Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Proyecto Fin de Grado

Diseño y desarrollo de una aplicación web para la búsqueda de empleo de personas con discapacidad utilizando Next.js

Grado Ingeniería del Software

Curso 2024 - 2025

Autor: Jiale Wang

Tutor: Borja Bordel Sánchez

Madrid, Mayo de 2025

Resumen

El mercado laboral siempre ha sido un tema que ha acaparado la atención. Sin embargo, a medida que la sociedad ha ido evolucionando, en lugar de beneficiarse del avance, la situación del mercado laboral se ha vuelto más complicada y especialmente para el colectivo de personas con discapacidad. Cabe destacar que la tasa de empleo de las personas con discapacidad en España mostró una evolución creciente, no obstante, con una velocidad moderada y en los últimos años, se observó una tendencia de disminución progresiva en la tasa de contratación de personas con discapacidad.

Asimismo, las plataformas de búsqueda de empleo son herramientas más comunes que se utilizan para conocer las ofertas de trabajo disponibles en el mercado. A pesar de que actualmente existe multitudes plataformas, en común se presentan limitaciones para las personas con discapacidad y son poco accesibles.

Con el propósito de mejorar los problemas anteriores, se ha propuesto el presente proyecto de fin de grado, cuyo objetivo principal es el diseño y desarrollo de una aplicación web orientada a la búsqueda de empleo de personas con discapacidad. En ***CoWorld***, los usuarios podrán encontrar trabajos que se adapten mejor tanto en sus condiciones e intenciones. De igual forma, para parte de las empresas, obtendrá una gestión más eficiente de los candidatos inscritos en sus ofertas publicadas, además de poder agregar operadores para asistir en dicha gestión.

Al finalizar el proyecto, se espera que la aplicación web ***CoWorld*** se convierta en una plataforma útil y eficiente al frente de reducir la proporción de personas con discapacidad en paro, y ayudar a los usuarios, tanto candidatos como empresas a alcanzar sus objetivos de forma eficaz, lo que está en consonancia con la misión de ***CoWorld***: **Un Mundo Colaborativo puede llegar todo lo posible.**

Abstract

The labour market has always been a subject of public attention. However, despite the significant societal progress, the labour market has not benefited from this progress. Indeed, the situation has become more complex, particularly for people with disabilities. It is important to highlight that, although there has been an increase in the employment rate of people with disabilities in Spain, the pace has remained moderate. Moreover, in recent years, a decline in the rate of contracts of this collective has also been observed.

Likewise, job-hunting platforms have become one of the most common tools for exploring available job offers. Although a large amount of platforms exist today, in general, they present limitations for people with disabilities and lack of accessibility.

With the goal of improving the previously mentioned issues, this final degree project proposes the design and development of a job search web application dedicated to people with disabilities. In ***CoWorld***, users will be able to find job offers that best match their conditions and intentions. In the same way, companies will benefit from more efficient management of candidates subscribed to in their job publications, and more, they will also have the opportunity to add operators to assist the management.

Upon completion, the ***CoWorld*** web application is expected to become a useful and efficient platform aimed at reducing the unemployment rate of people with disabilities. It will help both candidates and companies achieve their goals efficiently, which aligns with the mission of ***CoWorld***: A Collaborative World Can Unlock All Possibilities.

Índice

Índice de figuras

1. Introducción

1.1. Importancia de la inclusión social en empleo

Antes de hablar la importancia de la inclusión social, es esencial comprender qué se entiende por este concepto. Según la definición dada por ONU (Organización de las Naciones Unidas), “La inclusión social asegura que todas las personas sin distinción puedan ejercer sus derechos y garantías, aprovechar sus habilidades y beneficiarse de las oportunidades que se encuentran en su entorno.” (Comisión Económica para América Latina y el Caribe [CEPAL], 2018). El concepto empieza aparecer frecuentemente a nivel global a partir de los años 90, con una visión amplia que incluye distintos tipos de inclusión. En el proyecto se centra en la inclusión social de las personas con discapacidad.

El objetivo principal de este tipo de inclusión consiste en nadie se sienta excluido ni dejado atrás. Existe varias formas para contribuir a dicho objetivo, una de ellas es promover la contratación de personas con discapacidad. De esta manera, se impulsa un entorno laboral inclusivo y equitativo, en la cual la discapacidad no debe convertirse una barrera que les impida el desarrollo profesional, sino se busca que todas las personas tengan las mismas oportunidades de encontrar un empleo adecuado.

Cabe mencionar que el enfoque de mejorar la inclusión social para las personas con discapacidad a través del empleo está presente tanto en la meta 8.5 de la Agenda 2030: “De aquí a 2030, lograr el empleo pleno y productivo y el trabajo decente para todas las mujeres y los hombres, incluidos los jóvenes y las personas con discapacidad, así como la igualdad de remuneración por trabajo de igual valor” (Naciones Unidas, s.f.). , como en la meta 10.2: “De aquí a 2030, potenciar y promover la inclusión social, económica y política de todas las personas, independientemente de su edad, sexo, discapacidad, raza, etnia, origen, religión o situación económica u otra condición” (Naciones Unidas, s.f.). Asimismo, es posible conocer la importancia de promover la inclusión social para alcanzar una planeta más justo y sostenible, ya que todas las mejoras comienzan con una vida mejor. Apoyar a las personas con discapacidad para que obtengan un trabajo decente contribuye a garantizar su independencia financiera, lo cual representa un punto de partida para mejorar vida de ellos.

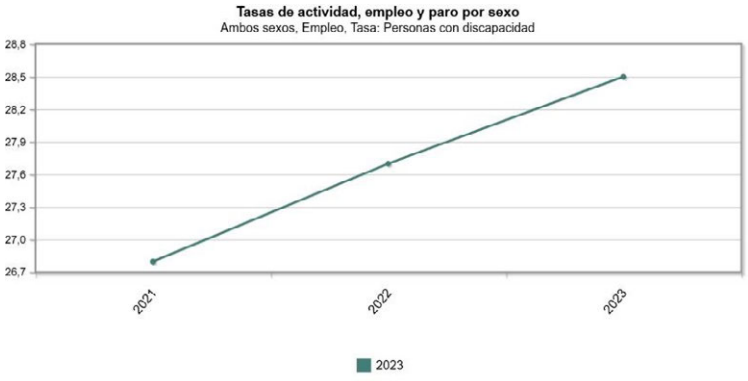
1.2. Problemática

Gracias al Informe del Mercado de Trabajo de las Personas con Discapacidad 2025 publicado por el SEPE (Servicio Público de Empleo Estatal, 2025), donde se abordan claramente las situaciones y estadísticas actuales del mercado laboral de las personas con discapacidad, y por la misma razón, las problemáticas actuales son extraídas en dicho informe.

En primer lugar, se destaca un crecimiento en la tasa de empleo de las personas con discapacidad. Sin embargo, el ritmo ha sido relativamente moderado y limitado, manteniéndose el porcentaje por debajo del 30%. (véase Figura 1). Además, la tasa de contratación en los últimos años presenta una tendencia decreciente. (véase Figura 2).

Figura 1

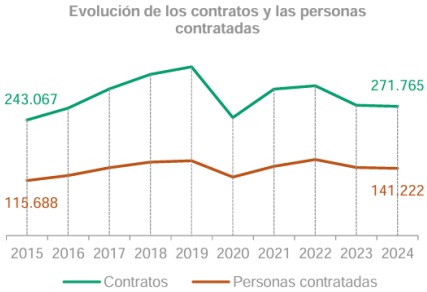
Tasa de Empleo de las Personas con Discapacidad



Nota. Adaptado de *El empleo de las personas con discapacidad* [Gráfico], por INE Instituto Nacional de Estadística, 2023, (https://www.ine.es/jaxi/Datos.htm?tpx=49172#_tabs-grafico). CC BY 4.0

Figura 2

Evolución de los contratos y las personas



Nota. Adaptado de *Evolución de los principales indicadores de empleo* (p. 25), por el Observatorio de las Ocupaciones del SEPE, 2025, SEPE.

De forma paralela, dentro de la tasa de contratación, se observa que el porcentaje de contratos de tipo temporal (68.43%) es más del doble que el de contratos de tipo indefinidos (31.57%), a partir de la cual, refleja la posibilidad de la situación de inestabilidad laboral.

El informe también abarca un estudio sobre las ocupaciones más contratados para este colectivo y se destacaron las siguientes cinco ocupaciones: Personal de limpieza (43.019) un 15,83 %, Peones de las industrias (19.438), Camareros asalariados (12.217), Conserjes de edificios (8.519) y Otro personal de limpieza (8.461).

Como se puede observar, las ocupaciones se concentran principalmente en los sectores de Servicios e Industria. Esto muestra la necesidad de ofrecer más variedades de opciones de trabajo que se ajusten a los intereses y condiciones individuales. Hay que seguir ofreciendo los tipos de demandas populares, pero también es importante evitar limitarlas solo a estos sectores habituales.

Complementariamente, según la noticia publicada por la Universidad de Sevilla, se comprende que "Un 60,6% del estudiantado con discapacidad (esto es casi 2 de cada 3) considera que tendrá más dificultades que el resto de sus compañeros/as en este sentido." (Universidad de Sevilla, 2023). Al reflexionar sobre esta información junto con el problema anterior, se puede inferir que este colectivo necesita nuevas oportunidades y posibilidades laborales que alineen mejor a sus estudios y con las profesiones que desean ejercer, más allá de las ocupaciones más comunes.

Por otra parte, es cierto que, actualmente existen múltiples plataformas de búsqueda de empleo. Sin embargo, tras de haber investigado algunas de ellas desde la perspectiva de una persona con discapacidad, se resulta que es bastante difícil encontrar información sobre ofertas de trabajo orientadas a este colectivo. Incluso cuando es posible de encontrar algunas ofertas, la información detallada que se proporciona suele ser ambigua o incompleto, lo cual complica el usuario a saber si la oferta se ajusta a sus condiciones y necesidades. En resumen, las plataformas actuales presentan dificultad en la usabilidad y accesibilidad para las personas con discapacidad y la experiencia negativa puede llevar a que muchas de ellas opten por dejar de seguir la búsqueda de trabajo.

1.3. Aproximación a la resolución

Al responder el objetivo de fomentar la tasa de empleo de las personas con discapacidad y ofrecerles una mejor experiencia durante la búsqueda de empleo, se propone el diseño y desarrollo de una aplicación web de búsqueda de empleo exclusiva

para este colectivo. Al ofrecer una plataforma especializada, se elimina la necesidad de invertir tiempo como en otras plataformas habituales, donde cuesta de encontrar trabajos que ofrece a las personas con discapacidad debido la mezcla con otros trabajos que no son para ellos. Además, las empresas podrán incluir en sus publicaciones de empleo las condiciones e informaciones relacionadas con la discapacidad, lo que permitirán a los usuarios que busquen trabajo filtrar los empleos que se adapten mejor a sus condiciones y necesidades. Con respecto a la accesibilidad de la aplicación, la interfaz gráfica se diseñará y gestionará bajo el principio de simplicidad, con el fin de facilitar el uso de la aplicación y ofrecer una experiencia lo más satisfactoria posible.

Las siguientes son funcionalidades que ofrece la aplicación:

Para los candidatos (usuarios que busquen trabajo):

- La aplicación contara un grupo de filtros, que permite los candidatos buscar y filtrar las publicaciones de trabajos disponibles en la aplicación por las condiciones especificadas, obteniendo así los resultados que les adapten mejor y reducir tiempo en la búsqueda.
- Se ofrecerá una gestión eficiente de las ofertas de trabajos guardadas o solicitadas, donde podrán conocer claramente el estado de las ofertas aplicadas y guardar las ofertas interesadas para facilitar la búsqueda posterior.
- El perfil de los candidatos juega un papel fundamental para ellos, donde pueden especificar sus informaciones. Las informaciones del perfil, por un lado que se usarán para realizar el cálculo del algoritmo de recomendación de ofertas de empleo que les ajusten. Por otro lado, se usarán como currículum del candidato, que se enviará a la empresa cuando aplique una oferta.

Para las empresas:

- La aplicación dispondrá de una gestión de publicaciones de empleo, lo que ayudará a crear y modificar de una oferta de la empresa.
- La aplicación contará también una gestión eficiente de los candidatos inscritos en sus ofertas de trabajo. Permitirá visualizar el currículum del candidato y cambiarle el estado, lo que simplificará la administración de las decisiones relativa a los candidatos.

- De la misma manera que los candidatos, las empresas podrán especificar sus informaciones en el perfil, la cual se mostrará en la página de información de la empresa.
- Por lo último, la aplicación ofrecerá una gestión de operadores para empresas. Los operadores tendrán las mismas funcionalidades que la empresa, excepto la gestión del perfil de la empresa y la gestión de operadores. Esto agilizará el proceso de administración de ofertas de empleo y el proceso de selección de los candidatos.

Gracias estas funcionalidades, la aplicación podrá ayudar a las personas con discapacidad a encontrar trabajos adaptados y acelerar el crecimiento de la tasa de empleo de este colectivo. Es más, proporciona una plataforma que facilita a las empresas la selección de candidatos. Actualmente existe muchas empresas que apoyan a los empleados discapacitados, solo que faltaba una plataforma que dejara a los candidatos conocer la empresa. En este sentido, la aplicación CoWorld promueve una conexión directa entre las empresas y persona con discapacidad, y contribuye de manera tangible a un mundo equitativo y colaborativo.

1.4. Lista de objetivos

Para desarrollar la aplicación web que facilita las personas con discapacidad en el proceso de búsqueda de empleo y a las empresas en el proceso de selección de candidatos se han definen los siguientes objetivos:

1. **Conocer el contexto actual e identificar los problemas existentes:**
    a. Realizar un estudio en profundidad sobre la situación actual del mercado laboral de las personas con discapacidad en España e investigar los objetivos relacionadas con la Agenda 2030.
    b. Conocer por práctica, la experiencia de búsqueda de empleo como una persona con discapacidad en las plataformas existentes y analizar las posibles mejoras y problemas.
2. **Elección de las herramientas y el entorno de desarrollo:**
    a. Estudiar las pilas populares que se usan para desarrollar aplicaciones web y conocer las herramientas y técnicas que usa cada una. Elegir una y justificar la elección realizada.
3. **Definir requisitos de la aplicación:**

a. Identificar las funcionalidades que debe tener la aplicación, definirlas como requisitos a implementar.

4. **Diseño e implementación de la aplicación:**
    a. Diseñar una interfaz gráfica simple e intuitiva para la aplicación y realizar el diseño planificado. (Front-End)
    b. Estructurar la base de datos de la aplicación e implementar la lógica de negocio definida en los requisitos. (Back-End)

    En concreto, en la parte de implementación de la aplicación, se divide en los siguientes subobjetivos específicos:

        i. Desarrollar de un sistema de autenticación, que permita el iniciar y cerrar sesión de usuarios con diferentes roles y gestionar de forma eficiente el token del usuario para garantizar la seguridad de la aplicación.
        ii. Desarrollar una gestión de perfiles que permita tanto a los candidatos como a las empresas añadir sus informaciones esenciales.
        iii. Desarrollar de una gestión de ofertas de empleo que permita a las empresas administrar sus ofertas de manera eficiente y a los candidatos filtrar, guardar, aplicar y conocer estado de las ofertas.
        iv. Desarrollar de una gestión de candidatos que permita a las empresas conocer a los candidatos inscritos en cada una de sus ofertas y cambiar el estado de los candidatos para facilitar el proceso de selección.
        v. Desarrollar de una gestión de operadores que permita a las empresas administrar sus operadores, de modo que los operadores agregados puedan asistir a la empresa en la gestión de ofertas y el proceso de selección de candidatos, lo que mejoraría la eficiencia del proceso de contratación.

5. **Probar y validar la aplicación:**
    a. Comprobar que la aplicación funciona correctamente e se han implementado todos los requisitos funcionales definidos.
    b. Validar que la aplicación funciona como esperaba en el diseño.

c. Garantizar que la aplicación pueda lograr una accesibilidad de alta calidad.

6. **Elaboración de la memoria:**
    a. Documentar en detalle el flujo completo del desarrollo.
    b. Presentar el resultado, analizar las posibles mejoras futuras y concluir la memoria.

1.5. Estructura del documento

El presente documento aborda todos los aspectos esenciales para comprender exhaustivamente el proyecto, estructurados en las siguientes secciones:

1. **Introducción:** En este apartado se comienza hablando de la importancia de la inclusión social y de los problemas identificados en el tema de la búsqueda de empleo de las personas con discapacidad, que es la razón principal por la que se inicia el proyecto actual. Se ha propuesto una resolución aproximada con una lista de objetivos a implementar, y por último, se ha presentado la estructura del documento.
2. **Estado de Arte:** En este apartado se incluye la presentación de la pila de desarrollo que se ha utilizada para implementar la aplicación, centrándose en el framework Next.js, que es el núcleo del desarrollo. Además, se detallan y justifican las herramientas, bibliotecas y softwares utilizados durante el proceso de desarrollo.
3. **Arquitectura, Metodología y Diseños:** En este apartado, por una parte, se explicará la arquitectura de la aplicación y la metodología utilizada durante el proceso de desarrollo. Por otra parte, se explicará la base de datos elegida y su estructura diseñada, y se concluirá con la demostración del diseño de la interfaz gráfica de la aplicación.
4. **Implementación de la Aplicación:** En este apartado se explica con detalle el desarrollo de la aplicación, tanto en la parte del Front-End como en la del Back-End.
5. **CoWorld:** En este apartado se demostrará el resultado del desarrollo en completo.
6. **Conclusión:** En este apartado se hará un resumen de todo lo que aprendido durante el proyecto y los avances realizados.

7. **Aspectos Legales, Éticos y Sociales:** En este apartado se presentarán los aspectos legales, éticos, medioambientales y sociales relacionados con el proyecto.
8. **Futuras Ampliaciones:** En este apartado se plantean las posibles ampliaciones que se podrían llevar a cabo la aplicación en el futuro para optimizar la aplicación.
9. **Bibliografía:** Se listará todas las referencia y fuentes utilizadas para elaborar la memoria.
10. **Anexo:** En este apartado se proporciona la información necesaria para adquirir el repositorio de código de la aplicación, así como una breve explicación de saber cómo conectarse y utilizarla.
11. **Agradecimientos**

2. Estado del Arte

2.1. La pila de desarrollo

La pila de desarrollo elegida para el desarrollo de la aplicación es: Next.js + MongoDB + Node.js. Se trata de una alternativa que la pila de desarrollo más popular, MREN, en la que se sustituye la parte de React y Express por el framework Next.js.

Empezando por la parte cambiada, es cierto que actualmente existen numerosas discusiones que comparan entre React y Next.js con el fin de sacar la conclusión de cuál es mejor que otro. Sin embargo, en realidad no son competidores, sino que colaboran entre sí. Ya que React solo es una biblioteca para construir el Front-End, para crear una aplicación completa con React es necesario tener una pila de tecnología completa, y Next.js es el primer framework recomendada la página oficial de React.

Next.js es un full-stack framework basado en React, que conserva todas las ventajas de React como la construcción de interfaces de usuario en componentes reutilizables y ser eficiente con el Virtual DOM Tree, y además, añade múltiples características nuevas y mejoradas. Entre las mejoras principales y potenciales de Next.js destacan:

- **Renderización**: permite el SSR (Server Side Rendering) con el componente servidor y el SSG (Static Side Generation). Estas técnicas, junto con la estrategia nuclear de renderización: Hidratación, optimiza el rendimiento de carga y el SEO (optimización para motores de búsqueda) de la aplicación.
- **Enrutamiento**: proporciona un enrutamiento basando en el sistema de archivos y eliminando la necesidad de utilizar una librería externa de enrutamiento como en React.
- **API Routes**: proporciona una gestión directa de los endpoints de API a través de la carpeta /api del proyecto de forma segura y sencilla. Asimismo, permite desarrollar la aplicación con la arquitectura monolítica, sin separar la parte Front-End y la de Back-End.

En cuanto a las tecnologías que se han mantenido del stack MREN: Node.js y MongoDB. Para desarrollar una aplicación web con el lenguaje JavaScript, Node.js es un entorno de ejecución en el servidor maduro y potente. Mientras MongoDB, es un sistema de base de datos NoSQL, que ofrece gran flexibilidad y rendimiento para el almacenamiento de datos.

Al combinar estas tecnologías potentes facilita el desarrollo de la aplicación y permite obtener un resultado moderno y sólido.

2.2. Herramientas de desarrollo

2.2.1. Visual Studio Code

Según la descripción dada de la Wikipedia:

Visual Studio Code (también llamado VS Code) es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. (Wikipedia contributors, 2025)

La razón principal para utilizar el Visual Studio Code es que se trata de un editor ligero que soporta la programación en TypeScript y el entorno de ejecución Node.js, los cuales son elementos fundamentales para el desarrollo de la aplicación.

2.2.2.GitHub

La plataforma de repositorio escogida para alojar el proyecto ha sido GitHub. Se trata de una de las comunidades de desarrolladores más populares del mundo y ofrece el sistema Git para gestionar de manera eficiente el control de versiones del proyecto, con el que también se ha trabajado a lo largo del estudio de la carrera.

2.2.3.MongoDBCompass

Es la interfaz gráfica de usuario proporcionada por MongoDB, que puede permite visualizar de manera clara los documentos almacenados en una colección, realizar consultas rápidas, gestionar índices y diseñar pipelines de agregación, entre otras funcionalidades. Gracias a MongoDBCompass, que facilita el trabajo relacionado con la base de datos, se mejora considerablemente la eficiencia del trabajo.

2.2.4.Postman

“Postman es una empresa mundial de software de origen indio que ofrece una plataforma de API para que los desarrolladores diseñen, creen, prueben y colaboren en API” (Wikipedia contributors, 2025). Con Postman se pueden configurar fácilmente las pruebas de llamadas a endpoints y, gracias a la representación de respuesta, se

entiende claramente el estado de respuesta y se puede averiguar el contenido de respuesta para asegurarse que es lo que se espera de obtener.

2.2.5.Jira

"Jira es un producto de software propietario para la gestión de proyectos, seguimiento de errores e incidencias" (Wikipedia contributors, 2024). Dado que el proyecto presente sigue la metodología ágil Scrum, se utilizan principalmente las funcionalidades Backlog, Tablero y Cronogramas de Jira para plantear sprints, gestionar requisitos funcionales y hacer un seguimiento del proceso de desarrollo de forma organizada.

2.3. Tecnologías principales

2.3.1.Next.js Framework

Next.js es un framework basado en React para desarrollar aplicaciones web a nivel full-stack. Aprovecha las ventajas y beneficios de React en la parte de front-end y completa la parte de back-end que React no abarca. Es el núcleo en la construcción de la aplicación del presente proyecto, que se desarrollará siguiendo una arquitectura monolítica y utilizando la estructura de App Route de Next.js.

2.3.2.MongoDB

Es un sistema de base de datos no SQL, almacena datos en formato de documento y facilita consultas complicadas, y que proporciona un gran flexibilidad y escalabilidad. La aplicación se almacena la información recopilada en MongoDB y desarrolla algoritmos utilizando el pipeline de aggregation del mismo.

2.3.3.Node.js

"Concebido como un entorno de ejecución JavaScript asíncrono basado en eventos, Node.js está diseñado para construir aplicaciones de red escalables" (Node.js contributors, s.f.). Además de su sólido funcionamiento, Node.js cuenta con un gran número de paquetes disponibles en el NPM (Node Package Manager), lo que facilita la importación de funcionalidades necesarias y la gestión de dependencias.

2.4. Librerías utilizadas

2.4.1.React

“React (también llamada React.js o ReactJS) es una biblioteca JavaScript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página” (Wikipedia contributor, 2024). Entre sus características principales destacan el desarrollo basando en componentes, el uso de Virtual Dom Tree y la sintaxis JSX, entre otras, que simplifican el desarrollo de un Front-end con buen rendimiento y alta mantenibilidad.

2.4.2.Mongoose

Es una biblioteca de JavaScript para Node.js, que se actúa un intermedio para establecer la conexión entre el Back-End de la aplicación y la base de datos MongoDB. En la aplicación se utilizará para estructurar los esquemas y realizar consultas, desempeñando así un papel fundamental en la interacción con la base de datos.

2.4.3.JSON Web Token

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autocontenida de transmitir información de forma segura entre las partes como un objeto JSON. Esta información puede ser verificada y fiable porque está firmada digitalmente. (AUTH0, s.f.)

En la aplicación, JWT se usa para controlar el acceso de los usuarios y autorizar operaciones mediante la verificación de tokens. Además de garantizar la seguridad de la aplicación, el token también contiene información básica del usuario, lo que facilita significativamente el proceso de identificación.

2.4.4.Bcryptjs

Se trata de una biblioteca que proporciona la funcionalidad de codificar contraseñas aplicando la función hash. En la aplicación se usa para codificar contraseñas y compararlas para verificar la autenticación del usuario.

2.4.5.Axios

Axios es un Cliente HTTP basado en promesas para node.js y el navegador. Es isomórfico (puede ejecutarse en el navegador y nodejs con el mismo código base) En el lado del servidor usa el módulo nativo http de node.js, mientras que en el lado del cliente (navegador) usa XMLHttpRequests. (Axios, s.f.)

En la aplicación, además de usar Axios para realizar solicitudes de HTTP desde la parte de Front-End, se ha implementado el sistema esencial de autenticación mediante tokens con la funcionalidad de interceptor de Axios para actualizar la sesión de usuario de forma silenciosa, garantizando así una mejor experiencia de usuario y la seguridad de la aplicación.

2.4.6.Resend

Se trata de una API de correo electrónico, en la aplicación se usa para las funcionalidades de restablecimiento la contraseña y activar cuenta de operador.

2.4.7.Bootstrap

Se trata de una biblioteca para el diseño e la implementación de la parte Front-end de la aplicación. Bootstrap ofrece una serie de componentes de interfaz gráfica de usuario diseñados y con posibilidad de configuración para que sea web responsive, lo que maximiza la eficiencia del desarrollo de la interfaz gráfica de usuario responsive.

2.4.8.React-loading-skeleton

Se trata de una biblioteca de Node.js fácil de utilizar para implementar esqueletos de carga. Permite personalizar el formato de los componentes esqueleto mediante una sintaxis similar a la del CSS, lo que facilita embellecer la página de carga en un tiempo mínimo.

3. Arquitectura, Metodología y Diseños

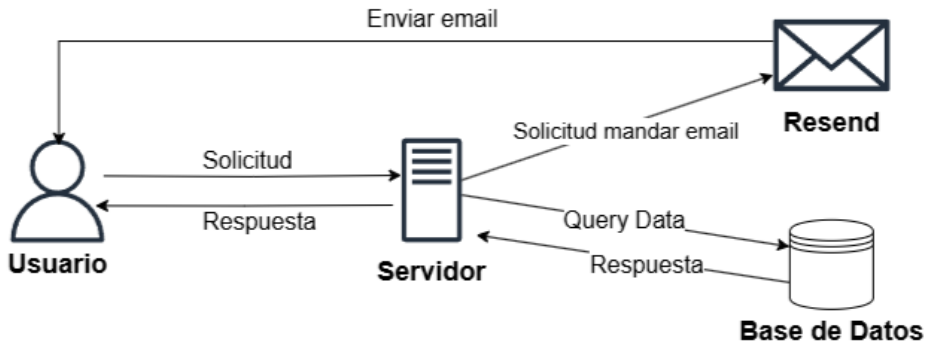
3.1. Arquitectura general

La aplicación CoWorld está adaptada a la arquitectura Client-Server, el patrón más utilizado los servicios online. Como describe por la Wikipedia: “La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes” (Wikipedia contributors, 2025). Lo cual destaca el objetivo principal de la arquitectura que es la clara separación de responsabilidades.

El flujo de trabajo de esta arquitectura es simple y claro. Por un lado, el componente cliente se encarga de realizar las peticiones desde la interfaz gráfica de usuario, y por otro lado, el componente servidor es responsable del funcionamiento nuclear de la aplicación. Tras recibir las peticiones del cliente, se actúa las operaciones correspondientes para responder la misma, como por ejemplo: realizar consultas en la base de datos para recuperar datos, autenticar al cliente, interactuar con sistemas externos, y entre otras. En la aplicación presente, el diseño de la arquitectura puede ilustrarse con el diagrama que se muestra a continuación (véase la figura 3), a partir del cual se pueden conocer los flujos principales de la aplicación, lo que facilitar a entender mejor el funcionamiento de la aplicación.

Figura 3

Arquitectura Cliente-Servidor



Nota. El diagrama representa la *arquitectura cliente-servidor* en la aplicación y sus flujos de trabajo principales.

Con el uso de la arquitectura cliente-servidor, permite construir una aplicación escalable y mantenible. Y con la característica de gestión central por el lado servidor nos

evita la situación de sobrecarga por el lado Front-end y facilita la gestión de seguridad e integridad de los datos, ya que el componente servidor es el único role que se interactúa con ellos. Sin embargo, también existe algunas desventajas relacionadas con la arquitectura, por ejemplo la gran dependencia que genera con el componente servidor, y lo cual puede causar una larga duración de caída de servicio cuando el servidor tenga algún problema o necesita realizar mantenimiento. Para mitigar dicho riesgo, se puede adoptar la opción de multiservidor para balancear la carga de trabajo y realizar backups de forma paralela con el servidor principal para garantizar una mayor disponibilidad del sistema.

3.2. Metodología

3.2.1.La metodología adoptada y las razones:

Debido la razón de la familiaridad que se ha adquirido durante el estudio del grado y es una de las metodologías más utilizadas en la actualidad, la metodología que se ha adaptado durante el proceso de desarrollo es la metodología Scrum.

Scrum es marco de trabajo ágil que está diseñado para gestionar el desarrollo de un proyecto en un equipo pequeño. Existe una estructura simple y efectiva para resumir la metodología: 3-5-3, que se refiere a sus 3 componentes principales: tres roles, cinco eventos y tres artefactos.

Una de las características principales es su forma de trabajar. El modelo de desarrollo que sigue es iterativo e incremental. Del mismo modo, el equipo Scrum trabaja en iteraciones cortas y más o menos de duración fija, denomina a Sprint, y al final de cada uno de ellos se entregan valores funcionales, lo que permite alcanzar a la máxima productividad.

Otro concepto importante es que el Scrum se basa en tres pilares principales: transparencia, inspección y adaptación. En primer lugar, la transparencia se refiere a todo el proceso y el trabajo debe ser transparente para conseguir una comunicación abierta y clara con el cliente y dentro del equipo. En segundo lugar, la inspección se refiere a la importancia de realizar una inspección continua para asegurar la calidad del producto y la satisfacción al cliente. Y, por lo último, la adaptación se refiera la capacidad que tiene el equipo para adaptarse ante los cambios y desafíos, lo cual se logra mediante a la experiencia y la cohesión del equipo.

Aunque la metodología Scrum se utiliza normalmente para proyectos en equipo, en este proyecto se ha aplicado a nivel individual por los siguientes motivos:

- Trabajar en sprints es una buena forma de organizar los objetivos que se pretende alcanzar en cada fase de desarrollo y de tener una visión clara del avance del proyecto.
- Los tres artefactos (pila del producto, pila del sprint e incremento), son herramientas útiles para gestionar el proyecto y registrar información de forma transparente, lo que mejora la eficiencia del desarrollo.
- La plataforma Jira permite gestionar y planificar proyectos con el modelo de trabajo Scrum, que incluye todos los componentes necesarios, lo que facilita en gran medida su gestionar. Asimismo, en el mercado laboral actual, la mayoría utiliza Jira para gestionar proyectos, por lo que también es un buen momento para conocer la herramienta.

3.2.2. Demostración

A continuación se presentan las demostraciones del uso de la plataforma Jira para gestionar el proyecto actual aplicando la metodología Ágil Scrum.

En primer lugar, se mostrarán los contenidos del artefacto pila de producto. La pila de producto es la única fuente del trabajo del equipo Scrum, donde se registran todos los trabajos que se pretenden a realizar para lograr el producto final. En el presente proyecto, los trabajos están descritos en formato de historias de usuario, cada una de las cuales se representa una funcionalidad de la aplicación cuyo tamaño adecuado para que pueda completarse en un único sprint. Las historias de usuario están agrupadas por épicas, que abarcan las historias de usuario relacionadas entre sí. La aplicación consta de cinco épicas, cada una con sus historias de usuario incluidas. (Véase los detalles de cada épica en las figuras 4 a 8.)

Figura 4

Épica: Gestión de autenticación

Tipo	Clave	Resumen	Persona asignada
Backlog			
	COW-56	Gestión de Autenticación	Jiale Wang
	COW-9	Registrar cuenta Empresa	Jiale Wang
	COW-8	Registrar cuenta candidato	Jiale Wang
	COW-58	Activar cuenta Operador	Jiale Wang
	COW-10	Iniciar sesión con cuenta candidato	Jiale Wang
	COW-15	Iniciar sesión con cuenta Empresa/Operador	Jiale Wang
	COW-13	Cambiar contraseña	Jiale Wang
	COW-11	Cerrar sesión	Jiale Wang

Nota. La imagen muestra la épica gestión de autenticación y sus historias de usuario.

Figura 5

Épica: Gestión de perfil

Tipo	Clave	Resumen	Persona asignada
	COW-56	Gestión de Autenticación	Jiale Wang
	COW-55	Gestión de Perfil	Jiale Wang
	COW-16	Editar perfil Candidato	Jiale Wang
	COW-12	Editar perfil Empresa	Jiale Wang
	COW-63	Ver perfil de una empresa (Candidato)	Jiale Wang
	COW-69	Ver perfil (Candidato)	Jiale Wang
	COW-70	Ver perfil (Empresa)	Jiale Wang

Nota. La imagen muestra la épica gestión de perfil y sus historias de usuario.

Figura 6 Épica:

Gestión de inscripciones

	COW-29	Gestión de inscripciones	Jiale Wang
	COW-30	Listar inscripciones recibidos	Jiale Wang
	COW-31	Ver información de solicitante	Jiale Wang
	COW-37	Cambiar estado de una inscripción	Jiale Wang

Nota. La imagen muestra la épica gestión de inscripciones y sus historias de usuario.

Figura 7 Épica:

Gestión de operadores

	COW-65	Gestión de operadores	Jiale Wang
	COW-60	Añadir Operador	Jiale Wang
	COW-68	Eliminar Operador	Jiale Wang
	COW-67	Modificar Operador	Jiale Wang

Nota. La imagen muestra la épica gestión de operadores y sus historias de usuario.

Figura 8

Épica: Gestión de oferta de trabajo

Tipo	Clave	Resumen	Persona asignada
	COW-14	Gestión de Oferta de trabajo	Jiale Wang
	COW-17	Publicar oferta de trabajo	Jiale Wang
	COW-19	Eliminar oferta de trabajo	Jiale Wang
	COW-18	Modificar oferta de trabajo	Jiale Wang
	COW-20	Listar ofertas publicados activas/cerradas (Empresa)	Jiale Wang
	COW-21	Listar ofertas recomendadas (Candidato)	Jiale Wang
	COW-61	Listar ofertas de trabajo por filtros seleccionados (Candidato)	Jiale Wang
	COW-62	Listar ofertas de trabajo por busqueda (Candidato)	Jiale Wang
	COW-64	Listar ofertas de trabajo activos desde pagina de información de u...	Jiale Wang
	COW-24	Ver una oferta en detalle (Candidato)	Jiale Wang
	COW-35	Guardar/No guardar oferta(Candidato)	Jiale Wang
	COW-32	Listar ofertas guardadas(Candidato)	Jiale Wang
	COW-26	Aplicar una oferta de trabajo(Candidato)	Jiale Wang
	COW-27	Listar solicitudes realizados por diferentes estados(Candidato)	Jiale Wang

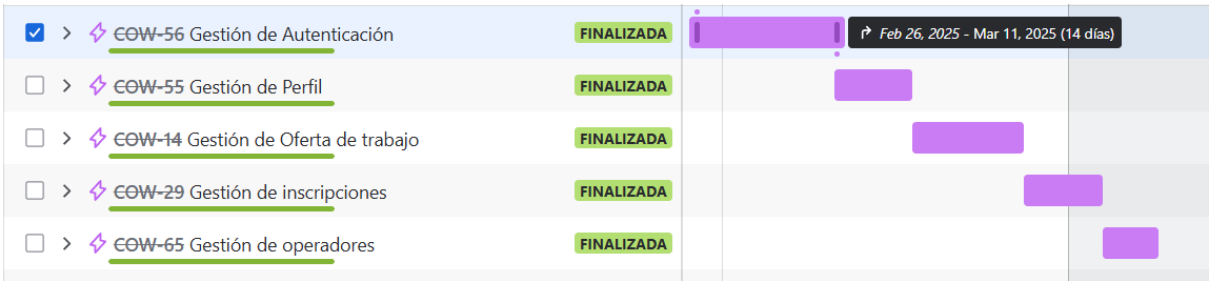
Nota. La imagen muestra la épica gestión de oferta de trabajo y sus historias de usuario.

Acto seguido se ilustrará la organización de cada sprint. Según la metodología Scrum, al finalizar un sprint debe entregarse un incremento como su resultado final, y cada incremento debe ser usable para asegurar que en cada sprint se aporta valor al producto. En consecuencia, como cada épica se representa como una funcionalidad completa, se ha definido que para cada sprint se terminar una épica completa. En ese sentido, cada sprint dura entre siete hasta catorce días, dependiendo del número de historias de usuario que abarca la épica y su dificultad en general. Además de los incrementos, cada sprint también debe disponer de una pila del sprint, donde se registra el trabajo que los desarrolladores planean realizar durante el sprint. En nuestro caso, la pila de sprint incluirá las historias de usuario que forman parte de la épica que se va a desarrollar durante el sprint.

La aplicación consta de un total de cinco sprints y el orden de desarrollo de las épicas es el siguiente: Gestión de autenticación, Gestión de perfil, Gestión de oferta de trabajo, Gestión de inscripciones y Gestión de operadores. (Véase el cronograma de los sprints junto con la fecha de inicio y fin de cada uno en las figuras 9 a 13. El contenido de cada pila de sprint puede consultarse en las figuras 4 a 8.)

Figura 9

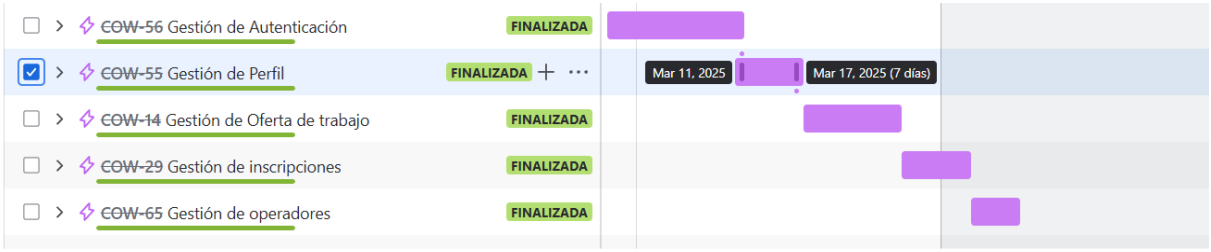
Sprint 1: Gestión de Autenticación



Nota. La imagen muestra el cronograma de los sprints y la fecha de inicio y fin del Sprint 1.

