UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una app web para la gestión de múltiples cuentas de correo electrónico

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Hernandez Muñoz, Sergio

Tutor/a: Casamayor Rodenas, Juan Carlos

CURSO ACADÉMICO: 2023/2024

Resumen

Debido al crecimiento de internet, en los últimos años el correo se ha convertido en una herramienta fundamental. Por ello las personas disponen de varias cuentas de correo, ya sea para el trabajo, vincular nuestros dispositivos, registrarnos en servicios y plataformas, pudiendo ser todas estas cuentas de distintos proveedores. Por eso se ha desarrollado una aplicación web para poder gestionar el envio y recepción de mensajes de correo electrónico sin importar cual sea el proveedor de dicha cuenta y así mejorar la productividad, eficiencia y rapidez con la que se accede a las cuentas de correo ya que con una simple cuenta en esta herramienta nos permitirá consultar todos los correos de aquellas cuentas que añadamos de forma independiente. Además, todos aquellos mensajes consultados en esta herramienta se almacenarán permitiendo mantenerlos o borrarlos del servidor original de correo si se desea. Está herramienta está orientada a tecnologías de desarrollo como *Java Spring Boot*, *React* y *MongoDB*.

Palabras clave: aplicación web, gestionar el envio y recepción, proveedor, productividad, eficiencia, rapidez. servidor original de correo. tecnologías de desarrollo. *Java Spring Boot*, *React* y *MongoDB*.

Abstract

Due to the growth of the Internet, in recent years email has become a fundamental tool. That is why people have several email accounts, either for work, linking our devices, registering in services and platforms, and all these accounts can be from different providers. That is why a web application has been developed to manage the sending and receiving of email messages regardless of the provider of that account and thus improve productivity, efficiency and speed with which access to email accounts because with a simple account in this tool will allow us to consult all emails from those accounts that we add independently. In addition, all those messages consulted in this tool will be stored allowing them to keep them or delete them from the original mail server if desired. This tool is oriented to development technologies such as *Java Spring Boot*, React and *MongoDB*.

Keywords: web application, manage sending and receiving, provider, productivity, efficiency, speed, original mail server, development technologies, Java Spring Boot, React and MongoDB.

Tabla de contenidos

Índice de figuras

1. Introducción

El proyecto presentado tiene como objetivo el desarrollo de una aplicación web para las personas que poseen varias cuentas de correo electrónico pudiendo ser estas de diferentes proveedores y necesitan una rápida y eficaz gestión de las cuentas, mediante una sencilla interfaz, permitiendo el envío y recepción de correos electrónicos con todo tipo de documentos sin borrar ningún elemento del servidor de correo original.

1.1 Motivación

La motivación de este proyecto viene dada por la propia necesidad de poder gestionar de una forma rápida y sencilla todas aquellas cuentas de correo electrónico que he ido acumulando con el tiempo por diferentes causas.

Esta necesidad es producida debido a que las diferentes aplicaciones existentes en actualidad cumplen con esta necesidad de poder gestionar varias cuentas de correo electrónico pero fallan en algunos aspectos como una interfaz compleja, con muchos elementos que complican la agilidad de enviar y consultar un correo, dificultad para añadir y configurar una nueva cuenta, algunas de ellas están pensadas para solo manejar un cuenta a la vez, es decir, permiten añadir diferentes cuentas pero no permiten tener fácil acceso a todas las cuentas, además algunas de ellas solo tienen versión de escritorio complicando el acceso desde más puntos.

Por eso se vio la necesidad de crear una nueva aplicación, que simplifique la interfaz del usuario facilitando no solo el envío y recepción de correos electrónicos, sino una fácil configuración de las diferentes cuentas de correo electrónico, permitiendo al usuario ahorrar tiempo.

1.2 Objetivos

El objetivo principal que se quiere alcanzar es la elaboración de una aplicación web fácil de entender y manejar y con un coste de funcionamiento y mantenimiento bajo, además de que se adapte a diferentes dispositivos. También se tendrán en cuenta otros aspectos como la seguridad y privacidad de las cuentas de correo mediante el uso del cifrado de las cuentas de la propia aplicación y de las cuentas de correo. En cuanto al fácil manejo de la aplicación lo podremos conseguir con la implementación de una interfaz minimalista, con el menor número de elementos posible en pantalla para no saturar al usuario.

Otros de los objetivos que se quieren alcanzar, es la creación de nuevas cuentas de la propia aplicación, añadir diferentes cuentas de correo electrónico, poder enviar y recibir correos electrónicos con cualquier tipo de adjunto y formato HTML. Y no menos importante, que la

recepción de todos los correos electrónicos se haga de forma concurrente, es decir, que se puedan recibir correos de todas las cuentas al mismo tiempo sin tener que ir cambiando de unas a otras.

Y como último, no se deben olvidar objetivos como una buena estructura del código implementado, usando *React* que es una librería JavaScript diseñada para crear interfaces de usuario facilitando el desarrollo de aplicaciones en una solo página y *Java Spring Boot* que es un *framework* que facilita el desarrollo de aplicaciones web y la creación de microservicios[1], que permitan que si en un futuro queremos añadir redes sociales como *Whatsapp*,*Telegram*, entre otras para facilitar y agilizar el manejo de estas, ya tengamos una estructura preparada y sea solo añadir un nuevo microservicio de la red social.

2. Estado del Arte

En la actualidad disponemos de una multitud de aplicaciones para gestionar el envío y recepción de correos electrónicos, siendo todas ellas muy completas permitiendo realizar casi cualquier acción posible a la hora de las gestiones, ya sea como la edición de los propios correos, guardarlos, marcarlos, filtrarlos entre otras. Con esta aplicación lo que se pretende solucionar es la problemática del tiempo y gestionar varias cuentas, es decir, estas aplicaciones son muy completas y por ello a el usuario en ciertas ocasiones dedica más tiempo del requerido para poder enviar correos simples con las diferentes cuentas de correo.

2.1 Aplicaciones Similares

Algunas de las aplicaciones que compararemos son muy conocidas y usadas a diario por todas las personas ya que el correo electrónico es uno de los medios de comunicación más antiguos desde que existen los ordenadores e internet. Entre estas aplicaciones tenemos *Microsoft Outlook*, *Gmail*, *Thunderbird*, *Apple Mail* entre muchas otras siendo las más conocidas y usadas la de *Outlook* y la de *Gmail* y por lo tanto las que trataremos.

2.1.1 Microsoft Outlook

Outlook es la aplicación de correo electrónico desarrollada por *Microsoft*. Su origen se remonta a 1988, la cual fue lanzada junto al paquete de aplicaciones de *Microsoft Office*. Esta aplicación no solo está pensada como un gestor de correos electrónicos, también lo está para ser una agenda personal. Actualmente se encuentra desarrollado en C++ y que la versión comentada será la de la aplicación web, puesto que la aplicación desarrollada en este TFG es también una aplicación web.

Esta herramienta es muy completa, ya que se ha desarrollado a lo largo de muchos años y con una gran inversión, pero el hecho de que tenga tantos elementos, aunque estén bien organizados, puede provocar que algunos usuarios no puedan encontrar ciertas características que necesiten durante el uso de la aplicación.

La herramienta dispone de una barra de navegación en la parte superior que contiene elementos que no tienen que ver directamente con los mensajes, como puede ser la configuración de la aplicación, accesos a otras aplicaciones complementarias a la de *Outlook*, una barra de búsqueda entre otros.

Figura 1: Barra de navegación de Outlook

A continuación, en la parte inferior de la barra anterior, nos encontramos otra, esta ya está dedicada al tratamiento de los mensajes, con múltiples elementos para tratar los mensajes como los botones de enviar, responder, fijar, eliminar, guardar, etc.

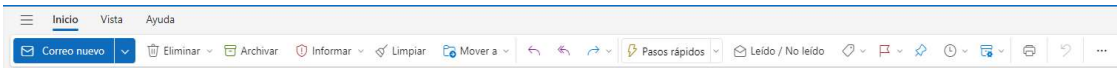

Figura 2: Barra para gestión de mensajes de Outlook

A la izquierda nos encontramos tres elementos desplegables cuya finalidad es para la organización de los mensajes. Estos elementos son el de Favoritos, Carpetas y Grupos los cuales pueden ser configurados para que mostremos los mensajes de forma organizada según las preferencias del usuario.

Figura 3: Panel para gestionar el almacenamiento de mensajes de Outlook

Y en el margen izquierdo tenemos un conjunto de botones que son accesos al paquete de aplicaciones de *Office 365* que complementa a *Outlook*, como *Word*, *Excel Paint*, ya que son archivos que se pueden adjuntar en los diferentes mensajes.

Figura 4: Barra de herramientas de Outlook

Por último, en el centro podemos visualizar todos los mensajes de correo según los filtros que tengamos seleccionados, mostrando cual es la cuenta de correo desde la que se nos envía el mensaje, información sobre su contenido, la fecha, además de incluir algunos botones que ya estaban en la barra superior como el de fijar mensajes. Y a su derecha encontraremos el espacio para la visualización del mensaje completo, con algunos botones para responder el mensaje o reenviarlo.

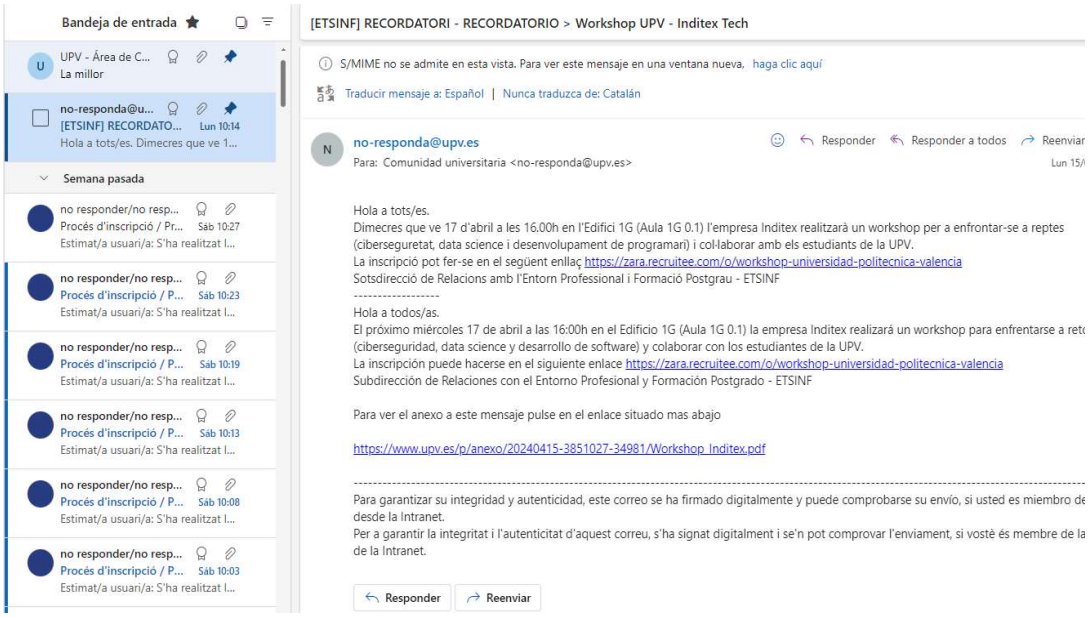

Figura 5: Pantalla de visualización de mensajes de Outlook

Hay que añadir que en el caso de la aplicación web no tenemos la opción de añadir cuentas de correo de otros proveedores o no he sido capaz de encontrar esta opción, dando a entender que algunas funcionalidades no son tan fáciles de manejar. Sí queremos añadir otras cuentas tendremos que usar la aplicación.

2.1.2 Gmail

Gmail es otra de las grandes aplicaciones web de email usadas, creada por *Google* en 2004, llegando a superar a *Outlook* en 2012 con una cantidad de usuarios estimada de 425 millones. Está desarrollada con la tecnología AJAX[2], la cual usa JavaScript, XHTML y CSS.

En el caso de *Gmail* se presenta una interfaz menos cargada que la de *Outlook*, por lo tanto a simple vista facilita bastante el uso de la aplicación.

En la parte superior de la interfaz, disponemos de una barra de navegación sencilla, con el logo de *Gmail*, botones de configuración, el icono de nuestra cuenta y un buscador para los mensajes.

Figura 6: Barra de navegación de Gmail

En el margen izquierdo, tenemos varios elementos para organizar todos nuestros correos como en el caso de *Outlook*, ya sea poniendo en mensajes de spam, enviados, Borradores, etc, dando la posibilidad de añadir todas las categorías que queramos para filtrar los mensajes a nuestro gusto.

