ResearcherZone

Memoria de Proyecto Final de Máster

Desarrollo de Sitios y Aplicaciones Web

Autor: Carlos Martínez Gómez

Profesor: César Pablo Córcoles Briongos

Profesor: Miguel Calvo Matalobos

7 de junio de 2021

Dedico este trabajo a mis padres por su apoyo y paciencia durante estos últimos años; y a mis yo del futuro, para que no se olviden de esto ni de mí.

Resumen

La producción científica de una persona investigadora es la que marca su experiencia y le sirve para darse a conocer y conseguir nuevas oportunidades de trabajo; así pues, resulta importante reflejarlo en su CV. Existen servicios muy conocidos que realizan está acción pero que a veces se pueden ver limitados en el tipo de información que pueden albergar, así como en su forma.

ResearcherZone nace con la idea de ofrecer un espacio a investigadores que no poseen todavía un lugar donde tener reflejada su producción científica para que publiquen su trabajo teniendo un poco más de control sobre el tipo de información que deciden mostrar.

Para llevar a cabo el proyecto se desarrolla una aplicación escrita en Typescript/Javascript ejecutada con Deno, usando React.js en el *frontend*, con base de datos en MongoDB Atlas y publicada en forma de contenedor Docker en Cloud Run (Google Cloud Platform).

Palabras clave: investigador, cv, currículum, producción científica

Abstract

The experience of a researcher is equivalent to the amount of documents he has published and that helps him to become known and find new job opportunities; for that reason, it is important to have a good CV. There are services out there that perform this action, but sometimes can be limited in the type and form of the information that they can contain.

The idea of ResearcherZone is to offer a space to researchers who do not have a place for sharing their scientific production so they can publish their work, having a bit more control over the type of information they decide to show.

For making the project, I develop an application written in Typescript/Javascript executed with Deno, using React.js on the frontend, with a database in MongoDB Atlas and published as a Docker container in Cloud Run (Google Cloud Platform).

Keywords: researcher, cv, curriculum, scientific production

Sumario

Figuras y tablas

índice de figuras

índice de tablas

Contexto y justificación del proyecto

En el ámbito académico, la experiencia de un investigador queda reflejada en la cantidad de producción científica que ha generado a lo largo de su carrera. Esta cantidad es la que distingue a un investigador recién iniciado de otro más reconocido. Aunque no solo eso, sino que en ocasiones esta cantidad se traduce en la adopción de un cargo más elevado en la empresa o entidad de la que forma parte, o puede suponer la diferencia entre optar o no a una ayuda económica para financiar un nuevo proyecto.

En definitiva, cualquier persona dedicada a la investigación deberá, necesariamente, mantener la gestión de su propia producción en algún momento y durante toda su carrera.

Es bastante común que esta tarea no la lleve a cabo el mismo investigador y que, siempre y cuando disponga de los recursos –humanos–, se encargue el centro del que forma parte. Así es como esta responsabilidad se acaba delegando a la biblioteca o centro de documentación de la propia entidad, lo cual agilita el proceso.

Pero ¿qué pasa cuando el centro no dispone de los recursos necesarios o el usuario no puede acceder a dichos recursos? En estos casos, existen herramientas conocidas como “ResearchGate” o “Google Scholar”, entre otras. Sin embargo, estas herramientas a veces pueden parecer un tanto imponentes, mostrando información que no interesa, y quedar limitadas en cuanto a la personalización de la vista, impidiendo al usuario añadir información que pudiera considerar más relevante y deshacerse de la que considere prescindible.

Es de esta carencia de donde surge ResearcherZone. Su finalidad no es crear un sustituto de las herramientas anteriormente mencionadas, sino proporcionar una interfaz personalizable donde investigadores puedan crear y organizar su propia ficha personal: publicaciones destacadas, área de especialidad, biografía, otros enlaces, etc.

Objetivos

- Facilitar el proceso de creación de perfiles de investigadores.
- Ofrecer una interfaz para compartir los perfiles fácilmente.
- Permitir categorizar los documentos entrados en el CV.
- Facilitar la entrada documentos académicos.
- Dar margen a la personalización.

El objetivo principal del proyecto es proporcionar una herramienta donde investigadores puedan crear su perfil personal. Para ello se desarrollará una aplicación que permita a los usuarios registrarse, incorporar información, así como artículos, capítulos, libros, etc., y ofrecer la posibilidad de compartirlo.

Además los usuarios deberían ser capaces de modificar estos datos fácilmente, ordenar los resultados y añadir cualquier tipo de información relevante.

Planificación

Se ha dividido el proyecto en cuatro partes, donde cada parte corresponde a cada una de las entregas de la asignatura.

Planteamiento	Entrega PEC 1: 3 de marzo
Diseño	Entrega PEC 2: 31 de marzo
Inicio del desarrollo	Entrega PEC 3: 9 de abril
Final del desarrollo	Entrega PEC 4: 7 de junio

Tabla 1. Entregas del proyecto

A continuación, se proporciona el cronograma del proyecto, también disponible en los anexos.