Figura 10

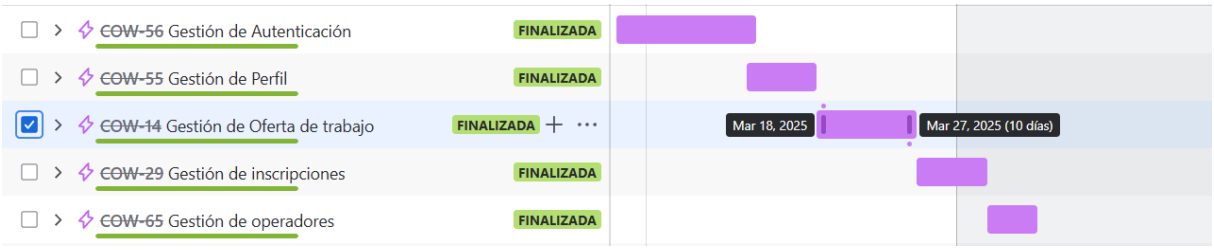
Sprint 2: Gestión de Perfil



Nota. La imagen muestra el cronograma de los sprints y la fecha de inicio y fin del Sprint 2.

Figura 11

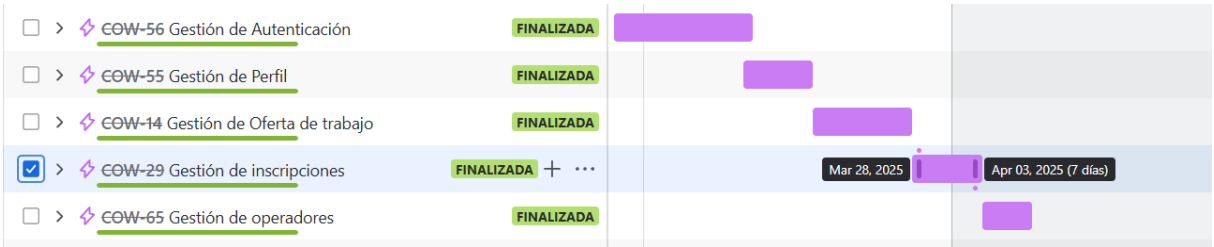
Sprint 3: Gestión de Oferta de trabajo



Nota. La imagen muestra el cronograma de los sprints y la fecha de inicio y fin del Sprint 3.

Figura 12

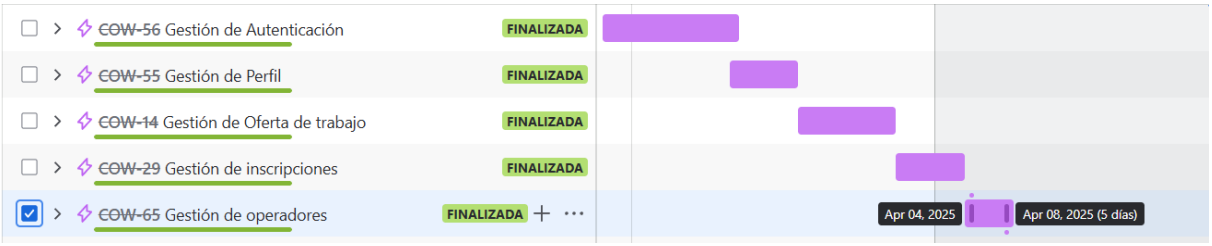
Sprint 4: Gestión de Inscripciones



Nota. La imagen muestra el cronograma de los sprints y la fecha de inicio y fin del Sprint 4.

Figura 13

Sprint 5: Gestión de Operadores



Nota. La imagen muestra el cronograma de los sprints y la fecha de inicio y fin del Sprint 5.

3.3. Diseños

3.3.1. Base de datos

El tipo de base de datos elegido para la aplicación es NoSQL, y el sistema de base de datos NoSQL más utilizado es MongoDB, que también se utiliza en la aplicación actual.

A diferencia a los bases de datos relacionales, los datos que se almacena en un base de datos NoSQL no tienen un formato fijo, el cual es la ventaja principal del NoSQL, la flexibilidad. Y por otro lado, el NoSQL trata de solucionar a los problemas de escalabilidad y rendimiento que puede surgir en un base de datos relacional mediante el gran volumen de almacenamiento y datos desnormalizados. Además, permite realizar consultas complejas, el cual puede ser otro problema para base de datos relacional.

En la actualidad, existen múltiples tipos de bases de datos no relacionales y MongoDB es una base de datos orientada a documentos. Los datos se almacenan en MongoDB en formato de documentos de BSON, y los documentos están agrupados en colecciones. Como mencionado anteriormente, las bases de datos NoSQL permiten realizar consultas complejas, y en MongoDB, esto se puede lograr utilizando agregaciones, que permite realizar consultas en formato de pipeline. Cada fase del pipeline se ejecuta el trabajo definido y su resultado se pasa como entrada a la siguiente, hasta llegar al final del pipeline. Dicha funcionalidad se utiliza ampliamente en la aplicación, ya que facilita de manera significativa los procesos relacionados con la manipulación de datos y la implementación del algoritmo de recomendación.

La aplicación se utiliza la librería Mongoose para establecer la conexión entre la base de datos MongoDB y el Back-End de la aplicación. En Mongoose, primero se definen esquemas que se utilizan para mapear a colecciones de la base de datos MongoDB. En los esquemas se definen básicamente los tipos de datos que almacena en una colección, pero a diferencia de las bases de datos relacionales, el esquema es flexible y no restringe el formato de los documentos, lo cual es ideal para construir datos semiestructurados ya que conservan las reglas necesarias para definir el formato de los datos sin perder la flexibilidad. Además de los esquemas, Mongoose utiliza modelos para crear y leer los documentos de las colecciones, que se representan como constructores de esquemas.

A continuación se muestran las seis colecciones creadas en la base de datos de la aplicación y sus esquemas correspondientes, junto con una explicación detallada de cada una de ellas.

- **users**: Se genera un documento cuando el usuario crea una cuenta como candidato en la aplicación. En él se almacenarán los datos personales introducidos por el usuario en el formulario de registro.

  Los campos que se definen en el esquema *User* son:

  - **firstname**: el nombre del candidato.
  - **lastname**: los apellidos del candidato.
  - **email**: el correo electrónico del candidato. Se utiliza para iniciar sesión y también como dato de contacto para las empresas.
  - **password**: la contraseña de la cuenta del candidato. Se utiliza para iniciar sesión, y el formato de la contraseña debe contener al menos ocho caracteres, una letra minúscula, una mayúscula, un número y un carácter especial.
  - **savedJob**: el array de trabajos guardados por el candidato. Almacenará los ID de los documentos de las ofertas de trabajo correspondientes, lo que representa una relación de 1: N con la colección Job; inicialmente es una lista vacía.
  - **role**: el número que representa el rol de la cuenta actual. El usuario con rol de candidato tendrá un valor de 0.

Figura 14

Esquema User

```
const userSchema:Schema = new Schema({
    firstname:{
        type: String,
        required: true
    },
    lastname:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,
        unique: true
    },
    password:{
        type: String,
        required: true
    },
    savedJob:{
        type:[{ type: mongoose.Schema.Types.ObjectId, ref: "Job" }],
        default:[]
    },
    role:{
        type: Number,
        default: CANDIDATE
    }
    },{timestamps: true});
```

Nota. La imagen muestra la definición completa del esquema User para la colección users.

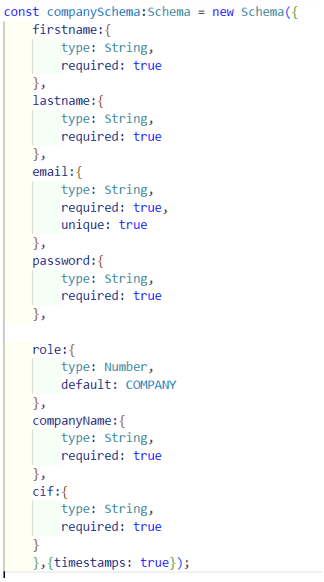
- **companies**: Se genera un documento cuando el usuario crea una cuenta empresarial en la aplicación. En él se almacenarán los datos personales del creador de la cuenta empresarial, así como la información básica de la empresa que el usuario introducido en el formulario de registro.

  Los campos que se definen en el esquema *Company* son:

  - **firstname**: el nombre del creador de la cuenta empresarial.
  - **lastname**: los apellidos del creador de la cuenta empresarial.
  - **email**: el correo electrónico del creador de la cuenta empresarial. Se utiliza para iniciar sesión.
  - **password**: la contraseña de la cuenta del usuario. Se utiliza para iniciar sesión, y el formato de la contraseña debe contener al menos ocho caracteres, una letra minúscula, una mayúscula, un número y un carácter especial.
  - **role**: el número que representa el rol de la cuenta actual. El usuario con rol de empresa tendrá un valor de 1.
  - **companyName**: el nombre de la empresa.
  - **cif**: el código de identificación fiscal de la empresa se usa para verificar la existencia de la empresa y el formato del cif debe ser una letra seguida de ocho números.

Figura 15

Esquema Company



```
const companySchema:Schema = new Schema({
    firstname:{
        type: String,
        required: true
    },
    lastname:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,
        unique: true
    },
    password:{
        type: String,
        required: true
    },

    role:{
        type: Number,
        default: COMPANY
    },
    companyName:{
        type: String,
        required: true
    },
    cif:{
        type: String,
        required: true
    }
    },{timestamps: true});
```

Nota. La imagen muestra la definición completa del esquema Company para la colección companies.

- **operators**: Se genera un documento cuando una cuenta empresarial agrega una cuenta de operador en la aplicación. En él se almacenarán los datos personales del operador introducidos por la cuenta empresarial en el formulario de registro.

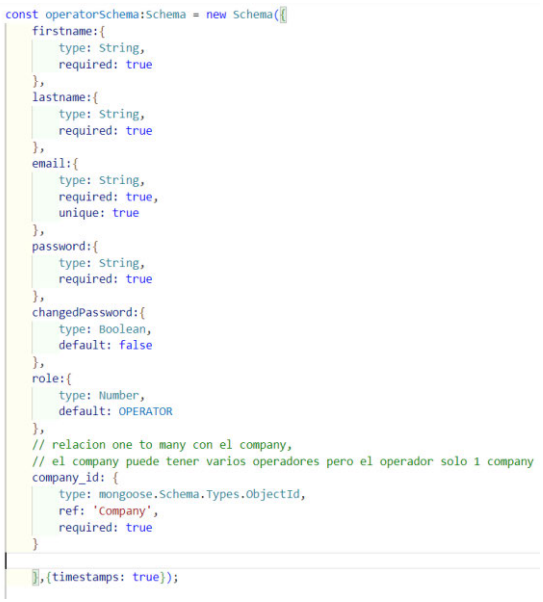
  Los campos que se definen en el esquema *Operator* son:

  - **firstname**: el nombre del operador.
  - **lastname**: los apellidos del operador.
  - **email**: el correo electrónico del operador. Se utiliza para enviar el correo de aviso de activación de la cuenta del operador e iniciar sesión.
  - **password**: la contraseña de la cuenta del operador. Se utiliza para iniciar sesión, y el formato de la contraseña debe contener al menos ocho caracteres, una letra minúscula, una mayúscula, un número y un carácter especial. Inicialmente se almacena la contraseña generada aleatoriamente, que usa para comprobar en el proceso de activación.
  - **changedPassword**: el valor booleano que indica si la cuenta está activada o no.

- **role**: el número que representa el rol de la cuenta actual. El usuario con rol de operador tendrá un valor de 2.

Figura 16

Esquema Operator



```
const operatorSchema:Schema = new Schema({
    firstname:{
        type: String,
        required: true
    },
    lastname:{
        type: String,
        required: true
    },
    email:{
        type: String,
        required: true,
        unique: true
    },
    password:{
        type: String,
        required: true
    },
    changedPassword:{
        type: Boolean,
        default: false
    },
    role:{
        type: Number,
        default: OPERATOR
    },
    // relacion one to many con el company,
    // el company puede tener varios operadores pero el operador solo 1 company
    company_id: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Company',
        required: true
    }

    },{timestamps: true});
```

Nota. La imagen muestra la definición completa del esquema Operator para la colección operators.

- **candidateprofiles:** Se genera un documento automáticamente cuando se crea con éxito una cuenta de usuario con el rol de candidato, y se establece una relación 1:1 con la colección users. En él se almacenarán los datos informativos del currículum, que inicialmente estarán vacíos de los campos relativos.

  Los campos que se definen en el esquema *CandidateProfile* son:

  - **phone**: el teléfono del candidato se utiliza como dato de contacto para las empresas.
  - **city**: la provincia de residencia del candidato.
  - **photo**: la foto del perfil del candidato.
  - **disabilities**: un array de situación de discapacidad del candidato, en cada situación en el que se registrará el tipo de discapacidad y el grado de la discapacidad. Se trata de un campo crítico para que la búsqueda de empleo se ajuste a la condición del candidato y el algoritmo de recomendación.
  - **state**: el estado del empleo actual del candidato. El valor puede ser: libre, estudiando o trabajando.

- **huntingJob**: el valor booleano que indica si el candidato está buscando empleo.
- **desiredJob**: un array de empleos deseados, donde el candidato puede registrar los tipos de trabajo a los que intenta de buscar. Se trata de un campo fundamental para el algoritmo de recomendación.
- **description**: la descripción personal del candidato.
- **studies**: un array de experiencia educativa del candidato. Para cada estudio se registrará: el nombre de la institución educativa, el título del estudio, la disciplina académica, la fecha de inicio y fin.
- **workExperience**: un array de experiencia laboral del candidato. Para cada experiencia laboral se registrará: el cargo, el nombre de la empresa, el tipo de contrato, la fecha de inicio y fin.
- **skills**: un array de habilidades del candidato. Se trata de un campo fundamental para el algoritmo de recomendación.
- **languages**: un array de idiomas que maneja el candidato. Para cada habilidad de idioma se registrará: el nombre del idioma y su nivel.
- **certifications**: un array de certificaciones del candidato. Para cada certificación se registrará: el título del certificado y el emisor del título.

Figura 17

Esquema CandidateProfile

Nota. La imagen muestra la definición completa del esquema CandidateProfile para la colección candidateprofiles.

- **companyprofiles:** Se genera un documento automáticamente cuando se crea con éxito una cuenta empresarial, y se establece una relación 1:1 con la colección companies. En él se almacenarán los datos de la empresa que inicialmente estarán vacíos de los campos relativos.

  Los campos que se definen en el esquema *CompanyProfile* son:

  - **industry**: la industria que dedica la empresa.
  - **city**: la provincia que ubica la sede principal de la empresa.
  - **scale**: la escala aproximada de la empresa.
  - **url**: el enlace de la página oficial de la empresa.
  - **logo**: el logotipo de la empresa.
  - **description**: la descripción de la empresa.

Figura 18

Esquema CompanyProfile

```
const companyProfileSchema:Schema = new Schema({
    // Informaciones empresa
    // relacion one to one con la empresa
    company_id: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Company'
    },
    industry:{
        type: String,
        default:""
    },
    city:{
        type: String,
        default:""
    },
    scale:{
        type: String,
        default:""
    },
    url: {
        type:String,
        default: ""
    },
    logo:{
        type: String,
        default:""
    },
    description:{
        type: String,
        default:""
    }
    },{timestamps: true});
```

Nota. La imagen muestra la definición completa del esquema CompanyProfile para la colección companyprofiles.

- **jobs**: Almacena toda la información en detallada relacionado con las ofertas de trabajo proporcionadas por las empresas. Se genera automáticamente

cuando la empresa se publica una oferta de trabajo tras de rellenar todos los datos necesarios para la oferta de trabajo y se establece una relación 1: N con la colección *companies*.

Los campos que se definen en el esquema *Job* son:

- **companyName**: el nombre de la empresa que pertenece la oferta de trabajo.
- **applicants**: un array de los candidatos que se han aplicado a la oferta de trabajo. Para cada candidato se registrará el id de su documento y se le asignará automáticamente el estado "solicitado" en el momento de la creación de la solicitud del empleo.
- **currentStatus**: el estado actual de la oferta de trabajo, se asigna automáticamente el estado "active" en el momento de la publicación de la oferta de empleo.
- **jobTitle**: el título de la oferta de trabajo.
- **city**: la ciudad del puesto de trabajo.
- **mode**: el modo del trabajo, su valor puede ser: Presencial, Híbrido y Remoto.
- **workHours**: la jornada del puesto de trabajo, su valor puede ser: Completa, Parcial – Mañana, Parcial – Tarde, Parcial – Noche.
- **experience**: la experiencia requerida para el puesto de trabajo, su valor puede ser: No requerido, 1 – 3 años, 4 – 6 años, 7 – 9 años, 10 – 10+ años.
- **intership**: el valor booleano que indica si la oferta de trabajo está orientada a becarios.
- **workCategory**: la categoría que pertenece el puesto de trabajo.
- **disabilities**: un array de requisitos de aceptación de discapacidad del candidato. Se registrará el tipo y el grado de la discapacidad aceptados para cada tipo de discapacidad. Se trata de un campo esencial para facilitar a los candidatos en su búsqueda de empleo.
- **minumumEducation**: la educación mínima requerida para el puesto de trabajo.
- **languages**: un array de requisitos de idiomas para el puesto de trabajo. Para cada requisito se registrará el nombre del idioma y el nivel requerido.

- **requiredKnowledge**: un array de requisitos de habilidades para el puesto de trabajo.
- **companysRequirements**: otros requisitos complementarios proporcionados por la empresa para el puesto de trabajo.
- **description**: la descripción del puesto de trabajo.

Figura 19

Esquema Job

Nota. La imagen muestra la definición completa del esquema Job para la colección jobs.

3.3.2. Diseño de la aplicación

Con el fin de mostrar las funcionalidades clave de la aplicación, se han desarrollado una serie de diagramas de casos de uso, acompañados de explicaciones detalladas, con el fin de ilustrar en una visión general del funcionamiento del sistema y las interacciones principales entre el sistema y los usuarios finales según sus roles. No se han detallado los casos en uso, ya que el propósito es facilitar la comprensión del sistema a un nivel global.

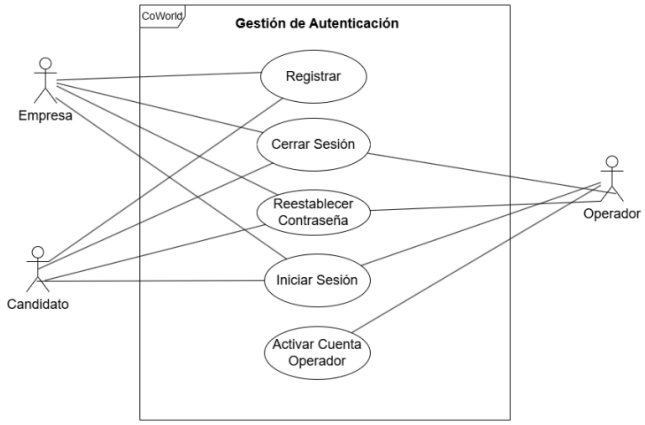
Los diagramas de casos de uso están organizados por épicas. En total se han diseñado cinco diagramas, cada uno con la participación de diferentes actores que representan a los usuarios finales del sistema según el contenido de la épica, cuales son: Candidato, Empresa y Operador.

Gestión de Autenticación: Esta épica se abarcan las funcionalidades relacionadas con la autenticación del sistema. Todos los actores podrán iniciar sesión, cerrar sesión y restablecer la contraseña de su cuenta, excepto los actores como

candidato y empresa, que se crean su cuenta a través del canal de registro. Y en el caso del actor operador, no podrá registrar una cuenta operador, ya que su cuenta debe ser añadida al sistema por una cuenta empresarial, en consecuencia, también es el único actor que tiene la funcionalidad de activar una cuenta operador tras recibir el correo con el aviso de activación de la cuenta.

Figura 20

Diagrama de casos de uso: Gestión de Autenticación



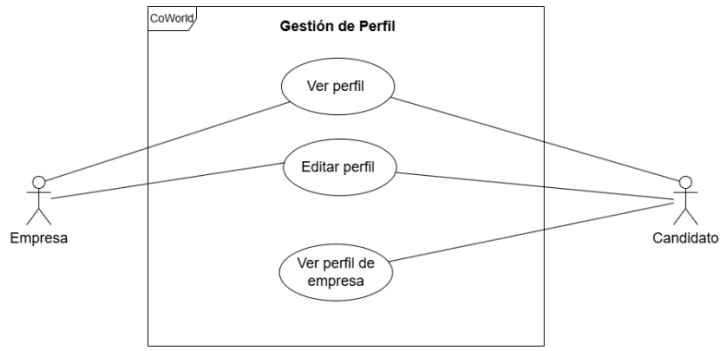
Nota. El diagrama de casos de uso correspondiente a la épica gestión de autenticación muestra los casos de uso implicados, así como los actores que participan en ellos.

Gestión de Perfil: Esta épica se abarcan las funcionalidades relacionadas con la gestión del perfil de los actores candidato y empresa. Se ha decidido no disponer un perfil específico para el actor operador, dado que, los perfiles de los candidatos y las empresas son visibles mutualmente, es decir, los candidatos pueden conocer el perfil de la empresa a través de su página de información, y las empresas tienen acceso al perfil de los candidatos inscritos en sus ofertas de empleo. En cambio, la información de los operadores consiste únicamente en sus datos personales y solo se gestionan internamente por la empresa a la que pertenece, por lo que no resulta necesario disponer un perfil visible para los operadores.

Por consiguiente, los candidatos y empresas pueden ver y editar sus perfiles, además, los candidatos tienen una funcionalidad aparte de ver perfil de la empresa. Para aclarar, las empresas también disponen de la funcionalidad de ver los perfiles de los candidatos inscritos en sus ofertas, pero dicha funcionalidad se encuentra en la épica de gestión de inscripciones por ser más adecuada.

Figura 21

Diagrama de casos de uso: Gestión de Perfil



Nota. El diagrama de casos de uso correspondiente a la épica gestión de perfil muestra los casos de uso implicados, así como los actores que participan en ellos.

Gestión de Oferta de Trabajo: Esta épica abarcan las funcionalidades relacionadas con las ofertas de trabajo. En el diagrama actual, se emplean las relaciones del tipo herencia y extend. Antes de detallar todas las funcionalidades de la épica, se ofrece una breve explicación de dichas relaciones, con el fin de facilitar la comprensión del diagrama.

Como se observa en la figura, todas las funcionalidades relacionadas con la listar ofertas de empleo comparten una relación de herencia con el caso de uso padre Listar ofertas. Esto se debe a que todas mantienen la misma forma de listar las ofertas, diferenciándose únicamente en el contexto de uso de cada una. Por tanto, el caso de uso padre Listar ofertas representa la parte común, mientras que sus casos de uso hijos extienden de esta funcionalidad con diferentes enfoques.

Entre los casos de uso hijos, se observa que existe una relación de extend entre la funcionalidad de Listar ofertas buscadas y Listar ofertas filtradas. Dicha relación se interpreta de la siguiente manera: tras de listar las ofertas resultantes de una búsqueda, el candidato puede aplicar filtros adicionales a ellas y listar el resultado del filtro. Asimismo, se han representado otras dos relaciones de extend entre la funcionalidad de Ver detalle de la oferta con la funcionalidad de Aplicar la oferta y la funcionalidad de Guarda la oferta. Estas relaciones indican que, tras de ver el detalle de una oferta, el candidato puede aplicar o guarda/no guardar dicha oferta.

En cuanto a los actores, se encuentra otra relación de herencia entre Empresa y Operador, lo que refleja que el operador puede realizar las mismas funcionalidades disponibles para la empresa.

A continuación, se presentan las funcionalidades incluidas en esta épica según el tipo de actores. Por un lado, tanto los operadores como las empresas pueden gestionar sus ofertas de empleo que son: publicar oferta, modificar oferta, eliminar oferta y listar sus ofertas publicadas según el estado de las ofertas (activas o cerradas). Y por el lado, los candidatos tienen distintas formas de listar las ofertas de trabajo, entre ellas: ofertas recomendadas por el algoritmo, ofertas filtradas, ofertas buscadas, ofertas desde el perfil de una empresa y ofertas clasificadas en distintos estados (guardados, solicitados, en progreso y cerrados). Además, pueden ver una oferta en detalle y, desde allí, aplicarla o guardar/no guardar la oferta.

Figura 22

El diagrama de casos de uso: Gestión de Oferta de Trabajo

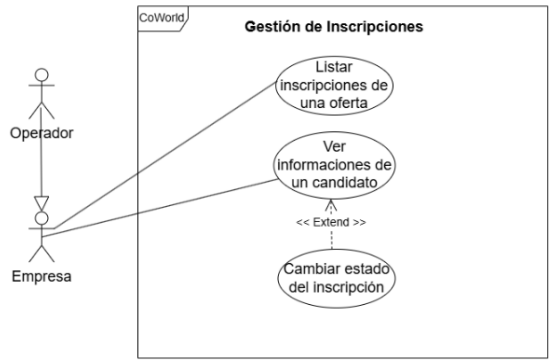
Nota. El diagrama de casos de uso correspondiente a la épica gestión de oferta de trabajo muestra los casos de uso implicados así como los actores que participan.

Gestión de inscripciones: Esta épica abarca todas las funcionalidades relacionadas con las inscripciones de las ofertas de trabajo, por esta razón, por lo que es exclusivas para los actores Operador y Empresa. Se observa que existe una relación de herencia entre ambos actores, lo que refleja que los operadores pueden utilizar las mismas funcionalidades disponibles para las empresas. En cuanto a las funcionalidades, las empresas y los operadores pueden listar las inscripciones recibidas de una oferta y ver la información de un candidato que ha solicitado el

puesto de trabajo. De forma paralela, se puede cambiar el estado de la inscripción del candidato para actualizar el avance del proceso de selección.

Figura 23

Diagrama de casos de uso: Gestión de Inscripciones

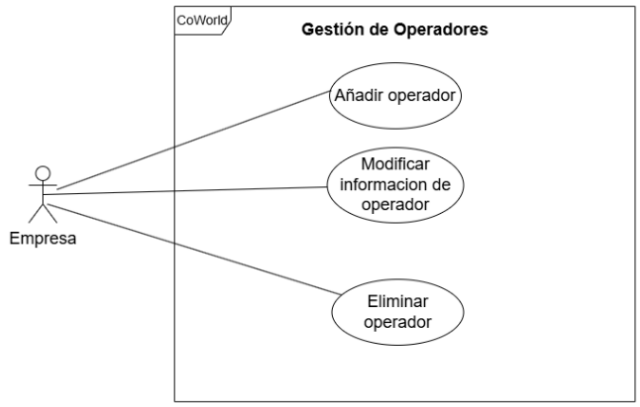


Nota. El diagrama de casos de uso correspondiente a la épica gestión de inscripciones muestra los casos de uso implicados así como los actores que participan.

Gestión de Operadores: Esta épica es exclusiva para el actor Empresa y abarca todas las funcionalidades relacionadas con la gestión de operadores. Las empresas pueden añadir, modificar y eliminar operadores de su cuenta.

Figura 24

Diagrama de casos de uso: Gestión de Operadores



Nota. El diagrama de casos de uso correspondiente a la épica gestión de operadores muestra los casos de uso implicados así como los actores que participan.

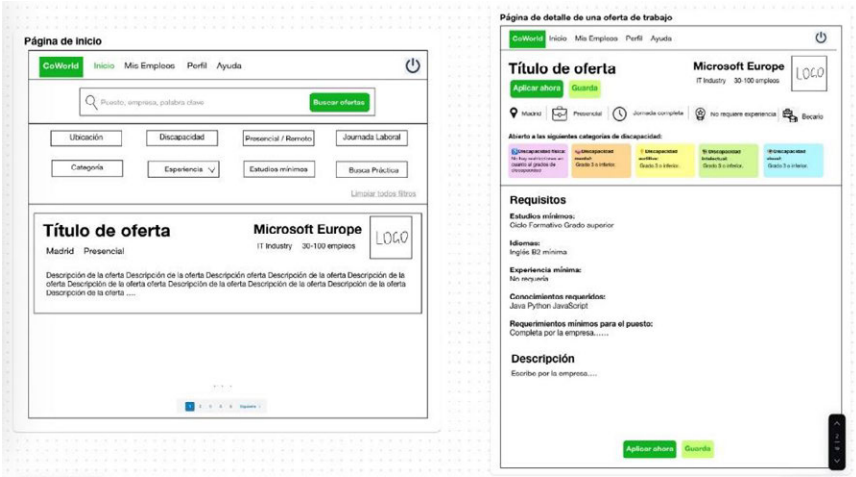
3.3.3.Diseño de la Interfaz Grafica

Antes de empezar la implementación de la aplicación, se ha elaborado un sencillo boceto de diseño Front-End a mano, que ha servido como un prototipo y una vista general de la aplicación. El diseño se ha actuado como una estructura general de la aplicación, ayudando a mantener la coherencia visual y funcional de la aplicación. Durante el proceso de desarrollo, se ha utilizado como una referencia constante, lo que ha facilitado significativamente a la eficiencia en la hora de codificar.

En primer lugar, se muestra el diseño de las páginas que pertenecen a los usuarios con el rol de candidato:

Figura 25

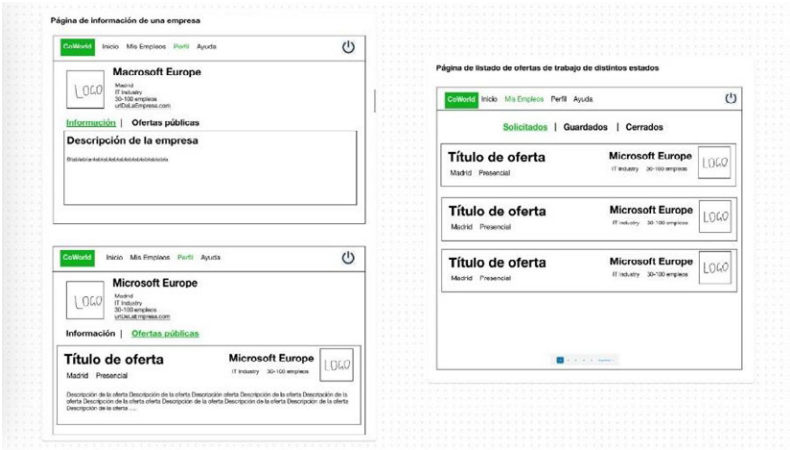
Diseño de Front-End: Parte 1 de los Candidatos



Nota. La imagen se muestra el diseño de front-end de la página inicial y la página de detalle de una oferta de trabajo para los usuarios con el rol de candidato.

Figura 26

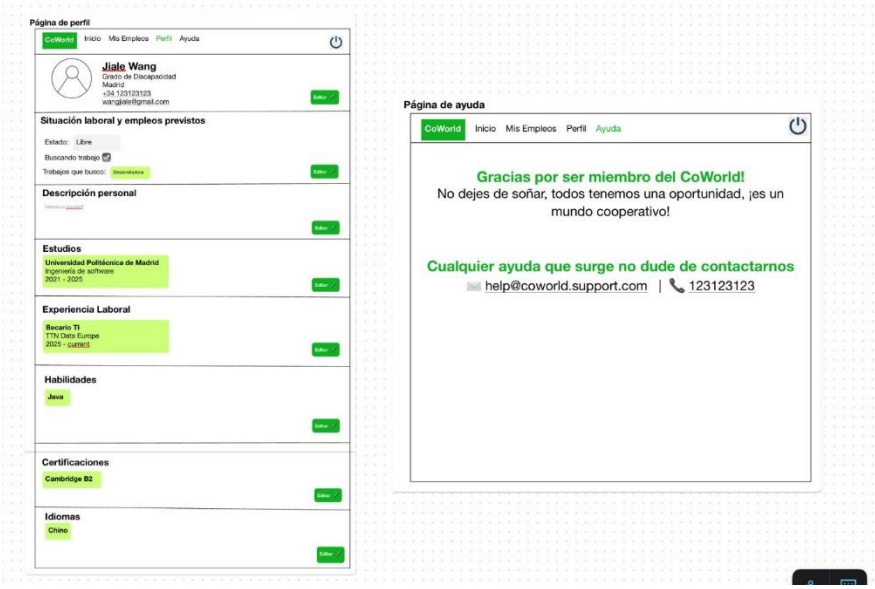
Diseño de Front-End: Parte 2 de los Candidatos



Nota. La imagen se muestra el diseño de front-end de la página de información de una empresa y la página de listado de ofertas de trabajo de distintos estados para los usuarios con el rol de candidato.

Figura 27

Diseño de Front-End: Parte 3 de los Candidatos

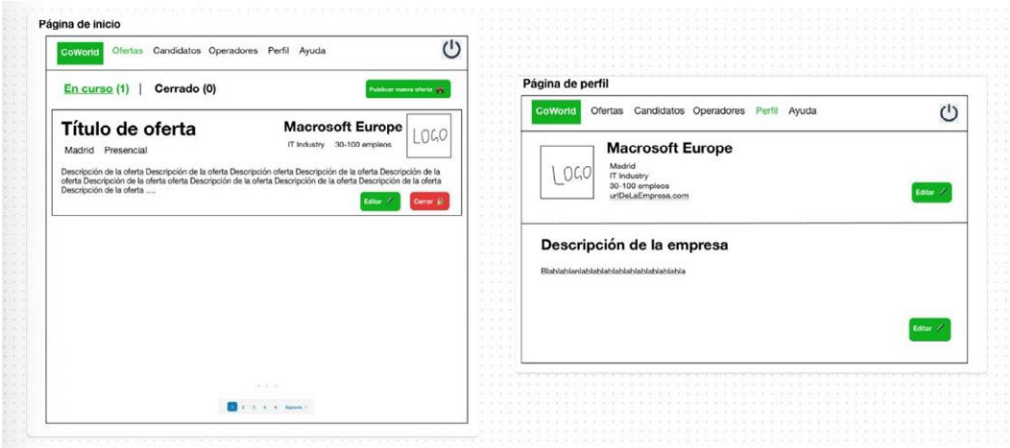


Nota. La imagen se muestra el diseño de front-end de la página de perfil y la página de ayuda para los usuarios con el rol de candidato.

A continuación, se muestra el diseño de las páginas que pertenecen a los usuarios con el rol de empresa:

Figura 28

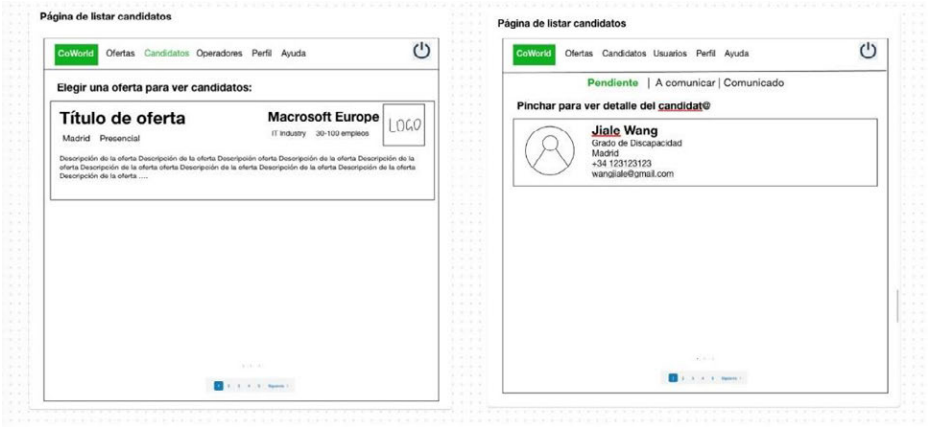
Diseño de Front-End: Parte 1 de las Empresas



Nota. La imagen se muestra el diseño de front-end de la página inicial y la página de perfil para los usuarios con el rol de empresa.

Figura 29

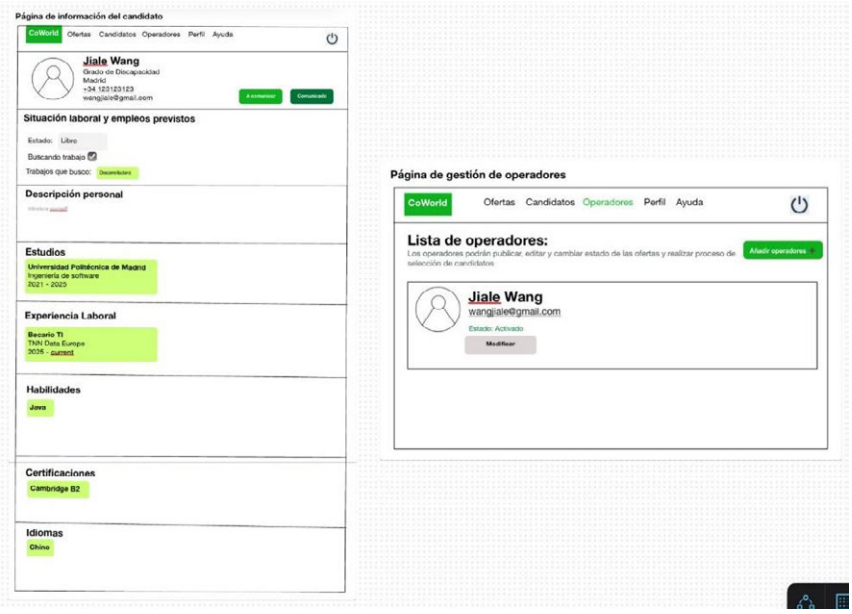
Diseño de Front-End: Parte 2 de las Empresas



Nota. La imagen se muestra el diseño de front-end de las páginas de listado de candidatos de una oferta de trabajo para los usuarios con el rol de empresa.

Figura 30

Diseño de Front-End: Parte 3 de las Empresas



Nota. La imagen se muestra el diseño de front-end de la página de información de un candidato y la página de gestión de operadores para los usuarios con el rol de empresa.

Por lo último, se muestra el diseño de las páginas que pertenecen a los usuarios con el rol de operador. Los operadores tendrán las mismas páginas que las empresas, excepto las páginas relacionado con la funcionalidad de gestión de perfil y gestión de operadores:

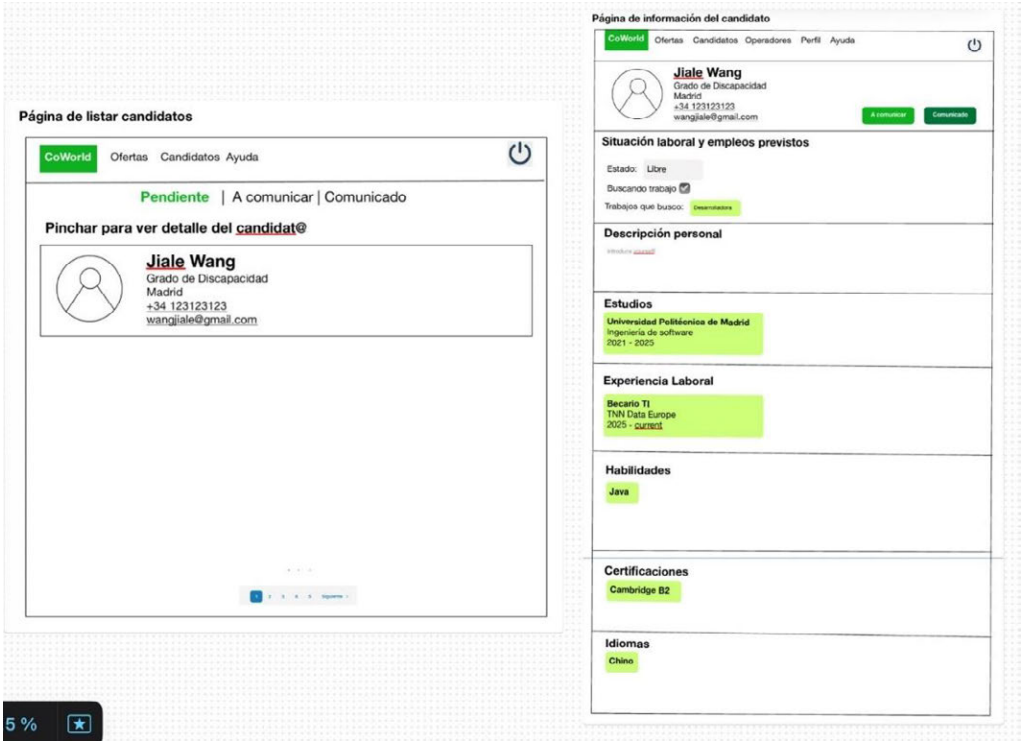
Figura 31

Diseño de Front-End: Parte 1 de los Operadores

Nota. La imagen se muestra el diseño de front-end de la página inicial y la página de listado de candidatos de una oferta de trabajo para los usuarios con el rol de operador.