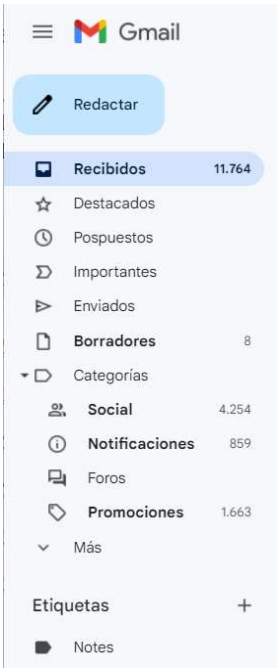

Figura 7: Menú para la gestión de mensajes Gmail

En el centro de la interfaz tenemos la bandeja de entrada con los mensajes que se mostrarán según los filtros aplicados previamente. De primeras se muestran algunos botones para recargar los mensajes, el número de mensajes que hay, la lista de los mensajes, de los que se nos muestra el correo electrónico que envió el mensaje, información que contiene y la hora a la que fue recibido. También contiene elementos para marcar y fijar los correos que llegan.

Figura 8: Lista de mensajes de Gmail

Cuando pinchamos sobre un mensaje, se cambia la bandeja de los mensajes por una vista nueva donde podemos visualizar el correo. Dentro del propio mensaje tenemos botones para reenviar y responder.

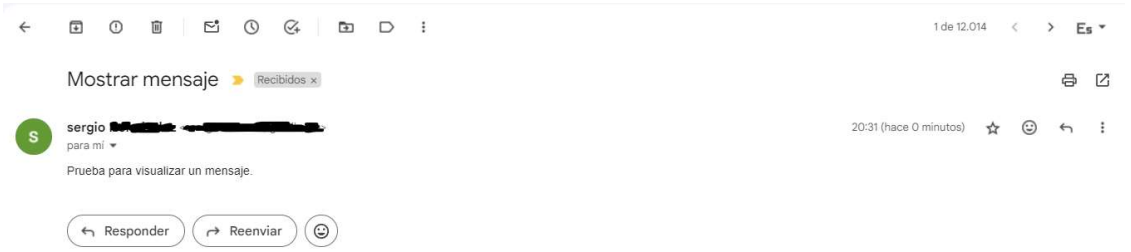

Figura 9: Visualización de mensajes en Gmail

Para el envío de un nuevo correo, tendremos que pinchar sobre el botón de redactar, situado debajo del logo y aparecerá una ventana emergente que nos permitirá la edición de un nuevo mensaje ya que contiene bastantes botones para dar formato al texto introducido, permitiendo añadir cualquier tipo de adjunto.

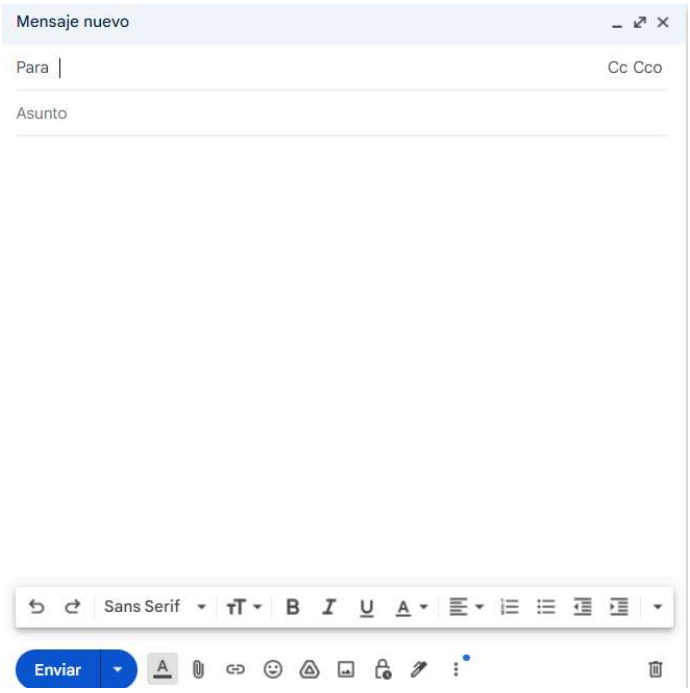

Figura 10: Ventana para redactar nuevo mensaje en Gmail

Al igual que en la aplicación de *Outlook* no he sido capaz de encontrar la opción de añadir otras cuentas de correo electrónico de diferentes proveedores en la aplicación web. Si queremos añadirlas tendremos que descargar la aplicación.

2.2 Conclusiones

La búsqueda de aplicaciones de correo electrónico ha sido una tarea sencilla, ya que en la actualidad disponemos de una gran variedad de estas, debido a que es uno de los primeros medios comunicación desde que existe internet y porque al tener tantos años, el sistema de envío está estandarizado, por lo que hay mucha información y recursos en internet para poder crear una aplicación.

En esta ocasión se han expuesto las aplicaciones de *Microsoft* y *Google*, ya que a día de hoy son las más empleadas. En el caso de *Outlook*, se ha observado una aplicación muy completa, cargada de muchos elementos lo que provoca que el usuario tenga que aprender la interfaz, para poder manejarla correctamente y en el caso de *Gmail*, tenemos una estructura más simple, que permite un uso más fácil de la interfaz, siendo esta una de las características que buscadas en este TFG. Aun con todo esto, se ha intentado añadir cuentas de correo de diferentes proveedores, como lo haría un usuario promedio (navegando por la interfaz en busca de la opción) y no se ha encontrado.

Como conclusión sobre el análisis tanto del mercado, como de las aplicaciones examinadas, se puede afirmar que hay muchas aplicaciones de este estilo, a causa de los años que llevan en el mercado y que ambas aplicaciones son muy completas, pero en sus versiones de navegador no encontramos algunas de las características deseadas.

3. Metodología de desarrollo Software

En este apartado se tratará la metodología empleada para el desarrollo de la aplicación web[3]. Pese a que haya muchas opciones, ya que estas se subdividen en metodologías tradicionales y metodologías ágiles, nosotros emplearemos la metodología incremental perteneciente al modelo tradicional, debido a que para nuestro caso va a ser la que mejor se va a adaptar como trataremos a continuación.

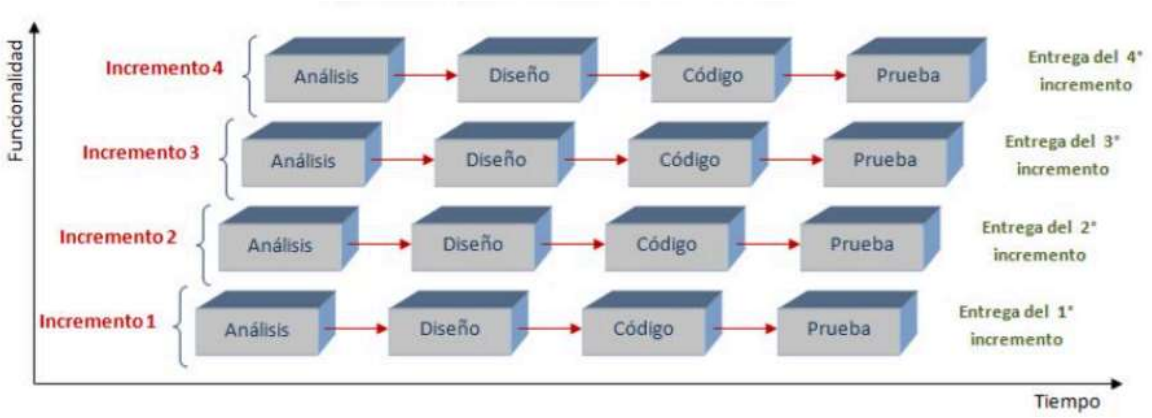

Figura 11: Esquema Metodología Incremental

3.1 Metodología Incremental

Para abordar este desarrollo, se ha utilizado la metodología incremental[5] dado que nos permite adaptar y moldear nuestro proyecto de una forma sencilla. El modelo incremental está compuesto de diferentes etapas que van incrementando una y otra vez permitiendo añadir funcionalidades nuevas, hasta concluir con el desarrollo final de la aplicación. Estas etapas son:

1. Análisis: En esta etapa nos encargamos de reunir toda la información necesaria para obtener los requisitos necesarios para su posterior diseño y programación.
2. Diseño: En la etapa de diseño se dará forma a la información obtenida en la fase anterior.
3. Programación: Se desarrollará el código necesario que se ajuste al diseño para conseguir la implementación de la funcionalidad.
4. Pruebas: Se realizarán las pruebas para comprobar que el código implementado cumple con la funcionalidad implementada.

Por último, una vez acabada las fases del incremento, este se añadirá a los incrementos realizados anteriormente para ir completando poco a poco la aplicación y así poder finalizar el producto, pudiendo observar un avance progresivo de este.

3.2 Ventajas

1. Rápido desarrollo del software.
2. Gran flexibilidad y adaptabilidad a cambios durante el desarrollo.
3. Bajo coste en comparación a otros modelos.
4. El cliente puede estar fácilmente implicado, con gran precisión, sobre los avances del proyecto.
5. Fácil identificación de los errores

3.3 Desventajas

1. Se necesita un buen análisis de los requisitos y funcionalidades.
2. En ocasiones el análisis de los requisitos y funcionalidades para desarrollar el software se puede convertir en una tarea compleja.
3. Una vez realizada una etapa del incremento y pasado a la siguiente, no se puede volver atrás.
4. Resolver problemas se puede convertir en una tarea difícil, puesto que pueden implicar la modificación de varias funcionalidades.
5. Cada entrega del incremento debe seguir manteniendo la aplicación en un funcionamiento correcto, por lo tanto se deben hacer pruebas exhaustivas.

3.4 Estructura del proyecto

En este TFG, la estructura del proyecto es bastante simple, ya que solo va a ser desarrollada y pensada por una persona, además, esta se adaptará al modelo incremental explicado anteriormente.

Para la definición de los requisitos funcionales, se analizó alguna de las diferentes aplicaciones web de correo electrónico existentes y se evaluó por el desarrollador todas las funcionalidades necesarias para el proyecto, uniendo esta información y creando una lista con la gran mayoría de los requisitos para realizar el desarrollo. Posteriormente se definió un orden para todos estos requisitos, para que la aplicación fuese saliendo con versiones funcionales, siendo los primeros incrementos los login de la aplicación.

Como ejemplo tomaremos el login, el cual se analizó y se dedujo que debería ser implementado en dos incrementos. A continuación se diseñó la estructura que éste tendría en el proyecto, siendo diseñado primero el *back end*[4]. Por lo tanto la siguiente fase fue implementar el *back end*, creando los *endpoints* necesarios para poder conectarnos y registrar un nuevo usuario. Como siguiente paso del primer incremento, tenemos las pruebas realizadas con Postman hacia el CRUD del *login.* Ya probado se añadió al proyecto principal en el repositorio empleado, este caso GitLab.

Y ya para el segundo incremento, que sería la funcionalidad del *login*, se han realizado los mismo pasos pero implementado la parte de la interfaz para poder realizar las funciones de *login* y registro, siendo probado en la siguiente etapa con la propia API del *back end.* Una vez probado se uniría al proyecto principal, al igual que el *back end* pero en este caso a su repositorio correspondiente.

4. Especificación de requisitos

En todo proyecto, una de las partes más importantes para el éxito de este, es una buena definición de los requisitos que describen todo el sistema a realizar, para un posterior desarrollo de sus casos de uso y así cumplir con todas las funcionalidades y características esperadas.

4.1 Requisitos Funcionales

En este apartado trataremos las funciones y comportamientos que la aplicación debe realizar, detallando las acciones que la aplicación debe ser capaz de ejecutar y la interacción con los usuarios y otros sistemas. Los Requisitos funcionales[5] (RF) para la aplicación son:

- Login: Se debe desarrollar una funcionalidad que permita la creación de usuarios, su posterior inicio de sesión, actualización de la cuenta, su eliminación.
- Logout: El usuario podrá cerrar la sesión.
- Recepción y visualización de mensajes: El sistema debe ser capaz de recibir los mensajes de la API y el usuario de visualizarlos.
- Envío de mensajes: El usuario debe poder crear nuevos mensajes, enviarlos y responderlos.
- Mensajes marcados: Los correos podrán ser marcados para que el usuario pueda localizarlos con mayor facilidad.
- Filtrado de mensajes: Debe haber una función que permite buscar los mensajes
- Gestión de cuentas de email: El usuario debe ser capaz de crear y eliminar nuevas cuentas de email.