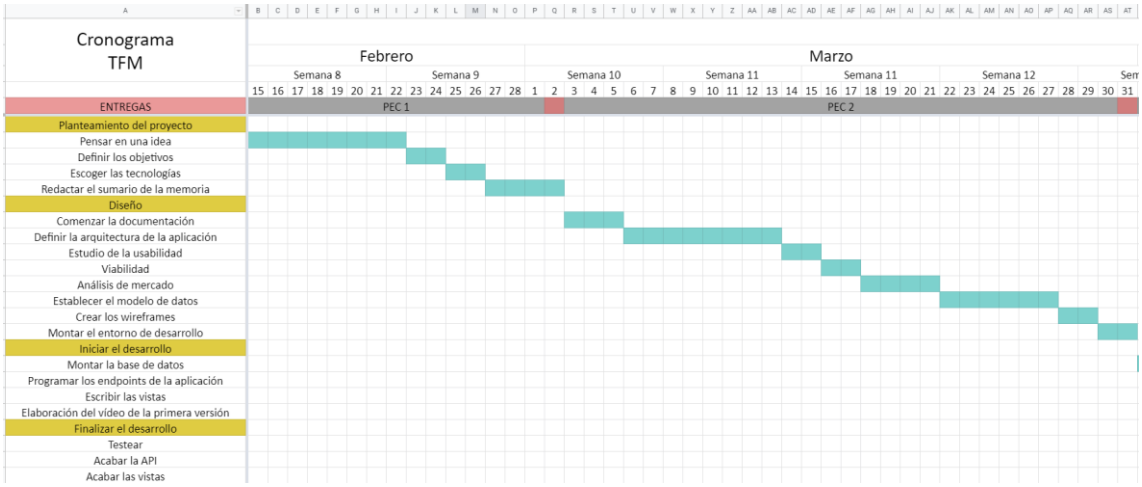

Figura 1: Cronograma del proyecto. PEC 1 y PEC 2

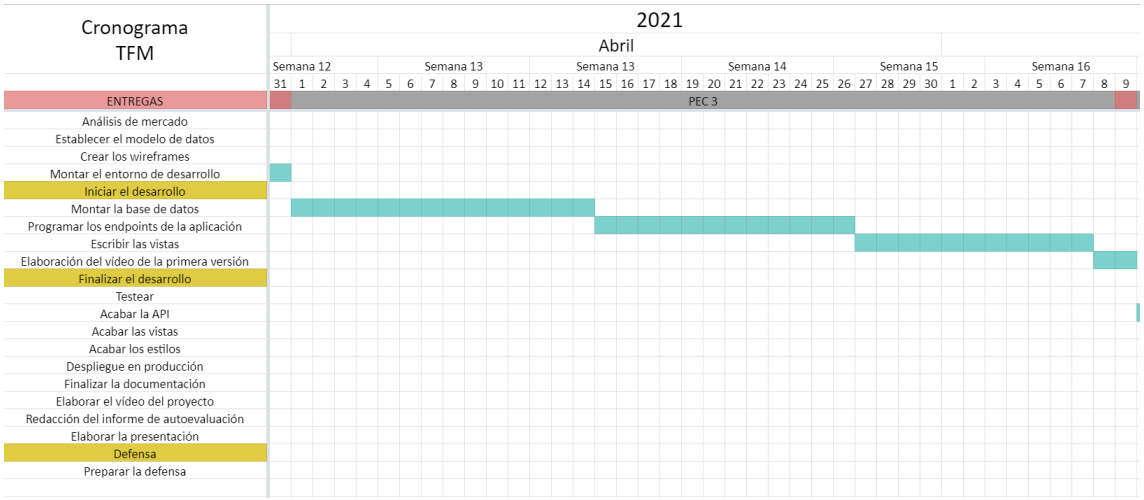

Figura 2: Cronograma del proyecto. PEC 3

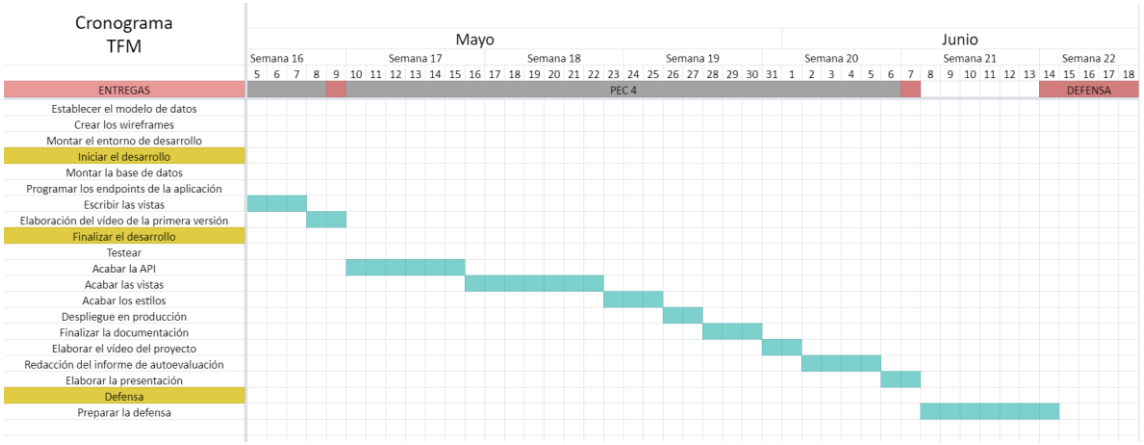

Figura 3: Cronograma del proyecto. PEC 4

Metodología seguida

La intención original de implementar una metodología para llevar a cabo el proyecto fue recurrir a una metodología Agile, como la popular *SCRUM* o *Extreme Programming*. Esta última se descartó porque en el proyecto faltaba la figura del cliente como parte activa del desarrollo; y tampoco se decidió implementar *SCRUM* porque esta metodología solamente tendría sentido en entornos de trabajo colaborativo no unipersonales donde participaran dos o tres personas como mínimo -asumiendo ellos mismos los roles de *Product Owner* y *Scrum Master*.

También se pensó en aplicar el método Kanban y clasificar las tareas de la planificación en tarjetas. Sin embargo, se llegó a la conclusión que hacerlo de esta manera no aportaba ningún beneficio para el desarrollo, puesto que las tareas ya estaban definidas en el tiempo y no se requería de ningún tipo de ayuda visual adicional al diagrama de la planificación que ya se había elaborado.

Por lo tanto, para llevar a cabo este proyecto de desarrollo, y de forma un tanto pesarosa y excepcional, se decide implementar una metodología de cascada que se define a continuación:

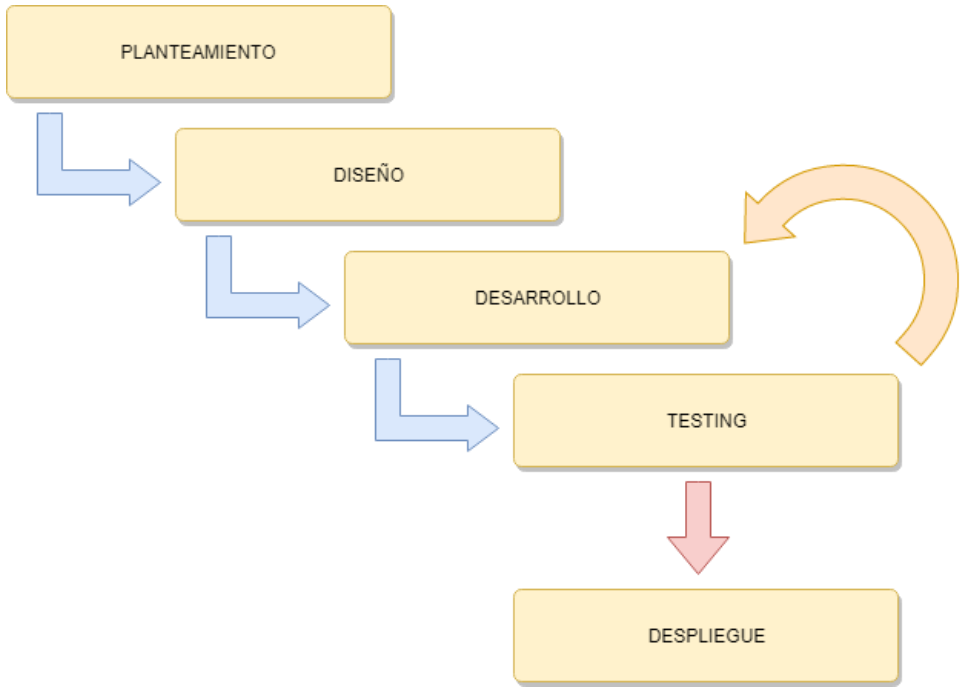

Figura 4: Diagrama en cascada

Las primeras fases de Planteamiento y Diseño corresponden a la entrega de las dos primeras prácticas. Después, se inicia el desarrollo y, juntamente con las pruebas, se realiza una segunda iteración del desarrollo. Finalmente, de cara a la última entrega del proyecto, se ejecuta un el despliegue en producción, donde entra en estado de mantenimiento.