Figura 32

Diseño de Front-End: Parte 2 de los Operadores



Nota. La imagen se muestra el diseño de front-end de la página de listado de los candidatos y la página de información de un candidato para los usuarios con el rol de operador.

4. Implementación

Antes de empezar explicar los detalles del desarrollo completo de la aplicación CoWorld, se considera es importante hacer una breve introducción sobre la arquitectura de desarrollo y la estructura del proyecto utilizando el framework Next.js.

Inicialmente, la arquitectura que se utiliza por Next.js es la arquitectura monolítica. A diferencia de la arquitectura que adoptan la mayoría de las aplicaciones actualmente, que separan estrictamente la parte Front-End y Back-End, la arquitectura monolítica permite que todo funcione de manera integral como un único módulo. Al tener la capa de presentación, la lógica de negocio y la persistencia altamente cohesionadas, el proceso de desarrollo se simplifica significativamente y permite entregar el software lo antes posible.

Con la rápida evolución de la tecnología, el software se ha vuelto cada vez más complejo y ha aumentado la exigencia sobre el tema de flexibilidad, lo que ha provocado que la arquitectura monolítica deje de ser una opción popular. Gracias al framework Next.js, se ha conseguido con éxito a que la arquitectura monolítica volva a aparecer en el mercado. El framework Next.js no solo ha abordado las limitaciones de la arquitectura monolítica para mantener las ventajas de la misma, sino que también ha reunido las nuevas tecnologías para ofrecer mejores opciones de desarrollo.

A continuación, se muestra la estructura del proyecto utilizando el framework Next.js, en la que se refleja claramente la característica de la arquitectura monolítica.

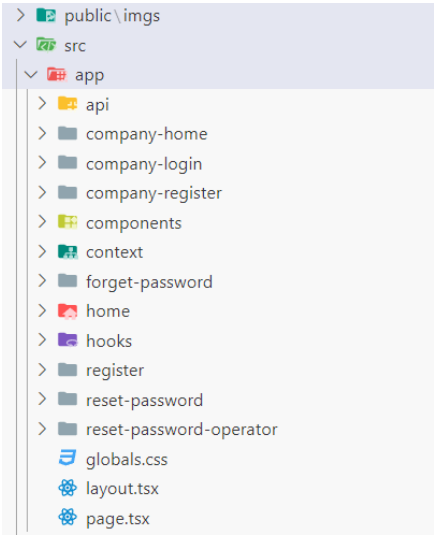
Según la documentación oficial de Next.js (Next.js contributors, s.f.):

En primer lugar, se encuentran las carpetas del nivel superior, que se utilizan para organizar el código de la aplicación y los conjuntos estáticos. En el proyecto actual, las carpetas de este nivel son:

- **src**: una carpeta opcional que representa la carpeta de raíz del proyecto y que contiene todos los archivos de código de la aplicación.
- **app**: la carpeta App Route, que se ubica dentro de la carpeta src. Se utiliza para definir las rutas de la aplicación. La carpeta se representa la ruta raíz (/) de la aplicación.
- **public**: la carpeta que se utiliza para guardar los ficheros estáticos, como imágenes. Los ficheros dentro de la aplicación se pueden referenciar directamente desde los ficheros ubicados dentro de la carpeta app.

Figura 33

Las carpetas del nivel superior



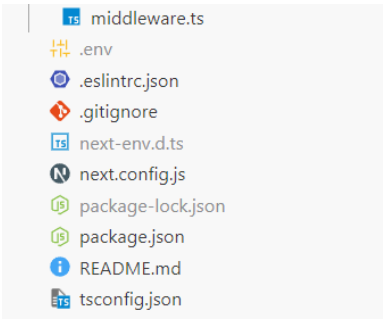
Nota. La imagen muestra las carpetas del nivel superior que son: src, app y public, así como el contenido de la carpeta app.

En segundo lugar, se encuentran los ficheros del nivel superior, que son utilizados para configurar la aplicación, gestionar las dependencias, ejecutar el middleware y definir las variables del entorno:

- **next.config.js:** el fichero de configuración para el Next.js.
- **package.json:** el fichero para gestionar las dependencias de la aplicación y definir cómo se ejecuta la aplicación en las diferentes fases de desarrollo.
- **middleware.ts:** el fichero de script del middleware de la aplicación. El middleware se ejecuta antes de que se complete una petición. En la aplicación se utiliza para comprobar la existencia de un accessToken antes de las peticiones.
- **.env:** el fichero donde se define las variables del entorno para garantizar el correcto funcionamiento de la aplicación. En la aplicación actual, se definen las variables para: la dirección de la base de datos de MongoDB, las claves del API Resend, las claves del accessToken y RefreshToken.
- .**gitignore:** el fichero donde se definen los ficheros que hay que excluir del git.
- .**eslint.json:** el fichero de configuración para el plugin ESLint.
- **tsconfig.json**: el fichero de configuración de TypeScript, dada que la aplicación utiliza el lenguaje TypeScript.

Figura 34

Los ficheros del nivel superior



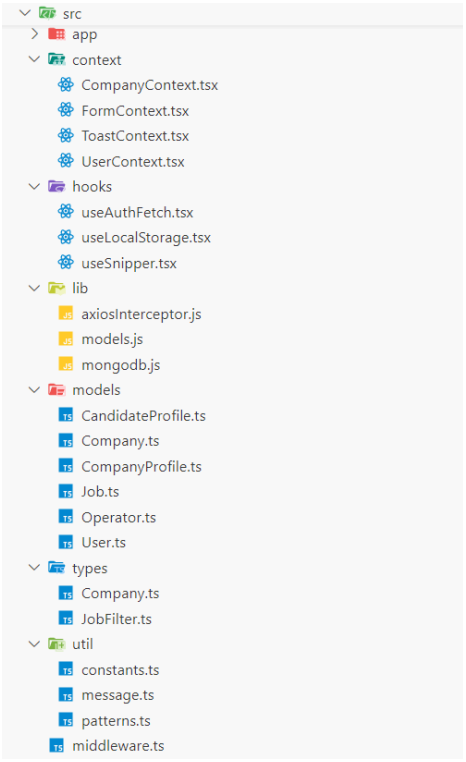
Nota. La imagen muestra los archivos del nivel superior.

En tercer lugar, se presenta la organización dentro de la carpeta raíz (src). Existen varias formas de organizarla. En el proyecto actual, el contenido se divide entre la carpeta app, que se utiliza únicamente para el propósito de enrutamiento (excepto la carpeta component, que son componentes de UI y se dejan juntos con las páginas) y el resto de los espacios, que están destinados a otras carpetas y ficheros de código:

- **app**: la carpeta de enrutamiento de la aplicación.
- **context**: la carpeta donde guarda los ficheros del useContext personalizados.
- **hooks**: la carpeta donde guarda los ficheros de los hooks personalizados.
- **lib**: la carpeta en la que se almacena las funciones personalizadas relacionadas con las bibliotecas externas. En la aplicación actual, almacena el fichero para conectar con la base de datos de MongoDB y el fichero de definición del interceptor de la librería Axios.
- **models**: la carpeta que se utiliza para guardar los ficheros de definiciones de los esquemas de la base de datos de MongoDB de la aplicación. (La explicación detallada de los esquemas se encuentra en el capítulo 3.3.1 Base de datos.)
- **types**: la carpeta que se utiliza para guardar los ficheros relacionado con la definición de tipos de datos personalizados para la aplicación.
- **útil**: la carpeta que se utiliza para guardar los ficheros con contenidos que se reutilizan con frecuencia dentro de la aplicación, con el fin de mantener el código limpio. En la aplicación se guardan los constantes, mensajes y patrones de RGX.
- **middleware.ts**: el fichero de script del middleware de la aplicación.

Figura 35

La carpeta src



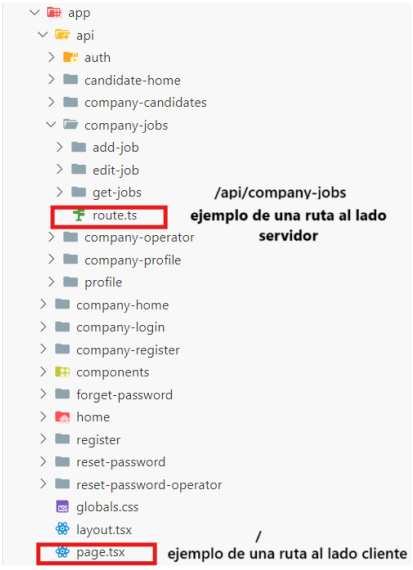
Nota. La imagen el contenido de la carpeta src de la aplicación.

Por lo último, se centra en explicar la estructura interna de la carpeta app. Como se ha mencionado anteriormente, dicha carpeta se utiliza para gestionar el enrutamiento de la aplicación. En Next.js, las carpetas anidadas dentro del directorio app se corresponden con la estructura de las rutas de la aplicación. Dentro de la misma carpeta, las rutas que correspondientes a los endpoints de la aplicación se deben situar dentro de otra carpeta, api, para diferenciar entre las rutas dirigidas al lado cliente y al lado servidor. Cada carpeta representa un segmento de la ruta, y dicha ruta no es públicamente accesible hasta que se incluye un fichero enrutable page.js o route.js. Además, el contenido de los ficheros page.js o route.js es la única información que se devuelve al lado cliente. En cuanto a los ficheros enrutables, existen varios tipos, y en la aplicación se han utilizado los siguientes:

- **layout.js (lado cliente):** el fichero donde se define el diseño de la aplicación, lo que permite definir los elementos que se repiten en todas las páginas en un único fichero, lo que mejora la consistencia de la aplicación.
- **page.js (lado cliente):** el fichero final que se convierte en la interfaz gráfica de usuario de una ruta. En caso de existen ficheros layout.js en el mismo nivel o en niveles superiores, se mostrarán también sus contenidos.
- **route.js (lado servidor)**: el fichero donde se define el contenido para gestionar una petición de un endpoint determinado.

Figura 36

La carpeta app



Nota. La imagen muestra el contenido de la carpeta app junto con los ejemplos de la ruta de la aplicación en el lado servidor y en el lado cliente.

4.1. Componentes de utilidad y lógica compartida

Los componentes que se intentan explicar en esta sección son los que pertenecen a las carpetas lib, context, hooks y así como el fichero middleware.ts. El motivo de separarlos de los demás contenidos de la aplicación (que son los componentes claramente distinguibles por el lado cliente y servidor) es, en primer lugar, para poder explicar con mayor claridad los detalles de la implementación y, en segundo lugar, porque son componentes clave que dan soporte común a toda la aplicación y garantizan su reutilización y el mantenimiento, por lo que se quiere reflejar su importancia. Sobre todo, se quiere destacar la importancia de los componentes nucleares que componen el sistema de autenticación de la aplicación.

En primer lugar, se presentan los ficheros ubicados dentro de la carpeta context. Todos los ficheros que pertenecen a esta carpeta son el contexto personalizado. El contexto es un término de React, que permite compartir el dato globalmente sin tener que pasarlo como propiedad de componente manualmente nivel por nivel, lo que puede causar el problema de “prop drilling”.

- **FormContext:** Dado a que la aplicación usa frecuentemente los formularios para poder recopilar todos los datos como un conjunto y almacenarlos

posteriormente en la base de datos. Se ha creado el FormContext para gestionar globalmente los datos recibidos por distintos componentes que forman un formulario en pares de clave-valor. De esta forma, se puede enviar fácilmente el contenido del contexto como un objeto a la base de datos para su almacenamiento.

Figura 37

FormContext

```
export type formValues = | string | number | boolean | string[] | undefined| null | study[] | date|
workExperience[]| certification[]|language[]| disability[];

export type FormProperties = Record<string, formValues>;
// Definimos el tipo de propiedades del formulario, para que sea
// un objeto con claves de tipo string y valores de tipo formValues

interface FormContextType { // Para definir el tipo de contexto, para compartir y actualizar los datos del formulario
    formProperties: FormProperties;
    setFormProperties: (formProperties: FormProperties) => void;
}

export const FormContext = createContext<FormContextType|undefined>(undefined)
```

Nota. La imagen muestra el contenido principal del fichero FormContext.tsx de la aplicación.

- **ToastContext**: un toast es un componente que se ha implementado para mostrar un aviso en la pantalla del usuario durante un tiempo determinado. Dado que la aplicación solo necesita un único toast para mostrar el aviso en el momento, se ha definido un contexto para gestionarlo. El contexto se sitúa en la raíz de la aplicación (el layout raíz de la aplicación) para que todas las páginas de la aplicación puedan utilizar el toast cuando sea necesario. Los elementos compartidos en el contexto son:
    - **msg**: el contenido del aviso.
    - **type**: el tipo de aviso, puede ser *Good* o *Bad*. En el caso es del tipo *Good*, se muestra el contenido del aviso con un marco verde, que se usa para indicar el éxito de una operación y, por el contrario, con el tipo *Bad*, se muestra el contenido del aviso con un marco rojo, que se usa para indicar el fracaso de una operación.
    - **visible**: una variable booleana con que se determina si hay que mostrar el toast o no.

Figura 38

ToastContext

```tsx
export const ToastContext = createContext<IToast>({} as IToast)

export const ToastProvider=({children}:toastProviderProps)=>{
    const [toast, setToast] = useState<toastContextType>(initialToastContextType)

    const showToast=(props:toastContextType)=>{ // handler de toast
        if(props){
            setToast(props)

            setTimeout(()=>{ // despues de mostrar, volver a estado iniciar
                setToast({msg:null, type:null,visible:false})
            },3000)
        }
    }
 // defino el provider aqui, porque todo el sys solo voy a necesitar 1 toast para todos
    return(
        <ToastContext.Provider value={{...toast, showToast}}>
            {children}
            {toast.visible&& (<Toast msg={toast.msg} type={toast.type}/>)}
        </ToastContext.Provider>
    )
}
```

Nota. La imagen muestra el contenido principal del fichero ToastContext.tsx de la aplicación.

- **UserContext**: El motivo principal de implementar este contexto es para facilitar la obtención y actualización de la información del usuario actual (con el rol de candidato), tanto en la gestión del perfil del usuario como en la gestión de sus ofertas de trabajos guardados. Además de la información del candidato que se comparte en este contexto, tiene también un método getUser() para actualizar los datos del contexto haciendo llamada al back-end cuando hay cambios en la información del candidato.

Figura 39

UserContext

```tsx
export const UserContext = createContext<IUser>({} as IUser);

export const UserProvider=({children}:userProviderProps)=>{
    const [user, setUser] = useState<userContextType>()
    const [waiting, setWaiting] = useState(true);

    const getUser = async() =>{
        try{
        const {data} =await axiosInstance.get(`/profile/get-profile` ,{
            withCredentials:true
        });
        const user:userContextType = {···
        }
        setUser(user);
        setWaiting(false);
        }catch(e){
            console.log("Hubido problema en cargar datos de usuario:"+e);
            setWaiting(false);
        }
    }
    useEffect(()=>{
        getUser()
    },[]) //No ponemos dependencia de actualizacion en aqui es porqu sino se causa un bulce infinito de actualizacionnes.
        //Por eso actualizamos manualmente cada vez hay cambio en el profile, haciendo getUser().

    return(
        <UserContext.Provider value={{user,waiting,getUser}}>
            {children}
        </UserContext.Provider>
    )
}
```

Nota. La imagen muestra el contenido principal del fichero UserContext.tsx de la aplicación.

- **CompanyContext**: Este contexto funciona de manera similar al UserContext, pero está destinado a los usuarios con el rol de company o operator. Además de compartir la información de la empresa, también se guarda la información de

las ofertas de empleo que pertenece a la empresa y así como el rol de la cuenta. El rol se utiliza para limitar el acceso a las secciones de la gestión del perfil y de operador de la empresa para los operadores.

Figura 40

CompanyContext

```
export const CompanyContext = createContext<ICompany>({} as ICompany);

export const CompanyProvider=({children}:companyProviderProps)=>{
    const [company, setCompany] = useState<companyContextType>()
    const [waiting, setWaiting] = useState(true);
    const getCompany = async() =>{
        try{
        const {data} =await axiosInstance.get(`/company-profile/get-profile` ,{
            withCredentials:true
        });
        const  jobs = await axiosInstance.get(`/company-jobs/get-jobs` ,{
            withCredentials:true
        });
        const role = await axiosInstance.get(`/auth/company-check-role` ,{
            withCredentials:true
        });
        const company:companyContextType = {···
        }
        setCompany(company);
        setWaiting(false);
        }catch(e){
            console.log("Hubido problema en cargar datos de empresa:"+e);
            setWaiting(false);
        }
    }
    useEffect(()=>{
        getCompany()
    },[]) //No ponemos dependencia de actualizacion en aqui es porqu sino se causa un bulce infinito de actualizacionnes.
        //Por eso actualizamos manualmente cada vez hay cambio en el profile, haciendo getCompany().
    return(
        <CompanyContext.Provider value={{company,waiting,getCompany}}>
            {children}
        </CompanyContext.Provider>
    )
}
```

Nota. La imagen muestra el contenido principal del fichero CompanyContext.tsx de la aplicación.

En segundo lugar, se muestran los ficheros que pertenece a la carpeta hooks. Gracias a React, que permite a los desarrolladores personalizar sus propios hooks, aparte de los hooks que ofrece, lo que es ideal para tener un hook adaptado a las necesidades de la aplicación. En la aplicación actual tienen en total de tres hooks personalizadas:

- **useAuthFetch**: el hook definido para facilitar a la realización de llamadas a los endpoints relacionados con la gestión de autenticación, que comparten el mismo prefijo en cada una de las rutas: */api/auth/*. Tras de realizar la llamada, el hook también se encarga de dirigir al usuario a la página de destino correspondiente o de mostrar el resultado de la llamada con el toast del sistema. Para usar este hook, es necesario proporcionar valores a las propiedades del mismo, que son*:* el endpoint(la dirección del endpoint que va a enviar), nextPath(el enlace que se redirige cuando la respuesta es exitosa), fetchdata(los datos que se envía al back-end, normalmente, son los datos recopilados por los formularios de la página correspondiente) y opcionalmente el variable config(se usa para definir otras configuraciones adicionales de la petición HTTP, como el contenido del encabezado, etc.).

Figura 41

useAuthFetch

```
export function useAuthFetch() {
  const router = useRouter();
  const {showToast} = useContext(ToastContext)
  const authFetch=async({endpoint,nextPath,config,fetchdata}:authProps)=>{
    try{
        const updatedConfig = {
          ...config,
          withCredentials: true,
        };
      const {data} =await axios.post(`/api/auth/${endpoint}`,fetchdata,updatedConfig)
      showToast({msg:data.sucess, type:'Good',visible:true})
      router.push(nextPath)
    }catch(e:any){
      showToast({msg:e.response.data.error as string, type:'Bad',visible:true})
    }
  }

  return authFetch
}
```

Nota. La imagen muestra el contenido principal del fichero useAuthFetch.tsx de la aplicación.

- **useLocalStorage**: en la aplicación se utiliza el local storage para almacenar los filtros elegidos por el usuario con el rol de candidato, con el fin de mantener del resultado de filtro cuando usuario vuelve a la página de inicio, mejorando así la experiencia de usuario. En consecuencia, se ha creado este hook para gestionar el uso del local storage. Dada que existen múltiples combinaciones de filtros que pueden ser elegidas por el usuario y no se puedan predecir sus elecciones, los filtros se guardan en el local storage con el formato de clave-valor. La clave es *JobFilters* y el valor es un tipo genérico que permite recuperar todos los filtros elegidos por el usuario de una vez.

  En el archivo se han definido dos useEffect. El primer useEffect se utiliza para recuperar los valores almacenados en el local storage en caso de que existan, en caso contrario, se guardan con valores vacíos. Y el segundo se utiliza para monitorizar el valor del local storage y actualizarlo cuando hay cambios.

Figura 42

useLocalStorage

```
export default function useLocalStorage<T>(key:string, initalValues:T){

  const [value, setValue] = useState<T>(initalValues);

  useEffect(() => {
      try {
      const stored = localStorage.getItem(key);
      if (stored) setValue(JSON.parse(stored));
      } catch (e) {
      console.error("LocalStorage read error:", e);
      }
  }, [key]);

  useEffect(()=>{
      try{
          localStorage.setItem(key,JSON.stringify(value));
      }catch(e){
          console.log("Error en actualizar contenido de localStorage!!!"+e)
      }
  },[key,value])

  return [value,setValue] as const
}
```

Nota. La imagen muestra el contenido principal del fichero useLocalStorage.tsx de la aplicación.

- **useSnipper**: es un hook sencillo para gestionar la visibilidad del snipper. Lo que maneja es un valor booleano, en caso de ser verdadero se muestra el snipper, y en caso contrario, no se muestra.

Figura 43

useSnipper

```
import { useState } from "react";

interface snipperProps {
    isLoading: boolean;
    setIsLoading: (isLoading: boolean) => void;
}

export const useSnipper = (): snipperProps => {
    const [isLoading, setIsLoading] = useState(false);
    return {
        isLoading,
        setIsLoading
    };
};
```

Nota. La imagen muestra el contenido principal del fichero useSnipper.tsx de la aplicación.

En tercer lugar, se explica primero los ficheros relacionados con la base de datos MongoDB que se ubican en la carpeta lib y, a continuación se explica en detalle el sistema de autenticación de la aplicación, ya que utiliza los restos componentes de la carpeta lib y el middleware.

El fichero mongodb.js se utiliza para definir la función de establecer conexión con la base de datos de MongoDB de la aplicación, y se utiliza principalmente en la parte de back-end para realizar operaciones con dicha base de datos.

Figura 44

mongodb

```
import mongoose from "mongoose";
import "./models";
const MONGODB_URL = process.env.MONGODB_URL;

export const connectMD = async () => {
    try{
        await mongoose.connect(MONGODB_URL);
        console.log("Connected to MongoDB");
    }catch(e){
        console.log("Error connecting to MongoDB", e);
    }
}
```

Nota. La imagen muestra el contenido principal del fichero mongodb.js de la aplicación.

A continuación, se centra en explicar el sistema de autenticación de la aplicación. El sistema de autenticación de la aplicación actual se basa en dos tipos de tokens para

gestionar la sesión del usuario una vez iniciada la sesión con éxito. De esta forma, se garantiza una mayor seguridad y se mejora a la máxima la experiencia de usuario al actualizar su sesión en silencio, evitando que tenga que iniciar sesión repetidamente.

Los dos tipos de token son por un lado el accessToken, que tiene una duración de validez menor, y se utiliza para garantizar que los contenidos protegidos solo son accesibles por los usuarios con el accessToken válido. Por otro lado, el refreshToken, a diferencia al accessToken, tiene una duración de validez mayor y se utiliza para actualizar el accessToken cuando se detecta que este caducado, lo que permite la actualización en silencio de la sesión del usuario.

El flujo de trabajo del sistema de autenticación es el siguiente: una vez que el usuario se ha iniciado sesión con éxito, se obtienen sus accessToken y refreshToken, los cuales se almacenan en la cookie del navegador del usuario. Cuando el usuario se realiza una petición con un accessToken caducado, se activa el interceptor al detectar el estado 401 de la respuesta. El interceptor intenta de enviar una petición para actualizar el accessToken con el refreshToken del usuario. Aquí pueden producir dos resultados: el refreshToken del usuario todavía es válido, se actualiza el accessToken y el interceptor vuelve a intentar la petición original con el accessToken actualizado. Y otro caso es que el refreshToken también está caducado, no se actualiza el accessToken, lo que significa que el usuario debe iniciar la sesión de nuevo para obtener los dos nuevos tokens.

Se ha definido un middleware, un interceptor de Axios y un componente AccessChecker que trabajen de forma colaborativa para conseguir el flujo de trabajo anterior:

- **middleware.ts**: la responsabilidad del middleware es comprobar la existencia del accessToken del usuario. Es importante destacar que el middleware solo se comprueba la existencia del accessToken, no su validez, con el fin de realizar un filtro temprano antes de que comprobar su validez. Asimismo, cuando detecta que el usuario no dispone de un accessToken en su cookie y está en una página protegida, lo redirige directamente a la página de inicio de sesión sin que carguen componentes innecesarios para los usuarios sin permisos. El middleware está configurado de que solo se activa cuando el usuario está navegando por páginas protegidas y se ejecuta antes de cualquier otro proceso, garantizando así la eficiencia de la aplicación.

Figura 45

El middleware de la aplicación

```
import { NextRequest, NextResponse } from "next/server";

export async function middleware(request: NextRequest) {

  // EL middleware solo tiene la responsabildiad de comprobar la existencia de accessToken

  const path = request.nextUrl.pathname;
  const accessToken = request.cookies.get('accessTokenCookie');

  if (!accessToken && path.startsWith('/home')) { // sino, significa el usuario esta en la paginas rela
    const url = request.nextUrl.clone(); // hacer una copia del objeto Request actual para modificar el
    url.pathname = '/';
    return NextResponse.redirect(url);
  }

  console.log('Middleware triggered for:', request.nextUrl.pathname); // ELIMINA CUANDO TERMINAMOS!
  return NextResponse.next();
}

export const config = {
  //matcher: ['/home/:path*'],
  matcher: ['/((?!api|_next/static|_next/image|favicon.ico).*)']
  // recomendacion de documentacion oficial, hace que el middleware ignora estos reques
};
```

Nota. La imagen muestra el contenido principal del fichero middleware.ts de la aplicación.

- **axiosInterceptor.js:** en este mismo archivo se han definido tanto el axios instance como el axios interceptor. El axios instance se utiliza para enviar peticiones de HTTP desde el navegador del cliente al Back-end. A continuación se ha implementado una función, handleTokenRefresh, para manejar el caso en que sea necesario actualizar del accessToken del usuario cuando se intercepta una respuesta con el estado 401(No autorizado). En la función se ha considerado la situación en la que el usuario se realiza varias peticiones al mismo tiempo mientras tiene el accessToken pendiente de actualizar con su refreshToken. Para ello, se ha definido un array de subscribers, donde se guardan todas las peticiones realizadas por el usuario mientras actualiza el accessToken, con el fin de que, cuando tenga el nuevo accessToken disponible, pueda volver a tratar con estas peticiones.

  En cuanto la función, primero se comprueba si el error es de tipo 401 y si no se ha reintentado. Si cumple la condición, se comprueba si ya existe una petición en el proceso de actualización. Si es caso confirmativo, se guarda la petición actual en la cola de suscriptores. En caso contrario, significa que la petición actual es la primera petición que inicia el proceso de actualización, por lo que se marcan las variables correspondientes a true (para bloquear) y se envía al Back-end la petición de renovación del accessToken con el refreshToken. Si el resultado es satisfactorio, se vuelve a intentar la petición original y también las

peticiones almacenadas en la cola de suscriptores. En el caso contrario, significa que el refreshToken también está expirado, el usuario debe iniciar sesión de nuevo.

Y por lo último, se ha configurado el interceptor que se intercepta las peticiones con respuesta de error para procesar la función de *handleTokenRefresh* cuando se trata de un error 401.

Figura 46

El interceptor e instancia de axios de la aplicación

```
axiosInstance.interceptors.response.use(
  response => response, // si no falla, devolvemos la respuesta
  error => handleTokenRefresh(error)
);

export default axiosInstance;
const handleTokenRefresh = async (error) => {
  const originalRequest = error.config;

  if (error.response?.status === 401 && !originalRequest._retry) {
    if (isRefreshing) {
      return new Promise((resolve) => {
        subscribers.push(() => resolve(axiosInstance(originalRequest))); // manejar caso de que se realiza multiples rec
      });
    }

    originalRequest._retry = true;
    isRefreshing = true;

    try {
      const { data } = await axios.post('/api/auth/refresh-token', {}, {
        withCredentials: true,
        headers: {
          'Cache-Control': 'no-cache' // para que no usa contenido de cache! Comprueba siempre si el contenido actualzia
        }
      });

      const retryOriginal = await axiosInstance(originalRequest);
      subscribers.forEach(cb => cb());
      subscribers = [];
      return retryOriginal;

    } catch (refreshError) {
      console.error('No se ha podido actualizar el accessToken', refreshError);
      window.location.href = '/'; // par que redirige a pagina de login
      return Promise.reject(refreshError);
    } finally {
      isRefreshing = false;
    }
  }
  return Promise.reject(error);
};
const axiosInstance = axios.create({
  baseURL: 'http://localhost:3000/api',
  timeout: 5000,
  withCredentials: true
});

let isRefreshing = false;
let subscribers = [];
```

Nota. La imagen muestra el contenido principal del fichero axiosInterceptors.js de la aplicación.

- **AccessChecker**: mientras que el interceptor se encarga de monitorizar las solicitudes HTTP iniciadas por el usuario y el middleware solo se comprueba la existencia del accessToken cuando el usuario se navega entre las páginas protegidas. Falta una validación del accessToken cuando el usuario solo se navega entre páginas y no realiza ninguna petición HTTP. Por esta razón, se ha implementado el componente AccessChecker para solucionar este problema. Dicho componente se sitúa por encima de las páginas protegidas, y se

monitoriza la ruta en la que se encuentra el usuario. Cuando cambia de página y el middleware comprueba la existencia del accessToken, se inicia la petición al back-end para validarlo. En caso se valida, se permite al usuario continuar navegando por la página protegida, en caso contrario se redirige a la página de inicio de sesión.

Figura 47

El componente AccessChecker

```
export default function AccessChecker({children}:accessCheckerProps){

    const router = useRouter();
    const path = usePathname();
    const {showToast} = useContext(ToastContext);
    const {user, waiting} = useContext(UserContext);

    useEffect(() => {
        const checkAccessToken = async () => {
            try {
                await axiosInstance.get('/auth/check-access-token');
            } catch (e) {
                showToast({ msg: 'EL session ha sido expirado, volver a iniciar.', type: 'Bad', visible: true })
                router.push('/')
            }
        };
        checkAccessToken();
    }, [user,router, path]) // cada vez si el usuario ir a paginas protegidas, se comprueba su token de acceso

    if(waiting){
        return<div>Cargandno contenido</div>
    }
    return <>{children}</>
}
```

Nota. La imagen muestra el contenido principal del componente AccessChecker de la aplicación.

4.2. Lado cliente

El lado cliente está implementado con la biblioteca de JavaScript React, que se caracteriza por permitir crear la interfaz de usuario en formato de componente, lo que mejora la modularidad y la reusabilidad de la aplicación.

En esta sección se explica en detalle la lógica de implementación de los componentes principales de la aplicación, así como las funcionalidades que se ofrecen a los usuarios.

En la carpeta app, se destacan los siguientes componentes de front-end:

layout.tsx raíz:

En Next.js, es necesario definir un layout raíz en el directorio app. En este layout se incluye la estructura general del proyecto, y las etiquetas html y body son elementos obligatorios. Además, se ha incluido también la etiqueta head junto con los metadatos necesarios con el fin de mejorar el SEO de la aplicación.

Dentro de la etiqueta body, se encuentra el componente SkeletonTheme, el cual es proporcionado por la librería React-loading-skeleton, para que unificar el estilo de todos los esqueletos de la aplicación. El componente BootstrapClient se ha añadido debido a que es necesario crear un componente cliente para importar el JavaScript de Bootstrap en el lado cliente y poder interactuar con el DOM Tree y realizar cambios. Y el componente ToastProvider como se ha comentado en la sección anterior, permite compartir el único Toast entre toda la aplicación.

Figura 48

Root Layout

```
export default function RootLayout({ children }: RootLayoutProps) {
  return (
    <html lang="en">
      <head>
        <title>CoWorld</title>
        <meta charSet="UTF-8"></meta>
        <meta name="viewport" content="width=device-width, initial-scale=1.0"></meta>
        <meta name="description" content="Una aplicación de busqueda de empleo orientado a las personas con discapacidad." />
        <meta name="keywords" content="CoWorld, empleo, discapacidad, busqueda de empleo, personas con discapacidad" />
        <meta name="author" content="Jiale Wang" />
      </head>
      <body className={inter.className}>
      <SkeletonTheme baseColor="#f2f4f4" highlightColor="#444">
        <BootstrapClient/>
        <ToastProvider>
          <main className='container-fluid p-0'>{children}</main>
        </ToastProvider>
        </SkeletonTheme>
      </body>
    </html>
  )
}
```

Nota. La imagen muestra el contenido principal del componente *layout raíz* de la aplicación.

LogInPage:

La página de la ruta raíz / de la aplicación, que representa el punto de entrada de esta. En esta página se muestra el formulario de inicio de sesión para los usuarios con el rol de candidato. Como se ha mencionado en la sección anterior, cada formulario de la aplicación tiene su propio contexto que se comparte entre los componentes del formulario. En el formulario de inicio de sesión se recopilan el email y la contraseña introducido por el usuario, y se envían de forma conjunta mediante el hook personalizado *authFetch* al back-end para realizar validaciones e inicialmente, los dos campos tendrán valores vacíos. Durante el tiempo de espera de la respuesta de la petición, se muestra el componente snipper en el botón de envío para indicar que está procesando la petición hasta que se retorna la respuesta.

Figura 49

Lógica interna de la página LoginPage

```
export default function LoginInPage() {
    const authFetch = useAuthFetch();
    const {isLoading,setIsLoading} = useSnipper();
    const [oldValues, setOldValues] = useState<FormProperties>({});

    const login = async (data:any)=>{
        setIsLoading(true)
        await authFetch({endpoint:'login',nextPath:'/home',fetchdata:data})
        setIsLoading(false)
        console.log('fin')
    }
    useEffect(() => {
            const initialValues = {
                email: '',
                password: ''
            }
            setOldValues(initialValues)
        }, [])
```

Nota. La imagen muestra la función de manejo del evento submit del formulario de inicio de sesión de las cuentas del rol candidato y el estado inicial del formulario.

Figura 50

LogInPage

```
export default function LoginInPage() {
    return(
        <div className="card m-0 p-0" style={{ width: '100%', height: '100vh' }}>
            <div className="row g-0 h-100">
                <div className="col-sm-5 d-none d-md-block">
                    <img src="/imgs/team-green.jpg" className="img-fluid" alt="Log in image" style={{ width: '100%', height: '100%' }}/>
                </div>
                <div className="col-md-7">
                    <div className="card-body" style={{ height:'100%', flexDirection:'column', display:'flex', justifyContent:'center', ali

                        <div className='row mb-3'>
                            <div className='col'></div>
                            <img src="/imgs/CoWorldLogoLogin.png" alt="CoWorld logo" className='col-sm-8' style={{ width: '80%', height: 'a
                        <div className='col'></div>
                        </div>
                        <Form title="Iniciar sesión" onSubmit={login} oldValues={oldValues}>
                            <Form.Input
                            id="email"
                            htmlfor="email"
                            label="EMAIL"
                            type="text"
                            placeholder="Introduce tu email"
                            className='mb-3'
                            required={true}
                            validationClass='invalid-feedback'
                            validationMsg='Indique un email válido!'/>
                            <Form.Input id="password" htmlfor="password" label="CONTRASEÑA" type="password" placeholder="Introduce tu contr
                            <Form.Links href="/forget-password" text="¿Has olvidado tu contraseña? " linkText='Recuperar contraseña'/>
                            <Form.SubmitButton text="ENTRAR" loading={isLoading}/>
                            <div className='text-center'>
                            <Form.Links href="/register" text="¿No tienes cuenta? " linkText='Regístrate'/>
                            <Form.Links href="/company-login" text="¿Buscaba iniciar como empresa? " linkText='Iniciar sesión como empresa'
                            </div>
                        </Form>
                    </div>
                </div>
            </div>
        </div>
    );
```

Nota. La imagen muestra el contenido que se renderiza la página *LogInPage*.

CompanyLoginInPage:

La página de inicio de sesión para las cuentas empresariales. Funciona de la misma forma que la página LogInPage para los usuarios con el rol de candidato. La razón de separar los dos procesos de inicios de sesión en diferentes páginas es, por un lado, evitar posibles confusiones a los usuarios a la hora de iniciar sesión y, por otro lado, facilitar el proceso de identificación del rol del usuario haciendo diferentes peticiones HTTP de inicio de sesión al back-end desde rutas distintas.

Figura 51

Lógica interna del componente CompanyLoginPage

```
export default function CompanyLoginInPage() {
    const authFetch = useAuthFetch();
    const {isLoading,setIsLoading} = useSnipper();
    const [oldValues, setOldValues] = useState<FormProperties>({});
    const login = async (data:any)=>{
        setIsLoading(true)
        await authFetch({endpoint:'company-login',nextPath:'/company-home',fetchdata:data})
        setIsLoading(false)
    }
        useEffect(() => {
                const initialValues = {
                    email: '',
                    password: ''
                }
                setOldValues(initialValues)
            }, [])
```

Nota. La imagen muestra la función de manejo del evento submit del formulario de inicio de sesión de las cuentas empresariales y el estado inicial del formulario.

RegisterPage:

Es la página de registro de la cuenta del rol candidato, funciona de manera similar que las páginas de inicio de sesión comentadas anteriormente. La diferencia es que incluye más componentes en el formulario de registro, ya que es necesario recopilar información personal del candidato. Se observa que el formulario de registro es de tipo *ValidationForm*. La diferencia con el *Form* normal es que se marca con el borde roja el input cuando falta de rellenar alguno de los campos obligatorios y con el borde verde cuando están rellenados, lo que permite avisar a los usuarios de forma más clara y ayudarles en el proceso de registro de manera más eficiente.

Figura 52

Lógica interna de la página RegisterPage

```
export default function RegisterPage(){
    const authFetch=useAuthFetch();
    const {isLoading,setIsLoading}=useSnipper();

    const register = async(data:any)=>{
        console.log('Registrado');
        setIsLoading(true);
        await authFetch({endpoint:'register',nextPath:'/home',fetchdata:data})
        setIsLoading(false);
        console.log('Finished')
    }
```

Nota. La imagen muestra la función de manejo del evento submit del formulario de registro de una cuenta de candidato.