4.2 Requisitos No Funcionales

Este punto trata sobre las propiedades y características de la aplicación que no tienen una relación directa con funciones específicas, pero son necesarias para su calidad y rendimiento. Los Requisitos No Funcionales (RNF) para la aplicación son:

- Seguridad: La aplicación guardará las contraseñas cifradas
- Usabilidad: La interfaz de la aplicación debe ser fácil de usar sin que el usuario requiera ningún tipo de formación para poder usarla.
- Tiempo de respuesta: La aplicación realizará las solicitudes de los usuarios en menos de 2 segundos.
- Compatibilidad: La aplicación será compatible con los diferentes navegadores y dispositivos.

4.3 Diagrama de casos de uso

Para representar los casos de uso se va a utilizar un diagrama UML (Unified Modeling Language), el cual está estandarizando, ayudando así a la visualización, el análisis y el diseño de todas las funcionalidades de la aplicación.

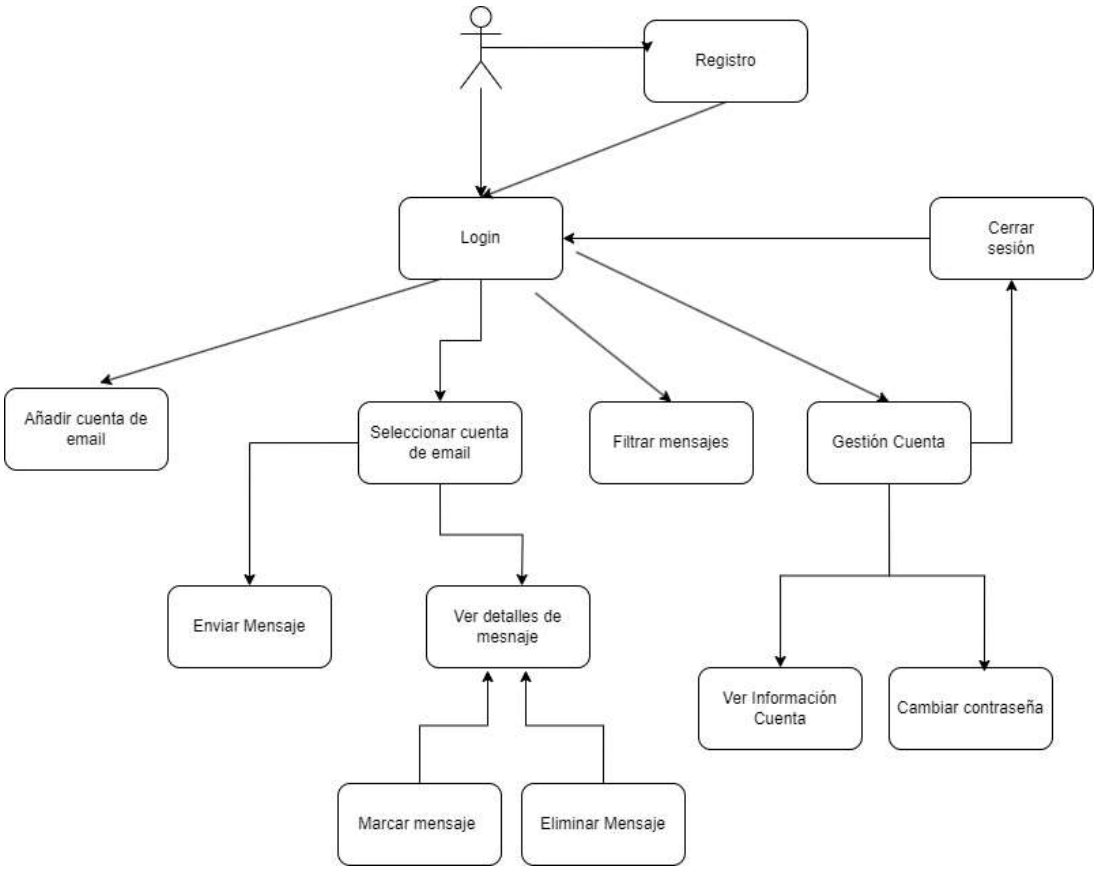

Figura 12: Diagrama UML de casos de uso

4.4 Descripción de casos de uso

A continuación se detallarán los casos de uso de nuestra aplicación, que aparecen en la *Figura 12*. Son los siguientes:

Caso de uso	Registro.
Actores	Usuario.
Propósito	Crear un usuario nuevo para la aplicación.
Descripción	El usuario tendrá que introducir los datos del

	formulario para el registro: nombre de usuario, contraseña, correo electrónico, número de teléfono.
Flujo	1. El usuario pulsa el botón de registrar. 2. Saldrá una ventana con los campos a rellenar. 3. Una vez rellenado el formulario con los datos, el usuario pulsa el botón registrar. y crea el usuario.
Precondición	-
Postcondición	El usuario se ha creado y se ha validado que los datos sean correctos.

Caso de uso	Login.
Actores	Usuario.
Propósito	Iniciar sesión para entrar a la aplicación.
Descripción	El usuario introduce los datos de la cuenta: usuario y contraseña.
Flujo	1. El usuario introduce el su nombre de usuario y la contraseña 2. Aprieta el botón de *login* y si los datos son correctos accede a la aplicación.
Precondición	El usuario tiene que estar registrado.
Postcondición	El usuario queda autenticado

Caso de uso	Gestión Cuenta.
Actores	Usuario.
Propósito	Poder acceder a las funcionalidades para gestionar la cuenta.
Descripción	El usuario carga la funciones de gestión de usuario
Flujo	1. El usuario pulsa el botón para desplegar la funciones de gestión de cuenta

	2. Se cargan las funciones con sus respectivos datos
Precondición	El usuario tiene que haber iniciado sesión
Postcondición	-

Caso de uso	Cerrar sesión.
Actores	Usuario.
Propósito	Salir al inicio de sesión de la aplicación.
Descripción	El usuario cierra sesión y sale al login de la aplicación.
Flujo	1. El usuario aprieta el botón de cerrar sesión. 2. La aplicación cierra se cierra y se vuelve al menú de inicio, para poder volver a iniciar sesión.
Precondición	El usuario tiene que estar autenticado
Postcondición	Se vuelve al menú de inicio

Caso de uso	Ver información cuenta.
Actores	Usuario.
Propósito	Ver los datos de la cuenta.
Descripción	El usuario visualiza los datos de la cuenta.
Flujo	1. El usuario aprieta el botón de ver información de la cuenta 2. Aparece un popup con la información de la cuenta. Una vez visualizada el usuario cierra el popup y se vuelve a la vista anterior.
Precondición	El usuario tiene que estar autenticado.

Caso de uso	Ver información cuenta.
Postcondición	Se vuelve a la vista anterior

Caso de uso	Cambiar contraseña.
Actores	Usuario.
Propósito	Cambiar la contraseña del usuario.
Descripción	El usuario cambiará su contraseña en el caso que quiera hacerlo.
Flujo	1. El usuario pulsa el botón de cambiar contraseña. 2. Se despliega una ventana donde el usuario podrá escribir la nueva contraseña. 3. El usuario aprieta el botón de cambiar y se realiza el cambio de contraseña y se vuelve a la vista anterior o el usuario pulsa cerrar y se vuelve a la vista anterior sin realizar ningún cambio.
Precondición	El usuario tiene que estar autenticado
Postcondición	Se valida el cambio de contraseña

Caso de uso	Añadir cuenta de email
Actores	Usuario
Propósito	Añadir una cuenta de email
Descripción	El usuario introduce los datos para añadir la cuenta: correo electrónico, contraseña, host, port, ssl, protocolo.
Flujo	1. El usuario pulsará el botón de añadir cuenta. 2. Se desplegará una ventana con el

	formulario y el usuario los rellenará. 3. El usuario pulsará añadir y se validará que los datos sean correctos para poder añadir la cuenta o cancelar volviendo a la vista anterior.
Precondición	El usuario tiene que estar autenticado.
Postcondición	Se comprueban los datos añadidos.

Caso de uso	Seleccionar cuenta de email
Actores	Usuario
Propósito	Elegir la cuenta de email que se quiere cargar.
Descripción	Usuario selecciona una de las cuentas que tenga añadidas para cargarla.
Flujo	1. El usuario pulsa el botón de seleccionar cuenta y aparecen las cuentas de email guardadas. 2. El usuario selecciona la cuenta que quiere mostrar y se cargan los mensajes asociados.
Precondición	El usuario tiene que estar autenticado y la cuenta de correo debe estar creada.
Postcondición	Se deben mostrar los mensajes asociados a la cuenta seleccionada.

Caso de uso	Enviar mensaje
Actores	Usuario
Propósito	Enviar un mensajes de correo electrónico
Descripción	El usuario rellena el formulario para poder mandar el mensaje: para, cc, asunto, mensaje, adjunto.
Flujo	1. El usuario pulsa el botón de enviar mensaje.

	2. Se abre una ventana con los campos para rellenar el mensaje. 3. El usuario pulsa el botón de enviar y se manda el correo.
Precondición	El usuario tiene que estar autenticado y la cuenta de email seleccionada.
Postcondición	Verificar que el mensaje se mande

Caso de uso	Ver detalles de mensaje
Actores	Usuario.
Propósito	Visualizar el correo electrónico.
Descripción	-
Flujo	1. El usuario pulsa sobre un mensaje de la lista. 2. Se abre una ventana con la información del mensaje
Precondición	El usuario tiene que estar autenticado y la cuenta de email seleccionada.
Postcondición	-

Caso de uso	Marcar mensaje
Actores	Usuario.
Propósito	Marcar un mensaje.
Descripción	El usuario marcará los mensajes que quiera para poder consultarlos más rápidamente.
Flujo	1. El usuario pulsará el botón de marcar. 2. El mensaje quedará marcado en la lista

	de mensajes.
Precondición	El usuario tiene que estar autenticado, la cuenta de email seleccionada y tiene que estar visualizando el mensaje.
Postcondición	El mensaje quedará marcado en la lista de mensajes.

Caso de uso	Eliminar mensaje
Actores	Usuario.
Propósito	Eliminar un mensaje de la lista.
Descripción	El usuario eliminará el mensaje de la lista.
Flujo	1. El usuario pulsa el botón de eliminar. 2. El mensaje es eliminado de la lista de mensajes.
Precondición	El usuario tiene que estar autenticado, la cuenta de email seleccionada y tiene que estar visualizando el mensaje.
Postcondición	El mensaje tiene que quedar eliminado de la lista de mensajes.

Caso de uso	Filtrar mensaje
Actores	Usuario.
Propósito	Mostrar los mensajes que quiere filtrar.
Descripción	El usuario introduce las palabras que quiere para mostrar los mensajes deseados.
Flujo	1. El usuario introduce en la barra de búsqueda las palabras claves para poder filtrar los mensajes. 2. Pulsa el botón de la lupa. 3. Se muestran los mensajes de la cuenta de email seleccionada.
Precondición	El usuario tiene que estar autenticado y la cuenta de email seleccionada.
Postcondición	Al quitar las palabras claves deben volver los mensajes de nuevo.

5. Diseño

5.1 Introducción

Durante el desarrollo de una aplicación, una de las etapas más importantes como se ha comentado en el TFG, es el diseño. Esta fase es muy importante, ya que tratará de dar forma a todo ese análisis de requisitos que se ha realizado previamente, para finalizar el proyecto de forma exitosa, puesto que un mal desarrollo del diseño puede provocar problemas en todo el proyecto, generando pérdidas de tiempo y de dinero. Dentro del proyecto podemos encontrar el diseño de las bases de datos, el diseño de la arquitectura y el diseño de la interfaz.

5.2 Diseño y modelado de la base de datos

Para poder almacenar todos los datos de nuestra aplicación, hemos utilizado *MongoDB*, un sistema de gestión de base de datos NoSQL[7] de la familia de sistemas orientados a documentos, donde los documentos se estructuran como objetos JSON(se almacena como BSON[8]), por lo tanto nos da la oportunidad de guardar varias formas de datos, creando un entorno altamente escalable. En este caso, como no se trata de base datos relacional, se ha optado por representar las estructuras en esquemas como se observa en la *Figura 14,* mostrando el nombre de la estructura y los datos que almacena. Cabe destacar que algunos de los datos almacenados de los mensajes no se utilizarán en la aplicación entregada en el TFG, sino que se guardarán para posibles desarrollos en el futuro.

