas en JavaScript, concretamente en Node.js y se encargan de leer inicialmente el archivo de configuración del usuario y realizar la operación CRUD a continuación. Esto es para poder relacionar el id proporcionado por el usuario con las claves de las tablas reales de DynamoDB.

Además, las Lambdas deben tener configurado un rol de lectura y escritura de recursos para poder realizar estas tareas. Estas funciones, como ya se ha

comentado, pueden recibir los parámetros de entorno, los parámetros de *query* y los parámetros codificados de las peticiones.

Si vemos el detalle en la siguiente ilustración, vemos que se extraen los parámetros bucket e id. A partir de ellos se pide la información al bucket correspondiente de S3 y de ahí se extrae el nombre de la tabla que tenga el mismo id que el id recibido, gracias a este paso se leen las entidades de la tabla esperada.

```
'use strict';
const s3Controller = require('./s3Controller');
const dynamoController = require('./dynamoController');

exports.handler = async (event, context) => {
    try {
        const { bucket, id } = event;
        // Fetch the S3 Table Info
        const [ key, data ] = await s3Controller.handler(bucket, id);
        // Fetch the DynamoDB data
        const tableData = await dynamoController.handler(key);
        return { Metadata: data, ...tableData};
    } catch (error) {
        return error;
    }
};
```

Ilustración 31: Funciones Lambda - Detalle

Por otro lado, las Lambdas a su vez se subdividen en archivos que separan la lógica. La separación actual es en tres archivos:

- **Index:** Archivo principal. Se encarga de hacer llamadas a los otros dos archivos y retornar la respuesta o el error.
- **S3Controller:** Archivo encargado de la lectura de archivos del servicio S3.
- **DynamoController:** Archivo encargado de las operaciones CRUD sobre las tablas de DynamoDB.

Aunque esta es la casuística actual, una lógica mayor podría llevar a necesitar carpetas, librerías externas, otros recursos de AWS, etc.

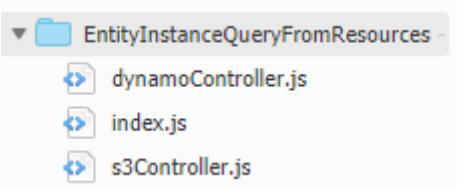

Ilustración 32: Funciones Lambda - Estructuración de Archivos

5.3 Implementación Frontend

La inicialización del proyecto relativo al Frontend, se ha realizado mediante el comando ***npx create-next-app*** [45]. Este comando, inicializa un proyecto

mediante unos pasos de configuración, con los que, entre otras cosas, se puede elegir TypeScript como lenguaje para la aplicación.

Hecho esto, se han generado una serie de carpetas, recogidas en la siguiente ilustración, bajo las cuales se estructura el código del proyecto:

- **Types:** Bajo esta carpeta se encuentran, únicamente, los tipos de las librerías externas, las cuales tienen soporte para TypeScript.

- **Components:** Bajo esta carpeta se encuentran los componentes de la aplicación. Los componentes, son aquellos archivos que es probable que sean reutilizados. Ayudan a separar la lógica y estructurar el código de una manera ordenada. En NextJS, lo común es hacer uso de los **Componentes Funcionales** [46]. Esta carpeta, a su vez, se subdivide en componentes genéricos y componentes relativos a una vista, en función de su carácter

```tsx
// Out of the box imports
import { FunctionComponent } from 'react';
// Custom hooks
import { useDropdown } from '@/hooks/useDropdown';
// FontAwesome imports
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
// Interfaces of the component
import { IDropdown } from '@/interfaces/components';
// Custom imports
import styles from '@/styles/modules/components/Dropdown.module.scss';

const Dropdown: FunctionComponent<IDropdown> = ({ options }) => {
    // Update the display of the dropdown from the custom hook
    useDropdown(`.${styles.dropdown}`);

    return (
        <ul className={styles.dropdown}>
            {options.map(({ id, value, icon, callback }) => (
                <li
                    key={`${id}-dropdown-option`}
                    className={styles.dropdown__option}
                    onClick={() => callback && callback()}
                >
                    {icon && <FontAwesomeIcon className={styles.dropdown__icon} icon={icon} />}
                    <span> {value} </span>
                </li>
            ))}
        </ul>
    );
};

export default Dropdown;
```

Ilustración 33: Ejemplo Componente Funcional

- **Helpers:** Bajo esta carpeta, se encuentran los textos estáticos y las constantes de configuración de vistas y componentes. Básicamente, son textos que podrían repetirse, por lo que idealmente deben centralizarse.