Diseño

Funcionalidades

Las funcionalidades de la aplicación corresponden a los objetivos del proyecto y se listan a continuación:

- Añadir, quitar o editar información personal.
- Manipular el orden de los elementos.
- Añadir, quitar o editar categorías.
- Entrar información personalizada mediante un editor de texto.
- Añadir, quitar o editar publicaciones.
- Compartir el perfil.

Arquitectura

La arquitectura de la aplicación responde a un patrón Modelo – Vista – Controlador.

En la parte de la vista, se dispone de una interfaz de usuario reactiva construida con React.js que se encargará de realizar las peticiones al controlador y renderizar las respuestas en tiempo real.

El controlador está formado por las rutas, que representan los *endpoints* de la API: los métodos que modifican la petición del frontend, realizan la consulta contra la base de datos y devuelven la información al frontend. Estos métodos se encuentran escritos en Typescript y el entorno de ejecución que los mantiene es Deno. Se ha escogido este entorno de ejecución porque soporta Typescript y sintaxis ES6 por defecto –entre otros beneficios– lo cual evita la necesidad de preparar configuraciones de transpilación e instalación de dependencias y, por tanto, se traduce en un ahorro de tiempo.

Finalmente, el modelo consiste en una base de datos NoSQL basada en MongoDB. Se ha escogido este tipo de base de datos debido a su simplicidad, rapidez de instalación y puesta en marcha. Al tratarse de una aplicación pequeña y con poca previsión de mantenimiento y expansión, la elección entre SQL y NoSQL no supone una consideración tan importante en cuanto a su rendimiento y/o escalabilidad, sino en el tiempo que se dispone para desarrollarla.

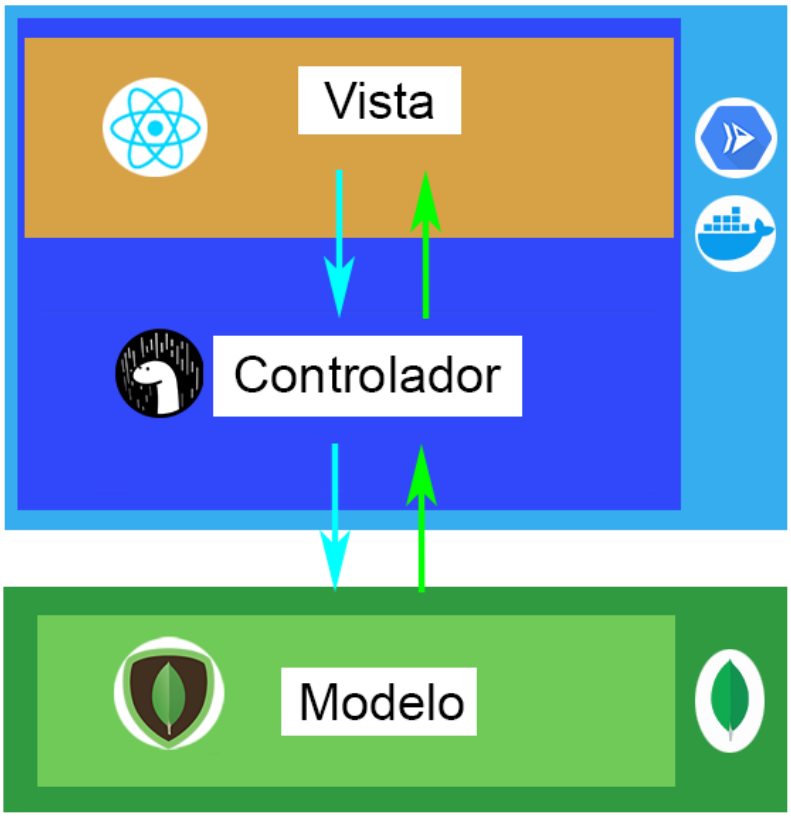

Figura 5: Arquitectura de la aplicación

La vista y el controlador forman una imagen de Docker, y esta se ejecuta en *Cloud Run*. Por otra parte, el modelo consiste en un clúster de *MongoDB Atlas*. Por lo tanto, se trata de una arquitectura basada en el *Cloud*.

Modelo de datos

El modelo de datos se divide en dos colecciones: *users* y *documents*. La primera colección es la más simple y uniforme, es decir, su contenido siempre es el mismo para todos los documentos de la colección:

```
class User {
    _id?: string;
    name: string;
    surname: string = '';
    email: string;
    pwd: string;
    profile_slot_1: string = '';
    profile_slot_2: string = '';
    categories: Array<{id: number, category: string, category_order: numb
er, visible: boolean}> = [];
    image: any = '';
    optional_image: any = '';
    social_media: Array<{name: string, url: string}> = [];
    verified: boolean;
}
```

Figura 6: Modelo del usuario

Por otra parte, la colección *documents* contiene todos los recursos que un usuario puede entrar en su perfil. Parte de una interfaz *IProfileDocument*, que implementa los demás recursos.

```
interface IProfileDocument {
    _id?: string;
    type: string;
    can_be_cited: boolean;
    category: number;
    order: number;
    user: string;
}
```

Figura 7: Interfaz de los documentos

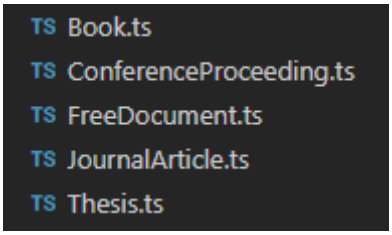

Figura 8: Tipos de documentos principales

El modelo *Book* contiene una clase para los libros y otra para los capítulos de libros, que extienden de *Book*.

```
export class Book implements IProfileDocument {
    order: number;
    _id?: string;
    category: number;
    user: string;
    type: string = 'book';
    can_be_cited: boolean;
    volume: number;
    author: {name: string, surname: string};
    title: string;
    subtitle: string;
    edition: string;
    publication_place: string;
    publisher: string;
    publication_year: number;
    doi: string;
}

export class BookChapter extends Book {
    editors: {name: string, surname: string};
    number: number;
    start_page: number;
    end_page: number;
    chapter_title: string;
    coordinator: string;
}
```

Figura 9: Modelos Book y BookChapter

El modelo *FreeDocument* representa un tipo de recurso que se proporciona para que el usuario pueda entrar la información que desee y está formado por una única propiedad "html". En la parte de la vista, se tiene pensado proporcionar un editor HTML para facilitar la inclusión de dichos datos y donde el usuario podrá ver en tiempo real cómo va quedando el contenido.

```
class FreeDocument implements IProfileDocument {
    order: number;
    _id?: string;
    type: string;
    can_be_cited: boolean;
    category: number;
    user: string;
    html: string;
}
```

Figura 10: Modelo FreeDocument

Wireframes

A continuación se presenta la forma que se pretende que tenga la interfaz. Esta consta de dos partes dinámicas y una estática. La parte estática corresponde al pie de página, que siempre será el mismo y no se ha incluido en los modelos.