Figura 53

RegisterPage

```
export default function RegisterPage(){
    return(
        <div className="card m-0 p-0" style={{ width: '100%', height: '100vh' }}>
            <div className="row g-0 h-100">
                <div className="col-sm-5 d-none d-md-block">
                    <img src="/imgs/team-green.jpg" className="img-fluid" alt="Log in image" style={{ width: '100%', height: '100%' }}/>
                </div>
                <div className="col-md-7">
                    <div className="card-body" style={{ height:'100%', flexDirection:'column', display:'flex', justifyContent:'center', alignItems:'center' }}>

                        <div className='row mb-3'>
                            <div className='col'></div>
                            <img src="/imgs/CoWorldLogoLogin.png" alt="CoWorld logo" className='col-sm-8' style={{ width: '80%', height: 'auto' }}/>
                            <div className='col'></div>
                        </div>
                        <ValidationForm title="Únete a CoWorld " onSubmit={register} >
                            <ValidationForm.Input id="firstname" htmlfor="firstname" label="NOMBRE" type="text" className="col-md-6" required={true}
                            validationClass="invalid-feedback" validationMsg="Nombre es necesario!"/>
                            <ValidationForm.Input id="lastname" htmlfor="lastname" label="APELLIDOS" type="text" className="col-md-6" required={true}
                            validationClass="invalid-feedback" validationMsg="Apellidos es necesario!"/>
                            <ValidationForm.Input id="email" htmlfor="email" label="EMAIL" type="text" className="col-md-12" required={true}
                            validationClass="invalid-feedback" validationMsg="Email es necesario!"/>
                            <ValidationForm.Input id="password" htmlfor="password" label="CONTRASEÑA" type="password" className="col-md-12" required={true} help={tr
                            validationClass="invalid-feedback" validationMsg="Contraseña es necesario!"/>
                            <ValidationForm.SubmitButton text="UNIRSE" loading={isLoading} />
                            <div className='text-center'style={{marginTop:0}}>
                            <ValidationForm.Links href="/" text="¿Ya tienes cuenta? " linkText='Iniciar sesión'/>
                            <ValidationForm.Links href="/company-register" text="¿Buscaba registrarse como empresa? " linkText='Crear cuenta de empresa'/>
                            </div>
                        </ValidationForm>
                    </div>
                </div>
            </div>
        </div>
    );
}
```

Nota. La imagen muestra el contenido que se renderiza la página *RegisterPage*.

CompanyRegisterPage:

La página para realizar registrar una cuenta empresarial (solo del rol company). Funciona de la misma forma que la página RegisterPage, solo que el formulario de registro de la cuenta empresarial incluye más campos de datos que el formulario de registro de la cuenta de candidato, ya que se necesitan más datos para validar la existencia real de la empresa. Y la razón por la que se han separarado los dos procesos de registro es la misma que se ha mencionado en la explicación de la página CompanyLogInPage, para evitar posibles confusiones y manejar mejor los roles a la hora de crear cuentas mediante diferentes peticiones HTTP desde distintas páginas.

Figura 54

Lógica interna del componente CompanyRegisterPage

```
export default function CompanyRegisterPage(){
    const authFetch=useAuthFetch();
    const {isLoading,setIsLoading}=useSnipper();

    const companyRegister = async(data:any)=>{
        setIsLoading(true);
        await authFetch({endpoint:'company-register',nextPath:'/company-home',fetchdata:data})
        setIsLoading(false);
    }
```

Nota. La imagen muestra la función de manejo del evento submit del formulario de registro de una cuenta empresarial.

Figura 55

CompanyRegisterPage

```
export default function CompanyRegisterPage(){
    return(
        <div className="card m-0 p-0" style={{ width: '100%', height: '100%' }}>
            <div className="row g-0 h-100">
                <div className="col-sm-5 d-none d-md-block">
                    <img src="/imgs/company-logo.jpg" className="img-fluid" alt="Log in image" style={{ width: '100%', height: '100%', overflow:'hidden' }}/>
                </div>
                <div className="col-md-7">
                    <div className="card-body" style={{ height:'100%',flexDirection:'column', display:'flex', justifyContent:'center', alignItems:'center' }}>

                        <div className='row mb-3'>
                            <div className='col'></div>
                            <img src="/imgs/CoWorldLogoLogin.png" alt="CoWorld logo" className='col-sm-8' style={{ width: '80%', height: 'auto' }}/>
                            <div className='col'></div>
                        </div>
                        <ValidationForm title="Crear una cuenta de empresa en CoWorld" onSubmit={companyRegister} >
                            <ValidationForm.Input id="firstname" htmlfor="firstname" label="SU NOMBRE " type="text" className="col-md-6" required={true}
                            validationClass="invalid-feedback" validationMsg="Nombre es necesario!"/>
                            <ValidationForm.Input id="lastname" htmlfor="lastname" label="SUS APELLIDOS" type="text" className="col-md-6" required={true}
                            validationClass="invalid-feedback" validationMsg="Apellidos es necesario!"/>
                            <ValidationForm.Input id="email" htmlfor="email" label="EMAIL" type="text" className="col-md-12" required={true}
                            validationClass="invalid-feedback" validationMsg="Email es necesario!"/>
                            <ValidationForm.Input id="password" htmlfor="password" label="CONTRASEÑA" type="password" className="col-md-12" required={true} help={true
                            validationClass="invalid-feedback" validationMsg="Contraseña es necesario!"/>
                            <ValidationForm.Input id="companyName" htmlfor="companyName" label="NOMBRE DE LA EMPRESA" type="text" className="col-md-6" required={true}
                            validationClass="invalid-feedback" validationMsg="Nombre de la empresa es necesario!"/>
                            <ValidationForm.Input id="cif" htmlfor="cif" label="C.I.F / N.I.F" type="text" className="col-md-6" placeholder="Ej: B12345678" required={
                            validationClass="invalid-feedback" validationMsg="C.I.F es necesario!"/>
                            <div style={{marginTop:'2rem'}}>
                            <ValidationForm.SubmitButton text="UNIRSE" loading={isLoading} />
                            </div>
                            <div className='text-center'style={{marginTop:0}}>
                            <ValidationForm.Links href="/company-login" text="¿Ya tienes cuenta? " linkText='Iniciar sesión'/>
                            <ValidationForm.Links href="/register" text="¿Buscaba registrar como candidato? " linkText='Crear una cuenta de candidato' />
                            </div>
                        </ValidationForm>
                    </div>
                </div>
            </div>
```

Nota. La imagen muestra el contenido que se renderiza la página *CompanyRegisterPage*.

ForgetPwdPage:

La página para solicitar el restablecimiento de la contraseña de la cuenta solicitada. Esta página contiene simplemente un formulario para introducir el email (que representa la cuenta de la aplicación). En caso de que se compruebe que el email está registrado en la base de datos, el usuario recibirá un email para empezar el proceso de restablecimiento de la contraseña.

Figura 56

Lógica interna del componente ForgetPwdPage

```
export default function ForgetPwdPage(){
    const authFetch = useAuthFetch();
        const {isLoading,setIsLoading} = useSnipper();
        const [oldValues, setOldValues] = useState<FormProperties>({});

        const forgetPwd = async (data:any)=>{
            console.log('fogrt starting')
            setIsLoading(true)
            await authFetch({endpoint:'forget-pwd',nextPath:'/',fetchdata:data})
            setIsLoading(false)
            console.log('fin')
        }

    useEffect(() => {
        const initialValues = {
                email: ''
        }
        setOldValues(initialValues);
    }, [])
```

Nota. La imagen muestra la función de manejo del evento submit del formulario de restablecimiento de contraseña de los usuarios y el estado inicial del formulario.

Figura 57

ForgetPwdPage

Nota. La imagen muestra el contenido que se renderiza la página *ForgetPwdPage*.

ResetPwdPage:

Es la página donde se realiza el proceso de restablecimiento de contraseña. El acceso a esta página se realiza mediante el correo que recibe el usuario cuando solicita el restablecimiento de su contraseña. Además, la página estará configurada con un token de validez de una hora desde la recepción del email con el aviso de restablecimiento de contraseña por motivos de seguridad.

En consecuencia, el primer paso que se realiza al cargar la página es intentar obtener el token del enlace de la página actual, para, en posteriormente enviarlo junto con los datos recopilados del formulario de la página (que son: la contraseña nueva y la confirmación de la misma) al back-end para realizar las validaciones necesarias.

Figura 58

Lógica interna del componente ResetPwdPage

```
useEffect(() => {
    const urlToken = searchParams.get('token');
    if (urlToken) setToken(urlToken);
    const initialValues = {
        pwd: '',
        confirmpwd: ''
    }
    setOldValues(initialValues);
}, [searchParams]);


const resetPwd = async(data:any)=>{
    setIsLoading(true)
    const config:AxiosRequestConfig<any> ={
        headers:{
            token: token ?? ''
        }
    }
    await authFetch({endpoint:'/reset-pwd',nextPath:'/',fetchdata:data,config:config})
    setIsLoading(false)
}
```

Nota. La imagen muestra el proceso de obtención del token desde el enlace de la página ResetPwdPage y la configuración del estado inicial del formulario de dicha página. Así como el contenido de la función de manejo del evento submit del formulario.

Figura 59

ResetPwdPage

```
export default function ResetPwdPage(){
    return(
        <div className={`${styles.allViewPort} row`}>
            <div className='col'></div>
            <div className='col-md-7 col-sm-12'>
                <div className='card'>
                    <div className={`${styles.center} card-body`} >
                        <Form title="Restablecer tu contraseña" onSubmit={resetPwd} oldValues={oldValues}>
                            <Form.Input
                                id="pwd"
                                htmlfor="pwd"
                                label="NUEVA CONTRASEÑA"
                                type="password"
                                placeholder="Introduce tu nueva contraseña"
                                className='mb-3'
                                help={true}
                                required={true}
                            />
                            <Form.Input
                                id="confirmpwd"
                                htmlfor="confirmpwd"
                                label="CONFIRMAR CONTRASEÑA"
                                type="password"
                                placeholder="Confirmar tu nueva contraseña"
                                className='mb-3'
                                required={true}
                            />
                            <Form.SubmitButton text="RESTABLECER" loading={isLoading}/>
                        </Form>
                    </div>
                </div>
            </div>
            <div className='col'></div>
        </div>
    );
}
```

Nota. La imagen muestra el contenido que se renderiza la página *ResetPwdPage*.

ResetPwdOperatorPage:

Esta página se funciona de la misma manera que la página ResetPwdPage. Su función principal es activar la cuenta de un operador, y el acceso a la misma se realiza mediante el correo que recibe el usuario cuando hay cuenta empresa que le añade como un operador. Además, en el correo también se adjunta la contraseña temporal establecida por la empresa. El usuario debe introducir la contraseña temporal para verificar su identidad, así como la nueva contraseña y su confirmación, que se establecen para su uso futuro. Del mismo modo, se obtiene el token del enlace actual de la página en el momento de cargarla y se envía junto con los datos del formulario al Back-end para realizar validaciones.

Figura 60

Lógica interna del componente ResetPwdOperatorPage

```
useEffect(() => {
    const urlToken = searchParams.get('token');
    if (urlToken) setToken(urlToken);
    const initialValues = {
        tempPwd:'',
        pwd: '',
        confirmpwd: ''
    }
    setOldValues(initialValues);
}, [searchParams]);


const resetPwd = async(data:any)=>{
    setIsLoading(true)
    const config:AxiosRequestConfig<any> ={
        headers:{
            token: token ?? ''
        }
    }
    await authFetch({endpoint:'/reset-pwd-operator',nextPath:'/company-login',fetchdata:data,config:config})
    setIsLoading(false)
}
```

Nota. La imagen muestra el proceso de obtención del token desde el enlace de la página ResetPwdOperatorPage y la configuración del estado inicial del formulario de dicha página. Así como el contenido de la función de manejo del evento submit del formulario.

Figura 61

ResetPwdOperatorPage

```
export default function ResetPwdOperatorPage(){
    return(
        <div className={`${styles.allViewPort} row`}>
            <div className='col'></div>
            <div className='col-md-7 col-sm-12'>
                <div className='card'>
                    <div className={`${styles.center} card-body`} >
                        <Form title="Cambiar su contraseña" onSubmit={resetPwd} oldValues={oldValues}>
                            <Form.Input
                                id="tempPwd"
                                htmlfor="tempPwd"
                                label="CONTRASEÑA TEMPORAL"
                                type="password"
                                placeholder="Introduce tu contraseña temporal"
                                className='mb-3'
                                required={true}
                            />
                            <Form.Input
                                id="pwd"
                                htmlfor="pwd"
                                label="NUEVA CONTRASEÑA"
                                type="password"
                                help={true}
                                placeholder="Introduce tu nueva contraseña"
                                className='mb-3'
                                required={true}
                            />
                            <Form.Input
                                id="confirmpwd"
                                htmlfor="confirmpwd"
                                label="CONFIRMAR CONTRASEÑA"
                                type="password"
                                placeholder="Confirmar tu nueva contraseña"
                                className='mb-3'
                                required={true}
                            />
                            <Form.SubmitButton text="RESTABLECER" loading={isLoading}/>
                        </Form>
                    </div>
```

Nota. La imagen muestra el contenido que se renderiza la página *ResetPwdOperatorPage*.

En este punto se termina la presentación de todas las páginas relacionadas con el sistema de autenticación de la aplicación. A continuación se introducirán las páginas protegidas de los usuarios con el rol de candidato.

HomeLayout:

En este layout se definen los elementos comunes para todas las páginas protegidas dirigidas a los usuarios con el rol de candidato. En primer lugar, se encuentra el componente *AccessChecker* que se ha comentado con detalle

anteriormente. Su función es validar el accessToken del candidato cuando navega entre las páginas protegidas para garantizar la seguridad del sistema. En seguida, se encuentra el componente *UserProvider,* que también se ha comentado detalladamente en la sección de explicación de la *carpeta context*. Es el contexto que comparte la información del candidato actual entre sus componentes hijos, para facilitar la gestión de la información del usuario. Y por último, dada que todas las páginas protegidas de los candidatos incluyen un navbar exclusivo para ellos en la parte superior. En este navbar se ofrecen los accesos rápidos a distintas páginas con el fin de mejorar la experiencia del usuario. Entre ellos, los accesos rápidos son:

- **El logo "CoWorld":** acceso rápido a la página de inicio (/home) de los candidatos, ya que las aplicaciones en común tienen dicha funcionalidad.
- **Ofertas:** acceso rápido a la página de inicio (/home) de los candidatos.
- **Mis Empleos:** acceso rápido a la página donde puede consultar las ofertas acerca del candidato actual en distintos estados (/home/view-applications).
- **Perfil**: acceso rápido a la página de perfil del candidato (/home/profile).
- **Ayuda**: acceso rápido a la página de ayuda a los candidatos (/home/help).
- **El logo de "Cerrar sesión"**: enlace en formato de imagen de un botón para cerrar la sesión del candidato.

Figura 62

Lógica interna del componente NavbarCandidate

```
const handleActivePage = (page:string) => {
    setActivePage(page);
}

const handleLogOut = async () =>{
    try{

    const {data} =await axios.post(`/api/auth/logout`)
    showToast({msg:data.sucess, type:'Good',visible:true})
    router.push('/')
    }catch(e){
        showToast({msg:e.response.data.error as string, type:'Bad',visible:true})
    }
}
const handleProfile = async () =>{
    try{
    await getUser()
    router.push('/home/profile')
    }catch(e){
        showToast({msg:e.response.data.error as string, type:'Bad',visible:true})
    }
}
```

Nota. En la imagen muestran las funciones de manejo del evento de clic del enlace Profile y del logo de cerrar sesión, así como la función para destacar la página donde se encuentra el candidato del componente navbarCandidate.

Figura 63

NavbarCandidate

```
export function NavbarCandidate(){
    return (
        <nav className="navbar navbar-expand-lg navbar-light " style={{marginBottom:'20px', backgroundColor:"#d5f5e3"}}>
            <div className="container-fluid">
                <a className={`${styles.logo} navbar-brand logo`} href="/home"
                onClick={() => handleActivePage('')}>CoWorld</a>
                <button classNname="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" ar
                <span className="navbar-toggler-icon"></span>
                </button>
                <div className="collapse navbar-collapse" id="navbarNav">
                <ul className="navbar-nav linknav">
                    <li className="nav-item">
                    <a className={`${styles.hov} nav-link ${activePage ==='ofertas' ? styles.active : ''}`} aria-current="page" href="/home"
                    onClick={()=>{handleActivePage('ofertas')}}>Ofertas</a>
                    </li>
                    <li className="nav-item">
                    <a className={`${styles.hov} nav-link ${activePage ==='misEmpleos' ? styles.active : ''}`} href="/home/view-applications"
                    onClick={()=> handleActivePage('misEmpleos')}>Mis Empleos</a>
                    </li>
                    <li className="nav-item">
                    <a className={`${styles.hov} nav-link ${activePage ==='perfil' ? styles.active : ''}`}  href="/home/profile"
                    onClick={()=>{handleActivePage('perfil'), handleProfile()}}>Perfil</a>
                    </li>
                    <li className="nav-item">
                    <a className={`${styles.hov} nav-link ${activePage ==='ayuda' ? styles.active : ''}`} href="/home/help"
                    onClick={()=>{handleActivePage('ayuda')}}>Ayuda</a>
                    </li>
                </ul>
                <a className="nav-link ms-auto" href="/"
                    onClick={()=>{handleLogOut()}}><img src='/imgs/power-off.png' aria-label='Haz click al link para cerrar sesion.' style={{paddingRight:'8px'}} alt='Lo
                </div>
            </div>
        </nav>
    );
}
```

Nota. La imagen muestra el contenido que renderiza por el componente *NavbarCandidate*.

Figura 64

HomeLayout

```
export default function HomeLayout({ children }: HomeLayoutProps) {
  return (
    <AccessChecker>
      <UserProvider>
            <NavbarCandidate/>
            {children}
      </UserProvider>
    </AccessChecker>
  )
}
```

Nota. La imagen muestra el contenido principal del componente *HomeLayout*

JobListDisplay:

El componente está diseñado para listar las ofertas de trabajo recibidas. Es necesario pasar las ofertas de una lista, y, según la longitud de la lista pasada, se muestra el contenido correspondiente: si se trata de una lista vacía, se muestra un texto de aviso, y si no, se muestra un listado de ofertas en formato de carta (véase la figura).

Figura 65

JobListDisplay

Nota. La imagen muestra la forma de listar las ofertas de trabajo por el componente *JobListDisplay*, en caso de que la longitud de la lista sea mayor que cero.

JobFilter:

Es un componente esencial para ayudar a los candidatos a realizar búsquedas de empleo más ajustadas a sus necesidades. El componente abarca un conjunto de filtros, que permiten a los usuarios filtrar las ofertas por distintas variables, cuales son: *Ubicación, Discapacidad, Presencial/Remoto, Jornada Laboral, Categoría, Experiencia, Estudios mínimos* y *Busco Práctica*. Entre ellos, excepto las variables *Experiencia* y *Busco Práctica*, todos se muestran en el formato de modal y permiten realizar selecciones múltiples. La variable *Experiencia* se muestra en formato de dropdown y mientras la variable *Busco Práctica* es un checkbox.

En el componente, se ha definido un estado para registrar los valores de cada filtro seleccionado por el usuario. Además, se ha implementado un useEffect para que monitorizar los cambios en los filtros y mantener actualizado dicho estado. De esta forma, el componente puede enviar inmediatamente los valores de filtros a su componente padre *HomePage,* el cual que envía una solicitud al back-end para obtener los resultados filtrados y mostrarlos al usuario, mejorando así la experiencia de usuario. Por otro parte, se ha considerado también la posibilidad de realizar varios cambios en los filtros en un mismo tiempo lo que provoca un bloqueo en el back-end. Para evitar esto, se ha configurado un timeout de un intervalo de 500 milisegundos cada vez que se llama la función de actualización del estado del componente.

Figura 66

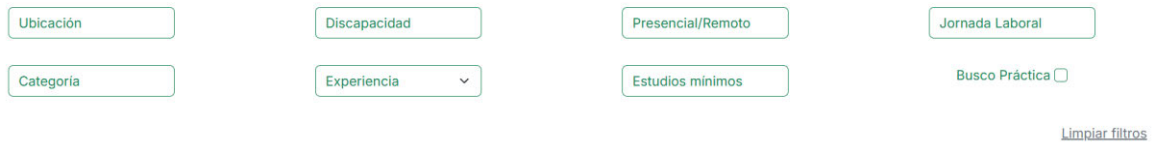
Lógica interna del componente JobFilter

```
export default function JobFilter ({filters,setFilters}:Props){

    useEffect(()=>{
        const debounce = setTimeout(()=>{
            setFilters(filters);
        },500)
        return ()=>clearTimeout(debounce);
    },[filters])

    const handleFilters=(id:keyof JobFilters, value:string[]|{type:string, degree:number}[]|string|boolean)=>{
        setFilters({ ...filters, [id]: value })
    }

    const handleReset=()=>{
        setFilters({
            city:[],
            disabilities:DISABILITIES_INITIAL_VALUE,
            mode:[],
            workHours:[],
            workCategory:[],
            experience:"",
            minumumEducation:[],
            intership:false
        });
    }
```

Nota. La imagen muestra el useEffect definido para monitorizar el estado del componente JobFilter, así como el contenido de la función para resetear todos los filtros a vacíos.

Figura 67

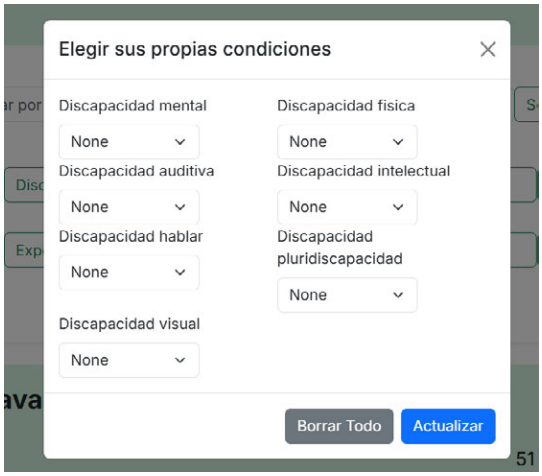
JobFilter



Nota. La imagen muestra la forma que se renderiza el componente JobFilter.

Figura 68

Modal de selección de un filtro



Nota. La imagen se muestra como un ejemplo del formato de un filtro de selección múltiple.

HomePage:

La página principal de la aplicación, donde permite buscar y filtrar las ofertas de trabajo. Debido a que es posible que el usuario quiera aplicar filtros a las ofertas de trabajo obtenidas tras de realizar una búsqueda, se han definido tres estados para abordar esta situación. Uno es para guardar las ofertas de trabajo filtradas(*jobList*), otro es para guardar las ofertas de trabajo buscadas(*searchedList*), y el último es un estado de tipo booleano(*searched)*, y se usa para determinar si la lista de trabajo que se está mostrando es del estado searchedList.

Si el usuario solo está aplicando filtros a las ofertas de trabajo, cada vez que se cambian los valores de los filtros, se envía los valores de los filtros actualizados al back-end para obtener el resultado filtrado (comentado en el componente anterior) y mostrarlo al usuario con el componente JobListDisplay. Si el usuario ha realizado la búsqueda (con el estado *searched* a true), y está aplicando filtros, significa que el usuario quiere aplicar los filtros al resultado de la búsqueda, y esta operación de filtración se realiza directamente desde el front-end sobre el estado de la lista de ofertas buscadas. Asimismo, el estado *searched* se vuelve a false cuando el usuario hace una búsqueda vacía o actualiza la página.

Las ofertas de trabajo se muestran según el orden establecido por el algoritmo de recomendación. Tanto el detalle de la implementación de la filtración, la de búsqueda y la del algoritmo de recomendación se explicará detalladamente en la sección del lado servidor.

Figura 69

Lógica interna del componente HomePage Parte 1

```
export default function HomePage(){
    useEffect(()=>{
        const fetchFilteredJobs = async (page:number)=>{

            try{

                // convertimos filtros como parametros de url antes de hacer llamada
                const param ={
                    city: filter.city.join(','),
                    disabilities: encodeURIComponent(JSON.stringify(filter.disabilities)),
                    mode: filter.mode.join(','),
                    workHours: filter.workHours.join(','),
                    workCategory: filter.workCategory.join(','),
                    experience: filter.experience,
                    minumumEducation: filter.minumumEducation.join(','),
                    intership:filter.intership,
                    page:page
                }
                setLoading(true);

                const {data} = await axiosInstance.get(`/candidate-home/get-jobs` ,{
                    params: param,
                    withCredentials:true,
                    signal:controller.signal
                })

                setJobList(data.jobList);
                setCurrentPage(page);
                setTotalPages(data.totalPage);
                if(data.totalPage>5){
                    setDemonstratingPages(5);
                    setPaginationLimit(5);
                }else{
                    setDemonstratingPages(data.totalPage);
                    setPaginationLimit(data.totalPage);
                }
                setError("")
```

Nota. La imagen muestra parte del useEffect que se ejecuta cuando el usuario solo está filtrando las ofertas de trabajo.

Figura 70

Lógica interna del componente HomePage Parte 2

```
    if(debounce){
        clearTimeout(debounce)
    }
    if(!searched){
        const timeOut = setTimeout(()=>fetchFilteredJobs(1),500)
        setDebounce(timeOut)
    }else{
        var jobs:IJobAndCompany[] = filterJobs(jobList,filter);
        setTotalPages(Math.ceil(jobs.length/5));
        if(Math.ceil(jobs.length/5)>5){
            setDemonstratingPages(5);
            setPaginationLimit(5);
        }else{
            setDemonstratingPages(Math.ceil(jobs.length/5));
            setPaginationLimit(Math.ceil(jobs.length/5));
        }
        jobs = jobs.slice(0,5);
        setSearchedList(jobs);
        setCurrentPage(1);

    }

    return () => {
        controller.abort()
        clearTimeout(debounce)
    }
},[filter])
```

Nota. La imagen muestra parte del useEffect que se ejecuta cuando el usuario solo está filtrando las ofertas de trabajo tras de realizar una búsqueda.

jobViewPage

La página que se muestra el detalle de una oferta de trabajo. En la parte superior de la página disponen opciones para guardar o aplicar el puesto. El usuario puede guardar o deshacer el guardado de la oferta. Sin embargo, una vez aplicado el puesto, ya no es posible deshacer la aplicación y se mostrará con un mensaje de aviso indicando que el puesto ya está aplicado.

La información detallada de la oferta se divide en secciones. En primer lugar, se encuentran con las variables de la oferta junto con iconos. A continuación se muestran los requisitos de aceptación de discapacidad de los candidatos, cada requisito se muestra con una carta de color diferente para distinguir mejor entre los tipos de discapacidad. La siguiente sección muestra los requisitos del puesto, y a continuación, la descripción de la oferta.

Al final de la página disponen también hay opciones para guardar o aplicar el trabajo, según el caso del candidato, de esta forma, el usuario no tiene que volver al inicio de la página para realizar estas operaciones, lo que mejora la experiencia del usuario.

Figura 71

Lógica interna del componente jobViewPage

```
export default function jobViewPage(){
    const handleSaveJob= async()=>{
        setIsLoading(true)
        try{
            const response = await axiosInstance.post(`/candidate-home/save-job`,{job:job._id},{
                withCredentials:true
            })
            showToast({msg:response.data.sucess, type:'Good',visible:true})
            getUser()
            setSaved(saved? false:true);
            setIsLoading(false);
        }catch(e:any){
            showToast({msg:e.response.data.error as string, type:'Bad',visible:true})
            setIsLoading(false);
        }
    }

    const handleApplyJob= async()=>{
        setIsLoading(true)
        setdisableApply(true)
        try{
            const response = await axiosInstance.post(`/candidate-home/apply-job`,{job:job._id},{
                withCredentials:true
            })
            showToast({msg:response.data.sucess, type:'Good',visible:true})
            if(user.savedJob.find(elem => elem===job._id)){
                try{
                    const response = await axiosInstance.post(`/candidate-home/save-job`,{job:job._id},{
                        withCredentials:true
                    })
                    showToast({msg:response.data.sucess, type:'Good',visible:true})
                    getUser()
                    setSaved(!saved)
                }catch(e:any){
                    showToast({msg:e.response.data.error as string, type:'Bad',visible:true})
                }
            }
```

Nota. La imagen muestra el contenido de la función que maneja la operación de guardar la oferta de trabajo y el contenido de la función de aplicar la oferta del componente jobViewPage.

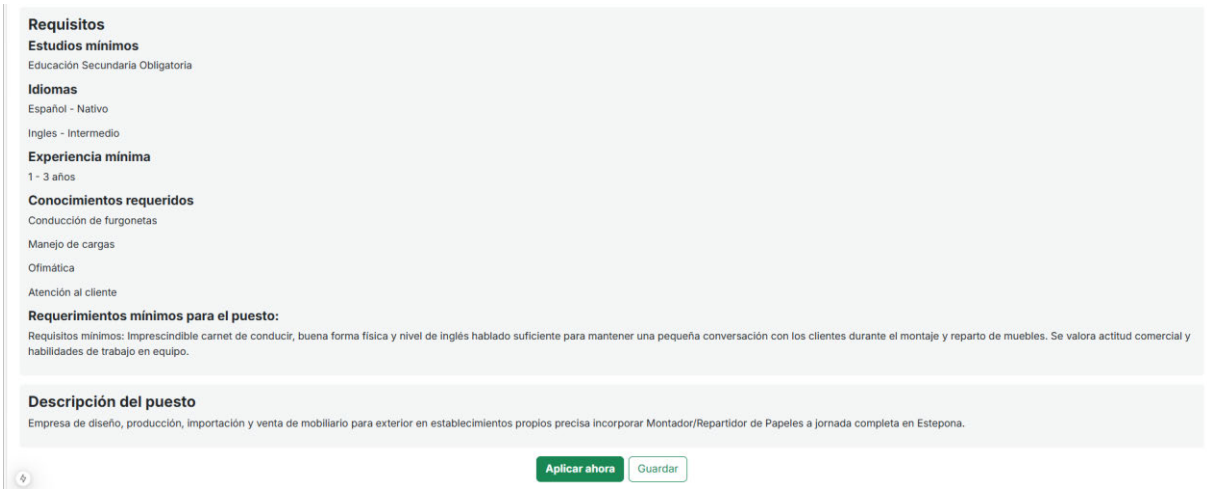
Figura 72

JobViewPage Parte 1

Nota. La imagen muestra la parte superior que se renderiza del componente JobViewPage. Incluye la sección para ofrecer opciones de guardar o aplicar la oferta, así como la demostración de las informaciones de las variables y los requisitos de discapacidad de la oferta.

Figura 73

JobViewPage Parte 2



Nota. La imagen muestra la parte inferior que se renderiza del componente JobViewPage. Incluye la sección que lista los requisitos y la descripción de la oferta, así como la sección para realizar opciones de guardar o aplicación la oferta.

companyViewPage

En esta página, se muestra la información acerca de una empresa. En la parte superior, se muestra la información del perfil de la empresa, cuales son: nombre, industria, escala, enlace de la página oficial y descripción de la empresa. El resto del espacio, el contenido que muestra depende del enlace elegido por el usuario, y son dos posibilidades: por un lado, muestra la descripción de la empresa. Y Por otro lado, se lista todas las ofertas de trabajo que tiene publicadas y con estado activo de la empresa, utilizando el componente JobListDisplay mencionado anteriormente.

El control de mostrar qué parte de la página está implementado con un estado (*activePage*). Si el *activePage* coincide con “Info”, se muestra la descripción de la empresa, y si coincide con “Jobs”, se muestra el listado de ofertas de trabajo de la empresa. El valor del *activePage* se actualiza al hacer clic en el enlace correspondiente, y su valor por defecto es “Info”.

Figura 74

Lógica interna del componente companyViewPage

```
{
    activePage==="Info"? (<div className="row" style={{border:"solid", borderWidth:"1px", marginLeft:"6px", marginRight:"6px", marginBottom:"1.5rem", padding:"1rem"}}>
    <h2>Descripón de la empresa</h2>
    <p style={{minHeight:"280px"}}>{company.description? company.description : "La empresa no ha añadido todavía su descripción."}</p>
</div> ):(<div>
    { <div className="container">
        {company.jobs.length > 0 ? (
            <JobListDisplay jobList={company.jobs} />
        ) : (
            <div className="row">
                <div className="col-sm-12" style={{fontSize:"1.4rem", color:"GrayText", fontWeight:"500"}}> No se ha encontrado ningún trabajo que cumpla con los filtro
            </div>
        )}
        {(company.jobs.length > 0)&&<div style={{ marginTop: '20px', textAlign: 'center' }}>

            <button className={`${styles.paginationButton}`}
            onClick={()=>{handlePrePagination(),handlePaginationJobs(currentPage-1)}}
            disabled={currentPage===1}
            >
            &laquo;
            </button>
            {displayPagesNumber()}
            <button className={`${styles.paginationButton}`}
            onClick={()=>{handleNextPagenation(),handlePaginationJobs(currentPage+1)}}
            disabled={currentPage===(totalPages)}
            >
            &raquo;
            </button>
        </div>}
</div>}
div>)}
```

Nota. La imagen muestra el parte de la implementación del componente companyViewPage para cambiar el contenido que muestra la parte inferior de la información de la empresa.

Figura 75

companyViewPage parte 1

Nota. La imagen muestra lo que renderiza el componente *jobViewPage* cuando el usuario elige la opción de Información para mostrar la descripción de la empresa.

Figura 76

jobViewPage Parte 2

Nota. La imagen muestra lo que renderiza el componente *jobViewPage* cuando el usuario elige la opción de Ofertas publicadas para mostrar el listado de las ofertas activas de la empresa.

ApplicationViewPage:

La página donde muestran las ofertas de trabajo relacionadas con el candidato en diferentes estados, que son: *Guardados*, *Solicitados*, *En curso* y *Cerrados*. Excepto el estado *Guardados*, los demás están relacionados con las ofertas de empleo que el candidato ha aplicado. Debido a que se almacenan en distintas formas, se han creado dos estados para guardar las ofertas, uno se dirige a las ofertas guardadas y otro para las ofertas aplicadas.

Las ofertas guardadas están almacenadas en el formato de array (*savedJob*) en el documento del candidato en la base de datos, y mientras las ofertas aplicadas, se mantienen en relación de 1: N con los candidatos en la forma de documentos referenciados, es decir, en cada documento de las ofertas de empleo, hay un array (*applicants*) para guardar el Id de los candidatos aplicados junto con el estado de la aplicación. De esta forma, podemos asegurar una base de datos escalable y facilitar el proceso de gestión de los candidatos de las ofertas de empleo.

Y los estados de la aplicación los gestionan por parte de las empresas. En el momento de aplicarla, se asigna con el estado de "*solicitado*" por defecto, y posteriormente, la empresa puede cambiar el estado de la aplicación a "*a comunicar*" o "*comunicado*" o cambiar directamente el estado de la oferta a cerrado.

Por el lado de los candidatos, las ofertas se muestran en forma de lista según el estado que se encuentran: las ofertas guardadas se muestran en el listado de *Guardados*, las aplicaciones que están en el estado de *solicitado* se muestran en el listado de *Solicitados*, y las aplicaciones con el estado de *a comunicar* o *comunicado* se muestran en el listado de *En curso*, y, tanto las ofertas guardadas como las aplicaciones que tienen el estado de *cerrado* se muestran en el listado de *Cerrados*.

Al sincronizar el estado de las ofertas por ambos lados (candidato y empresa), se puede proporcionar una mejor comprensión acerca del avance de las aplicaciones de los candidatos, y evitar que los usuarios pasen por una espera incierta por falta de información sobre el estado de sus solicitudes.

Figura 77

Lógica interna del componente ApplicationViewPage

```
useEffect(()=>{

    const fetchJob = async() =>{
        try{
            setLoading(true);
            const {data} = await axiosInstance.post(`/candidate-home/get-applied-jobs`,{activePage:activePage, page:1} ,{
                withCredentials:true
            })
            const savedJob = await axiosInstance.post(`/candidate-home/get-saved-jobs`,{ page:1},{
                withCredentials:true
            })
            setError("");
            setJobList(data.job);
            setSavedJobList(savedJob.data.job);
            if(data.totalPage>5){
                setDemonstratingPages(5);
                setPaginationLimit(5);
            }else{
                setDemonstratingPages(data.totalPage);
                setPaginationLimit(data.totalPage);
            }
            setCurrentPage(1);
            setTotalPages(data.totalPage);
        }catch(e){
            setError("Hubido error al cargar datos de las ofertas...");
            console.log("Error al cargar los trabjos... check it out:"+e)
        }finally{
            setLoading(false);
        }
    }
    fetchJob();

},[])
```

Nota. La imagen muestra el contenido del useEffect del componente ApplicationViewPage, donde puede verse claramente que las ofertas de trabajo guardadas de un candidato y las aplicaciones de un candidato se obtienen en distintas formas y se gestionan también en distintos estados.

Figura 78

ApplicationViewPage

Nota. La imagen se muestra la forma de demostrar un listado de ofertas de trabajo de un estado, se utiliza para dar un ejemplo al contenido que renderiza el componente ApplicationViewPage.

profilePage:

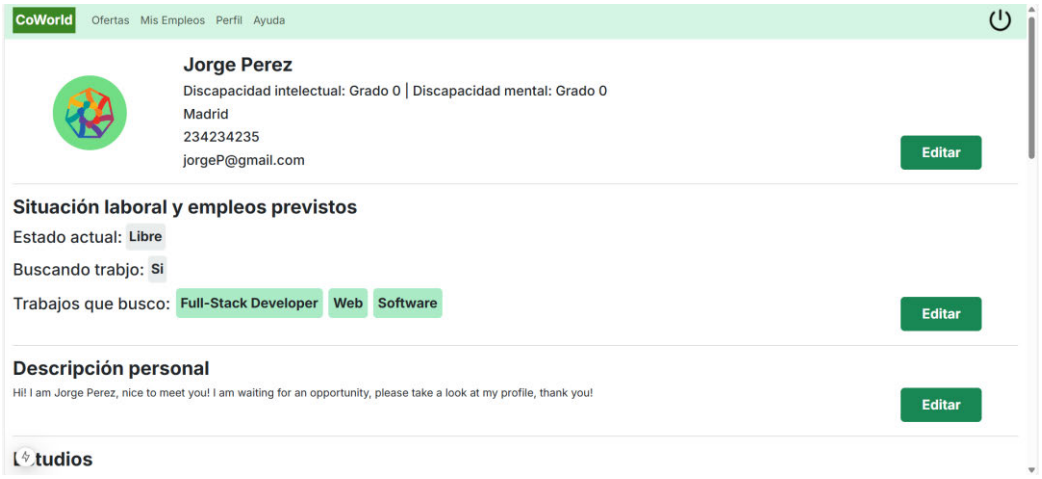
Como se ha mencionado previamente, el perfil es esencial para los candidatos, ya que cuando se aplica una oferta de trabajo, se envía su perfil en formato de solo lectura a la empresa que ha publicado la oferta. De esta forma, se usa como un currículum digital para que las empresas puedan conocer mejor a los candidatos y realizar el proceso de selección.

Esta página es donde los candidatos pueden gestionar el contenido de su perfil. La información está dividida en secciones según su contenido, que son:

- **Información personal:** donde puede editar el nombre, la situación de discapacidad, la ciudad habitual, el teléfono y la foto de perfil del candidato.
- **Situación laboral y empleos previstos:** donde puede editar la situación laboral que encuentra actualmente del candidato (Libre, Estudiando, o Trabajando), indicar si está buscando trabajo, así como los tipos de empleo que está buscando.
- **Descripción personal:** donde puede editar una breve descripción personal.
- **Estudios**: donde puede añadir los estudios realizados o en la que actualmente está realizando el candidato.
- **Experiencia Laboral:** donde puede añadir las experiencias laborales obtenidas o en la que actualmente se encuentra el candidato.
- **Habilidades:** donde puede editar las habilidades del candidato.
- **Licencias y certificaciones:** donde puede adjuntar las licencias y certificaciones obtenidas por el candidato.
- **Idiomas:** donde puede editar las habilidades de idioma que poseen, así como niveles que tiene en ellos.