- **Hooks:** Bajo esta carpeta, se encuentran las **React Hooks** [47], definidos por el programador, propios y creados de acuerdo con una necesidad concreta. Estas Hooks, simplemente permiten manejar el estado de la aplicación de una manera mucho más sencilla que su antigua alternativa, las Clases.

- **Interfaces:** Bajo esta carpeta, se encuentran las interfaces que determinan los tipos de componentes y vistas. Estas interfaces, se aseguran de que, un componente o vista, reciban solo las propiedades esperadas, y que sean del tipo esperado.

```
// Interface for the Container component
export interface IContainer {
    children: ReactNode;
    title?: string;
    subtitle?: string;
}
```

Ilustración 34: Ejemplo Interfaz

- **Pages:** Bajo esta carpeta, se encuentran las rutas que la aplicación va a saber interpretar y los archivos base de dichas páginas. Por supuesto, no es necesario desarrollar una nueva página para cada caso si es exactamente igual que el anterior. Para esto existen las rutas dinámicas, en las que una variable se expresa entre corchetes: *[id]*. De este modo, todos los tipos de entidad o todas las instancias de una entidad, pueden cubrirse con una única página.

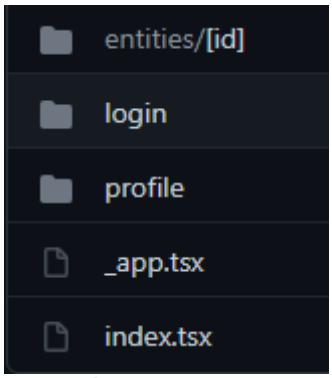

Ilustración 35: Ejemplo rutas

- **Public:** Bajo esta carpeta, se encuentran los archivos públicos de la plataforma. Esto son, imágenes, metainformación, información relativa a los robots de SEO, etc.

- **Services:** Bajo esta carpeta, se encuentran los archivos relativos a librerías de terceros, como puede ser el servicio de Cognito o de Nookies, y archivos relativos a las peticiones asíncronas. Es también en esta capa, donde el Frontend y la librería Axios, entran en contacto.

- **Styles:** Bajo esta carpeta, se encuentran los archivos relativos a los estilos de la plataforma. Estos se encuentran divididos en: módulos, si hacen referencia a componentes y vistas, genéricos, si son genéricos, y los referentes a la guía de estilos, que permite configurar la aplicación rápidamente.

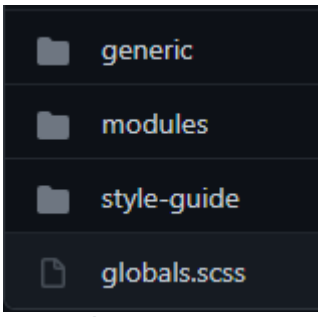

Ilustración 36: Ejemplo estilos

- **Utils:** Bajo esta carpeta, se encuentran los archivos, relativos a vistas y componentes, que contienen la lógica de estos. Esto es, con el objetivo de liberar los archivos principales, de toda la lógica innecesaria, dada la separación entre capas.