El esquema que representa la estructura de la base de datos, consta de 3 colecciones que son la de Cuenta, Cuenta de Email y Mensajes, cada una de ellas tiene flechas que indican los elementos que guardará. A su vez estos elementos pueden guardar otras estructuras de elementos, para que se puedan ajustar a un formato JSON.


```
{
  "_id": {...},
  "emailAddress": "antoniopruebatfg@gmail.com",
  "password": "",
  "accountId": "662ab739166e0647f7fb8502",
  "enabledAccount": true,
  "receiveAccount": {
    "host": "imap.gmail.com",
    "port": "993",
    "sslEnabled": true,
    "seenflag": false,
    "deleteflag": false,
    "protocol": "imap"
  },
  "sendAccount": {
    "host": "smtp.gmail.com",
    "port": "465",
    "protocol": "smtp",
    "sslEnable": true
  },
  "_class": "com.multimail.email.entities.mailAccount"
}
```


Figura 13: Estructura JSON de la entidad Cuenta de Email

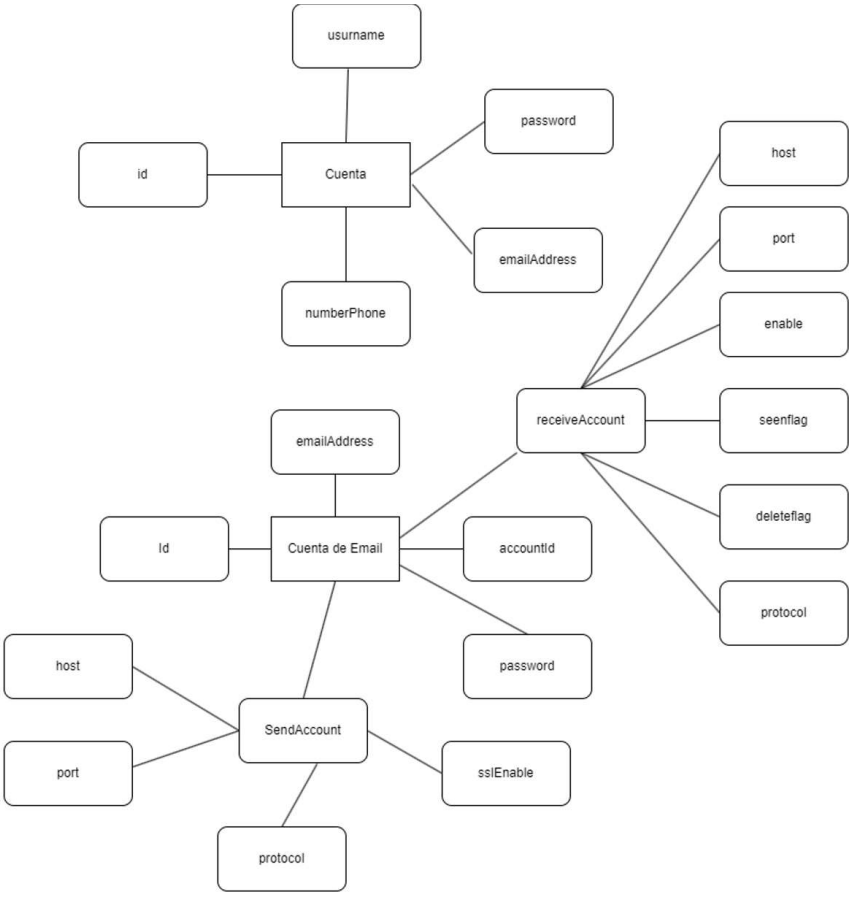

Figura 14: Esquema representativo de estructura de Cuenta de Email y Cuenta en MongoDB

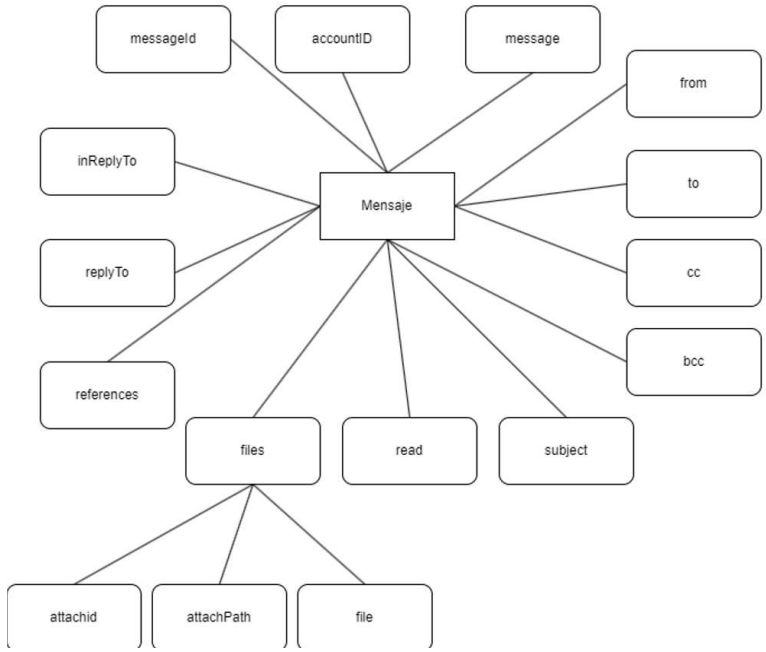

Figura 15: Esquema representativo de estructura de Cuenta en MongoDB

Las estructuras que guardan la información de la aplicación son las siguientes:

- **Cuenta**: En esta estructura se almacenan las cuentas para poder iniciar sesión en la aplicación, siendo los datos almacenados:
    - id: Identificación de la cuenta de tipo string y valor único.
    - username: Nombre de usuario de usuario de tipo string y valor único.
    - password: Contraseña de la cuenta y de tipo string.
    - emailAddres: Cuenta de email asociada a la cuenta de la aplicación y de tipo string.
    - numberphone: Teléfono asociado a la cuenta de tipo string.
- **Cuentas de email**: En esta estructura se almacenarán los datos necesarios para poder cargar los mensajes de la cuenta de correo electrónico. Cada cuenta de la aplicación puede tener varias cuentas de email.
    - id: Identificador de la cuenta de correo electrónico, de tipo string y valor único.
    - emailAddress: Cuenta de correo electrónico para cargar mensajes, de tipo string y valor único.
    - accountId: Identificador de la cuenta a la que pertenece la cuenta de correo electrónico y de tipo string.
    - password: Contraseña asociada a la cuenta de correo electrónico y de tipo string.
    - receiveAccount: Estructura que guardará la información necesaria para poder recibir los mensajes desde el servidor de correo electrónico.
        - host: Nombre del servidor de recepción de servidor de correo, de tipo string.
        - port: Número de puerto del servidor, de tipo string.
        - protocol: Protocolo de recepción[9], valores IMAP o POP3, de tipo string.
        - seenflag: Booleano que indica al servidor si marcar el mensajes como visto o no.
        - deleteflag: Booleano que indica si borrar el mensaje del servidor o no.
        - sslEnable: Booleano que indica el tipo de seguridad de la conexión con el servidor.
    - sendAccount: Estructura que guarda los datos necesario para poder enviar un mensaje a través de servidor de correo electrónico.
        - host: Nombre del servidor de envío de mensajes, de tipo string.
        - port: Número de puerto del servidor de envío, de tipo string.
        - protocol: Protocolo de envío de mensajes, SMTP.
        - sslEnable: Booleano que indica el tipo de seguridad al conectar al servidor para enviar un mensaje.
- **Mensajes**: Estructura que almacena la información de los mensajes. Por cada cuenta de correo electrónico se pueden tener muchos mensajes.
    - messageId: Identificador del mensaje, de tipo string y valor único.
    - accountId: Identificador de la cuenta de correo electrónico.
    - message: Mensaje enviado, de tipo string.
    - from: Cuenta de correo electrónico que envió el mensaje, tipo string.

- to: Cuenta o cuentas de email a la que se destina el correo, lista tipo string.
- cc: Cuenta o cuentas de email que están en copia, lista tipo string.
- bcc: Cuenta o cuentas de email que están en copia pero el destinatario no la puede ver, lista de tipo string.
- subject: Asunto del mensaje, de tipo string.
- read: Booleano que indica si el mensaje ya ha sido enviado al front o no.
- files: Estructura que guarda los adjuntos del correo electrónico.
  - attachId: El identificador de los adjuntos, de tipo string.
  - attachPath: Las ruta donde se guarda el mensaje, de tipo string.
  - file: Indica el tipo de archivo que es, de tipo string.
- references: Lista que contiene referencias a los demás mensajes contestados, de tipo string(campo que se puede usar en futuros desarrollos).
- replyTo: Lista que indica a quien se quiere reenviar o responder un mensaje, de tipo string.
- inReplyTo: Contiene el messageId al que se responde el correo electrónico, de tipo string.
- fecha: fecha que se recibió el mensaje en el servidor de correo, de tipo date.

Esta base de datos al no ser relacional, la forma que tenemos de obtener los mensajes de una cuenta de correo electrónico, sería guardando el id de la cuenta de correo en cada mensajes, para que cuando tengamos que obtener los mensajes, podamos hacerlo mediante ese campo. Lo mismo para obtener las cuentas de correo que están asociadas a una cuenta de la aplicación. Por último, comentar que la base de datos no ha sido clusterizada[10] y por lo tanto la instalación es *standalone*[11].

5.3 Arquitectura Software

La arquitectura empleada para el desarrollo de la aplicación es la Arquitectura en Capas, siendo su uso bastante extendido. Como su propio nombre indica se dividirá en diferentes capas, en este caso son la capa de presentación, la capa de aplicación, la capa de dominio y la capa de infraestructura.

5.3.1 Arquitectura en capas

Como se ha comentado en el punto anterior las capas de la arquitectura empleada son:

- Capa de Presentación: Esta capa es la que se encarga de manejar las interacciones con el *front end*[12], recibiendo la solicitud HTTP y devolviendo las respuestas.

- Capa de Aplicación: La capa de aplicación actúa como mediador entre la capa de presentación y la de dominio, haciendo las llamadas necesarias a los servicios que están en esta última capa. En este proyecto podemos decir que la capa de presentación y la capa de aplicación se encuentran juntas, debido al uso de *Java Spring Boot* en nuestro proyecto.
- Capa de Dominio: En esta capa se encuentra toda la lógica de negocio de la aplicación, además esta capa debe de ser independiente de las demás.
- Capa de Infraestructura: Esta capa se encargará de la interacción con infraestructuras externas como puede ser la base de datos.

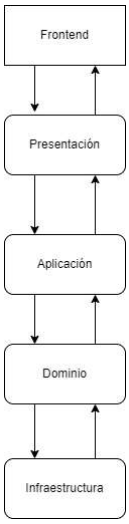

Figura 16: Flujo de la distintas capas de la arquitectura

Como podemos ver en la *Figura 16* cada capa se comunica con la siguiente sin conocer nada sobre las demás y sin saltarse el orden.

5.4 Diseño de la interfaz

A continuación trataremos el diseño final de la interfaz que se aplicará a nuestra aplicación, puesto que todos los realizados son muy parecidos debido a la sencillez de la interfaz, cambiando solo en algunos aspectos como el orden de los botones en el menú. La interfaz se ha realizado con la aplicación Pencil ya que es un software gratuito y permite un buen diseño de los prototipos con una curva de aprendizaje muy pequeña.

5.4.1 Login

Comenzaremos con el login, que marcará el comienzo de la aplicación, permitiéndonos entrar en esta. Está compuesto por el nombre de la aplicación y un formulario con dos campos y dos botones, que son el de iniciar sesión y el de registrar cuenta.

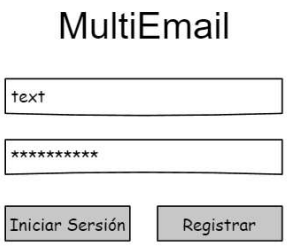

Figura 17: Prototipo de interfaz de iniciar sesión

Al pulsar sobre el botón el de iniciar sesión si los datos son correctos se cargará el menú, pero si apretamos el botón de registrar se aparecerá un ventana emergente con el formulario necesarios para registrar un nuevo usuario y el botón para crearlo, como se observa en la *Figura 18*.