Las partes dinámicas son las que siguen:

- Cabecera: componente ubicado en la parte superior de la pantalla que representa la información del usuario activo en la plataforma o del perfil que se esté visualizando.

- Cuerpo: es la sección central de la página. Corresponde al componente que muestra la información de los documentos del usuario.

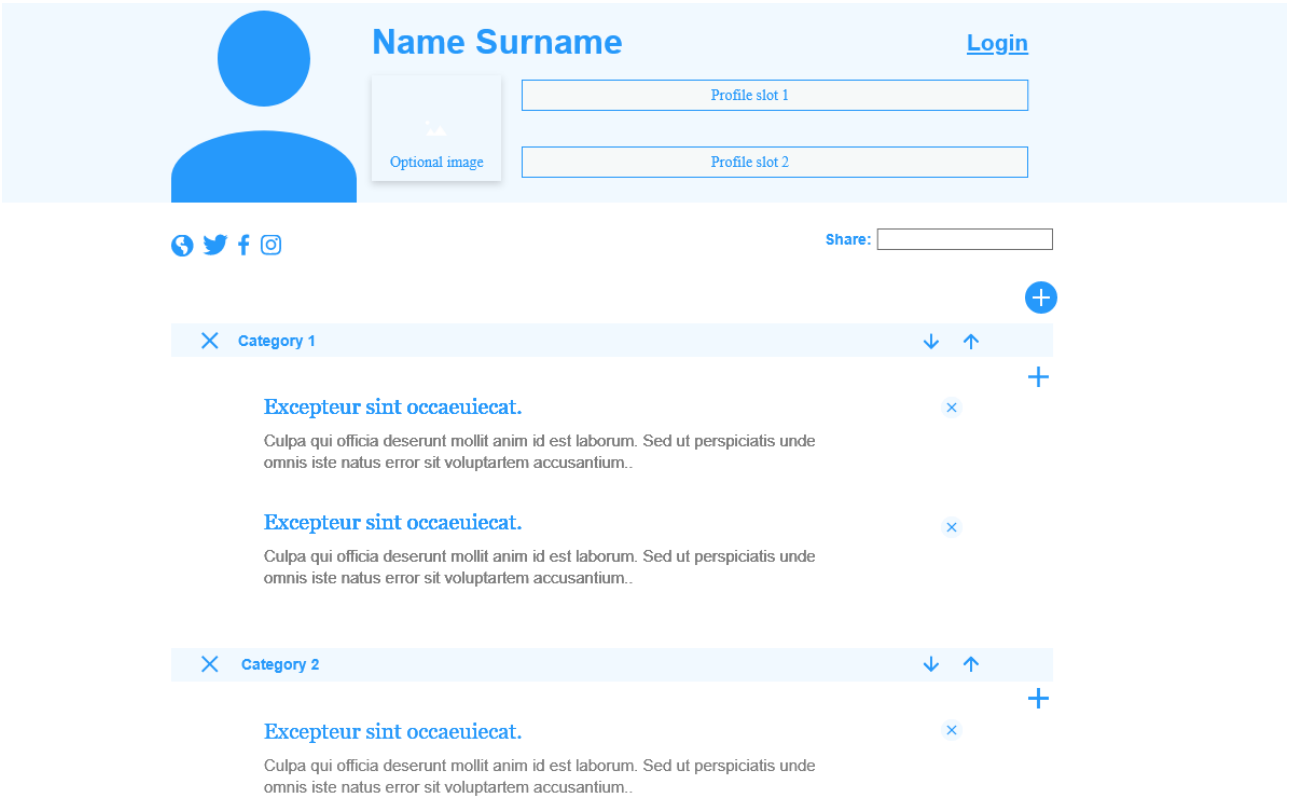

Figura 11: Diseño desktop

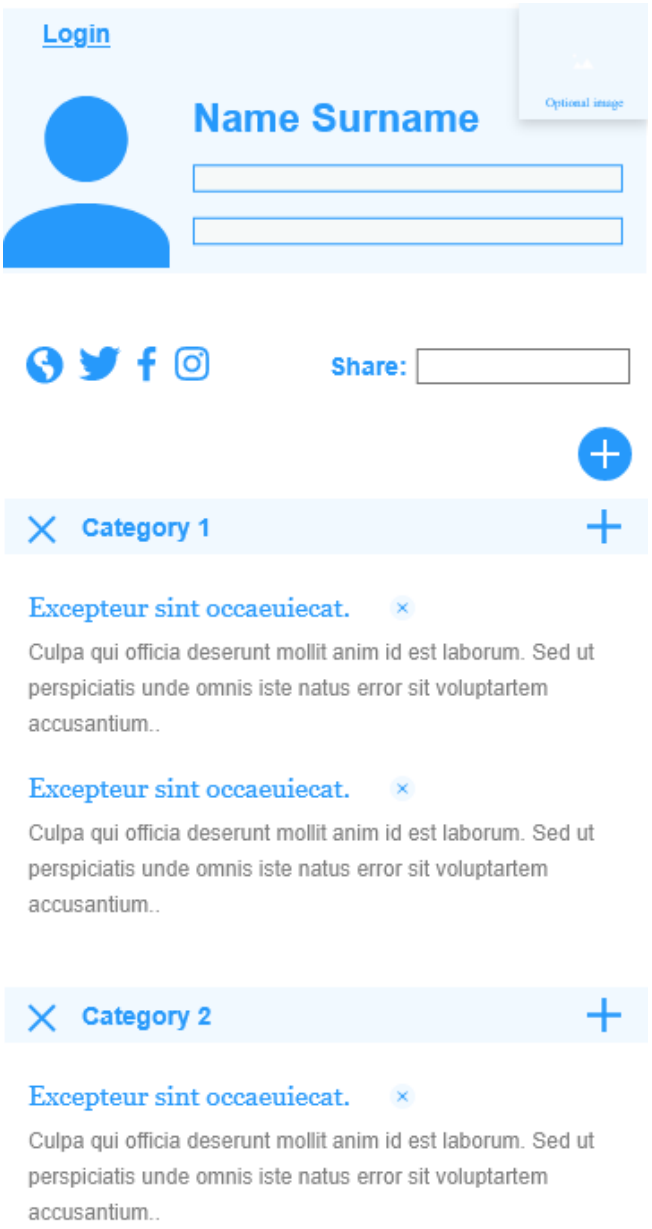

Figura 12: Diseño mobile

Análisis de mercado

La idea de ResearcherZone surge de una carencia detectada en algunas plataformas de difusión de la investigación que permiten a los usuarios establecer un perfil personal con su información y producción.

En este apartado se pretende realizar un análisis de cuáles son estas plataformas, inspeccionando un poco las funcionalidades que ofrecen con el fin de deducir dicha carencia.

ResearchGate

ResearchGate es una plataforma orientada a profesionales, estudiantes o interesados en el sector de la ciencia y la investigación con el objetivo de conectar a todas estas personas y difundir el conocimiento. Actualmente posee más de 20 millones de usuarios.