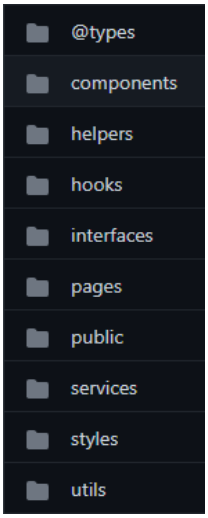

Ilustración 37: Estructura de carpetas del proyecto

En etapas anteriores de la memoria, se ha hablado del renderizado Server-Side. NextJS, permite realizar peticiones desde el lado del servidor, generando archivos estáticos que ya contienen la información que deben mostrar, aunque siguen siendo reactivos. Esto quiere decir, que, aunque son archivos HTML, siguen manteniendo toda la funcionalidad propia del *framework*.

Esto se consigue mediante el uso del método *getServerSideProps*:

```
export const getServerSideProps: GetServerSideProps = async (context) => {
    // Get the user's cookie based on the request
    const token = getUserCookie(context);
    const userProject = getProjectCookie(context);

    if (!token) {
        return {
            redirect: {
                destination: '/login',
                permanent: false,
            },
        };
    }

    if (!userProject) {
        return {
            redirect: {
                destination: '/profile',
                permanent: false,
            },
        };
    }

    try {
        setToken(token);
        const data = await getEntities(userProject);
        return { props: { token, data } };
    } catch (error) {
        return { notFound: true };
    }
};
```

Ilustración 38: Ejemplo Server-Side Props

Como se puede observar en la ilustración, es en este método donde: se comprueba que el usuario esté autenticado, se extrae su token, se añade ese token al contexto de Axios, se realizan las redirecciones pertinentes y donde se ejecutan las peticiones asíncronas. Y todo esto, ocurre de modo transparente al usuario, por ejecutarse en el lado del servidor.

Sin embargo, aquí hay una trampa. Si estamos hablando de ejecución sin servidor, no puede ser posible que este método se ejecute en el lado del servidor. Aquí es donde entra AWS Amplify. Este servicio de AWS permite integrar un repositorio remoto, en GitHub en este caso, elegir una rama de dicho repositorio, y automáticamente generar todo el entorno necesario para su correcto funcionamiento sin servidor, pero ejecutándose en el lado del servidor.

Este servicio básicamente genera todos los archivos que realmente pueden ser estáticos y los mueve a su *bucket* de S3 correspondiente, prepara la distribución de contenido y prepara instancias de servidores de Node.js, auto escalables y con cero configuraciones por parte del propietario. De este modo se consigue el funcionamiento sin servidor, pero ejecutándose en el lado del servidor. También permite asociar un dominio propio, configurar un subdominio, redirigir el tráfico entre versiones, integración continua, etc.

Es necesario remarcar, que, en realidad, en ningún caso existe la computación sin servidor real, es simplemente que dichos servidores son

remotos, automáticos y auto escalables, que a efectos prácticos es como si no hubiera servidores de facto.

6. Validación del Proyecto

6.1 Pruebas realizadas

Las pruebas realizadas con respecto al proyecto únicamente tienen que ver con el correcto funcionamiento de las Funciones Lambda programadas. Idealmente habría que haber implementado **Jest** [48], o alguna otra librería para el correcto testeo de los componentes de NextJS, pero por falta de tiempo se ha realizado un testeo manual unitario.

En cuanto a las Lambdas, su testeo quiere decir, tener un escenario en el que se puede probar que las funciones se comporten de acuerdo con lo esperado con los roles de permisos que se les han asignado.

Se han configurado entornos de prueba para todas y cada una de las seis Lambdas. Estas pruebas, además permiten probar que ejecutan su tarea de manera satisfactoria. Mediante estas pruebas, también se comprueba que las Lambdas, reciben o no sus parámetros de la manera esperada.