Figura 79

ProfilePage



Nota. En la imagen se muestra un parte del contenido que renderiza por el componente ProfilePage.

Para editar la información de cada sección, se han implementado componentes para cada una de ellas con el fin de facilitar el proceso de gestión del perfil. Entre ellos, destacan los siguientes:

TagInput:

El componente tiene formato de entrada de texto, y cuando se introduce un texto, se muestra como un tag por debajo de la entrada de texto, que utiliza para los candidatos editar sus tipos de puestos deseados y habilidades.

Existe un conjunto de restricciones sobre el contenido que se introduce: no se puede superar una cierta cantidad de tags (se puede personalizar el límite) y el contenido del tag no puede ser vacío ni repetido con otros tags.

Figura 80

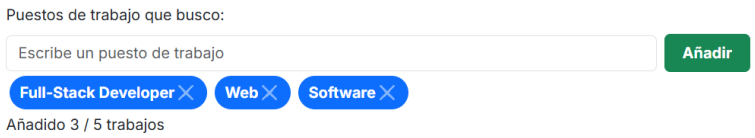
Lógica interna del componente TagInput

```
const handleAddJob = () => {
    if(job===""){
        setError("No se puede añadir contenido vacio!")
        return
    }
    if(tag.includes(job.trim())){
        setError("Contenido repetido!")
        return
    }
    if(tag.length < maxTag && !tag.includes(job.trim())){
        if(job.trim()===""){
            setError("No se puede añadir contenido vacio!")
            setJob("");
            return
        }
        setTag([...tag, job.trim()]);
        setJob("");
        setError("")
    }
}

const handleRemoveJob = (index:number) => {
    setTag(tag.filter((_, i) => i !== index));
}
```

Nota. La imagen muestra las funciones que controlan los tags del componente TagInput.

Figura 81

TagInput



Nota. La imagen muestra el componente TagInput renderizado en el navegador.

AskAgainModal:

El componente está diseñado para que cuando el candidato se intenta de eliminar una información, aparece un modal para confirmar si desea continuar con la operación, se usa para evitar minimizar las posibilidades de eliminar información por error.

Figura 82

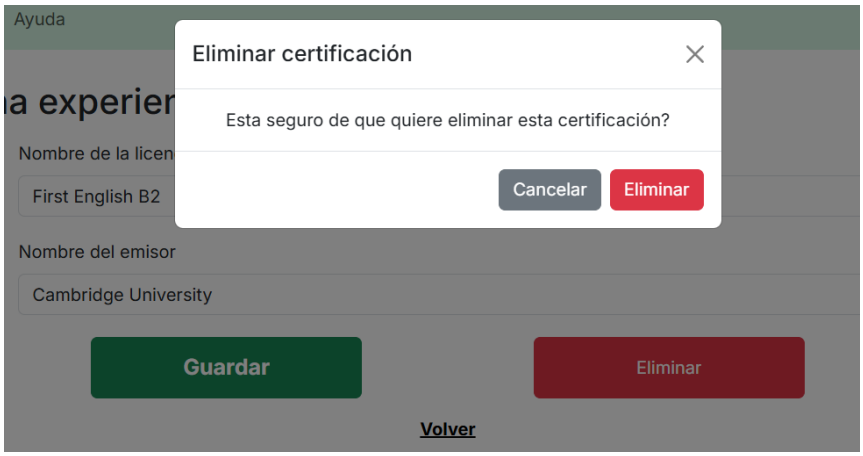
Implementación del componente AskAgainModal

```
<div className='text-center'>
    <div className="row" >
    <div className="col"/>
        <div className="col-md-4 col-sm-12">
        <Form.SubmitButton text="Guardar" loading={isLoading}/>
        </div>
        <div className="col-md-4 col-sm-12">
        <button type="button" className="col-6 btn btn-danger" data-bs-toggle="modal" data-bs-target="#askAgainModal"
        Eliminar
        </button>
        </div>
    <div className="col"/>
    </div>
        <div className="modal fade" id="askAgainModal" tabIndex={-1} aria-labelledby="askAgainModalLabel" >
        <div className="modal-dialog">
            <div className="modal-content">
            <div className="modal-header">
                <h5 className="modal-title" id="askAgainModalLabel">Eliminar certificación</h5>
                <button type="button" className="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
            </div>
            <div className="modal-body">
                Esta seguro de que quiere eliminar esta certificación?
            </div>
            <div className="modal-footer">
                <button type="button" className="btn btn-secondary" data-bs-dismiss="modal">Cancelar</button>
                <button type="button" className="btn btn-danger" data-bs-dismiss="modal" onClick={deleteCertification}
            </div>
            </div>
        </div>
        </div>
    <Form.Links href="/home/profile" text="" linkText='Volver'/>
</div>
```

Nota. La imagen muestra la implementación del componente AskAgainModal.

Figura 83

AskAgainModal

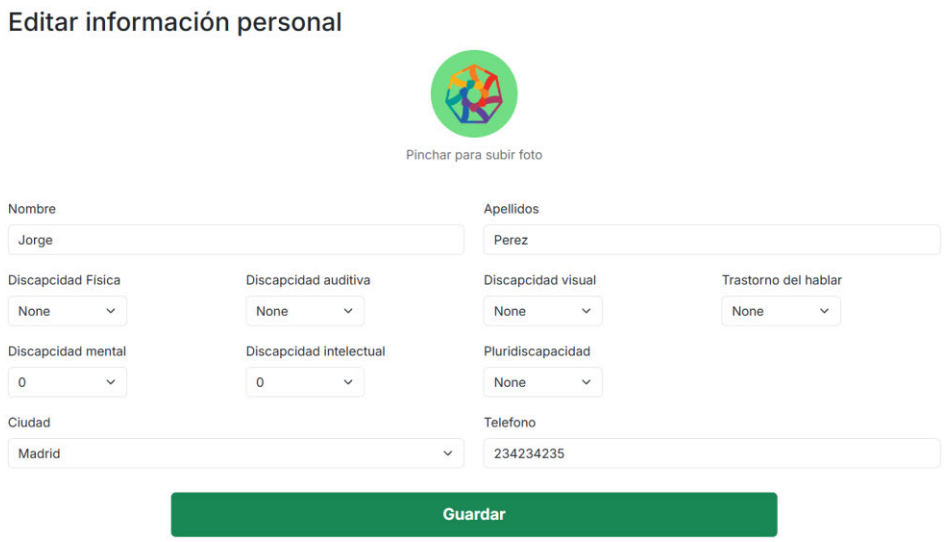


Nota. La imagen muestra un ejemplo del renderizado del componente AskAgainModal.

La mayoría de las secciones del perfil utilizan formularios para editar la información, lo que garantiza una mayor reusabilidad y facilita la gestión de datos para enviados al back-end para realizar operaciones. Dado que el detalle del funcionamiento del formulario ya ha sido explicado anteriormente, en esta sección se proporciona un ejemplo de su renderizado con el fin de mostrar el formato general del formulario. (véase la figura 84)

Figura 84

Renderizado del formulario



Nota. La imagen proporciona un ejemplo de renderizado del formulario.

Hasta este punto, se ha explicado todos los componentes y páginas esenciales destinados a los usuarios con el rol de candidato. A continuación, se presentan los correspondientes a las cuentas empresariales, entre ellos, destacan los siguientes:

CompanyHomeLayout:

El layout de las páginas protegidas destinadas a las cuentas empresariales y mantiene la misma estructura que el *HomeLayout* para los candidatos que se ha explicado anteriormente. Los componentes comunes en este layout son, en primer lugar, el componente *AccessChecker*, que se utiliza para validar el accessToken del usuario actual cuando navega entre páginas protegidas. Y en segundo lugar, se encuentra el componente *CompanyProvider*, cuyo detalle de este está en la explicación de la carpeta *context*. Este contexto comparte la información de la empresa, las ofertas de trabajo que tiene publicadas y el rol de la cuenta actual con sus componentes hijos, que son información necesaria para el correcto funcionamiento de la aplicación. Por último, el componente *NavbarCompany*, de la misma forma que las páginas protegidas de los candidatos, todas disponen un navbar, y en este caso, es el navbar exclusivo para las cuentas empresariales. A diferencia del *NavbarCandidate*, a las cuentas operadoras, se oculta la opción de *Perfil* y *Operador*, que son funcionalidades orientadas a la cuenta empresa. Además, para conocer el rol de la cuenta actual, el *NavbarCompany* se lee esta información desde el contexto *CompanyProvider.*

El navbar ofrece acceso rápido a distintas páginas con el fin de mejorar la experiencia del usuario. Entre ellos, los accesos rápidos son:

- **El logo “CoWorld”:** acceso rápido a la página de inicio (/company-home) de las cuentas empresariales, ya que las aplicaciones en común tienen dicha funcionalidad.
- **Ofertas:** acceso rápido a la página de inicio (/company-home) de las cuentas empresariales.
- **Candidatos:** acceso rápido a la página donde puede consultar información de los candidatos inscritos en distintas ofertas de la empresa(/company-home/candidates).
- **Operadores:** acceso rápido a la página de gestión de los operadores de la empresa(/company-home/operators), oculto a los operadores.
- **Perfil**: acceso rápido a la página de perfil de la empresa (/company-home /profile), oculto a los operadores.
- **Ayuda**: acceso rápido a la página de ayuda a las cuentas empresariales (/company-home /help).
- **El logo de “Cerrar sesión”**: enlace en formato de imagen de un botón para cerrar la sesión de la cuenta actual.

Figura 85

CompanyHomeLayout

```
export default function CompanyHomeLayout({ children }: HomeLayoutProps) {
  return (
    <AccessChecker>
      <CompanyProvider>
        <NavbarCompany/>
        {children}
      </CompanyProvider>
    </AccessChecker>
  )
}
```

Nota. La imagen muestra el contenido principal del componente *CompanyHomeLayout.*

Figura 86

NavbarCompany

Nota. La imagen muestra el renderizado del componente *NavbarCompany*.

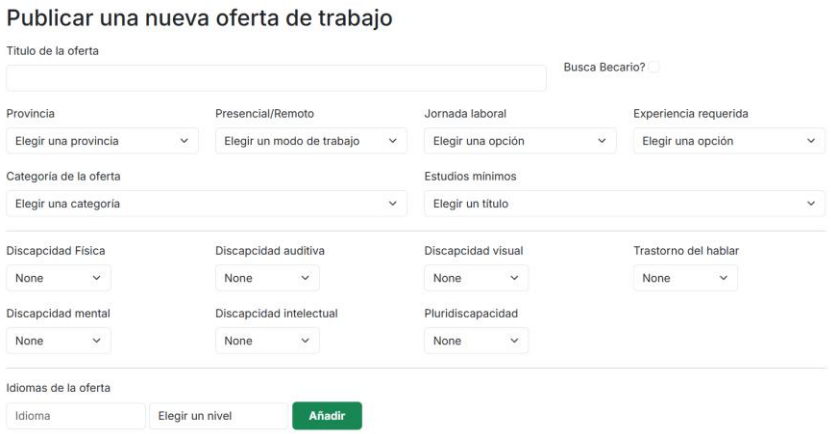
CompanyJobListPage:

La página inicial de las cuentas empresariales, donde se gestionan las ofertas de trabajo publicadas. Al cargar la página, se muestran las ofertas de trabajo

de la empresa separadas por dos estados: *En curso* o *Cerrado*, y cada una de las ofertas mostradas dispone de un botón de modificar para gestionar su información. Además, en la página se ofrece también la opción de publicar una nueva oferta rellenando el formulario de información de la oferta (véase la figura 87).

Figura 87

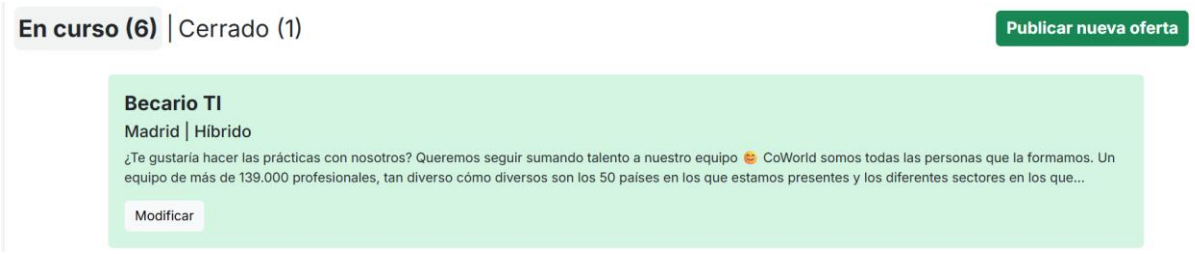
Formulario de publicar nueva oferta



Nota. La imagen muestra un parte del formulario de publicar nueva oferta.

Figura 88

Listado de ofertas publicadas



Nota. La imagen muestra la forma de listar las ofertas de trabajo publicadas por la empresa en distintos estados (En curso o Cerrado).

ViewCandidatePage:

La página de acceso de la funcionalidad de gestión de inscripciones de las ofertas de trabajo de la empresa. Al cargar la página, se muestran todas las ofertas leídas desde el contexto de empresa, y el usuario puede elegir entre ellas para empezar el proceso de gestionar las inscripciones de esta oferta.

Figura 89

Lógica interna del componente ViewCandidatePage

```
useEffect(()=>{
    if(company?.jobs){
        setJoblist(company.jobs.slice(0,5));
        setTotalPages(Math.ceil(company?.jobs.length/5));
        if(Math.ceil(company?.jobs.length/5)>5){
            setDemonstratingPages(5);
            setPaginationLimit(5);
        }else{
            setDemonstratingPages(Math.ceil(company?.jobs.length/5));
            setPaginationLimit(Math.ceil(company?.jobs.length/5));
        }
        setCurrentPage(1);
    }
},[company?.jobs])
```

Nota. La imagen muestra el contenido del useEffect del componente ViewCandidatePage, que se encarga de leer las ofertas de empleo desde el contexto de empresa y monitorizarlas para actualizarlas cuando hay cambios.

ViewOneJobCandidatesPage:

Es la página que se muestra cuando se selecciona una oferta de trabajo para empezar el proceso de gestión de inscripciones de la misma. Al cargar la página, se muestra todos los candidatos de la oferta en lista y separados por distintos estados. Como se ha mencionado anteriormente, existen tres estados para una aplicación de la oferta: inicialmente todos tienen el estado de "*solicitado*", que se muestra en la lista de *Pendiente*. La empresa o el operador de la empresa pueden cambiar el estado de la aplicación según la decisión tomada sobre el candidato, y los estados son: *a comunicar* y *comunicado*. Estos tres estados funcionan como un pipeline, ya que una vez cambiado el estado de la aplicación, ya no es posible de retroceder al estado anterior, dada que los cambios de estado también se reflejan en el lado de los candidatos (en *ApplicationViewPage*). Y el orden de los estados es: pendiente → a comunicar → comunicado, además, es posible pasar desde pendiente a comunicado.

Cada vez que se cambia de una lista a otra, se envía de una solicitud HTTP con el estado elegido para obtener las aplicaciones de la oferta que tienen el mismo estado. Los candidatos se muestran con un formato de carta, junto con unos de sus datos personales, para que la empresa u operador pueda tener una vista previa de los candidatos, y facilitar así el proceso de gestión. Al elegir un candidato mostrado, se pasa al siguiente paso del proceso de gestión las inscripciones.

Figura 90

Lógica interna del componente ViewOneJobCandidatesPage

```
export default function ViewOneJobCandidatesPage(){
    const fetchCandiates= async(status:string,page:number)=>{
        try{
            setLoading(true);
            const {data} = await axiosInstance.post(`/company-candidates/get-previews`,{id:params.id, status:status, page:page},{
                withCredentials:true
            })
            setCandidateList(data.candidates);
            setTotalPages(data.totalPage);
            setCurrentPage(page);
            if(data.totalPage>5){
                setDemonstratingPages(5);
                setPaginationLimit(5);
            }else{
                setDemonstratingPages(data.totalPage);
                setPaginationLimit(data.totalPage);
            }
            setError("")
            setLoading(false)
        }catch(e){
            setError("Error al cargar candidatos de la oferta");
            setLoading(false)
        }
    }
    useEffect(()=>{
        fetchCandiates("solicitado",1);
    },[params.id])

    useEffect(()=>{
        if(activePage==="pending"){
            fetchCandiates("solicitado",1);
        }else if(activePage==="aComunicar"){
            fetchCandiates("a comunicar",1);
        }
        else if(activePage==="Comunicado"){
            fetchCandiates("comunicado",1);
        }
    },[activePage])
```

Nota. La imagen muestra el contenido de las funciones relacionadas con el manejo de cambios entre las listas de estados.

Figura 91

ViewOneJobCandidatesPage

Nota. La imagen muestra el renderizado del componente *ViewOneJobCandidatesPage*.

candiateDetailPage:

La página que se muestra en el último paso del proceso de gestión de una inscripción de la oferta. En ella se muestra el perfil del candidato en modo lectura, y tanto en la parte superior como en la inferior de la página, disponen de botones para cambiar el estado de la inscripción según su estado actual que se encuentra. Si pasa a estado "a *comunicar*", la página se muestra el botón para cambiarlo a

estado “comunicado”, y si pasa a estado de “*comunicado”*, se muestra un mensaje de aviso en el que se indica que el candidato ya ha sido comunicado.

Al pulsar sobre los botones de cambio de estado, se envía inmediatamente una solicitud HTTP con el estado que se desea actualizar al back-end para que se actualice en la base de datos.

Con esta funcionalidad, las empresas pueden gestionar las inscripciones de manera más eficiente, y, como el cambio de estado de inscripción se sincroniza también con los candidatos correspondientes, quienes pueden conocer el avance de sus aplicaciones realizadas.

Figura 92

Lógica interna del componente candiateDetailPage

```
const handleUpdateStatus= async(status:string)=>{
    setIsLoading(true);
    try{
        const resul = await axiosInstance.post(`/company-candidates/change-status`,{candidate:params.candidate,id:params.id,status:status},{
                withCredentials:true
        })
        showToast({msg:resul.data.sucess, type:'Good',visible:true})
        router.push(`/company-home/candidates/${params.id}/${params.candidate}?status=${status}`);
    }catch(e){
        showToast({msg:e.response.data.error as string, type:"Bad",visible:true})
    }finally{
        setIsLoading(false);
    }

}
```

Nota. La imagen muestra el contenido de la función que realiza la petición HTTP al back-end para actualizar el estado de la inscripción.

Figura 93

candiateDetailPage

Nota. La imagen muestra el parte del renderizado del componente candiateDetailPage.

ViewOperatorsPage:

La página donde se gestiona los operadores de la empresa y está oculta a ellos. Al cargar la página se muestra todos los operadores de la empresa, y de forma similar a la página inicial, cada carta de operador dispone de un botón para

gestionar su información. Y en la parte superior de la página, dispone de una opción para añadir un nuevo operador rellenando un formulario. Al entregar el formulario, se procesa de enviar un correo de aviso de activación de la cuenta del operador al correo electrónico proporcionado. En el listado de operadores se puede comprobar si el operador ha realizado la activación.

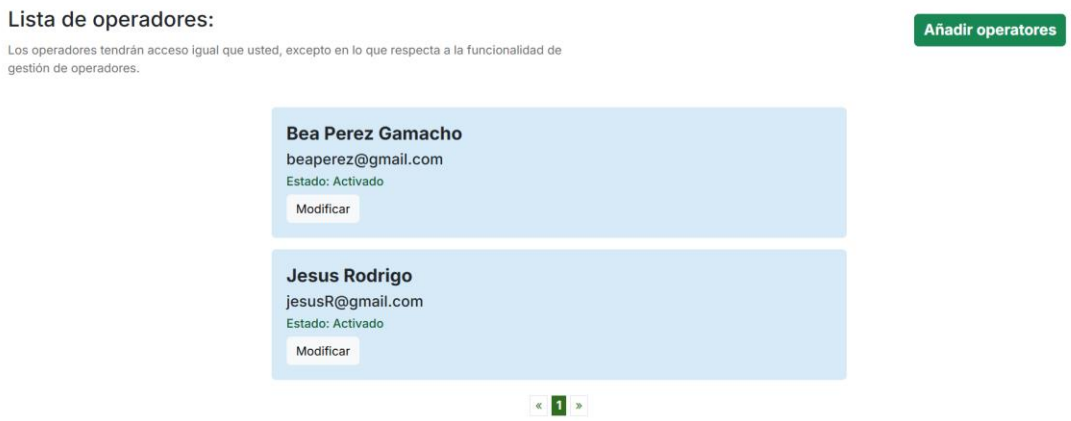
Figura 94

Formulario de añadir nuevo operador

Nota. La imagen muestra el contenido del formulario para añadir un nuevo operador a la empresa. ´

Figura 95

ViewOperatorsPage



Nota. La imagen muestra el renderizado del componente *ViewOperatorsPage*.

companyProfilePage:

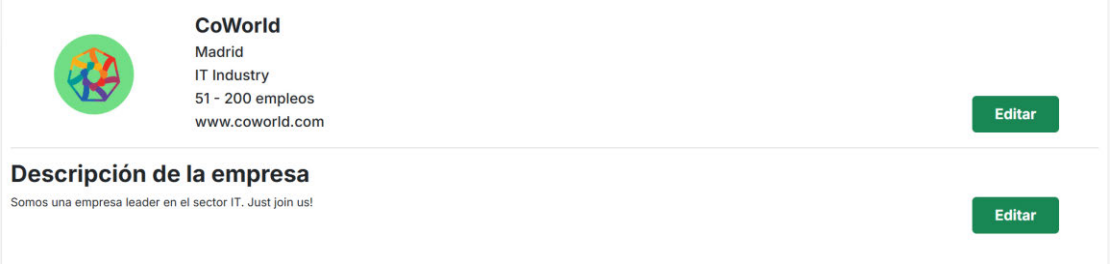
La página donde puede gestionar el perfil de la empresa y está oculta a los operadores, ya que es una funcionalidad exclusiva para las empresas. Como el

perfil de los candidatos, el perfil de las empresas también está divididos por secciones, pero con menos contenidos, cuales son:

- **Información de la empresa:** Donde puede editar el nombre, la ciudad de la sede principal, la escala, la industria y el enlace de la página oficial de la empresa.
- **Descripción de la empresa:** Donde puede editar una breve descripción de la empresa.

Figura 96

companyProfilePage



Nota. La imagen muestra el renderizado del componente companyProfilePage.

4.3. Lado servidor

En esta sección se aborda la explicación detallada del lado servidor de la aplicación. El lado servidor está implementado en el lenguaje TypeScript utilizando las funcionalidades proporcionadas por el Next.js. La función del lado servidor consiste en interactuar con la base de datos de la aplicación a través de la librería Mongoose tras de recibir las peticiones HTTP enviadas desde el lado cliente, lo que permite una arquitectura coherente y eficiente de la aplicación.

En Next.js, todas las rutas API se guardan en la carpeta *app/api*, y la implementación de cada ruta se ubica en el fichero enrutable *route.ts* en el último nivel de cada ruta.

A continuación, se explica la lógica central de cada endpoint:

Para comenzar, se han organizado todos los endpoints relacionados con el sistema de autenticación dentro de la carpeta *auth*, que destacan los siguientes:

/api/auth/login:

Es el endpoint que se envía cuando el usuario introduce su correo electrónico y contraseña para iniciar sesión como un candidato. El flujo de trabajo de este endpoint consiste, en primer lugar, en realizar un conjunto de validaciones.

Cuando se falla alguna de ellas, se envía inmediatamente la respuesta con el estado 400 junto con el mensaje de error para informar del fallo. En el caso de superan todas las validaciones, se genera un accessToken y un refreshToken para la sesión actual y cada uno se guarda en una cookie correspondiente creada en el navegador del usuario y se le da acceso a la aplicación.

Se ha elegido cookie para guardar los tokens, porque es una forma común de gestionar las sesiones y es segura (al tener *httpOnly* configurado), además es conveniente a la hora de intentar recuperarlo para realizar el proceso de autenticación.

Las validaciones se realizan en el siguiente orden: Primero se comprueba si están rellenados tanto el correo electrónico y la contraseña. A continuación, se comprueba si el correo electrónico tiene un formato correcto, y, después de eso, se intenta buscar en la colección *users* el documento del usuario que tiene registrado con ese correo, y, por último, se utiliza la biblioteca *bcrypt* para comparar la contraseña proporcionada con la que tiene guardada en la base de datos.

Figura 97

Configuración de tokens de sesión

```
// Generar tokens para la sesion
const accsessToken = jwt.sign({data:rest},process.env.ACCESS_TOKEN_SECRET,{expiresIn:'15m'})
const refreshToken = jwt.sign({data:rest},process.env.REFRESH_TOKEN_SECRET,{expiresIn:'7d'})

const response = NextResponse.json({
    sucess:message.sucess.UserLogged,
    findUser:rest
},{status:200})

// generar tokens para la sesion
response.cookies.set("accessTokenCookie",accsessToken,{
    sameSite:"lax",
    secure:process.env.NODE_ENV==="production",
    maxAge:7200, //2H 2x60x60
    httpOnly:true,
    path:"/"
});

response.cookies.set("refreshTokenCookie",refreshToken,{
    sameSite:"strict",
    secure:process.env.NODE_ENV==="production",
    maxAge:604800, //7dias = 7x24x60x60
    httpOnly:true,
    path:"/"
});
return response;
```

Nota. En la imagen se muestra la configuración de accessToken, refreshToken y las cookies correspondientes para una sesión de la aplicación actual.

/api/auth/register:

El endpoint que se envía cuando se registra una cuenta con el rol de candidato. Tiene un funcionamiento similar al del endpoint de inicio de

sesión(/api/auth/login): ambos se realizan primero un conjunto de validaciones, y, a continuación, en este caso se cifra la contraseña y crea un nuevo documento en la colección users con los datos proporcionados.

Además, al mismo tiempo, se crea un documento en la colección candidateprofiles con las variables vacías y se estable una relación de 1:1 con el documento de la colección users. Finalmente, se configuran los tokens para la sesión actual en las cookies correspondientes y se le da acceso a la aplicación.

Las validaciones de registro se realizan en siguiente orden: primero se comprueba si tienen todos los campos necesarios rellenados, y a seguidas, comprobar si el correo electrónico proporcionado ya existe en la colección users, ya que un email solo puede estar asociado con una cuenta. Y si el correo es nuevo, se revisa el formato de los demás datos: la contraseña debe tener el formato restringido por la aplicación por motivo de seguridad y tanto el nombre como los apellidos deben contener únicamente letras.

Figura 98

Creación del documento candidateProfile

```
// creo el perfil ya tambien para el usuario nuevo
console.log("El usuario creado :"+newUser._id)
const profile: ICandidateProfileDocument = new CandidateProfile({
    user_id: newUser._id,
    phone: "",
    city: "",
    photo: "",
    disabilities: [
        {type:FISICA, degree:-1},
        {type:AUDITIVA,degree:-1},
        {type:VISUAL,degree:-1},
        {type:MENTAL,degree:-1},
        {type:HABLAR,degree:-1},
        {type:INTELECTUAL,degree:-1},
        {type:PLURIDISCAPACIDAD,degree:-1}],
    state: "Libre",
    huntingJob: false,
    desiredJob:[],
    description: "",
    studies:[],
    workExperience:[],
    skills:[],
    languages:[],
    certifications:[]
});
await profile.save();
```

Nota. La imagen muestra la creación de un documento en la colección candiateprofiles durante el proceso de registro desde el endpoint /api/auth/register.

/api/auth/company-login

El endpoint de inicio de sesión para las cuentas empresariales. Se mantiene el mismo flujo y validaciones que el endpoint de inicio de sesión de los candidatos (/api/auth/login), la única diferencia es que, en el paso de buscar el documento de la cuenta, en este caso se busca tanto en la colección *companies* como en la

colección *operators* para confirmar la existencia de la cuenta y determinar el rol de la cuenta. Además, en el caso de que se trate de un operador, se utiliza el documento de su empresa para iniciar la sesión, ya que el operador se realizar operaciones en la cuenta de su empresa con unas funcionalidades limitadas.

Figura 99

Comprobación de existencia de una cuenta empresarial

```
// comprobar la existencia del correo:
const findCompany = await Company.findOne({email:email})
const findOperator = await Operator.findOne({email:email})

if(!findCompany && !findOperator){
    return NextResponse.json({
        error:message.error.notFoundEmailCompany
    }, {status:400})
}

if(findOperator&& findOperator.changedPassword===false){
    return NextResponse.json({
        error:message.error.operatorLoadError
    }, {status:400})
}
// comprobar si la contraseña es correcta
const comparePwd = findCompany? await bcrypt.compare(password, findCompany.password) : await bcrypt.compare(password, findOperator.password)

if(!comparePwd){
    return NextResponse.json({
        error:message.error.pwdIncorrect
    }, {status:400})
}

var operatorCompany = null;
if(findOperator){
    operatorCompany = await Company.findOne({_id:findOperator.company_id});
}
const {password:UserPass, ...rest} = operatorCompany? operatorCompany._doc : findCompany._doc;
```

Nota. La imagen muestra el proceso de identificar el rol de una cuenta empresarial.

/api/auth/company-register

El endpoint de registro de cuenta empresarial, funciona de la misma forma que el endpoint de registro de cuenta de candidato, solo que en este caso se proporcionan datos diferentes. Además, el perfil se crea en la colección *companyprofiles* y se establece una relación de 1:1 con el documento creado en la colección *companies*.

Figura 100

Creación de cuenta empresa

```
const newCompany:ICompanyDocument = new Company({
    firstname:firstname,
    lastname:lastname,
    email:email,
    password:pwdEncrypted,
    role:role,
    companyName: companyName,
    cif: cif
});
await newCompany.save();

const profile: ICompanyProfileDocument = new CompanyProfile({
        company_id: newCompany._id,
        industry: "",
        city: "",
        scale: "",
        url: "",
        logo: "",
        description: ""
})

await profile.save()
```

Nota. La imagen muestra la creación del documento company y así como el documento de su perfil.

/api/auth/forget-pwd

El endpoint que se envía cuando el usuario solicita el restablecimiento de la contraseña. En primer lugar, se comprueba el formato del email proporcionado y, a continuación, se comprueba la existencia de una cuenta registrada con dicho email en las tres colecciones: *users*, *companies* y *operators*. En el caso de operador, se comprueba además si la cuenta esta activada, en caso negativo, no se permite el restablecimiento de la contraseña. Y finalmente, una vez comprobada la existencia de la cuenta, se envía un correo junto con el enlace de acceso al proceso de restablecimiento a la dirección de correo proporcionado. El enlace está configurado con un token de validez de una hora por motivos de seguridad.

Figura 101

Envío del correo de restablecimiento de la contraseña

```
const tokenForChangePwd = {
    email:targetEntity.email,
    userId: targetEntity._id,
}

// dejamos que se invalida este proceso en una hora 60*60*1000
const token = jwt.sign({data:tokenForChangePwd},process.env.RESET_PWD_TOKEN_SECRETE,{expiresIn:'1h'});

const url = `http://localhost:3000/reset-password?token=${token}`;

await resend.emails.send({
    from: "onboarding@resend.dev",
    to: [redacted]
    subject: "Reset your password",
    react: EmailTemplate({url:url,firstname:targetEntity.firstname}),
});
```

Nota. La imagen muestra el proceso de envío del correo de aviso del restablecimiento de la contraseña.

/api/auth/reset-pwd

El endpoint que se envía cuando se realiza el proceso de restablecimiento de contraseña. Antes de todo, se comprueba la validez del token, si ha expirado, el usuario es necesario volver a solicitar el restablecimiento de la contraseña. Y posteriormente, se realiza las validaciones para la nueva contraseña: primero se comprueba si la nueva contraseña cumple con la restricción del formato de contraseña, y a continuación, se comprueba si coincide con la contraseña de confirmación. Una vez superadas las validaciones, se cifra la nueva contraseña y se reemplaza la antigua en el documento del usuario.

Figura 102

Reemplazo de la contraseña antigua

```
const newPwd = await bcrypt.hash(pwd,10)

var user = await User.findOneAndUpdate({_id:data.userId},{password:newPwd},{new:true});
if(!user){
    user = await Company.findOneAndUpdate({_id:data.userId},{password:newPwd},{new:true});
}
if(!user){
    user = await Operator.findOneAndUpdate({_id:data.userId},{password:newPwd},{new:true});
}
if(!user){
    return NextResponse.json({
        error:message.error.pwdCantChange
    },{status:400})
}
```

Nota. La imagen muestra el proceso de sustituir la contraseña antigua por la contraseña nueva en el documento del usuario.

/api/auth/reset-pwd-operator

El endpoint que se envía cuando se activa una cuenta de rol de operador. Tiene un flujo similar al del restablecimiento de la contraseña(/api/auth/reset-pwd). La única diferencia es que, en este caso, se añade la validación de la contraseña temporal. Esto significa que en el correo de aviso de activación de la cuenta de operador, se adjunta una contraseña temporal establecida aleatoriamente por el sistema para la cuenta, por motivos de seguridad, y el operador debe introducir esta contraseña para validar su identidad.

Figura 103

Validación de la contraseña temporal

```
const isValidPwd = await bcrypt.compare(tempPwd,operator.password);
if(!isValidPwd){
    return NextResponse.json({
        error:message.error.pwdIncorrect
    },{status:400})
}
```

Nota. La imagen muestra el proceso de comparación de la contraseña temporal durante la activación de la cuenta de operador.

/api/auth/refresh-token

El endopoint que se llama el axiosInterceptor cuando captura una respuesta con el estado 401 para renovar el accessToken del usuario. Primero se valida el refreshToken, si también está expirado, el usuario debe volver a iniciar sesión. En el caso contrario, se genera un nuevo accessToken y se guarda en su cookie.

Figura 104

Renovación del accessToken

```
const isVliadToken = jwt.verify(refreshToken,process.env.REFRESH_TOKEN_SECRET);
if(!isVliadToken){
    return NextResponse.json({
        error: message.error.invalidToken
    },{status:401})
}
// @ts-ignore
const {data} = isVliadToken;
// Si el refreshToken es valido, generamos un nuevo accessToken con la data del refreshToken
const newAccsessToken = jwt.sign({data},process.env.ACCESS_TOKEN_SECRET,{expiresIn:'15m'})

const response = NextResponse.json({
    newAccsessToken: newAccsessToken
},{status:200})

response.cookies.set("accessTokenCookie",newAccsessToken,{
    sameSite:"lax",
    secure:process.env.NODE_ENV==="production",
    maxAge:7200, //2H 2x60x60
    httpOnly:true,
    path: "/"
});
```

Nota. La imagen muestra el proceso de renovación de accessToken del usuario usando su refreshToken.

/api/auth/check-access-token

El endpoint que se envía para validar el accessToken del usuario cuando navegan entre las páginas protegidas. Se lee el accessToken del usuario desde la cookie donde está almacenado, y se comprueba su validez con la librería jwt. En caso de que haya expirado, el axiosInterceptor captura la respuesta con el estado 401 e invoca el endpoint /api/auth/refresh-token para renovar el accessToken.

Figura 105

Comprobación del accessToken

```
try{
    const data = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
    return NextResponse.json({
        authorized:true, sucess:message.sucess.UserAuthorized
    },{status:200})

}catch(e){
    return NextResponse.json({
        error:message.error.invalidToken, e
    },{status:401})

}
```

Nota. La imagen muestra el proceso de validación del accessToken del usuario.

A continuación, empiezan las explicaciones de las rutas API relacionadas con el usuario del rol operador, que destacan los siguientes endpoints:

/api/candidate-home/get-jobs

Este endpoint permite obtener las ofertas de trabajo filtradas, y los filtros se envían a través del parámetro de la URL. Cuando el usuario no ha seleccionado algún filtro, se manda con valor vacío. En el caso de que todos los filtros se reciban con vacíos, el endpoint devuelve todas las ofertas de trabajo disponibles ordenadas por el algoritmo de recomendación.

Inicialmente, se transforma el parámetro del enlace al objeto de tipo *JobFilters* para facilitar su uso posterior. En los filtros de selección múltiple (ciudad, modo de trabajo, jornada de trabajo, categoría y educación mínima), sus valores se encuentran como una cadena de texto separada por comas. Con esta característica, se pueden transformar en un array de cadenas de texto, excepto para el filtro de discapacidad, que se transforma en un array de objetos JSON en los que se informa del tipo de discapacidad y el nivel asociado.

Construcción del query MongoDB:

Una vez tienen los filtros transformados, se procede a construir el query de filtración para MongoDB. A los filtros transformados en array de cadenas de texto, se les aplica el operador *$in* para cada uno de ellos. Este operador significa que, si el valor de la variable de la oferta de trabajo que se está evaluando coincide con algún valor del array, se devuelve true. Y a los filtros de selección única (experiencia laboral y busca becario), se aplica el operador *$eq*, que requiere que el valor de la variable de la oferta que se está evaluando coincida exactamente con el valor del filtro para que devuelva a true.

El filtro de discapacidad tiene una logia de negocio más compleja. Como se ha mencionado anteriormente, en el momento de publicación de la oferta de trabajo, la empresa debe elegir el requisito de aceptación de nivel para todos los tipos de discapacidad. El concepto de nivel se interpreta de manera distinta entre las empresas y los candidatos. A las empresas, la asignación del nivel -1 a un tipo de discapacidad significa que no hay restricciones para esa discapacidad. Asimismo, para los con un nivel marcado mayor que -1, se entiende que la oferta tiene una restricción de igual o menor nivel marcado para el tipo de discapacidad. Y para los candidatos, el nivel -1 indica que no tiene el tipo de discapacidad, por el contrario, cualquier otro número mayor que -1, señala que el usuario tiene dicho nivel en el tipo de discapacidad.

Por consiguiente, primero se busca en el filtro de discapacidad, aquellos tipos de discapacidad con nivel mayor que -1, porque son los que hay que comparar con las restricciones de la oferta. Para ellos, se devuelve el true cuando la oferta tiene el nivel -1 (sin restricción) o un nivel superior al nivel marcado por el usuario (el usuario tiene un nivel igual o menor a la restricción) en el tipo de discapacidad.

Algoritmo de recomendación:

El algoritmo se ha diseñado con la estrategia de puntuación y es una fase del pipeline MongoDB. Se comparan entre los datos de la oferta con los datos del perfil de usuario, que son: *provincia habitual, situación de discapacidad, trabajos deseados* y *habilidades*. Se define un campo para registrar el punto ganado de cada variable comparada.

La lógica de la puntuación se basa en siguientes criterios:

- **Provincia:** si la provincia de la oferta coincide con la provincia habitual del candidato, se suma un punto para este variable.
- **Discapacidad:** Comparando con el resto, es más complicado debido a su estructura de datos y lógica de negocio. El algoritmo recorre cada objeto del array de discapacidades de la oferta de trabajo. Para cada tipo de discapacidad encontrado, se busca una coincidencia en el array de discapacidades del usuario y comprueba si el nivel indicado en la oferta tiene es igual a -1 o mayor que el nivel del usuario para el tipo de discapacidad, en caso confirmativo, se suma un punto para este variable.
- **Trabajos deseados y habilidades:** a cada uno de ellos se comparar con varias variables de la oferta (*titulo, descripción, requerimientos de empresa* y *habilidades requeridos de la oferta*). La comparación se realiza mediante la coincidencia con expresión regulares *regexMatch*. El algoritmo recorre todos los valores de trabajos deseados y habilidades del candidato, para ver cuánto de ellos se coinciden con los campos mencionados de la oferta de trabajo, y suma este resultado como puntuaciones en cada variable.