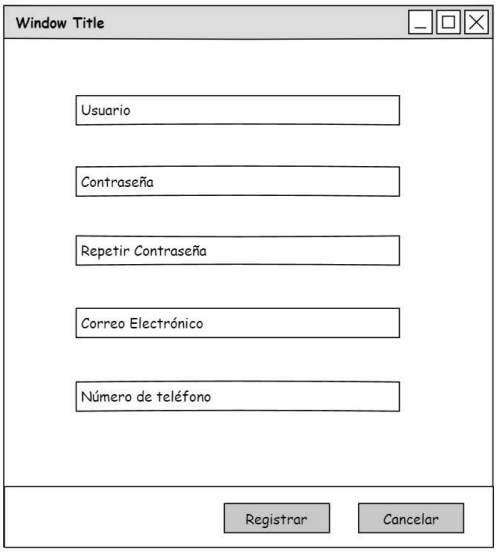

Figura 18: Prototipo de interfaz de registrar usuario

5.4.2 Menú

El menú será la siguiente parte de la aplicación que se cargará una vez se haya iniciado sesión. Dada la simplicidad de la aplicación, en este se encuentran casi todas las funcionalidades de la aplicación disponibles, las cuales son accesibles mediante botones que al pulsarlos realizan las acciones correspondientes. Algunas de estas son las de enviar mensajes, añadir cuentas de correo electrónico nuevas, entre otras, pero la que sí tendrá un impacto directo sobre el menú serán las acciones de seleccionar la cuenta, que mostrarán los mensajes de la cuenta seleccionada y el de eliminar cuenta que quitará todos los mensajes de la cuenta eliminada. Todo esto se observa en la *Figura 19*.

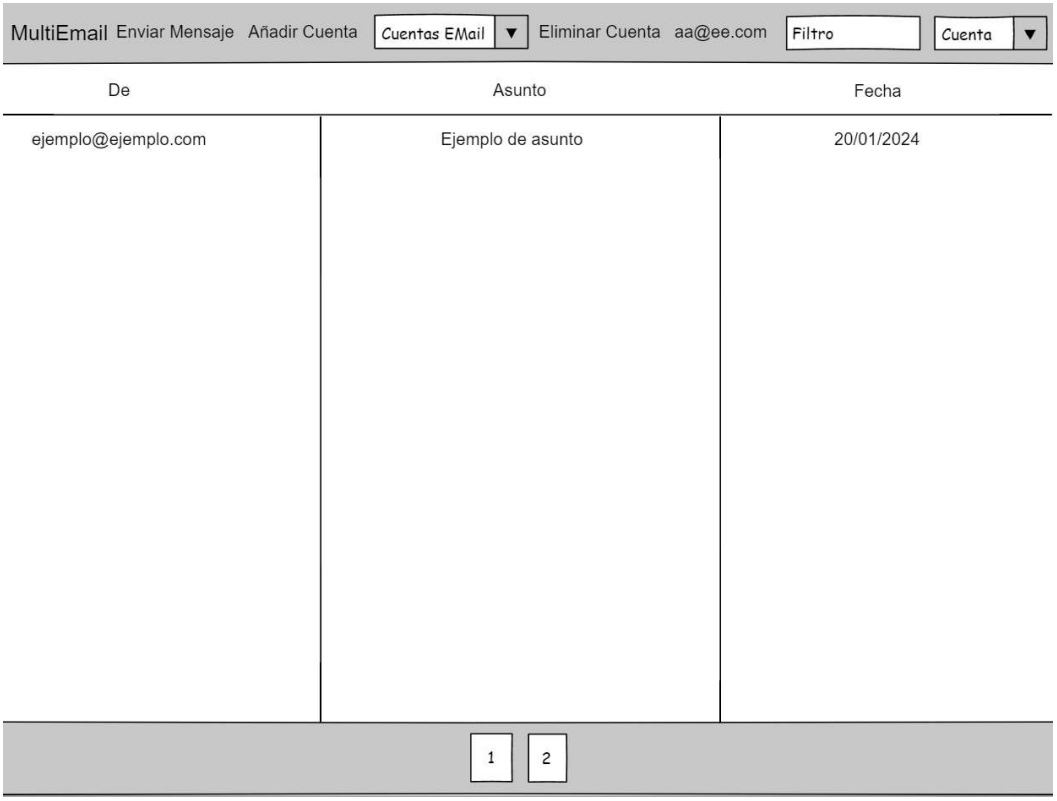

Figura 19: Prototipo de interfaz de la pantalla principal

5.4.3 Modales

En la aplicación todos los modales se encuentran en el menú, siendo estos los botones que al pulsarlos realizan una determinada función.

En este caso no se explicarán todos los modales, ya que todos realmente son muy parecidos aunque realicen una acción diferente, es decir, al pulsar sobre ellos saltará una ventana emergente la cual mostrará un formulario para introducir los datos necesarios para cada acción y en el margen inferior de la ventana encontraremos los botones que según la acción serán diferentes.

En las *Figuras 20* y *21*, se puede observar lo comentado en el párrafo anterior, teniendo dos funcionalidades distintas como son las de añadir una cuenta de correo electrónico y la de enviar un mensaje, ambas tienen los inputs necesarios y los botones para poder realizar la acción requerida.

Figura 20: Prototipo de interfaz para añadir cuenta de email

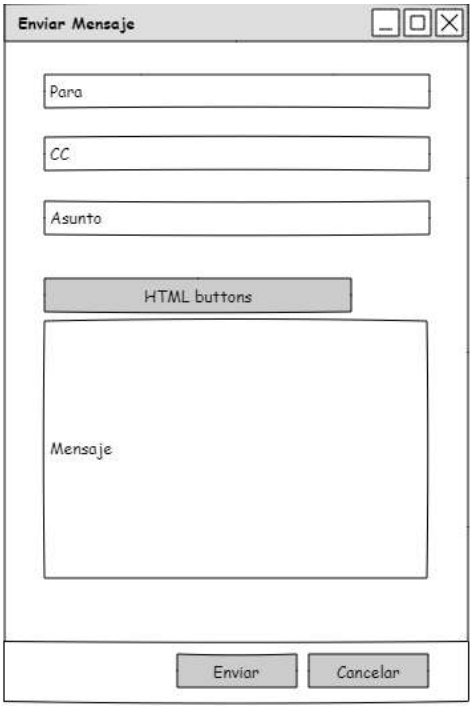

Figura 21: Prototipo de interfaz para enviar mensaje

5.4.4 Conclusiones

En conclusión se ha observado que el uso de una aplicación como Pencil, que permite con un aprendizaje muy rápido, crear prototipos de interfaces arrastrando el elemento que queramos añadir al resto del diseño ya creado, disponiendo de muchos de estos en la aplicación por defecto. Además el uso de prototipo no solo nos ayudará a la hora de implementar la interfaz por tener una referencia, sino que nos hará más conscientes de cuáles van a ser a requisitos funcionales que va a necesitar la aplicación, permitiendo así poder contemplar requisitos que faltaron en la fase de análisis y que ahora se podrán añadir. También nos da una idea aproximada a la realidad de cómo quedará el producto final.

6. Tecnologías y Herramientas

6.1 Introducción

Antes de comenzar el desarrollo de la aplicación se deben definir todas las tecnologías a utilizar, tanto los lenguajes de programación para el *front end* y el *back end* y las herramientas que se usarán para implementarlos y las herramientas que se usarán para la gestión de la base de datos. Cabe destacar que las tecnologías y herramientas han sido seleccionadas por ser gratuitas, de código abierto relativamente modernas y con soporte y porque el desarrollador tiene cierta familiaridad con ellas.

6.2 Front end

La tecnología empleada para el desarrollo de *front end* ha sido *React*. *React* es una biblioteca de *JavaScript* que se emplea para la construcción de interfaces de usuario, creada en 2013 por *Facebook*. React utiliza elementos reutilizables llamados componentes, que facilitan el desarrollo de la interfaz.

Junto a React ha de instalarse *Node.js* para poder generar un entorno de ejecución de JavaScript y npm, siendo este un gestor de paquetes para poder instalar las bibliotecas y dependencias. Los lenguajes usados por React en este proyecto han sido *JavaScript*, JSX, que es muy similar HTML pero en este caso se escribe dentro del propio *JavaScript* y CSS, para los estilos, aunque en este caso hemos utilizado *Reactstrap* que es una biblioteca que permite el uso de los componentes del framework de CSS de Bootstrap para facilitar la implementación de los elementos de la interfaz. Como último punto podemos añadir que también se ha utilizado la librería de *Axios* para poder realizar las peticiones HTTP a nuestro *back end*.

Figura 22: Logotipo de React, Node.js y Bootstrap

6.3 Back end

Para el desarrollo del *back end*, se ha empleado *Java*, un lenguaje de programación muy extendido, que fue lanzado al mercado por *Sun Microsystems* en 1995 y siendo posteriormente adquirido por *Oracle* el 27 de enero de 2010.

Java es un lenguaje de programación orientado a objetos, permitiendo ser ejecutado en equipos que tengan la máquina virtual de *Java*. *Java* es muy utilizado debido a su gran cantidad de librerías, en nuestro caso hemos empleado *Jakarta Mail*, para poder conseguir comunicar nuestra aplicación con los servidores de correo de cada empresa y así poder enviar y obtener mensajes a través de ellos. Esto se combina con el *framework* de *Spring Boot*, que nos permite simplificar el desarrollo de nuestra aplicación causado por el uso de las anotaciones que nos ayuda a manejar muchas configuraciones y procesos de forma automática.

Figura 23: Logotipo de Java, Spring Boot y Jakarta Mail

6.4 Base de Datos

Para la base de datos, se ha empleado *MongoDB*, creada por *10gen* en 2007, siendo esta una base de datos *NoSQL* en la que se ha optado por el uso de documentos BSON (Binary JSON). Al almacenarse como documentos, permitiendo mayor flexibilidad y escalabilidad a la hora de guardar los datos.

Figura 24: Logotipo de MongoDB

6.5 Herramientas

Las herramientas empleadas para el desarrollo de la aplicación son:

- Visual Studio Code: Software para la edición de código fuente, lanzado por Microsoft, disponible para Linux, Mac y Windows, que soporta diversos lenguajes de programación, pudiendo depurarlos y facilitar su desarrollo debido a su gran número de extensiones que permite agregar funcionalidades extras, además de su integración de con Git para guardar nuestros proyectos en un repositorio. En nuestro proyecto esta herramienta ha sido utilizada para el desarrollo del código de React y de Java.

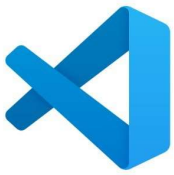

Figura 25: Logotipo de Visual Studio Code

- MongoDB Compass: Software que permite interactuar con la base de datos, ofreciendo las herramientas necesarias para explorar, editar y analizar el rendimiento, sin la necesidad del uso de comandos.

Figura 26: Logotipo de MongoDB Compass

- PostMan: Software que se emplea para probar, diseñar y documentar API, creando de forma sencilla peticiones HTTP y pudiendo analizar las respuestas de estas. En nuestro caso, esta aplicación se ha empleado para realizar las

peticiones HTTP a nuestro *back end*, para comprobar el correcto funcionamiento de los *endpoints* creados.

Figura 27: Logotipo de Postman

- Git: Software para el control de versiones que permite a los desarrolladores la colaboración con otros y el seguimiento de cambios. En nuestro caso, se ha empleado GitLab[13] como repositorio ya que amplía las funcionalidades disponibles sobre nuestro código en el repositorio.

Figura 28: Logotipo de Git y GitLab

- Pencil: Software que proporciona herramientas para el diseño y prototipado de interfaces de una forma sencilla y rápida. Esta aplicación es bastante completa ya que permite el uso de plantillas, elementos prediseñados, agregar extensiones, compatibilidad con varias plataformas, exportar los prototipos, entre otras muchas más acciones.

Figura 29: Logotipo de Pencil

7. Desarrollo de la solución

Para la implementación de esta aplicación se han usado diferentes tecnologías como ya se ha comentado anteriormente. Muchas de estas tecnologías han facilitado el desarrollo, ya sea a la hora de crear los *endpoints*, la base de datos, la interfaz de la aplicación entre otros. En este apartado se explicará de forma más detallada la estructura y cómo se han utilizado estas tecnologías para desarrollar el proyecto en la ya comentada Arquitectura en Capas.

7.1 Estructura e implementación del proyecto

En este proyecto nos encontramos con dos estructuras diferentes como ya se ha comentado, el *back end* y el *front end*, por lo tanto se deben tratar por separado, ya que las estructuras y tecnologías que usan, aunque están relacionadas ya que estas se comunican entre sí, son diferentes. Tanto la estructura y la implementación se explican en los mismo puntos ya que se aprovecha como está estructurado el proyecto para tratar también el desarrollo del mismo.