En esta plataforma, los usuarios pueden crearse un perfil, que a su vez cuelga de alguna institución de la que forma parte. En dicho perfil, el usuario es capaz de subir sus publicaciones y destacar las que desee, añadir información básica y crear una red de contactos. Sin embargo, a pesar de que posea todas estas funcionalidades, el tipo de información que se puede publicar está limitada a las categorías y la nomenclatura que proporciona la interfaz.

Scopus

Scopus es una base de datos de citaciones y resúmenes de publicaciones científicas que a su vez ofrece un perfil a los autores de dichas publicaciones.

Al utilizar los datos de su propia base de datos, los perfiles de Scopus ofrecen información bastante precisa sobre el número de citaciones que posee dicho, así como en listar sus publicaciones y co-autores.

Sin embargo, de nuevo esta interfaz se ve limitada y a las categorías de la interfaz y la estructura de la información no permite ser modificada.

Google Scholar

Google Scholar es un servicio “gratuito” de Google que sirve como buscador de literatura científica y tiene la peculiaridad de ser capaz de recuperar muchos artículos donde se destaque a un autor porque, además de utilizar el motor de Google, realiza la búsqueda en la mayoría de los repositorios más famosos.

Permite crear un perfil de forma muy sencilla, donde se recuperan los artículos donde el usuario es autor y una lista de contactos, junto con los datos básicos de contacto y algunas estadísticas.

De nuevo, el perfil queda limitado a las categorías y estructura de la interfaz.

ORCID

ORCID es una organización cuya misión es ofrecer una identificación única y confiable a los investigadores.

También ofrece un perfil a los usuarios, donde se puede entrar información sobre publicaciones, trabajos, educación, etc. Esta vez, la interfaz permite ordenar las categorías, estas siguen estando limitadas por la interfaz.

En definitiva, aunque todos estos servicios ofrezcan una muy buena forma de crear y compartir perfiles de investigadores, queda al descubierto la carencia de la personalización de la interfaz o las categorías, el tipo de carencia que se pretende cubrir en este proyecto.

Usabilidad

La idea sobre el desarrollo de la interfaz pretende que esta sea lo más simple e intuitiva posible. A continuación, se detallan los patrones de usabilidad que se aplicarán con tal de llevar esto a cabo.

En primer lugar, la interfaz pretende basarse en el patrón *mobile first*. El *layout* de la aplicación estará diseñado de tal forma que priorice una visualización para dispositivos pequeños y que de cara a los dispositivos más grandes los cambios sean mínimos y muy localizados. De esta forma se ofrece una mayor consistencia al diseño y previene posibles confusiones a los usuarios.

Además, se pretende reducir la navegación al mínimo. Esto se quiere conseguir haciendo que la interfaz ofrezca toda la información necesaria en la misma pantalla. La aplicación permite gestionar un CV, editar la información de perfil, compartir el perfil y añadir, editar y quitar documentos. Los espacios asignados para cada una de estas funcionalidades no cambiarán de lugar y serán destacados por la interfaz con tal de que el usuario los tenga presentes desde el primer momento.

También se pretende simplificar al usuario la creación de documentos personalizados incorporando un editor WYSIWYG (What You See Is What You Get).

Con tal de facilitar la edición de la información personal del usuario, se proporciona la posibilidad de poder editar dichos campos haciendo un solo clic en los mismos. Cuando la información ha sido editada correctamente, se ofrece *feedback* al usuario haciendo que dichos campos cambien su color al verde brevemente.

Por último, la iconografía y los colores de los botones se presenta de forma clara para cada funcionalidad. En la misma línea, el usuario será preguntado si desea proceder con una determinada acción cuando esta sea crítica (p. ej.: borrar un documento o cancelar la edición de un documento largo).

Viabilidad

En este apartado se realizará una previsión sobre el coste que supondría el desarrollo y el mantenimiento de la aplicación.

El coste de la aplicación se calculará a partir del cronograma establecido en el apartado de la planificación.

Así pues:

- se deduce un coste de 35€ la hora.
- con 8 horas de trabajo al día.
- y suprimiendo aquellas tareas no aplicables a un contexto de trabajo normal.

Diseño	**7280€**
Definir la arquitectura de la aplicación	2240€
Estudio de la usabilidad	560€
Viabilidad	560€
Análisis de mercado	1120€
Establecer el modelo de datos	1680€
Crear los wireframes	560€
Montar el entorno de desarrollo	560€
Iniciar el desarrollo	**10360€**
Montar la base de datos	2800€
Programar los endpoints de la aplicación	2800€
Escribir las vistas	3080€
Testear	1680€
Finalizar el desarrollo	**5040€**
Acabar la API	1680€
Acabar las vistas	1960€
Acabar los estilos	840€
Despliegue en producción	560€
Precio total del desarrollo	**22680€**

Tabla 2: Coste del desarrollo

El coste del mantenimiento de la aplicación queda sujeto a los precios de los servicios de hosting ofrecidos por los proveedores con los que se pretende integrar la versión de producción.

Estos servicios son *Cloud Run* de la *Google Cloud Platform* y *MongoDB Atlas*.

Para el contexto práctico del trabajo, se pretende utilizar las cuotas gratuitas que ofrecen ambos servicios, que son suficiente para poner en marcha la aplicación. Sin embargo, en el caso de que se quisiera aumentar la potencia de computación y en beneficio al desarrollo del manual, también se propone la siguiente cuota:

Propuesta *Cloud Run* (realizada con la *Google Cloud Pricing Calculator*):	
Region	Cualquier región de EE. UU.
Número de CPUs	2
Memoria asignada	2GB
Número de peticiones concurrentes por instancia	200
Tiempo de ejecución por petición	500ms
Número de peticiones al mes	10,000,000
Número mínimo de instancias	1
Cuota mensual estimada	24,30€

Tabla 3: Coste del alojamiento en Cloud Run

Propuesta de *MongoDB Atlas*	
Backup	Respaldo y restauración de instantáneas habilitado
Storage	10GB (con posibilidad de escalado hasta 4TB)
RAM	2GB (con posibilidad de escalado hasta 768GB)
Cuota mensual estimada	48,45€ (si no se requiere escalar)

Tabla 4: Coste del alojamiento en MongoDB Atlas

Horas al mes de mantenimiento	2 (35€/h)
Mantenimiento mensual total estimado	142,75€

Tabla 5: Costes del mantenimiento

Desarrollo

Backend

Puesta en marcha

Como se ha mencionado en un apartado anterior, el *backend* del proyecto está basado en Deno como entorno de ejecución de TypeScript, y a su vez la aplicación es compilada en una imagen y ejecutada con Docker. Sin embargo, para ahorrar tiempo y simplificar el desarrollo, se pretende ejecutar la aplicación de Deno utilizando el paquete Denon (instalado de forma global en el sistema) que nos permite reiniciar el servidor cada vez que detecta cambios en el código.