Las pruebas pueden incluir desde un único parámetro para comprobar una lectura:

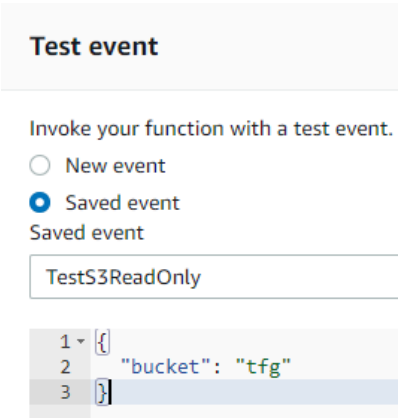

Ilustración 39: Funciones Lambda - Pruebas

O hasta tres o cuatro, en los casos en los que se comprueba una operación CRUD a nivel de instancia de entidad:

Ilustración 40: Funciones Lambda - Pruebas II

El objetivo al finalizar todas las pruebas es obtener los resultados esperados. Si examinásemos los detalles de la ejecución de la siguiente captura, podríamos ver los resultados de retorno al terminar el test.

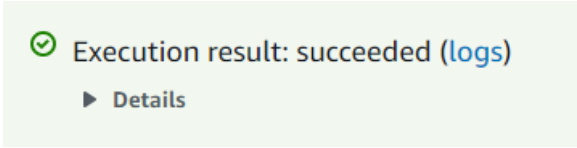

Ilustración 41: Funciones Lambda - Resultados Prueba

6.2 Resultado Final

Terminada la implementación del proyecto, alcanzados los objetivos y concluidas las pruebas pertinentes, se obtiene un resultado final que se presenta junto con esta memoria.

A continuación, se muestran el diseño final de la parte frontal pública. Esto no estaba recogido en los prototipos, debido a que se ha reutilizado y adaptado unos diseños más antiguos:

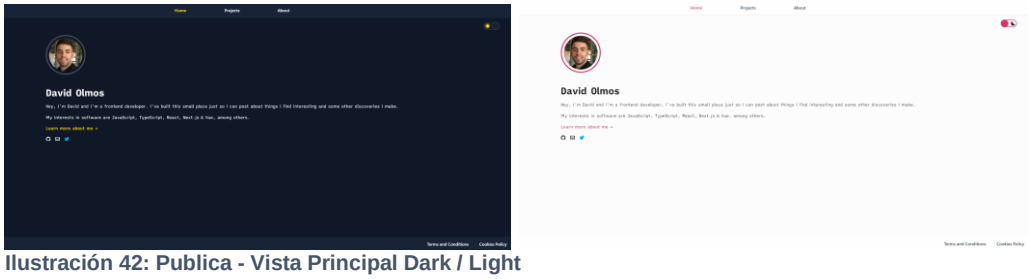

Ilustración 42: Publica - Vista Principal Dark / Light

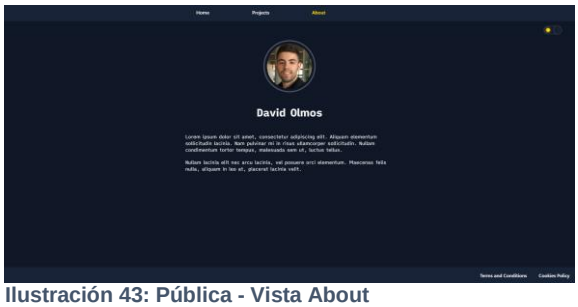

Ilustración 43: Pública - Vista About

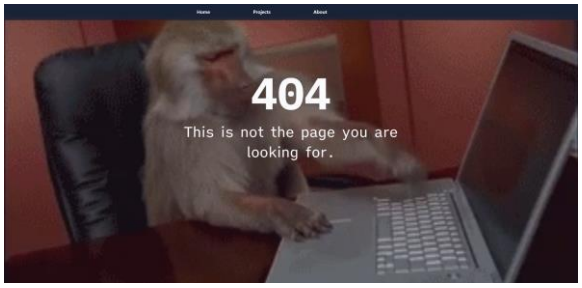

Ilustración 44: Publica - Vista 404

Ilustración 45: Publica - Vista Bloque Legal

En cuanto a los estilos del panel privado de administración, aunque los colores y fuentes son diferentes de los presentados en el prototipo de baja fidelidad, el resultado no varía mucho de la estructura presentada. En cuanto a los tipos de entidades, como ejemplo se mostrarán las vistas referentes al tipo de Bloques Legales:

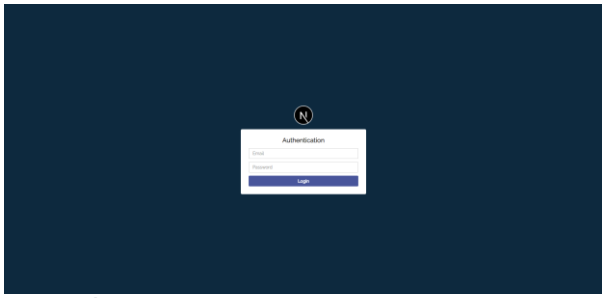

Ilustración 46: CRM – Login

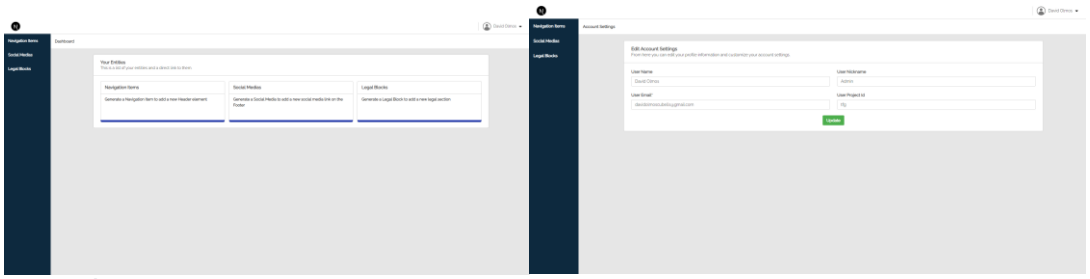

Ilustración 47: CRM - Vistas principal y perfil

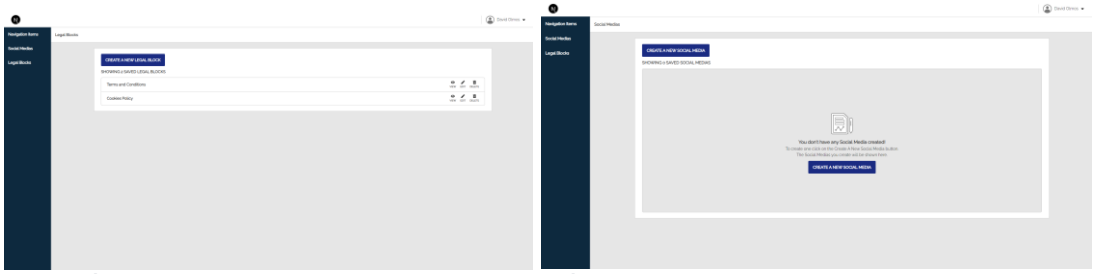

Ilustración 48: CRM - Vistas Lista de Entidad / Lista Vacía

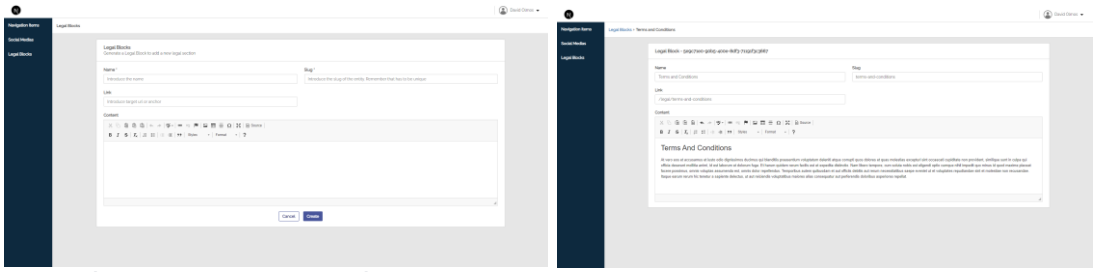

Ilustración 49: CRM - Vistas Creación Entidad y Entidad

7. Conclusiones

7.1 Conclusiones obtenidas

Llegados a este punto, es necesario repasar el trabajo realizado y valorarlo objetivamente. Aún con los retrasos sufridos, se ha entregado un producto completamente funcional y que desde luego no está cerrado a modificaciones y mejoras en el futuro.