7.1.1 Estructura e implementación del front end

El *front end* presenta una estructura bastante sencilla, ya que al utilizar *React* y *Reactstrap*, casi todos los elementos del componente van a estar en un mismo archivo .jsx como se ve en la *Figura 30*. Cierto es que algunos de los componentes están en carpetas aparte, esto es debido a que sí que incluyen archivos CSS, y al no tener muchos componentes en el proyecto se decidió incluirlos aparte.

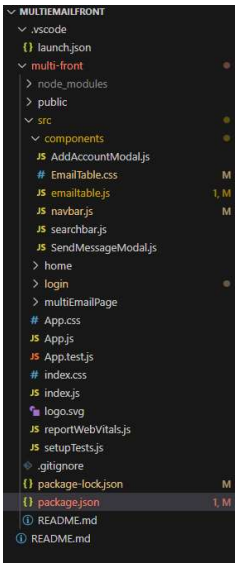

Figura 30: Estructura de archivos del front end

El archivo .jsx de cada componente lo podemos dividir en diferentes secciones para llevar a cabo la implementación del código de la interfaz. En este caso como secciones tenemos:

- Importaciones: En esta sección se declaran las dependencias, librerías o archivos CSS que necesitemos usar para desarrollar el componente. Como se aprecia en la *Figura 31* tenemos el import de react, es decir, de la propia librería de React con el "useState" y el "useEffect", que nos sirve para gestionar los estados en los componentes. Luego tenemos el import de Reactstrap, que es la biblioteca externa para usar componentes que ya están definidos para ahorrar tiempo. En último lugar tenemos la librería externa *Axios* que nos servirá para realizar las peticiones HTTP a nuestros *endpoints* que se encuentran en el *back end*.

Figura 31: Importación de librerías

- Definición de componentes: En esta otra sección se haría la declaración del componente propio que vamos a crear y todos los atributos que éste recibiría. Dentro de este componente al igual que todos los demás creados encontraremos el código JavaScript, que es el que se encargará de toda la lógica del *front end* con la declaración de las constantes, el uso de hooks[14], como el "useState" y el "useEffect" como ya se mencionó anteriormente, las peticiones a los *endpoints* usando axios, que se definieron en nuestro *back end* para diversas funcionalidades y el XML que se encontrará dentro del "return" del componente, que se encargará de renderizar los elementos que se hayan definido en el para la interfaz. En el XML podemos observar el uso de la librería ReacStrap, en este caso como ejemplo el componente FormGroup. Todo esto se puede apreciar en las *Figuras 32* y *33*.

```
const AddAccountModal = ({ isOpen, toggle, addAccount, IdCuenta }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [sslsend, setSslsend] = useState(false);
  const [sslreceive, setSslreceive] = useState(false);
  const [accountId, setAccountId] = useState(IdCuenta);
  const [enabledAccount, setEnabledAccount] = useState(true);
  const [host, setHost] = useState('');
  const [port, setPort] = useState('');
  const [receiveEnable, setReceiveEnable] = useState('');
  const [receiveSeenflag, setReceiveSeenflag] = useState(false);
  const [receiveDeleteflag, setReceiveDeleteflag] = useState(false);
  const [receiveProtocol, setReceiveProtocol] = useState('imap');
  const [sendHost, setSendHost] = useState('');
  const [sendPort, setSendPort] = useState('');
  const [sendProtocol, setSendProtocol] = useState('smtp');
  const [addEmailAccountresponse, setResponse] = useState('');

  useEffect(() => {
    setAccountId(IdCuenta);
  }, [IdCuenta]);

  const handleEmailChange = (e) => {
    setEmail(e.target.value);
  };

  const handlePasswordChange = (e) => {
    setPassword(e.target.value);
  };

  const handleAddAccount = async () => {
    try {
      const params = new URLSearchParams();
      params.append('id', '');
      params.append('emailAddress', email);
      params.append('password', password);
      params.append('accountId', accountId);
      params.append('enabledAccount', enabledAccount);
      params.append('receiveAccount.host', host);
      params.append('receiveAccount.port', port);
      params.append('receiveAccount.enable', true);
      params.append('receiveAccount.sslEnabled', sslreceive);
      params.append('receiveAccount.seenflag', receiveSeenflag);
      params.append('receiveAccount.deleteflag', receiveDeleteflag);
      params.append('receiveAccount.protocol', receiveProtocol);
      params.append('sendAccount.host', sendHost);
      params.append('sendAccount.port', sendPort);
      params.append('sendAccount.seenflag', true);
      params.append('sendAccount.deleteflag', false);
      params.append('sendAccount.protocol', sendProtocol);
      params.append('sendAccount.sslEnable', sslsend);

      const response = await axios.post('http://localhost:8081/mailAccount/add', params);

      if (response.data) {
        delete response.data.password;
```

Figura 32: Código de componente AddAccountModal

```
return (
  <Modal isOpen={isOpen} toggle={toggle}>
    <ModalHeader toggle={toggle}>Añadir Cuenta</ModalHeader>
    <ModalBody>
      <Form>
        <FormGroup floating>
          <Input type="email" name="email" id="email"  placeholder="Email"  onChange={handleEmailChange} />
          <Label for="email">Correo Electrónico</Label>
        </FormGroup>
```

Figura 33: Código de componente AddAccountModal

Para finalizar con la estructura, ya que el resto de archivos no tienen gran importancia, cabe destacar el archivo package-lock, que contendrá todas las dependencias instaladas en nuestra aplicación *React*. Como observamos en la *Figura 34* podemos ver todas las dependencias y sus versiones como *Bootstrap* y *Reacstrap* entre otras.

Figura 34: Dependencias del front end

7.1.2 Estructura e implementación del back end

En cuanto a la estructura del *back end*, nos encontramos con un mayor número de elementos debido a que aquí se maneja toda la lógica necesaria para poder crear cuentas nuevas, añadir las cuentas de correo y recibir y enviar correos.

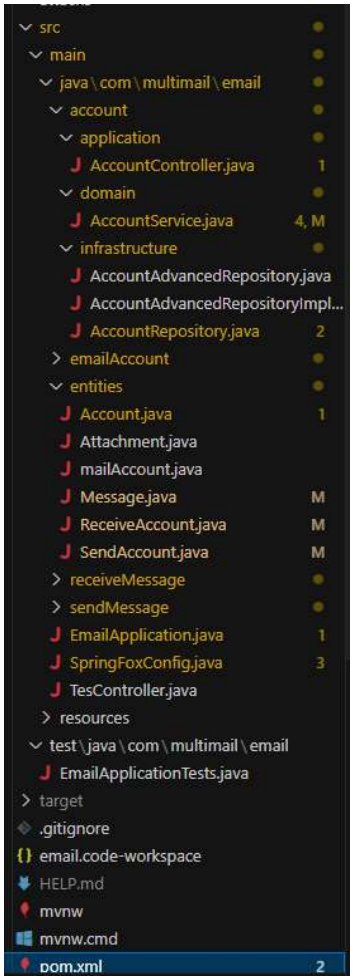

Figura 35: Estructura de archivos del back end

Para comenzar con la estructura y la implementación, hablaremos de las carpetas account, emailAccount, recevieMessage, sendMessage que se pueden observar en la *Figura 35*. Todas estas carpetas albergan dentro de ellas otra subestructura compuesta por tres carpetas que son:

- *Application*: En estas carpetas encontraremos un archivo .java que actúa como controlador, es decir, en este se recibirán las peticiones HTTP que se enviarán desde el *front end*, que este caso pueden ser de tipo *GET*, *POST*, *PUT* y *DELETE*. Después de recibir esta petición se decidirá la lógica de negocio a seguir. Más tarde el controlador enviará las respuestas al *front end* con los elementos necesarios para continuar la ejecución. Como ejemplo tenemos la *Figura 36,* que corresponde al controlador AccountController.java, que como se observa recibirá la petición HTTP en la ruta "http://localhost:8081/account/getId" , de tipo *GET*, que ejecutará el método *finById()* del servicio mailAccountService, devolviendo al *front end* una entidad de tipo Account.

```java
@RestController
@RequestMapping("/account")
@CrossOrigin(origins = "http://localhost:3000")
public class AccountController {

    @Autowired
    AccountService mailAccountService;


    @GetMapping("/getId")
    public Account getAccountById(@RequestBody String id){

        return mailAccountService.findById(id);

    }
```

Figura 36: Código de la clase controlador AccountController

- *Domain*: En este archivo encontraremos una capa que actúa como servicio, en la cual se encontrará la lógica de negocio que se ejecutará después de que sea llamada por el controlador. Como ejemplo tenemos la clase *ReceiveSchedulingService*, cuya función será la de crear y cancelar los hilos de ejecución para cada cuenta de correo electrónico que se añada, para que así la aplicación pueda estar recibiendo correos electrónicos de todas las cuentas que estén registradas en la cuenta del usuario. Todo esto lo podemos ver en la *Figura 37*.

```java
@Slf4j
@Component
@RequiredArgsConstructor
public class ReceiveSchedulingService {


    private long checkDelaySec = 15;
    private ScheduledThreadPoolExecutor scheduler = (ScheduledThreadPoolExecutor) Executors.newScheduledThreadPool(corePoolSize:10);
    private Map<String, ScheduledFuture<?>> clients = new HashMap <String, ScheduledFuture<?>>();

    @Autowired
    private ReceiveMailRepository receiveMailRepository;

    public void scheduleAccount (mailAccount account){

        MultiReceiveEmails client = new MultiReceiveEmails(account, receiveMailRepository);

        ScheduledFuture<?> executeReceiveEmails = scheduler.scheduleWithFixedDelay(client, initialDelay:0, checkDelaySec, TimeUnit.SECONDS);
        clients.put (account.getId(), executeReceiveEmails);

        System.err.println("Sheduled account () to check for new mails every () seconds" +  account.getId() + " : " +  checkDelaySec);
    }

    public void cancelAccountschedule(mailAccount account){
        var client = this.clients.remove(account.getId());
        if (client == null) return;
        client.cancel(mayInterruptIfRunning:true);

        System.err.println("Unscheduled account (), it will not check for new mails anymore" + account.getId());
    }
}
```

Figura 37: Código de la clase de servicio ReceiveSchedullingService

- *Infrastructure*: En esta carpeta nos encontraremos los archivos necesarios para poder trabajar con la base de datos de *MongoDB*. Los archivos que nos encontramos aquí son interfaces de *MongoRepository*, por lo tanto, los métodos de guardar, eliminar, crear, entre otros, los tendremos disponibles sin tener que crear nada. En la *Figura 38* tenemos el ejemplo de *AccountRepository*, que tendrá un solo método definido ya que como se ha comentado ya tenemos los métodos de la interfaz y como se puede observar, se ha definido un método nuevo, en este caso para filtrar las cuentas de la aplicación por su nombre de usuario y no por la id de la entidad, que sería el método que nos daría por defecto esta interfaz.

```java
import java.io.Serializable;
import java.util.Optional;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

import com.multimail.email.entities.Account;

@RepositoryRestResource(exported = false)
public interface AccountRepository extends MongoRepository<Account, Serializable>, AccountAdvancedRepository {

    Account findByUsurname(@Param("usurname") String usurname);

}
```

Figura 38: Código de clase repositorio Account Repository

Continuando con la estructura de nuestro proyecto, nos encontramos con la carpeta de entidades. Las entidades son las estructuras de datos con las que trabajaremos en nuestro proyecto, es decir, en el caso de *Java* son los objetos y en este caso estos objetos están asociados a una colección de *MongoDB*, el equivalente a una tabla en una base de datos relacional. Para ejemplificar analizaremos la *Figura 39*, en la cual aparece la entidad *mailAccount*. Lo primero que nos encontramos es la anotación *Document* de *Spring Boot*, que nos ayudará a mapear la entidad en la colección *mailAccounts*, usando esta anotación en el resto de entidades para que se pueda guardar en la base de datos. A Continuación se declararán más anotaciones como *NoArgsConstructor* y *Data*, que se emplearán para ahorrarnos tener que declarar los métodos *GET* y *SET,* que devuelven los o asignan los valores de los atributos de la entidad, o evitarnos crear los constructores de la entidad. En la *Figura 40* vemos cómo queda la colección en la base de datos, estando este almacenado como si fuese un JSON.

```java
@Document(collection = "mailAccounts")
@NoArgsConstructor
@AllArgsConstructor
@Data
public class mailAccount {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private String id;

    private String emailAddress;

    private String password;

    private String accountId;

    private Boolean enabledAccount;

    private ReceiveAccount receiveAccount;

    private SendAccount sendAccount;


}
```

Figura 39: Código de clase entidad mailAccount

Figura 40: Representación de MongoDB Compass de una entidad de tipo mailAccount

Para finalizar cabe destacar el fichero pom.xml, donde encontraremos la configuración del proyecto en *Maven*[15], en el que se añadirán todas las dependencias de nuestro proyecto, como en el caso de nuestro proyecto puede ser la API de *Jakarta Mail*. En la *Figura 41* podemos apreciar cómo quedaría la configuración dentro del archivo pom.xml.