Acto seguido, se aplica ponderaciones a cada uno de los valores de las variables de puntuación. Las ponderaciones se han definido de la siguiente forma:

- Discapacidad: 40% (Porque es el factor más importante)
- Trabajos deseados: 30%
- Habilidades: 20%
- Provincia: 10%

El resultado final de la oferta de trabajo se calcula con la suma de los productos de cada puntuación de las variables a su ponderación correspondiente. Y finalmente, se ordenan las ofertas por la puntuación final obtenida.

Para conectar todos estos procesos (filtración con query, cálculo del algoritmo de recomendación y ordenación de ofertas) se ha definido un pipeline de

aggregation de MongoDB y las etapas del pipeline están ordenadas de la siguiente forma:

- Busca todas las ofertas que tienen estado activo.
- Buscar todas las ofertas que cumplan con el query definido.
- Aplicar el algoritmo de recomendación al resultado del query.
- Popular las ofertas.
- Ordenar las ofertas por el resultado del algoritmo.

Figura 106

Implementación del query de filtración

```
const query=(filter:JobFilters)=>{
    var query:any={};
    // los trabajos que esten uno de estos ciudades
    if(!(filter.city?.length==1 && filter.city.at(0)==="")){
        query.city ={$in: filter.city}
    }
    if(filter?.disabilities.length>0){
        query.$and = filter.disabilities.map(({ type, degree }) => ({
        $or: [
            { disabilities: { $elemMatch: { type: type, degree: -1 } } }, // Si la empresa no impone restricciones
            { disabilities: { $elemMatch: { type: type, degree: { $gte: degree } } } } // Si el trabajo acepta el grado del candidato o uno mayor
        ]
    }));
    }
    if(!(filter.mode?.length==1 && filter.mode.at(0)==="")){
        query.mode={$in:filter.mode}
    }
    if(!(filter.workHours?.length==1 && filter.workHours.at(0)==="")){
        query.workHours={$in:filter.workHours}
    }
    if(!(filter.workCategory?.length==1 && filter.workCategory.at(0)==="")){
        query.workCategory={$in:filter.workCategory}
    }
    if(filter?.experience.length>0){
        query.experience={$eq:filter.experience}
    }
    if(!(filter.minumumEducation?.length==1 && filter.minumumEducation.at(0)==="")){
        query.minumumEducation={$in:filter.minumumEducation}
    }
    if(filter?.intership){
        query.intership=(String(filter.intership)==="true")
    }else{
        query.intership=(String(filter.intership)==="false")
    }

    return query;
```

Nota. La imagen muestra la implementación del query de la filtración de las ofertas de trabajo.

/api/candidate-home/search-jobs:

Este endpoint permite obtener los trabajos buscados por el usuario. Gracias al índice compuesto de texto de MongoDB, que facilita significativamente el proceso de búsqueda.

La búsqueda se realiza en varios campos de los documentos de la colección *jobs*: *jobTitle, companyName, requiredKnowledge, description, companysRequirements*. Para realizar la búsqueda, se ha definido un índice compuesto de texto con estos cincos campos. Este índice simplifica el proceso de búsqueda definiéndole como una fase del pipeline con el operador *$text*. Además, también permite definir ponderaciones en los campos del índice para obtener una búsqueda más precisa. Como el proceso de filtración, después de buscar las ofertas, el pipeline se entra la fase de cálculo de algoritmo para ordenar las ofertas.

Figura 107

Índice compuesto de texto

```
> db.jobs.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    {
      v: 2,
      key: { _fts: 'text', _ftsx: 1 },
      name: 'search_index',
      weights: {
        companyName: 4,
        companysRequirements: 1,
        description: 2,
        jobTitle: 5,
        requiredKnowledge: 3
      },
      default_language: 'spanish',
      language_override: 'language',
      textIndexVersion: 3
    },
    {
      v: 2,
      key: { companyName: 1 },
      name: 'companyName_1',
      background: true
    }
  ]
```

Nota. La imagen muestra el índice compuesto de texto definido para el proceso de búsqueda de ofertas de trabajo.

/api/candidate-home/get-saved-jobs

El endpoint que permite obtener las ofertas de trabajo guardadas por el candidato. Dichas ofertas se guardan en el array *savedJobs* del documento del candidato. Para obtener la información completa de la oferta, es necesario realizar dos lookups. Por un lado, para recuperar la información de la oferta en detalle y por otro lado, para la información de la empresa publicadora. Para ello, se ha definido un pipeline aggregate y dentro de este se ha configurado otro pipeline interno. El motivo es que el pipeline externo se realiza el primer lookup con la colección *jobs*, para obtener la información detallada de la oferta y el id de la empresa publicadora. Y el pipeline interno es el responsable de hacer el segundo lookup con el id de la empresa obtenido con el pipeline externo, para recuperar la información de la empresa.

Figura 108

El pipeline para obtener las ofertas guardadas

```
const user = await User.aggregate([
    {$match:{_id: new ObjectId(data._id)}},
    {$lookup:{
        from:"jobs",
        let: {savedJobs:"$savedJob"}, // para luego el pipline puede usar ese dato
        pipeline:[
            {$match:{
                $expr:{ $in:["$_id","$$savedJobs"] } // para poder usar la varianle local, con expr
            }},
            {$lookup:{
                from:"companyprofiles",
                localField:"company_id",
                foreignField:"company_id",
                as:"companyProfile"
            }},
            {$unwind:{path:"$companyProfile",preserveNullAndEmptyArrays: true}}
        ],
        as: "savedJobPopulated"
    }},
    {
        $project: {
            savedJobPopulated:1,
            firstname:1,
            lastname:1,
            email:1,
            _id:1
        }
    }
])
```

Nota. La imagen muestra la implementación del pipeline que se usa para recuperar las ofertas guardadas por los candidatos.

/api/candidate-home/get-applied-jobs

El endpoint que se envía para obtener las ofertas de trabajo aplicadas por el candidato en distintos estados. Para poder recuperar las ofertas por estados, el endpoint comprueba el estado que quiere consultar el candidato y ejecutar el pipeline de aggregate para recuperar las aplicaciones de puesto con el estado correspondiente. Por este motivo, para cada comprobación se ha definido un pipeline especifico. Cada pipeline se filtra primero por el estado de la oferta (activa o cerrada), y luego por el estado de la aplicación (solicitado, a comunicar, comunicado). Finalmente, popularizarlas para obtener información completa de las ofertas filtradas.

Figura 109

Función para obtener las ofertas aplicadas

```
if(activePage==="solicitados"){
    job= await Job.aggregate([
    {$match:{currentStatus:"active"}},
    {$match: {
        applicants: { $elemMatch: { user: new ObjectId(data._id), status:"solicitado"  } }
    }},
    {$lookup:{
        from:"companyprofiles",
        localField:"company_id",
        foreignField:"company_id",
        as:"companyProfile"
    }},
    {$unwind:{path:"$companyProfile",preserveNullAndEmptyArrays: true}}
    ])
}else if(activePage ==="enCurso"){
    job= await Job.aggregate([
        {$match:{currentStatus:"active"}},
        {$match: {
            applicants: { $elemMatch: { user: new ObjectId(data._id), $or:[{status:"a comunicar"},{status:"comunicado"}] } }
        }},
        {$lookup:{
            from:"companyprofiles",
            localField:"company_id",
            foreignField:"company_id",
            as:"companyProfile"
        }},
        {$unwind:{path:"$companyProfile",preserveNullAndEmptyArrays: true}}
        ])
}else if(activePage==="cerrados"){
    job= await Job.aggregate([
        {$match:{currentStatus:"closed"}},
        {$match: {
            applicants: { $elemMatch: { user: new ObjectId(data._id) } }
        }},
        {$lookup:{
            from:"companyprofiles",
            localField:"company_id",
            foreignField:"company_id",
```

Nota. La imagen muestra la lógica de obtener las aplicaciones de ofertas de trabajo de los candidatos en distintos estados.

/api/candidate-home/get-job

Este endpoint permite obtener la información completa de una oferta de trabajo para la página de detalles. Primero, busca la oferta en la colección jobs, a continuación, hacen dos lookups para obtener la información de empresa publicadora: uno para obtener información básica de la empresa con la colección de *companies* y el otro para obtener información perfil de la empresa con la colección *companyprofiles*.

Figura 110

El pipeline para obtener detalles de una oferta de trabajo

```
const job= await Job.aggregate([
    {$match:{_id: new ObjectId(id)}},
    {$lookup:{
        from: "companies",
        localField:"company_id",
        foreignField:"_id",
        as: "companyInfo"
    }}, // para obtener companyName
    {$unwind:{path:"$companyInfo",preserveNullAndEmptyArrays:true}},
    {$lookup:{
        from:"companyprofiles",
        localField:"company_id",
        foreignField:"company_id",
        as:"companyProfile"
    }},
    {$unwind:{path:"$companyProfile",preserveNullAndEmptyArrays: true}}
])
```

Nota. La imagen muestra la implementación del pipeline que usa para obtener la información detallada de una oferta de trabajo.

/api/candidate-home/apply-job

El endpoint que se envía cuando el candidato aplica una oferta de trabajo. El proceso es sencillo, se actualiza el array de applicants en el documento de la oferta añadiendo el id del candidato y definiendo la aplicación con el estado inicial como "*solicitado*".

Figura 111

Actualización para aplicar una oferta de trabajo

```
const body = await request.json();
const {job} = body;

const resul = await Job.findOneAndUpdate(
    {_id:job},
    {
    $push: {applicants: {
        user:data._id,
        status:"solicitado"
    }}
    },
    {new:true})
```

Nota. La imagen muestra el proceso que se ejecuta cuando un candidato aplica una oferta de trabajo.

/api/candidate-home/save-job

El endpoint que se envía cuando el candidato guarda o quitar de guardadas una oferta de trabajo. En el endpoint se comprueba primero si el id de la oferta actual existe ya en el array savedJobs del documento del candidato, en el caso confirmativo, significa que el candidato quiere quitar la oferta en la lista de ofertas guardadas y se elimina. En el caso contrario, se guarda en el array savedJobs.

Figura 112

Guarda / No guardar una oferta de trabajo

```
const body = await request.json();
const {job} = body

if (user.savedJob.includes(job)) {
    let index = user.savedJob.indexOf(job);
    user.savedJob.splice(index,1);
} else {
    user.savedJob.push(job);
}

const resul = await user.save();
```

Nota. La imagen muestra el proceso para guardar o quitar de guardadas una oferta de trabajo para un candidato.

/api/profile/get-profile

Es el endpoint que permite obtener el perfil del candidato para la página de gestión de perfil. Busca el perfil en la colección *profiles*, pupula con la colección *users* para obtener la información básica del candidato y devuelve el resultado.

Figura 113

Obtener el perfil de un candidato

```
try{
    //@ts-ignore
    const {data} = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
    const profile = await CandidateProfile.findOne({user_id: data._id}).populate('user_id').lean();

    if(!profile){
        return NextResponse.json({
            error:message.error.profileLoadError
        }, {status:400})
    }
    return NextResponse.json({
        profile:profile
    },{status:200})
```

Nota. La imagen muestra el proceso de obtención del perfil de un candidato.

/api/profile/edit-profile

En endpoint que se envía cuando gestiona el perfil del candidato. Dado que el perfil del candidato está divido en secciones. Con los datos recibidos, se

identifica primero la sección que pertenecen los datos y se actualiza sobre dicha sección. Porque hay varias secciones, la información detallada sobre la implementación puede consultarse en el código fuente.

A continuación, empiezan las explicaciones de las rutas API relacionadas con la cuenta empresarial, que destacan los siguientes endpoints:

/api/company-jobs/get-jobs

Este endpoint permite obtener todas las ofertas de trabajo publicadas por la empresa, se usa en el contexto de empresa. El proceso es sencillo, se busca en la colección *jobs* las ofertas con el campo *company_id d*el id de la empresa actual.

Figura 114

Obtención de ofertas de una empresa

```
try{
    //@ts-ignore
    const {data} = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
    const jobs = await Job.find({company_id: data._id}).populate('company_id').lean();

    if(!jobs){
        return NextResponse.json({
            error:message.error.jobsLoadError
        }, {status:400})
    }
    return NextResponse.json({
        jobs:jobs
    },{status:200})
```

Nota. La imagen muestra el proceso de obtención de ofertas de una empresa.

/api/company-jobs/edit-job

El endpoint que se envía cuando se gestiona una oferta de trabajo. Primero, busca el documento de la oferta actual en la colección de *jobs*, y actualizarlo con los datos pasados por el usuario.

Figura 115

Actualización de información de una oferta de trabajo

```
job.currentStatus=currentStatus
job.jobTitle =jobTitle
job.city=city
job.mode=mode
job.workHours=workHours
job.experience=experience
job.intership=intership
job.workCategory=workCategory
job.disabilities = [
    {type:FISICA, degree:fisica},
    {type: AUDITIVA, degree:auditiva},
    {type: VISUAL, degree:visual},
    {type: HABLAR, degree:hablar},
    {type: MENTAL, degree:mental},
    {type: INTELECTUAL, degree:intelectual},
    {type: PLURIDISCAPACIDAD, degree:pluridiscapacidad},
]
job.minumumEducation=minumumEducation
job.languages=languages
job.requiredKnowledge=requiredKnowledge
job.companysRequirements=companysRequirements
job.description=description
const resul = await job.save();
```

Nota. La imagen muestra el proceso para actualizar información de una oferta de trabajo.

/api/company-jobs/add-job

El endpoint que se envía cuando se publica una nueva oferta de trabajo. El proceso es sencillo, se crea un documento nuevo en la colección *jobs* con los datos introducidos por el usuario.

Figura 116

Creación de una nueva oferta de trabajo

```
const newJob:IJobDocument = new Job({
        company_id: data._id,
        applicants:[],
        currentStatus:currentStatus,
        jobTitle: jobTitle,
        city: city,
        mode: mode,
        workHours:workHours,
        experience:experience,
        intership:intership,
        workCategory: workCategory,
        disabilities: [
            {type:FISICA, degree:fisica},
            {type: AUDITIVA, degree:auditiva},
            {type: VISUAL, degree:visual},
            {type: HABLAR, degree:hablar},
            {type: MENTAL, degree:mental},
            {type: INTELECTUAL, degree:intelectual},
            {type: PLURIDISCAPACIDAD, degree:pluridiscapacidad},
        ],
        minumumEducation: minumumEducation,
        languages:languages,
        requiredKnowledge:requiredKnowledge,
        companysRequirements:companysRequirements,
        description:description
});

    await newJob.save();
```

Nota. La imagen muestra el proceso para publicar una nueva oferta de trabajo.

/api/company-jobs

Este endpoint permite obtener todas las ofertas activas de una empresa. Se utiliza en la página de información de una empresa, en la sección de visualizar las ofertas disponibles. El pipeline definido busca primero todas las ofertas de la empresa, luego filtrar aquellas que tienen con el estado activa, y por último, popularizarlas con la colección de *companies* y la colección *companyprofiles* para obtener información de la empresa.

Figura 117

Obtención de ofertas activas de una empresa

```
const job = await Job.aggregate([
    {$match:{company_id: new ObjectId(id)}},
    {$match:{currentStatus:"active"}},
    {$lookup:{
        from: "companies",
        localField:"company_id",
        foreignField:"_id",
        as: "companyInfo"
    }}, // para obtener companyName
    {$unwind:{path:"$companyInfo",preserveNullAndEmptyArrays:true}},
    {$lookup:{
        from:"companyprofiles",
        localField:"company_id",
        foreignField:"company_id",
        as:"companyProfile"
    }},
    {$unwind:{path:"$companyProfile",preserveNullAndEmptyArrays: true}}
])
```

Nota. La imagen muestra el pipeline para obtener todas las ofertas activas de una empresa.

/api/company-candidates/get-previews

Este endpoint permite obtener todos los candidatos solicitados con un estado especificado de una oferta de trabajo. Se ha usado el operador *$unwind* para generar un documento por cada valor del array applicants de la oferta. Con esto, se puede distinguir de manera más calara la información de cada candidato después de popularizar con la colección *users*. El flujo del pipeline es el siguiente: busca el documento de la oferta, unwind su array applicants, filtra los candidatos por el estado indicado y populariza con la colección *users* y *candiateProfiles* para obtener los datos de los candidatos.

Figura 118

Obtención de candidatos en un estado especificado de una oferta

```
const candidates = await Job.aggregate([
    { $match: { _id: new ObjectId(id) } },
    { $unwind: "$applicants" },
    { $match: { "applicants.status": status } },
    {$lookup:{
        from:"candidateprofiles",
        localField:"applicants.user",
        foreignField:"user_id",
        as:"userProfileInfo"
    }},
    {$unwind:{path:"$userProfileInfo", preserveNullAndEmptyArrays:true}},
    {$lookup:{
        from:"users",
        localField:"applicants.user",
        foreignField:"_id",
        as:"userInfo"
    }},
    {$unwind:{path:"$userInfo", preserveNullAndEmptyArrays:true}},
]);
```

Nota. La imagen muestra el proceso para obtener todos los candidatos solicitados con un estado especificado de una oferta de trabajo.

/api/company-candidates/get-candidate-detail

Este endpoint permite obtener el perfil de un candidato cuando la empresa consulta el detalle del candidato. Busca en la colección *candidateprofiles* el documento del candidato que desea consultar y luego, popularizar con la colección *users* sin incluir la contraseña del candidato para obtener la información básica del candidato.

Figura 119

Obtención de detalles de un candidato

```
const {data} = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
const body = await request.json();
const {candidate} = body

const profile = await CandidateProfile.findOne({user_id: candidate}).populate('user_id','-password').lean();
```

Nota. La imagen muestra el proceso de obtención de perfil de un candidato para una empresa.

/api/company-candidates/get-candidate-detail

El endpoint que se envía cuando modifica el estado de aplicación de un candidato en una oferta. Se busca primero el índice del candidato en el array del applicants de la oferta, y a continuación, obtener el objeto con el índice y actualizar su estado con el estado indicado.

Figura 120

Actualización de estado de una aplicación

```
const job = await Job.findOne({_id:id});
const index = job.applicants.findIndex(elem=> elem.user.toString() === candidate);
job.applicants.set(index, {user: new ObjectId(candidate), status:status});
const resul = await job.save();
if(!resul){
    return NextResponse.json({
        error:message.error.statusUpdateError
    },{status:400})
}
return NextResponse.json({
    sucess:message.sucess.StatusUpdated
},{status:200})
```

Nota. La imagen muestra el proceso de actualización de estado de una aplicación.

/api/company-operators/add-operator

El endpoint que se envía cuando la empresa se añade un operador nuevo par su cuenta. En primer lugar, se realiza un conjunto de validaciones, que son: comprobar si todos los campos estén informados, comprobar si el email ya existe en la colección *operators*, comprobar el formato de email y el de nombre y apellidos. Cuando supera las validaciones, se genera una contraseña temporal aleatoriamente y se crea un documento nuevo en la colección *operators* con los datos proporcionados y la contraseña generada. Además, se inicializa el valor del campo *changedPassword* como false para indicar que la cuenta no está activada.

Finalmente, se envía un correo de aviso de activación de cuenta operador al email del operador configurado con un token de validez de dos días.

Figura 121

Envío de email de aviso de activación de cuenta operador

```
const url = `http://localhost:3000/reset-password-operator?token=${token}`;

await resend.emails.send({
    from: "onboarding@resend.dev",
    to: [redacted]
    subject: "Bienvenido a la plataforma CoWorld",
    react: EmialOperatorTemplate({url:url,firstname:newOperator.firstname,companyName:data.companyName,tempPwd:tempPassword}),
});

// datas de response
const response = NextResponse.json({
    newOperator: rest,
    sucess:message.sucess.OperatorCreated
},{status:200})
```

Nota. La imagen muestra el proceso de enviar el email de aviso de activación de cuenta operador.

/api/company-operators/edit-operator

El endpoint que se envía para gestionar un operador. Se actualiza el documento del operador con los datos enviados. En caso de eliminar, se borra del documento en la colección operators.

Figura 122

Actualización de un operador

```
if(isDelete){
    await Operator.deleteOne({_id: id});
    return NextResponse.json({
        sucess:message.sucess.OperatorDeleted
    },{status:200})
}else{
    if(!onlyLetters(firstname)){
        return NextResponse.json({
            error: message.error.onlyLetterName
        }, {status:400})
    }
    if(!onlyLastNames(lastname)){
        return NextResponse.json({
            error: message.error.onlyLetterSpace
        }, {status:400})
    }
    operator.firstname = firstname;
    operator.lastname = lastname;
    await operator.save();
    return NextResponse.json({
        sucess:message.sucess.OperatorModified
    },{status:200})
}
```

Nota. La imagen muestra el proceso de actualización de información de un operador.

/api/company-operators/get-operators

Este endpoint permite obtener todos los operadores de la empresa. El proceso es sencillo, se busca todos los operadores con el campo *company_id* del valor del id de la empresa actual.

Figura 123

Obtención de operadores de una empresa

```
const {data} = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
const body = await request.json();
const {page} = body;

const operators = await Operator.find({company_id: data._id}).populate('company_id').lean();
```

Nota. La imagen muestra el proceso para obtener todos los operadores de una empresa.

/api/company-profile/get-profile

Este endpoint permite obtener el perfil de la de la empresa. Busca el documento de perfil de la empresa en la colección *companyprofiles* y populariza con la colección *companies* para obtener información completa de la empresa.

Figura 124

Obtención de perfil de una empresa

```
const accessToken = request.cookies.get('accessTokenCookie').value;
try{
    //@ts-ignore
    const {data} = jwt.verify(accessToken,process.env.ACCESS_TOKEN_SECRET)
    const profile = await CompanyProfile.findOne({company_id: data._id}).populate('company_id').lean();

    if(!profile){
```

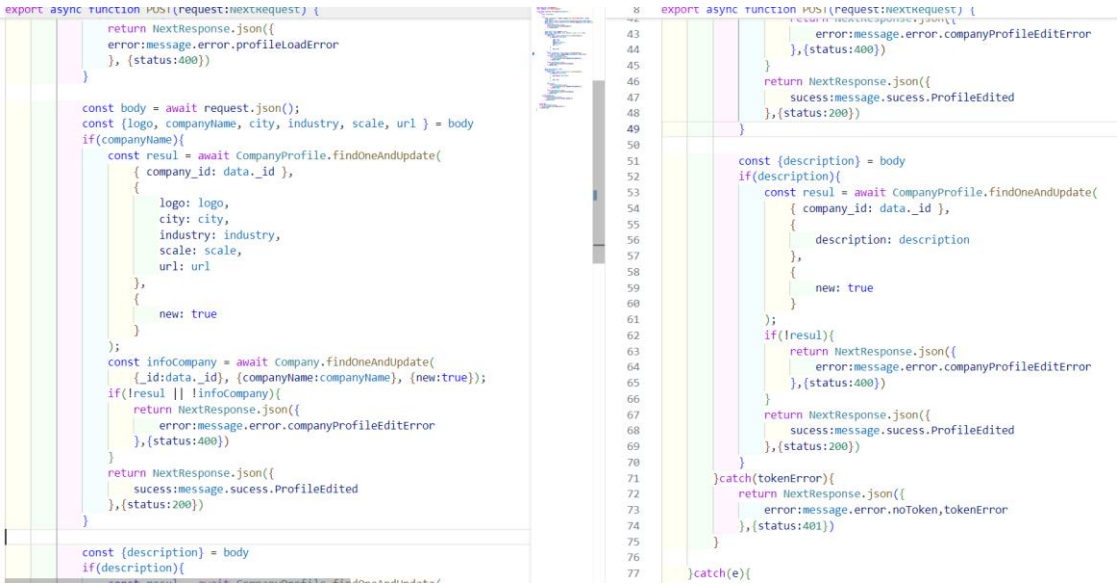
Nota. La imagen muestra el proceso para obtener el perfil de una empresa.

/api/company-profile/edit-profile

En endpoint que se envía cuando gestiona el perfil de la empresa. Dado que el perfil está divido en secciones. Con los datos recibidos, se identifica primero la sección que pertenecen los datos y se actualiza sobre dicha sección.

Figura 125

Gestión de perfil de una empresa



```
export async function POST(request:NextRequest) {
        return NextResponse.json({
        error:message.error.profileLoadError
        }, {status:400})
    }

    const body = await request.json();
    const {logo, companyName, city, industry, scale, url } = body
    if(companyName){
        const resul = await CompanyProfile.findOneAndUpdate(
            { company_id: data._id },
            {
                logo: logo,
                city: city,
                industry: industry,
                scale: scale,
                url: url
            },
            {
                new: true
            }
        );
        const infoCompany = await Company.findOneAndUpdate(
            {_id:data._id}, {companyName:companyName}, {new:true});
        if(!resul || !infoCompany){
            return NextResponse.json({
                error:message.error.companyProfileEditError
            },{status:400})
        }
        return NextResponse.json({
            sucess:message.sucess.ProfileEdited
        },{status:200})
    }

    const {description} = body
    if(description){
```

```
export async function POST(request:NextRequest) {
                error:message.error.companyProfileEditError
            },{status:400})
        }
        return NextResponse.json({
            sucess:message.sucess.ProfileEdited
        },{status:200})
    }

    const {description} = body
    if(description){
        const resul = await CompanyProfile.findOneAndUpdate(
            { company_id: data._id },
            {
                description: description
            },
            {
                new: true
            }
        );
        if(!resul){
            return NextResponse.json({
                error:message.error.companyProfileEditError
            },{status:400})
        }
        return NextResponse.json({
            sucess:message.sucess.ProfileEdited
        },{status:200})
    }
}catch(tokenError){
    return NextResponse.json({
        error:message.error.noToken,tokenError
    },{status:401})
}

}catch(e){
```

Nota. La imagen muestra el proceso de gestionar el perfil de una empresa.

4.4. Pruebas

En esta sección, se muestran los resultados obtenidos de las pruebas realizadas al finalizar la implementación de la aplicación web CoWorld. Con el propósito de verificar el cumplimiento de las especificaciones funcionales y técnicas, se han realizado las pruebas de sistema para comprobar el funcionamiento completo del sistema.

Las pruebas están organizadas por gestión de autenticación y los tres roles de la aplicación: Candidato, Empresa y Operador. Y para cada bloque se ha definido una serie

de pruebas funcionales correspondientes a las funcionalidades claves que puede realizar el usuario correspondiente.

A continuación se muestran las pruebas y los resultados de estas por bloque:

Gestión de autenticación:

En este bloque, las pruebas están diseñadas en torno a los comportamientos que la aplicación debe actuar en diferentes escenarios para cada funcionalidad del sistema de autenticación. Dichas funcionalidades son las siguientes: Registrar una cuenta nueva de Candidato/Empresa, Iniciar/Cerrar sesión, Restablecer una contraseña, Dar de alta una cuenta de operador, y Activar una cuenta operador.

Figura 126

Tests de gestión de autenticación

Autenticacion	Candidato	Empresa	Operador
Registrar correctamente cuando se envia con los datos completos y cumpliendo con las restricciones	✓	✓	
No permitir el registro cuando falten datos	✓	✓	✓
Inicio de sesión con éxito cuando se introduce un correo electrónico existente en la base de datos y la contraseña es correcta	✓	✓	✓
No permita iniciar sesión cuando se introduce una contraseña incorrecta	✓	✓	✓
No permitir iniciar sesión cuando se introduce una dirección de correo electrónico no existente	✓	✓	✓
Forget password --> Recibe correctamente el correo de forget password y puede acceder al link	✓	✓	✓
Puede cambiar correctamente la contraseña en la página de restablecimiento de contraseña	✓	✓	✓
Cerrar sesión correctamente --> Eliminará el accessToken y refreshToken de la sesión actual.	✓	✓	✓
No permitir que el operador acceda sin cambiar la contraseña			✓
Los operadores reciben correctamente el correo de aviso de activación de cuenta			✓
No permitir los operadores activar la cuenta sin introducir la contraseña temporal correcta			✓
Los operadores pueden activar su cuenta correctamente			✓
No es posible darse de alta como operador por su cuenta, solo puede hacerlo su empresa			✓

Nota. La imagen se muestran los resultados obtenidos en las pruebas de la gestión de autenticación.

Las celdas amarillas indican que las funcionalidades en prueba no son del rol correspondiente.

En cuanto a las funcionalidades de registro y de restablecimiento de contraseña, existen varias restricciones relacionadas con los datos que deben proporcionarse para realizar estas funcionalidades, por lo que también se han definido pruebas para cada restricción en función de los criterios que tiene cada uno. Y las restricciones se encuentran en los siguientes datos: Nombre y apellidos, Correo electrónico, Contraseña de la cuenta nueva, Contraseña que se desea restablecer y C.I.F. de la empresa.

Figura 127

Tests de restricción de registración y de restablecimiento de contraseña

Restricciones de Registración	Resultado
Nombre solo puede contener letras	✓
Aplleidos solo pueden coneter letras y entre aplleidos solo puede tener 1 espacio blanco	✓
El correo debe tener formato valido	✓
La contraseña debe tener >= 8 de longitud y al menos 1 mayuscula 1 digito y 1 caracter especial	✓
No permitir registrar otra vez con un correo que ya existe	✓
Formtao de C.I.F: 1 letra seguida de 8 digitos	✓

Restriccion de reset password	Resultado
Las contraseñas introducidas deben conincidir y cumplir con formato	✓

Nota. La imagen se muestran los resultados obtenidos en las pruebas de las restricciones de registración y de restablecimiento de contraseña.

Las funcionalidades de los candidatos:

En las pruebas del bloque candidato, se puede ver las pruebas que están clasificadas en cinco subgrupos, cada grupo contiene las pruebas relacionadas entre sí.

El primer grupo contiene las pruebas relacionadas con las funcionalidades de filtrar y de buscar ofertas de trabajo.

En el segundo grupo están las pruebas para comprobar que las ofertas de trabajo que el candidato ha guardado o aplicado se actualizan correctamente en distintos listados según su estado. Dado que las funcionalidades de guardar o aplicar una oferta de trabajo tienen como consecuencia el cambio de estado de una oferta de trabajo, las pruebas de dichas funcionalidades se realizan en este grupo. Además, se comprueba también cuando las empresas de las ofertas actualizan el estado de las aplicaciones de los candidatos, ya que los cambios se reflejan en los listados de ofertas de los candidatos.

En el tercer grupo están las pruebas que comprueban si la información del perfil del candidato se ha actualizado correctamente cuando el candidato la gestiona. De manera que, el algoritmo de recomendación de la aplicación se basa en la información del perfil del candidato, se comprueba también si las recomendaciones se actualizan correctamente según los cambios realizados en el perfil.

En el cuarto grupo están las pruebas relacionadas con la funcionalidad de ver información detallada de una oferta. Se comprueba centralmente que la información de la oferta y de las opciones que puede realizar el candidato según el estado actual que tiene de la oferta se renderizan correctamente.

Por último, están las pruebas relacionadas con la funcionalidad de ver la página de información de una empresa y las operaciones que se pueden realizar en ella.

Figura 128

Tests de las funcionalidades de los candidatos

Candidato	Resultado
Listado de ofertas sin filtros	✓
Listado de ofertas solo con filtros	✓
Listado de ofertas de buscado	✓
Listado de ofertas de buscado y filtrado	✓
Se puede guardar/ quitar un trabajo	✓
Se puede aplicar un trabajo	✓
Al guardar trabajo se muestra en el listado de guardar	✓
Al aplicar el trabajo guardado se muestra en el listado de solictados	✓
Al aplicar directamente se muestra en el listado de solictados	✓
Cuando empresa dar "A comunicar ", la apliacion correspondiente se muestra en el listado de en curso	✓
Cuando empresa dar "Cerrar", la apliacion correspondiente se muestra en el listado de en cerrado	✓
Se puede editar todas las secciones de perfil y actualiza bien	✓
Al editar: Discapcidad, trabajo deseados, skills, ciudad, se ve cambios el listado de ofertas	✓
Se puede ver detalle de una trabajo	✓
Si el candidato tiene guardado el trabajo --> Mustra boton de Quitar de guardado	✓
Si el candidato no tiene guardado el trabajo --> Mustra boton de Guardar	✓
Si el candidto tiene aplicado el trabajo --> Muestra mensaje y no deja guardar ni aplicar ni los contrarios	✓
Se puede ver detalle de una empresa desde listado	✓
Se puede ver detalle de una empresa desde detalle de un trabajo	✓
Se puede ver listados de ofertas de trabajo que tiene la empresa	✓
Al acceder 1 oferta desde detalle de empresa, mustra correctamente opciones de acciones que puede realizar sobre el trabajo	✓

Nota. En la imagen se muestran los resultados obtenidos en las pruebas de las funcionalidades de los candidatos.

Las funcionalidades de las empresas:

De forma similar al bloque de candidatos, las pruebas de empresas están divididas en cuatro subgrupos.

En el primer grupo están las pruebas relacionadas con las operaciones CRUD de la gestión de ofertas de trabajo, que son: crear, eliminar, actualizar y listar las ofertas de trabajo de la empresa.

El segundo grupo está compuesto por las pruebas relacionadas con las funcionalidades de gestión de candidatos inscritos en las ofertas de la empresa. Se comprueba si es posible obtener todos los candidatos inscritos clasificados en diferentes estados de una oferta, así como ver un candidato en detalle y las opciones de cambio de estado que tiene según su estado actual.

El tercer grupo están las pruebas relacionadas con las operaciones CRUD de la gestión de operadores de la empresa, que son: añadir, modificar, eliminar y listar los operadores de la empresa.

Por último, se comprueba la correcta actualización de información de perfil de la empresa.

Figura 129

Tests de las funcionalidades de las empresas

Empresa	Resultado
Se puede publicar trabjos	✓
Se puede modificar trabajos	✓
Se puede eliminare trabajos, cuando elimina, no debe mostrar mas en el parte de candidatos	✓
Se mustra correctamente trabajos en la lista En curso y Cerrados	✓
Se puede visualisar los candidatos que tiene 1 oferta	✓
Se puede ver en detalle el candidato de 1 oferta	✓
Se puede cambiar estado de candidato a "A comunicar", se debe actualizar en parte de candidato	✓
Se puede cambiar estado de candidato a "Comunicado"	✓
Se puede dar alta un operador	✓
Se puede listar todos los operadores de la empresa	✓
Se puede modificar info de 1 operador	✓
Se puede Eliminar 1 operador y operador ya no puede acceder la aplicacion con rol de "Operador"	✓
Se puede editar perfil de la empresa y actualiza correctamente	✓

Nota. La imagen se muestran los resultados obtenidos en las pruebas de las funcionalidades de las empresas.

En cuanto a las pruebas de gestión del perfil tanto en la parte del candidato como en la de la empresa, se comprueban también las restricciones existentes en la información

del perfil. Se ha definido una prueba para cada restricción en función de sus criterios, las restricciones se encuentran en los siguientes datos: nombre y apellidos, teléfono de los candidatos, el componente “tag input” y fechas.

Figura 130

Tests de las restricciones de información del perfil

Restriccion de Perfil	Rsultado
Nombre y apellidos solo puede contener letras y entre apellidos solo 1 espacio blanco	✓
Telefono solo puede ser de 9 digitos	✓
Tags no se puede repetir ni espacio blanco	✓
fecha inicio y fin de esdio y experiencia laboral	✓

Nota. La imagen se muestran los resultados obtenidos en las pruebas de las restricciones del perfil.

Las funcionalidades de los operadores:

En este último bloque, se incluyen todas las pruebas relacionadas con las funcionalidades de los operadores y en total hay tres subgrupos. El primero de ellos consiste en garantizar que el control de acceso funciona correctamente, ya que los operadores tienen acceso a las mismas funcionalidades que las empresas, excepto a las de gestión del perfil y gestión de los operadores de la empresa. Asimismo, las pruebas de las funcionalidades comunes con las empresas se mantienen iguales que las pruebas diseñadas en el bloque de empresa.

El segundo grupo están las pruebas de las funcionalidades de gestión de las ofertas de trabajo de la empresa, que son las mismas que las del bloque de empresa.

Del mismo modo, el tercer grupo se compone de las pruebas de las funcionalidades de gestión de candidatos inscritos para las ofertas de la empresa, que son iguales que las del bloque de empresa.

Figura 131

Tests de las funcionalidades de los operadores

Operador	Resultado
Al acceder aplicacion, no se muestra opciones de "Operador" y "Perfil"	✓
Se puede publicar trabjos y actualiza en cuenta de empresa	✓
Se puede modificar trabajos y actualiza en cuenta de empresa	✓
Se puede eliminare trabajos, cuando elimina, no debe mostrar mas en el parte de candidatos y actualiza en cuenta de empresa	✓
Se mustra correctamente trabajos en la lista En curso y Cerrados	✓
Se puede visualisar los candidatos que tiene 1 oferta	✓
Se puede ver en detalle el candidato de 1 oferta	✓
Se puede cambiar estado de candidato a "A comunicar", se debe actualizar en parte de candidato y actualiza en cuanta de empresa	✓
Se puede cambiar estado de candidato a "Comunicado" y actualiza en cuenta de empresa	✓

Nota. La imagen se muestran los resultados obtenidos en las pruebas de las funcionalidades de los operadores.

5. CoWorld

5.1. Resultados

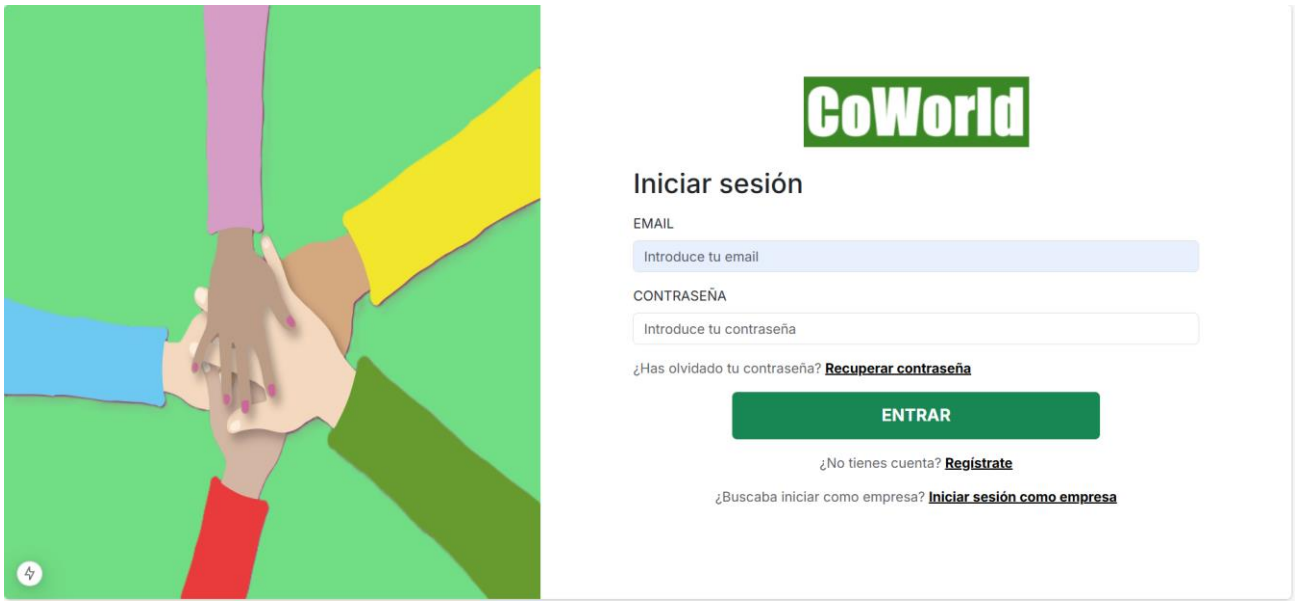
En este apartado, se presentan los resultados obtenidos de la implementación: la aplicación web CoWorld. El propósito principal de esta sección es exponer las vistas finales de la aplicación y presentar de manera breve las funcionalidades que ofrece a los usuarios.