Al comienzo del desarrollo del proyecto, se realizó una planificación temporal, la cual no se ha cumplido en algunos términos. Sin embargo, se han cumplido la mayoría o todos los objetivos que se planteaban al comienzo del proyecto.

El principal objetivo que no se ha podido cumplir, es la responsividad de la plataforma, esta, actualmente, no se muestra como debería en dispositivos de baja resolución como móviles o tabletas. Esto se ha debido a los retrasos que causó la integración de la librería y tecnología Cognito, una losa en la planificación inicial.

Esta librería presenta una serie de problemas en su versión de JavaScript, especialmente al hacer uso de TypeScript, pues las interfaces de tipos que presenta no están bien construidas. Esto ha repercutido en tener que invertir mucho tiempo en crear métodos adaptadores, en una capa de servicios, que permita usar las funciones que provee.

Por otro lado, actualmente se recomienda su uso mayoritariamente con cookies, hecho que ha provocado la pérdida de los datos del usuario en el lado del servidor, por lo que ha sido necesario integrar otra librería de terceros, Nookies. Esta librería ha permitido solucionar el problema de Cognito en el lado del servidor, persistiendo una cookie invisible en el lado del cliente, que contiene el JWT Token del usuario.

Sumado a esto, la creación de las vistas de entidades e implementación de sus peticiones CRUD, ocupó un tiempo ligeramente superior.

En cualquier caso, habiendo pasado el tiempo, la planificación claramente era ligeramente ostentosa. Si bien la cantidad de tareas que puede realizar la aplicación no es excesiva, la complejidad en ciertos puntos si lo ha sido. La realización del proyecto a requerido incontables horas de investigación sobre las tecnologías, al fin y al cabo, no había una referencia que seguir para implementarlas todas juntas.

Otro cambio sustancial, es la diferencia de aspecto entre los prototipos y el producto final. Aunque era igualmente funcional, la combinación de colores y como se mostraban los elementos, no terminaba de convencer y por tanto se rehízo.

Por tanto, aún habiendo conseguido casi todo lo propuesto, se habrían necesitado muchas horas más, para implementar muchas otras funcionalidades,

incluir test unitarios en los componentes, implementar las versiones móvil y Tablet, etc.

7.2 Líneas de futuro

Como ya se ha comentado, este proyecto no ha terminado, tiene muchas líneas de futuro sobre las que continuar avanzando y que habría sido una opción estupenda haber podido incluir en la entrega del proyecto. Aunque el producto no tiene referencias a nivel arquitectónico, las alternativas sí que existen en el mercado y añaden muchas funcionalidades extra. Por tanto, entre otras, las líneas de futuro son las siguientes:

- **Registro de nuevos usuarios**: O grupos de usuarios. Esta funcionalidad es básica si el objetivo es la comercialización del producto.

- **Gestión de permisos de usuarios de un grupo:** Dados los grupos de usuarios, alguno de ellos debe estar por encima de los otros. Este usuario "administrador", debe poder añadir nuevos usuarios al grupo, eliminarlos, darlos de baja, etc.

- **Permitir la carga del archivo de configuración:** Este usuario de tipo administrador, también debería ser capaz de subir el archivo de configuración mediante un *endpoint* de la API.

- **Creación de nuevas vistas desde la plataforma:** Conseguir, del mismo modo que ocurre con los bloques legales, crear vistas de manera dinámica desde la plataforma.

- **Creación de nuevos tipos de entidad:** Conseguir, del mismo modo que se crean nuevas instancias de un tipo de entidad, crear nuevos tipos de entidad y extender el archivo de configuración.

- **Creación de reportes:** Habilitar una descarga de reportes que indique, el SEO de la página web del cliente, o sus costes y facturas, estadísticas, etc.

8. Glosario

API: Interfaz de programación de aplicaciones. Es una interfaz cuyo objetivo es especificar como, diferentes componentes de programas informáticos deberían interaccionar entre sí.