En cuanto a la compilación y el empaquetado de la aplicación, Deno ya compila typescript de forma nativa y posee una herramienta para empaquetar la aplicación en un único ejecutable Javascript y es la que se estará utilizando en este proyecto.

```
{
  "$schema": "https://deno.land/x/denon@2.4.7/schema.json",
  "scripts": {
    "start": {
      "cmd": "deno bundle src/frontend/index.jsx src/static/index.js --
config ./tsconfig.json --import-map=src/import_map.json && deno run -A --
config ./tsconfig.json --import-map=src/import_map.json src/main.tsx"
    }
  },
  "watcher": {
    "skip": [
      "*/.git/*",
      "src/static/index.js"
    ]
  }
}
```

Figura 13: Configuración del paquete Denon

La estructura de carpetas que sigue el *backend* es la siguiente:

raíz – donde tenemos los archivos de configuración, variables de entorno e importaciones de módulos.

-/src – carpeta que contiene todo el código de la aplicación.

-/controllers – carpeta donde se almacenan los archivos que contienen la lógica de la aplicación: los que modifican el modelo y devuelven los datos a la vista.

-/api –middlewares de las rutas de la API de datos.

-/authentication – middlewares de las rutas de la API de autentificación.

-/security – middlewares de seguridad (ver apartado "Seguridad").

-/database – métodos que llaman al modelo.

-routes.ts – definición de las rutas de la aplicación.

-/frontend – aplicación de React. Se hará uso de la técnica *Server Side Rendering* (SSR) desde Deno, por eso es necesario incluirla en la imagen.

-/models – clases e interfaces de la aplicación que representan los objetos del modelo de datos.

-/static – contiene la aplicación de React compilada. Es necesario "rehidratar" la aplicación exponiendo los archivos de forma estática si no se quiere perder la reactividad al aplicar SSR.

-database.ts – contiene la conexión con la base de datos.

-main.ts – ejecutable principal.

Librerías

Deno soporta sintaxis de módulos de forma nativa y por tanto puede importar librerías escritas siguiendo esta sintaxis. Algunas de las librerías populares ya se han migrado para ser compatibles con Deno, aunque todavía no posee un catálogo tan extenso como su predecesor.

Por suerte, se pueden utilizar herramientas como jspm.org, skypack.dev o esm.sh para importar estos módulos.

A continuación se listan las librerías que se han decidido instalar. Estas librerías se pueden utilizar tanto en el backend como en el frontend.

- react y react-dom – las librerías propias de React.
- react-dom/server – librería utilizada para SSR.
- oak – servidor web para Deno.
- deno-mongo – controlador para conectarse a bases de datos de MongoDB.
- dotenv – librería para guardar secretos utilizada durante el desarrollo.
- sha256 – librería utilizada para encriptar las contraseñas de los usuarios.
- djwt – librería para gestionar JWTs en Deno.
- cookie – librería para manejar las cookies de una petición.
- browser-image-compression – utilidad para comprimir las imágenes.
- asserts – librería que contiene diversas funciones *assertion* para hacer tests.

Rutas

Las rutas que se han decidido implementar en la aplicación se clasifican y explican a continuación:

Rutas de autenticación

- **/login** – Identificación de usuarios. Encargada de generar el token de acceso.
- **/register** – Registro de usuarios en la base de datos.
- **/verify_account/:id** – Ruta encargada de verificar la cuenta del usuario.
- **/logout** – Hace expirar el token de acceso.

Rutas de la API de datos

- **/getuserinfo** – Recupera los datos de un usuario para mostrar en la vista. Requiere un token válido.
- **/getguestinfo** – Recupera los datos de un usuario para mostrar en la vista. No requiere autenticación. Es utilizada para rehidratar la aplicación con los datos del usuario del perfil.
- **/updateuser** – Modifica el objeto del usuario en la base de datos.
- **/updatedocument** – Modifica un documento del perfil de un usuario en la base de datos.
- **/documentadd** – Añade un documento al perfil de un usuario en la base de datos.
- **/deletedocument** – Borra un documento del perfil de un usuario en la base de datos.
- **/deletecategory** – Modifica el objeto del usuario eliminando una categoría y borra todos los documentos relacionados con esta y el usuario.

Rutas de renderizado

- **/** – Ruta base. Requiere autenticación. Renderiza la aplicación de React en el lado del servidor y la expone.
- **/user/:id** – Ruta para los visitantes de los perfiles. No requiere autenticación y mostrará la información del perfil del usuario que coincida con el identificador del segmento “/:id”. También renderiza la aplicación de React en el servidor y la expone.

Sistema de autenticación

El sistema de autentificación de la aplicación se basa en el uso de Json web tokens (JWT). Cuando el usuario se autentica, el servidor genera un token que utiliza para dar acceso a la aplicación y llamar a la API.

Dicho token se almacena en la caché del navegador y posee un tiempo de validez de una hora.

La estructura del JWT es la que sigue:

- Header
  - alg: HS512.

  - typ: JWT.

- Payload
  - iss: Objeto con el email del usuario autenticado que utiliza como identificador para llamar a las rutas de la API de datos.
  - exp: 60 * 60 (1 hora).

- Secret
  - En este caso, se utiliza la contraseña del usuario encriptada en sha256 y se le concatena el email del usuario.

Seguridad

La aplicación posee una capa adicional de seguridad en la parte del servidor para evitar que un usuario visitante pueda modificar el estado del frontend y mandar peticiones en nombre de otro usuario. Para ello, también se hace uso de JWT.

En primer lugar, la aplicación rechazará cualquier petición que se realice desde el cliente y que no posea un token válido en la cabecera de las cookies.

Cada petición que se realiza desde el cliente hacia la API debe incluir un parámetro "email" con el email del usuario que realiza la acción. Si el servidor no encuentra dicho parámetro, también rechazará la petición.

Si el usuario posee un token válido (porque ya se ha autenticado con otra cuenta), el servidor descodifica dicho token y extrae el *payload* con su email.

A continuación se ejecuta el middleware de comprobación (verifyUserCall), recupera la contraseña encriptada a partir del email e intenta validar el JWT con el token encontrado en los *headers* de la petición y la *secret* que obtiene del usuario recuperado.

```
.put('/updateuser', verifyUserCall, users_api.update)

.put('/updatedocument', verifyUserCall, documents_api.update_document)

.post('/documentadd', verifyUserCall, documents_api.addDocument)

.delete('/deletedocument', verifyUserCall, documents_api.removeDocument)

.delete('/deletecategory', verifyUserCall, documents_api.removeCategory)
```

Figura 14: Rutas sensibles de la API

Así pues, si el token no se verifica, quiere decir que el usuario que está enviando la petición no es el usuario indicado.