5.1.1. Usuario con el rol de candidato

Al principio, el usuario tiene que iniciar sesión con su correo electrónico y contraseña para utilizar la aplicación.

Figura 132

Portal de inicio de sesión de candidatos

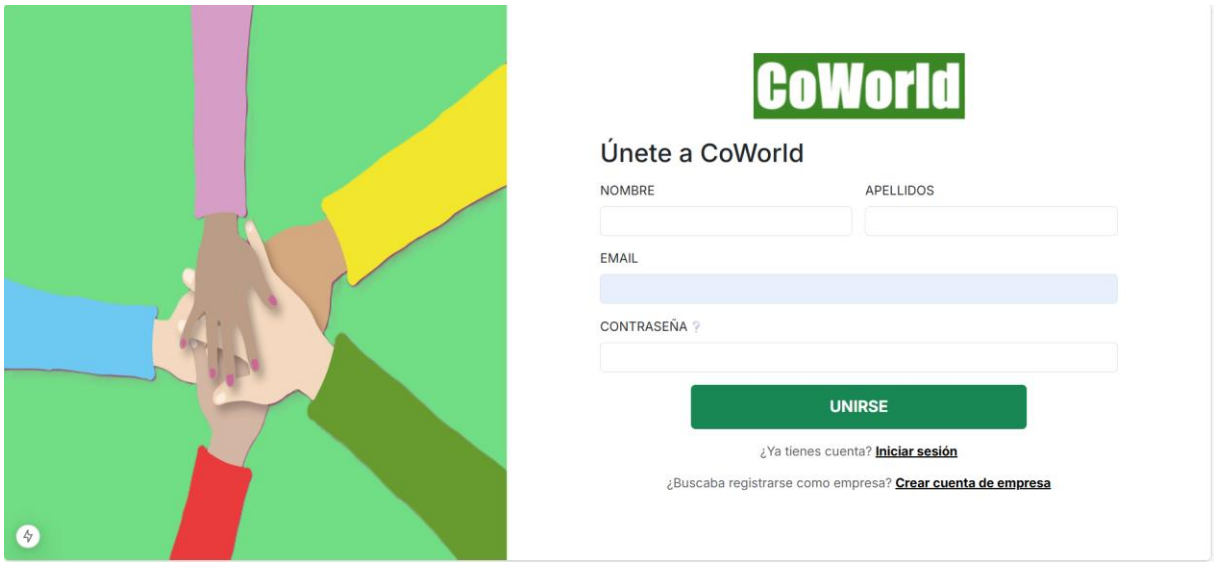


Nota. La imagen muestra la vista de inicio de sesión para los candidatos.

En el caso si es un usuario nuevo, se puede registrar una cuenta de rol de candidato con el enlace indicada con la palabra "*Regístrate*", es necesario completar todos los campos para registrar, y el formato de la contraseña restringida se puede consultar ubicando el ratón sobre el icono de interrogación .

Figura 133

Portal de registración de cuenta candidata

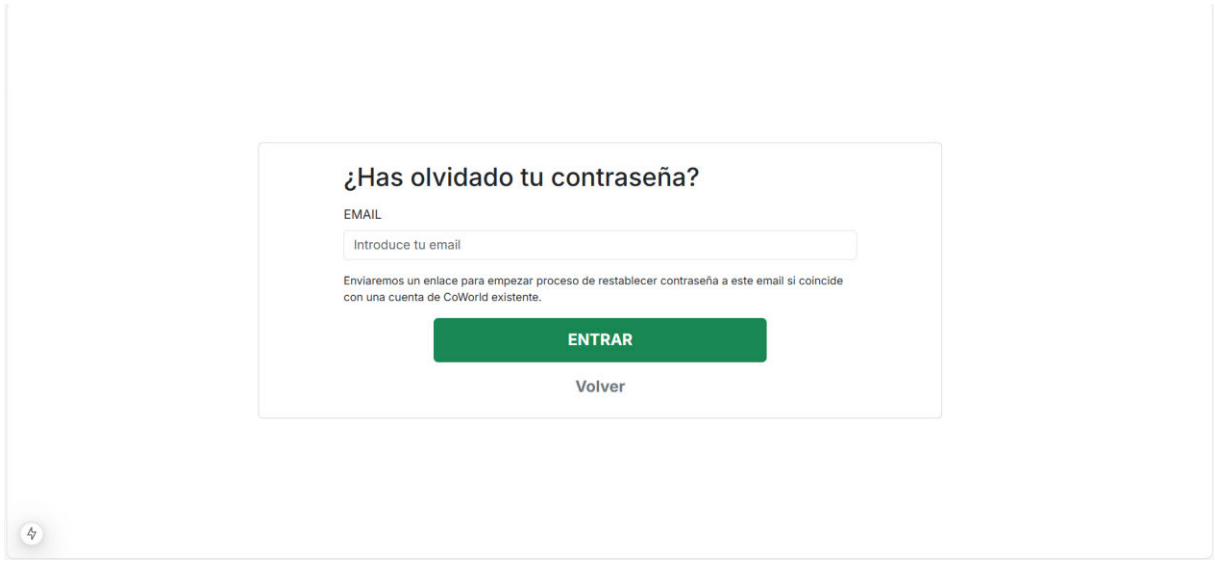


Nota. La imagen muestra la vista para realizar la registración de una cuenta de rol candidato.

Si el usuario no se acuerda de la contraseña, puede solicitar el proceso de restablecimiento de contraseña mediante el enlace indicado con "*Recuperar contraseña*". El enlace redirige a la página de solicitud de restablecimiento de contraseña, en la cual debe introducir el correo electrónico de la cuenta que desea recuperar. En el caso de que se haya comprobado la existencia de la cuenta, se emitirá un email de aviso de restablecimiento. En el correo se incluirá un enlace que redirige a la página donde podrá restablecer la contraseña, y la pagina cuenta con una validez de una hora por motivos de seguridad.

Figura 134

Portal de solicitud de restablecimiento de contraseña



Nota. La imagen muestra la vista donde se solicita el restablecimiento de contraseña.

Figura 135

Correo de aviso de restablecimiento de contraseña

Nota. La imagen muestra el contenido del correo que se recibe cuando se solicita el restablecimiento de la contraseña.

Figura 136

Portal de restablecimiento de contraseña

Nota. La imagen muestra la vista donde se restablece la contraseña de una cuenta.

Una vez iniciada la sesión con éxito, el usuario es redirigido a la página inicial de la aplicación. En la parte superior de la página se encuentra la barra de navegación, que permite los candidatos navegar entre distintas páginas. A continuación, se sitúa la barra de búsqueda y los filtros, son las herramientas diseñadas para facilitar los candidatos la búsqueda de empleos que mejor adopten a sus necesidades. Justo debajo

de ellos, se listan las ofertas de trabajo disponibles. En el caso de sin realizar ninguna filtración o búsqueda, se muestra las ofertas disponibles de la aplicación ordenado según el algoritmo de recomendación ajustando a la información de perfil de los candidatos. Cuando se aplica filtros, el resultado se actualiza en tiempo real a medida que cambian los filtros. Además, es posible aplicar filtros en los resultados una búsqueda.

Figura 137

Página inicial para los candidatos

Nota. La imagen muestra la vista que se presenta a los candidatos después de iniciar sesión.

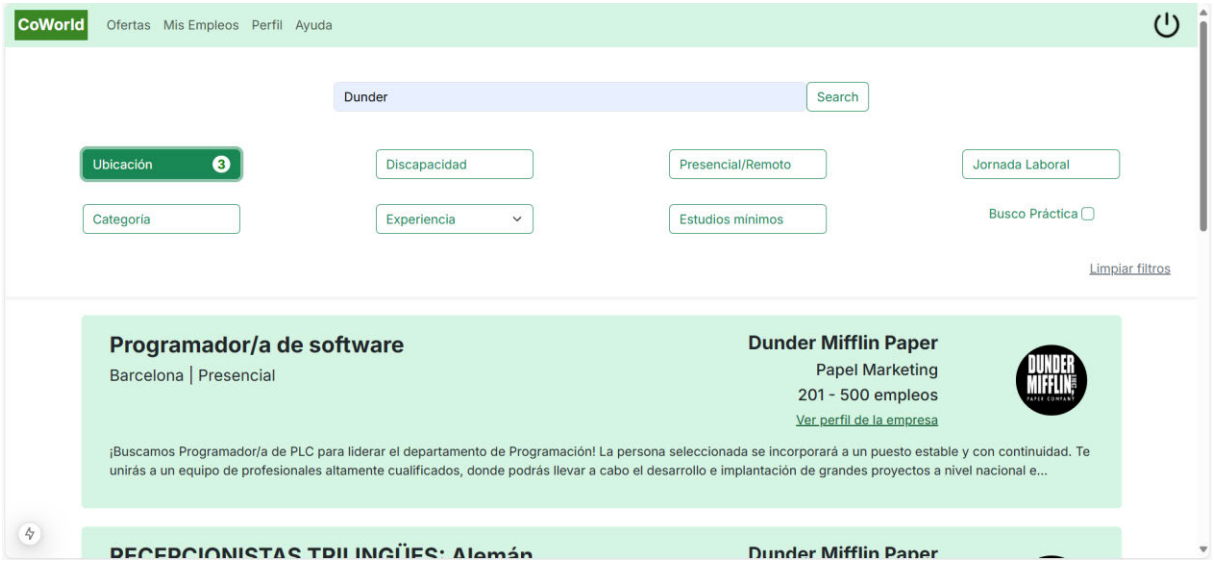
Figura 138

Ejemplo de aplicar filtros

Nota. La imagen muestra la vista que presenta a los candidatos después de aplicar filtros.

Figura 139

Ejemplo de buscar y aplicar filtros



Nota. La imagen muestra la vista que muestra tras de realizar una búsqueda y aplicar filtros.

El usuario puede seleccionar la oferta de trabajo para ver más detalles. Además, desde el detalle de la oferta es posible acceder a la página de información de la empresa anunciante. En la página de información de una empresa se puede ver, por un lado, el perfil de la empresa, y por otro lado, las ofertas disponibles de la empresa.

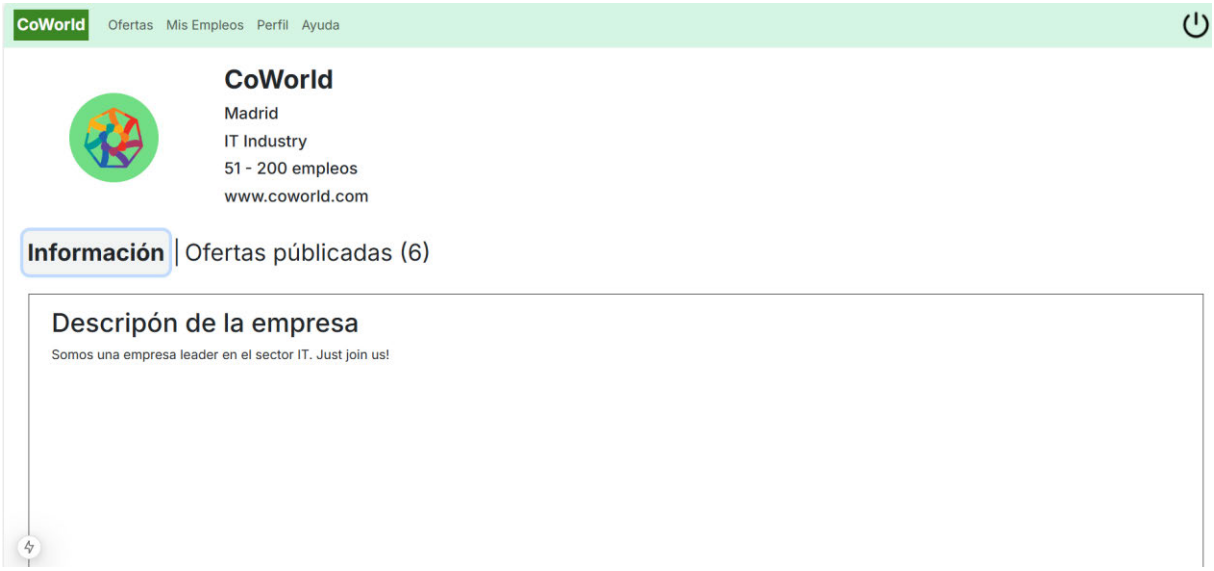
Figura 140

Página de detalle de una oferta de trabajo

Nota. La imagen muestra la vista de detalle de una oferta de trabajo.

Figura 141

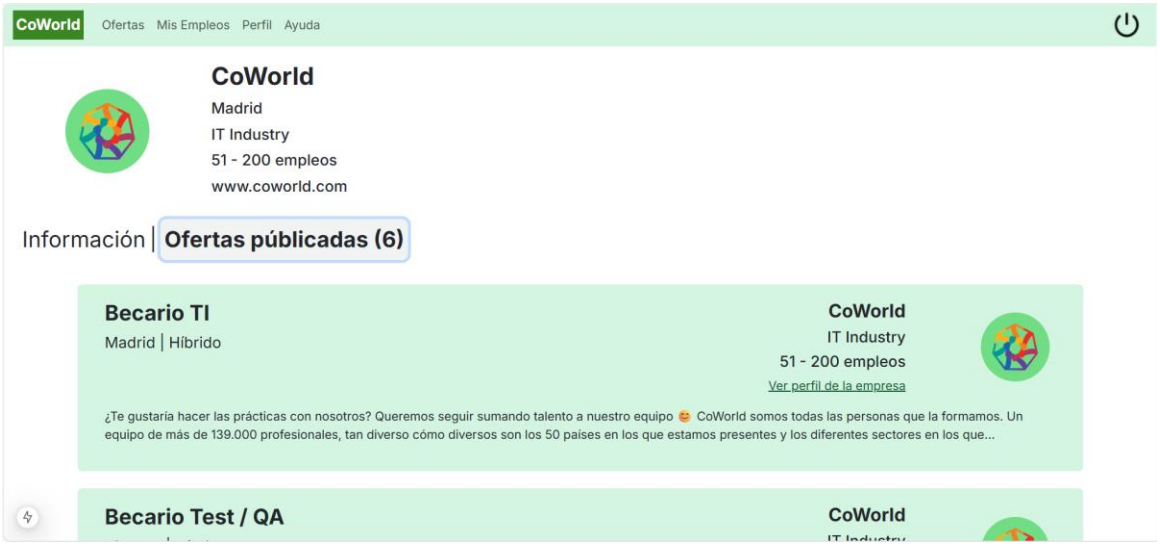
Portal de información de una empresa



Nota. La imagen muestra la vista que presenta la información de la empresa.

Figura 142

Portal de visualizar ofertas de una empresa

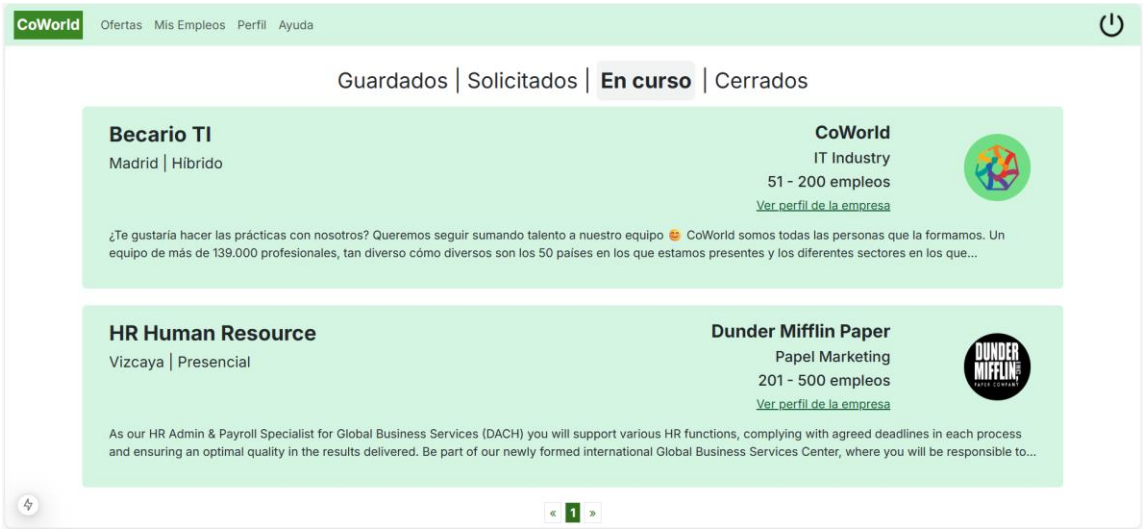


Nota. La imagen muestra la vista donde puede ver las ofertas disponibles de una empresa desde su página de información.

Al seleccionar la opción "*Mis Empleos*" en la barra de navegación, el usuario es redirigido a la página de gestión de sus ofertas de trabajos guardados y aplicados, organizadas en distintos estados. Las ofertas se muestran en función del estado elegido por el usuario.

Figura 143

Portal de gestión de ofertas guardadas y aplicadas de los candidatos

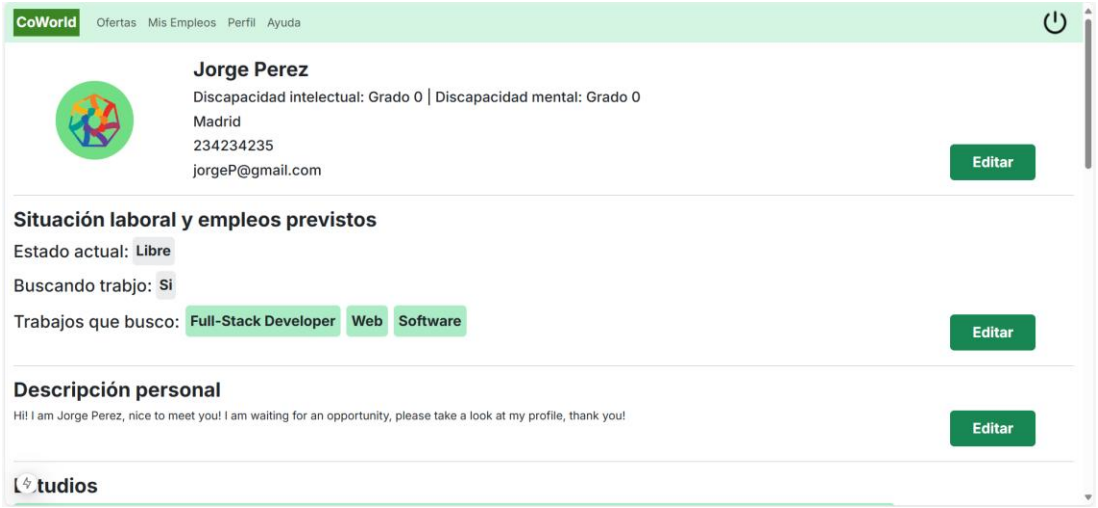


Nota. La imagen muestra la vista donde se gestiona las ofertas de trabajo guardadas o aplicadas por el candidato, clasificados por distintos estados.

Al seleccionar la opción "*Perfil*" en la barra de navegación, el usuario es redirigido a la página de gestión de su perfil. La información del perfil se organiza en varias secciones y cada una de ellas dispone un botón para editarla. El perfil es esencial para los candidatos, porque se utiliza como currículum digital y se envía a las empresas cuando se aplica a un puesto. Además, el algoritmo de recomendación del sistema toma información del perfil para priorizar las ofertas más ajustadas.

Figura 144

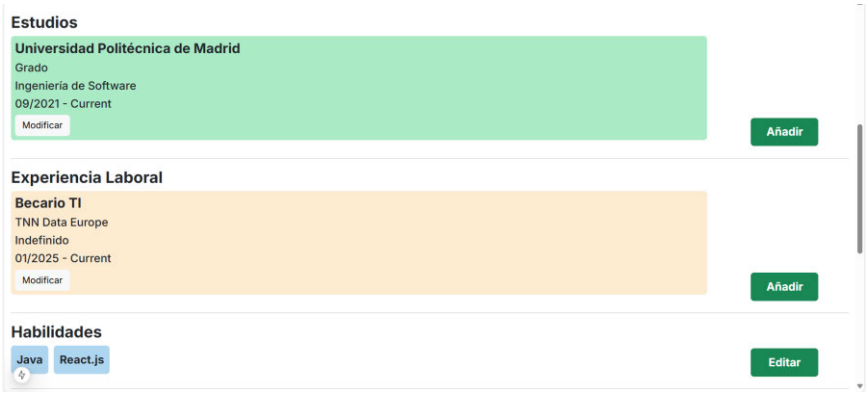
Portal de gestión de perfil para los candidatos (Parte 1)



Nota. La imagen muestra una parte de la vista de gestión de perfil de los candidatos.

Figura 145

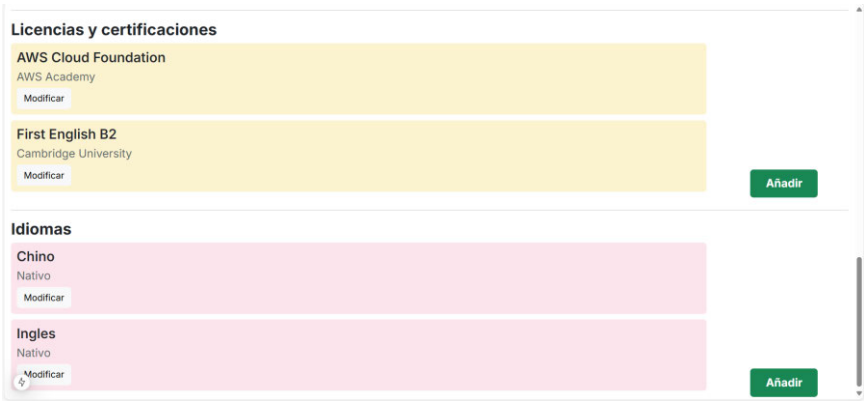
Portal de gestión de perfil para los candidatos (Parte 2)



Nota. La imagen muestra una parte de la vista de gestión de perfil de los candidatos.

Figura 146

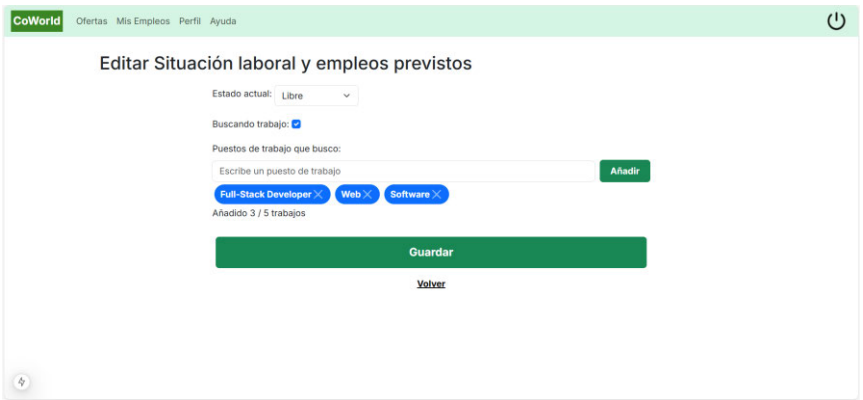
Portal de gestión de perfil para los candidatos (Parte 3)



Nota. La imagen muestra una parte de la vista de gestión de perfil de los candidatos.

Figura 147

Ejemplo de editar información de perfil del candidato

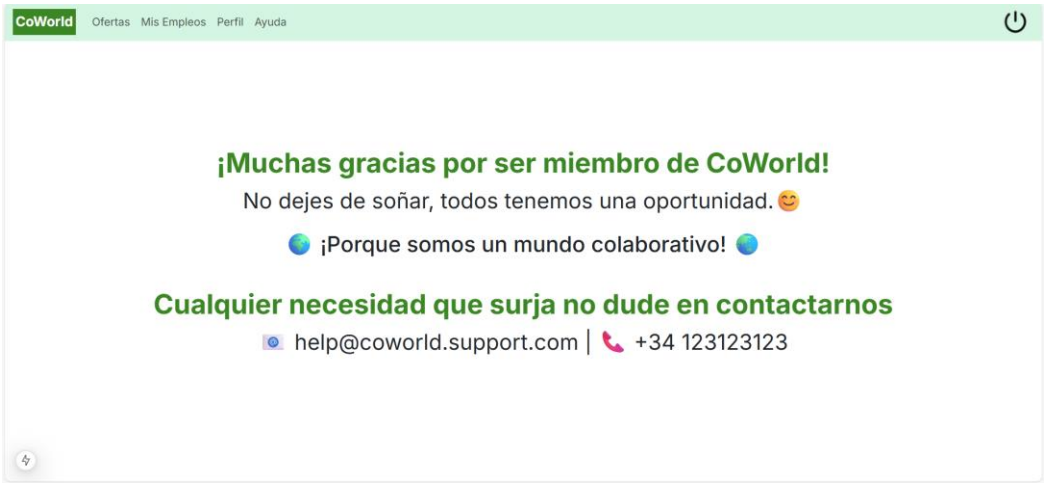


Nota. La imagen es como un ejemplo de la vista de edición de una sección del perfil del candidato.

Por el ultimo, al elegir la opción de ayuda en la barra de navegación, el usuario es dirigido a la página de ayuda de los candidatos. Donde se proporciona la dirección de correo electrónico y el teléfono de atención al cliente de la aplicación **CoWorld**.

Figura 148

Portal de ayuda de los candidatos



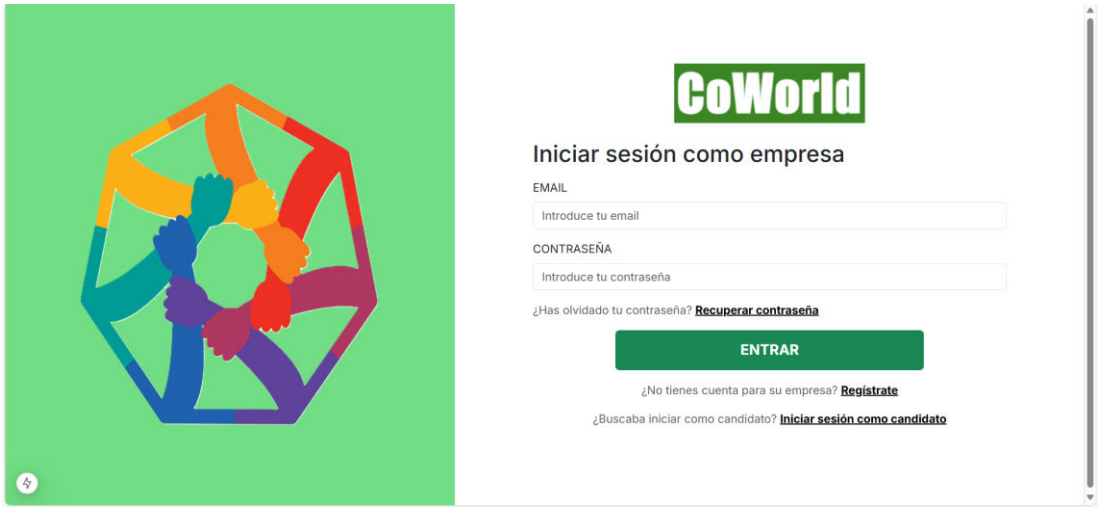
Nota. La imagen muestra el contenido de la página de ayuda para los candidatos.

5.1.2. Usuario con el rol de empresa/operador

Para las cuentas empresariales, el inicio de sesión se realiza en un portal diferente al de los candidatos con el propósito de facilitar la identificación de los roles de los usuarios.

Figura 149

Portal de inicio de sesión para las cuentas empresariales



Nota. La imagen muestra la vista de inicio de sesión para las cuentas empresariales.

En el caso si es un usuario nuevo, se puede crear una cuenta nueva con el enlace indicada con la palabra "*Regístrate*", es necesario completar todos los campos para registrar, y el formato de la contraseña restringida se puede consultar ubicando el ratón sobre el icono de interrogación .

Figura 150

Portal de registración de una cuenta empresarial

Nota. La imagen muestra la vista de registración de una nueva cuenta empresarial.

El procedimiento para la recuperación de la contraseña es igual para todos los tipos de usuarios. El detalle de este se puede consultar en la sección anterior.

Una vez inicio sesión con éxito, la página inicial difiere según el rol de usuario (empresa u operador), debido que los operadores no disponen todas las funcionalidades de las para las empresas, como gestión del perfil de empresa y gestión de operadores. Dicha restricción se implementa mediante diferentes barras de navegación adaptado a cada rol, cuales se diferencia en, la barra de navegación de operador no tiene las opciones de "*Perfil"* y "*Operadores"*.

En la parte superior de la página inicial se ubica la barra de navegación, y justo debajo se listan las ofertas de trabajo publicadas por la empresa, clasificadas según el estado que encuentra (En curso o Cerrado). Y cada oferta dispone de un botón para editar su información. Además, se puede publicar una nueva oferta de trabajo a través del botón de "*Publicar nueva oferta*", que redirige el usuario a la página correspondiente para realizar dicho procedimiento.

Figura 151

Página inicial de las empresas

Nota. La imagen muestra la página inicial para las cuentas con el rol de empresa.

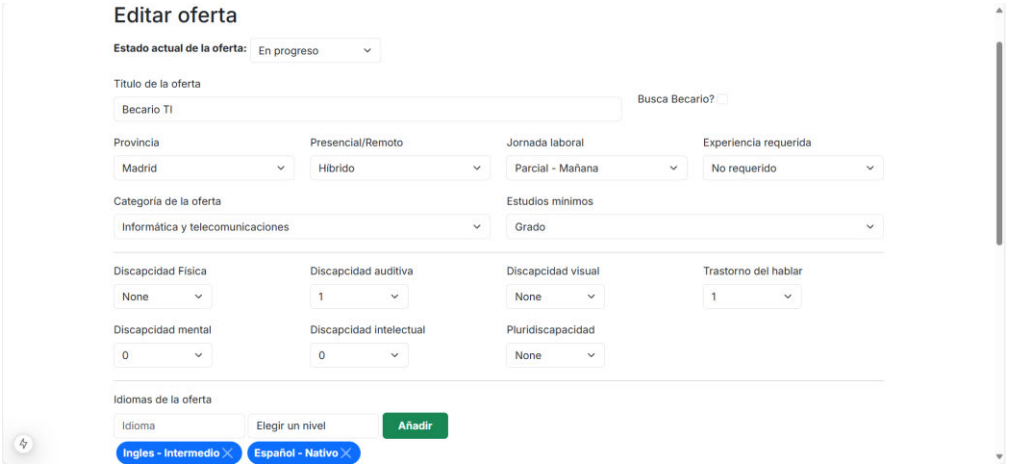
Figura 152

Página inicial para los operadores

Nota. La imagen muestra la página inicial para los usuarios con el rol de operador.

Figura 153

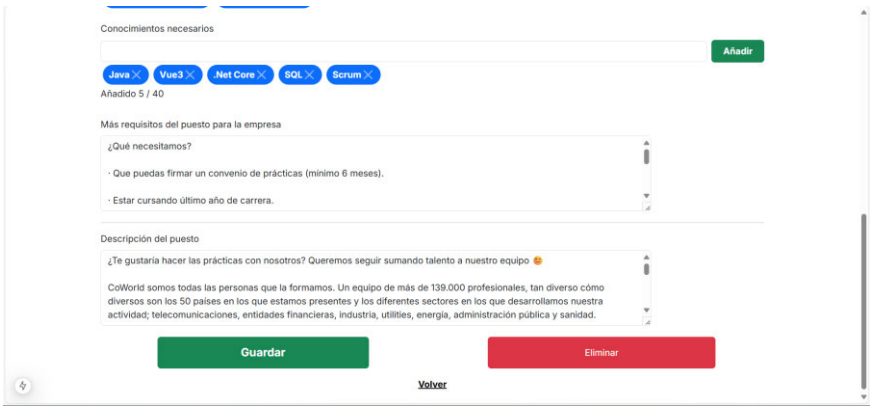
Ejemplo de editar una oferta de trabajo (Parte 1)



Nota. La imagen muestra una parte de la vista de modificación de información de una oferta de trabajo.

Figura 154

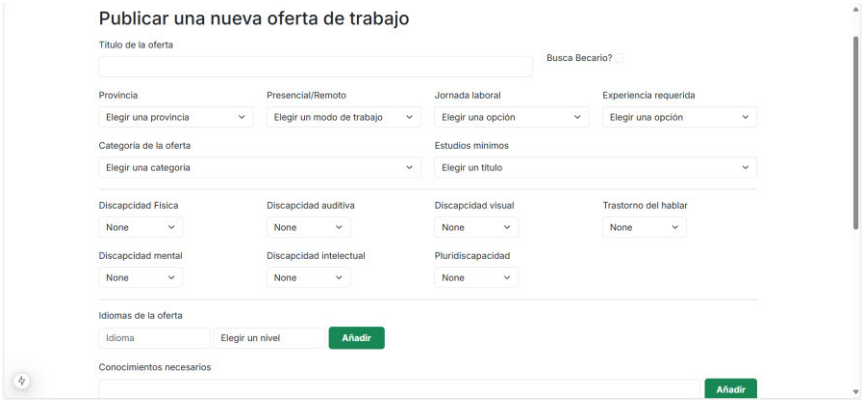
Ejemplo de editar una oferta de trabajo (Parte 2)



Nota. La imagen muestra una parte de la vista de modificación de información de una oferta de trabajo.

Figura 155

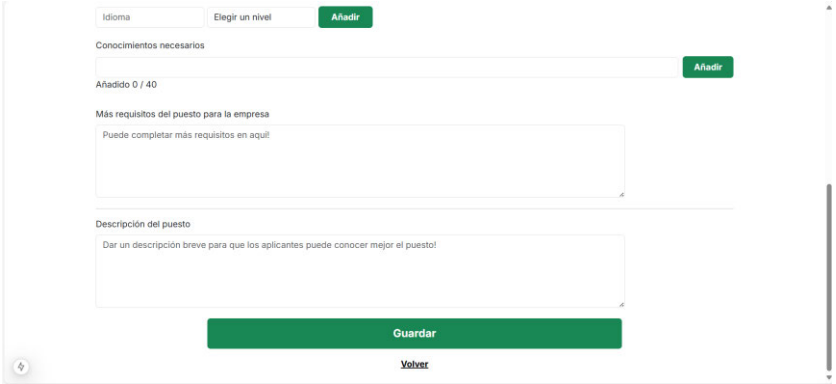
Portal de publicar una nueva oferta de trabajo (Parte 1)



Nota. La imagen muestra una parte de la vista de publicar una oferta de trabajo.

Figura 156

Portal de publicar una nueva oferta de trabajo (Parte 2)



Nota. La imagen muestra una parte de la vista de publicar una oferta de trabajo.

Al seleccionar la opción de candidatos de la barra de navegación, el usuario es redirigido a la página de gestión de los candidatos inscritos en las ofertas de trabajo de la empresa. En primer lugar, se muestran todas las ofertas de la empresa para que pueda seleccionar la que quiere gestionar.

Figura 157

Portal de elección de oferta de trabajo para gestionar los candidatos

Nota. La imagen muestra la vista donde se puede elegir una oferta de trabajo de la empresa para realizar la gestión de candidatos de esta.

Después de seleccionar una oferta de trabajo de la empresa, se listan todos los candidatos inscritos en la oferta, clasificados por distintos estados (Pendiente, A comunicar y Comunicado).

Figura 158

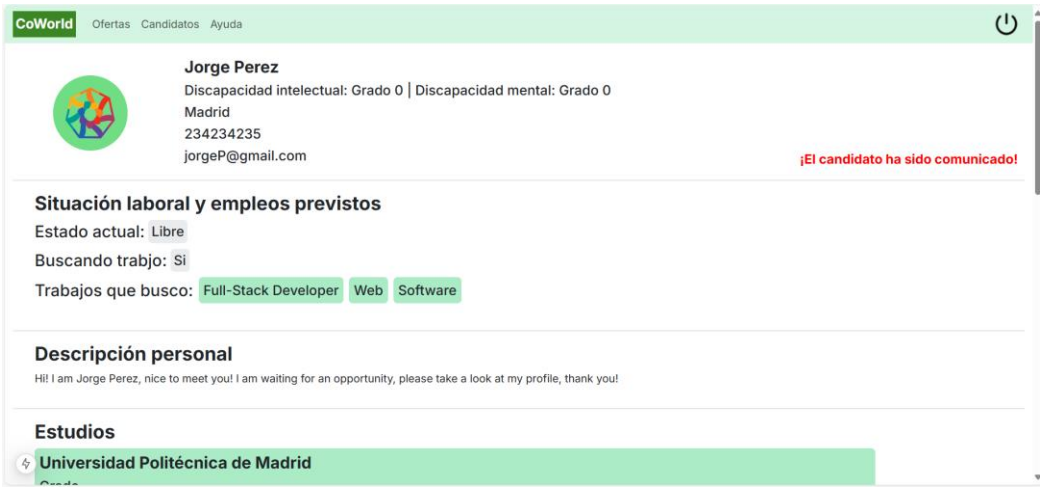
Portal de seleccionar candidatos de una oferta de trabajo

Nota. La imagen muestra la vista donde puede seleccionar un candidato a gestionar.

Al de seleccionar un candidato a gestionar, se muestra su perfil en modo solo lectura. Además, según el estado en el que se encuentre el candidato, se ofrecen distintas opciones para cambiar su estado. En el caso de que se trata de un candidato "Pendiente", se proporcionara las opciones de cambiar el estado a "*A comunicar*" y "*Comunicado*". Si el estado es "*A comunicar*", solo se puede cambiar el estado a "*Comunicado*". Y en el último caso de que se trata del estado "*Comunicado*", se muestra un aviso indicando que el usuario ya ha sido comunicado, y no es posible cambiar su estado.

Figura 159

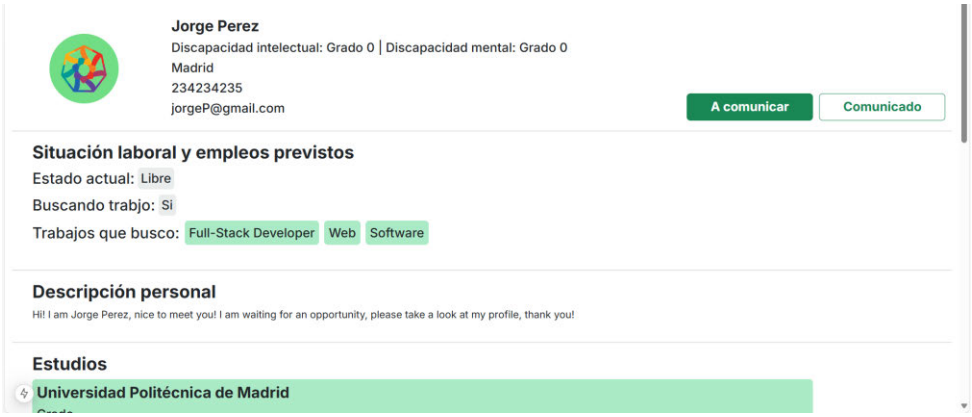
Ejemplo de gestión de candidato (Comunicado)



Nota. La imagen muestra la vista que se presenta cuando gestiona un candidato del estado "Comunicado".