CRUD: De sus siglas en inglés de crear, leer, actualizar y borrar. Hace referencia a las funciones básicas que se ejecutan sobre los datos de una base de datos.

Framework: O marco de trabajo, es un conjunto estandarizado de conceptos, técnicas, prácticas y criterios, que se ponen en común para enfocar la resolución de una problemática de una manera en concreto.

HTTP: O protocolo de transferencia de hipertexto, establece el protocolo para el intercambio de documentos de hipertexto y multimedia. Dispone de una variante, cifrada mediante la tecnología SSL, llamada HTTPS.

9. Bibliografía

[1] *https://wordpress.com/es/,* 2021.
[2] *https://www.prestashop.com/es,* 2021.
[3] *https://es.wix.com/,* 2021.
[4] *https://sonata-project.org/,* 2021.
[5] *https://symfony.com/,* 2021.
[6] *https://es.wikipedia.org/wiki/Kanban_(desarrollo),* 2021.
[7] *https://github.com/features/project-management/,* 2021.
[8] *https://github.com/,* 2021.
[9] *https://aws.amazon.com/es/cognito/,* 2021.
[10] *https://www.npmjs.com/,* 2021.
[11] *https://www.npmjs.com/package/nookies,* 2021.
[12] *https://es.wikipedia.org/wiki/Posicionamiento_en_buscadores,* 2021.
[13] *https://es.wikipedia.org/wiki/Customer_relationship_management,* 2021.
[14] *https://symfony.com/doc/current/bundles/SonataAdminBundle/index.html,* 2021.
[15] *https://www.docker.com/,* 2021.
[16] *https://es.wikipedia.org/wiki/JavaScript,* 2021.
[17] *https://vuejs.org/,* 2021.
[18] *https://es.reactjs.org/,* 2021.
[19] *https://angular.io/,* 2021.
[20] *https://es.wikipedia.org/wiki/Single-page_application,* 2021.
[21] *https://www.typescriptlang.org/,* 2021.
[22] *https://es.wikipedia.org/wiki/Interfaz_de_l%C3%ADnea_de_comandos,* 2021.
[23] *https://nextjs.org/,* 2021.
[24] *https://aws.amazon.com/es/lambda/,* 2021.
[25] *https://nodejs.org/es/,* 2021.
[26] *https://es.wikipedia.org/wiki/Intercambio_de_recursos_de_origen_cruzado,* 2021.
[27] *https://sass-lang.com/,* 2021.
[28] *https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada,* 2021.
[29] *https://axios-http.com/,* 2021.
[30] *https://www.json.org/json-en.html,* 2021.
[31] *https://jwt.io/,* 2021.
[32] *https://aws.amazon.com/es/what-is-aws/,* 2021.
[33] *https://aws.amazon.com/es/s3/,* 2021.
[34] *https://aws.amazon.com/es/dynamodb/,* 2021.
[35] *https://aws.amazon.com/es/cloudfront/,* 2021.
[36] *https://aws.amazon.com/es/api-gateway/,* 2021.
[37] *https://aws.amazon.com/es/amplify/,* 2021.
[38] *https://aws.amazon.com/es/serverless/sam/,* 2021.
[39] *https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/UsingBucket.html,* 2021.
[40] *https://aws.amazon.com/es/route53/,* 2021.
[41] *https://aws.amazon.com/es/certificate-manager/,* 2021.
[42] *https://es.wikipedia.org/wiki/Query_string,* 2021.
[43] *https://fontawesome.com/,* 2021.
[44] *https://es.wikipedia.org/wiki/WYSIWYG,* 2021.
[45] *https://nextjs.org/docs/api-reference/create-next-app,* 2021.

[46] *https://es.reactjs.org/docs/components-and-props.html,* 2021.

[47] *https://es.reactjs.org/docs/hooks-reference.html#gatsby-focus-wrapper*, 2021.

[48] *https://jestjs.io/es-ES/*, 2021.