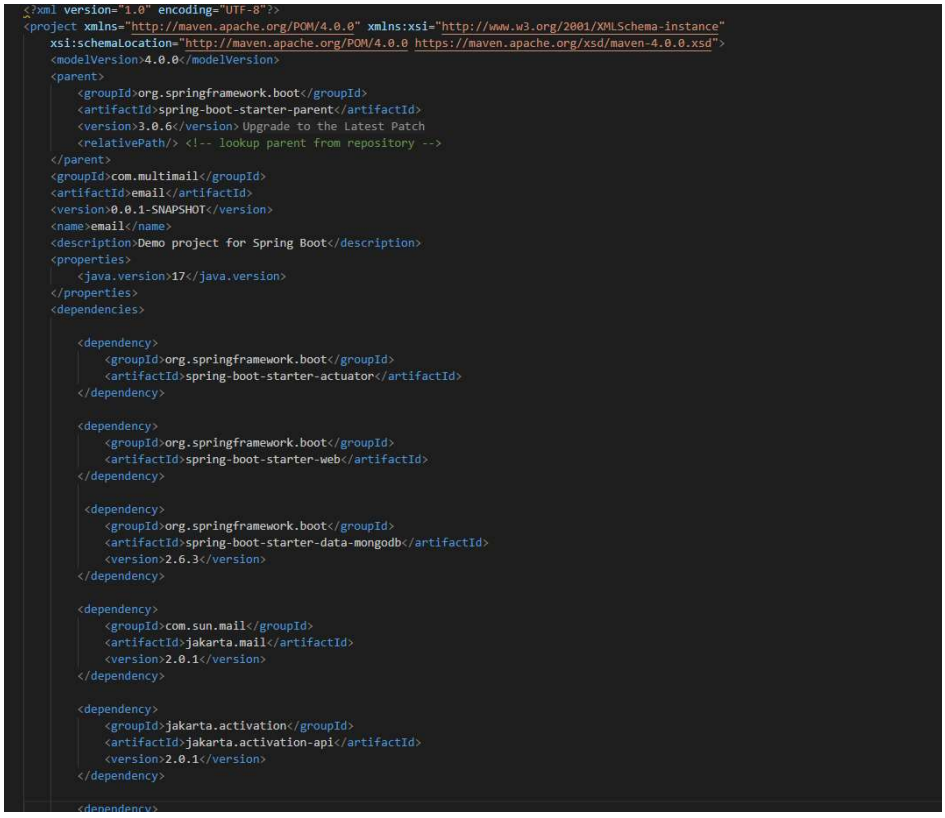

Figura 41: Archivo de configuración pom.xml

8. Aplicación Final

Una vez finalizado el desarrollo nos queda la aplicación final, por lo que este apartado tratará de explicar cómo ha explicar cómo ha sido el resultado final del desarrollo, mediante imágenes de la interfaz y cuál es su comportamiento al interactuar con ella. Estas imágenes están asociadas a algunas de las funcionalidades de la aplicación. que son:

- Iniciar Sesión: Esta parte de la interfaz será lo primero que visualizará el usuario al entrar a la web. En esta nos encontramos dos formularios que son el usuario y la contraseña para poder acceder. Si ya te has registrado previamente bastará con introducir ambos campos y pulsar el botón para poder entrar en la pantalla principal de la aplicación. En caso de no disponer una cuenta se podrá pulsar el botón para registrarte el cual abrirá una ventana emergente como la de la *Figura 42*, en la que nos encontraremos un formulario con los campos necesarios para poder registrarse en la aplicación.

Figura 42: Interfaz iniciar sesión de la aplicación final

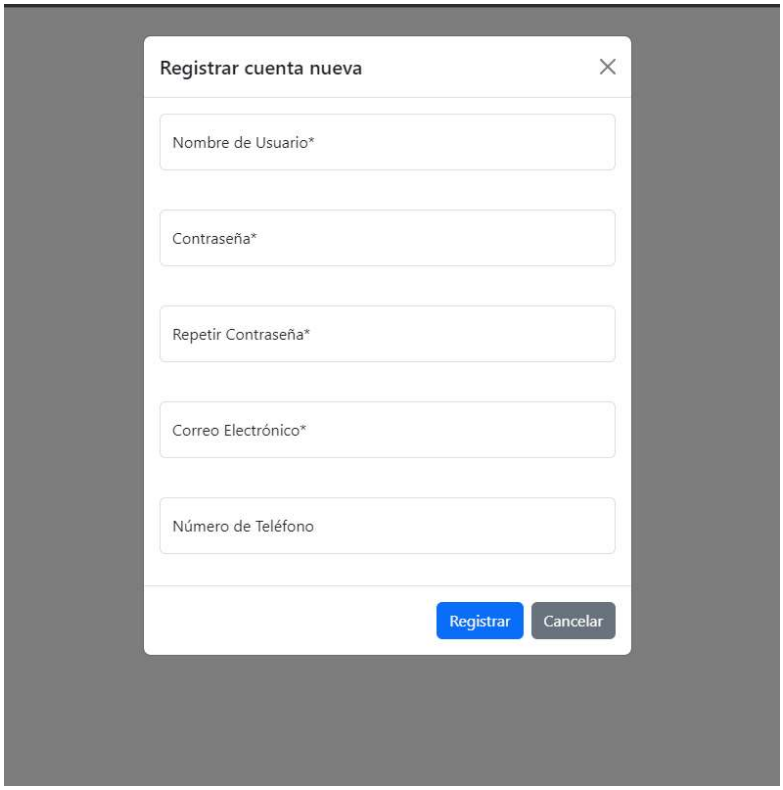

Figura 43: Interfaz registrar cuenta nueva de la aplicación final

- Pantalla Principal: Después del inicio de sesión nos encontramos la pantalla principal en la cual se podrán visualizar varios botones que son parte de la funcionalidad de la aplicación, como el botón de mandar mensajes, registrar cuentas entre otros. Si el usuario ya ha registrado una cuenta de correo electrónico, podrá visualizar los correos electrónicos que se recuperaron del servidor de correo electrónico.

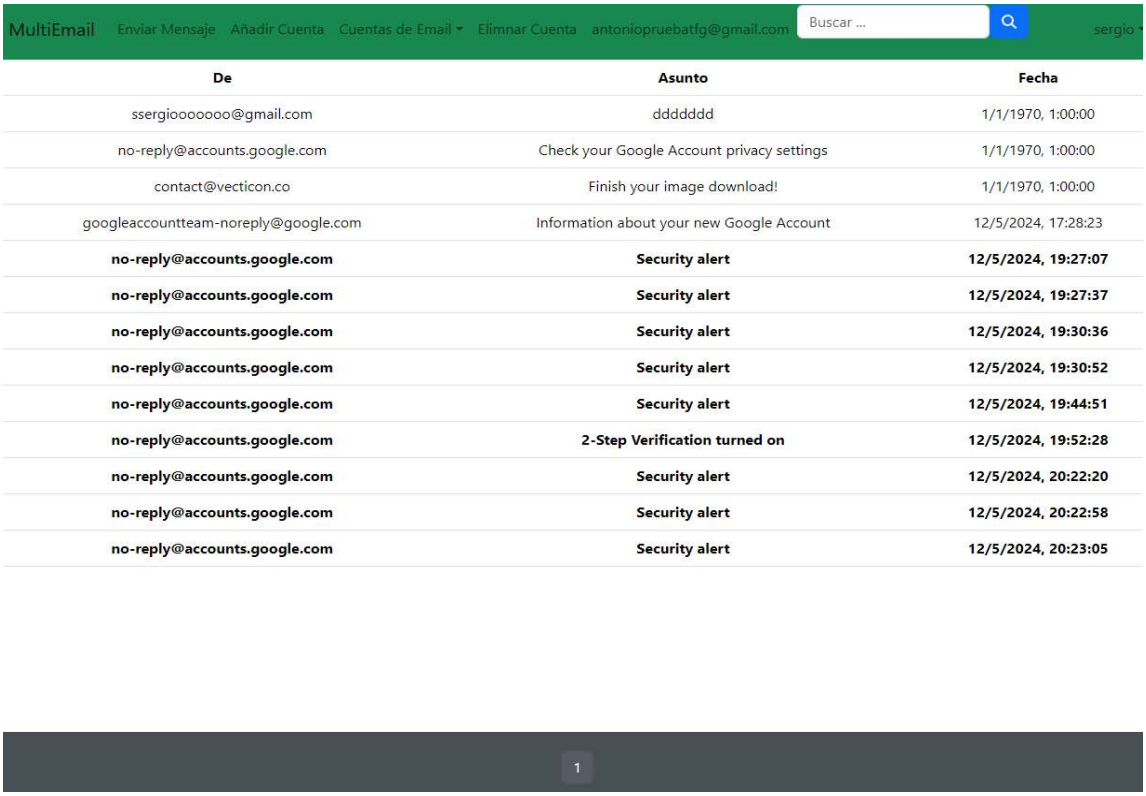

Figura 44: Interfaz de la pantalla principal de la aplicación final

- Visualizar mensaje: Para visualizar los mensajes el usuario pulsará sobre uno de ellos lo que provocará que se abra una ventana donde podremos visualizar la información del mensaje y además los botones para marcar el mensaje y eliminar el mensaje.

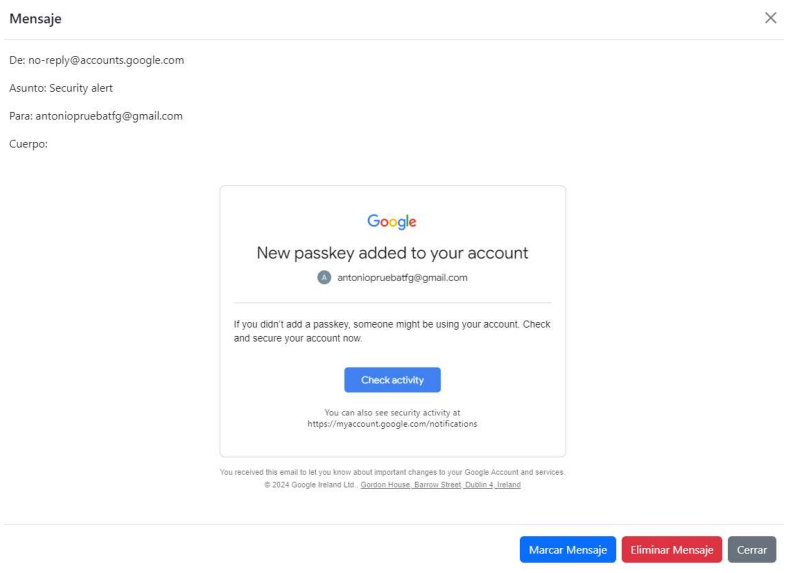

Figura 45: Interfaz visualizar mensaje de la aplicación final

- Enviar Mensaje: Para poder enviar un correo electrónico, el usuario debe pulsar el botón para enviar un mensaje, el cual abrirá una ventana emergente con el formulario necesario para poder enviar el mensaje.