Figura 160

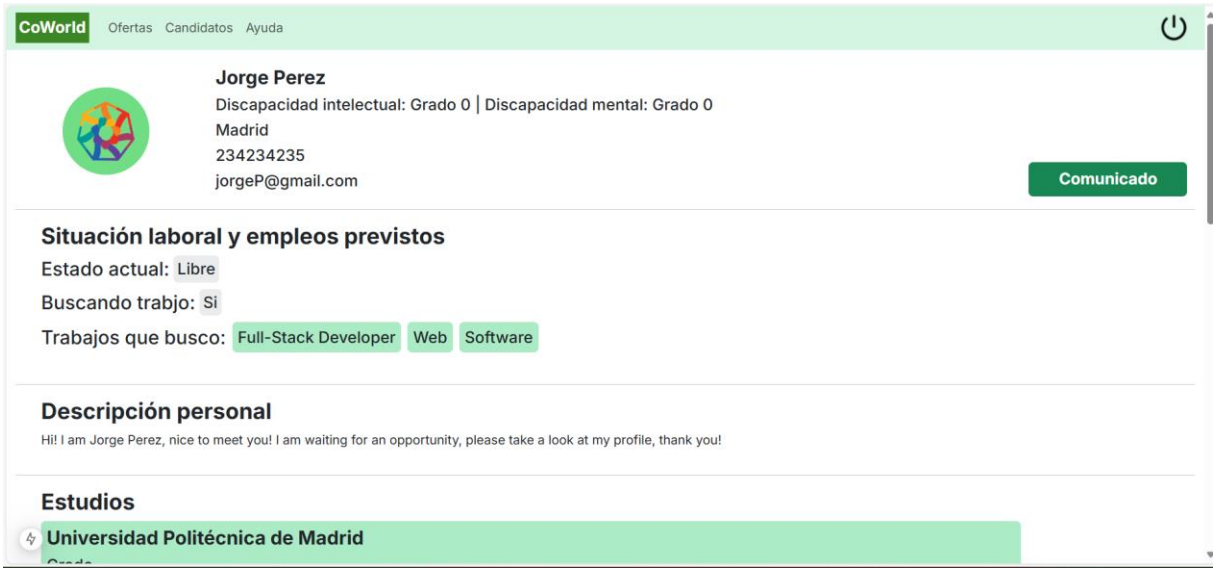
Ejemplo de gestión de candidato (Pendiente)



Nota. La imagen muestra la vista que se presenta cuando gestiona un candidato del estado "Pendiente".

Figura 161

Ejemplo de gestión de candidato (A Comunicado)

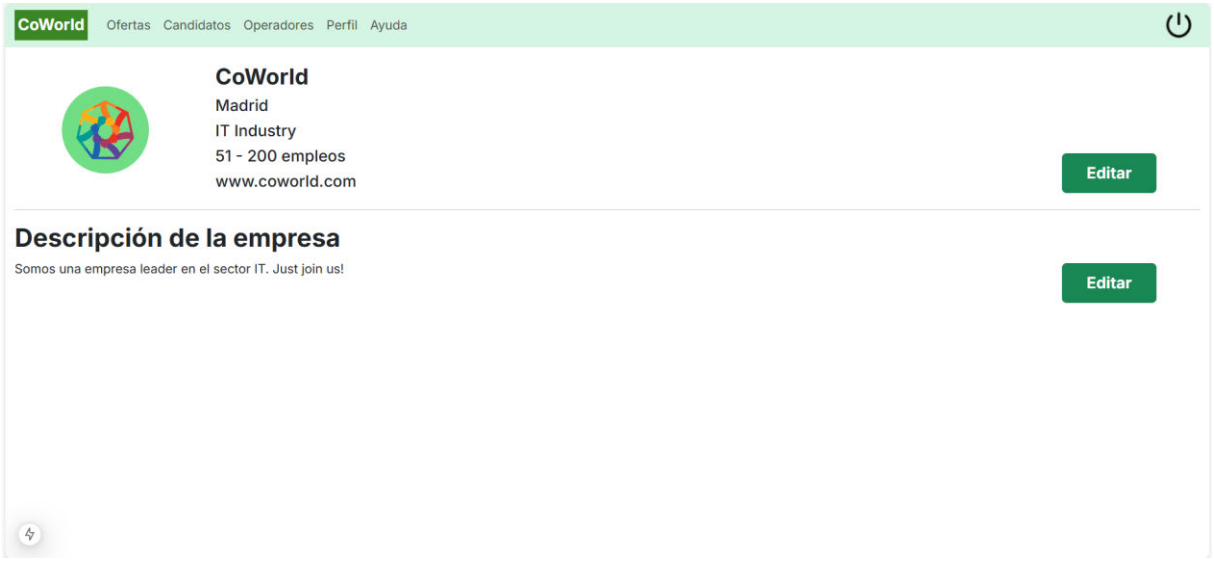


Nota. La imagen muestra la vista que se presenta cuando gestiona un candidato del estado "A comunicar".

En el caso de las empresas, al seleccionar la opción de *Perfil* en la barra de navegación, el usuario es redirigido a la página de gestión del perfil de la empresa. El perfil de las empresas está dividido en dos secciones, y para cada sección dispone de una opción para editarla.

Figura 162

Portal de gestión de perfil de la empresa



Nota. La imagen muestra la vista para gestionar el perfil de la empresa.

Figura 163

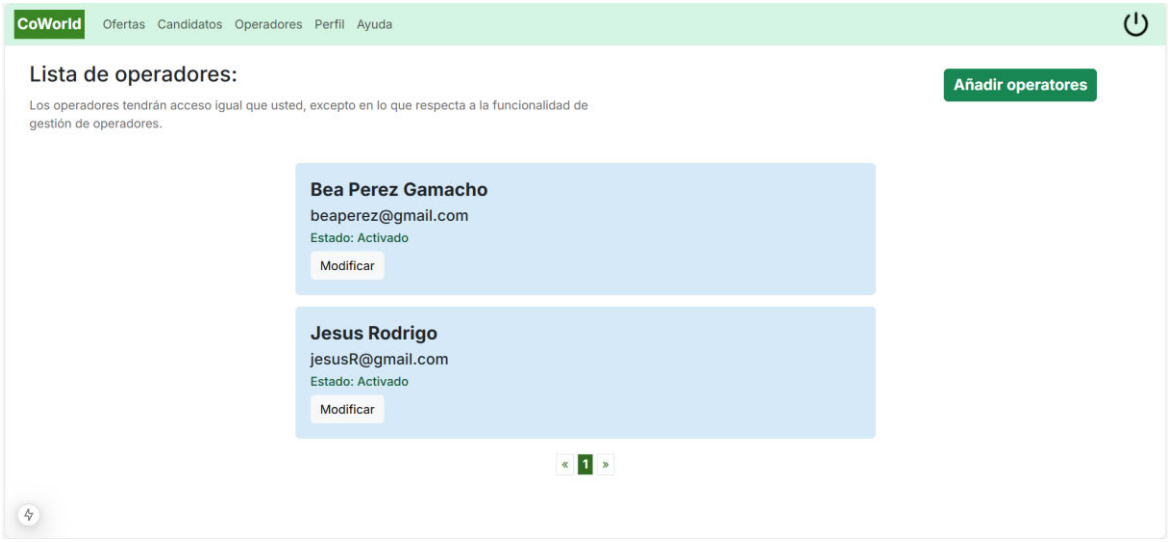
Ejemplo de edición de una sección del perfil de la empresa

Nota. La imagen muestra la vista de edición de una sección del perfil de la empresa.

Cuando las empresas seleccionan la opción de Operadores en la barra de navegación, el usuario es redirigido a la página de gestión del operador de la empresa. En dicha página se listan todos los operadores de la empresa, y para cada uno de ellos se dispone de la opción de modificar su información. Además, las empresas pueden añadir un nuevo operador a través del botón “Añadir operadores”, que redirige a la página correspondiente para realizar el procedimiento de agregar un nuevo operador.

Figura 164

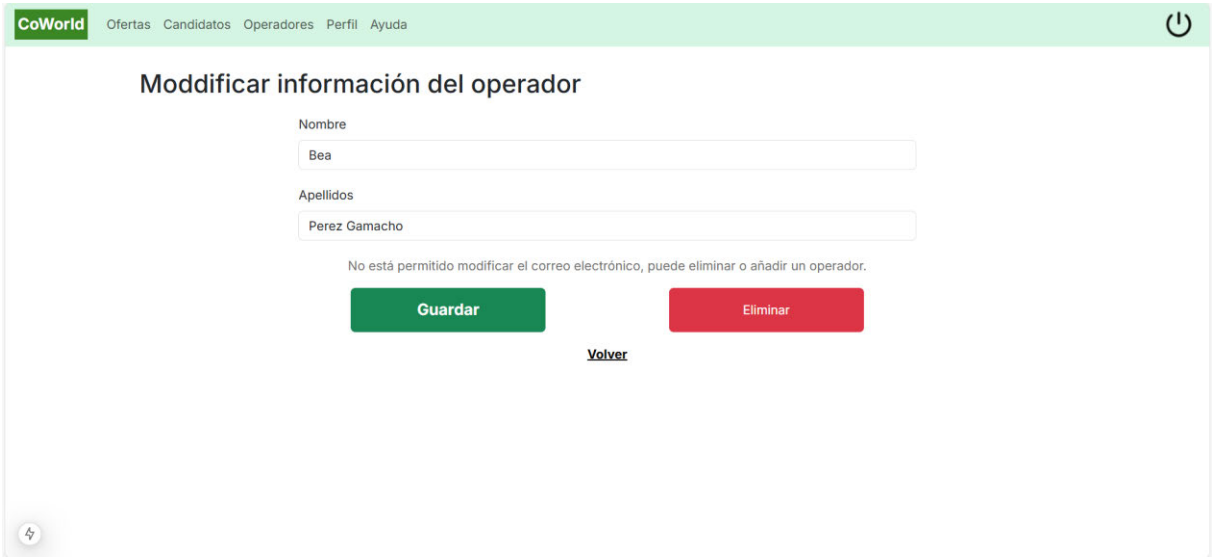
Portal de gestión de operadores



Nota. La imagen muestra la vista donde puede gestionar los operadores de la empresa.

Figura 165

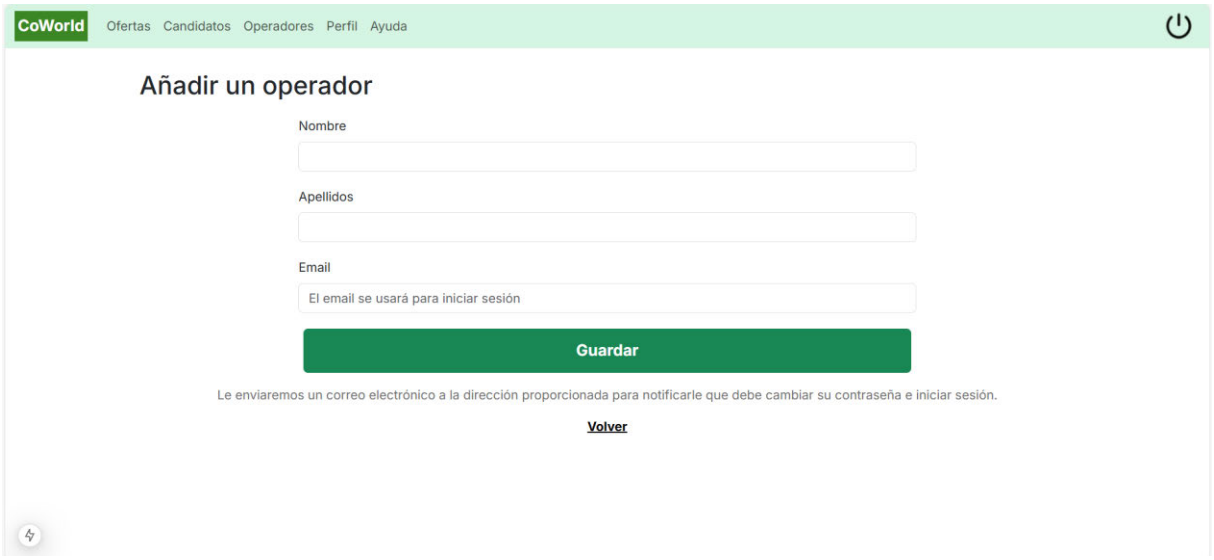
Portal de modificación de información de un operador



Nota. La imagen muestra la vista donde puede modificar la información de un operador.

Figura 166

Portal de añadir un nuevo operador



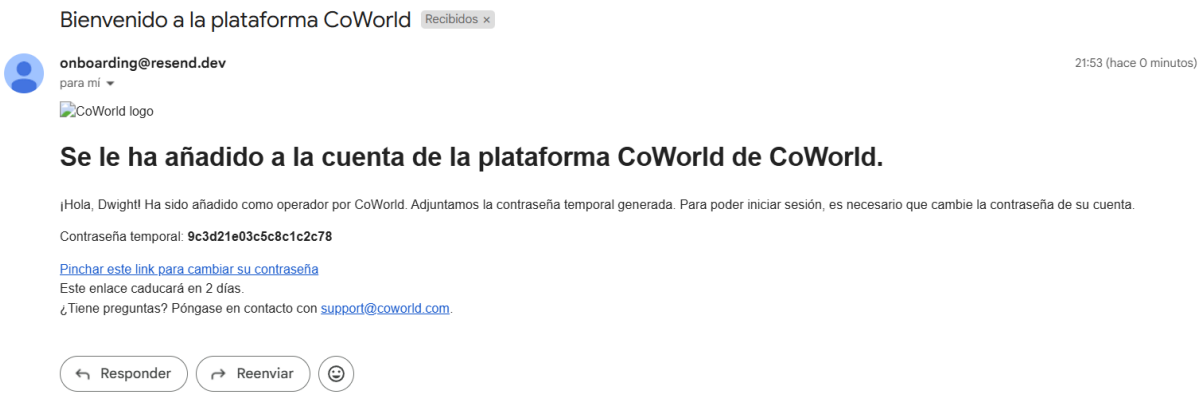
Nota. La imagen muestra la vista de añadir un nuevo operador para las empresas.

Cuando se completa el proceso, se envía un correo de aviso de activación de la cuenta a la dirección de correo del operador agregado. En el email se adjunta una contraseña temporal generada aleatoriamente por el sistema y un enlace para iniciar el proceso de activación. Con la contraseña temporal, el operador puede activar su cuenta estableciendo una nueva contraseña, y dicho enlace tiene una validez de tres días por

motivos de seguridad. Una vez que el operador haya activado su cuenta con éxito, podría iniciar sesión con la nueva contraseña establecida.

Figura 167

Correo de aviso de activación de cuenta de operador



Nota. La imagen muestra el contenido del correo de aviso de activación de cuenta de un operador.

Figura 168

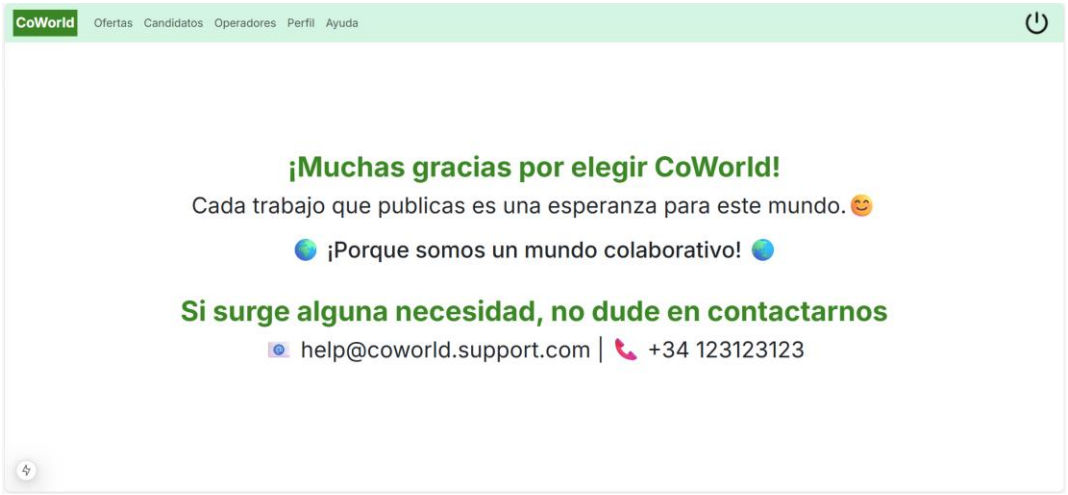
Portal de activación de cuenta de operador

Nota. La imagen muestra la vista para realizar activación de cuenta de operador.

Por el ultimo, al elegir la opción de ayuda en la barra de navegación, el usuario es dirigido a la página de ayuda de los candidatos. Donde se proporciona la dirección de correo electrónico y el teléfono de atención al cliente de la aplicación **CoWorld**.

Figura 169

Portal de ayuda para cuentas empresariales



Nota. La imagen muestra la vista de ayuda para las cuentas empresariales.

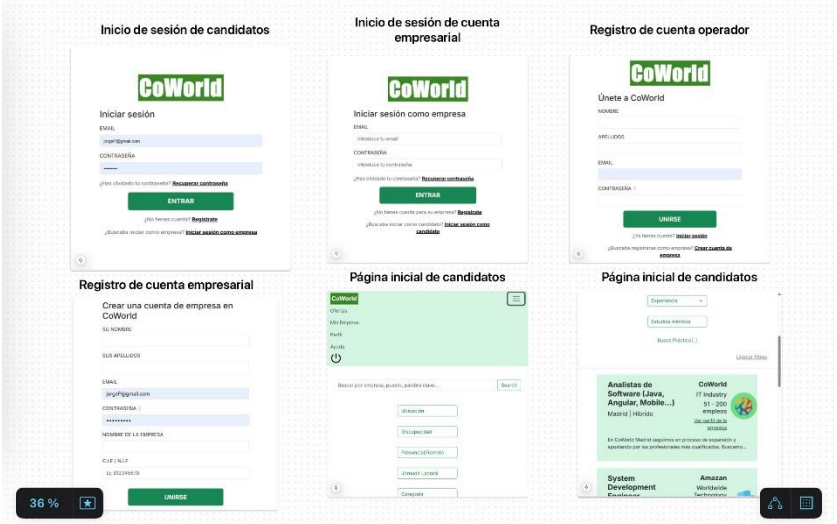
5.2. Responsive

La aplicación web CoWorld se ha desarrollado tiendo en cuenta el enfoque responsive. Gracias a Bootstrap, proporciona un sistema grid que facilita significativamente el diseño responsive. De este modo, la aplicación puede ofrecer una mejor visualización a los usuarios adaptándose distintos tamaños de pantalla, mejorando así la experiencia de usuario.

A continuación, se muestran las vistas del sistema en dispositivos con pantallas más pequeñas, como tabletas y móviles.

Figura 170

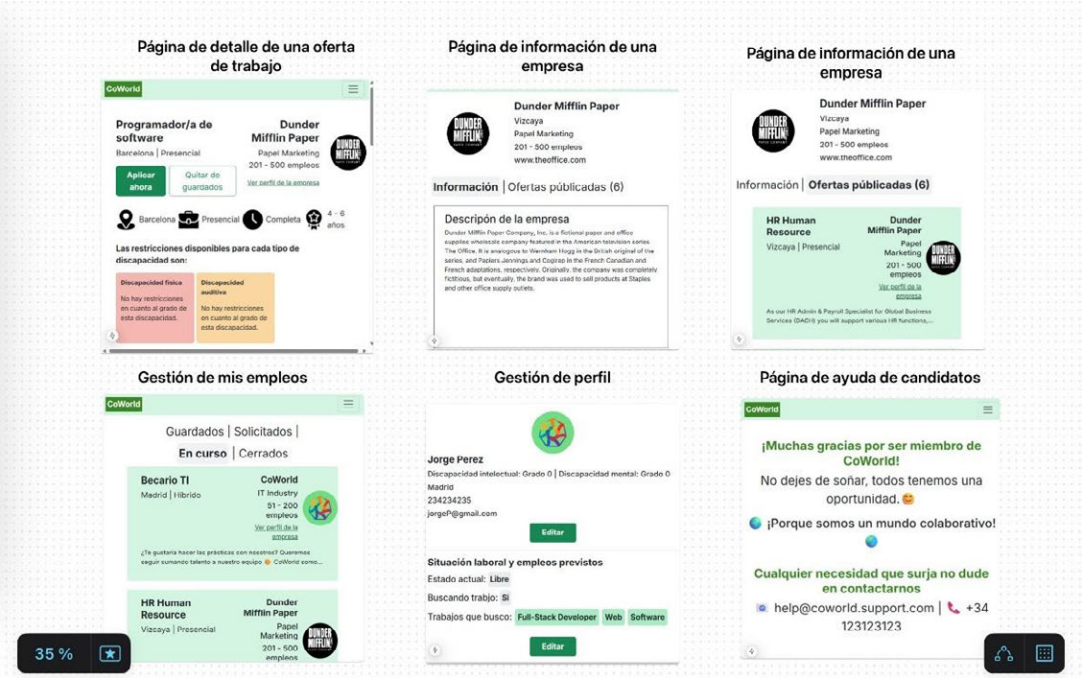
Vistas en modo responsive (Parte 1)



Nota. La imagen muestra unas vistas de la aplicación CoWorld en modo responsive.

Figura 171

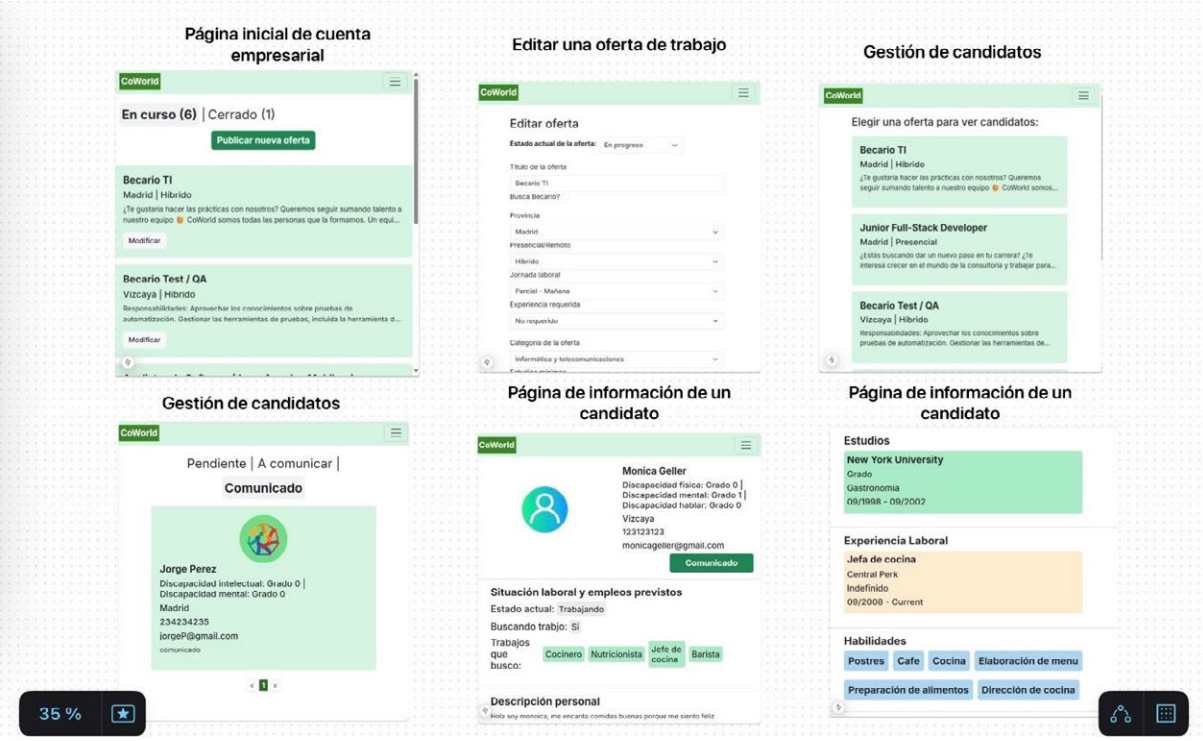
Vistas en modo responsive (Parte 2)



Nota. La imagen muestra unas vistas de la aplicación CoWorld en modo responsive.

Figura 172

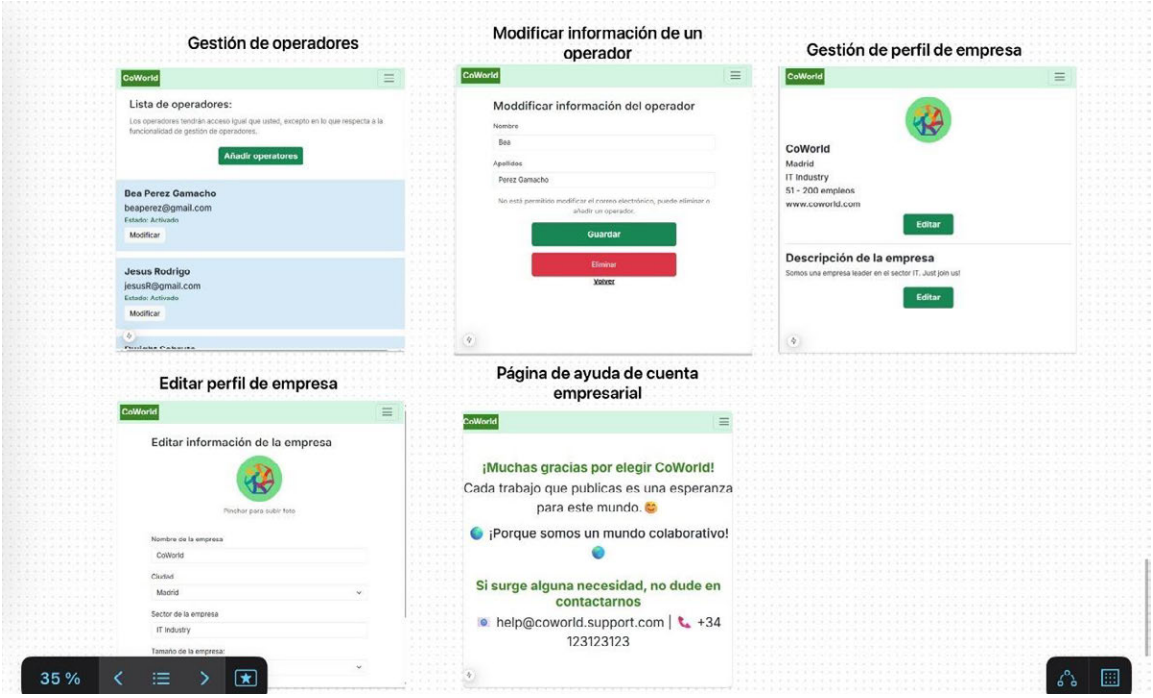
Vistas en modo responsive (Parte 3)



Nota. La imagen muestra unas vistas de la aplicación CoWorld en modo responsive.

Figura 173

Vistas en modo responsive (Parte 4)



Nota. La imagen muestra unas vistas de la aplicación CoWorld en modo responsive.

5.3. Accesibilidad

La accesibilidad es una práctica fundamental para las aplicaciones web sean utilizables para todas las personas. Especialmente en la aplicación **CoWorld**, cuyo su uso está orientada a personas con discapacidad, se ha desarrollado teniendo en cuenta de accesibilidad con el fin de proporcionar una mejor experiencia de usuario y garantizar la inclusión digital.

Para demostrar el compromiso con accesibilidad, se puede garantizar que todas las páginas de la aplicación han sido aprobadas con un porcentaje de 100% de accesibilidad mediante *Lighthouse*, que se basa en evaluaciones del impacto en los usuarios de *Axe*.

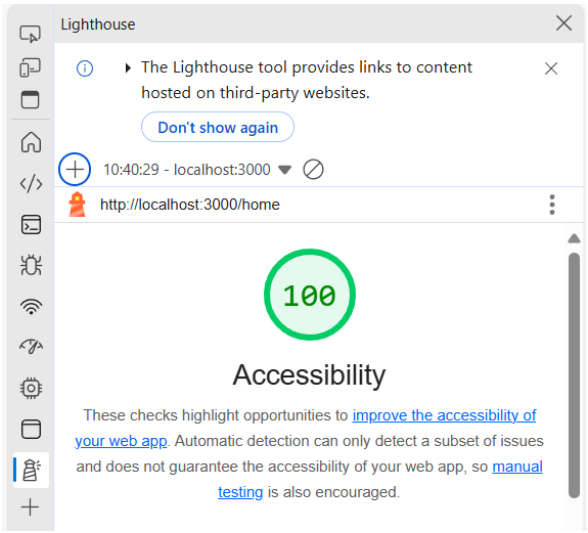
La accesibilidad de la aplicación se refleja en varios aspectos clave, entre los que destacan:

- **Contrate del color:** se asegura que el color del fondo de la pantalla con el color de los contenidos de la pantalla es contraste lo suficiente para garantizar la legibilidad. En la aplicación CoWorld, todas las páginas alcanzan el nivel de conformidad AAA de contraste del color.

- **Atributo alt en imágenes:** es un atributo fundamental para que los lectores de pantalla puedan leer el contenido de la página. En la aplicación CoWorld, todas las imágenes importantes tienen un atributo *alt* bien definido.
- **Enlaces accesibles:** todos los enlaces de la aplicación son fáciles de reconocer y están bien descritos, sin causar ambigüedad.
- **Campos de formulario accesibles:** todos los campos de entrada de un formulario de la aplicación están correctamente asociados con su etiqueta (*label*) y un placeholder descriptivo, para facilitar su comprensión por parte de los usuarios.
- **HTML semántico:** la aplicación se ha desarrollado haciendo uso de etiquetas de HTML semánticas, con el fin de facilitar la navegación y compresión a los lectores de pantalla.

Figura 174

Evaluación de accesibilidad de Lighthouse



Nota. La imagen muestra la puntación de accesibilidad obtenido por la aplicación en Lighthouse.

6. Conclusión

CoWorld presenta un pequeño gran hito en mi camino como ingeniera de software, ya que es la primera aplicación web completa que he desarrollado individualmente con mis capacidades de autoaprendizaje. Gracias a este proyecto de fin de grado me ha dado esta oportunidad de superar a mí mismo.

Durante el proceso de desarrollo, me encontré con muchos retos y dificultades, ya que todas las tecnologías, herramientas y lenguajes de programación utilizados eran completamente nuevos para mí. En particular, dado que Next.js es un framework de fullstack, supuso un desafío añadido debido a su curva de aprendizaje. Por ello, invertí mucho tiempo y esfuerzo a aprenderlos primero, asegúreme de tener una base sólida antes de comenzar el desarrollo.

Esta experiencia me ha permitido adquirir muchos conocimientos nuevos. Por un lado, las habilidades de desarrollo tanto en el Front-end como en el Back-end utilizando el framework Next.js, y he podido sentir la conveniencia y eficiencia de desarrollar una aplicación web utilizando un framework de fullstack. Por otro lado, he experimentado el flujo completo de desarrollo: desde una idea hasta una aplicación funcional. A lo largo del proceso, he tenido la oportunidad de revisar y aplicar con frecuencia los conocimientos adquiridos durante mi formación en el grado de Ingeniería de Software.

Respecto a la selección del tema, siempre he querido de hacer un proyecto que se puede contribuir a la sociedad. En mi último año de carrara, y al afrentarme las dificultades de la búsqueda de empleo en estos años, decidí investigar la situación de las personas con discapacidad con respecto a este tema. Fue entonces cuando surgió la idea de desarrollar **CoWorld**, una aplicación web destinada a las personas con discapacidad en búsqueda de empleo. Tengo la sincera esperanza de que **CoWorld** pueda aportar a mejorar a esta problemática y ayuda a este colectivo a sentirse la inclusión social. De ahí nace su nombre: **CoWorld**, un Mundo Colaborativo.

En definitiva, estoy satisfecha con los resultados obtenidos y confió en que todo lo aprendido en este proyecto personal me ayudará en mi futuro profesional. Sin duda, seguiré perfeccionándome, aprendiendo y mejorando para poder desarrollar proyectos cada vez mejor y significativo.

7. Aspectos Legales, Éticos y Sociales

7.1. Aspectos Legales

En la aplicación CoWorld, la privacidad de los datos de usuarios es lo más importante. La aplicación garantiza el cumplimiento del **Reglamento General de Protección de Datos** (RGPD). Los usuarios tienen derecho a conocer la información sobre los datos recopilados, su modo de uso y a quién se compartan. Los datos recopilados se almacenan de forma segura y conforme en la base de datos de la aplicación.

7.2. Aspectos Éticos

Toda la información publicada en la aplicación CoWorld debe ser respetuosa, honesta y transparente, evitando cualquier tipo de información engañosa ni errores.

7.3. Aspectos Sociales

La aplicación CoWorld se alinea con la Ley de Empleo para promover la inclusión social, según el artículo 54 **Personas con discapacidad demandantes de servicios de empleo:**

> Los servicios de empleo procurarán, prioritariamente, el acceso de dichas personas al empleo ordinario, el mantenimiento del empleo, la mejora de su empleabilidad a lo largo de su ciclo laboral y su desarrollo profesional, así como la sostenibilidad del empleo protegido (Boletín Oficial del Estado (BOE), 2023).

8. Futuras Ampliaciones

Aunque actualmente la aplicación web CoWorld cuenta con todas las funcionalidades establecidas en su diseño inicial, con el objetivo de mejorar la experiencia de usuario de usuario y asegurar una mayor calidad de la aplicación, se proponen las siguientes posibles futuras ampliaciones:

- **Desarrollo de un canal de comunicación entre candidatos y empresas:** lo que permitirá una interacción más directa entre ambas partes y un contacto más eficiente.
- **Integración de Inteligencia artificial (IA) en la búsqueda de ofertas por palabras claves:** actualmente, dicha funcionalidad solo esta implementado mediante coincidencias exactas utilizando expresiones regulares y el índice de texto proporcionado por MongoDB. Al integrar IA permitirá buscar por palabras borras y términos relacionados, lo que daría un resultado más precisa y relevante.
- **Desarrollo de un sistema de notificación en tiempo real**: para notificar a los candidatos cuando hay una actualización en el estado de las aplicaciones de ofertas de trabajo realizadas, mantenerlos informados sobre el progreso de sus procesos de selección.
- **Mejorar el diseño visual de la aplicación:** dado que actualmente la interfaz graficas de usuario solo está implementada con diseños básicos, se puede mejorar para ofrecer una mejor apariencia y añadir más dinámicas.

9. Bibliografía

AUTH0. (s.f. de s.f. de s.f.). *Introduction to JSON Web Tokens*. Obtenido de JWT Debugger: https://jwt.io/introduction

Axios. (s.f. de s.f de s.f.). *Empezando | Axios Docs*. Obtenido de Axios HTTP: https://axios-http.com/es/docs/intro

Boletín Oficial del Estado (BOE). (1 de Marzo de 2023). *Agencia Estatal Boletín Oficial del Estado.* Obtenido de Ley 3/2023, de 28 de febrero, de Empleo.: https://www.boe.es/buscar/doc.php?id=BOE-A-2023-5365

Comisión Económica para América Latina y el Caribe [CEPAL]. (12 de Diciembre de 2018). *Inclusión social, económica y política de las personas mayores*. Obtenido de Cepal: https://www.cepal.org/es/enfoques/inclusion-social-economica-politica-personas-mayores

Naciones Unidas. (s.f. de s.f. de s.f.). *Objetivo 10: Reducir la desigualdad en y entre los países*. Obtenido de Páginas de las Naciones Unidas: https://www.un.org/sustainabledevelopment/es/inequality/

Naciones Unidas. (s.f. de s.f. de s.f.). *Objetivo 8: Promover el crecimiento económico inclusivo y sostenible, el empleo y el trabajo decente para todos*. Obtenido de Páginas de las Naciones Unidas : https://www.un.org/sustainabledevelopment/es/economic-growth/

Next.js contributors. (s.f. de s.f. de s.f.). *Next.js.* Obtenido de Project structure and organization: https://nextjs.org/docs/app/getting-started/project-structure

Node.js contributors. (s.f. de s.f. de s.f.). *Sobre Node.js®*. Obtenido de nodejs: https://nodejs.org/es/about

Servicio Público de Empleo Estatal. (2025). *Informe del Mercado de Trabajo de las Personas con Discapacidad 2025 (Datos 2024).* Madrid: Servicio Público de Empleo Estatal .

Universidad de Sevilla. (27 de Octubre de 2023). El 60% de los estudiantes con discapacidad cree que tendrá más dificultades que sus compañeros para encontrar trabajo. *UNIVERSIDAD*, págs. 1-2.

Wikipedia contributor. (19 de Noviembre de 2024). *React*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/React

Wikipedia contributors. (27 de Noviembre de 2024). *Jira*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Jira

Wikipedia contributors. (12 de April de 2025). *Cliente-servidor*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Cliente-servidor

Wikipedia contributors. (15 de April de 2025). *Postman (software)*. Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Postman_(software)#

Wikipedia contributors. (13 de Abril de 2025). *Visual Studio Code*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Visual_Studio_Code#

10. Anexo

En este apartado se adjunta los pasos a seguir para poder arrancar la aplicación CoWorld en local. Se abrirá en el puerto 3000. (http://localhost:3000)

1. Clonar el repositorio https://github.com/JialeWangXu/CoWorld.git
2. Ubicar en la carpeta *CoWorld* (cd CoWorld)
3. Instalar las dependencias con el comando: *npm install*
4. Configurar las variables de entorno creando un archivo *.env*:
    - MONGODB_URL
    - RESEND_KEY
    - ACCESS_TOKEN_SECRET
    - REFRESH_TOKEN_SECRET
    - RESET_PWD_TOKEN_SECRETE
    - REACT_EDITOR=atom
5. Tras de conectar la base de datos, en la carpeta *collections* puede encontrar las colecciones de ejemplo para observar el funcionamiento de la aplicación.
6. Ejecutar la aplicación con el comando: *npm run dev*
7. Acceder con un navegador: http://localhost:3000

Consejo: es probable que al abrir la aplicación por primera vez, en algunos archivos pages de la parte Front-end se muestre un aviso de que no se encuentra la ubicación del archivo style.module.scss. Este problema es común en SCSS y se debe ejecutar la aplicación para que reconozca las rutas de los archivos SCSS. Una vez levantada la aplicación por primera vez, los avisos desaparecen.

11. Agradecimiento

Quiero transmitir mi sincero agradecimiento a todas las personas que me han apoyada constantemente en la realización de este proyecto.

A mi tutor, por aceptar mi propuesta, brindarme esta oportunidad de desarrollar la aplicación que quiero hacer, y por animarme siempre a tener la confianza para completar este proyecto. Muchas gracias.

A mis amigos, que me han dado apoyo emocional y me han acompañada a lo largo del desarrollo.

A ETSISI, es donde pasé mis mejores cuatros años de estudiante y donde realmente sentí la felicidad de aprender. Fue allí donde descubrí mi propio potencial.

Agradezco por cada encuentro en el camino. Que sigamos creciendo y estando cada vez mejor.