En el caso de que el usuario modificase la petición del frontend para encajar el parámetro "email" con su propio email de usuario, como el servidor utiliza este

parámetro para modificar la base de datos, solo modificaría los datos de su propio perfil, y no del perfil que intenta suplantar.

```
//utiliza el email como identificador para modificar el modelo
export const updateUser = async (email: any, user: User): Promise<any> =>
 await db_users.updateOne(
    { email },
    { $set: extractFields(user) }
);
```

Figura 15: Ejemplo usando el email como parámetro de control

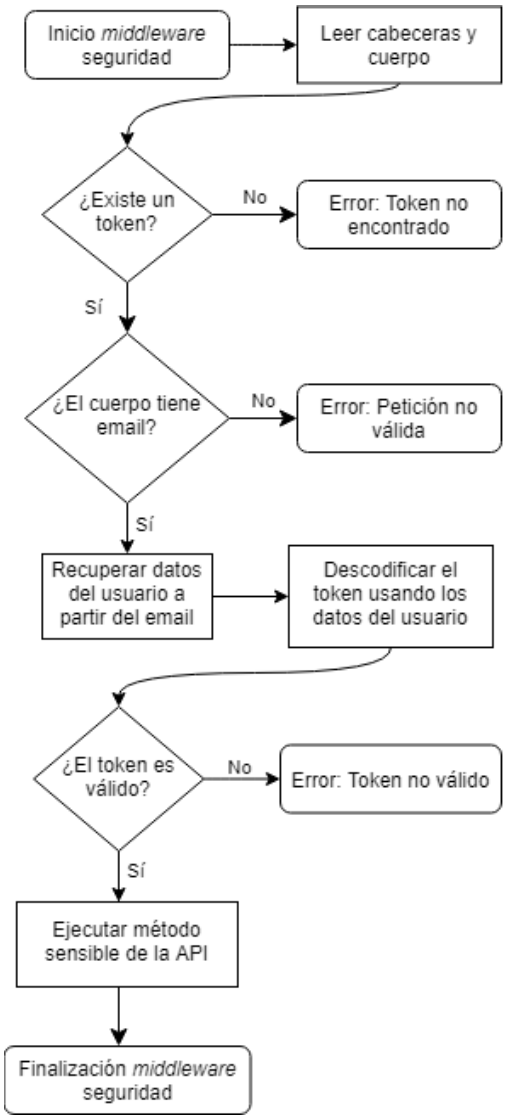

Figura 16: Flujo del middleware de seguridad

Como último sistema de seguridad a comentar en esta sección, se ha incorporado un sistema de verificación de correo electrónico para evitar que un usuario pueda registrar una cuenta que no le pertenezca. El envío de estos mensajes se realiza con la API de Mailjet, un servicio que permite enviar correos electrónicos de diversas formas, entre ellas realizando una simple petición http. Se deberá especificar la API Key de este servicio en las variables de entorno para que la aplicación funcione correctamente.

Frontend

Puesta en marcha

El proyecto de frontend de la aplicación se encuentra en la carpeta “src/frontend” y corresponde a una aplicación de React.

El frontend está escrito en Javascript/JSX. A continuación se destaca la estructura de carpetas junto con algunos de los archivos más relevantes.

-/src – directorio raíz.

-index.jsx – archivo de ejecución principal.

- **-/api** – contiene los métodos que llaman a las rutas de la API de la aplicación
- **-/context** – método que contiene el contexto de la aplicación utilizando la API de contexto de React y que corresponderá a un objeto con los datos y documentos del usuario.
- **-/util** – métodos de utilidad.
- **-/components** – carpeta que contiene los componentes (vistas) de la aplicación.
  - **-/Auth** – contiene los compontes relacionados con la vista de autenticación.
  - **-/Profile** – contiene los componentes relacionados con la vista del perfil del usuario.
    - **-/Citations** – contiene la generación de las citaciones de cada tipo de documento.
    - **-/DocumentForms** – contiene los formularios de los diferentes tipos de documentos.
    - **-/DocumentRenders** – contiene las vistas de los diferentes tipos de documentos.
  - **-Header.jsx** – componente que representa la cabecera con los datos del usuario de la vista Profile.
  - **-ProfileDocuments.jsx** – componente que representa la lista de documentos de la vista Profile.

Recorrido del usuario

La interfaz cuenta con tres tipos de componentes: los que manejan la información de autentificación, los que presentan la información de usuario y los que presentan la información de los documentos.

Cuando el usuario accede a la aplicación mediante la ruta “/”, primero se comprueba que posea un token válido. En caso contrario, redirige a la vista “Auth”, formada por los componentes “Login” y “Register”.

Cuando el usuario accede a la aplicación mediante la ruta “/user/:id”, se utiliza el identificador del segmento “/:id” para recuperar los datos del usuario correspondiente. Esta ruta no requiere autenticación y solo sirve para lectura de perfiles.

Toda la aplicación está encarrada bajo el contexto “UserInfoContext”. Una vez el usuario se ha autenticado y accede a la aplicación, se llama a este contexto, que realiza una petición a la ruta “getUserInfo” y devuelve la información del usuario autenticado.

Esta información del usuario corresponde a sus datos de perfil y a los documentos de su perfil y se almacena en el estado de la aplicación.

Seguidamente se carga la ruta “Profile”, que contiene los componentes “Header”, encargado de manejar los datos personales del perfil, y “ProfileDocuments”, que presenta y gestiona los documentos.

Estilos

Se utiliza el framework Materialize para dar estilos genéricos a la aplicación y maquetar los formularios.

Por otra parte, todo el CSS personalizado se encuentra en un único archivo “main.css” servido de forma estática. Se utiliza la metodología BEM para nombrar las clases.

Se decidió hacerlo de esta manera porque Deno hoy en día todavía no dispone ninguna librería de preprocesamiento de CSS.

Testing

El testing que se realiza en el proyecto tiene el objetivo de verificar que la API está enviando correctamente la información a la base de datos, que esta se almacena y que se recupera o elimina satisfactoriamente. Surgió la necesidad de escribir estas pruebas pensando en el momento de desplegar la aplicación en producción, para asegurar que los métodos de la API seguían funcionando y que la base de datos respondían correctamente de forma rápida y fiable.

Los archivos de tests se encuentran en la carpeta “src/tests”. A continuación se detalla qué tests realizan:

- documents_api_test

- Comprueba que se conecta a la colección de los documentos.
- Añade tres nuevos registros y comprueba que se han creado los documentos.
- Intenta recuperar los documentos que acaba de añadir.
- Actualiza un documento, lo recupera y comprueba que se ha cambiado.
- Borra los documentos de una categoría y comprueba que se han borrado.
- Borra los documentos de prueba restantes y comprueba que se han borrado.

- users_api_test
  - Comprueba que se conecta a la colección de los usuarios.
  - Añade un nuevo usuario y comprueba que se ha creado el documento en base de datos.
  - Intenta recuperar el usuario que acaba de añadir mediante el email.
  - Actualiza el usuario, lo recupera y comprueba que se ha actualizado correctamente.
  - Borra el usuario de prueba y verifica que se ha borrado.