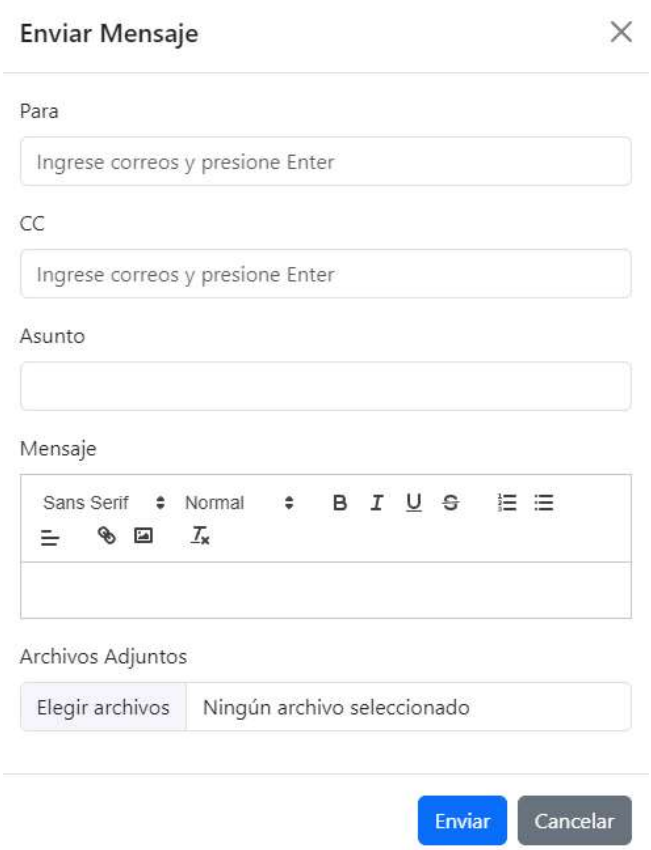

Figura 46: Interfaz enviar mensaje de la aplicación final

- Añadir cuenta: Esta funcionalidad es usada por el usuario al apretar el botón para añadir la cuentas de correo electrónico cuando se encuentra en la ventana principal, desplegándose una ventana emergente que contiene un formulario con los campos necesarios para poder añadir una cuenta de correo electrónico.

Figura 47: Interfaz añadir cuenta de correo de la aplicación final

- Datos de la Cuenta: Al apretar el usuario de la cuenta en la pantalla principal, desplegamos un menú con tres opciones, las dos primera nos permitirán visualizar los datos de la cuenta, pudiendo una vez que se despliegue la ventana emergente al pulsarlos, cambiar los datos almacenados de la cuenta mediante un formulario que en el caso de la venta de información nos mostrará los datos almacenados y en el caso de cambiarlos y pulsar sobre el botón para guardar los cambios, se editaran el valor de estos en la base de datos, y en el caso de cambiar contraseña será igual pero no se cargará ningún dato previamente. Esto se puede observar en las *Figuras 48* y *49*.

Figura 48: Interfaz opciones cuenta de la aplicación final

Figura 49: Interfaz información de la cuenta de la aplicación final

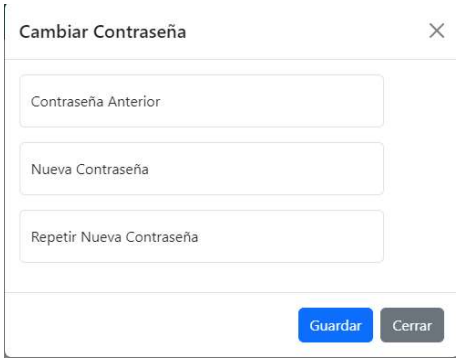

Figura 50: Interfaz cambiar contraseña de la aplicación final

9. Conclusiones

Tras el desarrollo del proyecto se ha podido observar cómo tener una metodología correcta para tu proyecto puede suponer el finalizarlo exitosamente o no, ya que esta nos irá acompañado durante todo el proceso y por lo tanto nos respaldará en las decisiones que tomemos. No obstante durante el desarrollo siempre suelen salir imprevistos, incluso aunque se haya tenido una buena metodología, ya que al final una aplicación puede tener muchos elementos que desarrollar que no se han tenido en cuenta en las fases iniciales de planteamiento de los requisitos, que tendrán que ir cambiando durante el desarrollo del proyecto pero que con la correcta metodología se pueden ir solucionando.

Otro de los factores claves que se ha observado que concede el éxito del desarrollo es la selección correcta de las herramientas y tecnologías empleadas, como *React*, que en este caso se ha aprendido solo para el desarrollo de este proyecto, permite una curva de aprendizaje buena y sobre todo una implementación de la interfaz también rápida, debido a que permite el uso de librerías como *BootStrap* y *Reacstrap*, ayudando a agilizar la integración de los elementos de la interfaz ya que estos vienen predefinidos. No obstante estas librerías nos dan menos libertad para personalizar las aplicaciones a nuestro gusto debido a sus limitaciones. El uso de Java con el *Framework Spring Boot* con el que ya estaba familiarizado ha facilitado mucho el desarrollo del *back end*, permitiendo ahorrar mucho tiempo de configuraciones de la base de datos, los controladores, la creación de entidades, mediante el uso de anotaciones. Y como no comentar que la compatibilidad de React y Java para desarrollar un proyecto ya que durante el proceso de desarrollo no se ha tenido ningún problema con la interacción de ambos.

Por lo tanto como conclusiones obtenemos que el uso de una metodología y de la selección correcta de herramientas y tecnologías permitirá no solo finalizar el proyecto de una forma adecuada, si no que agilizará el proceso de este.

9.1 Trabajo a futuro y líneas de mejora

Este proyecto aunque esté finalizado, todavía no está exento de en un futuro ir añadir nuevas funcionalidades y resolver los bugs, ya que la interfaz que se presenta es muy simple, y carece de funcionalidades que otras aplicaciones de correo electrónico que sí que tienen, como poder almacenar los correos en carpetas distintas filtrando el tipo de correo que quieres guardar, algunas opciones de envío que faltan, poder visualizar de forma correcta los mensajes que son respuestas a otros, poder responder a un mensajes sin tener que enviar uno nuevo y algunas integraciones en código. Estas integraciones son a causa de algunos proveedores de correo que para poder acceder a sus servidores requieren obtener claves de acceso mediante el uso de su API.

Bibliografía

[1] ¿Qué son los microservicios de Spring boot?. Fuente: https://www.qindel.com/que-son-los-microservicios-spring-boot/ (Consultado: 15 de Junio de 2024)

[2] ¿Qué es ajax? y¿qué tecnologías utiliza?. Fuente: https://aws.amazon.com/es/what-is/ajax/ (Consultado: 15 de Junio de 2024)

[3] ¿Qué son las metodologías de desarrollo software? y ¿qué tipos hay?. Fuente: https://www.valtx.pe/blog/metodologias-para-el-desarrollo-de-software-que-son-y-para-que-sirven (Consultado: 15 de Junio de 2024)

[4] ¿Qué es el back end?. Fuente: https://www.gluo.mx/blog/backend-que-es-y-para-que-sirve (Consultado: 22 de Junio de 2024)

[5] ¿Qué es el desarrollo incremental?. Fuente: https://mwebs.com.uy/blog/qu%C3%A9-es-el-desarrollo-incremental/23 (Consultado: 21 de Junio de 2024)

[6] ¿Qué son los requisitos funcionales?. Fuente: https://visuresolutions.com/es/blog/functional-requirements/ (Consultado: 21 de Junio de 2024)

[7] ¿Qué es una base de datos no relacional?. Fuente: https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/ (Consultado: 16 de Junio de 2024)

[8] ¿Qué es BSON?. Fuente: https://cursa.app/es/pagina/comprender-bson-en-mongodb (Consultado: 15 de Junio de 2024)

[9] ¿Qué son los protocolos de correo? y ¿qué protocolos hay?. Fuente: https://www.siteground.es/tutoriales/email/protocolos-pop3-smtp-imap/ (Consultado: 15 de Junio de 2024)

[10] ¿Qué son los clusters?. Fuente: https://www.salesforce.com/mx/blog/clusteres/#:~:text=%C2%BFQu%C3%A9%20significa%20clusterizar%3F,objetivo%20de%20potenciar%20su%20eficiencia. (Consultado: 5 de Septiembre de 2024)

[11] Base de datos instalada como standalone. Fuente: https://xtrmlobo29.blogspot.com/2010/10/tipos-de-bases-de-datos.html (Consultado: 5 de Septiembre de 2024)

[12] ¿Qué es el front end?. Fuente: https://www.arsys.es/blog/frontend-que-es-y-para-que-se-utiliza-en-desarrollo-web (Consultado: 22 de Junio de 2024)

[13] ¿Qué es GitLab? y ¿para qué se utiliza?. Fuente: https://formadoresit.es/que-es-gitlab-y-para-que-sirve/ (Consultado: 10 de Julio de 2024)

[14] ¿Qué son los hooks en react?. Fuente:
https://adictosaltrabajo.com/2024/04/19/guia-de-hooks-react-18/?utm_term=hook%20react&utm_campaign=SA+%7C+ES+%7C+adictosaltrabajo.com&utm_source=adwords&utm_medium=ppc&hsa_acc=9710751667&hsa_cam=21459028756&hsa_grp=163157053614&hsa_ad=705391841945&hsa_src=g&hsa_tgt=kwd-782421411492&hsa_kw=hook%20react&hsa_mt=b&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjwreW2BhBhEiwAavLwfMF6kFtd0gWCqaPeF6pigy2pfIOPfO0mGKzC1khkWGKsNqu8VDOx1BoCIBgQAvD_BwE
(Consultado: 17 de febrero de 2024)

[15] ¿Qué es Maven?. Fuente:
https://www.campusmvp.es/recursos/post/java-que-es-maven-que-es-el-archivo-pom-xml.aspx
(Consultado: 22 de Junio de 2024)

Apéndice A

ANEXO: OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**		x		
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**				x
ODS 8. **Trabajo decente y crecimiento económico.**		x		
ODS 9. **Industria, innovación e infraestructuras.**				x
ODS 10. **Reducción de las desigualdades.**				x
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. **Producción y consumo responsables.**				x
ODS 13. **Acción por el clima.**				x
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En la actualidad, el correo es una de las formas más utilizadas de comunicación entre empresas. Sin embargo, manejar múltiples cuentas puede resultar estresante y ser una carga innecesaria en cuanto a tiempo y energía se refiere. Una aplicación diseñada de manera que haga más sencillo el proceso de recibir y enviar correos puede tener no solo un impacto en la productividad, sino también en el bienestar de la persona usando la aplicación. Estos productos tecnológicos son indirectamente beneficiosos para los Objetivos de Desarrollo Sostenible ODS 3 sobre la Salud y el Bienestar, y 8 sobre Trabajo Decente y Crecimiento Económico.

La meta del ODS N° 3 es 'asegurar una vida saludable' y 'promover el bienestar para todos en todas las edades'. Aunque por lo general se refiere a la salud física del individuo, incluye el bienestar emocional y mental de la misma importancia. Como toda la información disponible para una persona promedio está en aumento, la sobrecarga informativa ha sido una preocupación común en entornos laborales y personales. La constante atención a las diversas bandejas de entrada de correo para vigilar la acumulación de mensajes y notificaciones en un intento por borrar el espacio puede resultar en fatiga mental y presión interna. Una aplicación que simplifique la gestión de varias cuentas y ayude a mantener una recopilación similar de correos nuevos ayuda de manera efectiva a aliviar la carga mental. Con ella, una persona sentirá menos presión y, por lo tanto, tendrá menos probabilidad de experimentar fatiga mental.

Además, si una aplicación es fácil de aprender y de usar, reduce la frustración que viene de los programas complejos, creando una mejor experiencia mejor con menos estrés. En otras palabras, cuanto más simple sea el uso de una solución, más tiempo tendrán los usuarios para otras actividades que los hacen felices: podrán pasar menos tiempo en los papeles y más tiempo descansando o socializando.

Otra relación es en el objetivo 8: promover "el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo y el trabajo decente para todos." Es un objetivo directo para la productividad. Al permitir que los trabajadores administren sus correos electrónicos sin problemas, los empleadores optimizan la eficiencia. Gestionar múltiples correos electrónicos consumirá menos tiempo, lo que significa que los trabajadores podrán dedicarse a actividades de más valor, siendo una ventaja para la empresa.

Un aumento de la productividad sin desgaste mental es un factor crucial para el bienestar en el trabajo. La administración adecuada de las comunicaciones a través del correo electrónico puede mitigar la multitarea, una de las razones más frecuentes de la fatiga mental en el trabajo. Al fusionar todas las cuentas en una sola plataforma, los empleados tienen una perspectiva clara de sus responsabilidades y la capacidad de priorizar tareas. Este aspecto no solo optimiza las operaciones laborales, sino que también ayuda a lograr un ambiente laboral más saludable y equilibrado.

En conclusión, una aplicación de gestión de la bandeja de entrada del correo electrónico es una herramienta que beneficia a los ODS número 3 y 8. En este caso, se reduce el estrés y se aumenta la eficiencia laboral, mejorando tanto la salud mental de los clientes como su desempeño laboral. Una herramienta digital correctamente simple y efectiva influye no solo

en un funcionamiento optimizado, sino en la consecución de objetivos cruciales de desarrollo sostenible.