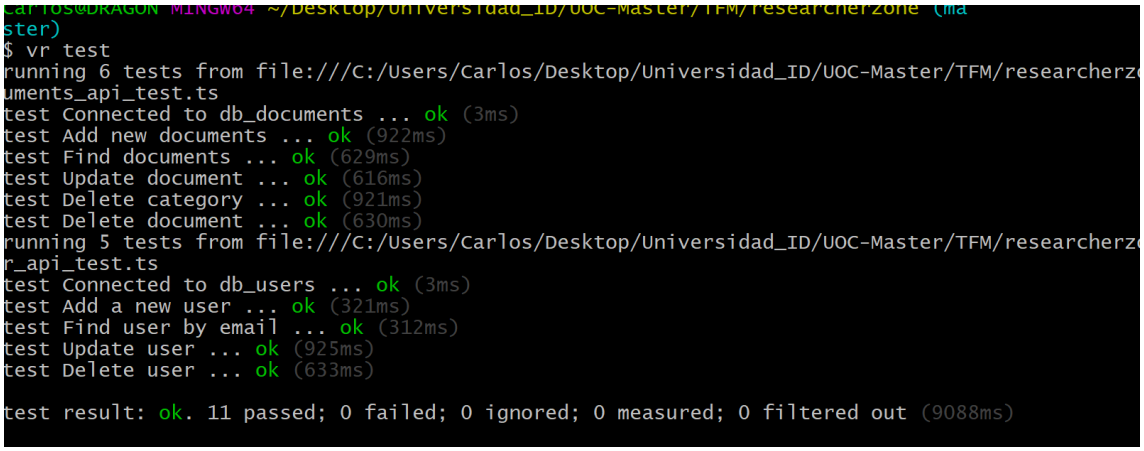

Figura 17: Tests pasando contra la base de datos de producción

Despliegue en producción

Una vez la aplicación ha sido validada funcionalmente y se ha comprobado que la base de datos de MongoDB Atlas recibe, modifica y envía datos de forma satisfactoria, es momento de subir el proyecto a producción.

Para continuar, se ha habilitado el servicio Cloud Run en el espacio de la Google Cloud Console que se utilizará en este proyecto.

Cloud Run es un servicio de la Google Cloud Platform (en adelante GCP) que permite publicar una aplicación *serverless* utilizando contenedores de Docker sin tener que preocuparse por la infraestructura.

En primer lugar, se debe empaquetar la aplicación en una imagen de Docker. Para ello se ha provisto del comando “vr build” en el proyecto, que elimina la imagen anterior –si existe– y utiliza el Dockerfile para generar la imagen con los tags requeridos.

Seguidamente se debe autenticar la cuenta de la GCP donde vamos a subir dicha imagen. Para ello se utiliza el comando “vr auth”, que lanza una URL donde se puede autorizar la cuenta deseada.

Finalmente, se utiliza el comando provisto “vr push”, que subirá la imagen al Container Registry, un servicio de la GCP que permite tener una biblioteca de imágenes que después se pueden utilizar en otros servicios.

A continuación, se crea un nuevo servicio en Cloud Run, al que se le especifica un nombre, una imagen y otras configuraciones como el tipo de tráfico o la autentificación.

Al finalizar la configuración, se añadirá una nueva revisión con la imagen del proyecto y en poco tiempo estará disponible.

Para finalizar el proceso de despliegue y conseguir una aplicación más visualmente agradable para acceder, se ha decidido adquirir el dominio “researcher.zone”. Una vez adquirido, se ha vinculado con el servicio de Cloud Run donde está alojada la aplicación.

Como resultado, la aplicación está disponible desde la URL “https://researcher.zone” en fecha de entrega de este documento.

Conclusión del proyecto

En el planteamiento inicial del proyecto se plantearon conseguir varios objetivos que debía satisfacer la aplicación y al finalizar esta se han podido cumplir de forma satisfactoria.

En primer lugar, se ha facilitado el proceso de creación de los perfiles haciendo que la interfaz de registro sea fácil de localizar, entendible con los campos justos y rápido de completar.

También se ha ofrecido la posibilidad de poder compartir dichos perfiles de forma rápida y fácil mediante una URL que es destacada por la interfaz desde el primer momento.

Así mismo, la interfaz permite categorizar los documentos entrados en los perfiles, facilitando la organización, y también separa claramente los tipos de documentos que pueden ser entrados, que serán reconocidos por los usuarios gracias a la terminología utilizada.

Finalmente se ha dado margen a la personalización ofreciendo la posibilidad de que los usuarios puedan añadir información adicional en la cabecera, incorporando una lista de redes sociales editable y permitiendo decidir el orden de los documentos en el perfil, además de añadir un tipo de documento especial donde los usuarios pueden entrar cualquier tipo de dato que consideren relevante de forma muy intuitiva.

Mejoras a futuro

En este apartado se listan algunas mejoras que se pueden realizar en el proyecto:

Crear una red de contactos o favoritos

- Permitir que los usuarios puedan dar “Recomendado” a otros usuarios.
- Añadir en el perfil el número de recomendaciones recibidas.
- Guardar los usuarios “Recomendados” en una agenda de contactos.
- Proporcionar la opción de que cada usuario pueda añadir a otros usuarios en una lista de “Recomendados”, creando así una red de contactos.

Añadir una funcionalidad para compartir bibliografías.

- Permitir a los usuarios crear una lista de bibliografías.
- Compartir estas listas de igual forma que se pueden compartir los perfiles.

Automatización de la entrada de documentos

- Añadir una funcionalidad con la que los usuarios serían capaces de recuperar la información de una publicación solamente introduciendo el DOI (Digital Object Identifier).

Añadir indicadores bibliométricos

- Añadir indicadores bibliométricos en el perfil del usuario como el *h*-index o el número de citaciones total de sus documentos.
- Añadir indicadores bibliométricos por documento, como por ejemplo el número de veces citado en otros documentos o el índice SRJ de la revista.

Anexo 1: Webgrafía

Deno. https://deno.land.

Docker. https://www.docker.com.

MongoDB Atlas. https://www.mongodb.com/cloud/atlas.

Google Cloud Platform. https://cloud.google.com/gcp.

Anexo 2: Librerías

ESM. react. https://esm.sh/react@17.0.2.

ESM. react-dom. https://esm.sh/react-dom@17.0.2.

Deno land. oak. https://deno.land/x/oak@v6.5.1/mod.ts.

Deno land. deno-mongo. https://deno.land/x/mongo@v0.23.0/mod.ts.

Deno land. dotenv. https://deno.land/x/dotenv/mod.ts.

Deno land. sha256. https://deno.land/x/sha256@v1.0.2/mod.ts.

Deno land. djwt. https://deno.land/x/djwt@v2.2/mod.ts.

ESM. cookie. https://esm.sh/cookie@0.4.1.

ESM. browser-image-compression. https://esm.sh/browser-image-compression@1.0.14.

Deno land. asserts. https://deno.land/std@0.97.0/testing/asserts.ts.

Materialize. https://materializecss.com.

Anexo 3: Cronograma del proyecto

Cronograma del Trabajo de fin de Máster. [Enlace]: https://docs.google.com/spreadsheets/d/1kW4E6_X1kvhsf3rz7wYLBNOoy90q5DvQzNDXhvlAHI4

Anexo 4: Repositorio del proyecto

https://github.com/cmgdragon/researcherzone